

MỤC LỤC

LỜI NÓI ĐẦU	- 6 -
Chương 1 : GIỚI THIỆU THIẾT KẾ, ĐÁNH GIÁ THUẬT TOÁN .	- 8 -
I. Định nghĩa trực quan về Thuật toán.....	- 8 -
1. Định nghĩa	- 8 -
2. Các đặc trưng cơ bản của thuật toán.....	- 9 -
3. Đặc tả thuật toán	- 9 -
II. Các dạng diễn đạt thuật toán	- 9 -
1. Dạng lưu đồ (sơ đồ khối)	- 10 -
2. Dạng ngôn ngữ tự nhiên	- 10 -
3. Ngôn ngữ lập trình.....	- 10 -
4. Dạng mã giả	- 10 -
III. Thiết kế thuật toán	- 12 -
1. Modul hóa và thiết kế từ trên xuống (Top-Down).....	- 13 -
2. Phương pháp làm mịn dần (hay tinh chế từng bước).....	- 13 -
3. Một số phương pháp thiết kế	- 15 -
IV. Phân tích thuật toán.....	- 17 -
1. Các bước trong quá trình phân tích đánh giá thời gian chạy của thuật toán	- 17 -
2. Các ký hiệu tiệm cận.....	- 18 -
3. Một số lớp các thuật toán	- 19 -
4. Phân tích thuật toán đệ qui.	- 21 -
5. Các phép toán trên các ký hiệu tiệm cận	- 25 -
6. Phân tích trường hợp trung bình	- 26 -
V. Tối ưu thuật toán	- 27 -
1. Kỹ thuật tối ưu các vòng lặp.....	- 27 -
2. Tối ưu việc rẽ nhánh	- 30 -
Bài tập.....	- 30 -
Chương 2 : PHƯƠNG PHÁP CHIA ĐỂ TRỊ	- 33 -
I. Mở đầu.....	- 33 -
1. Ý tưởng.....	- 33 -
2. Mô hình	- 33 -
II. Thuật toán tìm kiếm nhị phân.....	- 33 -
1. Phát biểu bài toán.....	- 33 -
2. Ý tưởng.....	- 33 -
3. Mô tả thuật toán	- 33 -

4. Độ phức tạp thời gian của thuật toán	34 -
5. Cài đặt.....	34 -
III. Bài toán MinMax	35 -
1. Phát biểu bài toán.....	35 -
2. Ý tưởng.....	35 -
3. Thuật toán	35 -
4. Độ phức tạp thuật toán	36 -
5. Cài đặt.....	36 -
IV. Thuật toán QuickSort.....	36 -
1. Ý tưởng.....	37 -
2. Mô tả thuật toán	37 -
3. Độ phức tạp của thuật toán.....	38 -
V. Thuật toán nhân Strassen nhân 2 ma trận.....	39 -
1. Bài toán	39 -
2. Mô tả	39 -
VI. Bài toán hoán đổi 2 phần trong 1 dãy.....	41 -
1. Phát biểu bài toán.....	41 -
2. Ý tưởng.....	41 -
3. Thuật toán	41 -
4. Độ phức tạp thuật toán	43 -
5. Cài đặt.....	43 -
VII. Trộn hai đường trực tiếp	44 -
1. Bài toán	44 -
2. Ý tưởng.....	44 -
3. Thiết kế	45 -
Bài tập.....	50 -
Chương 3 : PHƯƠNG PHÁP QUAY LUI	53 -
I. Mở đầu.....	53 -
1. Ý tưởng.....	54 -
2. Mô hình	53 -
II. Bài toán Ngựa đi tuần.....	54 -
1. Phát biểu bài toán.....	54 -
2. Thiết kế thuật toán	55 -
III. Bài toán 8 hậu	57 -
1. Phát biểu bài toán.....	57 -
2. Thiết kế thuật toán	57 -
IV. Bài toán liệt kê các dãy nhị phân độ dài n	59 -

1. Phát biểu bài toán.....	59 -
2. Thiết kế thuật toán	59 -
V. Bài toán liệt kê các hoán vị.....	60 -
1. Phát biểu bài toán.....	60 -
2. Thiết kế thuật toán	60 -
VI. Bài toán liệt kê các tổ hợp	61 -
1. Phát biểu bài toán.....	61 -
2. Thiết kế thuật toán	61 -
VII. Bài toán tìm kiếm đường đi trên đồ thị	61 -
1. Phát biểu bài toán.....	61 -
2. Thuật toán DFS (Depth First Search)	62 -
3. Thuật toán BFS (Breadth First Search)	64 -
Bài tập.....	66 -
Chương 4: PHƯƠNG PHÁP NHÁNH CẬN.....	69 -
I. Mở đầu.....	69 -
1. Ý tưởng.....	69 -
2. Mô hình	69 -
II. Bài toán người du lịch.....	70 -
1. Bài toán	70 -
2. Ý tưởng.....	70 -
3. Thiết kế	71 -
4. Cài đặt.....	73 -
III. Bài toán cái túi xách.....	74 -
1. Bài toán	74 -
2. Ý tưởng.....	74 -
3. Thiết kế thuật toán	75 -
4. Cài đặt.....	78 -
Bài tập.....	79 -
Chương 5: PHƯƠNG PHÁP THAM LAM.....	81 -
I. Mở đầu.....	81 -
1. Ý tưởng.....	81 -
2. Mô hình	81 -
II. Bài toán người du lịch.....	82 -
1. Bài toán	82 -
2. Ý tưởng.....	82 -
3. Thuật toán	82 -
4. Độ phức tạp của thuật toán.....	83 -

5. Cài đặt.....	83 -
III. Thuật toán Dijkstra -Tìm đường đi ngắn nhất trong đồ thị có trọng số	84 -
1. Bài toán	84 -
2. Ý tưởng.....	85 -
3. Mô tả thuật toán	85 -
4. Cài đặt.....	87 -
5. Độ phức tạp của thuật toán.....	90 -
IV. Thuật toán Prim – Tìm cây bao trùm nhỏ nhất	90 -
1. Bài toán	90 -
2. Ý tưởng.....	90 -
3. Mô tả thuật toán	90 -
4. Cài đặt.....	91 -
5. Độ phức tạp thuật toán	93 -
V. Bài toán ghi các bài hát.....	93 -
1. Phát biểu bài toán.....	93 -
2. Thiết kế	93 -
3. Độ phức tạp của thuật toán.....	94 -
4. Cài đặt.....	94 -
VI. Bài toán chiếc túi xách (Knapsack)	95 -
1. Phát biểu bài toán.....	95 -
2. Thiết kế thuật toán	95 -
3. Độ phức tạp của thuật toán.....	96 -
4. Cài đặt.....	96 -
VII. Phương pháp tham lam và Heuristic	97 -
Bài tập.....	98 -
Chương 6 : PHƯƠNG PHÁP QUY HOẠCH ĐỘNG.....	100 -
I. Phương pháp tổng quát	100 -
II. Thuật toán Floyd -Tìm đường đi ngắn nhất giữa các cặp đỉnh.....	100 -
1. Bài toán	100 -
2. Ý tưởng.....	101 -
3. Thiết kế	101 -
4. Cài đặt.....	103 -
5. Độ phức tạp của thuật toán.....	104 -
III. Nhân tổ hợp nhiều ma trận.....	104 -
1. Bài toán	104 -

2. Ý tưởng.....	- 104 -
3. Thiết kế	- 105 -
4. Độ phức tạp của thuật toán.....	- 106 -
5. Cài đặt.....	- 106 -
IV. Cây nhị phân tìm kiếm tối ưu (Optimal Binary Search Tree)	- 107 -
1. Phát biểu bài toán.....	- 108 -
2. Ý tưởng.....	- 108 -
3. Thiết kế thuật toán	- 109 -
4. Độ phức tạp của thuật toán.....	- 110 -
5. Cài đặt.....	- 111 -
V. Dãy chung dài nhất của 2 dãy số.....	- 111 -
1. Bài toán	- 111 -
2. Ý tưởng.....	- 112 -
3. Thuật toán	- 112 -
4. Độ phức tạp của thuật toán.....	- 114 -
5. Cài đặt.....	- 114 -
VI. Bài toán người du lịch	- 115 -
1. Ý tưởng.....	- 116 -
2. Thiết kế thuật toán	- 116 -
3. Độ phức tạp của thuật toán.....	- 118 -
Bài tập.....	- 118 -
PHỤ LỤC.....	- 120 -
TÀI LIỆU THAM KHẢO	- 122 -

LỜI NÓI ĐẦU

Giáo trình “ Thiết kế và đánh giá thuật toán “ có nội dung tiếp sau giáo trình “Cấu trúc dữ liệu và thuật toán 1” và “ Toán cao cấp A4”, trình bày trong 3 tín chỉ lý thuyết và 1 tín chỉ thực hành cho các sinh viên ngành Toán – Tin học và Công nghệ thông tin. Trọng tâm chính của giáo trình :

- Trình bày một số phương pháp thiết kế thuật toán thông dụng.
- Tìm hiểu cơ sở phân tích độ phức tạp của thuật toán.

Nội dung giáo trình gồm 6 chương :

CHƯƠNG 1 : GIỚI THIỆU THIẾT KẾ VÀ ĐÁNH GIÁ THUẬT TOÁN.

Chương này giới thiệu khái niệm trực quan của thuật toán, ngôn ngữ mô tả thuật toán, phân tích thuật toán, cải tiến thuật toán.

CHƯƠNG 2 : PHƯƠNG PHÁP CHIA ĐỂ TRỊ

Chương này trình bày kỹ thuật thiết kế chia để trị, mô hình thủ tục thường sử dụng và các bài toán minh họa như : bài toán MinMax, thuật toán Strassen về nhân ma trận, thuật toán trộn trực tiếp, . . .

CHƯƠNG 3 : PHƯƠNG PHÁP QUAY LUI

Giới thiệu mô hình đệ quy quay lui và các bài toán minh họa như : bài toán “ ngựa đi tuần”, bài toán “ tám hậu “, các bài toán tổ hợp, các thuật toán tìm kiếm trên đồ thị DFS, BFS. . .

CHƯƠNG 4 : PHƯƠNG PHÁP NHÁNH CẬN

Chương này mô tả kỹ thuật đánh giá nhánh cận trong quá trình quay lui để tìm lời giải tối ưu của bài toán. Các bài toán dùng để minh họa như bài toán “ Người du lịch “, bài toán “ chiếc túi xách “.

CHƯƠNG 5 : PHƯƠNG PHÁP THAM LAM

Giới thiệu phương pháp tìm kiếm nhanh lời giải chấp nhận được (và có thể là tối ưu) của bài toán tối ưu. Các bài toán minh họa như : bài toán “ Người du lịch”, thuật toán Dijkstra tìm đường đi ngắn nhất từ một đỉnh đến các đỉnh còn lại của đồ thị, bài toán “ chiếc túi xách “, . .

CHƯƠNG 6 : PHƯƠNG PHÁP QUY HOẠCH ĐỘNG

Chương này mô tả ý tưởng, các thao tác chính sử dụng trong thuật toán quy hoạch động. Các bài toán minh họa như thuật toán Floyd tìm đường đi ngắn nhất giữa các cặp đỉnh của một đơn đồ thị, bài toán nhân tổ hợp các ma trận, cây nhị phân tìm kiếm tối ưu ...

Vì trình độ người biên soạn có hạn nên tập giáo trình không tránh khỏi nhiều khiếm khuyết, chúng tôi rất mong sự góp ý của các bạn đồng nghiệp và sinh viên.

Cuối cùng, chúng tôi cảm ơn sự động viên, giúp đỡ nhiệt thành của các bạn đồng nghiệp trong khoa Toán-Tin học để tập giáo trình này được hoàn thành.

Đà Lạt, ngày 10 tháng 11 năm 2002
TRẦN TUẤN MINH

CHƯƠNG 1 : GIỚI THIỆU THIẾT KẾ, ĐÁNH GIÁ THUẬT TOÁN

Thuật ngữ thuật toán (Algorithm) là từ viết tắt của tên một nhà toán học ở thế kỷ IX : Abu Ja'fa Mohammed ibn Musa al-Khowarizmi . Đầu tiên, thuật toán được hiểu như là các quy tắc thực hiện các phép toán số học với các con số được viết trong hệ thập phân. Cùng với sự phát triển của máy tính , khái niệm thuật toán được hiểu theo nghĩa rộng hơn. Một định nghĩa hình thức về thuật toán được nhà toán học người Anh là Alan Turing đưa ra vào năm 1936 thông qua máy Turing. Có thể nói lý thuyết thuật toán được hình thành từ đó.

Lý thuyết thuật toán quan tâm đến những vấn đề sau :

1. Giải được bằng thuật toán : Lớp bài toán nào giải được bằng thuật toán, lớp bài toán không giải được bằng thuật toán.

2. Tối ưu hóa thuật toán : Thay những thuật toán chưa tốt bằng những thuật toán tốt hơn.

3. Triển khai thuật toán : Xây dựng những ngôn ngữ thực hiện trên máy tính để mã hóa thuật toán.

Hướng nghiên cứu thứ 2 thuộc phạm vi của lĩnh vực phân tích thuật toán : Đánh lượng mức độ phức tạp của thuật toán ; còn hướng thứ ba thường được xếp vào khoa học lập trình.

Chương đầu tiên của giáo trình sẽ giới thiệu thuật toán theo nghĩa trực quan và một số khái niệm mở đầu về phân tích và thiết kế thuật toán.

I. Định nghĩa trực quan về Thuật toán

1. Định nghĩa

Thuật toán là một dãy hữu hạn các thao tác được bố trí theo một trình tự xác định, được đề ra trước, nhằm giải quyết một bài toán nhất định.

- Thao tác , hay còn gọi là tác vụ, phép toán (Operation) hay lệnh (Command), chỉ thị (Instruction)...là một hành động cần được thực hiện bởi cơ chế thực hiện thuật toán.

Mỗi thao tác biến đổi bài toán từ một trạng thái trước (hay trạng thái nhập) sang trạng thái sau (hay trạng thái xuất).Thực tế mỗi thao tác thường sử dụng một số đối tượng trong trạng thái nhập (các đối tượng nhập)và sản sinh ra các đối tượng mới trong trạng thái xuất (các đối tượng xuất). Quan hệ giữa 2 trạng thái xuất và nhập cho thấy tác động của thao tác. Dãy các thao tác của thuật toán nối tiếp nhau nhằm biến đổi bài toán từ trạng thái ban đầu đến trạng thái kết quả.

Mỗi thao tác có thể phân tích thành các thao tác đơn giản hơn.

- Trình tự thực hiện các thao tác phải được xác định rõ ràng trong thuật toán. Cùng một tập hợp thao tác nhưng xếp đặt theo trình tự khác nhau sẽ cho kết quả khác nhau.

2. Các đặc trưng cơ bản của thuật toán

a) Tính xác định

Các thao tác, các đối tượng, phương tiện trong thuật toán phải có ý nghĩa rõ ràng, không được gây nhầm lẫn. Nói cách khác, hai cơ chế hoạt động khác nhau (người hoặc máy...) cùng thực hiện một thuật toán, sử dụng các đối tượng, phương tiện nhập phải cho cùng một kết quả.

b) Tính dừng (hay hữu hạn)

Đòi hỏi thuật toán phải dừng và cho kết quả sau một số hữu hạn các bước.

c) Tính đúng của thuật toán

Thuật toán đúng là thuật toán cho kết quả thỏa mãn đặc tả thuật toán với mọi trường hợp của các đối tượng, phương tiện nhập.

Thuật toán sai khi sai trong (ít nhất) một trường hợp.

d) Tính phổ dụng

Thuật toán để giải một lớp bài toán gồm nhiều bài toán cụ thể, lớp đó được xác định bởi đặc tả. Dĩ nhiên có lớp bài toán chỉ gồm 1 bài. Thuật toán khi đó sẽ không cần sử dụng đối tượng, phương tiện nhập nào cả.

3. Đặc tả thuật toán

Mỗi thuật toán nhằm giải quyết một lớp các bài toán cụ thể.

Mỗi lần thực hiện thuật toán cần phải cung cấp cho cơ chế thực hiện một số đối tượng hay phương tiện cần thiết nào đó. Các đối tượng hay phương tiện này phân biệt bài toán cụ thể trong lớp bài toán mà thuật toán giải quyết.

Làm sao định rõ lớp bài toán mà một thuật toán giải quyết? Đó là đặc tả thuật toán. Đặc tả thuật toán cần chỉ ra các đặc điểm sau:

1. Các đối tượng và phương tiện của thuật toán cần sử dụng (nhập).
2. Điều kiện ràng buộc (nếu có) trên các đối tượng và phương tiện đó.
3. Các sản phẩm, kết quả (xuất).
4. Các yêu cầu trên sản phẩm, kết quả. Thường xuất hiện dưới dạng quan hệ giữa kết quả và các đối tượng, phương tiện sử dụng.



II. Các dạng diễn đạt thuật toán

Thuật toán có thể diễn đạt dưới nhiều hình thức, chẳng hạn dưới dạng lưu đồ, dạng ngôn ngữ tự nhiên, dạng mã giả hoặc một ngôn ngữ lập trình nào đó.

1. Dạng lưu đồ (sơ đồ khối)

Dùng các hình vẽ (có qui ước) để diễn đạt thuật toán .Lưu đồ cho hình ảnh trực quan và tổng thể của thuật toán ,cho nên thường được sử dụng.

2. Dạng ngôn ngữ tự nhiên

Thuật toán có thể trình bày dưới dạng ngôn ngữ tự nhiên theo trình tự các bước thực hiện trong thuật toán .

3. Ngôn ngữ lập trình.

Dùng cấu trúc lệnh, dữ liệu của một ngôn ngữ lập trình nào đó để mô tả.

4. Dạng mã giả

Thuật toán trình bày trong dạng văn bản bằng ngôn ngữ tự nhiên tuy dễ hiểu nhưng khó cài đặt. Dùng một ngữ lập trình nào đó để diễn tả thì phức tạp, khó hiểu. Thông thường thuật toán cũng được trao đổi dưới dạng văn bản - tuy không ràng buộc nhiều vào cú pháp xác định như các ngôn ngữ lập trình, nhưng cũng tuân theo một số quy ước ban đầu - Ta gọi dạng này là mã giả. Tùy theo việc định hướng cài đặt thuật toán theo ngôn ngữ lập trình nào ta diễn đạt thuật toán gần với ngôn ngữ ấy. Trong phần này ta trình bày một số quy ước của ngôn ngữ mã giả trong dạng gần C/C++.

a) Ký tự

- Bộ chữ cái : 26 chữ cái.
- 10 chữ số thập phân.
- Các dấu phép toán số học.
- Các dấu phép toán quan hệ.

b) Các từ : Ghép các ký tự chữ, số, dấu gạch dưới (_).

Các từ sau xem như là các từ khóa :

if, else, case, for, while , do while ...

c) Các phép toán số học và logic

- Các phép toán số học : +, -, *, /, %.
- Các phép toán Logic : &&, ||, ! của C/C++.

d) Biểu thức và thứ tự ưu tiên các phép toán trong biểu thức (Như C/C++).***e) Các câu lệnh******1. Lệnh gán :***

x = Biểu thức;

2. Lệnh ghép (Khối lệnh) :

[

A₁ ;

...

A_n;

}

3. Cấu trúc rẽ nhánh :

if (C) A	if (C) A else B
-------------	--------------------------

Trong đó C là biểu thức logic, A và B là các khối lệnh.

