BỘ GIÁO DỤC VÀ ĐÀO TẠO

TRƯỜNG ĐẠI HỌC DÂN LẬP HẢI PHÒNG

-------o0o-------

ISO 9001:2000

ĐỒ ÁN TỐT NGHIỆP

NGÀNH CÔNG NGHỆ THÔNG TIN

HẢI PHÒNG 2013

BỘ GIÁO DỤC VÀ ĐÀO TẠO
TRƯỜNG ĐẠI HỌC DÂN LẬP HẢI PHÒNG
-------o0o-------

TÌM HIỂU PHƯƠNG PHÁP QUY HOẠCH ĐỘNG CHO TÍNH KHOẢNG CÁCH

ĐỒ ÁN TỐT NGHIỆP ĐẠI HỌC HỆ CHÍNH QUY

Ngành: Công nghệ Thông tin

HẢI PHÒNG - 2013

BỘ GIÁO DỤC VÀ ĐÀO TẠO
TRƯỜNG ĐẠI HỌC DÂN LẬP HẢI PHÒNG
-------o0o-------

TÌM HIỂU PHƯƠNG PHÁP QUY HOẠCH ĐỘNG CHO TÍNH KHOẢNG CÁCH

ĐỒ ÁN TỐT NGHIỆP ĐẠI HỌC HỆ CHÍNH QUY

Ngành: Công nghệ Thông tin

Sinh viên thực hiện: Vũ Hữu Trường
Giáo viên hướng dẫn: PGS.TS Ngô Quốc Tạo
Mã số sinh viên: 1351010055

HẢI PHÒNG - 2013

BỘ GIÁO DỤC VÀ ĐÀO TẠO
TRƯỜNG ĐẠI HỌC DÂN LẬP HẢI PHÒNG

CỘNG HÒA XA HỘI CHỦ NGHĨA VIỆT NAM
Độc lập - Tự do - Hạnh phúc
-------o0o-------

NHIỆM VỤ THIẾT KẾ TỐT NGHIỆP

Sinh viên: Vũ Hữu Trường Mã SV: 1351010055

Lớp: CT1301 Ngành: Công nghệ Thông tin

Tên đề tài: **Tìm hiểu thuật toán quy hoạch động cho tính khoảng cách**

NHIỆM VỤ ĐỀ TÀI

1. Nội dung và các yêu cầu cần giải quyết trong nhiệm vụ đề tài tốt nghiệp

a. Nội dung

- Tổng quan về thuật toán quy hoạch động
- Một số kinh nghiệm xây dựng thuật toán quy hoạch động
- Thử nghiệm trên ngôn ngữ

b. Các yêu cầu cần giải quyết

- Hiểu nội dung của quy hoạch động
- Viết xong đồ án
- Cài đặt thử nghiệm chương trình đặc trưng

CÁN BỘ HƯỚNG DẪN ĐỀ TÀI TỐT NGHIỆP

Người hướng dẫn thứ nhất:

Họ và tên: Ngô Quốc Tạo

Học hàm, học vị: Phó Giáo Sư - Tiến Sĩ

Cơ quan công tác: Trưởng phòng Nhận dạng và Công nghệ tri thức , Viện Công nghệ thong tin , Viện Hàn Lâm Khoa học và Công nghệ Việt Nam

Nội dung hướng dẫn: .....

……

……

Người hướng dẫn thứ hai:

Họ và tên: ……

Học hàm, học vị: ……

Cơ quan công tác: ……

Nội dung hướng dẫn: ……..

……

……

Đề tài tốt nghiệp được giao ngày tháng năm 2013

Yêu cầu phải hoàn thành trước ngày tháng năm 2013

Đã nhận nhiệm vụ: Đ.T.T.N	Đã nhận nhiệm vụ: Đ.T.T.N
Sinh viên	Cán bộ hướng dẫn Đ.T.T.N

PGS.TS. Ngô Quốc Tạo

Hải Phòng, ngày .....tháng.....năm 2013

HIỆU TRƯỞNG

GS.TS.NGUT Trần Hữu Nghị

PHẦN NHẬN XÉT TÓM TẮT CỦA CÁN BỘ HƯỚNG DẪN

1. **Tinh thần thái độ của sinh viên trong quá trình làm đề tài tốt nghiệp:**

.....
.....
.....
.....
.....
.....

2. **Đánh giá chất lượng của đề tài tốt nghiệp (so với nội dung yêu cầu đã đề ra trong nhiệm vụ đề tài tốt nghiệp)**

.....
.....
.....
.....
.....
.....
.....
.....

3. **Cho điểm của cán bộ hướng dẫn:**

(Điểm ghi bằng số và chữ)

.....
.....

Ngày.....tháng.....năm 2013

Cán bộ hướng dẫn chính

(Ký, ghi rõ họ tên)

PHẦN NHẬN XÉT ĐÁNH GIÁ CỦA CÁN BỘ CHẤM PHẢN BIỆN ĐỀ TÀI TỐT NGHIỆP

1. Đánh giá chất lượng đề tài tốt nghiệp (về các mặt như cơ sở lý luận, thuyết minh chương trình, giá trị thực tế, ...)

.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....

2. Cho điểm của cán bộ phản biện

(*Điểm ghi bằng số và chữ*)

.....

.....

.

Ngày.....tháng.....năm 2013

Cán bộ chấm phản biện

(Ký, ghi rõ họ tên)

LỜI CẢM ƠN

Tôi xin được bày tỏ lòng biết ơn chân thành đến Ban Giám Hiệu, các thầy giáo, cô giáo trường đại học Dân Lập Hải Phòng, đã giảng dạy và tạo mọi điều kiện cho tôi học tập, nghiên cứu và hoàn thành Đồ án này.

Đặc biệt, tôi xin bày tỏ sự kính trọng và lòng biết ơn sâu sắc đến **PGS.TS. Ngô Quốc Tạo** - người đã tận tình hướng dẫn và giúp đỡ tôi trong suốt quá trình học tập, nghiên cứu và hoàn thành Đồ án.

Cảm ơn gia đình, bạn bè đã hết lòng giúp đỡ, khích lệ, động viên tôi để tôi hoàn thành Đồ án. Xin chia sẻ niềm vui này với bạn bè và những người thân yêu.

MỤC LỤC

DANH MỤC CÁC BẢNG

DANH MỤC CÁC HÌNH

Chương 1: TỔNG QUAN VỀ PHƯƠNG PHÁP QUY HOẠCH ĐỘNG

1.1. Giới thiệu chung

Quy hoạch động (Dynamic Programming) là một phương pháp rất hiệu quả giải nhiều bài toán tin học, đặc biệt là những bài toán tối ưu. Những bài toán này thường có nhiều nghiệm chấp nhận được và mỗi nghiệm có một giá trị đánh giá. Mục tiêu đặt ra là tìm nghiệm tối ưu, đó là nghiệm có giá trị đánh giá lớn nhất hoặc nhỏ nhất (tối ưu). Ví dụ tìm đường đi ngắn nhất giữa hai đỉnh của đồ thị, tìm chuỗi con chung dài nhất của hai chuỗi, tìm chuỗi con tăng dài nhất,… Số lượng bài toán được giải bằng lập trình động cũng rất lớn. Ví dụ riêng kì thi Olympic quốc tế về Tin học 2004 có tới ba bài trong sáu bài có thể giải bằng quy hoạch động.

Quy hoạch động giải các bài toán bằng cách kết hợp các lời giải của các bài toán con của bài toán đang xét. Phương pháp này khả dụng khi các bài toán con không độc lập đối với nhau, tức là khi các bài toán con có dùng chung những bài toán "cháu". Quy hoạch động giải các bài toán "cháu" dùng chung này một lần và lưu lời giải của chúng trong một bảng và sau đó khỏi phải tính lại khi gặp lại bài toán cháu đó. Quy hoạch động được áp dụng cho những bài toán tối ưu hóa (optimization problem). Bốn bước của qui hoạch động : Đặc trưng hóa cấu trúc của lời giải tối ưu. + Định nghĩa giá trị của lời giải tối ưu một cách đệ quy. + Tính trị của lời giải tối ưu theo kiểu từ dưới lên. + Cấu tạo lời giải tối ưu từ những thông tin đã được tính toán. Các thành phần của quy hoạch động : + Tiểu cấu trúc tối ưu - Một bài toán có tính chất tiểu cấu trúc tối ưu nếu lời giải tối ưu chứa trong nó những lời giải tối ưu của những bài toán con. + Những bài toán con trùng lắp - Khi một giải thuật đệ quy gặp lại cùng một bài toán con nhiều lần, ta bảo rằng bài toán tối ưu hóa có những bài toán con trùng lặp.

Chuỗi con chung dài nhất LCS : O(m+n)

Bài toán cái túi (Knapsack) : Bài toán cái túi có thể dễ dàng giải được nếu M không lớn, nhưng khi M lớn thì thời gian chạy trở nên không thể chấp nhận được. Phương pháp này không thể làm việc được khi M và trọng lượng/kích thước là những số thực thay vì số nguyên. Giải thuật qui hoạch động để giải bài toán cái túi có thời gian chạy tỉ lệ với NM.

Giải thuật Warshall [$O(V^3)$- Giải thuật Floyd [$O(V^3)$] : thể hiện sự áp dụng chiến lược quy hoạch động vì sự tính toán căn cứ vào một hệ thức truy hồi nhưng lại không xây dựng thành giải thuật đệ quy. Thay vào đó là một giải thuật lặp với sự hỗ trợ của một ma trận để lưu trữ các kết quả trung gian.

Giải thuật tham lam

Các giải thuật tối ưu hóa thường đi qua một số bước với một tập các khả năng lựa chọn tại mỗi bước. Một giải thuật tham lam thường chọn một khả năng mà xem như tốt nhất tại lúc đó. Tức là, giải thuật chọn một khả năng tối ưu cục bộ với hy vọng sẽ dẫn đến một lời giải tối ưu toàn cục. VD : +Bài toán xếp lịch cho các hoạt động + Bài toán cái túi dạng phân số + Bài toán mã Huffman+ Giải thuật Prim để tính cây bao trùm tối thiểu.

Hai thành phần chính của giải thuật tham lam :

+ Tính chất lựa chọn tham lam : Lựa chọn được thực hiện bởi giải thuật tham lam tùy thuộc vào những lựa chọn đã làm cho đến bây giờ, nhưng nó không tùy thuộc vào bất kỳ lựa chọn trong tương lai hay những lời giải của những bài toán con.

Như vậy, một giải thuật tham lam tiến hành theo kiểu từ trên xuống, thực hiện mỗi lúc một lựa chọn tham lam.

+ Tiểu cấu trúc tối ưu: Một bài tóan có tính chất tiểu cấu trúc tối ưu nếu một lời giải tối ưu chứa trong nó những lời giải tối ưu cho những bài toán con.

Dùng giải thuật tham lam cho bài toán cái túi dạng phân số và qui hoạch động cho bài toán cái túi dạng 0-1.

Giải thuật tham lam cho bài toán xếp lịch các hoạt động:

Hoạt động được chọn bởi thủ tục GREEDY-ACTIVITY-SELECTER thường là hoạt động với thời điểm kết thúc sớm nhất mà có thể được xếp lịch một cách hợp lệ. Hoạt động được chọn theo cách "tham lam" theo nghĩa nó sẽ để lại cơ hội để xếp lịch cho được nhiều hoạt độngkhác. Giải thuật tham lam không nhất thiết đem lại lời giải tối ưu. Tuy nhiên thủ tục GREEDY-ACTIVITY-SELECTOR thường tìm được một lời giải tối ưu cho một thể hiện của bài toán xếp lịch các hoạt động.

Bài toán cái túi dạng phân số (knapsack) : O(n) + Giải thuật HUFFMAN (dùng phổ biến và rất hữu hiệu cho việc nén dữ liệu) trên tập n ký tự sẽ là : O(nlgn) + Bài toán tô màu đồ thị : Đầu tiên ta cố tô cho được nhiều đỉnh với màu đầu tiên, và rồi dùng một màu mới tô các đỉnh chưa tô sao cho tô được càng nhiều đỉnh càng tốt. Và quá trình này được lặp lại với những màu khác cho đến khi mọi đỉnh đều được tô màu. Độ phức tạp của thủ tục SAME_COLOR: O(n). Nếu m là số màu được dùng để tô đồ thị thì thủ tục SAME_COLOR được gọi tất cả m lần.

Do đó, độ phức tạp của toàn giải thuật: m* O(n). Vì m thường là một số nhỏ=>độ phức tạp tuyến tính . Ứng dụng : xếp lịch thi học kỳ , gán tần số trong lĩnh vực vô tuyến ,điện thoại di động.

Giải thuật quay lui : "bước hướng về lời giải đầy đủ và ghi lại thông tin về bước này mà sau đó nó có thể bị tháo gỡ và xóa đi khi phát hiện rằng bước này đã không dẫn đến lời giải đầy đủ, tức là một bước đi dẫn đến "tình thế bế tắc"(dead-end). (Hành vi này được gọi là quay lui - backtracking.) VD : bài toán tám con hậu ,bài toán con mã đi tuần

Một phương pháp tổng quát để giải quyết vấn đề: thiết kế giải thuật tìm lời giải cho bài tóan không phải là bám theo một tập qui luật tính toán được xác định mà là bằng cách thử và sửa sai .Khuôn mẫu thông thường là phân rã quá trình thử và sửa sai thành những công tác bộ phận. Thường thì những công tác bộ phận này được diễn tả theo lối đệ quy một cách thuận tiện và bao gồm việc thăm dò một số hữu hạn những công tác con.Ta có thể coi toàn bộ quá trình này như là một quá trình tìm kiếm mà dần dần cấu tạo và duyệt qua một cây các công tác con.

Tìm tất cả các lời giải : Một khi một lời giải được tìm thấy và ghi lại, ta tiếp

tục xét ứng viên kế trong quá trình chọn ứng viên một cách có hệ thống .

Thời gian tính toán của các giải thuật quay lui thường là hàm mũ (exponential). Nếu mỗi nút trên cây không gian trạng thái có trung bình α nút con, và chiều dài của lối đi lời giải là N, thì số nút trên cây sẽ tỉ lệ với α^N .

Giải thuật nhánh và cận (branch-and-bound)

Ý tưởng nhánh và cận: Trong quá trình tìm kiếm một lối đi tốt nhất (tổng trọng số nhỏ nhất) cho bài toán TSP, có một kỹ thuật tỉa nhánh quan trọng là kết thúc sự tìm kiếm ngay khi thấy rằng nó không thể nào thành công được. Giả sử một lối đi đơn có chi phí x đã được tìm thấy. Thì thật vô ích để duyệt tiếp trên lối đi chưa đầy đủ nào mà chi phí cho đến hiện giờ đã lớn hơn x. Điều này có thể được thực hiện bằng cách không gọi đệ quy thủ tục visit nếu lối đi chưa-đầy-đủ hiện hành đã lớn hơn chi phí của lối đi đầy đủ tốt nhất cho đến bây giờ. Rõ ràng ta sẽ không bỏ sót lối đi chi phí nhỏ nhất nào nếu ta bám sát một chiến lược như vậy. Kỹ thuật tính cận (bound) của các lời giải chưa-đầy-đủ để hạn chế số lời giải phải dò tìm được gọi là giải thuật nhánh và cận .

Giải thuật này có thể áp dụng khi có chi phí được gắn vào các lối đi.

Bài toán người thương gia du hành (TSP) + Bài toán Chu trình Hamilton(HCP) : Để giải bài toán (HCP), ta có thể cải biên giải thuật tìm kiếm theo chiều sâu trước (DFS) để giải thuật này có thể sinh ra mọi lối đi đơn mà đi qua mọi đỉnh trong đồ thị.

NP-Complete

P : Tập hợp tất cả những bài toán có thể giải được bằng những giải thuật tất định trong thời gian đa thức.

NP: tập hợp tất cả những bài toán mà có thể được giải bằng giải thuật không tất định trong thời gian đa thức.

VD : Bài toán có tồn tại lối đi dài nhất từ đỉnh x đến đỉnh y ; Bài toán thỏa mãn mạch logic CSP là một bài toán thuộc lớp NP

Tất định : khi giải thuật đang làm gì, cũng chỉ có một việc duy nhất có thể được thực hiện kế tiếp.

VD : Xếp thứ tự bằng phương pháp chèn thuộc lớp P vì có độ phức tạp đa thức $O(N^2)$

Không tất định: khi một giải thuật gặp một sự lựa chọn giữa nhiều khả năng, nó có quyền năng "tiên đóan" để biết chọn một khả năng thích đáng. VD : Cho A là một mảng số nguyên. Một giải thuật không tất định NSORT(A, n) sắp thứ tự các số theo thứ tự tăng và xuất chúng ra theo thứ tự này.

Sự phân giải một giải thuật không tất định có thể được thực hiện bằng một sự song song hóa không hạn chế .Mỗi lần có bước lựa chọn phải thực hiện, giải thuật tạo ra nhiều bản sao của chính nó .Mỗi bản sao được thực hiện cho khả năng lựa chọn.

Như vậy nhiều khả năng được thực hiện cùng một lúc : +Bản sao đầu tiên kết

thúc thành công thì làm kết thúc tất cả các quá trình tính tóan khác + Nếu một bản sao kết thúc thất bại thì chỉ bản sao ấy kết thúc mà thôi.

NP-complete : Có một danh sách những bài toán mà đã biết là thuộc về lớp NP nhưng không rõ có thể thuộc về lớp P hay không. Tức là, ta giải chúng dễ dàng trên một máy không tất định nhưng chưa ai có thể tìm thấy một giải thuật hữu hiệu chạy trên máy tính thông thường để giải bất kỳ một bài toán nào của chúng.Những bài toán NP này lại có thêm một tính chất:"Nếu bất kỳ một trong những bài toán này có thể giải được trong thời gian đa thức thì tất cả những bài toán thuộc lớp NP cũng sẽ được giải trong thời gian đa thức trên một máy tất định." Đây là bài toán NP-complete . Để chứng minh một bài toán thuộc loại NP là NP-đầy đủ, ta chỉ cần chứng tỏ rằng một bài toán NP-đầy đủ đã biết nào đó thì khả thu giảm đa thức về bài toán mới ấy.

Một số bài toán NP-đầy đủ : - Bài toán thỏa mãn mạch logic CSP : Nếu tồn tại một giải thuật thời gian đa thức để giải bài toán thỏa mãn mạch logic thì tất cả mọi bài toán trong lớp NP có thể được giải trong thời gian đa thức - Bài toán phân hoạch số: Cho một tập những số nguyên, có thể phân hoạch chúng thành hai tập con mà có tổng trị số bằng nhau ? - Bài toán qui hoạch nguyên: Cho một bài toán qui hoạch tuyến tính, liệu có tồn tại một lời giải toàn số nguyên - Xếp lịch công việc trên đa bộ xử lý : Cho một kỳ hạn và một tập các công tác có chiều dài thời gian khác nhau phải được thực thi trên hai bộ xử lý. Vấn đề là có thể sắp xếp để thực thi tất cả những công tác đó sao cho thỏa mãn kỳ hạn không - Bài toán phủ đỉnh (VERTEX COVER): Cho một đồ thị và một số nguyên N, có thể kiếm được một tập nhỏ hơn N đỉnh mà chạm hết mọi cạnh trong đồ thị - Bài toán xếp thùng (BIN PACKING): cho n món đồ mà phải đặt vào trong các thùng có sức chứa bằng nhau L. Món đồ i đòi hỏi l_i đơn vị sức chứa của thùng. Mục đích là xác định số thùng ít nhất cần để chứa tất cả n món đồ đó.? Bài toán người thương gia du hành (TSP): cho một tập các thành phố và khoảng cách giữa mỗi cặp thành phố, tìm một lộ trình đi qua tất cả mọi thành phố sao cho tổng khoảng cách của lộ trình nhỏ hơn M+? Bài toán chu trình Hamilton (HCP): Cho một đồ thị, tìm một chu trình đơn mà đi qua tất cả mọi đỉnh.

Bài toán NP-đầy đủ trong các lãnh vực : giải tích số, sắp thứ tự và tìm kiếm, xử lý dòng ký tự, Mô hình hóa hình học, xử lý đồ thị. Sự đóng góp quan trọng nhất của lý thuyết về NP-đầy đủ là: nó cung cấp một cơ chế để xác định một bài toán mới trong các lãnh vực trên là "dễ" hay "khó".Một số kỹ thuật để đối phó với những bài toán NP-đầy đủ : + Dùng "giải thuật xấp xỉ để tìm lời giải xấp xỉ tối ưu (near-optimal) + Dựa vào hiệu năng của trường hợp trung bình để phát triển một giải thuật mà tìm ra lời giải trong một số trường hợp nào đó, mặc dù không làm việc được trong mọi trường hợp+ Sử dụng những giải thuật có độ phức tạp hàm mũ nhưng hữu hiệu, ví dụ như giải thuật quay lui+ Đưa heuristic vào giải thuật để tăng thêm hiệu quả của giải thuật+ Sử dụng metaheuristic.

Heuristic là tri thức về bài toán cụ thể được sử dụng để dẫn dắt quá trình tìm ra lời giải của giải thuật. Nhờ sự thêm vào các heuristic mà giải thuật trở nên hữu hiệu hơn.

Meta heuristic là loại heuristic tổng quát có thể áp dụng cho nhiều lớp bài tóan.Gần đây meta heuristic là một lãnh vực nghiên cứu phát triển mạnh mẽ, với sự ra đời của nhiều meta heuristic như:- giải thuật di truyền - giải thuật mô phỏng luyện kim - tìm kiếm tabu (Tabu search) …

Bốn lớp bài toán phân theo độ khó:

Những bài toán bất khả quyết : Đây là những bài toán chưa hề có giải thuật để giải. VD: Bài toán quyết định xem một chương trình có dừng trên một máy Turing

+ Những bài toán khó giải : đây là những bài toán mà không tồn tại giải thuật thời gian đa thức để giải chúng. Chỉ tồn tại giải thuật thời gian hàm mũ để giải chúng

+Những bài toán NP-đầy đủ : Những bài toán NP-đầy đủ là một lớp con đặc biệt của lớp bài toán NP + Những bài toán P.

Cách đơn giản nhất để tìm nghiệm tối ưu của một bài toán là duyệt hết toàn bộ tập nghiệm của bài toán đó (vét cạn). Cách này chỉ áp dụng được khi tập nghiệm nhỏ, kích thước vài chục byte. Khi gặp những bài toán với tập nghiệm lớn thì phương pháp trên không đáp ứng được yêu cầu về mặt thời gian tính toán. Nếu tìm đúng hệ thức thể hiện bản chất quy hoạch động của bài toán và khéo tổ chức dữ liệu thì ta có thể xử lí được những tập dữ liệu khá lớn.

Quy hoạch động cũng như chia để trị là các phương pháp giải một bài toán bằng cách tổ hợp lời giải các bài toán con của nó.

Phương pháp quy hoạch động cùng nguyên lí tối ưu được nhà toán học Mỹ Richard Bellman (1920 - 1984) đề xuất vào những năm 50 của thế kỷ 20. Phương pháp này đã được áp dụng để giải hàng loạt bài toán thực tế trong các quá trình kỹ thuật công nghệ, tổ chức sản xuất, kế hoạch hóa kinh tế,… Tuy nhiên cần lưu ý rằng có một số bài toán mà cách giải bằng quy hoạch động tỏ ta không thích hợp.

Ưu điểm

Điểm khác nhau cơ bản giữa quy hoạch động và phương pháp phân rã là :

- Phương pháp phân rã giải quyết bài toán theo hướng top-down, nghĩa là để giải bài toán ban đầu, ta phải đi giải tất cả các bài toán con của nó. Đây là một phương pháp hay, tuy nhiên phương pháp này sẽ gặp hạn chế về mặt thời gian, tốc độ do phải tính đi tính lại nhiều lần một số bài toán con giống nhau nào đó.

- Phương pháp quy hoạch động sử dụng nguyên lý bottom-up, nghĩa là "đi từ dưới lên". Đầu tiên, ta sẽ phải các bài toán con đơn giản nhất, có thể tìm ngay ra nghiệm. Sau đó kết hợp các bài toán con này lại để tìm lời giải cho bài toán lớn hơn và cứ như thế cho đến khi giải được bài toán yêu cầu. Với phương pháp này, mỗi bài toán con sau khi giải xong đều được lưu trữ lại và đem ra sử dụng nếu cần. Do đó tiết kiệm bộ nhớ và cải thiện được tốc độ.

Hạn chế

Không phải lúc nào việc kết hợp các bài toán con cũng cho ta kết quả của bài toán lớn hơn. Hay nói cách khác là việc tìm kiếm "công thức truy hồi" rất khó khăn. Số lượng các bài toán con cần lưu trữ có thể rất lớn, không chấp nhận được vì dữ liệu và bộ nhớ máy tính không cho phép.

1.2. Thuật toán chia để trị

Đối với nhiều thuật toán đệ quy, nguyên lý chia để trị (divide and conquer) thường đóng vai trò chủ đạo trong việc thiết kế thuật toán. Để giải quyết một bài toán lớn, ta chia nó làm nhiều bài toán con cùng dạng với nó để có thể giải quyết độc lập.

Khi giải một bài toán P với kích thước ban đầu nào đó nếu gặp trở ngại vì kích thước quá lớn, người ta thường nghĩ đến việc giải các bài toán tương tự nhưng với kích thước nhỏ hơn (gọi là các bài toán con của P). Tư tưởng chia để trị thường được nhắc tới như hình ảnh "bẻ dần từng chiếc đũa để bẻ gãy cả bó đũa".

Chia để trị thực hiện "tách" một bài toán ban đầu thành các bài toán *con độc lập,* các bài toán con cùng được sinh ra sau mỗi lần "tách" được gọi là cùng mức. Những bài toán con sinh ra sau hơn thì ở mức dưới (thấp hơn) và cứ tiến hành như vậy cho đến khi gặp các bài toán nhỏ đến mức dễ dàng giải được. Sau đó giải các bài toán con này và tổ hợp dần lời giải từ bài toán con nhỏ nhất đến bài toán ban đầu.

Thủ tục đệ quy luôn là cách thường dùng và hiệu quả để thực hiện thuật toán chia để trị. Quá trình đệ quy lần lượt xếp dần các bài toán con vào ngăn xếp bộ nhớ và sẽ thực hiện giải các bài toán con theo thứ tự ngược lại từ bài toán đơn giản nhất trên đỉnh ngăn xếp cho đến khi giải được bài toán ban đầu ở đáy ngăn xếp .

Ví dụ: Tìm số hạng thứ N của dãy Fibonacci. Công thức đệ quy (truy hồi) của dãy Fibonaci: $F(1) = 1$, $F(2) = 1$, $F(N) = F(N-1) + F(N-2)$ với $N > 2$.

Lời giải.

Xây dựng hàm $F()$ để tính số hạng thứ N của dãy Fibonacci theo đúng định nghĩa toán học của dãy.

```
Function  F(N:integer): longint;
Begin
    If (N=1) or (N=2) then F:=1
    Else F:=F(N-1)+F(N-2);
End;
```

Với cách này khi gọi $F(N)$, đã sinh ra các lời gọi cùng một bài toán con tại nhiều thời điểm khác nhau. Ngăn xếp chứa các biến tương ứng với các lời gọi hàm nhanh chóng tăng nhanh dễ dẫn tới tràn ngăn xếp. Ví dụ khi gọi $F(5)$, đã lần lượt gọi

1. $F(5)$

2. $F(4) + F(3)$

3. $(F(3) + F(2)) + (F(2) + F(1))$

4. $((F(2) + F(1)) + F(2)) + F(2) + F(1)$

Như vậy đã ba lần gọi $F(2)$. Khi $N = 40$, số lần gọi $F(2)$ đã tăng tới 63245986 lần. Thời gian thực hiện chương trình khá lâu vì số lần gọi hàm quá lớn, gần như tăng theo hàm mũ.

1.3. Nguyên lý tối ưu của Bellman

Trong thực tế, ta thường gặp một số bài toán tối ưu loại sau: Có một đại lượng *f* hình thành trong một quá trình gồm nhiều giai đoạn và ta chỉ quan tâm đến kết quả cuối cùng là giá trị của *f* phải lớn nhất hoặc nhỏ nhất, ta gọi chung là giá trị tối ưu của *f*. Giá trị của *f* phụ thuộc vào những đại lượng xuất hiện trong bài toán mà mỗi bộ giá trị của chúng được gọi là một *trạng thái* của hệ thống và cũng phụ thuộc vào cách thức đạt được giá trị *f* trong từng giai đoạn mà mỗi cách thức được gọi là một *điều khiển*. Đại lượng *f* thường được gọi là *hàm mục tiêu* và quá trình đạt được giá trị tối ưu của *f* được gọi là *quá trình điều khiển tối ưu*. Có thể tóm lược nguyên lí quy hoạch động do Bellman phát biểu như sau: Quy hoạch động là lớp các bài toán mà quyết định ở bước thứ i phụ thuộc vào quyết định ở các bước đã xử lí trước hoặc sau đó.

Chú ý rằng nguyên lý này được thừa nhận mà không chứng minh.

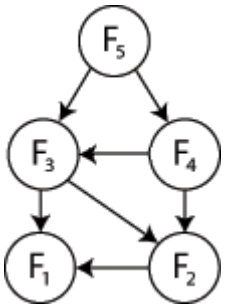
Phương pháp tìm điều khiển tối ưu theo nguyên lý Bellman thường được gọi là *quy hoạch động*.

1.4. Đặc điểm chung của phương pháp quy hoạch động

Phương pháp quy hoạch động dùng để giải bài toán tối ưu có bản chất đệ quy, tức là việc tìm phương án tối ưu cho bài toán đó có thể đưa về tìm phương án tối ưu của một số hữu hạn các bài toán con.

Phương pháp quy hoạch động giống phương pháp chia để trị ở chỗ: lời giải bài toán được tổ hợp từ lời giải các bài toán con. Trong phương pháp quy hoạch động, nguyên lý này càng được thể hiện rõ. Khi không biết cần phải giải quyết những bài toán con nào, ta sẽ đi giải quyết các bài toán con và lưu trữ những lời giải hay đáp số của chúng với mục đích sử dụng lại theo một sự phối hợp nào đó để giải quyết những bài toán tổng quát hơn.

"Chia để trị" sẽ phân chia bài toán ban đầu thành bài toán con độc lập (hiểu theo nghĩa sự phân chia có cấu trúc dạng cây), giải các bài toán con này thường bằng đệ quy, sau đó tổ hợp lời giải của chúng để được lời giải của bài toán ban đầu. Quy hoạch động cũng phân chia bài toán thành các bài toán con, nhưng các bài toán con phụ thuộc nhau, mỗi bài toán con có thể tham chiếu tới cùng một số bài toán con mức dưới (gọi là các bài toán con gối lên nhau, sự phân chia không có cấu trúc dạng cây).



Hình 1.1. Đồ thị mô tả quan hệ giữa các bài toán con của bài toán tìm số hạng thứ năm của dãy Fibonacci

Đồ thị này không là cây nhưng là một đồ thị có hướng phi chu trình. Mỗi bài toán có những bài toán con gối lên nhau đó là hiện tượng có bài toán con đồng thời được sử dụng để giải bài toán khác với kích thước lớn hơn. Ví dụ $F_3 = F_1 + F_2$ và $F_4 = F_2 + F_3$ nên việc tính mỗi số F_3 hoặc F_4 đều phải tính F_2. Mặt khác cả F_3 và F_4 đều cần cho tính F_5 do đó để tính F_5 cần phải tính F_2 ít nhất hai lần. Điều tính toán này được áp dụng ở bất cứ chỗ nào có bài toán con gối nhau xuất hiện sẽ tiêu phí thời gian để tìm lại kết quả tối ưu của những bài toán con đã được giải lúc trước. Để tránh điều này, thay cho việc giải lại các bài toán con, chúng ta lưu kết quả những bài toán con đã giải. Khi giải những bài toán sau (mức cao hơn), chúng ta có thể khôi phục lại những kết quả đã lưu và sử dụng chúng. Cách tiếp cận này được gọi là cách *ghi nhớ* (lưu trữ vào bộ nhớ máy tính những kết quả đã tính để phục vụ cho việc tính các kết quả tiếp theo). *Ghi nhớ* là một đặc trưng đẹp đẽ của quy hoạch động. Người ta cũng còn gọi cách tiếp cận này là cách lập bảng phương án lưu trữ những kết quả đã tính được để khi cần có thể sử dụng lại. Nếu ta chắc chắn rằng một lời giải nào đó không còn cần thiết nữa, ta có thể xóa nó đi để tiết kiệm không gian bộ nhớ. Trong một số trường hợp, ta còn có thể tính lời giải cho các bài toán con mà ta biết trước rằng sẽ cần đến.

Bài toán tối ưu P cần đến lập trình động khi có hai đặc điểm sau đây:

- Bài toán P thỏa mãn nguyên lí tối ưu Bellman. Khi đó người ta nói bài toán P có cấu trúc con tối ưu, nghĩa là có thể sử dụng lời giải tối ưu của các bài toán con từ mức thấp để tìm dần lời giải tối ưu cho bài toán con ở các mức cao hơn, và cuối cùng là lời giải tối ưu cho bài toán toàn thể.

- Bài toán P có các bài toán con phủ chồng (gối) lên nhau. Nghĩa là không gian các bài toán con "hẹp" không tạo thành dạng hình cây (tree). Nếu gọi hai bài toán con cùng được sinh ra từ một bài toán là hai bài toán con cùng mức thì có thể mô tả hình ảnh các bài toán con phủ chồng lên nhau là: khi giải hai bài toán con cùng mức chúng có thể đòi hỏi cùng tham chiếu một số bài toán con thuộc mức dưới chúng.

Quy hoạch động là một phương pháp phân tích và thiết kế thuật toán cho phép giảm bớt thời gian thực hiện khi khai thác tốt hai đặc điểm nêu trên. Tuy nhiên thông thường quy hoạch động lại đòi hỏi nhiều không gian bộ nhớ hơn (để thực hiện ghi nhớ). Ngày nay, với sự mở rộng bộ nhớ máy tính và nhiều phần mềm lập trình mới cho phép sử dụng bộ nhớ rộng rãi hơn thì phương pháp quy hoạch động càng có nhiều khả năng giải các bài toán trước đây khó giải quyết do hạn chế bộ nhớ máy tính .

1.5. Ý tưởng và nội dung của thuật toán quy hoạch động

1.5.1. Các khái niệm

- Bài toán giải theo phương pháp quy hoạch động gọi là bài toán quy hoạch động
- Công thức phối hợp nghiệm của các bài toán con để có nghiệm của bài toán lớn gọi là công thức truy hồi của quy hoạch động
- Tập các bài toán nhỏ nhất có ngay lời giải để từ đó giải quyết các bài toán lớn hơn gọi là cơ sở quy hoạch động
- Không gian lưu trữ lời giải các bài toán con để tìm cách phối hợp chúng gọi là bảng phương án của quy hoạch động .

1.5.2. Ý tưởng

Quy hoạch động bắt đầu từ việc giải các bài toán nhỏ nhất (bài toán cơ sở) để từ đó từng bước giải quyết những bài toán lớn hơn, cho tới khi giải được bài toán lớn nhất (bài toán ban đầu). Vậy ý tưởng cơ bản của quy hoạch động là : *Tránh tính toán lại mọi thứ hai lần, mà lưu giữ kết quả đã tìm kiếm được vào một bảng làm giả thiết cho việc tìm kiếm những kết quả của trường hợp sau*.

Chúng ta sẽ làm đầy dần giá trị của bảng này bởi các kết quả của những trường hợp trước đã được giải. Kết quả cuối cùng chính là kết quả của bài toán cần giải. Nói cách khác phương pháp quy hoạch động đã thể hiện sức mạnh của nguyên lý chia để trị đến cao độ [7].

Tư tưởng của thuật toán quy hoạch động khá đơn giản. Tuy nhiên khi áp dụng thuật toán vào trường hợp cụ thể lại không dễ dàng (điều này cũng tương tự như nguyên tắc Dirichlet trong toán học).

1.5.3. Nội dung

Quy hoạch động là kỹ thuật thiết kế bottom-up (từ dưới lên). Nó được bắt đầu với những trường hợp con nhỏ nhất (thường là đơn giản nhất và giải được ngay). Bằng cách tổ hợp các kết quả đã có (không phải tính lại) của các trường hợp con, sẽ đạt tới kết quả của trường hợp có kích thước lớn dần lên và tổng quát hơn, cho đến khi cuối cùng đạt tới lời giải của trường hợp tổng quát nhất.

Trong một số trường hợp, khi giải một bài toán A, trước hết ta tìm họ bài toán $A(p)$ phụ thuộc tham số p (có thể p là một véc tơ) mà $A(p_0)=A$ với p_0 là trạng thái ban đầu của bài toán A. Sau đó tìm cách giải họ bài toán $A(p)$ với tham số p bằng cách áp dụng nguyên lý tối ưu của Bellman. Cuối cùng cho $p = p_0$ sẽ nhận được kết quả của bài toán A ban đầu [7].

1.6. Các bước thực hiện

Bước 1: ***Lập hệ thức***

Dựa vào nguyên lý tối ưu tìm cách chia quá trình giải bài toán thành từng giai đoạn, sau đó tìm hệ thức biểu diễn tương quan quyết định của bước đang xử lý với các bước đã xử lý trước đó. Hoặc tìm cách phân rã bài toán thành các "bài

toán con" tương tự có kích thước nhỏ hơn, tìm hệ thức nêu quan hệ giữa kết quả bài toán kích thước đã cho với các kết quả của các "bài toán con" cùng kiểu có kích thước nhỏ hơn của nó dạng hàm hoặc thủ tục đệ quy.

Khi đã có hệ thức tương quan chúng ta có thể xây dựng ngay thuật giải, tuy nhiên hệ thức này thường là các biểu thức đệ quy, do đó dễ gây ra hiện tượng tràn miền nhớ khi ta tổ chức chương trình trực tiếp bằng đệ quy.

Bước 2: ***Tổ chức dữ liệu và chương trình***

Tổ chức dữ liệu sao cho đạt các yêu cầu sau:

a) Dữ liệu được tính toán dần theo các bước.

b) Dữ liệu được lưu trữ để giảm lượng tính toán lặp lại.

c) Kích thước miền nhớ dành cho lưu trữ dữ liệu càng nhỏ càng tốt, kiểu dữ liệu được chọn phù hợp, nên chọn đơn giản dễ truy cập.

Bước 3: ***Làm tốt***

Làm tốt thuật toán bằng cách thu gọn hệ thức và giảm kích thước miền nhớ. Thường tìm cách dùng mảng một chiều thay cho mảng hai chiều nếu giá trị một dòng (hoặc cột) của mảng hai chiều chỉ phụ thuộc một dòng (hoặc cột) kề trước.

Trong một số trường hợp có thể thay mảng hai chiều với các giá trị phần tử chỉ nhận giá trị 0, 1 bởi mảng hai chiều mới bằng cách dùng kỹ thuật quản lý bit .

KẾT LUẬN CHƯƠNG 1

Phương pháp quy hoạch động là phương pháp hay được dùng để giải các bài tập tin học, đặc biệt các bài tập trong các kỳ thi học sinh giỏi và một số bài tập trong thực tế.

Khi giải bài toán bằng phương pháp quy hoạch động, chúng ta phải thực hiện hai yêu cầu quan trọng sau:

- Tìm công thức truy hồi xác định nghiệm bài toán qua nghiệm các bài toán con nhỏ hơn.
- Với mỗi bài toán cụ thể, ta đề ra phương án lưu trữ nghiệm một cách hợp lý để từ đó có thể truy cập một cách thuận tiện nhất.

Cho đến nay, vẫn chưa có một định lý nào cho biết một cách chính xác những bài toán nào có thể giải quyết hiệu quả bằng quy hoạch động. Tuy nhiên để biết được bài toán có thể giải bằng quy hoạch động hay không, ta có thể tự đặt câu hỏi: *"Một nghiệm tối ưu của bài toán lớn có phải là sự phối hợp các nghiệm tối ưu của các bài toán con hay không?" và "Liệu có thể nào lưu trữ được nghiệm các bài toán con dưới một hình thức nào đó để phối hợp tìm được nghiệm bài toán lớn"*.

Việc tìm công thức truy hồi hoặc tìm cách phân rã bài toán nhiều khi đòi hỏi sự phân tích tổng hợp rất công phu, dễ sai sót, khó nhận ra như thế nào là thích hợp, đòi hỏi nhiều thời gian suy nghĩ. Đồng thời không phải lúc nào kết hợp lời giải của các bài toán con cũng cho kết quả của bài toán lớn hơn. Khi bảng lưu trữ đòi hỏi mảng hai, ba chiều… thì khó có thể xử lý dữ liệu với kích cỡ mỗi chiều lớn hàng trăm.

Chương 2
MỘT SỐ KỸ THUẬT GIẢI BÀI TOÁN QUY HOẠCH ĐỘNG

2.1. Lập hệ thức

Thông thường khi dùng phương pháp quay lui, vét cạn cho các bài toán quy hoạch động thì chỉ có thể vét được các tập dữ liệu nhỏ, kích thước chừng vài chục byte. Nếu tìm được đúng hệ thức thể hiện bản chất quy hoạch động của bài toán và khéo tổ chức dữ liệu thì ta có thể xử lý được những tập dữ liệu khá lớn .

2.1.1. Tạo một công thức truy hồi từ một công thức đã có

Đôi khi bài toán ban đầu đã cho một công thức truy hồi nhưng nếu ta áp dụng luôn công thức đó thì không thể đáp ứng được yêu cầu về thời gian và bộ nhớ vì phát sinh nhiều lần gọi hàm trùng lặp và nếu có lưu trong bảng phương án thì tốn quá nhiều bộ nhớ. Vì vậy từ công thức truy hồi đã cho trước ta có thể tìm ra một công thức truy hồi mới mặc dù công thức mới có thể phức tạp hơn nhưng nó sẽ giúp ta không phải lưu trữ nhiều trong bảng và làm việc được với dữ liệu rất lớn.

Ví dụ: Hàm f(n)

Tính hàm $f(n)$ với biến số nguyên n cho trước, $0 \le n \le 1.000.000.000$ (1 tỷ). Biết: $f(0) = 0$; $f(1) = 1$; $f(2n) = f(n)$; $f(2n+1) = f(n) + f(n+1)$.

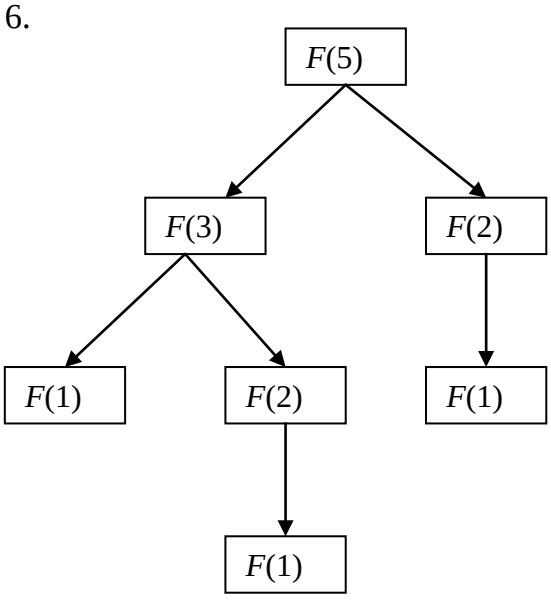
Thuật toán 1

Dựa vào công thức truy hồi đã cho ta có thể viết được ngay đoạn chương trình bằng đệ quy. Bảng dưới đây liệt kê số lần gọi hàm $f(n)$ khi giải bài toán với $n = 5$.

Bảng 2.1. Bảng số lần gọi hàm f(n) với n = 5

N	0	1	2	3	4	5
Số lần gọi hàm *f*(*n*)	0	3	2	1	0	1

Ví dụ, $f(1)$ được gọi ba lần, $f(2)$ được gọi hai lần, $f(3)$ được gọi một lần,... tổng số lần gọi hàm f là 6.



Hình 2.1. Cây biểu diễn lời gọi hàm f của bài toán tính hàm f(5)

Đoạn chương trình viết bằng ngôn ngữ Pascal

```
Function  f(n:int64):int64;
Begin
   c[n]:=c[n]+1; {c[i]: số lần gọi hàm f(i)}
   if (n=0) then f:=0
   else if (n=1) then f:=1
      else if (n mod 2 = 0) then f:=f(n div 2)
         else f:= f(n div 2) + f(n div 2+1);
End;
```

Thuật toán 2

Dùng bảng để lưu lại giá trị của hàm *f* để tránh những lần gọi lặp. Tuy nhiên với *n* = 1000000000 thì không đủ bộ nhớ để lưu trữ giá trị của hàm *f*.