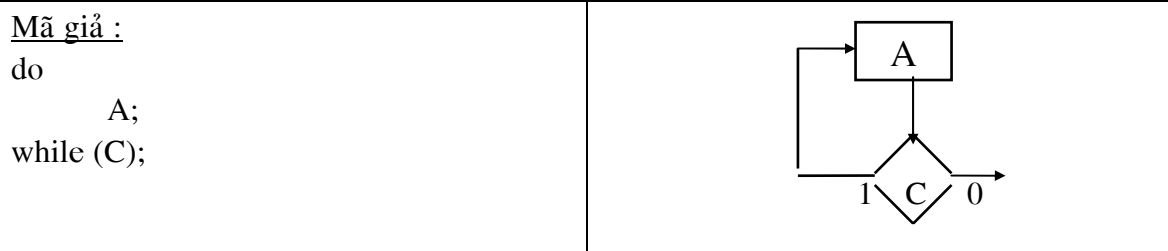
4. Cấu trúc chọn :

<p><u>Mã giả</u> Switch(Bt) Case C1 : A1; Case C2 : A2; Case Cn : An [default : An+1;]</p> <p>Trong đó :</p> <ul style="list-style-type: none"> - bt : Biểu thức nguyên. - Ci là các giá trị nguyên đôi một khác nhau. - Ai là nhóm lệnh. 	
--	--

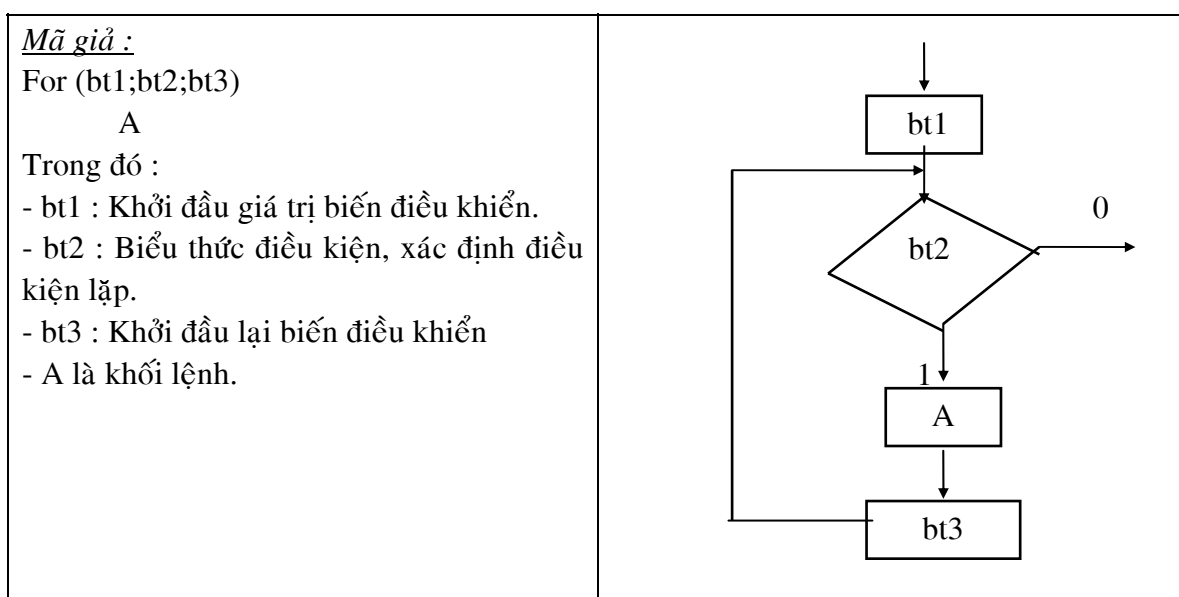
5. Lặp với kiểm tra điều kiện trước (While).

<p><u>Mã giả :</u> While C A;</p>	
---	--

6. Lặp với kiểm tra điều kiện sau (do .. while).



7. Lặp với số lần lặp xác định



8. Câu lệnh vào ra :

 Đọc : scanf(danh_sách_biến);

 Viết : printf(Danh_sách_biến);

9. Câu lệnh bắt đầu và kết thúc :

```
{
    ...
}
```

10. Hàm (Function):

 Type tên_hàm (Danh sách các type và đối)

```
{
    ...
}
```

11. Lời gọi hàm :

 tên_hàm (Danh sách các tham số thực);

12. Câu lệnh return

 return (bt) : Gán giá trị biểu thức bt cho hàm.

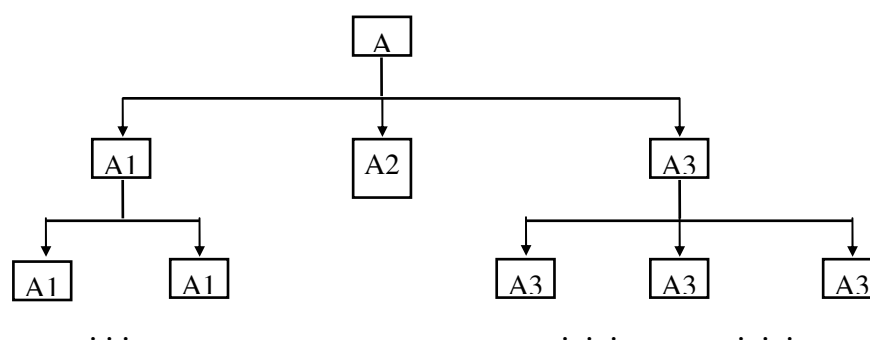
III. Thiết kế thuật toán

Thuật toán được thiết kế một cách có cấu trúc, công cụ chủ yếu là :

1. Modul hóa và thiết kế từ trên xuống (Top-Dow)

Các bài toán giải được trên máy tính ngày càng phức tạp và đa dạng. Các thuật toán giải chúng ngày càng có quy mô lớn đòi hỏi nhiều thời gian và công sức của nhiều người. Tuy nhiên công việc sẽ đơn giản hơn nếu như ta chia bài toán ra thành các bài toán nhỏ. Điều đó cũng có nghĩa là nếu coi bài toán là modul chính thì cần chia thành các modul con. Đến lượt mình các modul con lại phân rã thành các modul con thích hợp...

Như vậy việc tổ chức lời giải thể hiện theo một cấu trúc phân cấp :



Chiến thuật giải bài toán như vậy là “chia để trị”, thể hiện chiến thuật đó ta dùng thiết kế từ trên xuống. Đó là cách nhìn nhận vấn đề một cách tổng quát, đề cập đến các công việc chính, sau đó mới bổ sung dần các chi tiết.

2. Phương pháp làm mịn dần (hay tinh chế từng bước)

Là phương pháp thiết kế phản ánh tinh thần modul hóa và thiết kế từ trên xuống.

Đầu tiên thuật toán được trình bày dưới dạng ngôn ngữ tự nhiên thể hiện ý chính công việc. Các bước sau sẽ chi tiết hóa dần tương ứng với các công việc nhỏ hơn. Đó là các bước làm mịn dần đặc tả thuật toán và hướng về ngôn ngữ lập trình mà ta dự định cài đặt.

Quá trình thiết kế và phát triển thuật toán sẽ thể hiện dần từ ngôn ngữ tự nhiên, sang ngôn ngữ mã giả rồi đến ngôn ngữ lập trình, và đi từ mức “làm cái gì” đến “làm như thế nào”.

Ví dụ :

Bài toán nắn tên .

Một tên có thể có một hay nhiều từ, các từ tách biệt bởi ít nhất 1 dấu cách (khoảng trắng, tab, ..). Từ là một dãy các ký tự khác dấu cách.

Việc nắn tên thực hiện theo các quy cách :

- (i) Khử các dấu cách ở đầu và cuối của tên (cả họ và tên được gọi tắt là tên).
- (ii) Khử bớt các dấu cách ở giữa các từ, chỉ để lại một dấu cách.
- (iii) Các chữ cái đầu từ được viết hoa, ngoài ra mọi chữ cái còn lại được viết thường.

Chương trình được phát thảo bởi :

Mức 0 :

Nấn x thành x theo các quy tắc (i-iii).

Mức 1 :

Do tên được tạo bởi các từ , nên nấn tên thì ta phải nấn các từ. Ta nấn từng từ trong tên cho đến hết các từ. Ý tưởng ở mức 1 được làm mịn hơn như sau :

Khi (còn từ w trong x) ta thực hiện

Nấn lại từ w trong x ;

Đặt một dấu cách nếu cần;

Mức 2 :

Ta chi tiết hơn thao tác : "Đặt một dấu cách nếu cần".

Rõ ràng dấu cách nối chỉ đặt sau mỗi từ, trừ từ cuối cùng. Như vậy sau khi xử lý xong từ cuối thì ta không đặt dấu cách. Vậy ta có thể viết :

Khi (còn từ w trong x) ta thực hiện

Nấn lại từ w trong x ;

Nếu w chưa phải là từ cuối trong x thì

Đặt một dấu cách sau w ;

Mức 3 :

Để xử lý dữ liệu được rõ ràng, tạm thời ta coi tên đích là y và tên nguồn là x .

$y = \text{từ rỗng};$

Khi (còn từ w trong x) ta thực hiện

Nấn lại từ w trong x ;

Ghép w vào sau y ;

Nếu w chưa phải là từ cuối trong x thì

Ghép dấu cách vào sau y ;

Mức 4 :

Ta cụ thể hóa thế nào là 1 từ .

Dễ thấy là một từ w của x là một dãy ký tự không chứa dấu cách và được chặn đầu và cuối bởi dấu cách hoặc từ rỗng.

Có thể nhận dạng được từ w trong x bằng thao tác đơn giản sau đây :

a) Vượt dãy dấu cách để đến đầu từ.

b) Vượt dãy ký tự khác dấu cách để đến hết từ.

Ta chú ý rằng tín hiệu kết thúc của x là ký tự NULL. Ta có thể viết hàm nấn tên như sau :

```
void Nanten(char x[])
```

```
{
```

```
    char y[max];
```

```
    int i;
```

```
    y[0] = '\0';
```

```
    // Vượt dãy dấu cách biên trái
```

```
    i = 0;
```

```

while (x[i] == cach )
    i++;
//Cho kết quả : x[i] là đầu 1 từ hay là x[i] = NULL
while (x[i] != NULL)      // Trường hợp x[i] đầu 1 từ
{
    ghepkt(Hoa(x[i]),y); // Ký tự đầu là Hoa
    i++; //Sang thân từ hoặc rơi vào kết
    while ((x[i] != cach)&& (x[i] != NULL)) // Thân 1 từ
    { // Xử lý thân từ
        ghepkt(Thuong(x[i]),y); // Trong thân từ, KT viết thường
        i++;
    }
    // Xử lý xong 1 từ, tìm đến từ tiếp theo
    while (x[i] == cach)
        i++; // Vượt dấu cách sau 1 từ
    if (x[i] != NULL) // Từ vừa xử lý chưa phải là từ cuối
        ghepkt(cach,y);
}
strepy( y,x);
}

```

Mức 5 :

Ta viết thêm các hàm :

Hoa(char x) : Đổi ký tự thường thành Hoa;

Thuong(char x): Đổi ký tự hoa thành thường.

ghepkt (char ch, char y[]); Ghép ký tự ch vào cuối xâu y, lưu trữ lại vào y.

Nhận xét rằng khoảng cách $d = | 'A' - 'a' |$ ($= 32$) chính là độ lệch bộ chữ hoa đến chữ thường.

Vậy nếu ch là chữ thường thì $ch - d$ sẽ là mã của từ hoa tương ứng, và ngược lại, nếu ch là chữ hoa thì $ch + d$ sẽ là mã của từ thường tương ứng. Từ đó suy ra cách cài đặt các hàm Hoa() và Thuong().

Còn hàm ghép(), Chỉ cần xác định cuối của y, sau đó chép ch vào cuối của y là xong.

3. Một số phương pháp thiết kế

Trên cơ sở lý thuyết máy Turing, ta chia được các bài toán thành 2 lớp không giao nhau : Lớp giải được bằng thuật toán , và lớp không giải được bằng thuật toán.

Đối với lớp các bài toán giải được bằng thuật toán, dựa vào các đặc trưng của quá trình thiết kế của thuật toán, ta có thể chỉ ra một số các phương pháp thiết kế thuật toán cơ bản sau đây :

a) Phương pháp chia để trị. (Divide-and-Conquer method).

Ý tưởng là : Chia dữ liệu thành từng miền đủ nhỏ, giải bài toán trên các miền đã chia rồi tổng hợp kết quả lại .

Chẳng hạn như thuật toán Quicksort.

b) Phương pháp quay lui (BackTracking method).

Tìm kiếm theo ưu tiên.

Đối với mỗi bước thuật toán, ưu tiên theo độ rộng hay chiều sâu để tìm kiếm.

Chẳng hạn thuật toán giải bài toán 8 hậu.

c) Phương pháp tham lam (Greedy Method).

Ý tưởng là : Xác định trật tự xử lý để có lợi nhất, Sắp xếp dữ liệu theo trật tự đó, rồi xử lý dữ liệu theo trật tự đã nêu. Công sức bỏ ra là tìm ra trật tự đó.

Chẳng hạn thuật toán tìm cây bao trùm nhỏ nhất (Shortest spanning Trees).

d) Phương pháp Quy hoạch động (Dynamic Programming method).

Phương pháp quy hoạch động dựa vào một nguyên lý, gọi là nguyên lý tối ưu của Bellman :

“ Nếu lời giải của bài toán là tối ưu thì lời giải của các bài toán con cũng tối ưu ”.

Phương pháp này tổ chức tìm kiếm lời giải theo kiểu từ dưới lên. Xuất phát từ các bài toán con nhỏ và đơn giản nhất, tổ hợp các lời giải của chúng để có lời giải của bài toán con lớn hơn...và cứ như thế cuối cùng được lời giải của bài toán ban đầu.

Chẳng hạn thuật toán “chiếc túi xách” (Knapsack).

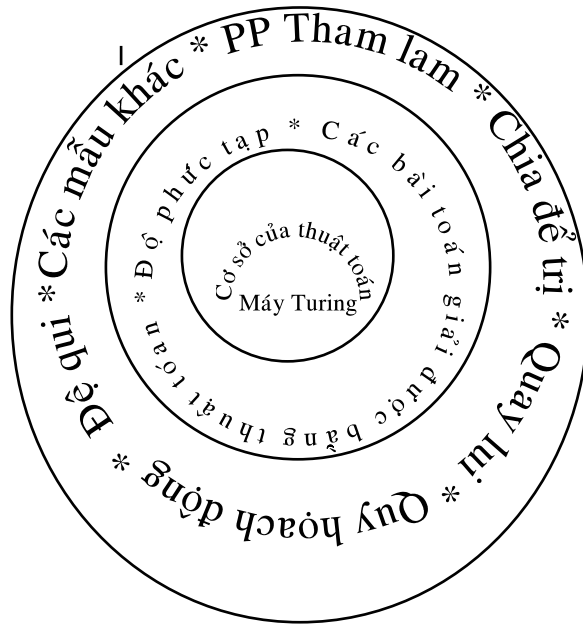
e) Phương pháp nhánh cận (branch-and-bound method).

Ý tưởng là : Trong quá trình tìm kiếm lời giải, ta phân hoạch tập các phương án của bài toán ra thành hai hay nhiều tập con được biểu diễn như là các nút của cây tìm kiếm và cố gắng bằng phép đánh giá cận cho các nút, tìm cách loại bỏ các nhánh của cây mà ta biết chắc không chứa phương án tối ưu.

Chẳng hạn thuật toán giải bài toán người du lịch.

. . .

Ta có thể minh họa bởi hình vẽ sau :



IV. Phân tích thuật toán

Khi xây dựng được thuật toán để giải bài toán thì có hàng loạt vấn đề được đặt ra để phân tích. Thường là các vấn đề sau :

- Yêu cầu về tính đúng đắn của thuật toán, thuật toán có cho lời giải đúng của bài toán hay không ?

- Tính đơn giản của thuật toán. Thường ta mong muốn có được một thuật toán đơn giản, dễ hiểu, dễ lập trình. Đặc biệt là những thuật toán chỉ dùng một vài lần ta cần coi trọng tính chất này, vì công sức và thời gian bỏ ra để xây dựng thuật toán thường lớn hơn rất nhiều so với thời gian thực hiện nó.

- Yêu cầu về không gian : thuật toán được xây dựng có phù hợp với bộ nhớ của máy tính hay không ?

- Yêu cầu về thời gian : Thời gian chạy của thuật toán có nhanh không ? Một bài toán thường có nhiều thuật toán để giải, cho nên yêu cầu một thuật toán dẫn nhanh đến kết quả là một đòi hỏi đương nhiên.

.....

Trong phần này ta quan tâm chủ yếu đến tốc độ của thuật toán. Ta cũng lưu ý rằng thời gian chạy của thuật toán và dung lượng bộ nhớ nhiều khi không cân đối được để có một giải pháp trọn vẹn. Chẳng hạn, thuật toán sắp xếp nội sẽ có thời gian chạy nhanh hơn vì dữ liệu được lưu trữ trong bộ nhớ trong, và do đó không phù hợp trong trường hợp kích thước dữ liệu lớn. Ngược lại, các thuật toán sắp xếp ngoài phù hợp với kích thước dữ liệu lớn vì dữ liệu được lưu trữ chính ở các thiết bị ngoài, nhưng khi đó tốc độ lại chậm hơn.

1. Các bước trong quá trình phân tích đánh giá thời gian chạy của thuật toán

- Bước đầu tiên trong việc phân tích thời gian chạy của thuật toán là quan tâm đến kích thước dữ liệu, sẽ được dùng như dữ liệu nhập của thuật toán và quyết

định phân tích nào là thích hợp. Ta có thể xem thời gian chạy của thuật toán là một hàm theo kích thước của dữ liệu nhập. Nếu gọi n là kích thước của dữ liệu nhập thì thời gian thực hiện T của thuật toán được biểu diễn như một hàm theo n , ký hiệu là $T(n)$.

- Bước thứ hai trong việc phân tích đánh giá thời gian chạy của một thuật toán là nhận ra các thao tác trừu tượng của thuật toán để tách biệt sự phân tích và sự cài đặt. Bởi vì ta biết rằng tốc độ xử lý của máy tính và các bộ dịch của các ngôn ngữ lập trình cấp cao đều ảnh hưởng đến thời gian chạy của thuật toán, nhưng những yếu tố này ảnh hưởng không đồng đều với các loại máy trên đó cài đặt thuật toán, vì vậy không thể dựa vào chúng để đánh giá thời gian chạy của thuật toán. Chẳng hạn ta tách biệt sự xem xét có bao nhiêu phép toán so sánh trong một thuật toán sắp xếp khỏi sự xác định cần bao nhiêu micro giây chạy trên một máy tính cụ thể. Yếu tố thứ nhất được xác định bởi tính chất của thuật toán, còn yếu tố thứ hai được xác định bởi tính năng của máy tính. Điều này cho ta thấy rằng $T(n)$ không thể được biểu diễn bằng giây, phút...được; cách tốt hơn là biểu diễn theo số các chỉ thị trong thuật toán.

Ví dụ :

Xét :

for($i = 1$; $i < n$; $i++$) (1)

for($j = 1$; $j < n$; $j++$) (2)

Ký hiệu : $T(n)$ là thời gian thực hiện câu lệnh (2) :

1	1	2	3	$n-1$
(2)	n	n	n		n

$$T(n) = \underbrace{n + \dots + n}_{(n-1) \text{ lần}} = (n-1)n$$

- Bước thứ ba trong việc phân tích đánh giá thời gian chạy của một thuật toán là sự phân tích về mặt toán học với mục đích tìm ra các giá trị trung bình và trường hợp xấu nhất cho mỗi đại lượng cơ bản. Chẳng hạn, khi sắp xếp một dãy các phần tử, thời gian chạy của thuật toán hiển nhiên còn phụ thuộc vào tính chất của dữ liệu nhập như :

- * Dãy có thứ tự thuận.
- * Dãy có thứ tự ngược.
- * Các số hạng của dãy có thứ tự ngẫu nhiên.

2. Các ký hiệu tiệm cận

a) Ký hiệu O lớn (big – oh) :

Định nghĩa :

Cho hàm $f : \mathbb{N}^* \longrightarrow \mathbb{N}^*$. Ta định nghĩa :

$$O(f(n)) = \{t : \mathbb{N}^* \longrightarrow \mathbb{N}^* \mid \exists c \in \mathbb{R}^+, \exists n_0 \in \mathbb{N}, \forall n \geq n_0, t(n) \leq cf(n)\}$$

$O(f(n))$ gọi là cấp của $f(n)$.

Với $t : \mathbb{N}^* \longrightarrow \mathbb{N}^*$

$$t(n) \in O(f(n)) \Leftrightarrow \exists c \in \mathbb{R}^+, \exists n_0 \in \mathbb{N}, \forall n \geq n_0, t(n) \leq cf(n).$$

Nhận xét :

a) $t(n) \in O(t(n))$

b) $t(n) \in O(f(n)) \Rightarrow \exists c \in \mathbb{R}^+, \forall n \in \mathbb{N}, t(n) \leq cf(n)$.

Các tính chất :

Tính chất 1 :

Với mọi hàm $f : \mathbb{N}^* \longrightarrow \mathbb{N}^*$:

$$f(n) \in O(n) \Rightarrow \begin{cases} \bullet f(n)^2 \in O(n^2) \\ \bullet 2^{f(n)} \in O(2^n) \end{cases}$$

Tính chất 2:

a) $f(n) \in O(g(n))$ và $g(n) \in O(h(n)) \Rightarrow f(n) \in O(h(n))$.

b) $g(n) \in O(h(n)) \Rightarrow O(g(n)) \subseteq O(h(n))$.

Tính chất 3:

a) $O(f(n)) = O(g(n)) \Leftrightarrow g(n) \in O(f(n))$ và $f(n) \in O(g(n))$.

b) $O(f(n)) \subset O(g(n)) \Leftrightarrow f(n) \in O(g(n))$ và $g(n) \notin O(f(n))$.