Đoạn chương trình viết bằng ngôn ngữ Pascal

```
Procedure  Make_table;
Var i:longint;
Begin
   f[0]:=0; f[1]:=1;
   for i:=2 to n do
     if i mod 2 = 0 then f[i]:=f[i div 2]
     else f[i]:= f[i div 2] + f[i div 2 +1];
End;
```

Độ phức tạp: O(n)

Thuật toán 3

Từ hàm đã cho ta có thể xây dựng hàm mới có giá trị tương đương với hàm đã cho.

Xét hàm 3 biến $g(n,a,b) = af(n) + bf(n+1)$. Ta có

1) $g(n,1,0) = 1.f(n) + 0.f(n+1) = f(n)$.

2) $g(0,a,b) = af(0) + bf(1) = a.0 + b.1 = b$.

3) $g(2n,a,b) = af(2n) + bf(2n+1) = af(n) + bf(n) + bf(n+1) = (a+b)f(n) + bf(n+1) = g(n,a+b,b)$.

4) $g(2n+1,a,b) = af(2n+1) + bf(2n+2) = af(n) + af(n+1) + bf(2(n+1)) = af(n) + af(n+1) + bf(n+1) = af(n) + (a+b)f(n+1) = g(n,a,a+b)$.

Từ bốn tính chất trên ta thiết kế được hàm $f(n)$ như sau:

Để tính $f(n)$ ta tính $g(n,a,b)$ với $a = 1$, $b = 0$. Để tính $g(n)$ ta lặp đến khi $n = 0$. Nếu n chẵn ta gọi hàm $g(n/2,a+b,b)$; ngược lại, nếu n lẻ ta gọi hàm $g(n/2,a,a+b)$. Khi $n = 0$ ta thu được $f = g(0,a,b) = b$.

Đoạn chương trình viết bằng ngôn ngữ Pascal

```
Function f(n: longint): longint;
Var a,b: longint;
Begin
  a := 1; b := 0;
  while (n <> 0) do
  begin
    if odd(n) then b := b + a else a := a + b;
    n := n div 2;
  end;
  f := b;
End;
```

Độ phức tạp: $\log_2(n)$ vòng lặp .

Dữ liệu test

$f(100) = 7$; $f(101) = 19$; $f(1000000) = 191$; $f(1000000000) = 7623$.

2.1.2. Dựa theo thứ tự xây dựng

Để xây dựng công thức truy hồi ta có thể dựa theo đối tượng đầu tiên hay đối tượng cuối cùng trong tập các đối tượng đang xét.

2.1.2.1. Xây dựng dựa theo thứ tự đầu

Dựa theo đối tượng đầu tiên ta có thể chia bài toán ban đầu thành nhiều bài toán con. Sau đó sử dụng kết quả của các bài toán con đó để tìm nghiệm của bài toán. Chẳng hạn như bài *Cóc* sau đây dựa theo bước nhảy đầu tiên ta có thể chia các cách nhảy thành nhiều nhóm không giao nhau.

Ví dụ: ***Bài* Cóc.**

Một chú cóc máy có thể nhảy k bước với độ dài khác nhau (b_1, b_2, ..., b_k) trên đoạn đường thẳng. Đặt cóc trên đoạn đường thẳng tại vạch xuất phát 0. Cho biết số cách nhảy để cóc đến được điểm N.

Thí dụ, số bước $k = 2$, $b_1 = 2$, $b_2 = 3$, đoạn đường dài $N = 8$.

Có 4 cách: (2,2,2,2) (2, 3, 3) (3, 3, 2) (3, 2, 3).

Thuật toán: Quy hoạch động.

Gọi $S(n)$ là số cách để cóc vượt đoạn đường dài n. Dựa theo bước nhảy đầu tiên ta chia toàn bộ các phương án nhảy của cóc thành k nhóm không giao nhau.

- Nhóm 1 sẽ gồm các phương án bắt đầu bằng bước nhảy độ dài b_1, tức là gồm các phương án dạng $(b_1,...)$. Sau bước nhảy đầu tiên, cóc vượt đoạn đường b_1, đoạn đường còn lại sẽ là $n–b_1$, do đó tổng số phương án của nhóm này sẽ là $S(n–b_1)$.
- Nhóm 2 sẽ gồm các phương án bắt đầu bằng bước nhảy độ dài b_2, tức là gồm các phương án dạng $(b_2,...)$. Sau bước nhảy đầu tiên, cóc vượt đoạn đường b_2, đoạn đường còn lại sẽ là $n–b_2$, do đó tổng số phương án của nhóm này sẽ là $S(n–b_2)$.
- ...
- Nhóm i sẽ gồm các phương án bắt đầu bằng bước nhảy độ dài b_i, tức là gồm các phương án dạng $(b_i,...)$. Sau bước nhảy đầu tiên, cóc vượt đoạn đường b_i, đoạn đường còn lại sẽ là $n–b_i$, do đó tổng số phương án của nhóm này sẽ là $S(n–b_i)$.
- ...
- Nhóm k sẽ gồm các phương án bắt đầu bằng bước nhảy độ dài b_k, tức là gồm các phương án dạng $(b_k,...)$. Sau bước nhảy đầu tiên cóc vượt đoạn đường b_k, đoạn đường còn lại sẽ là $n–b_k$, do đó tổng số phương án của nhóm này sẽ là $S(n–b_k)$.

Dĩ nhiên, bước đầu tiên là b_i sẽ được chọn nếu $b_i \le n$.

Vậy ta có:

$$S(n) = \Sigma\{ S(n–b_i) \mid n \ge b_i \; i = 1, 2, \dots , k \} \qquad (*)$$

Để ý rằng khi đoạn đường cần vượt có chiều dài $n = 0$ thì cóc có một cách nhảy (đứng yên), do đó $S(0) = 1$. Ngoài ra ta quy định $S(n) = 0$ nếu $n < 0$.

Sử dụng mảng s ta tính dần các trị s[0] = 1; s[1], s[2],..., s[n] theo hệ thức (*) nói trên. Kết quả được lưu trong s[n] [7].

Đoạn chương trình viết bằng ngôn ngữ Pascal

```
Procedure  make_table;
Var d,i:byte;v:longint;
Begin
   s[0]:=1;
   for d:=1 to n do
     begin
       v:=0;
       for i:=1 to k do
         if (d>= b[i]) then v:=v+s[d-b[i]];
       s[d]:=v;
     end;
```

End;

Độ phức tạp: Với mỗi giá trị *n* ta tính *k* bước nhảy, vậy độ phức tạp thời gian cỡ *k.n*.

2.1.2.2. Xây dựng theo thứ tự cuối

Dựa theo đối tượng cuối cùng trong tập các đối tượng đang xét, ta xét các trường hợp có thể xảy ra nếu nhận hoặc không nhận đối tượng cuối. Từ đó ta cũng có thể chia bài toán lớn thành nhiều bài toán con. Chẳng hạn bài toán *chia thưởng* dưới đây xét từ người cuối cùng, bài *du hành* xét bit cuối cùng.

Ví dụ 1. Bài toán *Chia thưởng*

Cần chia hết *m* phần thưởng cho *n* học sinh sắp theo thứ tự từ giỏi trở xuống sao cho mỗi bạn không nhận ít phần thưởng hơn bạn xếp sau mình. Với $1 \le m, n \le 100$. Hãy tính số cách chia.

Thí dụ, với số phần thưởng là 7, số học sinh là 4 sẽ có 11 cách chia

Bảng 2.2. Các phương án chia kẹo với m = 7, n = 4

Phương án	①	②	③	④
1	**7**	**0**	**0**	**0**
2	**6**	**1**	**0**	**0**
3	**5**	**2**	**0**	**0**
4	**5**	**1**	**1**	**0**
5	**4**	**3**	**0**	**0**
6	**4**	**2**	**1**	**0**
7	**3**	**3**	**1**	**0**
8	**3**	**2**	**2**	**0**
9	**4**	**1**	**1**	**1**
10	**3**	**2**	**1**	**1**
11	**2**	**2**	**2**	**1**

Thuật toán: Quy hoạch động

- Lập hệ thức

Gọi $C(i, j)$ là số cách chia *i* phần thưởng cho *j* học sinh, ta thấy:

- Nếu không có học sinh nào ($j = 0$) thì không có cách chia nào ($C(i, 0) = 0$).

- Nếu không có phần thưởng nào ($i = 0$) thì chỉ có một cách chia ($C(0, j) = 1$ - mỗi học sinh nhận 0 phần thưởng). Ta cũng quy ước $C(0, 0) = 1$.

- Nếu số phần thưởng ít hơn số học sinh ($i < j$) thì trong mọi phương án chia, từ học sinh thứ $i + 1$ trở đi sẽ không được nhận phần thưởng nào:

$C(i, j) = C(i, i)$ nếu $i < j$.

Ta xét tất cả các phương án chia trong trường hợp $i \geq j$. Ta tách các phương án chia thành hai nhóm không giao nhau dựa trên số phần thưởng mà học sinh đứng cuối bảng thành tích, học sinh thứ j, được nhận:

- Nhóm thứ nhất gồm các phương án trong đó học sinh thứ j không được nhận thưởng, tức là i phần thưởng chỉ chia cho $j - 1$ học sinh và do đó, số cách chia, tức là số phần tử của nhóm này sẽ là: $C(i, j - 1)$.
- Nhóm thứ hai gồm các phương án trong đó học sinh thứ j cũng được nhận thưởng. Khi đó, do học sinh đứng cuối bảng thành tích được nhận thưởng thì mọi học sinh khác cũng sẽ có thưởng. Do ai cũng được thưởng nên ta bớt của mỗi người một phần thưởng (để họ lĩnh sau), số phần thưởng còn lại ($i - j$) sẽ được chia cho j học sinh. Số cách chia khi đó sẽ là $C(i - j, j)$.

Tổng số cách chia cho trường hợp $i \geq j$ sẽ là tổng số phần tử của hai nhóm, ta có:

$C(i, j) = C(i, j - 1) + C(i - j, j)$.

Tổng hợp lại ta có:

Điều kiện	
i: số phần thưởng	$C(i, j)$
j: số học sinh	
$j = 0$	$C(i, j) = 0$
$i = 0$ and $j \neq 0$	$C(i, j) = 1$
$i < j$	$C(i, j) = C(i, i)$
$i \geq j$	$C(i, j) = C(i, j - 1) + C(i - j, j)$

Ví dụ 2. Du hành

ACM/ICPC 2011

Đoàn du hành có nhiệm vụ xuất phát từ hành tinh s tìm cách tốn ít năng lượng nhất để đến được hành tinh e. Mỗi hành tinh có mã số là một dãy nhị phân dài n (gồm các giá trị 0 và 1). Biết

- Từ hành tinh x có thể đến được hành tinh y nếu x và y khác nhau tại đúng 1 vị trí.
- Mỗi khi hạ cánh xuống hành tinh $y = (y_1, y_2, ..., y_n)$ thì năng lượng chi phí sẽ là

$p_1y_1 + p_2y_2 + ... + p_ny_n$.

Input: Gồm nhiều test kết thúc bằng dòng 0. Mỗi test chiếm 2 dòng văn bản:

- Dòng thứ nhất: ba giá trị n s e; $1 \le n \le 1000$.

- Dòng thứ hai: n giá trị p_1 p_2 ... p_n là các chi phí tương ứng với vị trí thứ i trong mã số hành tinh.

Output: Với mỗi test hiển thị chi phí nhỏ nhất để di chuyển từ hành tinh s đến hành tinh e.

Ví dụ

Input

3 110 011

3 1 2

5 00000 11111

1 2 3 4 5

4 1111 1000

100 1 1 1

30 000000000000000000000000000000 111111111111111111111111111111

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30

0

Output

4

35

106

4960

Thuật toán Quy hoạch động.

Ta phát biểu lại bài toán như sau:

Cho 2 xâu bit 0/1 s và e cùng chiều dài n bit. Cần tính chi phí thấp nhất để biến đổi s thành e theo các điều kiện sau:

- Mỗi bước biến đổi chỉ được phép đảo duy nhất 1 bit.
- Mỗi khi biến đổi xâu x[1..n] thành xâu y[1..n] thì phải trả thêm chi phí Cost(y) được tính bằng tổng các giá trị p[i] ứng với vị trí y[i] = '1', $1 \le i \le n$.

Ta giả thiết là các chi phí tại thành phần i, p[i] được sắp tăng và các xâu s và e cũng được sắp tương ứng theo p.

Gọi $c(i)$ là hàm chi phí thấp nhất khi phải biến đổi đoạn xâu bit (tiền tố) s[1..i] thành xâu bit e[1..i], $b(i)$ là hàm cho số bước ứng với phép biến đổi đoạn xâu bit s[1..i] thành xâu bit e[1..i] theo cách trên, $v(y,i)$ là tổng các giá trị p[k] ứng với vị trí y[k] = '1', $1 \le k \le i$. $v(y,i)$ chính là chi phí phải trả khi hạ cánh xuống hành tinh có

mã y[1..i]. Ta kí hiệu A và B là hai tiền tố gồm i-1 giá trị đầu của s và e, tức là A = s[1..i-1], B = e[1..i-1]. Ta xét bit i theo 4 trường hợp a, b, c và d sau đây:

a) A0, B0 tức là s[i] = '0' và e[i] = '0': Ta chỉ cần biến đổi A→B là hoàn tất. Ta có

- Số bước cần thiết: $b(i) = b(i\text{-}1)$;
- Chi phí: $c(i) = c(i\text{-}1)$.

b) A0, B1 tức là s[i] = '0' và e[i] = '1': Trước hết ta biến đổi A→B, giữ nguyên s[i] = 0 sau đó biến đổi thêm 1 bước để lật s[i] từ '0' sang 1 với chi phí $v(e,i)$. Ta có

- Số bước cần thiết: $b(i) = b(i\text{-}1)+1$;
- Chi phí: $c(i) = c(i\text{-}1) + v(e,i)$.

c) A1, B0 tức là s[i] = '1' và e[i] = '0': Trước hết ta lật s[i] từ '1' thành '0' với chi phí $v(s,i\text{-}1)$ sau đó biến đổi A→B. Ta có

- Số bước cần thiết: $b(i) = b(i\text{-}1)+1$;
- Chi phí: $c(i) = c(i\text{-}1) + v(s,i\text{-}1)$.

d) A1, B1 tức là s[i] = '1' và e[i] = '1': Ta cần chọn chi phí min theo hai khả năng d.1 và d.2 sau đây:

d.1 Giữ nguyên s[i] và e[i], chỉ biến đổi A→B

- Số bước cần thiết: $b(i) = b(i\text{-}1)$;
- Chi phí: $c(i) = c(i\text{-}1) + b(i)$*p[i].

d.2 Trước hết biến đổi để lật s[i] từ '1' thành '0', sau đó biến đổi A→B, cuối cùng lật lại s[i] từ '0' thành '1'

- Số bước cần thiết: $b(i) = b(i\text{-}1)+2$;
- Chi phí: $c(i) = v(s,i\text{-}1) + c(i\text{-}1) + v(e,i)$.

Khi tính toán ta cũng tranh thủ tính dần các giá trị $vs = v(s,i\text{-}1)$ và $ve = v(e,i)$.

2.1.3. Phụ thuộc vào số biến của hàm

Khi xây dựng công thức truy hồi dựa theo yêu cầu của bài toán ta cần xác định có bao nhiêu đối tượng cần quan tâm, từ đó xác định công thức truy hồi cần lập có thể phụ thuộc vào một biến, hai biến hoặc ba biến. Hàm thường trả về giá trị theo yêu cầu của bài toán.

2.1.3.1. Công thức truy hồi có một biến

Với các bài toán mà ta chỉ quan tâm đến một đối tượng nào đó. Khi lập công thức truy hồi chỉ phụ thuộc vào một biến. Ta xét bài toán sau:

Bài Tìm các đường ngắn nhất

Cho một đồ thị có hướng gồm n đỉnh mã số từ 1..n với các cung (u, v) có hướng đi từ đỉnh u đến đỉnh v và có chiều dài thể hiện đường đi nối từ đỉnh u đến

đỉnh v. Viết chương trình tìm mọi đường đi ngắn nhất từ một đỉnh s cho trước tới các đỉnh còn lại của đồ thị.

Thuật toán: Quy hoạch động

Đối tượng mà ta quan tâm ở đây là với đỉnh *i* nào đó ($i \neq s$) thì đường đi ngắn nhất từ đỉnh *s* đến đỉnh *i* là bao nhiêu.

Thuật toán quy hoạch động được trình bày dưới đây mang tên Dijkstra, một nhà tin học lỗi lạc người Hà Lan. Bản chất của thuật toán là sửa đỉnh, chính xác ra là sửa trọng số của mỗi đỉnh.

Theo sơ đồ giải các bài toán quy hoạch động trước hết ta xây dựng hệ thức cho bài toán.

Gọi $p(i)$ là độ dài đường ngắn nhất từ đỉnh *s* đến đỉnh *i*, $1 \leq i \leq n$. Ta thấy, hàm $p(i)$ phải thoả các tính chất sau:

a) $p(s) = 0$: đường ngắn nhất từ đỉnh xuất phát *s* đến chính đỉnh đó có chiều dài 0.

b) Với $i \neq s$, muốn đến được đỉnh *i* ta phải đến được một trong các đỉnh sát trước đỉnh *i*. Nếu *j* là một đỉnh sát trước đỉnh *i*, theo điều kiện của đầu bài ta phải có

a[j,i] > 0

trong đó $a[j, i]$ chính là chiều dài cung ($j \rightarrow i$).

Trong số các đỉnh *j* sát trước đỉnh *i* ta cần chọn đỉnh nào**?**

Kí hiệu path(*x, y*) là đường đi ngắn nhất qua các đỉnh, xuất phát từ đỉnh từ *x* và kết thúc tại đỉnh $y \neq x$. Khi đó đường từ *s* đến *i* sẽ được chia làm hai đoạn, đường từ *s* đến *j* và cung ($j \rightarrow i$):

path(s,i) = path(s,j)+ path(j,i)

trong đó path(*j, i*) chỉ gồm một cung:

path(j,i) = (j → i)

Do $p(i)$ và $p(j)$ phải là ngắn nhất, tức là phải đạt các trị min, ta suy ra điều kiện để chọn đỉnh *j* sát trước đỉnh *i* là tổng chiều dài đường từ *s* đến *j* và chiều dài cung ($j \rightarrow i$) là ngắn nhất. Ta thu được hệ thức sau:

p(i) = min {p(j)+a[j,i] | a[j,i] > 0, j = 1..n }

Để ý rằng điều kiện $a[j, i] > 0$ cho biết *j* là đỉnh sát trước đỉnh *i*.

Điều tài tình là Dijkstra đã cung cấp thuật toán tính đồng thời mọi đường đi ngắn nhất từ đỉnh s đến các đỉnh còn lại của đồ thị. Thuật toán đó như sau.

Thuật toán thực hiện *n* lần lặp, mỗi lần lặp ta chọn và xử lí một đỉnh của đồ thị. Tại lần lặp thứ *k* ta khảo sát phần của đồ thị gồm *k* đỉnh với các cung liên quan đến *k* đỉnh được chọn trong phần đồ thị đó. Ta gọi phần này là đồ thị con thu được tại bước xử lý thứ *k* của đồ thị ban đầu và kí hiệu là G(*k*). Với đồ thị này ta hoàn tất bài giải tìm mọi đường đi ngắn nhất từ đỉnh xuất phát *s* đến mọi đỉnh còn lại của

G(*k*). Chiều dài thu được ta gán cho mỗi đỉnh i như một trọng số *p*[*i*]. Ngoài ra, để chuẩn bị cho bước tiếp theo ta đánh giá lại trọng số cho mọi đỉnh kề sau của các đỉnh trong G(*k*).

Khởi trị: Gán trọng số *p*[*i*] = ∞ cho mọi đỉnh, trừ đỉnh xuất phát *s*, gán trị *p*[*s*] = 0.

Ý nghĩa của thao tác này là khi mới đứng ở đỉnh xuất phát *s* của đồ thị con G(0), ta coi như chưa thăm mảnh nào của đồ thị nên ta chưa có thông tin về đường đi từ *s* đến các đỉnh còn lại của đồ thị ban đầu. Nói cách khác ta coi như chưa có đường đi từ *s* đến các đỉnh khác *s* và do đó, độ dài đường đi từ *s* đến các đỉnh đó là ∞.

Giá trị ∞ được chọn trong chương trình là**:** MAXWORD = 65535

Tại bước lặp thứ *k* ta thực hiện các thao tác sau:

- Trong số các đỉnh chưa xử lí, tìm đỉnh *i* có trọng số min.
- Với mỗi đỉnh *j* chưa xử lí và kề sau với đỉnh *i*, ta chỉnh lại trọng số *p*[*j*] của đỉnh đó theo tiêu chuẩn sau:

Nếu *p*[*i*] + *a*[*i, j*] < *p*[*j*] thì gán cho *p*[*j*] giá trị mới:

p[j]=p[i]+a[i,j]

Ý nghĩa của thao tác này là: nếu độ dài đường đi path(*s, j*) trong đồ thị con G(*k* - 1) không qua đỉnh *i* mà lớn hơn độ dài đường đi mới path(*s, j*) có qua đỉnh *i* thì cập nhật lại theo đường mới đó.

- Sau khi cập nhật ta cần lưu lại vết cập nhật đó bằng lệnh gán before[*i*] = *j* với ý nghĩa là, đường ngắn nhất từ đỉnh *s* tới đỉnh *j* cần đi qua đỉnh *i*.
- Đánh dấu đỉnh *i* là đã xử lí.

Như vậy, tại mỗi bước lặp ta chỉ xử lí đúng một đỉnh *i* có trọng số min và đánh dấu duy nhất đỉnh đó.

```
(*------ Thuat toan Dijkstra -----*)
Procedure Dijkstra;
Var i,k,j: byte;
Begin
Init;
for k :=  1 to n do
begin
      i :=  Min; {tim dinh i co trong so p[i] -> min }
      d[i] :=  1; {danh dau dinh i la da xu li }
      for j :=  1 to n do
            if d[j] = 0 then {dinh chua tham }
```

```
    if a[i,j] > 0 then {co duong di i -> j }
                    if p[i] + a[i,j] < p[j] then
                begin {sua dinh }
                p[j] :=  p[i] + a[i,j];
                        before[j] :=  i;
                end;
end;
End;
```

Độ phức tạp: O(n^2)

2.1.3.2. Công thức truy hồi có hai biến

Với các bài toán mà ta phải quan tâm đến hai đối tượng trong bài. Khi lập công thức truy hồi loại này thường phụ thuộc vào hai biến. Ta xét bài toán sau:

Bài toán cái túi

Trong siêu thị có n gói hàng ($n \leq 100$), gói hàng thứ i có trọng lượng là $W[i] \leq 100$ và trị giá $V[i] \leq 100$. Một tên trộm đột nhập vào siêu thị, tên trộm mang theo một cái túi có thể mang được tối đa trọng lượng M ($M \leq 100$). Hỏi tên trộm sẽ lấy đi những gói hàng nào để được tổng giá trị lớn nhất.

Thuật toán: Quy hoạch động

Đối tượng mà ta quan tâm là n gói hàng và trọng lượng tối đa là W. Với n gói hàng và trọng lượng tối đa là W thì giá trị lớn nhất trong các gói hàng tên trộm lấy trong túi là bao nhiêu. Vì vậy ta gọi hàm $F(i, j)$ trả về giá trị lớn nhất có thể có bằng cách chọn trong các gói $\{1, 2,\ldots, i\}$ với giới hạn trọng lượng j. Thì giá trị lớn nhất khi được chọn trong số n gói với giới hạn trọng lượng M chính là $F(n, M)$.

Công thức truy hồi tính F(i, j).

Với giới hạn trọng lượng j, việc chọn tối ưu trong số các gói $\{1, 2,\ldots, i - 1, i\}$ để có giá trị lớn nhất sẽ có hai khả năng:

- Nếu không chọn gói thứ i thì $F(i, j)$ là giá trị lớn nhất có thể bằng cách chọn trong số các gói $\{1, 2,\ldots, i - 1\}$ với giới hạn trọng lượng là j. Tức là $F(i, j) = F(i - 1, j)$
- Nếu có chọn gói thứ i (tất nhiên chỉ xét tới trường hợp này khi mà $W[i] \leq j$) thì $F(i, j)$ bằng giá trị gói thứ i là $V[i]$ cộng với giá trị lớn nhất có thể có được bằng cách chọn trong số các gói $\{1, 2,\ldots, i - 1\}$ với giới hạn trọng lượng $j - W[i]$. Tức là về mặt giá trị thu được: $F(i, j) = V[i] + F(i - 1, j - W[i])$

Vì theo cách xây dựng $F(i, j)$ là giá trị lớn nhất có thể, nên $F(i, j)$ sẽ là Max trong hai giá trị thu được ở trên .

Đoạn chương trình viết bằng ngôn ngữ Pascal

```
Procedure   make_table;
Var  i,j:integer;
Begin
   for j:=0 to m do f[0,j]:=0;
   for i:=1 to n do
    for j:=0 to m do
      begin
        f[i,j]:=f[i-1,j];
        if (j>=w[i]) and (f[i,j]< f[i-1,j-w[i]]+v[i])        then
         f[i,j]:=f[i-1,j-w[i]]+v[i];
      end;
End;
```

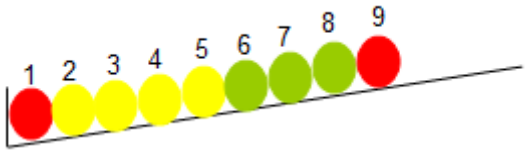
2.1.3.3. Công thức truy hồi có ba biến

Với các bài toán mà ta phải quan tâm đến ba đối tượng. Khi lập công thức truy hồi loại này thường phụ thuộc vào ba biến. Ta xét bài toán sau:

Trên một máng dốc người ta đặt *N* viên bi. Mỗi viên bi có một trong ba màu: đỏ (Đ), vàng (V), xanh (X). Người ta có thể bốc các viên bi cùng màu đứng sát nhau. Nếu lấy *K* viên bi như vậy, ta sẽ nhận được điểm là K^2. Số điểm ban đầu là 0. Sau khi bốc các viên bi còn lại tự động sát vào nhau. Ta có thể bốc nhiều lần cho đến khi máng không còn viên bi nào. Tổng số điểm nhận được phụ thuộc vào trình tự chọn.

Yêu cầu: Hãy xác định cách chọn sao cho tổng số điểm nhận được là lớn nhất.

Ví dụ: Ta đặt 9 viên bi trên máng và đánh số thứ tự các viên bi từ 1 đến 9.



Một cách chọn:

Chọn các viên bi ‘V, V, V, V’ được tổng điểm 16, còn lại: ‘Đ, X, X, X, Đ’.

Chọn tiếp các viên bi ‘X, X, X’ tổng điểm có được là 25, còn lại : ‘Đ, Đ’.

Chọn các viên còn lại ‘Đ, Đ’ tổng điểm có được là **29,** hết bi.

Một cách chọn khác:

Chọn viên bi đỏ đầu tiên ‘Đ’ tổng điểm 1, còn lại: ‘V, V, V, V, X, X, X, Đ’.

Chọn tiếp các viên bi ‘X, X, X’ tổng điểm là 10, còn lại: ‘V, V, V, V, Đ’.

Chọn tiếp các viên bi ‘V, V, V, V’ tổng điểm có được là 26, còn lại : ‘Đ’

Chọn tiếp viên bi ‘Đ’ tổng điểm có được là **27**, hết bi.

Thuật toán: Quy hoạch động

Ta có một nhận xét như sau: Với dãy các viên bi từ viên bi thứ *u* đến viên bi thứ *v*, ta luôn có cách chọn tối ưu sao cho trong lần chọn cuối cùng ta sẽ lấy một dãy các viên bi cùng màu trong đó có viên bi thứ *u*. Đối tượng mà ta quan tâm là dãy các viên bi từ viên bi thứ nhất đến viên bi thứ *N* sau khi chọn cuối cùng không thừa viên bi nào và thu được tổng điểm là lớn nhất. Với nhận xét đó, ta có hàm quy hoạch động với ý nghĩa như sau:

- *D*(*i, j, k*) là tổng điểm tối ưu thu được khi xét các viên bi từ viên bi thứ *i* đến viên bi thứ *j* và còn thừa lại đúng *k* viên bi cùng màu với viên bi thứ *i*.
- *Best*(*i, j*) là tổng điểm tối ưu thu được khi xét các viên bi từ viên bi thứ *i* đến viên bi thứ *j* và không thừa lại viên bi nào.

D(*i, j, k*) = max(*D*(*i, u*-1, *k*-1) + *Best*(*u*+1, *j*)) với *u* = (*i* . . *j*)

Best(*i, j*) = max(*D*(*i, j, k*) + *k***k*) với *k* = (1 .. *N*)

Khởi tạo:

D(*i, i*, 1) := 0; *Best*(*i, i*):= 1;

Đoạn chương trình viết bằng ngôn ngữ Pascal

```
Procedure  process;
Var   color,len,i,j,k,u:integer;
Begin
   for i:=1 to n do
    for j:=1 to n do
      for k:=1 to n do d[i]^[j,k]:=low(longint);
   for i:=1 to n do
    begin
      d[i]^[i,1]:=0;  best[i,i]:=1;
    end;
   for len:=2 to n do
    for i:=1 to n - len +1 do
      begin
       color:=a[i];
       j:=i+len-1;
       d[i]^[j,1]:=best[i+1,j];
       best[i,j]:=d[i]^[j,1]+1;
       for k:=2 to dem[j,color]-dem[i-1,color] do
        begin
```

```
        for u:=i to j do
        if a[u]=a[i] then
         if dem[u-1,color]-dem[i-1,color]>=k-1 then
        if d[i]^[j,k]<d[i]^[u-1,k-1]+best[u+1,j] then d[i]^[j,k]:=d[i]^[u-1,k-1]+best[u+1,j];
          if d[i]^[j,k]+k*k>best[i,j] then best[i,j]:=d[i]^[j,k]+k*k;
       end;
      end;
   write(best[1,n]);
End;
```

Lưu ý: Mảng dem[*j*,color] trả về số lượng viên bi màu color trong dãy từ 1 đến *j*. Ta có thể gán viên bi màu đỏ số 1, màu vàng số 2, màu xanh số 3.

2.2. Tổ chức dữ liệu

Phương pháp quy hoạch động lưu giữ kết quả đã tìm kiếm được vào một bảng làm giả thiết cho việc tìm kiếm những kết quả của trường hợp sau. Bảng sẽ được làm đầy giá trị bởi kết quả của những trường hợp trước đã được giải. Như vậy phương pháp quy hoạch động chính là điền vào bảng những kết quả lời giải của bài toán con đã giải với mục đích tránh khỏi việc tính toán thừa.

Khi lập trình ta thường dùng mảng hai chiều để lưu trữ các giá trị của bảng phương án. Để thu hẹp số chiều quy hoạch động ta sử dụng kĩ thuật chồng mảng, đưa thuật toán quy hoạch động hai chiều về quy hoạch động một chiều.

Bảng phương án (hay hàm quy hoạch động, hàm mục tiêu) được tính theo cột (hoặc hàng) mà cột sau (hàng sau) xác định chỉ thông qua một cột liền trước (hàng trước).

Ví dụ: Bài Chia thưởng

Ta có phương án đầu tiên như sau:

```
(*-----PHUONG AN 1: de quy.---*)
Function C(i,j: integer):longint;
Begin
if j = 0 then C:= 0
else {j > 0 }
          if i = 0 then {i = 0; j > 0 }
               C:= 1
               else {i,j > 0 }
                if i < j then {0 < i < j }
```

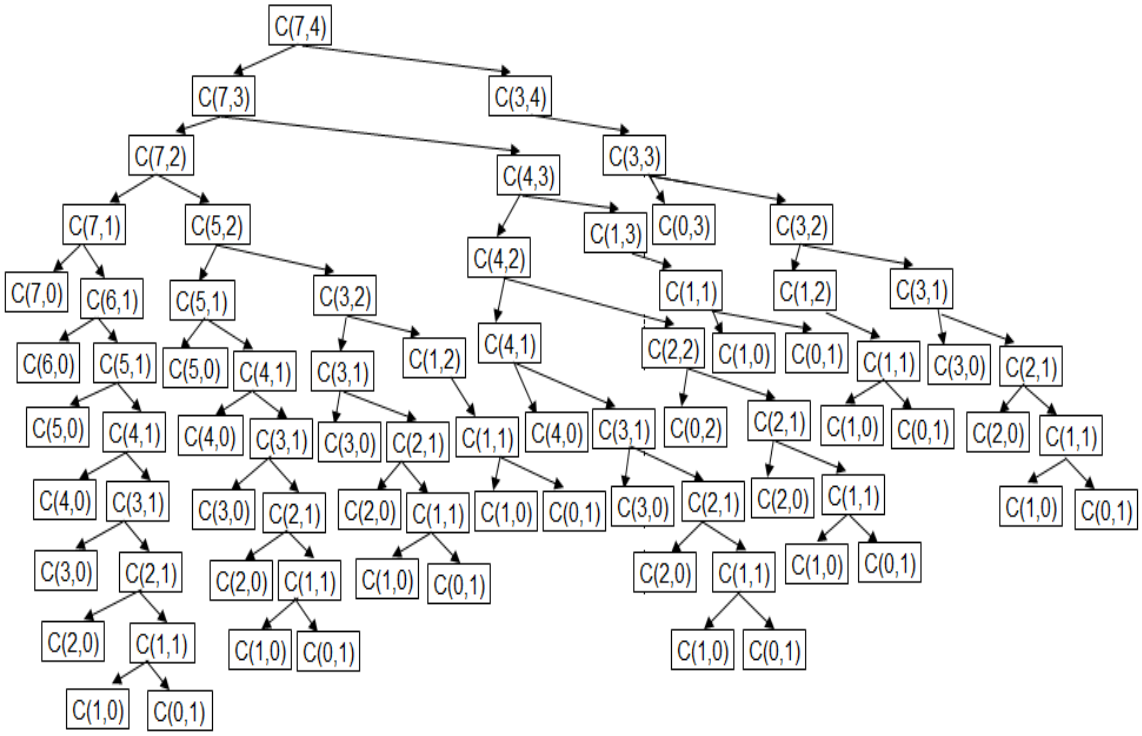
```
        C:= C(i,i)
      else
    C:= C(i,j-1)+C(i-j,j);
End;
```

Phương án này chạy chậm vì phát sinh ra quá nhiều lần gọi hàm trùng lặp. Bảng dưới đây liệt kê số lần gọi hàm cục bộ khi giải bài toán chia thưởng với $m = 7$ và $n = 4$.

Bảng 2.3. Số lần gọi hàm cục bộ khi gọi C(7, 4)

	❶	❷	❸	❹
⓪	0	9	1	1	0
①	9	9	2	1	0
②	6	6	1	0	0
③	5	5	2	1	1
④	3	3	1	1	0
⑤	2	2	1	0	0
⑥	1	1	0	0	0
⑦	1	1	1	1	1

Thí dụ, hàm *C*(1,1) sẽ được gọi 9 lần,… Tổng số lần gọi hàm Chia là 79. 79 lần gọi hàm để sinh ra kết quả 11 là quá tốn kém. Dưới đây là cây biểu diễn số lần gọi hàm cục bộ khi gọi *C*(7,4).



Hình 2.2. Cây biểu diễn số lần gọi hàm cục bộ khi gọi hàm C(7, 4)

Làm tốt lần 1: Phương án 1 khá dễ triển khai nhưng chương trình sẽ chạy rất lâu. Diễn tả đệ quy thường trong sáng, ngắn gọn, nhưng khi thực hiện sẽ sinh ra hiện tượng gọi lặp lại những hàm đệ quy. Để tránh những lần gọi lặp như vậy chúng ta tính sẵn các giá trị của hàm theo các trị của đầu vào khác nhau và điền vào một mảng hai chiều cc.

Mảng cc được mô tả như sau:

Const max = 100;

var cc: array[0 .. max, 0 .. max] of int64;

Ta quy ước *cc*[*i, j*] chứa số cách chia i phần thưởng cho *j* học sinh.

Theo phân tích của phương án 1, ta có:

- *cc*[0, 0] = 1; *cc*[*i*, 0] = 0, với *i*:=1..*m.*
- *cc*[*i, j*] = *cc*[*i, i*], nếu *i* < *j*
- *cc*[*i, j*] = *cc*[*i, j*-1]+*cc*[*i*-*j, j*], nếu *i* ≥ *j*.

Từ đó ta suy ra quy trình điền trị vào bảng cc như sau:

- cc[0,0]:= 1;
- với i := 1..m: cc[i,0] := 0;
- cc[0,j]:=1 với j := 1 .. n
- Điền bảng*: Lần lượt điền theo từng cột* j*:= 1*..n. *Tại mỗi cột* j *ta đặt:*
- với i := 1..j-1: cc[i,j] := cc[i,i];

- với i := j..m: cc[i,j] := cc[i,j-1]+cc[i-j,j];

Nhận kết quả*: Sau khi điền bảng, giá trị* cc*[*m, n*] chính là kết quả cần tìm.*

```
(*------------  PHUONG AN 2: Dùng mảng hai chiều  cc -------------*)
Procedure   make_table2;
Var i,j:integer;
Begin
  cc[0,0]:=1;
  for i:=1 to m do cc[i,0]:=0;
  for j:=1 to n do cc[0,j]:=1;
  for j:=1 to n do
   begin
    for i:=1 to j-1 do cc[i,j]:=cc[i,i];
    for i:=j to m do cc[i,j]:=cc[i,j-1]+cc[i-j,j];
   end;
  writeln(cc[m,n]);
End;
```

Làm tốt lần 2: Dùng mảng hai chiều chúng ta chỉ có thể tính toán được với dữ liệu nhỏ. Bước cải tiến sau đây khá quan trọng: chúng ta dùng mảng một chiều. Quan sát kĩ quy trình gán trị cho mảng hai chiều theo từng cột chúng ta dễ phát hiện ra rằng cột thứ *j* có thể được tính toán từ cột thứ *j* - 1. Nếu gọi *c* là mảng một chiều sẽ dùng, ta cho số học sinh tăng dần bằng cách lần lượt tính *j* bước, với *j* := 1..*n*. Tại bước thứ *j*, *c*[*i*] chính là số cách chia *i* phần thưởng cho *j* học sinh. Như vậy, tại bước thứ *j* ta có:

- *c*[*i*] tại bước *j* = *c*[*i*] tại bước (*j* - 1), nếu *i* < *j*. Từ đây suy ra đoạn *c*[0..(*j* - 1)] được bảo lưu.
- *c*[*i*] tại bước *j* = *c*[*i*] tại bước (*j* - 1) + *c*[*i* - *j*] tại bước *j*, nếu *i* ≥ *j*.

Biểu thức thứ hai cho biết khi cập nhật mảng *c* từ bước thứ *j* - 1 qua bước thứ *j* ta phải tính từ trên xuống, nghĩa là tính dần theo chiều tăng của *i* := *j..m*.

Mảng c được khởi trị ở bước *j* = 0 như sau:

- c[0] = 1; c[i] = 0, với i := 1..m.

Với ý nghĩa là, nếu có 0 học sinh thì chia 0 phần thưởng cho 0 học sinh sẽ được quy định là 1. Nếu số phần thưởng *m* khác 0 thì chia *m* phần thưởng cho 0 học sinh sẽ được 0 phương án.

Ta có phương án ba, dùng một mảng một chiều c như sau:

```
(* ------ PHUONG AN 3: dùng mảng một chiều c  ---------------*)
```

```
procedure   make_table1;
var  i,j:integer;
begin
   fillchar(c,sizeof(c),0); c[0]:=1;
   for j:=1 to n do
    for i:=j to m do c[i]:= c[i]+ c[i-j];
   writeln(c[m]);
end;
```
[7]

Với m = 80, n = 60 thì phương án dùng mảng hai chiều và một chiều chạy mất 0 mili giây, còn phương án đệ quy chạy mất 2590 mili giây.

KẾT LUẬN CHƯƠNG 2

Quy hoạch động là một phương pháp rất hay và mạnh trong tin học. Nhưng để giải được các bài toán bằng phương pháp quy hoạch động thật chẳng dễ dàng chút nào. Chủ yếu học sinh hiện nay sử dụng quy hoạch động theo kiểu làm từng bài cho nhớ mẫu và áp dụng vào những bài có dạng tương tự. Các kỹ thuật được trình bày trên đây là những kinh nghiệm được rút ra khi giải các bài toán quy hoạch động. Với từng bài toán phải có sự áp dụng linh hoạt.

Vấn đề khó nhất của việc giải một bài toán quy hoạch động là tìm *dấu hiệu nhận biết quy hoạch động* và tìm *quy luật quy hoạch dữ liệu* của bài toán đó. Vì vậy khi giải một bài toán quy hoạch động phải đặt từng bài toán dưới nhiều góc nhìn khác nhau và đánh giá rồi lựa chọn các phương án.

Chương 3
THUẬT TOÁN QUY HOẠCH ĐỘNG VÀ LÝ THUYẾT TRÒ CHƠI

3.1. Bài toán trò chơi

Trong công việc thường ngày để đạt được mục tiêu này hay mục tiêu khác, con người thường thường phải đưa ra sự lựa chọn tối ưu. Bài toán tối ưu gặp cả trong lý thuyết lẫn trong ứng dụng ở tất cả các ngành khoa học, từ y tế đến vật lý, từ toán học đến sinh học, từ quân sự đến kinh tế. Vấn đề lựa chọn giải pháp tối ưu được hình thức hóa và việc nghiên cứu mô hình toán học này dẫn đến sự ra đời của một số lĩnh vực nghiên cứu lý thuyết, trong đó tổng quát hơn cả là lý thuyết Nghiên cứu hỗ trợ quyết định.

Trong số các điều kiện tác động lên chiến lược lựa chọn quyết định điều kiện xung đột đóng một vai trò đặc biệt quan trọng. Nó quan trọng vì hai lý do:

- Xung đột là động lực phát triển của cuộc sống và xã hội.
- Đặc thù của xung đột là yếu tố chính tác động lên quyết định phải lựa chọn.

Trong điều kiện có xung đột, quyết định phải dựa trên cơ sở:

- Lợi ích của mình và lợi ích của các đối tác mà thông thường các lợi ích này mâu thuẫn lẫn nhau.
- Khi ra quyết định, ta phải lưu ý đến quyết định của đối tác, điều mà thông thường ta không biết trước.

Chuyên ngành nghiên cứu về việc đưa ra quyết định trong trường hợp có xung đột là Lý thuyết trò chơi. Xung đột trong trò chơi thường được gọi là đối kháng [2].

Chúng ta xét loại trò chơi với các giả thiết sau đây:

- Trò chơi gồm hai đấu thủ là A và B, luân phiên nhau, mỗi người đi một nước. Ta luôn giả thiết đấu thủ đi trước là A.
- Hai đấu thủ đều chơi rất giỏi, nghĩa là có khả năng tính trước mọi nước đi.
- Đấu thủ nào đến lượt mình không thể đi được nữa thì chịu thua và ván chơi kết thúc.
- Không có thế hòa, sau hữu hạn nước đi sẽ xác định được ai thắng, ai thua.

Giả thiết chơi giỏi nhằm tránh các trường hợp "*ăn may*", tức là các trường hợp do đối phương hớ hênh mà đi lạc nước. Điều này tương đương với giả thiết cả hai đấu thủ đều có thể tính trước mọi nước đi (với loại trò chơi hữu hạn) hoặc cả hai đấu thủ đều biết cách đi tốt nhất.

Các bài toán tin liên quan đến loại trò chơi này thường là:

- Lập trình để xác định với một thế cờ cho trước thì người đi trước (đấu thủ A) sẽ thắng hay thua.
- Lập trình để máy tính chơi với người. Dĩ nhiên chương trình bạn lập ra là dành cho máy tính.

- Lập trình để hai máy tính chơi với nhau [7].

Với loại trò chơi này có một *heuristic* mang tính chỉ đạo sau đây:

*c hết cần xác định được một tính chất **T** thỏa các điều kiện sau đây:*

*a) Thế thua cuối cùng thỏa **T**,*

*b) Mọi nước đi luôn luôn biến **T** thành **V** = not **T**,*

*c) Tồn tại một nước đi để biến **V** thành **T**.*

*chất **T** được gọi là bất biến thua của trò chơi.*

3.2. Lý thuyết trò chơi

Mỗi trò chơi bao giờ cũng liên quan tới hai không gian: không gian trạng thái và không gian điều khiển. Hàm mục tiêu thường được xác định trên tập trạng thái. Tập trạng thái của trò chơi có thể là hữu hạn hoặc vô hạn nhưng tập điều khiển luôn luôn hữu hạn và thông thường là không lớn. Xét trò chơi đối kháng giữa 2 người.