Tính chất 4:

a) $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = c \neq 0 \Leftrightarrow O(f(n)) = O(g(n))$

b) $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = 0 \Rightarrow O(f(n)) \subset O(g(n)) = O(g(n) \pm f(n))$

Ví dụ :

- Hàm $f(n) = 2n^5 + 3n^3 + 6n^2 + 2$ có cấp $O(n^5)$ vì :

$$\lim_{n \rightarrow \infty} \frac{2n^5 + 3n^3 + 6n^2 + 2}{n^5} = 2 \neq 0.$$

- Hàm $f(n) = 2^n$ là $O(n!)$ vì :

$$\lim_{n \rightarrow \infty} \frac{2^n}{n!} = 0.$$

- Hàm $2^{n+1} \in O(2^n)$.

b) Ký hiệu Ω :

Ký hiệu này dùng để chỉ chặn dưới của thời gian chạy của thuật toán

Ta định nghĩa :

$$\Omega(f(n)) = \{t : \mathbb{N} \longrightarrow \mathbb{N}^* \mid \exists c \in \mathbb{R}^+, \exists n_0 \in \mathbb{N}, \forall n \geq n_0, t(n) \geq cf(n)\}$$

Tính chất 6:

Cho $f, g : \mathbb{N} \longrightarrow \mathbb{N}^*$, Ta có :

$$f(n) \in O(g(n)) \Leftrightarrow g(n) \in \Omega(f(n)).$$

c) Ký hiệu θ :

Định nghĩa :

$$\theta(f(n)) = O(f(n)) \cap \Omega(f(n)).$$

Tính chất 7:

$$f(n) \in \theta(g(n)) \Leftrightarrow \exists c, d \in \mathbb{R}^+, \exists n_0 \in \mathbb{N}, \forall n \geq n_0, cg(n) \leq f(n) \leq dg(n).$$

3. Một số lớp các thuật toán

Hầu hết các thuật toán được giới thiệu trong giáo trình này tiệp cận tới một trong các hàm sau :

(1) 1 : Nếu tất cả các chỉ thị của chương trình đều được thực hiện chỉ một vài lần và ta nói thời gian chạy của nó là hằng số.

(2) Logn : Khi thời gian chạy của chương trình là Logarit. Thời gian chạy thuộc loại này xuất hiện trong các chương trình mà giải 1 bài toán lớn bằng cách chuyển nó thành 1 bài toán nhỏ hơn, bằng cách cắt bỏ kích thước một hằng số nào đó. Cơ sở Logarit có thể làm thay đổi hằng số đó nhưng không nhiều.

(3) n : Khi thời gian chạy của chương trình là tuyến tính.

(4) nLogn : Thời gian chạy thuộc loại này xuất hiện trong các chương trình mà giải 1 bài toán lớn bằng cách chuyển nó thành các bài toán nhỏ hơn, kế đến giải quyết chúng 1 cách độc lập, sau đó tổ hợp các lời giải.

(5) n^2 : Thời gian chạy của thuật toán là bậc 2, thường là xử lý các cặp phần tử dữ liệu (có thể là 2 vòng lặp lồng nhau). Trường hợp này chỉ có ý nghĩa thực tế khi bài toán nhỏ.

(6) n^3 : Một thuật toán xử lý bộ ba các phần tử dữ liệu (có thể là 3 vòng lặp lồng nhau) có thời gian chạy bậc 3. Trường hợp này chỉ có ý nghĩa thực tế khi bài toán nhỏ.

(7) 2^n :

Sau đây là các giá trị xấp xỉ của các hàm trên :

n \ Hàm	n	lg n	Nlgn	n^2	n^3	2^n
1	1	0	0	1	1	2
2	2	1	2	4	8	4
4	n	2	8	16	64	16
8	8	3	24	64	512	256
16	16	4	64	256	4096	65536
32	32	5	160	1024	32768	2.147.483.648

Để thấy rằng :

$$O(1) \subset O(\lg n) \subset O(n) \subset O(n \lg n) \subset O(n^2) \subset O(n^3) \subset O(2^n).$$

Các hàm loại : 2^n , $n!$, n^n thường được gọi là các hàm loại mũ. thuật toán với thời gian chạy có cấp hàm loại mũ thì tốc độ rất chậm.

Các hàm loại : n^3 , n^2 , $n \log_2 n$, n , $\log_2 n$ thường được gọi là các hàm loại đa thức. Thuật toán với thời gian chạy có cấp hàm đa thức thường chấp nhận được.

Ghi chú :

Các hằng số bị bỏ qua trong biểu thức đánh giá độ phức tạp của thuật toán có thể có ý nghĩa quan trọng trong ứng dụng cụ thể. Giả sử thuật toán 1 đòi hỏi thời gian là $C_1 n$, còn thuật toán 2 đòi hỏi thời gian là $C_2 n^2$. Dĩ nhiên là với n đủ lớn thì thuật toán 1 nhanh hơn thuật toán 2. Nhưng với n nhỏ thì có thể thuật toán 1 nhanh hơn thuật toán 2. Chẳng hạn, với $C_1 = 200$, $C_2 = 10$, và với $n = 5$, thì thuật toán 1 đòi hỏi thời gian 1000, trong khi đó thuật toán 2 chỉ có 250.

Ví dụ :

Thuật toán Chọn trực tiếp (Straight Selection) : SSS

Sắp xếp tăng dần dãy các khóa : $x[1], x[2], \dots, x[n]$.

Ý tưởng :

- Bước i chọn phần tử nhỏ nhất của dãy $x[i], x[i+1], \dots, x[n]$, đổi chỗ phần tử nhỏ nhất này cho $x[i]$.

- Lặp thao tác này với $i = 1..n-1$.

Thuật toán :

1	for (i = 1; i <= n-1; i++)	
	{	
2	k = i;	Khởi động chỉ số của giá trị nhỏ nhất : (k = i)
3	a = x[i];	Lấy ra giá trị của phần tử thứ i
4	for (j = i+1; j <= n; j++)	Tìm phần tử nhỏ nhất trong mảng x[i]...x[n]
5	if (x[j] < a)	
	{	
6	a = x[j];	
7	k = j;	Giữ vị trí của phần tử nhỏ nhất
	}	a là giá trị nhỏ nhất (Khi đó : x[k] = a)
8	x[k] = x[i];	Đổi vị trí của phần tử nhỏ nhất
9	x[i] = a;	Cho phần tử a vị trí thứ i.
	}	

Độ phức tạp thuật toán:

Lệnh (1) thực hiện n lần, (Lần n để thoát khỏi for).

Mỗi lệnh (2), (3), (8), (9) thực hiện $n-1$ lần.

Lệnh (4) thực hiện $n + (n-1) + \dots + 2 = \frac{n(n+1)}{2} - 1$ lần.

Lệnh (5) thực hiện $A = (n-1) + (n-2) + \dots + 1 = \frac{n(n-1)}{2}$ lần. // So sánh

- Xét trường hợp xấu nhất :

Tức là lệnh (5) luôn thỏa điều kiện, tương ứng dãy có thứ tự ngược lại,

Mỗi lệnh (6), (7) thực hiện $\frac{n(n-1)}{2}$ lần.

Do đó :

$$\begin{aligned} T(n) &= n + 4(n-1) + n(n+1)/2 - 1 + 3n(n-1)/2 \\ &= 2n^2 + 4n - 5 \leq 6n^2. \\ T(n) &\in \theta(n^2) \end{aligned}$$

- Xét trường hợp tốt nhất

Tức là lệnh (5) luôn không thỏa điều kiện, tương ứng dãy có thứ tự thuận, (6) và (7) không thực hiện lần nào. Ta có :

$$T(n) = n^2 + 5n - 5 \in \theta(n^2).$$

4. Phân tích thuật toán đệ qui.

Phần lớn các thuật toán đều dựa trên sự phân rã đệ qui một bài toán lớn thành các bài toán nhỏ, rồi dùng lời giải các bài toán nhỏ để giải bài toán ban đầu. Thời gian chạy của thuật toán như thế được xác định bởi kích thước và số lượng các

bài toán con và giá phải trả của sự phân rã. Nên các thuật toán đệ qui có thời gian chạy phụ thuộc vào thời gian chạy cho các dữ liệu nhập có kích thước nhỏ hơn, điều này được diễn dịch thành một công thức toán học gọi là công thức truy hồi. Do đó, để tính độ phức tạp của thuật toán, ta thường phải giải các phương trình truy hồi. Có nhiều cách giải, ta nghiên cứu các cách thường dùng sau.

A. Phương pháp thay thế:

Dựa vào dạng truy hồi của phương trình để tính độ phức tạp của thuật toán dựa vào các kích thước dữ liệu nhỏ hơn.

Một số công thức thường gặp sau đây được giải bằng phương pháp thay thế :
Công thức 1:

$$T_N = \begin{cases} T_{N-1} + N; N \geq 2 \\ 1; N = 1 \end{cases}$$

Công thức này thường dùng cho các chương trình đệ qui mà có vòng lặp duyệt qua dữ liệu nhập để bỏ bớt 1 phần tử.

$$\begin{aligned} T_N &= T_{N-1} + N \\ &= T_{N-2} + (N-1) + N \\ &= T_{N-3} + (N-2) + (N-1) + N = \dots \\ &= 1 + 2 + \dots + N \\ &= \frac{N(N+1)}{2}. \end{aligned}$$

Công thức 2:

$$T_N = \begin{cases} \frac{T_N}{2} + 1; N \geq 2 \\ 0; N = 1 \end{cases}$$

Công thức này thường dùng cho các thuật toán đệ qui mà tại mỗi bước chia dữ liệu nhập thành 2 phần.

Giả sử $N = 2^n$, có :

$$T_{2^n} = T_{2^{n-1}} + 1 = T_{2^{n-2}} + 2 = \dots = n.$$

Suy ra : $T_N = \log_2 N$.

Công thức 3:

$$T_N = \begin{cases} T_N + N; N \geq 2 \\ 0; N = 1 \end{cases}$$

Công thức này thường dùng cho các thuật toán đệ qui mà tại mỗi bước thực hiện, chia đôi dữ liệu nhập nhưng có kiểm tra mỗi phần tử của dữ liệu nhập.

$$T_N = N + \frac{N}{2} + \frac{N}{4} + \frac{N}{8} \dots \cong 2N.$$

Công thức 4:

$$T_N = \begin{cases} 2T_N + 1; N \geq 2 \\ 0; N = 1 \end{cases}$$

Công thức này thường dùng cho các thuật toán theo phương pháp chia để trị.

$$\begin{aligned}T_{2^n} &= 2T_{2^{n-1}} + 2^n \\ \frac{T_{2^n}}{2^n} &= \frac{T_{2^{n-1}}}{2^{n-1}} + 1 = \frac{T_{2^{n-2}}}{2^{n-2}} + 1 + 1 = \dots = n \\ \Rightarrow T_{2^n} &= n2^n \\ \Rightarrow T_N &= N \log_2 N\end{aligned}$$

B. Dùng phương trình đặc trưng để giải phương trình truy hồi :

B1) Phương trình truy hồi tuyến tính thuần nhất với các hệ số không đổi :

Xét phương trình dạng :

$$a_0 t_n + a_1 t_{n-1} + \dots + a_k t_{n-k} = 0 \quad (1)$$

Trong đó các $t_i, i = n, n-1, \dots, n-k$ là các ẩn số.

Đặt $t_n = X^n$, ta đưa (1) về dạng :

$$X^{n-k} (a_0 X^k + a_1 X^{k-1} + \dots + a_{k-1} X + a_k) = 0 \quad (2)$$

$$\Leftrightarrow \begin{cases} X^{n-k} = 0 \\ a_0 X^k + a_1 X^{k-1} + \dots + a_{k-1} X + a_k = 0 \end{cases}$$

$X = 0$ hiển nhiên là nghiệm của (2), nhưng ta quan tâm đến nghiệm của phương trình :

$$p(X) = a_0 X^k + a_1 X^{k-1} + \dots + a_{k-1} X + a_k = 0 \quad (3)$$

Phương trình (3) gọi là phương trình đặc trưng bậc k của phương trình truy hồi (1).

TH1 : Tất cả các nghiệm của (3) đều là nghiệm đơn.

Giả sử rằng X_1, X_2, \dots, X_k là các nghiệm đơn của (3), thì ta có thể kiểm tra

được $t_n = \sum_{i=1}^k c_i X_i^n$, với c_1, c_2, \dots, c_k là các hằng xác định từ k điều kiện ban đầu

là nghiệm của (3).

TH2 : Phương trình (3) có nghiệm bội.

- Giả sử u là nghiệm kép của (3).

Với $n > k$, xét đa thức p bậc n :

$$q(X) = a_0 n X^n + a_1 (n-1) X^{n-1} + \dots + a_k (n-k) X^{n-k} = X (X^{n-k} p(X))'$$

Vì u là nghiệm kép của (3), nên tồn tại đa thức r thỏa :

$$p(X) = (X - u)^2 r(X)$$

Khi đó : $q(X) = X [X^{n-k} (X - u)^2 r(X)]' = 0$

Do đó : $a_0 n u^n + a_1 (n-1) u^{n-1} + \dots + a_k (n-k) u^{n-k} = 0$

Tức là : $t_n = n u^n$ cũng là nghiệm của (1).

- Tổng quát, nếu u là nghiệm bội m của (3) thì :
 $t_n = u^n, nu^n, n^2u^n, \dots, n^{m-1}u^n$ cũng là nghiệm của (1)

Khi đó tổ hợp tuyến tính của các nghiệm này và các nghiệm khác của phương trình đặc trưng (3) sẽ là nghiệm của (1).

Ví dụ 1 :

$$\begin{cases} T(n) - 6T(n-1) + 8T(n-2) = 0 \\ T(0) = 1 \\ T(1) = 2 \end{cases}$$

Xét phương trình : $T(n) - 6T(n-1) + 8T(n-2) = 0$

Đặt : $X^n = T(n)$

Ta có : $X^n - 6X^{n-1} + 8X^{n-2} = 0$

Phương trình đặc trưng : $X^2 - 6X + 8 = 0$

Có các nghiệm : $X_1 = 2, X_2 = 4$

Vậy : $T(n) = c_1X_1^n + c_2X_2^n$

Do :

$$T(0) = 1 \Rightarrow c_1X_1^0 + c_2X_2^0 = 1 \Rightarrow c_1 + c_2 = 1$$

$$T(1) = 2 \Rightarrow c_1X_1 + c_2X_2 = 2 \Rightarrow 2c_1 + 4c_2 = 2$$

$$\text{Vậy có : } \begin{cases} c_1 + c_2 = 1 \\ 2c_1 + 4c_2 = 2 \end{cases} \Rightarrow c_1 = 1, c_2 = 0.$$

Ví dụ 2 :

$$T(n) = \begin{cases} 5T(n-1) - 8T(n-2) + 4T(n-3); n \geq 3 \\ 0; n = 0 \\ 1; n = 1 \\ 2; n = 2 \end{cases}$$

Ta có phương trình :

$$T(n) - 5T(n-1) + 8T(n-2) - 4T(n-3) = 0$$

Phương trình đặc trưng tương ứng là :

$$X^3 - 5X^2 + 8X - 4 = 0$$

$$\Leftrightarrow (X-1)(X-2)^2 = 0$$

Vậy ta có các nghiệm của phương trình đặc trưng : 1 (đơn), 2 (kép)

Nên nghiệm chung là : $T(n) = c_11^n + c_22^n + c_3n2^n$

Dựa vào các điều kiện đầu, ta có :

$$\begin{cases} c_1 + c_2 = 0 \\ c_1 + 2c_2 + 2c_3 = 1 \\ c_1 + 4c_2 + 8c_3 = 2 \end{cases}$$

$$\text{Suy ra : } c_1 = -2; c_2 = 2; c_3 = -\frac{1}{2}$$

$$\text{Vậy : } T(n) = 2^{n+1} - n2^{n-1} - 2$$

B2) Phương trình truy hồi tuyến tính không thuần nhất với các hệ số không đổi :

$$\text{Phương trình dạng : } a_0 t_n + a_1 t_{n-1} + \dots + a_k t_{n-k} = b^n p(n)$$

Với b là hằng số, p là đa thức bậc d theo n .

Biến đổi về dạng thuần nhất.

Ví dụ :

$$t_n - 2t_{n-1} = 3^n$$

$$\text{Ta có : } \begin{cases} (2) : t_{n+1} - 2t_n = 3^{n+1}; \\ (3) : 3t_n - 6t_{n-1} = 3^{n+1}; \text{ do nhân 2 vế của (1) cho 3} \end{cases}$$

Lấy (2) – (3), được dạng thuần nhất :

$$t_{n+1} - 5t_n + 6t_{n-1} = 0$$

5. Các phép toán trên các ký hiệu tiệm cận

a) Phép toán cộng :

$$\theta(f(n)) + \theta(g(n)) = \text{Max}(\theta(f(n)), \theta(g(n)))$$

Nhận xét :

$$- \theta(f(n)) + \theta(g(n)) = \theta(f(n) + g(n)) = \theta(\text{Max}(f(n), g(n)))$$

- Đặc biệt, Nếu c là hằng số, thì :

$$\theta(cf(n)) = \theta(f(n))$$

b) Phép toán nhân :

$$\theta(f(n))\theta(g(n)) = \theta(f(n)g(n))$$

Đặc biệt :

$$\theta(f(n)^2) = (\theta(f(n)))^2$$

c) Phép toán tích cực :

Đó là lệnh trong thuật toán mà thời gian thực hiện nó không ít hơn thời gian thực hiện các lệnh khác. Khi đánh giá thời gian thực hiện của thuật toán, ta chỉ cần quan tâm đến các bước thực hiện của phép toán này.

Ví dụ : Sắp tăng dần các phần tử của dãy số x .

Dùng thuật toán chèn trực tiếp *SIS* (*straight insertion Sort*):

Ý tưởng :

Ở bước i , giả sử dãy : $x[1], \dots, x[i]$ đã có thứ tự. Tìm vị trí thích hợp của phần tử $x[i+1]$ để chèn nó vào dãy $x[1], \dots, x[i]$, kết quả là ta có dãy $x[1], \dots, x[i+1]$ có thứ tự. Thực hiện thao tác trên với $i = 1, 2, \dots, n-1$.

Thuật toán :

	for (i = 1; i <= n-1; i++)	
	{	
	a = x[i+1];	biến tạm a nhận giá trị của x[i+1]
	x[0] = a;	
	j = i;	Chuẩn bị cho a tiến về trái (khởi động j
	While (a < x[j])	a còn < x[j], a còn tiến về trái
	{	
	x[j+1] = x[j];	dời giá trị về phải

	$j = j-1;$	Chuẩn bị cho a tiến tiếp về trái
	}	$a \geq x[j]$
	$x[j+1] = a;$	Chèn $x[i+1]$ vào vị trí thích hợp - là sau $x[j]$.
	}	

Có thể xem phép toán tích cực ở đây là : $a < x[j]$;

Trong trường hợp xấu nhất, tương ứng dãy giảm dần. Số lần thực hiện của phép toán này là :

$$2 + \dots + n = \frac{n(n+1)}{2} - 1 .$$

$$\text{Vậy : } T(n) \in \theta(n^2).$$

Trong trường hợp tốt nhất, tương ứng dãy tăng dần. Số lần thực hiện của phép toán này là : $n - 1$

$$\text{Vậy : } T(n) \in \theta(n).$$

6. Phân tích trường hợp trung bình

Ta biết rằng thời gian thực hiện thuật toán không phải chỉ phụ thuộc vào kích thước dữ liệu mà còn phụ thuộc vào tình trạng dữ liệu nhập nữa. Chẳng hạn, khi xếp tăng dần một dãy các số nguyên, thì các số đã có sẵn thứ tự tăng dần, hoặc ngược lại, hoặc ngẫu nhiên.

Phần trên ta đã xét trong những khả năng tốt nhất hoặc xấu nhất của thuật toán .

Bây giờ ta phân tích trong trường hợp ngẫu nhiên, tức là đánh giá độ phức tạp trung bình thời gian thực hiện thuật toán .

Việc đánh giá trung bình thường khó và phức tạp đòi hỏi những công cụ toán học tinh vi, hơn nữa việc tính trung bình có thể có nhiều cách quan niệm khác nhau. Ở đây, việc đánh giá độ phức tạp thuật toán trong trường hợp trung bình ta dựa trên cơ sở lý thuyết xác suất (rời rạc) .

Ví dụ :

Xét thuật toán chèn trực tiếp .

Xét bước thứ i của vòng lặp. Danh sách có i phần tử đã được sắp. Phần tử $x[i+1]$ có thể chèn vào $i-1$ khe giữa các $x[j]$, $j \in \{1, i\}$ và thêm 2 vị trí đầu cuối nữa (Chèn trước $x[1]$ và chèn sau $x[i]$), tổng cộng là có $i+1$ vị trí có thể chèn $x[i+1]$.

Giả sử khả năng của mỗi vị trí được $x[i+1]$ chèn vào là đồng đều. Tức là xác suất của mỗi vị trí được $x[i+1]$ chèn vào là $\frac{1}{i+1}$.

Gọi X_i là biến ngẫu nhiên chỉ số lượng các phép toán so sánh cần thiết để $x[i+1]$ chèn vào vị trí thích hợp của nó trong mỗi bước. Ta có :

	X_1	X_2		X_i	
$i+1$	i		3	2	1
Vị trí (Từ phải sang trái)					
1	i	...	3	2	1
$\frac{1}{i+1}$	$\frac{1}{i+1}$	$\frac{1}{i+1}$...	$\frac{1}{i+1}$	$\frac{1}{i+1}$
X_i					
p					

$$\begin{aligned}
 E(X_i) &= 1 \cdot \frac{1}{i+1} + 2 \cdot \frac{1}{i+1} + 3 \cdot \frac{1}{i+1} + \dots + i \cdot \frac{1}{i+1} + \frac{i}{i+1} \\
 &= \frac{1}{i+1} (1 + 2 + \dots + (i-1) + i + i) \\
 &= \frac{1}{i+1} \cdot \frac{i(i+1)}{2} + \frac{i}{i+1} \\
 &= \frac{i}{2} + \frac{i}{i+1}
 \end{aligned}$$

Nếu gọi Y là biến ngẫu nhiên xác định bởi tổng các so sánh trong sắp xếp thì :

$$Y = X_1 + X_2 + \dots + X_{n-1}$$

Ta có :

$$\begin{aligned}
 E(Y) &= \sum_{i=1}^{n-1} E(X_i) = \sum_{i=1}^{n-1} \frac{i}{2} + \sum_{i=1}^{n-1} \frac{i}{i+1} = \frac{1}{2} \sum_{i=1}^{n-1} i + \sum_{i=1}^{n-1} \left(1 - \frac{1}{i+1}\right) \\
 &= \frac{1}{2} \cdot \frac{n(n-1)}{2} + n - \sum_{i=1}^{n-1} \frac{1}{i+1} \\
 &= \frac{n^2}{4} + \frac{3n}{4} - H_n
 \end{aligned}$$

Trong đó : $H_n = \sum_{i=1}^n \frac{1}{i} \leq \lg n + 1$; (H_n là số Harmonic bậc n .)

Nên giá trị trung bình của các phép toán so sánh trong thuật toán tương đương với $\frac{n^2}{4}$, Vậy độ phức tạp trung bình của thuật toán là $\theta(n^2)$.

V. Tối ưu thuật toán

Tiến trình tổng quát của việc tạo ra các sửa đổi ngày càng tiến bộ hơn cho một thuật toán để sinh ra một phiên bản khác chạy nhanh hơn được gọi là tối ưu thuật toán. Khi tối ưu một thuật toán ta thường dựa vào một nguyên lý, đó là nguyên lý Profile : “Tìm điểm mất thời gian nhiều nhất của thuật toán”.

Một số kỹ thuật sau thường dùng để tối ưu thuật toán :

1. Kỹ thuật tối ưu các vòng lặp

Đây là điểm quan tâm đầu tiên khi cải tiến thuật toán, vì vòng lặp là câu lệnh thường làm tăng độ phức tạp của thuật toán. Việc cải tiến tập trung vào :

- Cố gắng giảm các vòng lặp lồng nhau.
- Tăng số lệnh thực hiện trong một bước lặp để giảm số lượng các bước lặp.
- Tách các lệnh không phụ thuộc vào chỉ số lặp ra khỏi vòng lặp.

...

Ví dụ 1:

Xét thuật toán tính giá trị của e^x theo công thức gần đúng :

$$\sum_{i \geq 0} \frac{x^i}{i!} = 1 + x + \frac{x^2}{2!} + \dots + \frac{x^n}{n!} + \dots$$

Thuật toán 1 : { Tính từng số hạng rồi cộng lại. }

Input x, n

Output $S = \sum_{i=1}^n \frac{x^i}{i!}.$

Mô tả :

```

S = 1;
for( i = 1; i <= n; i++)
{
    p = 1;
    for( j = 1; j <= i ; j++)
        p = p * x / j;
    S = S + p;
}

```

Ta có thể coi phép toán tích cực ở đây là phép $p := p * x / j$.

Ta thấy nó thực hiện được : $1 + 2 + \dots + n = \frac{n(n+1)}{2}$ lần.

Nên $T(n) \in \theta(n^2)$.

Thuật toán 2 :

Ta xem xét vòng lặp trong có cần thiết hay không ?

Nhận xét rằng vòng lặp trong được dùng để tính $\frac{x^i}{i!}$, nhưng mỗi số hạng trong tổng, số hạng sau có thể được tính dựa vào số hạng trước :

$$\begin{aligned} \frac{x^2}{2!} &= \frac{x}{1!} \cdot \frac{x}{2}, \\ \dots; \\ \frac{x^n}{n!} &= \frac{x^{n-1}}{(n-1)!} \cdot \frac{x}{n} \end{aligned}$$

Nên vòng lặp trong có thể bỏ đi vì có thể tính $\frac{x^i}{i!}$ theo công thức trên. Vậy thuật toán có thể viết lại như sau :

```

S = 1;

```

```

p = 1;
for( i = 1; i <= n; i++)
{
    p = p*x/i; //  $\frac{x^i}{i!}$ 
    S = S + p;
}

```

Chẳng hạn có thể coi phép toán tích cực ở đây là phép $p := p*x/i$.

Do đó : $T(n) \in \theta(n)$.

Ví dụ 2 :

Ta xét thuật toán nhân 2 ma trận vuông cấp n.

TT1 :

Input $a, b \in Mat_N(n)$

Output $c \in Mat_N(n)$

Nhan1(a,b,c) \equiv

```

for(i=1; i<=n; i++)
for(j=1; j<=n; j++)
{
    c[i][j] = 0;
    for (k=1; k<=n; k++)
        c[i][j] += a[i][k]*b[k][j];
}

```

Theo TT1, mỗi lần ta chỉ tính 1 phần tử $c[i][j]$ trong vòng lặp theo k. Giờ ta cải tiến, trong trường hợp n chẵn, tính một lần 4 giá trị :

$c[i][j]$	$c[i][j+1]$	$c[i][j]$	$c[i][jj]$
$c[i+1][j]$	$c[i+1][j+1]$	$c[ii][j]$	$c[ii][jj]$

TT2 :

Nhan2(a,b,c) \equiv

```

i = 1;
while (i < n)
{
    ii = i+1;
    j = 1;
    while (j < n)
    {
        jj = j+1;
        c[i][j] = 0;
        c[i][jj] = 0;
    }
}

```

```

        c[ii][j] = 0;
        c[ii][jj] = 0;
        k = 1;
        while (k < n)
        {
            kk = k+1;
            c[i][j] = c[i][j] + a[i][k]*b[k][j] + a[i][kk]*b[kk][j];
            c[i][jj] = c[i][jj]+a[i][k]*b[k][jj] + a[i][kk]*b[kk][jj];
            c[ii][j] = c[ii][j]+a[ii][k]*b[k][j] + a[ii][kk]*b[kk][j];
            c[ii][jj] = c[ii][jj]+a[ii][k]*b[k][jj] +
a[ii][kk]*b[kk][jj];
            k = k+2;
        }
        j = j + 2;
    }
    i = i + 2;
}
}

```

Giả thiết n chẵn vì ta dùng bước nhảy 2

2. Tối ưu việc rẽ nhánh

- Đối với biểu thức logic kết hợp bằng phép toán $\&\&$, nên viết theo thứ tự xác suất sai giảm dần :

$$A_1 \&\& A_2 \&\& \dots \&\& A_n$$

Xác suất sai của các A_i giảm dần .

- Đối với biểu thức logic kết hợp bằng phép toán $\|\|$, nên viết theo thứ tự xác suất đúng giảm dần :

$$A_1 \|\| A_2 \|\| \dots \|\| A_n$$

Xác suất đúng của các A_i giảm dần :

...

BÀI TẬP

Bài 1 :

Xác định độ phức tạp của các thuật toán sau :

```

r = m%n;
while(r)
{
    m = n;
    n = r;
    r = m%n;
}
return n

```

```

gt (n) ≡
    if (n == 0 || n == 1)
        return 1;
    else
        return (n * gt(n-1)) ;

```

```

Fib(n) ≡
if ( n < 2 )
    return 2;
else
    return Fib(n-1)+Fib(n-2);

```

```

Fibo(n) ≡
i = 1; j = 0;
for( k = 1 → n)
{
    j = i + j;
    i = j - i;
}
return j;

```

Bài 2 :

Tính độ phức tạp của các thuật toán sau trong trường hợp tốt nhất, xấu nhất :

<pre> for (i = 1; i <= n/2; i++) { Min = a[i]; Csmín = i ; Max = a[n-i+1]; Csmáx = n-i+1; for (j = i ;j <= n-i+1; j++) { if (a[j] < Min) { Min = a[j]; csmin = j ; } if (a[j] > Max) { Max = a[j]; csmax = j; } } } </pre>	<pre> if (csmin == n-i+1) { Hoán vị a[i] và a[csmin]; if (csmax != i) Hoán vị a[csmax] và a[n- I+1]; } else { Hoán vị a[csmax] và a[n- I+1]; Hoán vị a[i] và a[csmin]; } } </pre>
--	---

Bài 3 :

Tính thời gian thực hiện trung bình của các phép toán so sánh trong các thuật toán :

1. Đổi chỗ trực tiếp.
2. Chọn trực tiếp.

Bài 4 :

Xác định $T(n)$, với :

$$1. \begin{cases} T(n) - 3T(n-1) - 4T(n-2) = 0, n > 1 \\ T(0) = 1 \\ T(1) = 2 \end{cases}$$

$$2. \begin{cases} T(n) = n + 4T\left(\frac{n}{2}\right), n > 1 \\ T(1) = 1 \end{cases}$$

$$3. \begin{cases} T(n) = T(n-1) + T(n-2), n > 1 \\ T(0) = 1 \\ T(1) = 1 \end{cases}$$

$$4. \begin{cases} T(n) = 2T(n-1) + 1; n > 1 \\ T(0) = 0 \\ T(1) = 1 \end{cases}$$

Bài 5 :

Cải tiến thuật toán chèn trực tiếp bằng cách : Dùng phương pháp tìm kiếm nhị phân để xác định vị trí cần chèn của a_i trong dãy con đã có thứ tự a_1, \dots, a_{i-1} .

Thuật toán cải tiến gọi là chèn nhị phân. Hãy thiết kế, cài đặt và đánh giá độ phức tạp thời gian của thuật toán.

Bài 6 :

Tìm các ví dụ về các thuật toán mà khi cải tiến (bằng cách nào đó), số lần thực hiện của thuật toán giảm đáng kể (về độ phức tạp tiệm cận, hoặc về tỉ lệ ...)

CHƯƠNG 2 : PHƯƠNG PHÁP CHIA ĐỂ TRỊ

(Divide - and - conquer)

I. Mở đầu

1. Ý tưởng

Có lẽ quan trọng và áp dụng rộng rãi nhất là kỹ thuật thiết kế “Chia để trị”. Nó phân rã bài toán kích thước n thành các bài toán con nhỏ hơn mà việc tìm lời giải của chúng là cùng một cách. Lời giải của bài toán đã cho được xây dựng từ lời giải của các bài toán con này.

Ta có thể nói vắn tắt ý tưởng chính của phương pháp này là : chia dữ liệu thành từng miền đủ nhỏ, giải bài toán trên các miền đã chia rồi tổng hợp kết quả lại.

2. Mô hình

Nếu gọi $D\&C(\mathcal{R})$ - Với \mathcal{R} là miền dữ liệu - là hàm thể hiện cách giải bài toán theo phương pháp chia để trị thì ta có thể viết :

```
void D&C( $\mathcal{R}$ )
{
    If ( $\mathcal{R}$  đủ nhỏ)
        giải bài toán;
    Else
    {
        Chia  $\mathcal{R}$  thành  $\mathcal{R}_1, \dots, \mathcal{R}_m$ ;
        for ( $i = 1; i \leq m; i++$ )
            D&C( $\mathcal{R}_i$ );
        Tổng hợp kết quả;
    }
}
```

Sau đây là các minh họa kỹ thuật thiết kế “ Chia để trị “.

II. Thuật toán tìm kiếm nhị phân.

1. Phát biểu bài toán

Cho mảng n phần tử đã được sắp tăng dần và một phần tử x . Tìm x có trong mảng hay không ? Nếu có x trong mảng thì cho kết quả là 1, ngược lại cho kết quả 0.

Giải bằng thuật toán tìm kiếm nhị phân .

2. Ý tưởng

Chia đôi mảng , mỗi lần so sánh phần tử giữa với x , nếu phần tử x nhỏ hơn thì lấy nửa trái, ngược lại thì lấy nửa phải.

3. Mô tả thuật toán

Input : $a[1..n]$

Output : $\begin{cases} 1; x \in a \\ 0; x \notin a \end{cases}$

Mô tả :

Tknp(a, x, Đầu, Cuối) \equiv

If (Đầu > Cuối)

return 0 ; {dãy trống}

Else

{

Giữa = (Đầu + cuối) / 2;

If (x == a[Giữa])

return 1;

else

if (x > a[Giữa])

Tknp(a, x, Giữa + 1, Cuối) ;

else

Tknp(a, x, Đầu, Giữa - 1) ;

}

4. Độ phức tạp thời gian của thuật toán

a) Trường hợp tốt nhất : tương ứng với sự tìm được x trong lần so sánh đầu tiên, tức là : $a[\text{Giữa}] == a[n/2] == x$ (x nằm ở vị trí giữa mảng).

Ta có : $T_{\text{tốt}}(n) = O(1)$.

b) Trường hợp xấu nhất : Độ phức tạp là $O(\lg n)$.

Thật vậy, Nếu gọi $T(n)$ là độ phức tạp của thuật toán , thì sau khi kiểm tra điều kiện ($x == a[\text{giữa}]$) và sai thì gọi đệ qui thuật toán này với dữ liệu giảm nửa, nên thỏa mãn công thức truy hồi :

$T(n) = 1 + T[n/2]$; $n \geq 2$ và $T[1] = 0$.

5. Cài đặt

```
int tknp(int a[max],int x,int l, int r)
{
    int mid;
    if ( l > r )
        return 0;
    mid = (l+r)/2;
    if ( x == a[mid] )
        return 1;
    if ( x > a[mid] )
        return tknp(a,x,mid+1,r);
    return tknp(a,x,l,mid-1);
}
```

III. Bài toán MinMax

1. Phát biểu bài toán

Tìm giá trị Min, Max trong đoạn $a[l..r]$ của mảng $a[1..n]$.

2. Ý tưởng

Tại mỗi bước, chia đôi đoạn cần tìm rồi tìm Min, Max của từng đoạn, sau đó tổng hợp lại kết quả.

Nếu đoạn chia chỉ có 1 phần tử thì $\text{Min} = \text{Max}$ và bằng phần tử đó.

Minh họa :

i	1	2	3	4	5	6	7	8
a[i]	10	1	5	0	9	3	15	19

Tìm giá trị Min, Max trong đoạn $a[2..7]$ của mảng $a[1..7]$.

Ký hiệu :

$\text{MinMax}(a, l, r, \text{Min}, \text{Max})$ cho Min và Max trong đoạn $a[l..r]$.

$\text{MinMax}(a, 2, 7, \text{Min}, \text{Max})$ Cho $\text{Min} = 0$ và $\text{Max} = 15$ trong đoạn $a[2..7]$

3. Thuật toán

Input : $a[l..r]$, ($1 \leq r$)

Output : $\text{Min} = \text{Min}(a[l], \dots, a[r])$,

$\text{Max} = \text{Max}(a[l], \dots, a[r])$.

Mô tả :

$\text{MinMax}(a, l, r, \text{Min}, \text{Max}) \equiv$

if ($l == r$)

{

Min = $a[l]$;

Max = $a[l]$;

}

Else

{

$\text{MinMax}(a, l, (l+r) / 2, \text{Min1}, \text{Max1})$;

$\text{MinMax}(a, (l+r) / 2 + 1, r, \text{Min2}, \text{Max2})$;

If ($\text{Min1} < \text{Min2}$)

Min = Min1 ;

Else

Min = Min2 ;

If ($\text{Max1} > \text{Max2}$)

Max = Max1

Else

Max = Max2 ;

}

4. Độ phức tạp thuật toán

Gọi $T(n)$ là số phép toán so sánh cần thực hiện. Khi đó ta có :

$$T(n) = \begin{cases} T(n/2) + T(n/2) + 2 & ; n > 2 \\ 1 & ; n = 2 \\ 0 & ; n = 1 \end{cases}$$

Với $n = 2^k$, thì :

$$\begin{aligned} T(n) &= 2 + 2T(n/2) = 2 + 2^2 + 2^2T(n/2^2) = \dots = 2^{k-1}T(2) + \sum_{i=1}^{k-1} 2^i \\ &= \sum_{i=1}^k 2^i - 2^{k-1} = 2^{k+1} - 2^{k-1} - 2 = \frac{3n}{2} - 2. \end{aligned}$$

Vậy $T(n) \in O(n)$.

5. Cài đặt

```
void MinMax(int a[], int l, int r, int &Min, int &Max )
```

```
{
    int Min1, Min2, Max1, Max2;
    if (l == r )
    {
        Min = a[l];
        Max = a[l];
    }
    else
    {
        MinMax(a, l, (l+r)/2, Min1, Max1);
        MinMax(a, (l+r)/2 + 1, r, Min2, Max2);
        if (Min1 < Min2)
            Min = Min1;
        else
            Min = Min2;
        if (Max1 > Max2)
            Max = Max1;
        else
            Max = Max2;
    }
}
```

IV. Thuật toán QuickSort

Dùng thuật toán QuickSort (QS) để sắp xếp các giá trị trong một mảng các số theo một thứ tự, chẳng hạn tăng dần.

Phương pháp QuickSort (hay còn gọi là phân đoạn) là một cải tiến của phương pháp sắp xếp đổi chỗ trực tiếp, do C.A.R. Hoare đề xuất.

1. Ý tưởng

Chọn ngẫu nhiên một phần tử x .

Duyệt dãy từ bên trái (theo chỉ số i) trong khi còn $a_i < x$.

Duyệt dãy từ bên phải (theo chỉ số j) trong khi còn $a_j > x$.

Đổi chỗ a_i và a_j nếu hai phía chưa vượt qua nhau.

... tiếp tục quá trình duyệt và đổi chỗ như trên trong khi hai phía còn chưa vượt qua nhau (tức là còn có $i \leq j$).

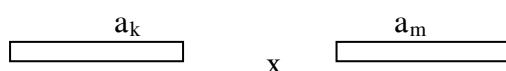
- Kết quả phân hoạch dãy thành 3 phần :

$a_k \leq x$ với $k = 1, \dots, j$ (Dãy con thấp);

$a_m \geq x$ với $m = i, \dots, n$ (Dãy con cao);

$a_h = x$ với $h = j+1, \dots, i-1$.

(Vì thế phương pháp này còn gọi là sắp xếp bằng phân hoạch).



Tiếp tục phân hoạch cho phần trái (dãy con thấp nhỏ hơn x), cho phần phải (lớn hơn x) ... cho đến khi các phân hoạch chỉ còn lại một phần tử, là sắp xếp xong. Thuật toán thể hiện ý tưởng đệ quy và cách thiết kế chia để trị.

2. Mô tả thuật toán

- Thuật toán QuickSort

Input : $a[1..n]$

Output : $a[1..n]$ không giảm.

Mô tả thuật toán :

QuickSort (a, n) \equiv

 QS($a, 1, n$) ;

- Thuật toán phân hoạch

Input : $a[1..n], l, r$;

Output : $a[l..r]$ tăng dần

Mô tả :

QS(a, l, r) \equiv

$i = l$;

$j = r$;

$x = a[(l+r)/2]$; // Chọn phần tử giữa

 do

 {

 while ($a[i] < x$)

$i++$;

 while ($a[j] > x$)

$j--$;

 if ($i \leq j$)

 {

 đổi chỗ $a[i]$ và $a[j]$;

$i++$;

```

        j--;
    }
}
while (i <= j);
if (l < j)
    QS(a,l,j);
if (r > i)
    QS(a,i,r);

```

3. Độ phức tạp của thuật toán

• Điều tốt nhất có thể xảy ra trong QuickSort là mỗi giai đoạn phân hoạch phân chia mảng thành 2 nửa. Điều này khiến cho số lần so sánh cần thiết của QuickSort thỏa mãn công thức truy hồi “chia để trị” sau đây :

$$T_n = 2T_{\frac{n}{2}} + n \cong nLg n.$$

$2T_{\frac{n}{2}}$: Phí tổn sắp xếp 2 mảng con.

n : Phí tổn kiểm tra mỗi phần tử.

• Trường hợp xấu nhất ứng cho việc chọn phần tử x lại có giá trị lớn nhất hoặc nhỏ nhất trong dãy. Giả sử phần tử lớn nhất được chọn (phần tử x), khi đó mỗi bước chia sẽ chia n phần tử thành $n-1$ phần tử trái và 1 phần tử phải. Kết quả là cần tối n phép chia (thay cho $lg n$), và như thế độ phức tạp sẽ là $T(n) = O(n^2)$.

Trong trường hợp dãy nhập vào đã có thứ tự (thuận hay ngược), phần tử lớn nhất được chọn sẽ nằm ở các biên (phải hoặc trái), cho nên thuật toán QuickSort không có hiệu quả.