Ví dụ, xét bài toán trò chơi lật xúc xắc.

Cho một con xúc xắc truyền thống, trên mỗi mặt của xúc xắc có một số chấm trong phạm vi từ 1 đến 6 xác định số điểm của mặt, không có 2 mặt nào có cùng số điểm và tổng điểm của 2 mặt đối luôn luôn bằng 7. Con xúc xắc được tung lên bàn và tổng S của trò chơi ban đầu nhận giá trị bằng số điểm mặt trên của xúc xắc. Hai người lần lượt đi. Khi đến lượt mình đi, người chơi lật một lần con xúc xắc qua cạnh của nó và cộng số điểm ở mặt trên mới vào ***S***. Ai đến lượt mình đi làm tổng S lớn hơn S_{max} sẽ thua.

Cuộc chơi đang diễn ra sôi nổi thì người đến lượt đi có điện thoại và lúng túng xin lỗi phải đi làm một việc gấp. Vì lý do tế nhị, không ai hỏi đó là việc gì, nhưng bạn được chỉ định thay thế. Với tổng ***S*** và mặt trên ***v*** hiện có, hãy xác định cuối cùng bạn có thể thắng được hay không và nếu có thì chỉ ra các cách lật để thắng.

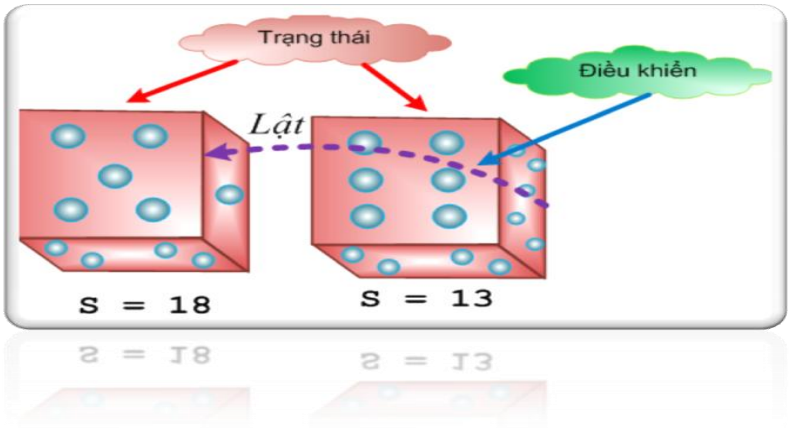
Dữ liệu: Vào từ file văn bản DICE.INP gồm một dòng chứa 3 số nguyên ***v, S*** và S_{max} ($1 \le v \le 6$, $1 \le S \le S_{max} \le 10^5$). Các số ghi cách nhau một dấu cách.

Kết quả: Đưa ra file văn bản DICE.OUT trên một dòng số nguyên ***m*** - số cách đi thắng và nếu ***m***> 0 thì sau đó là ***m*** số nguyên - các mặt cần đưa thành mặt trên, đưa ra theo thứ tự tăng dần. Các số ghi cách nhau một dấu cách. ***m*** = 0 ứng với trường hợp không có cách thắng.

Ví dụ:	***ICE.INP***	***ICE.OUT***
	3 20	***2 3 4***

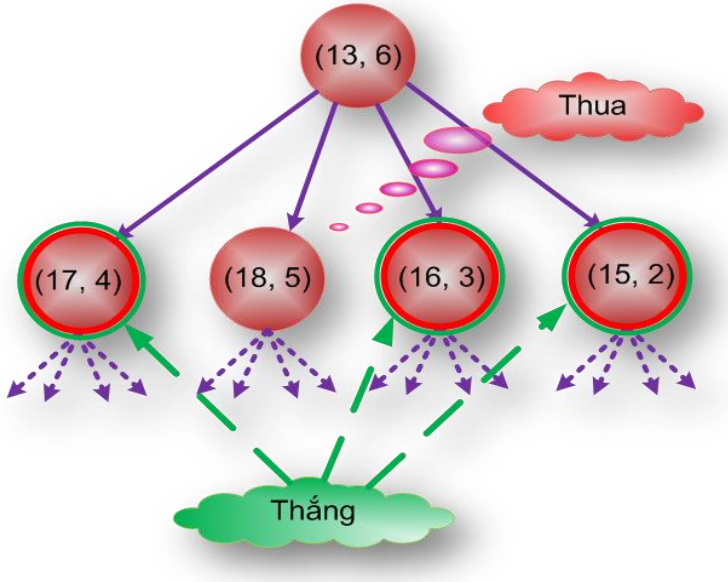
Mô hình:

Ở bài này, trạng thái của trò chơi là tổng S và mặt trên của con xúc xắc, còn điều khiển là cách lật. Có 4 cách lật: đưa mặt trước lên trên, đưa mặt phải lên trên, đưa mặt trái lên trên hoặc đưa mặt sau lên trên.



Hình 3.1. Không gian trạng thái và không gian điều khiển của bài toán lật xúc xắc

Ta có thể biểu diễn các nước đi của trò chơi dưới dạng một đồ thị có hướng, trong đó các nút là trạng thái. Điều khiển được thể hiện dưới dạng cung có hướng, nối các trạng thái với nhau.



Hình 3.2. Biểu diễn các nước đi của trò chơi dưới dạng một đồ thị có hướng

3.2.1. Trò chơi trên đồ thị

Trên nguyên tắc, mọi trò chơi đều có thể biểu diễn dưới dạng đồ thị có hướng. Tuy vậy, không phải lúc nào đồ thị có hướng này cũng dễ xây dựng, lưu trữ và thuận tiện trong việc hỗ trợ tìm chiến lược điều khiển.

Những trò chơi mà mô hình đồ thị cho phép chúng ta dễ dàng tìm ra chiến lược điều khiển để thắng (nếu có) được xếp vào lớp trò chơi trên đồ thị.

Trò chơi trên đồ thị có nội dung như sau:

Cho đồ thị có hướng G = (V, E), trong đó V - tập đỉnh của đồ thị, mỗi đỉnh tương ứng với một trạng thái của trò chơi, E - tập cạnh, mỗi cạnh tương ứng với một nước đi.

Giả thiết cả hai người chơi đều biết cách đi tối ưu, tức là nếu có khả năng thắng được thì họ sẽ không bỏ qua.

Ký hiệu W - tập con các đỉnh từ đó có cách đi để thắng, L - tập con các đỉnh từ đó không có cách đi để thắng và D - tập con các đỉnh từ đó trò chơi sẽ kết thúc với kết quả hòa. Khi đó $V = W \cup L \cup D$.

Mọi chiến lược điều khiển trò chơi là một phép ánh xạ $f: V \rightarrow E$ xác định nước đi cần thực hiện từ đỉnh đang đứng.

Số cung xuất phát từ một đỉnh được gọi là bậc của đỉnh đó. Các nút tương ứng với trạng thái kết thúc trò chơi có bậc bằng 0.

Giải thuật phân loại đỉnh đồ thị

3.2.1.1. Trường hợp đồ thị không có chu trình

Bổ đề 1. *Nếu G là đồ thị không chứa chu trình và mọi đỉnh bậc 0 đều thuộc tập* $W \cup L$*, khi đó mỗi đỉnh của G đều có thể phân loại thành đỉnh thắng hoặc thua (không có trường hợp hòa).*

Chứng minh: Sắp xếp các đỉnh của đồ thị theo thứ tự từ điển tăng dần khoảng cách tới đỉnh gốc. Gọi W_0 là tập các đỉnh bậc 0 thuộc W, L_0 là tập các đỉnh bậc 0 thuộc L. Như vậy, nếu u là một đỉnh bậc 0, nó phải thỏa mãn một trong hai điều kiện:

- $u \in W_0 \Rightarrow u \in W$,
- $u \in L_0 \Rightarrow u \in L$.

Do phải có ít nhất một cung dẫn tới đỉnh u nên:

- Nếu $u \in W \Leftrightarrow \exists v$: $uv \in E$ và $v \in L$,
- Nếu $u \in L \Leftrightarrow \forall v$ nếu $uv \in E$ thì $v \in W$.

Như vậy giải thuật có độ phức tạp là O(E+V).

3.2.1.2. Trường hợp đồ thị có chu trình

Xét trường hợp đồ thị G chứa chu trình. Một nước đi của trò chơi bao gồm hai lần đi: lần đi của một người và lần đi tiếp theo của đối phương.

Gọi W_k là tập các nút mà xuất phát từ đó người chơi sẽ thắng được sau khi thực hiện không quá k lần đi của mình, L_k là tập các nút mà xuất phát từ đó người chơi sẽ chắc chắn thua sau không quá k lần đi của mình.

Rõ ràng là $W_{k-1} \subset W_k$, $L_{k-1} \subset L_k$.

Nếu như một người nào đó sẽ thắng sau một số hữu hạn lần đi thì tồn tại các đẳng thức:

- $W = \bigcup_{k=0}^{\infty} W_k$,
- $L = \bigcup_{k=0}^{\infty} L_k$,
- Và tương ứng có $D = E \setminus (W \cup L)$.

Công thức truy hồi xác định W_i và L_i là như sau:

Xuất phát từ $i = 0$ (từ W_0 và L_0):

- $W_{i+1} = \{u | \exists v: uv \in E$ và $v \in L_i\}$,
- $L_{i+1} = \{u | \forall v: uv \in E \Rightarrow v \in Wi\}$.

3.2.1.3. Giải thuật xây dựng W và L độ phức tạp O(E)

Chuẩn bị mảng ***count*** với ***count[v]*** bằng số bậc của đỉnh v. Đưa các phần tử của W_0 và L_0 vào hàng đợi, sau đó bắt đầu lấy thông tin từ hàng đợi ra xử lý. Có thể xảy ra các trường hợp sau:

- Trường hợp 1: $u \in L$, khi đó:

```
for vu ∈ E do
 if ( v ∉ W) then
  begin
   W+= v; {Cho v vào tập W}
      push(v)
  end;
```

- Trường hợp 2: $u \in W$, khi đó:

```
for vu ∈ E do
  begin
count[v]--;
      if (count[v] ==0) then
            begin  L += v; {Cho v vào tập L}
                  push(v)
            end
  end;
```

3.2.2. Tổng trực tiếp. Hàm Sprague - Grundy

Có những trò chơi G có thể phân rã thành k trò chơi độc lập dạng đồ thị G1, G2,..., Gk. Mỗi trò chơi độc lập này được gọi là trò chơi con. Với mỗi trò chơi con

cho biết trạng thái thua. Ở trạng thái này và chỉ ở trạng thái này mới không tồn tại nước đi tiếp theo hợp lệ.

Người chơi sẽ chọn một trong số các trò chơi con và thực hiện nước đi của mình. Người chơi bị thua khi không còn trò chơi con nào có thể lựa chọn để thực hiện được nước đi. Trong trường hợp này G được gọi là trò chơi có tổng trực tiếp.

Trò chơi G trong trường hợp này có thể biểu diễn như trò chơi trên một đồ thị duy nhất:

- $G = < V_1 \times V_2 \times \ldots \times V_k, E>$,
- $<(v_1, v_2, \ldots, v_k), (u_1, u_2, \ldots, u_k)> \in E \Leftrightarrow \exists i: u_i v_i \in E_i$.

Định nghĩa 1. Hàm Sprague - Grundy (SG) là ánh xạ đỉnh sang tập nguyên không âm và thỏa mãn điều kiện:

SG(u) = min{m ≥ 0: m ≠ SG(v), uv ∈ E}.

SG(u) được gọi là số Sprague - Grundy.

Như vậy, SG(u) là số nguyên không âm nhỏ nhất chưa tìm thấy trong tập giá trị số Sprague - Grundy đối với u.

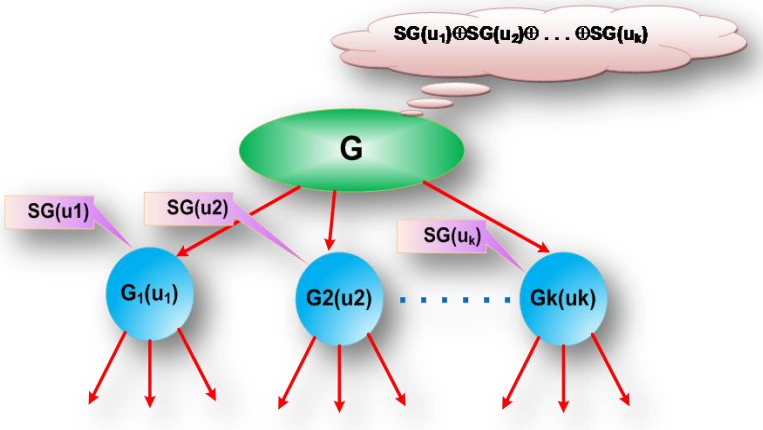
Số Sprague - Grundy có các tính chất sau:

- SG(u) = 0 nếu bậc của u bằng 0,
- SG(u) = 0 nếu u ∈ L,
- Nếu SG(u) = 0 và uv ∈ E, thì SG(v) ≠ 0,
- Nếu SG(u) ≠ 0 thì ∃v: uv ∈ E và SG(v) = 0.

Định lý Grundy

G = G1 × G2 × . . . × Gk và SGi là hàm Sprague - Grundy của Gi, i = 1 ÷ k, khi đó có:

SG(<u1, u2, . . ., uk>) = SG(u1) ⊕ SG(u2) ⊕ . . . ⊕ SG(uk)



Hình 3.3. Biểu diễn tính số Sprague - Grundy

Trong đó ⊕ là phép tính cộng từng bit không nhớ (phép XOR trong PASCAL, ^ trong C).

Chứng minh

Ta sẽ chứng minh định lý này bằng phương pháp quy nạp. Với k = 2 ta có G_1 G_2 và các hàm SG_1, SG_2. Ký hiệu SG(u1, u2) = SG1(u1) ⊕ SG2(u2). Ta sẽ chứng minh đây chính là hàm Sprague - Grundy cho G.

Nếu tồn tại cạnh (u1, u2) → (v1, u2) thì SG1(u1, u2) ≠ SG2(v1, u2). Giả thiết điều này không đúng, tức là có SG(u1, u2) = SG(v1, u2). Khi đó SG1(u1) ⊕ SG2(u2) = SG1(v1) ⊕ SG2(u2), suy ra SG1(u1) = SG1(v1) - mâu thuẫn với định nghĩa của hàm SG1!

Giả thiết x < SG1(u1) ⊕ SG2(u2). Ta sẽ chứng minh rằng tồn tại cạnh nối nút (u1, u2) tới nút (v1, v2) thỏa mãn điều kiện SG1(v1) ⊕ SG2(v2).

Gọi b là vị trí đầu tiên của bít trong x mà bít xb của x nhỏ hơn bít thứ b của SG1(u1) ⊕ SG2(u2). Khi đó tại vị trí thứ b các số SG1(u1) và SG2(u2) có giá trị bít khác nhau, còn xb = 0.

Không mất tính chất tổng quát, ta có thể coi bít thứ b của SG1(u1) bằng 1, còn bít thứ b của SG2(u2) bằng 0. Rõ ràng là SG2(u2) ⊕ x < SG1(u1). Như vậy trong đồ thị G1 có cạnh nối u1 → v1 và SG(v1) = SG1(u1) ⊕ x. Còn trong G ta có cạnh (u1, u2) → (v1, u2) và SG(<v1, u2>) = SG1(v1) ⊕ SG2(u2) = x.

Điều này nói lên rằng SG là hàm Sprague - Grundy của G. Đó là điều phải chứng minh. [2]

Bằng cách quy nạp, ta có thể dễ dàng chứng minh trong trường hợp tổng quát.

Ứng dụng: Trò chơi NIM.

Trò chơi NIM có xuất xứ từ Trung Hoa, dành cho hai đấu thủ A và B với các nước đi lần lượt đan nhau trên một đấu trường với N đống sỏi. Đống sỏi thứ **i** có $\mathbf{a_i}$ viên ($\mathbf{a_i}$>0, **i** = 1 ÷ **n**). Có 2 người chơi. Mỗi người, khi đến lượt mình phải bốc một số lượng sỏi tùy ý, lớn hơn 0 từ một đống tùy chọn. Ai đến lượt mình không còn cách bốc thì người đó thua.

Phân tích

Với mỗi đống có số lượng *x* SG(*x*) = *x*. Như vậy hàm Sprague - Grundy có dạng: SG(<a_1, a_2, . . ., a_n>) = $a_1 \oplus a_2 \oplus \ldots \oplus a_n$.

Đặt $x = a_1 \oplus a_2 \oplus \ldots \oplus a_n$. Ta chứng minh rằng *bất biến thua của trò chơi NIM là* $x = 0$, tức là nếu $x = 0$ thì đến lượt ai đi người đó sẽ thua.

Trước hết nhắc lại một số tính chất của phép toán ⊕ theo bit.

1) $a \oplus b = 1$ khi và chỉ khi a ≠ b.

2) $a \oplus 0 = a$

3) $a \oplus 1 = not\ a$

4) Tính giao hoán: $a \oplus b = b \oplus a$

5) *Tính kết hợp:* $(a \oplus b) \oplus c = a \oplus (b \oplus c)$

6) Tính lũy linh: $a \oplus a = 0$

7) $a \oplus b \oplus a = b$

8) Tính chất 7 có thể mở rộng như sau: Trong một biểu thức chỉ chứa phép xor ta có thể xóa đi chẵn lần các phần tử giống nhau, kết quả sẽ không thay đổi.

Để dễ nhớ ta gọi phép toán này là *so khác - so* xem hai đối tượng có *khác* nhau hay không. Nếu khác nhau là đúng (1) ngược lại là sai (0).

Bất biến $x = 0$ có ý nghĩa như sau: Nếu viết các giá trị a_i, $i = 1..N$ dưới dạng nhị phân vào một bảng thì số lượng số 1 trong mọi cột đều là số chẵn.

	Dạng nhị phân			
$a1 = 13$	1	1	0	1
$a2 = 14$	1	1	1	0
$a3 = 6$	0	1	1	0
$a4 = 7$	0	1	1	1
$a5 = 2$	0	0	1	0
$\oplus$ x = 0	0	0	0	0

Bảng bên cho ta $a_1 \oplus a_2 \oplus a_3 \oplus a_4 \oplus a_5 = 13 \oplus 14 \oplus 6 \oplus 7 \oplus 2 = 0$.

Nếu x là tổng xor của các a_i, $i = 1..N$, với mỗi $i = 1..N$ ta kí hiệu $K(i)$ là *tổng xor khuyết i của các* a_i với cách tính như sau: $K(i) = a_1 \oplus a_2 \oplus \ldots \oplus a_{i-1} \oplus a_{i+1} \oplus \ldots \oplus a_N$. Như vậy $K(i)$ là tổng xor của các a_j sau khi đã loại trừ phần tử a_i và x chính là tổng xor đủ của các a_i, $i = 1..N$. Do $a_i \oplus a_i = 0$ và $0 \oplus y = y$ với mọi y nên $K(i) = x \oplus a_i$. Để cho tiện, ta cũng kí hiệu $K(0)$ chính là tổng xor đủ của các a_i, $i = 1..N$. Với thí dụ đã cho ta tính được các tổng khuyết như sau:

$K(0) = a_1 \oplus a_2 \oplus a_3 \oplus a_4 \oplus a_5 = 13 \oplus 14 \oplus 6 \oplus 7 \oplus 2 = 0$.

$K(1) = a_2 \oplus a_3 \oplus a_4 \oplus a_5 = 14 \oplus 6 \oplus 7 \oplus 2 = 13$,

$K(2) = a_1 \oplus a_3 \oplus a_4 \oplus a_5 = 13 \oplus 6 \oplus 7 \oplus 2 = 14$,

$K(3) = a_1 \oplus a_2 \oplus a_4 \oplus a_5 = 13 \oplus 14 \oplus 7 \oplus 2 = 6$,

$K(4) = a_1 \oplus a_2 \oplus a_3 \oplus a_5 = 13 \oplus 14 \oplus 6 \oplus 2 = 7$,

$K(5) = a_1 \oplus a_2 \oplus a_3 \oplus a_4 = 13 \oplus 14 \oplus 6 \oplus 7 = 2$.

Ta phát hiện được qui luật lí thú sau đây:

Mệnh đề 1. Cho x là tổng xor của N số tự nhiên, a_i, $x = \mathbf{a}_1 \oplus \mathbf{a}_2 \oplus \ldots \oplus \mathbf{a}_N$. Khi đó $K(i) = x \oplus a_i$, $i = 1,2,\ldots,N$. Tức là muốn bỏ một số hạng trong tổng $\oplus$ ta chỉ việc $\oplus$ thêm tổng với chính số hạng đó. Nói riêng, khi $x = 0$ ta có $K(i) = a_i$, $i = 1,2,\ldots,N$.

Chứng minh

Gọi x là tổng xor đủ của các số đã cho, $x = a_1 \oplus a_2 \oplus \ldots \oplus a_N$. Vận dụng tính giao hoán và tính lũy đẳng ta có thể viết $x \oplus a_i = (a_1 \oplus a_2 \oplus \ldots \oplus a_{i-1} \oplus a_{i+1} \oplus a_N) \oplus (a_i \oplus a_i) = K(i) \oplus 0 = K(i)$, $i = 1,2,\ldots,N$, đpcm.

Ta chứng minh tiếp các mệnh đề sau:

Mệnh đề 2. Nếu x ≠ 0 thì có cách đi hợp lệ để biến đổi x = 0.

Chứng minh

Do $x \neq 0$ nên ta xét chữ số 1 trái nhất trong dạng biểu diễn nhị phân của $x = (x_m, x_{m-1},\ldots,x_0)$, $x_j = 1$, $x_i = 0$, $i > j$. Do x là tổng xor của các a_i, $i = 1..N$, nên tồn tại một $a_i = (b_m, b_{m-1},\ldots,b_0)$ để chữ số $b_j = 1$. Ta chọn đống a_i này (dòng có dấu *). Khi đó, ta tính được $K(i) = x \oplus a_i = (x_m \oplus b_m, x_{m-1} \oplus b_{m-1},\ldots,x_0 \oplus b_0) = (c_m, c_{m-1},\ldots,c_0)$ với $c_i = x_i \oplus b_i$, $0 \le i \le m$. Ta có nhận xét sau đây: * Tại các cột $i > j$: $c_i = b_i$, vì $c_i = x_i \oplus b_i = 0 \oplus b_i = b_i$,

	Dạng nhị phân			
	x3	x2	x1	X0
$a1 = 12$	1	1	0	0
$a2 = 14$	1	1	1	0
* $a3 = 6$	0	1	1	0
$a4 = 3$	0	0	1	1
$a5 = 2$	0	0	1	0
⊕ x = 5	0	1	0	1

* Tại cột j ta có: $c_j = 0$, vì $c_j = x_j \oplus b_j = 1 \oplus 1 = 0$.

Do $b_j = 1$, $c_j = 0$ và mọi vị trí $i > j$ đều có $c_i = b_i$ nên $a_i > K(i)$. Nếu ta thay dòng a_i bằng dòng $K(i)$ thì tổng xor y khi đó sẽ là:

$y = (x \oplus a_i) \oplus K(i) = K(i) \oplus K(i) = 0$.

Vậy, nếu ta bốc tại đống i số viên sỏi $v = a_i - K(i)$ thì số sỏi còn lại trong đống này sẽ là $K(i)$ và khi đó tổng xor sẽ bằng 0, đpcm.

Mệnh đề 3. Nếu x = 0 và còn đống sỏi khác 0 thì mọi cách đi hợp lệ đều dẫn đến x ≠ 0.

Chứng minh

Cách đi hợp lệ là cách đi làm giảm thực sự số sỏi của một đống a_i duy nhất nào đó, $1 \le i \le N$. Giả sử đống được chọn là $a_i = (b_m, b_{m-1},\ldots,b_0)$. Do a_i bị sửa nên chắc chắn có một bit nào đó bị đảo (từ 0 thành 1 hoặc từ 1 thành 0). Ta gọi bít bị sửa đó là b_j. Khi đó tổng số bít 1 trên cột j sẽ bị tăng hoặc giảm 1 đơn vị và do đó sẽ không còn là số chẵn. Từ đó suy ra rằng bit j trong x sẽ là 1, tức là $x \neq 0$ đpcm.

Phần lập luận chủ yếu trong mệnh đề 2 nhằm mục đích chỉ ra sự tồn tại của một tập a_i thỏa tính chất $a_i > x \oplus a_i$. Nếu tìm được tập a_i như vậy ta sẽ bốc $a_i - (x \oplus a_i)$ viên tại đống sỏi i .

3.2.3. Trò chơi trên ma trận

Cho ma trận **A** = (($\mathbf{a_{ij}}$)), **i** = 1 ÷ **m**, **j** = 1 ÷ **n**. Các trò chơi xác định trên ma trận **A** là trò chơi trên ma trận.

Ví dụ1: Với ma trận **A** nói trên, với tổng **S** ban đầu bằng 0 và chưa có hàng hay cột nào bị đánh dấu xóa hai người thực hiện **k** lần đoán số (**k**<*min*{**m**, **n**}). Mỗi lần hai người đồng thời chọn, người thứ nhất chọn một hàng **i** trong số các hàng chưa bị đánh dấu xóa, người thứ 2 chọn cột **j** trong số các cột chưa bị đánh dấu xóa. Phần tử nằm ở giao của hàng và cột này, tức là phần tử $\mathbf{a_{ij}}$ sẽ là điểm số của người thứ nhất và được cộng vào **S**, sau đó hàng **i** và cột **j** bị đánh dấu xóa. Hãy xác định tổng **S** lớn nhất có thể đạt được nếu cả hai cùng có sự lựa chọn tối ưu.

Ở bài toán này, người thứ nhất cố gắng cực đại hóa giá trị sẽ cộng vào **S**, còn người thứ 2 - tìm cách cực tiểu hóa giá trị này.

Phần lớn các trò chơi trên ma trận liên quan tới bài toán Minimax. Những bài toán này có nhiều ứng dụng trong kinh tế. Việc giải những bài toán thực tế đã là động lực ra đời và phát triển những lý thuyết như quy hoạch tuyến tính, quy hoạch cầu phương, lý thuyết Maximin,...

Trong phạm vi chương trình trung học phổ thông chúng ta gặp những bài toán đơn giản hơn và trong nhiều trường hợp - được phát biểu không ở dưới dạng một trò chơi tường minh

.3.3. Kỹ thuật lập trình

Việc lập trình giải các bài toán trò chơi là một vấn đề phức tạp. Mỗi chương trình giải bài toán trò chơi trên thực tế là một hệ thống trí tuệ nhân tạo thu nhỏ. Công ty IBM đã từng lắp ráp siêu máy tính Deep Blue chỉ để thử nghiệm và chứng minh khả năng xây dựng trí tuệ nhân tạo. Chương trình chơi cờ vua (cờ Quốc tế) đã được xây dựng, cài đặt trên Deep Blue và đại kiện tướng, vô địch cờ thế giới Garry Kasparov đã được mời tới đấu cờ với máy tính.

Các bài toán trò chơi có đặc trưng chung:

- Các phép xử lý trạng thái ít và đơn giản,
- Lôgic để dẫn xuất, lựa chọn phép xử lý - phức tạp.

Sự phức tạp này tạo ra đặc thù cho việc tiếp cận và lập trình giải quyết các bài toán trò chơi. Về nguyên tắc với các bài toán trò chơi trên đồ thị ta có thể tính giá trị hàm Sprague - Grundy (gọi ngắn gọn là ***số Grundy***) và dựa vào nó để điều khiển trò chơi. Tuy vậy, "*Tránh vỏ dưa, gặp vỏ dừa*"! Việc tính số Grundy trong nhiều trường hợp là hết sức phức tạp, thậm chí có thể là không tính được. Khi đó người ta phải tìm cách vòng tránh. Các bài toán lập trình thường gặp với bài toán trò chơi

3.3.1. Tính trực tiếp hàm Sprague - Grundy

Bản chất của việc điều khiển trò chơi dựa trực tiếp vào hàm Sprague - Grundy là ***xác định*** hoặc ***đánh giá*** số lượng nước đi còn lại. Trong trò chơi 2 người, nếu các người chơi đều biết cách đi đúng thì người đi ở bước tiếp theo thắng được khi và chỉ khi để lại một số chẵn nước đi. Nếu đối phương còn nước đi, ta cũng sẽ còn nước để đi. Nếu hết nước đi - đối phương hết trước.

Trò chơi NIM: Có **n** đống sỏi, đống sỏi thứ **i** có $\mathbf{a_i}$ viên ($\mathbf{a_i}$>0, **i** = 1 ÷ **n**). Có 2 người chơi. Mỗi người, khi đến lượt mình phải bốc một số lượng sỏi tùy ý, lớn hơn 0 từ một đống tùy chọn. Ai đến lượt mình không còn cách bốc thì người đó thua. Hãy xác định ở lần đi đầu tiên người thứ nhất có mấy cách bốc sỏi để thắng. Hai cách bốc gọi là khác nhau nếu nó được thực hiện ở những đống khác nhau.

Dữ liệu: Vào từ file văn bản NIM.INP:

- Dòng đầu tiên chứa số nguyên n ($2 \le \mathbf{n} \le 1000$),
- Dòng thứ **i** trong **n** dòng sau chứa số nguyên $\mathbf{a_i}$ ($1 \le \mathbf{a_i} \le 10^9$).

Kết quả: Đưa ra file văn bản NIM.OUT:

- Dòng đầu tiên đưa ra số nguyên **k** - số cách bốc khác nhau có thể thực hiện. **k** = 0 nếu không có cách bốc để thắng.
- Nếu **k**> 0 thì mỗi dòng trong **k** dòng sau đưa ra 2 số nguyên **j** và **bj** theo thứ tự tăng dần của **j**, xác định cần bốc **bj** viên sỏi từ đống thứ **j**.

Ví dụ:

M.INP	***M.OUT***
	9 1 13

$a_1 = 12_{10} = 1100_2 \rightarrow g\text{-}a_1 = 3 \rightarrow$ số sỏi cần bốc: 9

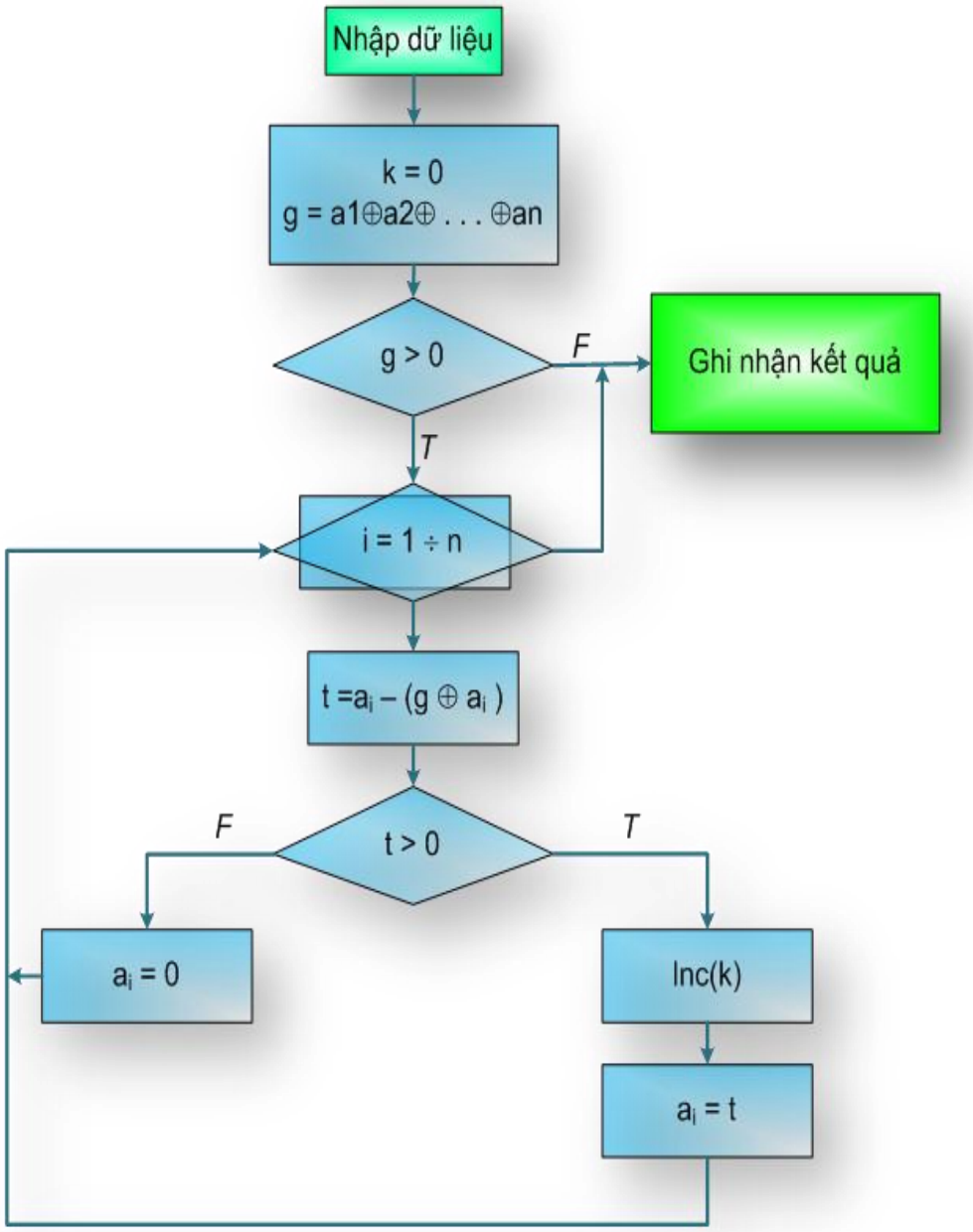
$a_2 = 8_{10} = 1000_2 \rightarrow g\text{-}a_2 = 7 \rightarrow$ số sỏi cần bốc: 1

$a_3 = 5_{10} = 0101_2 \rightarrow g\text{-}a_3 = 10 \rightarrow$ *không bốc được từ đống này*

$a_4 = 14_{10} = 1110_2 \rightarrow g\text{-}a_4 = 1 \rightarrow$ số sỏi cần bốc: 13

g = 1111

Lưu ý: Giá trị **g - a_i** có thể nhận được bằng phép tính lô gic **g⊕a_i.**



Hình 3.4. Sơ đồ thuật giải trò chơi NIM

Chương trình PASCAL

```
Program NIM;
Const tfi='NIM.INP';
      tfo='NIM.OUT';
Var a:array[1..1000] of longint;
   i,n,g,t,k:longint;
```

```
   fi,fo:text;
BEGIN
  assign(fi,tfi);
  reset(fi);
  readln(fi,n);
  for i := 1 to n do readln(fi,a[i]);
  close(fi);
  g:=0;
  for i:=1 to n do g:=g xor a[i];
  k:=0;
  for i:=1 to n do
   begin
     t:=a[i]-(a[i] xor g);
     if t>0 then
       begin
         inc(k);
         a[i]:=t
       end
         else a[i]:=0
   end;
   assign(fo,tfo); rewrite(fo);
   writeln(fo,k);
   if k > 0 then
     for i:=1 to n do if a[i]>0 then writeln(fo,i,' ',a[i]);
   close(fo)
END.
```

[2]

3.3.2. Kỹ thuật bảng phương án (Decide Table)

Giả thiết hành động cần thực hiện ở bước tiếp theo phụ thuộc vào kết quả kiểm tra n điều kiện C_1, C_2, . . ., C_n. Điều kiện C_i (i = 1 ÷ n) có thể là điều kiện lô gic với kết quả kiểm tra là Đúng (**True**) hoặc Sai (**False**) hoặc có thể là điều kiện với kết quả kiểm tra nằm trong tập có nhiều hơn 2 giá trị. Ví dụ, việc so sánh hai số p và q có thể cho kết quả <, = hoặc >.

Gọi v_i là kết quả kiểm tra điều kiện C_i. Việc kiểm tra tất cả các điều kiện từ C_1 đến C_n sẽ cho ta bộ giá trị **V** = (v1, v2, . . ., vn). Bộ giá trị này được gọi là *vec tơ điều*

kiện. Với mỗi **V** cần phải thực hiện một hành động A nào đó. Véc tơ (**V**, A) được gọi là *quy tắc hành động*.

Bảng phương án (Decide Table) là *bảng liệt kê các quy tắc hành động*.

Lý thuyết bảng phương án nghiên cứu các vấn đề:

- Phân loại bảng phương án,
- Cách xây dựng và biểu diễn bảng phương án,
- Tính chất bảng phương án,
- Sử dụng bảng phương án như công cụ phân tích và thiết kế giải thuật,
- Sử dụng bảng phương án như công cụ tự động hóa lập trình.

Bảng phương án đóng một vai trò hết sức quan trọng trong lập trình lô gic, trong các hệ thống trí tuệ nhân tạo, hệ hỗ trợ quyết định,...

Bằng bảng phương án ta có thể dễ dàng kiểm định giải thuật trong các sách báo, tài liệu giới thiệu trò chơi, chỉnh lý các sai sót liên quan tới việc xử lý các trường hợp tinh tế của trò chơi. Các sai sót này không phải là quá hiếm!

Ở đây chúng ta không đi sâu vào lý thuyết bảng phương án mà chỉ xem xét cách triển khai nó để giải quyết một số bài toán trong lĩnh vực trò chơi.

Trong bảng phương án này (cũng như trong tuyệt đại bộ phận các bảng phương án cho trò chơi) trong bảng chỉ chứa các thông tin cho phép người chơi tìm được sự lựa chọn phù hợp với chiến lược điều khiển mình theo đuổi.

Ví dụ: Bảng phương án cho bài toán Lật xúc xắc

Để điều khiển trò chơi này cần có bảng phương án B kích thước 6×Smax. $i = 1 \div 6$, $j = 1 \div$ Smax.

Bảng phương án cho bài toán này được xây dựng trên cơ sở phân tích diễn biến trò chơi từ cuối về đầu, tức là với j thay đổi từ Smax về 1.

Ở bài toán đang xét ta chỉ cần một phần của bảng phương án. Trên phương diện giải thuật, việc xây dựng một phần hay toàn bộ bảng phương án là như nhau.

B_{ij} bằng 1 để thắng cần đưa xúc xắc về trạng thái mặt trên là *i* khi S bằng *j*

B_{ij} bằng 0 sẽ thua nếu đưa xúc xắc về trạng thái mặt trên là *i* khi S bằng j

Bảng phương án được xây dựng theo nguyên tắc truy hồi. Giả thiết B_{ij} đã được xác định với $i = 1 \div 6$, $j =$ Smax $\div$ k+1. B_{ik} sẽ được xác định như sau:

$$t = \sum_{\substack{p=1,\\ p \neq i,\ p \neq 7-i}}^{6} B_{p,k+i}$$

$$B_{ik} = \begin{cases} 1 & \text{nếu } t = 0, \\ 0 & \text{trong trường hợp ngược lại.} \end{cases}$$

Để áp dụng được công thức truy hồi trên ta cần có 6 cột cuối cùng của bảng. Các cột này có thể dễ dàng xây dựng dựa trên nhận xét: khi j = Smax người đi ở bước tiếp theo, dù lật mặt nào của xúc xắc lên trên cũng thua vì làm cho tổng S > Smax, như vậy, $B_{i,Smax} = 0$, i = 1 ÷ 6.

Các cột này cũng có thể được tính theo sơ đồ chung nếu khởi tạo B_{ij} =**1**, i = 1 ÷ 6, j = Smax+1 ÷ Smax+6.

Nhận xét:

- Để xác định giá trị một cột ta chỉ cần dựa trên giá trị của 6 cột trước đó.
- Nếu xây dựng bảng phương án với 10^5 cột, ta có thể dùng nó để điều khiển trò chơi với Smax bất kỳ thỏa mãn $1 \le Smax \le 10^5$ mà không cần xây dựng lại bảng phương án (bằng cách sử dụng Smax cột cuối cùng của bảng).
- Với bài toán đang xét, ta chỉ cần B với các cột từ Smax đến S.

Chương trình PASCAL

```
Program DICE;
Const tfi='DICE.INP';
    tfo='DICE.OUT';
Var b:array[1..6,-5..1000] of byte;
   i,j,v,s,smax,n,m:longint;
   fi,fo:text;
Procedure xd_b(p,q:longint);
var  t,i,j:longint;
Begin
  t:=0;
  for i:=1 to 6 do t:= t + b[i,q-p];
  t:=t-b[p,q-p]-b[7-p,q-p];
  if t >0 then b[p,q]:=0 else b[p,q]:=1
End;
BEGIN
  assign(fi,tfi);
  reset(fi);
  readln(fi,v,s,smax);
  close(fi);
  assign(fo, tfo); rewrite(fo);
```

```
  for j := -5 to 0 do
   for i := 1 to 6 do b[i,j]:=1;
  n:=smax-s+1;
  for j:=1 to n do
   for i := 1 to 6 do xd_b(i,j);
  { Dua ra bang phuong an  }
   for i:=1 to 6 do
     begin
       for j:=1 to n do write(fo,b[i,j],' ');
       writeln(fo);
     end;
     writeln(fo);
  {  Het dua ra bang phuong an}
  m:=0; b[v,n]:=0; b[7-v,n]:=0;
  for i:= 1 to 6 do m:= m+b[i,n];
  write(fo,m);
  if m > 0 then
    for i:= 1 to 6 do if b[i,n]>0 then write(fo,' ',i);
  close(fo)
END.
```

Bảng 3.1. Bảng phương án cho bài toán lật xúc xắc

	-5	*-4*	*-3*	*-2*	*-1*	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*
1	1	1	1	1	1	1	0	1	1	0	0	0	0	0
2	1	1	1	1	1	1	0	0	1	0	0	0	0	1
3	1	1	1	1	1	1	0	0	0	1	0	0	1	1
4	1	1	1	1	1	1	0	0	0	0	1	0	0	1
5	1	1	1	1	1	1	0	0	0	0	0	1	0	0
6	1	1	1	1	1	1	0	0	0	0	0	0	1	1

Kỹ thuật bảng phương án cho phép ta tránh việc phải tính tường minh giá trị hàm Sprague - Grundy khi trò chơi không bị phân rã thành các trò chơi con độc lập.

Với những bài toán bị phân rã thành các trò chơi độc lập (không nhất thiết phải theo cùng một quy tắc chơi như bài toán ban đầu) bảng giá trị số Grundy là một thành phần của bảng phương án với ba chức năng:

- Cho biết người đi ở nước tiếp theo có thể thắng được hay không.
- Cung cấp các thông tin để tìm ra chiến lược điều khiển (tìm nước đi).
- Làm cơ sở để tính số Grundy cho các trò chơi với bộ tham số có giá trị lớn hơn.

Ở các loại trò chơi này việc lưu trữ điều khiển là không tối ưu vì miền xác định của tập trạng thái lớn. Với mỗi trạng thái của trò chơi điều khiển được tìm bằng con đường giải thuật, dựa vào bảng số Grundy đã có .

KẾT LUẬN CHƯƠNG 3

Các bài toán trò chơi khá đa dạng và thường là khó.

Lý thuyết đồ chơi được áp dụng với một số mô hình trên đồ thị, ma trận.

Kỹ thuật lập trình giải quyết bài toán trò chơi:

- Tính trực tiếp hàm Sprague - Grundy

- Kỹ thuật lập bảng phương án

Có nhiều cách tổ chức bảng phương án. Thông thường có 2 loại chính:

- Bảng phương án phục vụ cho *điều khiển tiến trình*,
- Bảng phương án phục vụ *triển khai cài đặt* chương trình, phục vụ *nhận dạng*.

Việc xây dựng tường minh bảng phương án cho phép:

- Dễ dàng tìm ra nước đi cần thiết tiếp theo, đặc biệt khi người chơi thứ hai phạm sai lầm.
- Cho phép điều khiển trò chơi theo một trong số các chiến lược:
+ Chiến thắng trong thời gian nhanh nhất có thể.
+ Kéo dài thời gian đến mức tối đa có thể.
+ Làm cho trò chơi diễn ra đa dạng: nếu chơi lại và người thứ hai lặp lại nước đi cũ, chương trình cho nước đi khác, đảm bảo xác xuất các nước đi trùng nhau ở hai ván là gần như bằng 0.

Chương 4
THUẬT TOÁN QUY HOẠCH ĐỘNG CHO TÍNH KHOẢNG CÁCH

4.1: Khoảng cách Levenshtein

Trong các thuật toán của bộ môn khoa học máy tính, khái niệm **Khoảng cách Levenshtein** thể hiện khoảng cách khác biệt giữa 2 chuỗi kí tự. Khoảng cách Levenshtein giữa chuỗi S và chuỗi T là số bước ít nhất biến chuỗi S thành chuỗi T thông qua 3 phép biến đổi là

- xoá 1 kí tự.
- thêm 1 kí tự.
- thay kí tự này bằng kí tự khác.