• Trong trường hợp trung bình :

Công thức truy hồi để tính số lần so sánh mà QuickSort cần để hoán vị ngẫu nhiên n phần tử là :

$$T_n = (n+1) + \frac{1}{n} \sum_{1 \leq k \leq n} (T_{k-1} + T_{n-k}); \text{ Với } n \geq 2; C_0 = C_1 = 1.$$

Giá trị $n+1$ bao hàm chi phí so sánh phần tử phân hoạch với mỗi phần tử còn lại, tổng còn lại mang ý nghĩa là mỗi phần tử k có thể là phần tử phân hoạch với xác suất $\frac{1}{n}$ và sau đó còn lại các mảng con có kích thước $k-1$ và $n-k$.

$$T_n = n+1 + \frac{2}{n} \sum_{k=1}^n T_{k-1}$$

Thực hiện liên tiếp các phép toán sau cho cả 2 vế : Nhân cho n và trừ cho $(n-1)C_{n-1}$:

$$\begin{aligned}
nT_n - (n-1)T_{n-1} &= n(n+1) + n \frac{2}{n} \sum_{k=1}^n T_{k-1} - (n-1)T_{n-1} \\
&= n(n+1) + 2 \sum_{k=1}^n T_{k-1} - (n-1) \left[n + \frac{2}{n-1} \sum_{k=1}^{n-1} T_{k-1} \right] \\
&= n(n+1) - n(n-1) + 2 \sum_{k=1}^n T_{k-1} - 2 \sum_{k=1}^{n-1} T_{k-1}
\end{aligned}$$

Ta được : $nT_n - (n-1)T_{n-1} = n(n+1) - n(n-1) + 2T_{n-1}$

Suy ra : $nT_n = (n+1)T_{n-1} + 2n$

Chia cả 2 vế cho $n(n+1)$:

$$\begin{aligned}
\frac{T_n}{n+1} &= \frac{T_{n-1}}{n} + \frac{2}{n+1} = \frac{T_{n-2}}{n-1} + \frac{2}{n} + \frac{2}{n+1} = \frac{2}{n+1} + \frac{2}{n} \cdots + \frac{2}{4} + \frac{2}{3} + \frac{T_1}{2} \\
&= \frac{1}{2} + \sum_{k=2}^n \frac{2}{k+1} = \frac{1}{2} + 2 \sum_{k=3}^{n+1} \frac{1}{k} \\
\frac{T_n}{n+1} &\cong 2 \sum_{k=3}^n \frac{1}{k} \cong 2 \int_1^n \frac{1}{x} dx = 2 \ln(n)
\end{aligned}$$

Như vậy, Độ phức tạp trung bình là $O(n \ln n)$

V. Thuật toán nhân Strassen nhân 2 ma trận

1. Bài toán

Cho 2 ma trận vuông a, b cấp n , n là lũy thừa 2.

Dùng thuật toán Strassen nhân 2 ma trận vuông cấp n .

2. Mô tả

Ứng dụng thiết kế chia để trị, mỗi ma trận A, B, C ta chia thành 4 ma trận con và biểu diễn tích 2 ma trận $A \times B = C$ theo các ma trận con đó :

$$\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

Trong đó :

$$\begin{aligned}
C_{11} &= A_{11}B_{11} + A_{12}B_{21} \\
C_{12} &= A_{11}B_{12} + A_{12}B_{22} \\
C_{21} &= A_{21}B_{11} + A_{22}B_{21} \\
C_{22} &= A_{21}B_{12} + A_{22}B_{22}
\end{aligned}$$

Nếu theo cách nhân thông thường, thì cách chia để trị này dẫn đến công thức truy hồi : $T(n) = 8T(n/2) + \theta(n^2)$. Đáng tiếc là kết quả này cho lời giải $T(n) \in \theta(n^3)$.

Nhưng theo khám phá của Strassen, chỉ cần 7 phép nhân đệ quy $n/2 \times n/2$ ma trận và $\theta(n^2)$ phép cộng trừ vô hướng theo công thức truy hồi :

$$T(n) = 7T(n/2) + 18(n/2)^2 \in O(n^{\lg 7}) = O(n^{2.81}).$$

Cụ thể, để nhân 2 ma trận vuông cấp 2, theo Strassen chỉ cần 7 phép nhân và 18 phép cộng (trừ) các số. Để tính :

$$\begin{bmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix}$$

- Đầu tiên tính 7 tích :

$$\begin{aligned} m_1 &= (a_{12} - a_{22}) (b_{21} + b_{22}) \\ m_2 &= (a_{11} + a_{22}) (b_{11} + b_{22}) \\ m_3 &= (a_{11} - a_{21}) (b_{11} + b_{12}) \\ m_4 &= (a_{11} + a_{12}) b_{22} \\ m_5 &= a_{11} (b_{12} - b_{22}) \\ m_6 &= a_{22} (b_{21} - b_{11}) \\ m_7 &= (a_{21} + a_{22}) b_{11} \end{aligned}$$

- sau đó tính c_{ij} theo công thức :

$$\begin{aligned} c_{11} &= m_1 + m_2 - m_4 + m_6 \\ c_{12} &= m_4 + m_5 \\ c_{21} &= m_6 + m_7 \\ c_{22} &= m_2 - m_3 + m_5 - m_7 \end{aligned}$$

Thuật toán có thể viết như sau :

```
strass(a, b, c, n)≡
  if ( n == 2 )
    nhan2(a,b,c);
  else
    {
      tach(a,a11,a12,a21,a22,n);
      tach(b,b11,b12,b21,b22,n);
      tach(c,c11,c12,c21,c22,n);

      strass(a11,b11,d1,n/2);
      strass(a12,b21,d2,n/2);
      cong(d1,d2,c11,n/2);

      strass(a11,b12,d1,n/2);
      strass(a12,b22,d2,n/2);
      cong(d1,d2,c12,n/2);

      strass(a21,b11,d1,n/2);
      strass(a22,b21,d2,n/2);
      cong(d1,d2,c21,n/2);

      strass(a21,b12,d1,n/2);
```



```

    strass(a22,b22,d2,n/2);
    cong(d1,d2,c22,n/2);

    Hop(c11,c12,c21,c22,c,n);
}

```

VI. Bài toán hoán đổi 2 phần trong 1 dãy

1. Phát biểu bài toán

$a[1..n]$ là một mảng gồm n phần tử. Ta cần chuyển m phần tử đầu tiên của mảng với phần còn lại của mảng ($n-m$ phần tử) mà không dùng một mảng phụ.

Chẳng hạn, với $n = 8, a[8] = (1, 2, 3, 4, 5, 6, 7, 8)$

Nếu $m = 3$, thì kết quả là : $(4, 5, 6, 7, 8, 1, 2, 3)$

Nếu $m = 5$, thì kết quả là : $(6, 7, 8, 1, 2, 3, 4, 5)$

Nếu $m = 4$, thì kết quả là : $(5, 6, 7, 8, 1, 2, 3, 4)$

2. Ý tưởng

* Nếu $m = n - m$: Hoán đổi các phần tử của 2 nửa mảng có độ dài bằng nhau :

* Nếu $m \neq n - m$:

- Nếu $m = n - m$:
- Nếu $m < n - m$: hoán đổi m phần tử đầu với m phần tử cuối của phần còn lại. Sau đó trong mảng $a[1..n-m]$ ta chỉ cần hoán đổi m phần tử đầu với phần còn lại.
- Nếu $m > n - m$: hoán đổi $n-m$ phần tử đầu tiên với $n-m$ phần tử của phần sau. Sau đó trong mảng $a[n-m+1..n]$ ta hoán đổi $n-m$ phần tử cuối mảng với các phần tử của phần đầu.

Như vậy, bằng cách áp dụng phương pháp chia để trị, ta chia bài toán thành 2 bài toán con :

- Bài toán thứ nhất là hoán đổi hai mảng con có độ dài bằng nhau, cụ thể là hoán đổi nửa số phần tử đầu và cuối của mảng cho nhau bằng cách đổi chỗ từng cặp phần tử tương ứng.
- Bài toán thứ hai cùng dạng với bài toán đã cho nhưng kích thước nhỏ hơn, nên có thể gọi thuật toán đệ qui để giải và quá trình gọi đệ qui sẽ dừng khi đạt tới sự hoán đổi 2 phần có độ dài bằng nhau

Vậy mấu chốt của thuật toán là thực hiện thao tác hoán đổi 2 nhóm phần tử, lặp lại thao tác này trong khi số lượng phần tử của 2 nhóm con khác nhau. Nên ta sẽ thay thuật toán đệ qui bằng thuật toán lặp.

3. Thuật toán

// Hoán đổi m phần tử đầu của mảng với phần còn lại.

Input : $a[1..n], m. (m \leq n)$

Output : $a[1..n]$ với tính chất m phần tử đầu mảng a (mảng nhập) nằm cuối mảng

kết quả, n-m phần tử cuối mảng nhập nằm ở đầu mảng kết quả.

Mô tả thuật toán :

$\text{Transpose}(a, n, m) \equiv$

$i = m; j = n - m; m = m + 1;$

Khi ($i \neq j$)

Nếu ($i > j$)

{

$\text{Exchange}(a, m - i, m, j);$

$i = i - j;$

}

Ngược lại

{

$j = j - i;$

$\text{Exchange}(a, m - i, m + j, i);$

}

$\text{Exchange}(a, m - i, m, i);$

* **Thuật toán exchange :**

input a,

i, j, // vị trí

m; // Số phần tử cần hoán đổi

output a

Mô tả :

$\text{Exchange}(a, i, j, m) \equiv$

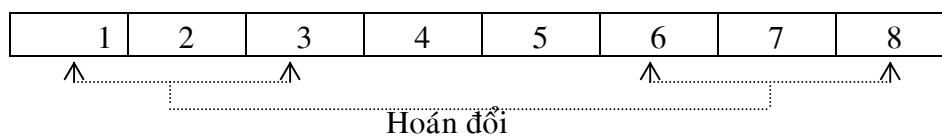
Với mọi $p = 0 \rightarrow m - 1$

Đổi chỗ ($a[i + p], a[j + p]$);

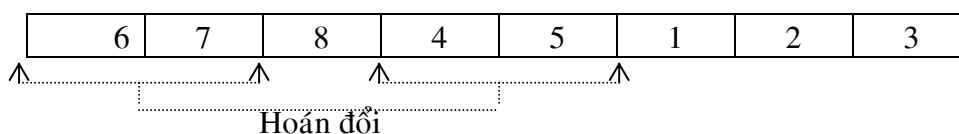
Minh họa :

$n = 8, a[8] = (1, 2, 3, 4, 5, 6, 7, 8), m = 3.$

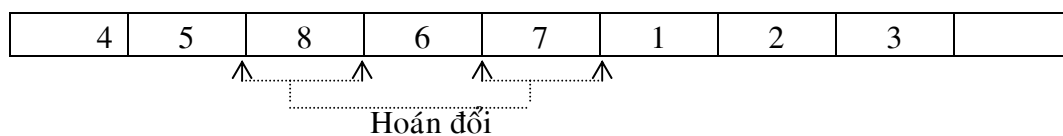
- Mảng nhập :



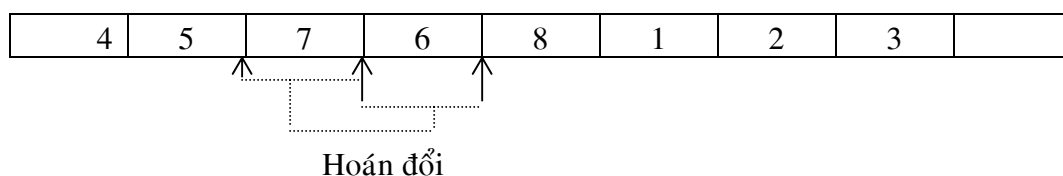
- $\text{Exchange}(a, 1, 6, 3)$



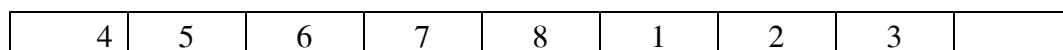
- Exchange(a,1,4,2)



- Exchange(a,3,5,1)



- Exchange(a,3,4,1)



- Kết thúc thuật toán.

4. Độ phức tạp thuật toán

Kí hiệu : $T(i, j)$ là số phần tử cần đổi chỗ để hoán đổi dãy i phần tử và dãy j phần tử, ta có công thức truy hồi sau :

$$T(i, j) = \begin{cases} i; & \text{nếu } i = j \\ j + T(i - j, j); & \text{nếu } i > j \\ i + T(i, j - i); & \text{nếu } i < j \end{cases}$$

$$\Rightarrow T(i, j) = i + j - \text{UCLN}(i, j)$$

UCLN(i, j) là ước chung lớn nhất của i, j .

5. Cài đặt

void Transpose(day a,int n,int m)

```
{
    int i, j;
    i = m;
    j = n-m;
    m = m+1;
    while ( i != j )
        if(i > j)
        {
            Exchange(a,m-i,m,j);
            i = i - j;
        }
}
```

```

        else
        {
            j = j - i;
            Exchange(a,m-i,m+j,i);
        }
        Exchange(a,m-i,m,i);
    }
}
//*****
void Exchange(day a, int i, int j, int m)
{
    for (int k = 0; k <= m-1; k++)
        doicho(a[i+k], a[j+k]);
}

```

VII. Trộn hai đường trực tiếp

1. Bài toán

Sắp tăng dần dãy các số bằng thuật toán trộn hai đường trực tiếp.

2. Ý tưởng

Thuật toán sắp xếp kiểu trộn hai đường trực tiếp được thực hiện theo nhiều bước lặp :

* Mỗi bước lặp bao gồm hai giai đoạn :

Giai đoạn 1 : (Phân bố)

Phân bố luân phiên từng p phần tử từ dãy F vào các dãy trung gian $F1$, $F2$ trong khi chưa hết dãy F .

Giai đoạn 2 : (Trộn)

Trộn từng bộ p phần tử trong dãy $F1$ với p phần tử trong $F2$, kết quả trộn được đưa vào F , trong khi chưa hết dãy $F1$ và chưa hết dãy $F2$.

* Các bước lặp còn được thực hiện trong khi p còn \leq số các phần tử của F .

Bước đầu tiên p được khởi động bằng 1.

Mỗi bước lặp sau(sau một lần phân bố và trộn), số phần tử p sẽ khởi động lại là : $p = p * 2$.

Minh họa :

Giả sử F là dãy dùng sắp thứ tự, F có nội dung như sau :

F	9	2	1	1	5	8	1	2	9	6	7	4	6	7	2	2	3	1	1
	0														1				

Đầu tiên ta phân bố luân phiên từng phần tử của dãy nguồn F vào các dãy trung gian $F1$, $F2$.

• Trong lần phân bố đầu tiên, ta có :

F1	90	1	5	1	9	7	6	21	3	1
F2	2	1	8	2	6	4	7	2	1	

Ta thực hiện trộn hai đường từng phần tử ở F1 với từng phần tử ở F2; Kết quả trộn được ghi vào F.

F	2-90	1-1	5-8	1-2	6-9	4-7	6-7	2-21	1-3	1
---	------	-----	-----	-----	-----	-----	-----	------	-----	---

• Sang lần thứ 2:

F1	2-90	5-8	6-9	6-7	1-3
F2	1-1	1-2	4-7	2-21	1

F	1-1 – 2-90	1-2-5-8	4-6-7-9	2-6-7-21	1-1-3
---	------------	---------	---------	----------	-------

• Sang lần thứ 3 :

F1	1-1 – 2-90	4-6-7-9	1-1-3
F2	1-2-5-8	2-6-7-21	

F	1-1-1-2-2-5-8-90	2-4-6-6-7-7-9-21	1-1-3
---	------------------	------------------	-------

• Sang lần thứ 4 :

F1	1-1-1-2-2-5-8-90	1-1-3
F2	2-4-6-6-7-7-9-21	

F	1-1-1-2-2-2-4-5-6-6-7-7-8-9-21-90	1-1-3
---	-----------------------------------	-------

• Sang lần thứ 5 :

F1	1-1-1-2-2-2-4-5-6-6-7-7-8-9-21-90
F2	1-1-3

F	1-1-1-1-1-2-2-2-3-4-5-6-6-7-7-8-9-21-90
---	---

Khi đó F được sắp thứ tự .

3. Thiết kế

* Thuật toán trộn 2 đường trực tiếp :

input F[1..n]; // dãy cần sắp

Output F đã được sắp

Mô tả:

p = 1;

Trong khi (p <= n) ta thực hiện :

{

- Phân bố luân phiên từng **p** phần tử từ dãy F vào các dãy trung gian F1, F2 trong khi chưa hết dãy F.

-Trộn từng cặp **p** phần tử trong dãy F1 với **p** phần tử trong dãy F2, kết quả ghi vào F, trong khi chưa hết dãy F1 và chưa hết dãy F2;

```

- Khởi động lại p : p = p*2;
}

```

Mô tả trên có thể viết thành hàm :

//F,n,F1,F2 là các biến toàn cục

```
void mergesort ()
```

```

{
    p = 1;
    while ( p <= n )
    {
        distribution (p);
        merge(p);
        p = p * 2;
    }
}

```

Như vậy, thuật toán chủ yếu được xây dựng trên 2 thao tác :

- distribution (p) : Phân bố luân phiên **p** phần tử từ dãy F vào các dãy trung gian F1, F2 trong khi chưa hết dãy F.

- merge(p) : Trộn từng cặp p phần tử trong F1, và p phần tử trong F2, kết quả lưu trữ vào F, trong khi chưa hết dãy F1 và chưa hết dãy F2;

a) Thuật toán phân bố

Phân bố luân phiên **p** phần tử từ dãy F vào các dãy trung gian F1, F2 cho đến hết dãy F.

a1) Thiết kế :

Input F;

Output F1,F2;

Mô tả

Thực hiện

```

{
    Đọc từng p phần tử trong F và chép luân phiên vào F1, F2;
}

```

Trong khi (chưa hết dãy F);

Trong mô tả trên, có 2 thao tác con cần phải lưu ý :

Thao tác 1 :

Làm thế nào để xử lý một cách tự động việc chép luân phiên vào F1 và F2.

Ta thực hiện bằng cách : Dùng một khoá k, với $k \in \{1,2\}$ và quy định :

Nếu $k = 1$ thì chép vào F1;

Nếu $k = 2$ thì chép vào F2;

Giả sử đầu tiên cho $k = 1$ để quyết định chép p phần tử của F vào F1 trước.

Sau mỗi lần chép xong p phần tử, ta chỉ cần khởi động lại giá trị $k = 3-k$.

Thao tác 2 :

Đọc p phần tử của F chép vào F1 như thế nào ? Ta đọc từng phần tử của F và chép phần tử đó vào F1; Việc đọc chép từng phần tử này còn được thực hiện trong khi chưa đủ p phần tử và chưa hết dãy F.

Vậy thao tác phân bố có thể mô tả chi tiết như sau :

```
do
{
    i = 1;
    while ( i <= p && chưa hết dãy F )
    {
        Đọc phần tử thứ i trong F;
        if ( k == 1)
            chép vào F1;
        else
            chép vào F2;
        i++;
    }
    k = 3-k;
}
while ( chưa hết dãy F);
```

Thao tác phân bố cài đặt thành một hàm như sau :

//F, F1, F2, n, h₁,h₂ là các biến toàn cục.

```
void distribute(int p)
{
    int i, k=1, l = 1;
    h1 = 0; h2 = 0;
    do
    {
        i = 1;
        while( i<=p && l <= n)
        {
            if(k==1)
            {
                h1++;
                F1[h1] = F[l];
            }
            else
            {
                h2++;
                F2[h2] = F[l];
            }
            i++;
            l++;
        }
    }
}
```

```

        k = 3-k;
    }
    while(l <= n);
}

```

b) Thuật toán trộn từng bộ p phần tử trong F1 và p phần tử trong F2.

Trộn từng bộ p phần tử trong F1, và p phần tử trong F2, kết quả lưu trữ vào F, trong khi chưa hết F1 và F2.

a1) Thiết kế :

Input F1, F2;

Output F;

Mô tả :

```

Trong khi ( chưa hết dãy F1 và chưa hết dãy F2 )
{
    Trộn từng cặp p phần tử của F1 và của F2 vào F;
}
Trong khi (chưa hết dãy F1)
    chép các phần tử còn lại của F1 vào F;
Trong khi (chưa hết dãy F2)
    chép các phần tử còn lại của F2 vào F;

```

- Ta cần chỉ rõ công việc trộn từng cặp p phần tử của F1 và của F2 vào F hoạt động như thế nào ? Đó là :

- (*) : Đọc từng phần tử trong F1, trong F2, so sánh giá trị của chúng, giá trị nào nhỏ hơn thì chép phần tử tương ứng vào F. Nếu là phần tử của F1 thì đọc tiếp 1 phần tử của F1; ngược lại thì đọc tiếp 1 phần tử của F2

- (**) : Thao tác trên còn được thực hiện trong khi : chưa đọc đủ p phần tử trong F1 và chưa đọc đủ p phần tử trong F2 và chưa hết dãy F1 và chưa hết dãy F2;

- Vòng lặp (**) dừng khi đã đọc đủ p phần tử trong F1, hoặc đã đọc đủ p phần tử trong F2, hoặc hết dãy F1 hoặc hết dãy F2; Và khi đó ta cần xử lý các trường hợp sau :

Trong trường hợp chưa hết dãy F1 và chưa hết dãy F2 :

Nếu chưa đủ p phần tử trong F1, thì đọc và chép các phần tử của F1 vào F cho đủ p; Tương tự như vậy cho F2.

a2) Cài đặt :

/* F1,F2,n,h1,h2 là các biến toàn cục.

Ta dùng các biến :

- r1 : đếm các phần tử đọc được trong dãy F1.
- r2 : đếm các phần tử đọc được trong dãy F2 .
- i1 : duyệt dãy F1
- i2 = 1 duyệt dãy F2

*/

void merge(int p)


```
{
    int t, i1 = 1, i2 = 1, r1, r2;
    int h = 0;
    while(i1 <= h1 && i2 <= h2)
    {
        r1=r2=1;
        while((r1 <= p) && (r2 <= p) && i1 <= h1 && i2 <= h2)
        {
            if(F1[i1] <= F2[i2])
            {
                h++;
                F[h] = F1[i1];
                r1++;
                i1++;
            }
            else
            {
                h++;
                F[h] = F2[i2];
                r2++;
                i2++;
            }
        }

        while(i1 <= h1 && r1 <= p)
        {
            h++;
            b[h] = b1[i1];
            i1++;
        }
        while(i2 <= h2 && r2 <= p)
        {
            h++;
            b[h] = b2[i2];
            i2++;
        }
    }
    while(i1 <= h1)
    {
        h++;
        b[h] = b1[i1];
        i1++;
    }
}
```

```

while(i2 <= h2)
{
    h++;
    b[h] = b2[i2];
    i2++;
}
n = h;
}

```

4. Độ phức tạp thuật toán

Ta nhận xét rằng, trong phương pháp sắp xếp bằng trộn hai đường trực tiếp, số lượng các bước sao chép các phần tử từ dãy này sang dãy kia còn lớn hơn số lượng các bước so sánh giữa các phần tử. Vì ứng với một lần so sánh thì có một thao tác sao chép, nhưng nếu một dãy nào đó xử lý cạn (hết dãy) thì phần đuôi của dãy còn lại được sao chép mà không ứng với một phép so sánh nào. Vì thế, đối với phương pháp này, ta chọn phép sao chép làm căn cứ đánh giá thời gian thực hiện của thuật toán.

Trong mỗi lần phân bố và trộn thì toàn bộ n phần tử được duyệt qua, so sánh và chép vào dãy đích (output). Như vậy thời gian chi phí cho mỗi bước có cấp là $O(n)$.

Vì trong mỗi bước (bước thứ k) ta giải quyết được $2^k = p$ giá trị và tiến trình dừng khi $p \geq n$, nên ta có $\lg n$ bước, do đó cấp thời gian chi phí cho phương pháp này là $O(n \lg n)$.

Một nhược điểm của phương pháp sắp xếp bằng kiểu trộn hai đường trực tiếp là chi phí cho không gian quá lớn: nó đòi hỏi cung cấp vùng nhớ 2^n phần tử, gấp đôi so với phương pháp thông thường. Do đó phương pháp này chỉ thích hợp khi ta thao tác trên các tệp.

Mặt khác, phương pháp sắp xếp kiểu trộn hai đường trực tiếp có một nhược điểm quan trọng nữa là nó tự giới hạn số lượng các giá trị cố định là $1, 2, 4, \dots, 2^k$, trong đó $2^k < n$. Như vậy ta luôn luôn phải duyệt qua k bước chia và trộn. Nếu cho phép số lượng các phần tử trong một lần trộn có kích thước khác thì số các bước có thể giảm đi và trong trường hợp này việc sắp xếp có khả năng kết thúc sớm.

BÀI TẬP

Bài 1 : (Nhân các số lớn)

Kỹ thuật chia để trị nhân 2 số nguyên dương x, y dưới dạng chuỗi :

Nhan(x, y) \equiv

if($l(x), l(y) \leq 4$)

 Nhân 2 số nguyên nguyên kiểu long;

else

 Giả sử $l(x) = l(y) = n$;

 Tách x thành 2 chuỗi con : a (Nửa trái), b (nửa phải)

 Tách y thành 2 chuỗi con : c (Nửa trái), d (nửa phải)

$$Kq = \text{nhan}(a,c) \cdot 10^n + \text{nhan}(a,d) \cdot 10^{n/2} + \text{nhan}(b,c) \cdot 10^{n/2} + \text{nhan}(b,d);$$

Bài 2 :

Thuật toán nhân 2 số nguyên n bit.

Giả sử x và y là 2 số nguyên n bit. Kỹ thuật “chia để trị” cho phép nhân xy là tách x, y ra 2 số nguyên $n/2$ bit :

x	<table border="1"><tr><td>a</td><td>b</td></tr></table>	a	b
a	b		
	$n/2$ bit trái $n/2$ bit phải		
y	<table border="1"><tr><td>c</td><td>d</td></tr></table>	c	d
c	d		

và tính theo công thức :

$$x \cdot y = a \cdot c \cdot 2^n + ((a-b) \cdot (d-c) + a \cdot c + b \cdot d) \cdot 2^{n/2} + b \cdot d$$

Ghi chú :

- Tách bit : Copy các bit.
- Nhân 2^n cho a : Dịch chuyển trái a n bit.

Bài 3 :

Cho $x, s, n \in \mathbb{Z}^+$. Giả sử $\sqrt[n]{x} \leq s$, tính $\sqrt[n]{x}$.

Bài 4 :

Sắp tăng dần một dãy x các số, bằng thuật toán trộn tự nhiên :

Trong khi (số đường chạy của $x > 1$)

- Tách luân phiên từng đường chạy của x vào các dãy trung gian x_1, x_2 ;
- Trộn từng cặp đường chạy của x_1, x_2 , lưu trữ vào x ;

Ghi chú :

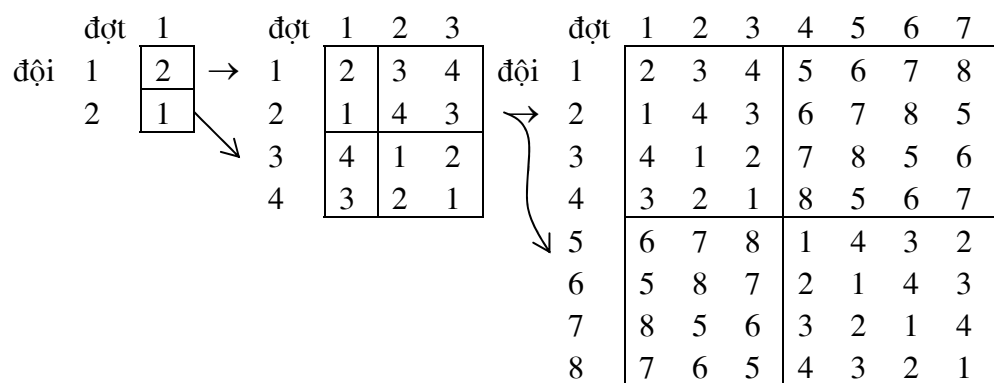
Đường chạy trong x là các dãy con có thứ tự (tăng dần) có chiều dài lớn nhất.

Bài 5 :

Lập lịch thi đấu vòng tròn 1 lượt cho n đội bóng đá, n là lũy thừa 2. Trong 1 đợt thi đấu, mỗi đội đấu 1 trận. Đấu trong $n-1$ đợt.

Kỹ thuật chia để trị xây dựng lịch cho một nửa số đội. Lịch này được lập nên do áp dụng đệ quy của thuật toán bằng cách tìm lịch cho một nửa số đội ... khi chỉ còn 2 đội thì ta có một cặp đấu đơn giản.

Các đội được đánh số từ 1 đến n . Giả sử có 8 đội. Lịch thi đấu cho 4 đội từ 1 đến 4 lấp đầy góc trái trên (4 hàng 3 cột) coi như đã lập xong. Góc trái dưới (4 hàng 3 cột) phải cho vào các đội có số thứ tự cao (từ 5 đến 8) ngược với các số khác. Lịch con này tạo ra bằng cách cộng 4 cho mỗi số nguyên của góc trái trên. Bây giờ ta đã làm đơn giản bài toán. Tất cả phần còn lại là các đội có số thấp đấu với các đội có số cao. Điều đó dễ hiểu là các đội từ 1-4 đấu với các đội 5-8 tương ứng từ đợt thứ 4 và hoán vị theo chu kỳ 5 đến 8 trong các đợt tiếp theo.



CHƯƠNG 3 : PHƯƠNG PHÁP QUAY LUI (Back Tracking)

I. Mở đầu

1. Ý tưởng

Nét đặc trưng của phương pháp quay lui là các bước hướng tới lời giải cuối cùng của bài toán hoàn toàn được làm thử.

Tại mỗi bước, nếu có một lựa chọn được chấp nhận thì ghi nhận lại lựa chọn này và tiến hành các bước thử tiếp theo. Còn ngược lại không có lựa chọn nào thích hợp thì làm lại bước trước, xoá bỏ sự ghi nhận và quay về chu trình thử các lựa chọn còn lại.

Hành động này được gọi là quay lui, thuật toán thể hiện phương pháp này gọi là quay lui.

Điểm quan trọng của thuật toán là phải ghi nhớ tại mỗi bước đi qua để tránh trùng lặp khi quay lui. Để thấy là các thông tin này cần được lưu trữ vào một ngăn xếp, nên thuật toán thể hiện ý thiết kế một cách đệ quy.

2. Mô hình

Lời giải của bài toán thường biểu diễn bằng một vec tơ gồm n thành phần $x = (x_1, \dots, x_n)$ phải thỏa mãn các điều kiện nào đó. Để chỉ ra lời giải x , ta phải xây dựng dần các thành phần lời giải x_i .

Tại mỗi bước i :

- Đã xây dựng xong các thành phần x_1, \dots, x_{i-1}
- Xây dựng thành phần x_i bằng cách lần lượt thử tất cả các khả năng mà x_i có thể chọn :
 - Nếu một khả năng j nào đó phù hợp cho x_i thì xác định x_i theo khả năng j . Thường phải có thêm thao tác ghi nhận trạng thái mới của bài toán để hỗ trợ cho bước quay lui. Nếu $i = n$ thì ta có được một lời giải, ngược lại thì tiến hành bước $i+1$ để xác định x_{i+1} .
 - Nếu không có một khả năng nào chấp nhận được cho x_i thì ta lùi lại bước trước (bước $i-1$) để xác định lại thành phần x_{i-1} .

Để đơn giản, ta giả định các khả năng chọn lựa cho các x_i tại mỗi bước là như nhau, do đó ta phải có thêm một thao tác kiểm tra khả năng j nào là chấp nhận được cho x_i .

Mô hình của phương pháp quay lui có thể viết bằng thủ tục sau, với n là số bước cần phải thực hiện, k là số khả năng mà x_i có thể chọn lựa.

```

Try(i) ≡
  for ( j = 1 → k)
    If (  $x_i$  chấp nhận được khả năng j)
    {
      Xác định  $x_i$  theo khả năng j;
    }
  
```

```

Ghi nhận trạng thái mới;
if( i < n)
    Try(i+1);
else
    Ghi nhận nghiệm;
    Trả lại trạng thái cũ cho bài toán;
}

```

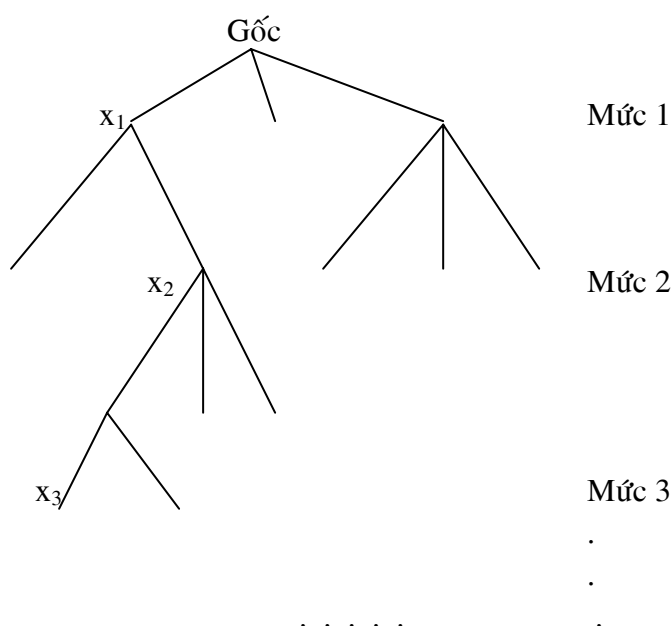
Ghi chú :

Tìm nghiệm bằng phương pháp quay lui có thể chuyển về tìm kiếm trên cây không gian các trạng thái, với cây được xây dựng từng mức như sau :

- Các nút con của gốc (thuộc mức 1) là các khả năng có thể chọn cho x_1 .
- Giả sử x_{i-1} là một nút ở mức thứ $i-1$, khi đó các nút con của x_{i-1} là các khả năng mà x_i có thể chọn, một khi đã tìm được các thành phần x_1, \dots, x_{i-1} .

Như vậy, mỗi nút x_i của cây biểu diễn một lời giải bộ phận, đó là các nút nằm trên đường đi từ gốc đến nút đó.

Ta có thể nói việc tìm kiếm nghiệm bằng phương pháp quay lui chính là tìm kiếm theo chiều sâu trên cây không gian các trạng thái.



Sau đây là các minh họa về kỹ thuật thiết kế quay lui.

II. Bài toán Ngựa đi tuần

1. Phát biểu bài toán

Cho bàn cờ có $n \times n$ ô. Một con ngựa được phép đi theo luật cờ vua, đầu tiên được đặt ở ô có tọa độ x_0, y_0 .

Vấn đề là hãy chỉ ra các hành trình (nếu có) của ngựa – Đó là ngựa đi qua tất cả các ô của bàn cờ, mỗi ô đi qua đúng một lần.

2. Thiết kế thuật toán

Cách giải quyết rõ ràng là xét xem có thể thực hiện một nước đi kế nữa hay không. Sơ đồ đầu tiên có thể phát thảo như sau :

```

Try(i) ≡
    for ( j = 1 → k)
        If ( xi chấp nhận được khả năng k)
        {
            Xác định xi theo khả năng k;
            Ghi nhận trạng thái mới;
            if( i < n2 )
                Try(i+1);
            else
                Ghi nhận nghiệm;
            Trả lại trạng thái cũ cho bài toán;
        }

```

Để mô tả chi tiết thuật toán, ta phải qui định cách mô tả dữ liệu và các thao tác, đó là :

- Biểu diễn bàn cờ .
- Các khả năng chọn lựa cho x_i ?
- Cách thức xác định x_i theo j.
- Cách thức ghi nhận trạng thái mới, trả về trạng thái cũ.
- Ghi nhận nghiệm.

* Ta sẽ biểu diễn bàn cờ bằng 1 ma trận vuông cấp n : $\text{int } h[n][n]$;

Sở dĩ thể hiện mỗi ô cờ bằng 1 số nguyên thay cho giá trị boole (để đánh dấu ô đã được đi qua chưa) là vì ta muốn lần dò theo quá trình di chuyển của con ngựa.


Ta qui ước như sau :

$h[x][y] = 0 \equiv \hat{O} \quad \langle x, y \rangle$ ngựa chưa đi qua;
 $h[x][y] = i \equiv \hat{O} \quad \langle x, y \rangle$ ngựa đã đi qua ở bước thứ i ($1 \leq i \leq n^2$).

* Các khả năng chọn lựa cho x_i ? Đó chính là các nước đi của ngựa mà x_i có thể chấp nhận được.

Với cặp tọa độ bắt đầu $\langle x, y \rangle$ như trong hình vẽ, có tất cả 8 ô $\langle u, v \rangle$ mà con ngựa có thể đi đến. Giả sử chúng được đánh số từ 0 đến 7 như hình sau :

Tọa độ (a,b)
(-2,-1)
(-1,-2)
(1,-2)
(2,-1)

1	2	3	4	5
	4		3	
5				2
				
6				1
	7		0	

Tọa độ (a,b)
(-2,1)
(-1,2)
(1,2)
(2,1)

hàng x →

↑
cột y

(8 bước đi có thể có của con ngựa)

Một phương pháp đơn giản để có được u, v từ x, y là ta dùng 2 mảng a và b lưu trữ các sai biệt về tọa độ. Nếu ta dùng một chỉ số k để đánh số “bước đi kế” thì chi tiết đó được thể hiện bởi : $u = x + a[k]$; $v = y + b[k]$; $k = \overline{0,7}$.

Điều kiện “chấp nhận được” có thể được biểu diễn như kết hợp của các điều kiện :

Ô mới phải thuộc bàn cờ ($1 \leq u \leq n$ và $1 \leq v \leq n$) và chưa đi qua ô đó, nghĩa là $h[u,v] = 0$;

* Để ghi nhận nước đi hợp lệ ở bước i , ta gán $h[u][v] = i$; và để hủy một nước đi thì ta gán $h[u][v] = 0$.

* Ma trận h ghi nhận kết quả nghiệm. Nếu có $\langle x, y \rangle$ sao cho $h\langle x, y \rangle = 0$ thì đó không phải là lời giải của bài toán, còn ngược lại h chứa đường đi của ngựa.

Vậy thuật toán có thể mô tả như sau :

Input n, //Kích thước bàn cờ

x, y;//Toa độ xuất phát ở bước i

Output h;

Mô tả :

$$\text{Try}(i, x, y) \equiv$$

```
for(k = 0; k <= 7; k++)
{
    u = x + a[k];
    v = y + b[k];
    if (1 <= u , v <= n && h[u][v] == 0)
    {
        h[u][v] = i;
        if (i < n*)
            Try(i+1,u,v);
        else
            xuất_h(); // In ma trận h
    }
}
```


$$h[u][v] = 0;$$

$$\}$$

Thủ tục này xuất tất cả các lời giải, nếu có.

Thủ tục đệ qui được khởi động bằng một lệnh gọi các tọa độ đầu x_0, y_0 là tham số. Ô xuất phát có trị 1, còn các ô khác được đánh dấu còn trống.

$$H[x_0][y_0] = 1;$$

$$\text{Try}(2, x, y);$$

Các mảng a và b có thể khởi đầu như sau :

$$\text{int } a[8] = \{2, 1, -1, -2, -2, -1, 1, 2\};$$

$$\text{int } b[8] = \{1, 2, 2, 1, -1, -2, -2, -1\};$$

* Các lời giải sau là một số kết quả cho từ thuật toán trên :

	n=5	x=1	y=1	
1	6	15	10	21
14	9	20	5	16
19	2	7	22	11
8	13	24	17	4
25	18	3	12	23

	n=6	x=2	y=3		
36	17	6	29	8	11
19	30	1	10	5	28
16	35	18	7	12	9
23	20	31	2	27	4
34	15	22	25	32	13
21	24	33	14	3	26

* Với $n = 5$, các tọa độ xuất phát sau không có lời giải : (2,3), (3,2)...

III. Bài toán 8 hậu

1. Phát biểu bài toán

TÁM QUÂN HẬU ĐƯỢC ĐẶT LÊN BÀN CỜ VUA

sao cho chúng không ăn được nhau .

Bài toán này là một ví dụ nổi tiếng về việc dùng các phương pháp thử và sai và phương pháp quay lui.

2. Thiết kế thuật toán

Mấu chốt của thuật toán rõ ràng là xét xem có thể đặt quân hậu tiếp theo như thế nào. Theo luật cờ vua, một quân hậu có thể ăn các quân khác nếu nằm trên cùng 1 đường, đường này có thể là :

- Hàng,
- Cột,
- Các đường chéo (đi qua tọa độ vị trí của hậu).

Suy ra rằng mỗi hàng chỉ có thể chứa 1 và chỉ 1 quân hậu. Nên việc chọn vị trí cho quân hậu thứ i có thể giới hạn được ở hàng thứ i . Như thế tham số i trở thành chỉ hàng, và quá trình chọn vị trí cho quân hậu tiến hành trên toàn giá trị có thể có của các cột j .

Ta quy ước :

$x[i]$ // Chỉ quân hậu thứ i : nằm ở hàng i .

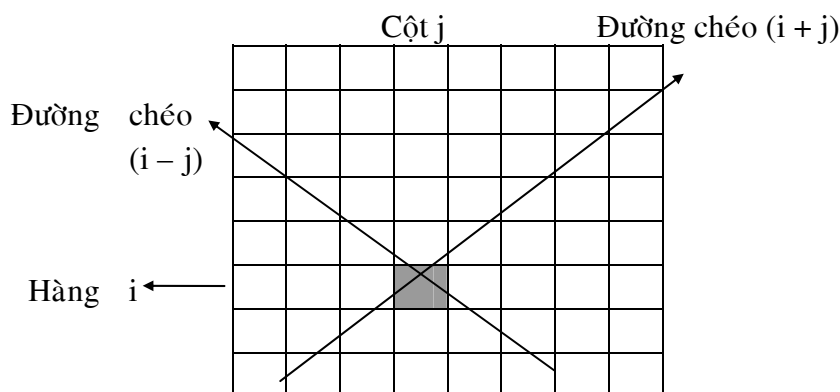
$x[i] = j$ // quân hậu thứ i đặt ở cột j ;

Để quân hậu i (trên hàng i) chấp nhận cột j thì cột j và 2 đường chéo qua ô $\langle i, j \rangle$ phải còn trống (tức là không có quân hậu khác chiếm lĩnh)

Lưu ý rằng trong 2 đường chéo :

- Đường chéo ngược (vuông góc với đường chéo chính) : tất cả các ô đều có tổng 2 tọa độ i và j là hằng;

- Đường chéo thuận (song song với đường chéo chính) : gồm tất cả các ô (i, j) mà có hiệu các tọa độ $(i - j)$ là hằng số.