Khoảng cách này được đặt theo tên Vladimir Levenshtein , người đã đề ra khái niệm này vào năm 1965. Nó được sử dụng trong việc tính toán sự giống và khác nhau giữa 2 chuỗi, như chương trình kiểm tra lỗi chính tả của winword spellchecker. Ví dụ: Khoảng cách Levenshtein giữa 2 chuỗi "kitten" và "sitting" là 3, vì phải dùng ít nhất 3 lần biến đổi.

1. kitten -> sitten (thay "k" bằng "s")
2. sitten -> sittin (thay "e" bằng "i")
3. sittin -> sitting (thêm kí tự "g")

4.1.1:Thuật toán

Để tính toán Khoảng cách Levenshtein, ta sử dụng thuật toán quy hoạch động, tính toán trên mảng 2 chiều (n+1)*(m+1), với n, m là độ dài của chuỗi cần tính. Sau đây là đoạn mã *(S, T là chuỗi cần tính khoảng cách, n, m là độ dài của chuỗi S, T)*:

```
int LevenshteinDistance(char s[1..m], char t[1..n])
  // d is a table with m+1 rows and n+1 columns
  declare int d[0..m, 0..n]

  for i from 0 to m
     d[i, 0] := i
  for j from 0 to n
     d[0, j] := j

  for i from 1 to m
    for j from 1 to n
```

```
    {
      if s[i] = t[j] then cost := 0
                     else cost := 1
      d[i, j] := minimum(
                         d[i-1, j] + 1,     // trường hợp xoá
                         d[i, j-1] + 1,     // trường hợp thêm
                         d[i-1, j-1] + cost   // trường hợp thay thế
                        )
    }


  return d[m, n]
```

ví dụ, giá trị của bảng d :

		k	**i**	**t**	**t**	**e**	**n**
	0	1	2	3	4	5	6
s	1	1	2	3	4	5	6
i	2	2	1	2	3	4	5
t	3	3	2	1	2	3	4
t	4	4	3	2	1	2	3
i	5	5	4	3	2	2	3
n	6	6	5	4	3	3	2
g	7	7	6	5	4	4	3

		S	**a**	**t**	**u**	**r**	**d**	**a**	**y**
	0	1	2	3	4	5	6	7	8
S	1	0	1	2	3	4	5	6	7
u	2	1	1	2	2	3	4	5	6
n	3	2	2	2	3	3	4	5	6
d	4	3	3	3	3	4	3	4	5
a	5	4	3	4	4	4	4	3	4
y	6	5	4	4	5	5	5	4	3

Như vậy, kết quả cần tính chính là giá trị của d[n, m]. Thực chất, thuật toán này là một phần trong giải pháp Longest common subsequence problem

Ví dụ :

Những từ " computer " và " commuter " rất giống nhau, và một sự thay đổi của chỉ một chữ, p-> m sẽ thay đổi từ đầu tiên thành từ thứ hai. Từ " sport "có thể được thay đổi thành " sort " bằng cách chữ " p ", hoặc tương tự ," sort " có thể được thay đổi thành " sport " bằng cách chèn chữ " p ".

Cách *tính khoảng cách* của hai chuỗi, s1 và s2, được định nghĩa là số lượng tối thiểu *đột biến điểm* cần thiết để thay đổi s1 trong s2, nơi một đột biến điểm là một trong những:

1. Thay đổi kí tự,
2. Chèn một kí tự hoặc
3. Xóa một kí tự

Các mối quan hệ tái phát sau đây xác định khoảng cách chỉnh sửa, d (s1, s2), của hai chuỗi s1 và s2:

d ("," ") = 0 -" " = chuỗi rỗng

d (s," ") = d (" ", s) = | s | - tức là chiều dài của s

d (s1 + ch1, s2 + ch2)

= Min (d (s1, s2) + if ch1 = ch2 then 0 else 1

d (s1 + ch1, s2) + 1,

d (s1, s2 + ch2) + 1)

Hai quy tắc đầu tiên ở trên rõ ràng là đúng, vì vậy nó chỉ cần thiết xem xét cuối cùng. Ở đây, không chuỗi là chuỗi rỗng, vì vậy mỗi chuỗi có một ký tự cuối cùng, ch1 và ch2 tương ứng. Bằng cách nào đó, ch1 và ch2 phải được giải thích trong một chỉnh sửa của s1 + ch1 vào ch2 + s2. Nếu ch1 bằng ch2, chúng có thể được kết hợp để không bị trùng, nghĩa là 0, và chỉnh sửa tổng thể khoảng cách là d (s1, s2). Nếu ch1 khác ch2, sau đó ch1 có thể được thay đổi thành ch2, tức là 1, đưa ra một tổng thể d chi phí (s1, s2) 1. Một khả năng khác là để xóa ch1 và chỉnh sửa s1 vào s2 + ch2, d (s1, s2 + ch2) 1. Khả năng cuối cùng là chỉnh sửa s1 + ch1 vào s2 và sau đó chèn CH2, d (s1 + ch1, s2) 1. Không có lựa chọn thay thế khác. Chúng ít tốn kém, tiết kiệm thời gian, các lựa chọn thay thế.

Các mối quan hệ bao hàm sự tái phát một thói quen bậc ba-đệ quy rõ ràng. Điều này là một ý tưởng không tốt vì nó là theo cấp số nhân chậm, và không thực tế cho chuỗi hơn một số rất ít ký tự.

Kiểm tra các mối quan hệ cho thấy d (s1, s2) chỉ phụ thuộc vào d (s1, s2 '), nơi s1' ngắn hơn s1, s2 hoặc 'ngắn hơn s2, hoặc cả hai. Điều này cho phép các chương trình quy hoạch động được sử dụng.

Một ma trận hai chiều, m [0 .. | S1 |, 0 .. | s2 |] được sử dụng để giữ các giá trị khoảng cách :

m [i, j] = d (s1 [1 .. i], s2 [1 .. j])

m [0,0] = 0

m [i, 0] = i, i = 1 .. | S1 |

m [0, j] = k, j = 1 .. | s2 |

m [i, j] = min (m [i-1, j-1]

+ Nếu s1 [i] = s2 [j] là 0 khác 1 fi,

m [i-1, j] + 1,

m [i, j-1] + 1), i = 1 .. | S1 |, j = 1 .. | s2 |

m [,] có thể được tính từng hàng . Hàng m [i,] chỉ phụ thuộc vào hàng m [i-1,]. Độ phức tạp của thuật toán này là O (| S1 | * | s2 |). Nếu s1 và s2 có một 'chiều dài, khoảng `n` tương tự nói, phức tạp là O (n2), tốt hơn nhiều hơn theo cấp số nhân!

Ví dụ:

Chuỗi s1: appropriate meaning

Chuỗi s2: approximate matching

appropri ate mean ing

| | | | | | | | | | | |

approximate matching

d(s1,s2) =7

4.1.2 : Độ phức tạp

Độ phức tạp của thuật toán là O (| s1 | * | s2 |), tức là O (n ^{2}) nếu độ dài của cả hai chuỗi là về `n '. Các không gian phức tạp cũng là O (n ^{2}) nếu toàn bộ ma trận được lưu giữ trong một dấu vết, trở lại để tìm một sự liên kết tối ưu. Nếu chỉ có giá trị khoảng cách sửa là cần thiết, chỉ có hai hàng của ma trận cần được phân bổ, chúng có thể được "thay thế", và sự phức tạp không gian là sau đó O (| S1 |), tức là O (n).

4.1.3: Biến thể

Chi phí của các đột biến điểm có thể thay đổi được số khác hơn là 0 hoặc 1.Khoảng cách tuyến tính, chi phí đôi khi được sử dụng khi chạy trong lần (hoặc xóa) chiều

dài `x ', có một chi phí` ax + b', cho các hằng số `a 'và' b '. Nếu b> 0, điều này gây bất lợi cho nhiều chạy ngắn của chèn và xóa bỏ.

4.2 : Dãy con chung dài nhất

Dãy con chung dài nhất (LCS) của hai trình tự, s1 và s2, là một dãy con của cả s1 và s2 có độ dài tối đa có thể. Sự giống nhau hơn là s1 và s2 là, còn là LCS của họ.

4.3 : Các thuật toán khác

Có những thuật toán nhanh hơn cho các vấn đề tính khoảng cách, và các vấn đề tương tự. Một số các thuật toán được nhanh chóng nếu một số điều kiện tổ chức, ví dụ như các dây tương tự, hoặc không giống nhau, hoặc bảng chữ cái là lớn, vv.

Ukkonen (1983) đã đưa ra một thuật toán với thời gian tồi tệ nhất trường hợp phức tạp O (n * d), và sự phức tạp trung bình là O (n + d^2), trong đó n là chiều dài của dây, và d là khoảng cách chỉnh sửa của họ. Này là nhanh chóng cho các chuỗi tương tự trong đó d là nhỏ, tức là khi d << n.

4.4 : Ứng dụng

Tập tin sửa đổi

Lệnh Unix là tìm thấy *sự khác biệt* giữa các tập tin f1 và f2, tạo ra một *kịch bản chỉnh sửa* để chuyển đổi thành f1 f2. Nếu hai (hoặc nhiều) máy tính chia sẻ các bản sao của một tập tin F lớn, và một người nào đó trên máy tính-1 sửa đổi F = F.bak , thực hiện một vài thay đổi, để cung cấp cho F.new , nó có thể rất tốn kém và / hoặc chậm để truyền tải các tập tin chỉnh sửa toàn bộ F.new để máy 2.Tuy nhiên, khác F.bak F.new sẽ cung cấp cho một *nhỏ* kịch bản chỉnh sửa có thể truyền một cách nhanh chóng để máy 2 trường hợp bản sao địa phương của tập tin có thể được cập nhật bằng F.new .

khác đối xử với một dòng toàn bộ như một "nhân vật" và sử dụng một thuật toán sửa-khoảng cách đặc biệt đó là nhanh chóng khi các "bảng chữ cái" là lớn và có rất ít cơ hội phù hợp giữa các yếu tố của hai chuỗi (các tập tin). Ngược lại, có rất nhiều cơ hội ký tự trận đấu trong DNA mà kích thước bảng chữ cái chỉ là 4, {A, C, G, T}.

Từ xa Cập nhật vấn đề màn hình :

Nếu một chương trình máy tính trên máy tính-1 đang được sử dụng bởi một người nào đó từ một màn hình trên (ở xa) máy 2, ví dụ như thông qua rlogin vv, sau đó máy-1 có thể cần phải cập nhật các màn hình trên máy tính-2 như số tiền thu được tính toán. Một cách tiếp cận là cho chương trình (trên máy tính-1) để giữ một "hình ảnh" của những gì màn hình hiện nay (trên máy 2) và một hình ảnh của những gì nó phải trở thành. Sự khác biệt có thể được tìm thấy (bằng một thuật toán liên quan đến chỉnh sửa-khoảng cách) và sự khác biệt truyền ... tiết kiệm trên truyền dẫn băng rộng.

Sửa lỗi chính tả :

Các thuật toán liên quan đến khoảng cách chỉnh sửa có thể được sử dụng trong sửa bản chính tả. Nếu một văn bản có chứa một từ, w, mà không có trong từ điển, một 'gần' từ, tức là một trong với một chỉnh sửa khoảng cách nhỏ với w, có thể được xem như một sự điều chỉnh.

Lỗi chuyển vị rất phổ biến trong văn bản bằng văn bản. Một chuyển vị có thể được coi như một xóa cộng với một chèn, nhưng một biến thể đơn giản trên các thuật toán có thể xử lý một chuyển vị như một đột biến điểm duy nhất.

Quy hoạch động chủ yếu là đệ quy mà không có sự lặp lại. Phát triển một thuật toán lập trình quy hoạch động thường bao gồm hai bước riêng biệt:

- **Formulate vấn đề đệ quy** . Viết ra một công thức cho toàn bộ vấn đề như một sự kết hợp đơn giản của câu trả lời cho bài toán nhỏ hơn.
- **Xây dựng các giải pháp để tái phát từ dưới lên** . Viết một thuật toán bắt đầu với trường hợp cơ sở và hoạt động theo cách của nó đến giải pháp cuối cùng.

Các thuật toán lập trình động cần phải lưu trữ các kết quả của bài toán trung gian. Điều này thường được *nhưng không luôn luôn* thực hiện với một số loại bảng. Bây giờ chúng tôi sẽ giới thiệu một số ví dụ về các vấn đề trong đó giải pháp được dựa trên chiến lược lập trình quy hoạch động tính khoảng cách.

Dòng chữ "computer" và "commuter" rất giống nhau, và một sự thay đổi của chỉ một từ, p-m, sẽ thay đổi từ đầu tiên vào thứ hai. Từ "sport" có thể được thay đổi thành "sort" bởi việc xoá 'p', hoặc tương đương, 'sort' có thể được thay đổi thành 'sport' bằng cách chèn của 'p'.Tính khoảng cách của hai chuỗi, s1 và s2, được định nghĩa là số lượng tối thiểu của đột biến điểm cần thiết để thay đổi s1 vào s2, trong đó một đột biến điểm là một trong những:

- thay đổi ký tự,
- chèn một ký tự hoặc
- xóa một ký tự

Ví dụ, khoảng cách hiệu chỉnh giữa các *FOOD* và *MONEY* là nhiều nhất là bốn:

F OOD - → MOO D - → MON-D

- → MONED - → MONEY

Tính khoảng cách : Ứng dụng

Có rất nhiều ứng dụng của các thuật toán tính Khoang cách. Dưới đây là một số ví dụ:

Sửa lỗi chính tả

Nếu một văn bản có chứa một từ không có trong từ điển, một 'close' từ, tức là một trong với một chỉnh sửa khoảng cách nhỏ, có thể được xem như một sự điều chỉnh. Hầu hết các xử lý văn bản các ứng dụng, chẳng hạn như Microsoft Word , đã

kiểm tra chính tả và cơ sở sửa chữa. Khi Word, ví dụ, tìm thấy một từ không đúng, nó làm cho lời đề nghị của người thay thế có thể.

Phát hiện đạo văn :

Nếu một người nào đó bản sao, nói rằng, một chương trình C và làm một vài thay đổi ở đây và ở đó, ví dụ, thay đổi tên biến, thêm một bình luận của hai, khoảng cách chỉnh sửa giữa nguồn và bản sao là rất nhỏ. Chỉnh sửa khoảng cách cung cấp một dấu hiệu của sự tương đồng đó có thể là quá gần trong một số trường hợp.

Tính toán Sinh học phân tử DNA là một polymer. Các đơn vị monome của DNA nucleotide, và các polymer được biết đến như một "polynucleotide." Mỗi nucleotide bao gồm một đường 5 carbon (deoxyribose), nitơ có chứa cơ sở gắn liền với đường, và một nhóm phosphate. Có bốn loại khác nhau của các nucleotide trong DNA, chỉ khác nhau ở các cơ sở đạm. Bốn nucleotide được cho một chữ viết tắt thư là viết tắt cho bốn cơ sở.

• A-adenine

• G-guanine

• C-cytosine

• T-thymine

Xoắn kép của phân tử DNA với nucleotide hình của đôi xoắn của phân tử DNA nucleotide đi đây. Các chỉnh sửa khoảng cách như các thuật toán được sử dụng để tính toán khoảng cách giữa các trình tự DNA (chuỗi hơn A, C, G, T, hoặc các chuỗi protein (trên một bảng chữ cái của 20 amino acid), cho các mục đích khác nhau, ví dụ như:

• tìm gen hoặc protein có thể chia sẻ các chức năng hoặc tài sản

• để suy ra các mối quan hệ gia đình và cây tiến hóa hơn sinh vật khác nhau.

Speech Recognition:

Các thuật toán tương tự như đối với các vấn đề tính-khoảng cách được sử dụng trong một số hệ thống nhận dạng giọng nói. Tìm một ma trận chặt chẽ giữa một lời nói mới và một trong một thư viện các lời phát biểu phân loại.

Khoảng cách chỉnh sửa thuật toán

Một cách tốt hơn để hiển thị quá trình chỉnh sửa này là đặt các từ trên khác:

S	D	I	M	D	M
MA	A	R	TT	H	S S

Từ đầu tiên có một khoảng cách cho mỗi chèn (I) và từ thứ hai có một khoảng cách cho mỗi xóa (D). Cột với hai nhân vật khác nhau tương ứng với thay thế (S). ma trận (M) không được tính. Các *bảng Chỉnh sửa* được định nghĩa như là một chuỗi trong bảng chữ cái M, S, I, D mô tả một chuyển đổi một chuỗi thành khác. Ví dụ

S D I M D M

1+ 1+ 1+ 0+ 1+ 0+ = 4

Nói chung, nó không phải là dễ dàng để xác định chỉnh sửa tối ưu khoảng cách. Ví dụ, khoảng cách giữg *ALGORITHM* và *ALTRUISTIC* nhiều nhất là 6.

A L G O R I T H M

A L T R U I S T I C

Là tối ưu này?

4.5: Tính Khoảng cách: Quy hoạch động, Lập trình thuật toán

Giả sử chúng ta có một m-chuỗi kí tự A và một chuỗi n ký tự B. Xác định điện tử (i, j) là khoảng cách giữa tôi sửa ký tự đầu tiên của A và các nhân vật j đầu tiên của B. Ví dụ,

A← L _G O R I T H M

A L T R U I S T I C

Chỉnh sửa khoảng cách giữa toàn bộ chuỗi A và B là E (m, n). Đại diện cho khoảng cách các trình tự sửa có một "quan trọng *Hạ tầng cơ sở tối ưu* ". Nếu chúng ta loại bỏ các cột cuối cùng, các cột còn lại phải đại diện cho sửa chuỗi ngắn nhất cho các chuỗi con còn lại.Chỉnh sửa khoảng cách là 6 cho hai từ sau đây.

A L G O R I T H M

A L T R U I S T I C

Nếu chúng ta loại bỏ các cột cuối cùng, khoảng cách chỉnh sửa làm giảm đến 5.

A L G O R I T H

A L T R U I S T I

Chúng ta có thể sử dụng tài sản hạ tầng cơ sở tối ưu để đưa ra một công thức đệ quy của vấn đề chỉnh sửa khoảng cách. Có một vài trường hợp cơ sở rõ ràng:

• Cách duy nhất để chuyển đổi một chuỗi rỗng vào một chuỗi các ký tự j là bằng cách làm chèn j. Do đó

E (0, j) = j

• Cách duy nhất để chuyển đổi một chuỗi của tôi ký tự vào chuỗi sản phẩm nào là với tôi xóa bỏ:

E (i, 0) = i

Có bốn khả năng cột cuối cùng trong sửa chuỗi ngắn nhất có thể:

Xóa: mục cuối trong hàng dưới cùng có sản phẩm nào.

i=3

A__L G O R I T H M
A L←j=2_T R U I S T I C
Trong trường hợp này
E (i, j) = E (i - 1, j) + 1
Chèn: Các mục cuối cùng trong hàng đầu tiên có sản phẩm nào.
i=5
A← L G_ O R I T H M
A L T R U I S T I C
j = 5

Trong trường hợp này
E (i, j) = E (i, j - 1) + 1
Thay thế : Cả hai hàng có ký tự trong cột cuối cùng.
i=4
A← L _ G O R I T H M
A L T R U I S T I C
j = 3

Nếu các kí tự khác nhau, sau đó
E (i, j) = E (i - 1, j - 1) + 1
A← L _G O R I T H M
i=5
A L T R U I S T I C
j = 4
Nếu kí tự là như nhau, không thay thế là cần thiết:
E (i, j) = E (i - 1, j - 1)
Do đó, khoảng cách sửa E (i, j) là nhỏ nhất trong bốn khả năng:
E (i, j) = min
E (i - 1, j) + 1
E (i, j - 1) + 1 E (i - 1, j - 1) + 1 nếu A [i] = B [j] E (i - 1, j - 1) nếu A [i] = B [j]
Xem xét các ví dụ về sửa giữa các từ "ARTS" và "MATHS":
A R T S
M A T H S
Chỉnh sửa khoảng cách sẽ là trong E (4, 5). Nếu chúng ta đệ quy để tính toán, chúng tôi sẽ có
E (3, 5) + 1
E (4, 5) = min E (4, 4) + 1
E (3, 4) + 1 nếu A [4] = B [5]

E (3, 4) nếu A [4] = B [5]

Đệ quy rõ ràng dẫn đến mô hình cuộc gọi lặp đi lặp lại tương tự như chúng ta đã thấy trong chuỗi Fibonnaci. Để tránh điều này, chúng tôi sẽ sử dụng phương pháp DP. Chúng tôi sẽ xây dựng các giải pháp từ dưới lên. Chúng tôi sẽ sử dụng các trường hợp cơ sở E (0, j) để điền vào dòng đầu tiên và trường hợp cơ sở điện tử (i, 0) để điền vào cột đầu tiên. Chúng tôi sẽ điền vào E hàng ma trận còn lại bởi hàng.

		A	R	T	S
M					
A					
T					
H					
S					

		A	R	T	S	
					4 →	
				3		
M	↓ 1					
A	↓ 2					
T	↓ 3					
H	↓ 4					
S	↓ 5					

Hàng đầu tiên và mục cột đầu tiên sử dụng các trường hợp cơ sở

Bây giờ chúng tôi có thể điền vào hàng thứ hai. Bảng này không chỉ cho thấy giá trị của các tế bào E [i, j] mà còn mũi tên

chỉ ra cách nó đã được tính toán sử dụng giá trị trong E [i - 1, j], E [i, j - 1] và E [i - 1, j - 1]. Vì vậy, nếu một tế bào E [i, j] có một mũi tên xuống từ E [i - 1, j] thì tối thiểu đã được tìm thấy sử dụng E [i - 1, j]. Đối với một mũi tên bên phải, tối thiểu đã được tìm thấy sử dụng E [i, j - 1]. Đối với một đường chéo mũi tên xuống bên phải, tối thiểu đã được tìm thấy

sử dụng E [i - 1, j - 1]. Có các tế bào nhất định có hai mũi tên chỉ vào nó. Trong trường hợp này, tối thiểu có thể được lấy từ E chéo [i - 1, j - 1] và một trong E [i - 1, j] và E [i, j - 1].Chúng tôi sẽ sử dụng các mũi tên sau để xác định kịch bản chỉnh sửa.