Do đó ta sẽ chọn các mảng Boole 1 chiều để biểu diễn các trạng thái này :

$a[j] = 1$: Có nghĩa là không có quân hậu nào ở cột.

$b[i+j] = 1$: Có nghĩa là không có quân hậu nào ở đường chéo ngược $(i+j)$.

$c[i - j] = 1$: Có nghĩa là không có quân hậu nào ở đường chéo thuận $(i - j)$.

Vì :

$$1 \leq i, j \leq 8 \Rightarrow 2 \leq i+j \leq 16 \quad \text{Và} \quad -7 \leq i - j \leq 7.$$

Nên ta có thể khai báo :

```
int x[8],
    a[8],
    b[15],
    c[15];
```

Với các dữ liệu đã cho, thì lệnh đặt quân hậu sẽ thể hiện bởi :

$x[i] = j$; // đặt quân hậu thứ i trên cột j .

$a[j] = 0$; // Khi đặt hậu tại cột j , thì cột j không còn trống nữa

$b[i + j] = 0$; // Các đường chéo tương ứng cũng không còn

$c[i - j] = 0$; // trống nữa .

Còn lệnh Dời quân hậu là :

//Làm cho hàng i và các đường chéo tương ứng trở thành trống

$a[j] = 1$;

$b[i + j] = 1$;

$c[i - j] = 1$;

Còn điều kiện an toàn là ô có tọa độ (i, j) nằm ở hàng và các đường chéo chưa bị chiếm (được thể hiện bằng trị True). Do đó, có thể được thể hiện bởi biểu thức logic :

$$a[j] \&\& b[i+j] \&\& c[i-j]$$

```
Try(i) ≡
{
    for (j = 1; j <= 8; j++)
        if (a[j] && b[i+j] && c[i-j])
        {
            x[i] = j;
            a[j] = 0;
            b[i+j] = 0;
            c[i-j] = 0;
            if (i < 8)
                try (i+1);
            else
                Xuất(x);
        }
    /* Sau khi in 1 lời giải xong, trả lại tình trạng ban đầu còn trống cho hàng
a[j],
    đường chéo i+j và đường chéo i-j, để tìm lời giải khác */
    a[j] = 1;
    b[i+j] = 1;
    c[i-j] = 1;
}
}
```

Ghi chú :

Thuật toán này tìm được tất cả 92 lời giải. Thực ra là chỉ có 12 lời giải khác nhau thật sự, đó là vì thuật toán không ghi nhận tính đối xứng.

IV. Bài toán liệt kê các dãy nhị phân độ dài n

1. Phát biểu bài toán

Liệt kê các dãy có chiều dài n dưới dạng $x_1x_2...x_n$, trong đó $x_i \in \{0,1\}$.

2. Thiết kế thuật toán

Ta có thể sử dụng sơ đồ tìm tất cả các lời giải của bài toán. Hàm Try(i) xác định x_i , trong đó x_i chỉ có 1 trong 2 giá trị là 0 hay 1. Các giá trị này mặc nhiên được chấp nhận mà không cần phải thỏa mãn điều kiện gì. Nên Hàm try(i) có thể viết như sau :

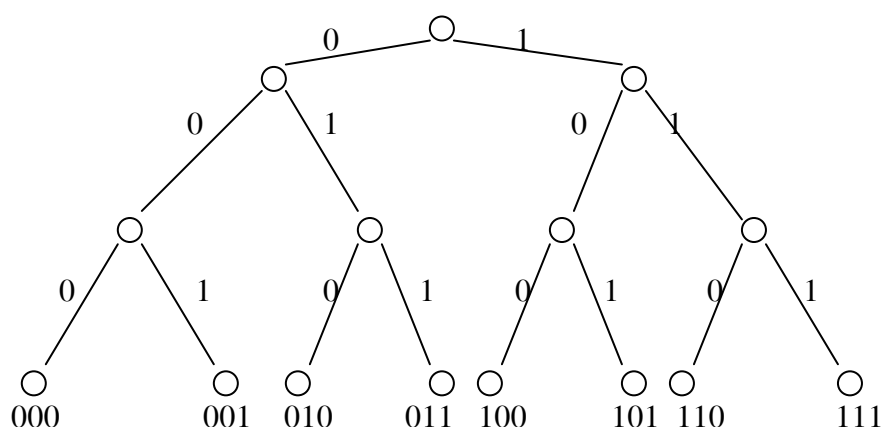
```
Try ( i) ≡
    for (j = 0; j <= 1; j++)
```

```

{
    x[i] = j;
    if (i < n)
        Try (i+1);
    else
        Xuất(x);
}

```

Cây không gian các trạng thái của bài toán có thể mô tả bởi :



V. Bài toán liệt kê các hoán vị

1. Phát biểu bài toán

Liệt kê các hoán vị của n số nguyên dương đầu tiên.

2. Thiết kế thuật toán

Ta biểu diễn các hoán vị dưới dạng $a_1 \dots a_n$; $a_i \in \{1, \dots, n\}$ và $a_i \neq a_j$ nếu $i \neq j$. Với mọi i , a_i chấp nhận giá trị j nếu j chưa được sử dụng, và vì vậy ta cần ghi nhớ j đã được sử dụng hay chưa khi quay lui. Để làm điều này ta dùng một dãy các biến logic b_j với quy ước :

$$\forall j = \overline{1, n} : b_j = \begin{cases} 1; & \text{nếu } j \text{ chưa sử dụng} \\ 0; & \text{nếu ngược lại.} \end{cases}$$

Sau khi gán j cho a_i , ta cần ghi nhớ cho b_j ($b_j = 0$) và phải trả lại trạng thái cũ cho b_j ($b_j = \text{True}$) khi thực hiện việc in xong một hoán vị.

Ta chú ý rằng dãy các biến b_j sẽ được khởi động bằng 1

Thuật toán có thể viết như sau :

```

Try( i)≡
{
    for ( j = 1; j <= n; j++)
        if ( b[j])
        {
            a[i] = j;

```

```

        b[j] = 0;          // Ghi nhận trạng thái mới
        if (i < n)
            Try(i+1);
        else
            Xuất();
        b[j] = True;      // Trả lại trạng thái cũ
    }
}

```

VI. Bài toán liệt kê các tổ hợp

1. Phát biểu bài toán

Liệt kê các tổ hợp chập k trong n phần tử.

2. Thiết kế thuật toán

Ta sẽ biểu diễn tổ hợp dưới dạng $x_1 \dots x_k$; Trong đó :

$$1 \leq x_1 < x_2 < \dots < x_k \leq n.$$

Ta nhận xét rằng với mọi $j \in \{1, \dots, n\}$:

x_i chấp nhận $j \Leftrightarrow j \in \{c_{i-1}+1, \dots, n-k+i\}$.

Các giá trị j thỏa điều kiện trên mặc nhiên được chấp nhận, nên ta không cần dùng các biến boole để ghi nhớ nữa.

Thuật toán có thể viết như sau :

```

Try( i ) ≡
    for ( j = 1; j <= n ; j++)
        if( x[i-1] + 1 <= j <= n - k + i )
        {
            x[i] = j;
            if (i < k)
                Try(i+1);
            else
                Xuất(x);
        }
}

```

VII. Bài toán tìm kiếm đường đi trên đồ thị

1. Phát biểu bài toán

$G = (V, U)$ là đơn đồ thị (có hướng hoặc vô hướng). $V = \{1, \dots, n\}$ là tập các đỉnh, U là tập cạnh (cung). Với $s, t \in V$, tìm tất cả các đường đi từ s đến t .

Các thuật toán tìm kiếm cơ bản :

Thuật toán DFS : Tìm kiếm theo chiều sâu.

Thuật toán BFS : Tìm kiếm theo chiều rộng.

2. Thuật toán DFS (Depth First Search)

a) Ý tưởng

Thuật toán DFS tiến hành tìm kiếm trong đồ thị theo chiều sâu. Thuật toán thực hiện việc thăm tất cả các đỉnh có thể đạt được cho tới đỉnh t từ đỉnh s cho trước. Đỉnh được thăm càng muộn sẽ càng sớm được duyệt xong (cơ chế LIFO – Vào Sau Ra Trước). Nên thuật toán có thể tổ chức bằng một thủ tục đệ quy quay lui.

b) Mô tả thuật toán

Input $G = (V, U)$, s , t

Output Tất cả các đường đi từ s đến t (nếu có).

DFS (int s) \equiv

```

for ( u = 1; u <= n; u++)
{
    if (chấp nhận được)
    {
        Ghi nhận nó;
        if (u  $\neq$  t)
            DFS(u);
        else
            In đường đi;
        bỏ việc ghi nhận;
    }
}

```

c) Cài đặt

Ta cần mô tả dữ liệu đồ thị và các mệnh đề được phát biểu trong mô hình. Ma trận sẽ được biểu diễn bằng ma trận kề :

$$a_{ij} = \begin{cases} 1; (i, j) \in U \\ 0; (i, j) \notin U \end{cases}$$

Ghi nhận đỉnh được thăm để tránh trùng lặp khi quay lui bằng cách đánh dấu. Ta sử dụng một mảng một chiều Daxet[] với qui ước :

Daxet[u] = 1 , u đã được thăm.

Daxet[u] = 0 , u chưa được thăm.

Mảng Daxet[] lúc đầu khởi động bằng 0 tất cả.

Điều kiện chấp nhận được cho đỉnh u chính là u kề với v ($a_{vu} = 1$) và u chưa được thăm (Daxet[u] = 0).

Để ghi nhận các đỉnh trong đường đi, ta dùng một mảng một chiều Truoc[] , với qui ước :

Truoc[u] = $v \Leftrightarrow v$ là đỉnh đứng trước đỉnh u , và u kề với v

Ta khởi động mảng Truoc[] bằng 0 tất cả.

Thuật toán có thể viết lại như sau :

Input $G = (a_{ij})_{n \times n}$, s , t

Output Tất cả các đường đi từ s đến t (nếu có).

```
void DFS( s) ≡
    int u;
    daxet[s] = 1;
    for( u = 1; u <= n; u++)
    {
        if( a[s][u] && !daxet[u])
        {
            Truoc[u] = s;
            if ( u == t )
                Xuat_duongdi();
            else
                DFS(u);
            daxet[u] = 0;
        }
    }
```

Mảng `truoc[]` lưu trữ các đỉnh trên đường đi,

Nếu kết thúc thuật toán, $Daxet[t] = 0$ ($Truoc[t] = 0$) thì không có đường đi từ s đến t .

Trong trường hợp tồn tại đường đi, xuất đường đi chính là xuất mảng `Truoc[]`. Thao tác này có thể viết như sau :

```
Xuat_duongdi()≡
    cout<<t<<"<--";
    j = t;
    while ( truoc[j] != s)
    {
        cout<<truoc[j]<<"<--";
        j = truoc[j];
    }
    cout<<s<<endl;
```

Với đồ thị có hướng cho bởi ma trận kề :

```
7
0 0 0 1 0 1 1
0 0 1 1 0 0 0
0 1 0 1 0 1 0
1 0 1 0 0 0 0
0 0 0 0 1 1 0
1 0 0 0 1 0 0
1 0 0 1 0 0 0
```

Kết quả :

$s = 1, t = 4$	$s = 2, t = 5$
$4 \leftarrow 1$	$5 \leftarrow 6 \leftarrow 1 \leftarrow 4 \leftarrow 3 \leftarrow 2$
$4 \leftarrow 7 \leftarrow 1$	$5 \leftarrow 6 \leftarrow 3 \leftarrow 2$
	$5 \leftarrow 6 \leftarrow 1 \leftarrow 4 \leftarrow 2$
	$5 \leftarrow 6 \leftarrow 3 \leftarrow 4 \leftarrow 2$

3. Thuật toán BFS (Breadth First Search)

a) Ý tưởng

Thuật toán BFS tiến hành tìm kiếm trên đồ thị theo chiều rộng. Thuật toán thực hiện việc thăm tất cả các đỉnh có thể đạt được cho tới đỉnh t từ đỉnh s cho trước theo từng mức kề. Đỉnh được thăm càng sớm thì sẽ càng sớm được duyệt xong (cơ chế FIFO – Vào Trước Ra Trước).

b) Mô tả thuật toán

Input $G = (V, E)$,

$s, t \in V$;

Output

Đường đi từ s đến t .

Mô tả :

- Bước 0 : $A_0 = \{s\}$.
- Bước 1 : $A_1 = \{x \in V \setminus A_0 : (s, x) \in E\}$.
- Bước 2 : $A_2 = \{x \in V \setminus [A_0 \cup A_1] : \exists y \in A_1, (y, x) \in E\}$.
- ...
- Bước i : $A_i = \{x \in V \setminus \bigcup_{k=0}^{i-1} A_k : \exists y \in A_{i-1}, (y, x) \in E\}$.

...

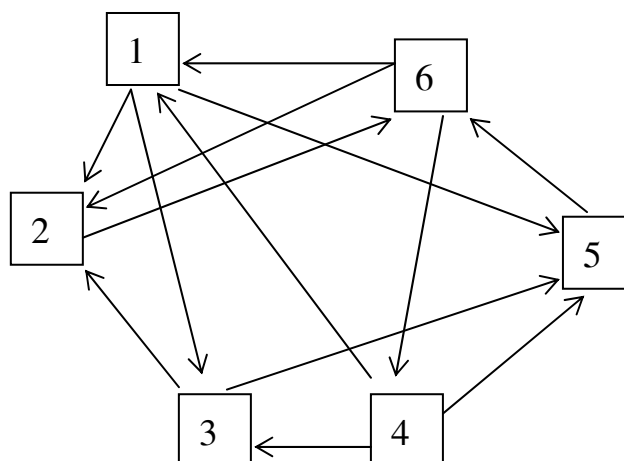
Thuật toán có không quá n bước lặp; một trong hai trường hợp sau xảy ra :

- Nếu với mọi i , $t \notin A_i$: không có đường đi từ s đến t ;
- Ngược lại, $t \in A(m)$ với m nào đó. Khi đó tồn tại đường đi từ s tới t , và đó là một đường đi ngắn nhất từ s đến t .

Trong trường hợp này, ta xác định được các đỉnh trên đường đi bằng cách quay ngược lại từ t đến các đỉnh trước t trong từng các tập trước cho đến khi gặp s .

Minh hoạ :

Cho đơn đồ thị có hướng :



Tìm đường đi từ đỉnh (1) đến đỉnh (4) :

$A(0) = \{1\};$

$A(1) = \{2,3,5\}$

$A(2) = \{6\}$

$A(3) = \{4\}$

Đường đi ngắn nhất tìm được là $4 \leftarrow 6 \leftarrow 5 \leftarrow 1$, có chiều dài là 3.

c) Cài đặt

Trong thuật toán BFS, đỉnh được thăm càng sớm sẽ càng sớm trở thành duyệt xong, nên các đỉnh được thăm sẽ được lưu trữ trong hàng đợi queue. Một đỉnh sẽ trở thành duyệt xong ngay sau khi ta xét xong tất cả các đỉnh kề của nó .

Ta dùng một mảng logic Daxet[] để đánh dấu các đỉnh được thăm, mảng này được khởi động bằng 0 tất cả để chỉ rằng lúc đầu chưa đỉnh nào được thăm.

Một mảng truoc[] để lưu trữ các đỉnh nằm trên đường đi ngắn nhất cần tìm (nếu có), với ý nghĩa Truoc[i] là đỉnh đứng trước đỉnh i trong đường đi. Mảng Truoc[] được khởi động bằng 0 tất cả để chỉ rằng lúc đầu chưa có đỉnh nào.

Đồ thị G được biểu diễn bằng ma trận kề $a = (a_{uv})_{n \times n}$

trong đó :
$$a_{uv} = \begin{cases} 1; (u, v) \in E; \\ 0; (u, v) \notin E; \end{cases}$$

Hàng đợi queue ta cài đặt bằng mảng . Thuật toán được cài đặt như sau :

```
BFS(s) ≡
int u, j, dauQ = 1, cuoiQ = 1;
queue[cuoiQ] = s;
Daxet[s] = 1;
while ( dauQ <= cuoiQ)
{
    u = queue[dauQ];
    dauQ++;
    for ( j = 1; j <= n; j++)
        if( a[u][j] == 1 && !Daxet[j] )
        {
            cuoiQ++;
```

```

        queue[cuoiQ] = j;
        Daxet[j] = 1;
        Truoc[j] = u;
    }
}

```

Nhận xét :

Ta có thể thấy mỗi lần gọi DFS(s), BFS(s) thì mọi đỉnh cùng thành phần liên thông với s sẽ được thăm, nên sau khi thực hiện hàm trên thì :

- $\text{Truoc}[t] == 0$: có nghĩa là không tồn tại đường đi từ s đến t,
- Ngược lại, có đường đi từ s đến t. Khi đó lời giải được cho bởi :

$$t \leftarrow p1 = \text{Truoc}[t] \leftarrow p2 = \text{Truoc}[p1] \leftarrow \dots \leftarrow s.$$

BÀI TẬP

Bài 1:

Cài đặt các thuật toán :

1. Liệt kê tất cả các dãy nhị phân độ dài n.
2. Liệt kê tất cả các hoán vị của n số nguyên dương đầu tiên.
3. Liệt kê tất cả các tổ hợp chập k trong tập gồm n số nguyên dương đầu tiên.
4. Giải bài toán ngựa đi tuần.
5. Giải bài toán n hậu.
6. DFS.
7. BFS.

Bài 2:

Cho dãy $a = (a_1, a_2, \dots, a_n)$ gồm các số đôi một khác nhau.

1. Liệt kê tất cả các hoán vị của dãy n phần tử của a.
2. Liệt kê tất cả các tổ hợp chập k trong tập gồm n phần tử của a.

Bài 3:

Giả sử ổ khóa có n công tắc. Mỗi công tắc có một trong 2 trạng thái “đóng” hay “mở”. Khóa mở được nếu có ít nhất $\lceil n/2 \rceil$ công tắc có trạng thái mở.

Liệt kê tất cả các cách mở khóa.

Bài 4 (Ngựa đi tuần).

Cho bàn cờ $n \times n$ ô. Một con ngựa được phép đi theo luật cờ vua.

Tìm hành trình của ngựa, bắt đầu từ ô $\langle x_0, y_0 \rangle$ đi qua tất cả các ô của bàn cờ, mỗi ô đúng một lần.

Liệt kê tất cả các hành trình.

Bài 5:

Cho bàn cờ $n \times n$ ô. Một con ngựa được phép đi theo luật cờ vua.

Tìm hành trình của ngựa, bắt đầu từ ô $\langle x_0, y_0 \rangle$, ô kết thúc $\langle x_1, y_1 \rangle$, mỗi ô trong hành trình ngựa đi qua đúng một lần.

1. Liệt kê tất cả các hành trình.
2. Chỉ ra hành trình ngắn nhất (có số ô trong hành trình ít nhất)

Bài 6 (Ngựa đi tuần).

Cho bàn cờ $n \times n$ ô. Một con ngựa được phép đi theo luật cờ tướng.

Tìm hành trình của ngựa, bắt đầu từ ô $\langle x_0, y_0 \rangle$ đi qua tất cả các ô của bàn cờ, mỗi ô đúng một lần.

Liệt kê tất cả các hành trình.

Bài 7 :

Một người du lịch muốn tham quan n thành phố T_1, T_2, \dots, T_n . Xuất phát từ một thành phố nào đó, người du lịch muốn đi qua tất cả các thành phố còn lại, mỗi thành phố đi qua đúng một lần rồi quay trở lại thành phố xuất phát.

Gọi C_{ij} là chi phí đi từ thành phố T_i đến thành phố T_j .

1. Liệt kê tất cả các hành trình đi từ thành phố T_i đến thành phố T_j thỏa yêu cầu bài toán và chi phí tương ứng.
2. Chỉ ra hành trình đi từ thành phố T_i đến thành phố T_j thỏa yêu cầu bài toán (nếu có) sao cho có chi phí ít nhất.

Bài 8 :

Cho một ma trận vuông cấp n , các phần tử của ma trận là các số tự nhiên. Ta nói đường đi trong ma trận là một đường gấp khúc không tự cắt xuất phát từ một ô nào đó của ma trận, sau đó có thể đi theo các hướng : lên trên, xuống dưới, rẽ trái, rẽ phải. Độ dài của đường đi là số ô nằm trong đường đi.