		A	R	T	S
M	↓ 1	1			
A	↓ 2				
T	↓ 3				
H	↓ 4				
S	↓ 5				

		A	R	T	S
					4
				3	→
M	↓ 1	1	2 →		
A	↓ 2				

	↓					
T	3					
	↓					
H	4					
	↓					
S	5					

Máy tính điện tử [1, 1] và E [1, 2]

		A	R	T	S
M	↓				
	↓				
A	2				
	↓				
T	3				
	↓				
H	4				
	↓				
S	5				

		A	R	T	S
					4
				3	→
					4
M	↓			3	→
	↓				
A	2				

	↓						
T	3						
	↓						
H	4						
	↓						
S	5						

Máy tính điện tử [1, 3] và E [1, 4]

Một kịch bản chỉnh sửa có thể được chiết xuất bằng cách làm theo một con đường duy nhất từ E [0, 0] E [4, 5]. Có ba con đường có thể trong ví dụ hiện tại. Chúng ta hãy làm theo các đường dẫn và tính toán các kịch bản chỉnh sửa. Trong việc thực hiện thực tế của phiên bản lập trình năng động của các thuật toán chỉnh sửa khoảng cách, các mũi tên sẽ được ghi lại cách sử dụng một cấu trúc dữ liệu thích hợp. Ví dụ, mỗi tế bào trong ma trận có thể là một kỷ lục với các lĩnh vực cho các giá trị (số) và cờ cho ba mũi tên đến.

		A	R	T	S
			→	→	4
		1	2	3	→
	↓		→	→	4
M	1	1	2	3	→
	↓		→	→	4
A	2	1	2	3	→
	↓	↓			3
T	3	2	2	2	→
	↓	↓	↓	↓	
H	4	3	3	3	3
	↓	↓	↓	↓	
S	5	4	4	4	3

Bảng cuối cùng với tất cả các E [i, j] mục tính

Giải pháp đường 1:

1 + 0 + 1 + 1 + 0 = 3

		A	R	T	S
			→	→	4
		1	2	3	→
	↓		→	→	4
M	1	1	2	3	→
	↓		→	→	4
A	2	1	2	3	→
	↓	↓			3
T	3	2	2	2	→
	↓	↓	↓	↓	
H	4	3	3	3	3
	↓	↓	↓	↓	
S	5	4	4	4	3

Chỉnh sửa có thể kịch bản. Các mũi tên màu đỏ từ E [0, 0] E [4, 5] cho thấy các đường dẫn có thể được theo sau để trích xuất chỉnh sửa kịch bản.

Đường 2 giải pháp:

1 + 1 + 0 + 1 + 0 = 3

Giải pháp 3 con đường:

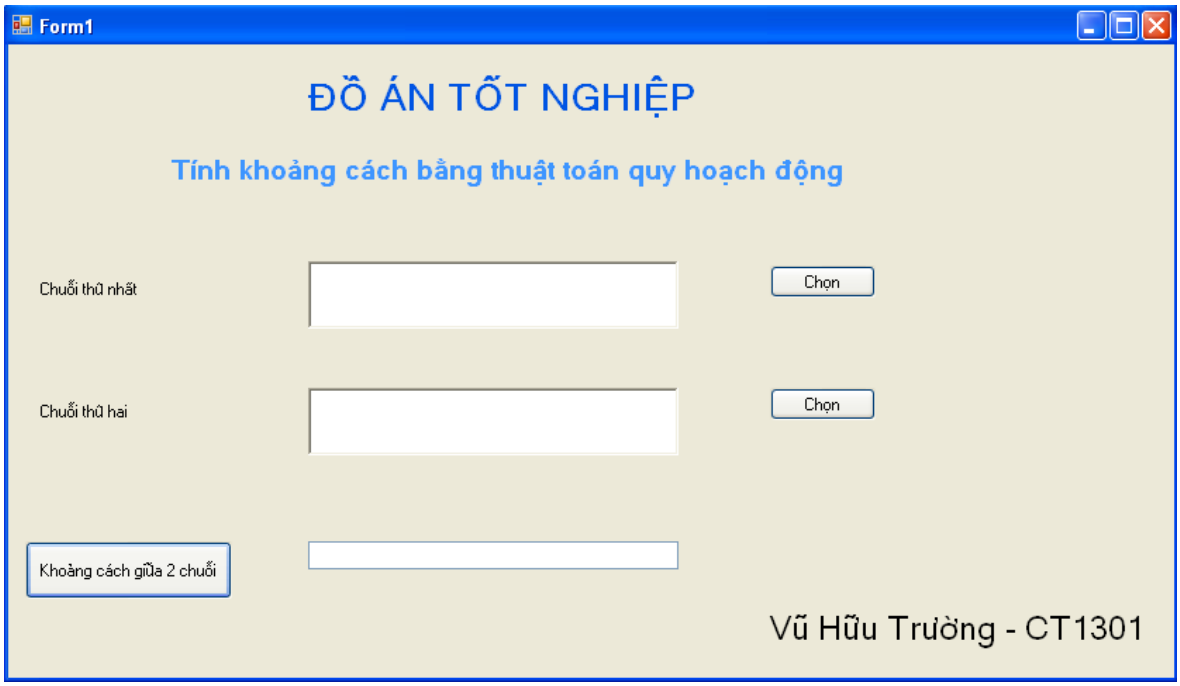
1 + 0 + 1 + 0 + 1 + 0 = 3

4.6 :Phân tích của DP Tính Khoảng cách

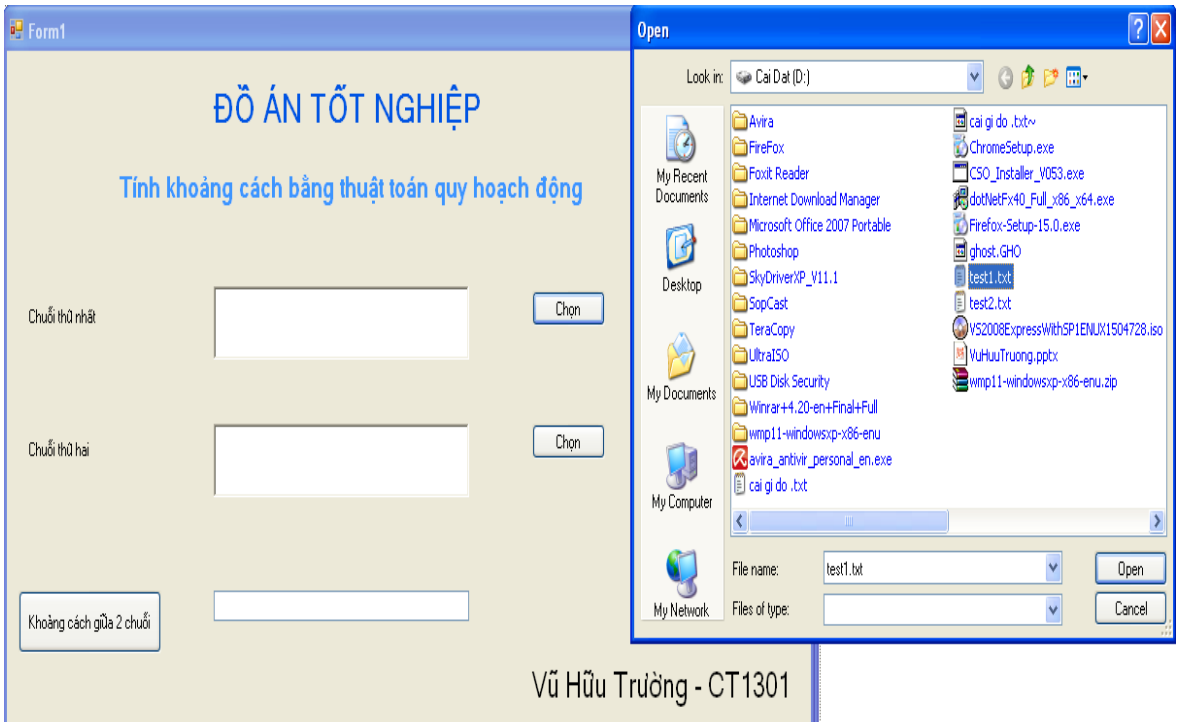
Có Θ (n2) mục trong ma trận. Mỗi mục E (i, j) có Θ (1) thời gian để tính toán. Tổng thời gian chạy là Θ (n2).

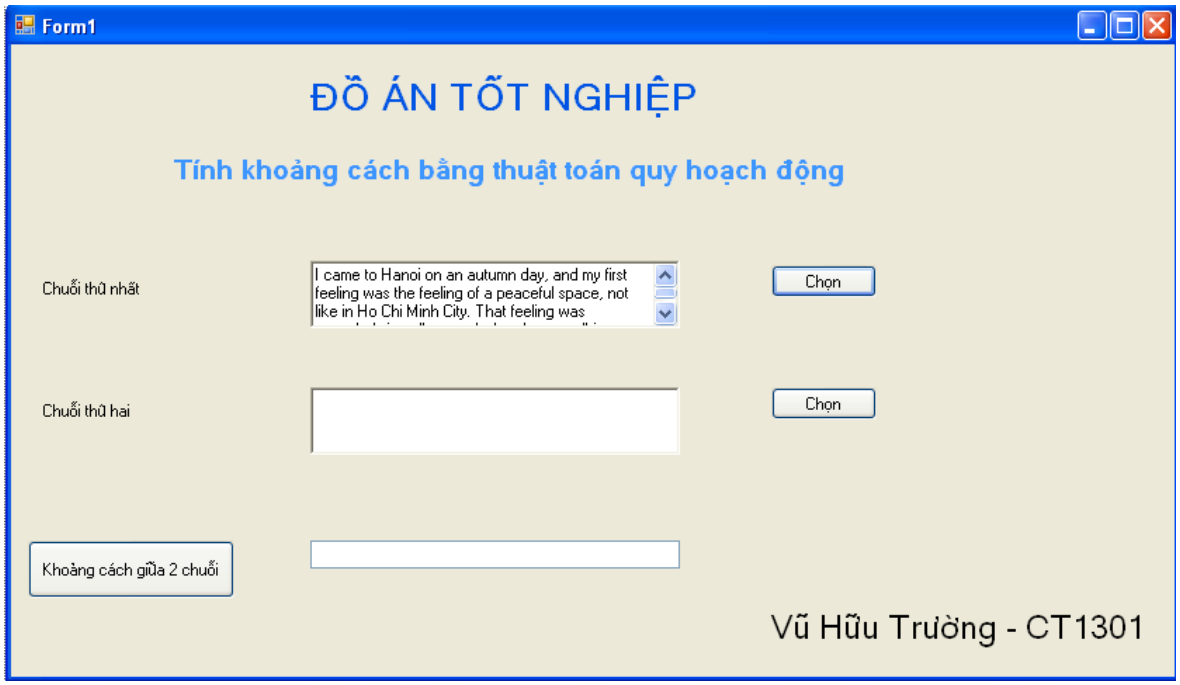
4.7. Xây dựng chương trình tính khoảng cách bằng thuật toán quy hoạch động

- Giao diện chương trình :

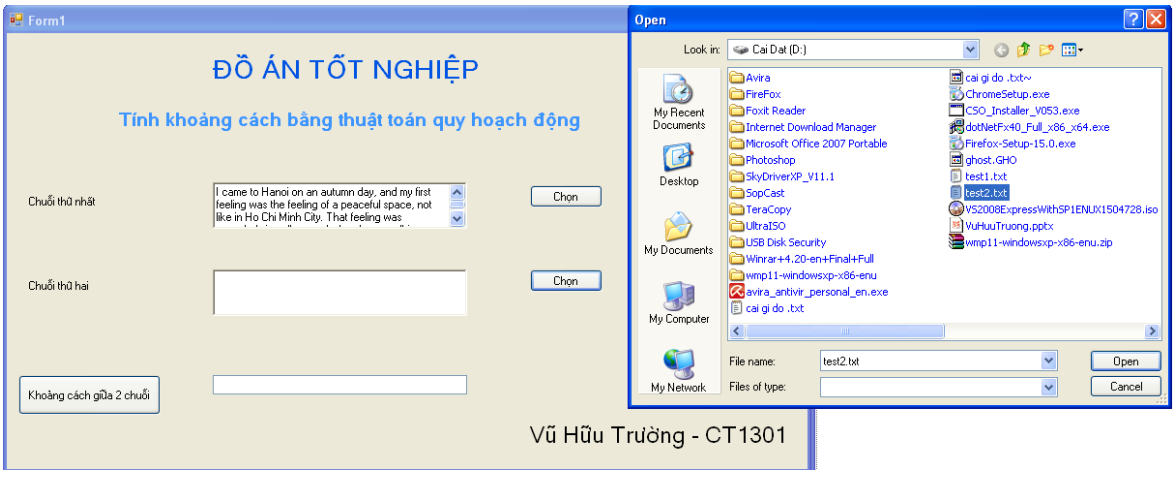


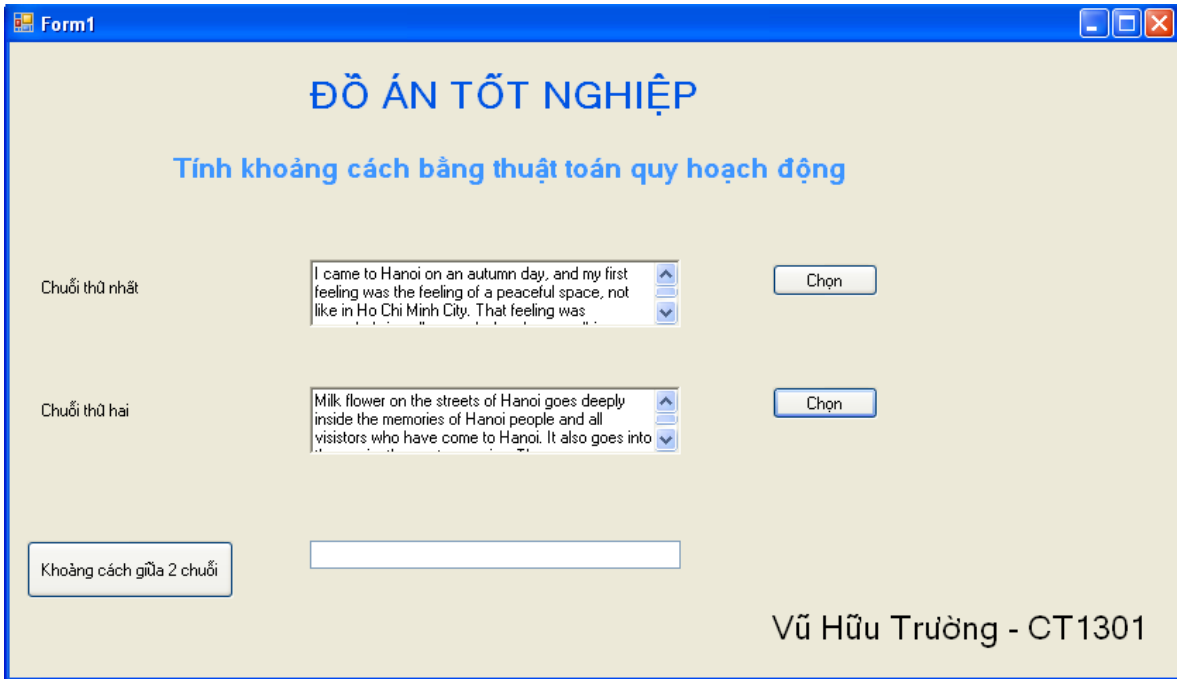
- Khởi chạy chương trình và nhập chuỗi thứ nhất :



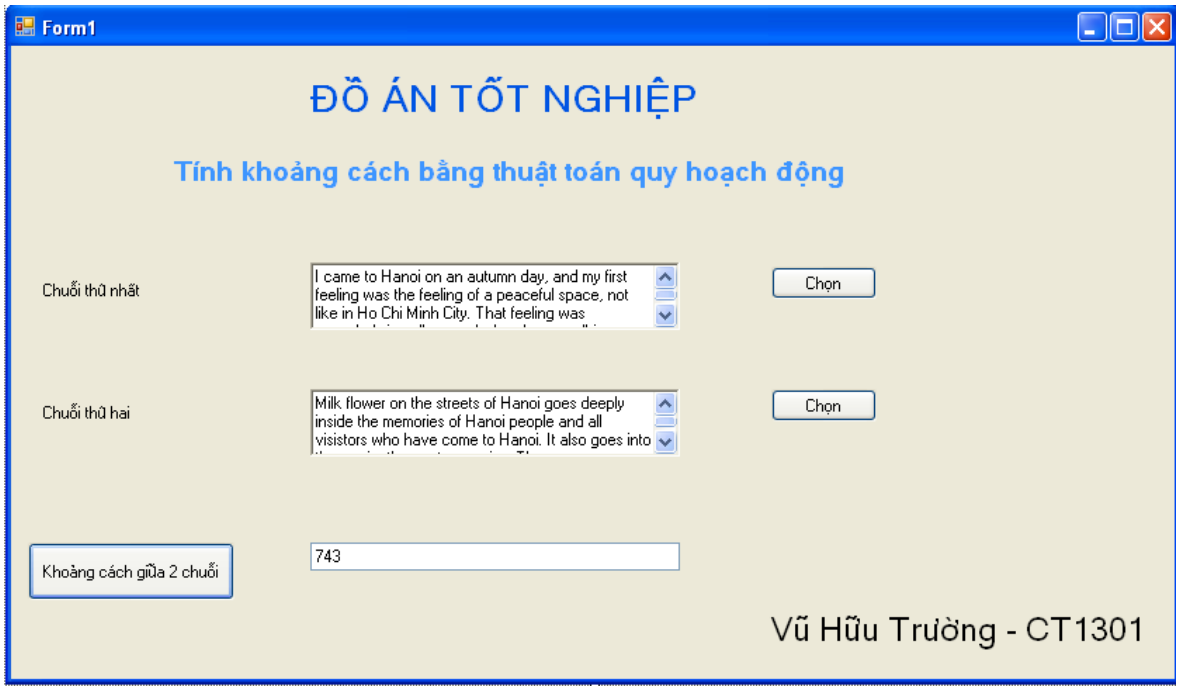


- Nhập chuỗi thứ 2 :





- Kết quả đưa ra là khoảng cách 2 chuỗi cần tìm :



KẾT LUẬN

Đồ án đã thu được một số kết quả cơ bản sau:

1. Đồ án đã hệ thống hóa được các vấn đề xung quanh phương pháp quy hoạch động. Đây là một phương pháp hay được sử dụng để giải các bài toán trong thực tế cũng như trong các kỳ thi học sinh giỏi.
2. Đồ án cũng đã đưa ra được một số kỹ thuật được dùng để giải các bài toán quy hoạch động. Những kỹ thuật này được rút ra từ kinh nghiệm giải các bài toán dựa theo phương pháp quy hoạch động.
3. Nhìn chung để biết được một bài toán nào đó có thể giải bằng phương pháp quy hoạch động là rất khó. Những kỹ thuật giải bài toán quy hoạch động đưa ra trong đồ án là định hướng giúp ta lập được hệ thức quy hoạch động một cách nhanh chóng và tổ chức chương trình sao cho đỡ tốn bộ nhớ.
4. Do thời gian thực hiện hạn chế và kiến thức còn hạn chế nên em mới chỉ nghiên cứu được một số lý thuyết cơ bản trong thuật toán quy hoạch động cho tính khoảng cách. Còn nhiều lý thuyết cũng như kỹ thuật em vẫn chưa tìm hiểu, khai thác và ứng dụng vào các bài toán thực tế. Mặc dù đã rất cố gắng, song do năng lực và trình độ có hạn nên trong quá trình thực hiện bài tập em đã không tránh khỏi những thiếu sót. Kính mong các thầy cô và các bạn quan tâm giúp đỡ chỉ bảo để chương trình của em một hoàn thiện hơn.
5. Như vậy những vấn đề mà đồ án nêu ra là phù hợp với yêu cầu của thực tiễn, đáp ứng nhu cầu đào tạo bồi dưỡng học sinh giỏi môn tin trong giai đoạn hiện nay, các nhiệm vụ đặt ra đã hoàn thành. Đồ án đã đạt được mục đích đề ra. Tuy nhiên các vấn đề xung quanh thuật toán quy hoạch động cũng như để giải được nhiều bài toán quy hoạch động và các bài toán tính khoảng cách trong thực tiễn cần phải có một quá trình nghiên cứu sâu hơn nữa.

TÀI LIỆU THAM KHẢO

Tiếng Việt

[1] Hồ Sĩ Đàm (chủ biên), Đỗ Đức Đông, Lê Minh Hoàng, Nguyễn Thanh Hùng (2009), *Tài liệu giáo khoa chuyên tin*, NXB Giáo dục, tr. 99-109 .

[2] Hồ Sĩ Đàm (chủ biên), Nguyễn Thanh Tùng, Lê Minh Hoàng, Nguyễn Thanh Hùng, Đỗ Đức Đông (2011), *Tài liệu tập huấn phát triển chuyên môn giáo viên Trường THPT Chuyên* (tài liệu lưu hành nội bộ), Bộ giáo dục và đào tạo, tr. 28-42.

[3] Nguyễn Hữu Điển (2005), *Một số vấn đề về thuật toán*, NXB Giáo dục, tr. 133-173.

[4] Vũ Mạnh Hà (2011), "Giới thiệu phương pháp quy hoạch động", *Tài liệu hội thảo phát triển chuyên môn giáo viên tin học Trường THPT chuyên* (tài liệu lưu hành nội bộ), Bộ giáo dục và đào tạo, tr. 92-107.

[5] Đỗ Thị Linh, Tổ chức dữ liệu và thuật toán cho các bài toán quy hoạch động, Đồ án thạc sĩ khoa học máy tính, Thái Nguyên - 2012

[6] Một số thông tin trên Website http://csharpviet.freevnn.com/thuat-toan/22/58-tim-hieu-thuat-toan-quy-hoach-dong.html

Tiếng Anh

[7] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest (2009), *Introduction to Algorithms*, pp. 359-397.

[8] David Harel with Yishai Feldman (2004), *Algorithmics the spirit of computing*, pp. 89-98.

[9] Donald E. Knuth, *The Art Of Computer Programming, Volumes 1-3***,** Addison Pub. Co.

[10] Một số thông tin trên các website :

http://www.csse.monash.edu.au/~lloyd/tildeAlgDS/Dynamic/Edit/

http://alikhuram.wordpress.com/2013/04/27/dynamic-programming-edit-distance/