Với ô xuất phát cho trước :

1. Tìm một đường đi dài nhất trong ma trận, theo nghĩa có nhiều ô nhất trong đường đi.
2. Tìm đường đi dài nhất trong ma trận sao cho các ô trên đường đi lập thành một dãy số không giảm.

Bài 9 :

Cho $G = (V, E)$ là một đơn đồ thị, không có trọng số, trong đó $V = \{1, \dots, n\}$ là tập đỉnh, E là tập cạnh (hay cung).

1. Xác định số thành phần liên thông của G .
2. Xuất các đỉnh nằm trong mỗi thành phần liên thông.

Bài 10 :

Trên một mảnh đất hình vuông, ta chia thành $n \times n$ ô, mỗi ô ta ghi một số là 0 hoặc 1. Ô mang số 0 ta đào ao, mang số 1 ta trồng cỏ. Hai ô trồng cỏ có cạnh liền nhau được xem là cùng nằm trong bồn cỏ. Hãy xác định diện tích của bồn cỏ lớn nhất (theo nghĩa có số ô nhiều nhất).

Bài 11 :

Cho 3 ký tự A, B, C và n là một số nguyên dương.

1. Liệt kê tất cả các chuỗi tạo ra từ 3 ký tự trên, với chiều dài n.
2. Liệt kê tất cả các chuỗi tạo ra từ 3 ký tự trên, với chiều dài n, thỏa điều kiện không có 2 chuỗi con liên tiếp nào giống nhau.
3. Chỉ ra chuỗi thỏa (1) , (2) và sao cho số ký tự B là ít nhất.

Bài 12 :

Giả sử $A \subset N^*$, có n phần tử. Cho $S \in N^*$.

$$\text{Xác định : } D = \left\{ (x_1, \dots, x_n) \in N^n : S = \sum_{i=1}^n x_i a_i ; a_i \in A, \forall i = \overline{1, n} \right\}$$

Bài 13 :

Cho n xâu ký tự khác rỗng a_1, a_2, \dots, a_n , và một xâu ký tự S. Tìm cách biểu diễn S qua các xâu a_i , dưới dạng ghép xâu, mỗi xâu a_i có thể không xuất hiện trong S, hoặc xuất hiện trong S nhiều lần.

Liệt kê tất cả cách cách biểu diễn.

Bài 14 :

Có n đồ vật, mỗi vật i đặc trưng bởi trọng lượng W_i và giá trị sử dụng V_i , với mọi $i \in \{1, \dots, n\}$.

Cần chọn các vật này đặt vào một chiếc túi xách có giới hạn trọng lượng m, sao cho tổng giá trị sử dụng các vật được chọn là lớn nhất.

Bài 15 :

Xét bài toán tìm đường bay trong mạng giao thông hàng không.

Trong cơ sở dữ liệu các tuyến bay của một hãng hàng không, giả sử mỗi tuyến bay xác định bởi các thành phần :

- Thành phố xuất phát.
- Thành phố đích.
- Chiều dài đường bay

Với thành phố xuất phát và thành phố đích cho trước.

1. Liệt kê tất cả các đường bay.
2. Chỉ ra đường bay có chiều dài ngắn nhất.

CHƯƠNG 4: PHƯƠNG PHÁP NHÁNH CẬN (Branch And Bound)

I. Mở đầu

1. Ý tưởng

Phương pháp quay lui, vét cạn có thể giải các bài toán tối ưu, bằng cách lựa chọn phương án tối ưu trong tất cả các lời giải tìm được. Nhưng nhiều bài toán không gian các lời giải là quá lớn, nên áp dụng phương pháp quay lui khó đảm bảo về thời gian cũng như kỹ thuật. Cho nên ta cần phải cải tiến thuật toán quay lui để hạn chế bớt việc duyệt các phương án. Có nhiều cách cải tiến, trong đó có phương pháp nhánh cận.

Phương pháp nhánh cận là một cải tiến của phương pháp quay lui, dùng để tìm lời giải tối ưu của bài toán. Ý tưởng chính của nó như sau :

Trong quá trình duyệt ta luôn giữ lại một phương án mẫu (có thể xem là lời giải tối ưu cục bộ – chẳng hạn có giá nhỏ nhất tại thời điểm đó). Đánh giá nhánh cận là phương pháp tính giá của phương án ngay trong quá trình xây dựng các thành phần của phương án theo hướng đang xây dựng có thể tốt hơn phương án mẫu hay không. Nếu không ta lựa chọn theo hướng khác.

2. Mô hình

Giả sử bài toán tối ưu cho là :

Tìm $\text{Min}\{f(x) : x \in D\}$;

Với $X = \left\{ a = (a_1, \dots, a_n) \in \prod_{i=1}^n A_i : P(x) \right\}$; $|A_i| < \infty; \forall i = \overline{1, n}$. P là một tính

chất trên $\prod_{i=1}^n A_i$.

Nghiệm của bài toán nếu có sẽ được biểu diễn dưới dạng $x = (x_1, \dots, x_n)$

Trong quá trình liệt kê theo phương pháp quay lui, ta xây dựng dần các thành phần của nghiệm.

Một bộ phận i thành phần (x_1, \dots, x_i) sẽ gọi là một lời giải (phương án) bộ phận cấp i . Ta gọi X_i là tập các lời giải bộ phận cấp i , $\forall i = \overline{1, n}$.

Đánh giá cận là tìm một hàm g xác định trên các X_i sao cho :

$$g(x_1, \dots, x_i) \leq \text{Min}\{f(a) : a = (a_1, \dots, a_n) \in X, x_i = a_i, \forall i = \overline{1, i}\}$$

Bất đẳng thức này có nghĩa là giá trị $g(x_1, \dots, x_i)$ không lớn hơn giá trị của các phương án mở rộng từ lời giải bộ phận (x_1, \dots, x_i) .

Sau khi tìm được hàm đánh giá cận g , ta dùng g để giảm bớt chi phí duyệt các phương án theo phương pháp quay lui.

Giả sử x^* là lời giải tốt nhất hiện có (phương án mẫu), còn f^* là giá trị tốt nhất tương ứng $f^* = f(x^*)$.

Nếu $g(x_1, \dots, x_i) > f^*$ thì :

$$f^* < g(x_1, \dots, x_i) \leq \text{Min}\{f(a) : a = (a_1, \dots, a_n) \in X, x_i = a_i, \forall i = \overline{1, i}\}$$

Nên chắc rằng các lời giải mở rộng từ (x_1, \dots, x_i) sẽ không tốt hơn phương án mẫu, do đó có thể bỏ đi không cần phát triển lời giải bộ phận (x_1, \dots, x_i) để tìm lời giải tối ưu của bài toán.

Thủ tục quay lui sửa lại thành thủ tục nhánh cận như sau :

```
Try (i) ≡
  for (j = 1 → n)
    if( Chấp nhận được )
      {
        Xác định  $x_i$  theo j;
        Ghi nhận trạng thái mới;
        if(i == n)
          Cập nhật lời giải tối ưu ;
        else
          {
            Xác định cận  $g(x_1, \dots, x_i)$  ;
            if(  $g(x_1, \dots, x_i) \leq f^*$  )
              Try (i+1);
          }
        // Trả bài toán về trạng thái cũ
      }
  }
```

Thực chất của phương pháp nhánh cận là tìm kiếm theo chiều sâu trên cây liệt kê lời giải như phương pháp quay lui, chỉ khác có một điều là khi tìm được x_i mà đánh giá cận $g(x_1, \dots, x_i) > f^*$ thì ta cắt bỏ các nhánh con từ x_i đi xuống, mà quay lên ngay cha của nó là x_{i-1} .

Vấn đề là xác định hàm đánh giá cận như thế nào ?

II. Bài toán người du lịch

1. Bài toán

Một người du lịch muốn tham quan n thành phố T_1, \dots, T_n . Xuất phát từ một thành phố nào đó, người du lịch muốn đi qua tất cả các thành phố còn lại, mỗi thành phố đi qua đúng 1 lần rồi quay trở lại thành phố xuất phát.

Gọi C_{ij} là chi phí đi từ thành phố T_i đến T_j . Hãy tìm một hành trình thỏa yêu cầu bài toán sao cho chi phí là nhỏ nhất.

2. Ý tưởng

Gọi π là một hoán vị của $\{1, \dots, n\}$ thì một hành trình thỏa yêu cầu bài toán có dạng : $T_{\pi(1)} \rightarrow T_{\pi(2)} \rightarrow \dots \rightarrow T_{\pi(n)}$.

Nên có tất cả $n!$ hành trình như thế.

Nếu ta cố định một thành phố xuất phát, chẳng hạn T_1 , thì có $(n-1)!$ hành trình.

Bài toán chuyển về dạng :

Tìm $\text{Min}\{f(a_2, \dots, a_n) : (a_2, \dots, a_n) \text{ là hoán vị của } \{2, \dots, n\}\}$.

Với $f(a_1, \dots, a_n) = C_{1,a_2} + C_{a_2,a_3} + \dots + C_{a_{n-1},a_n} + C_{a_n,1}$

Cách giải bài toán sẽ kết hợp đánh giá nhánh cận trong quá trình liệt kê phương án của thuật toán quay lui.

3. Thiết kế

Input $C = (C_{ij})$

Output - $x^* = (x_1, \dots, x_n)$ // Hành trình tối ưu

- $f^* = f(x^*)$ // Giá trị tối ưu

Try (i) \equiv

for ($j = 1 \rightarrow n$)

if(Chấp nhận được)

{

Xác định x_i theo j ;

Ghi nhận trạng thái mới;

if($i == n$)

Cập nhật lời giải tối ưu;

else

{

Xác định cận $g(x_1, \dots, x_i)$;

if($g(x_1, \dots, x_i) \leq f^*$)

Try ($i+1$);

}

// Trả bài toán về trạng thái cũ

}

- Nếu ta cố định xuất phát từ T_1 , ta duyệt vòng lặp từ $j = 2$.

- Đánh giá nhánh cận :

Đặt : $CMin = \text{Min}\{C_{ij} : i, j \in \{1, \dots, n\}\}$

Giả sử vào bước i ta tìm được lời giải bộ phận cấp i là (x_1, \dots, x_i) , tức là đã đi qua đoạn đường $T_1 \rightarrow T_2 \rightarrow \dots \rightarrow T_i$, tương ứng với chi phí :

$$S_i = C_{1x_2} + C_{x_2x_3} + \dots + C_{x_{i-1}x_i}$$

Để phát triển hành trình bộ phận này thành một hành trình đầy đủ, ta còn phải đi qua $n-i+1$ đoạn đường nữa, gồm $n-i$ thành phố còn lại và đoạn quay lại T_1 .

Do chi phí mỗi một trong $n-i+1$ đoạn còn lại không nhỏ hơn $CMin$, nên hàm đánh giá cận có thể xác định như sau :

$$g(x_1, \dots, x_i) = S_i + (n - i + 1)CMin$$

- Điều kiện chấp nhận được của j là thành phố T_j chưa đi qua.

Ta dùng một mảng logic $Daxet[]$ để biểu diễn trạng thái này

$$Daxet[j] = \begin{cases} 1; T_j \text{ đã được đi qua} \\ 0; T_j \text{ chưa được đi qua} \end{cases}$$

Mảng $Daxet[]$ phải được bằng 0 tất cả.

- Xác định x_i theo j bằng câu lệnh gán : $x_i = j$

Cập nhật trạng thái mới : $Daxet[j] = 1$.

Cập nhật lại chi phí sau khi tìm được x_i : $S = S + C_{x_{i-1}x_i}$

- Cập nhật lời giải tối ưu :

Tính chi phí hành trình vừa tìm được :

$$Tong = S + C_{x_n 1};$$

Nếu ($Tong < f^*$) thì

$$Lgtu = x;$$

$$f^* = Tong;$$

- Thao tác huỷ bỏ trạng thái : $Daxet[j] = 0$.

Trả lại chi phí cũ : $S = S - C_{x_{i-1}x_i}$

Thủ tục nhánh cận viết lại như sau :

Try(i)≡

for ($j = 2 \rightarrow n$)

if ($\neg Daxet[j]$)

{

$x[i] = j$;

$Daxet[j] = 1$;

$S = S + C[x[i-1]][x[i]]$;

if ($i == n$) //Cập nhật tối ưu

{

$Tong = S + C[x[n]][x[1]]$;

if ($Tong < f^*$)

{

$Lgtu = x$;

$f^* = Tong$;

}

}

else

{

$g = S + (n-i+1) * Cmin$; //Đánh giá cận

if ($g < f^*$)

Try(i+1);

}

$S = S - C[x[i-1]][x[i]]$;

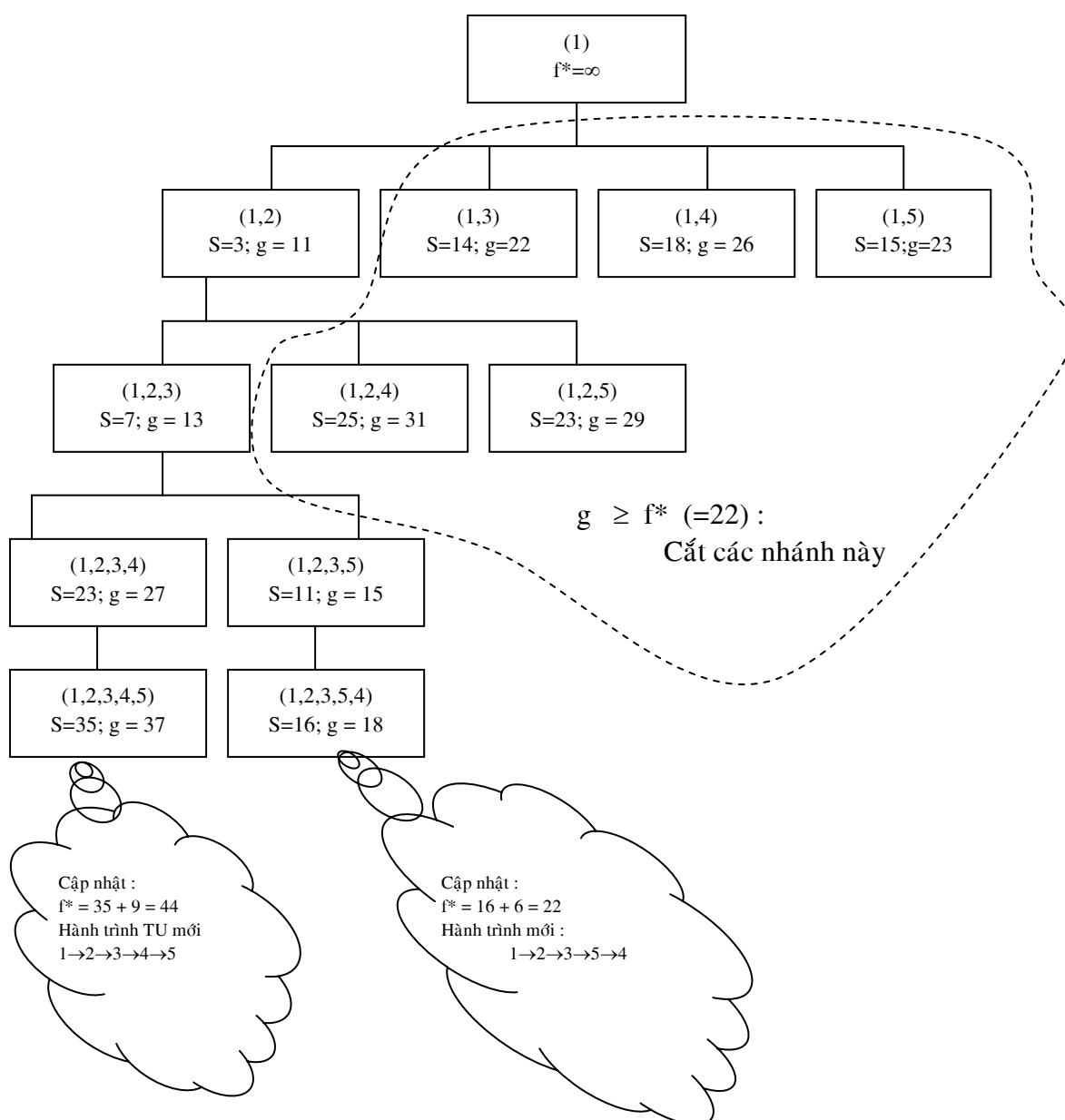
$Daxet[j] = 0$;

}

Minh họa :

Ma trận chi phí :

$$C = \begin{bmatrix} \infty & 3 & 14 & 18 & 15 \\ 3 & \infty & 4 & 22 & 20 \\ 17 & 9 & \infty & 16 & 4 \\ 6 & 2 & 7 & \infty & 12 \\ 9 & 15 & 11 & 5 & \infty \end{bmatrix}$$



4. Cài đặt

```
void Try(int i)
{
    int j, Tong, g;
    for (j = 2; j <= n; j++)
        if (!Daxet[j])
        {
            x[i] = j;
            Daxet[j] = 1;
            S = S + C[x[i-1]][x[i]];
            if (i==n) //Cap nhat hanh trinh toi uu
            {
                Tong = S + C[x[n]][x[1]];
            }
        }
    }
}
```

```

        if(Tong < Gttu)
        {
            Gan(Httu,x,n);
            Gttu = Tong;
        }
    }
    else
    {
        g = S + (n-i+1)*Cmin; //Danh gia can
        if ( g < Gttu)
            Try(i+1);
    }
    S = S - C[x[i-1]][x[i]];
    Daxet[j] = 0;
}
}

Khởi động các biến :
void Init()
{
    int i, j;
    Cmin = VC; //Chi phí nhỏ nhất giữa 2 thành phần
    for(i = 1; i <= n; i++)
        Daxet[i] = 0;
    for(i = 1; i <= n; i++)
        for(j = 1; j <= n; j++)
            if(Cmin > C[i][j])
                Cmin = C[i][j];
    Gttu = VC; //Giá trị tối ưu f*
    S = 0;
    x[1] = 1; // Xuất phát từ đỉnh 1
}

```

III. Bài toán cái túi xách

1. Bài toán

Có n loại đồ vật, mỗi loại có số lượng không hạn chế. Đồ vật loại i , đặc trưng bởi trọng lượng W_i và giá trị sử dụng V_i , với mọi $i \in \{1, \dots, n\}$.

Cần chọn các vật này đặt vào một chiếc túi xách có giới hạn trọng lượng m , sao cho tổng giá trị sử dụng các vật được chọn là lớn nhất.

2. Ý tưởng

$$\text{Đặt : } D = \left\{ u = (u_1, \dots, u_n) \in N^n : \sum_{i=1}^n u_i w_i \leq m \right\}$$

và : $f : D \rightarrow R^+$

$$(u_1, \dots, u_n) \mapsto f(u_1, \dots, u_n) = \sum_{i=1}^n u_i v_i ; (u_1, \dots, u_n) \in D$$

Bài toán chiếc túi xách chuyển về bài toán sau :

Tìm $x^* \in D : f^* = f(x^*) = \{f(u) : u \in D\}$

Cho nên ta sẽ kết hợp đánh giá nhánh cận trong quá trình liệt kê các lời giải theo phương pháp quay lui.

3. Thiết kế thuật toán

Mô hình ban đầu có thể sử dụng như sau :

Try(i) \equiv

for(j = 1 \rightarrow t)

if(Chấp nhận được)

{

Xác định x_i theo j;

Ghi nhận trạng thái mới;

if(i==n)

Cập nhật lời giải tối ưu;

else

{

Xác định cận trên g;

if($g(x_1, \dots, x_i) \leq f^*$)

Try(i+1);

}

Trả lại trạng thái cũ cho bài toán;

}

- Cách chọn vật :

$$\text{Xét mảng đơn giá : } Dg = \left(\frac{v_1}{w_1}, \dots, \frac{v_n}{w_n} \right)$$

Ta chọn vật theo đơn giá giảm dần.

Không mất tính tổng quát, ta giả sử các loại vật cho theo thứ tự giảm dần của đơn giá.

- Đánh giá cận trên :

Giả sử đã tìm được lời giải bộ phận : (x_1, \dots, x_i) . Khi đó :

$$\text{- Giá trị của túi xách thu được : } S = \sum_{j=1}^i x_j v_j = S + x_i v_i.$$

- Tương ứng với trọng lượng các vật đã được xếp vào chiếc túi :

$$TL = \sum_{j=1}^i x_j w_j = TL + x_i w_i.$$

$$\text{- Do đó, giới hạn trọng lượng của chiếc túi còn lại là : } m - TL = m - \sum_{j=1}^i x_j w_j.$$

Ta có :

$$\begin{aligned} & \text{Max}\{f(u) : u = (u_1, \dots, u_n) \in D; u_j = x_j, \forall j = \overline{1, i}\} = \\ & \text{Max}\left\{S + \sum_{j=i+1}^n u_j v_j : \sum_{j=i+1}^n u_j w_j \leq m_i\right\} = \\ & S + \text{Max}\left\{\sum_{j=i+1}^n u_j v_j : \sum_{j=i+1}^n u_j w_j \leq m_i\right\} \leq S + v_{i+1} * \left(\frac{m_i}{w_{i+1}}\right) \end{aligned}$$

Do đó, cận trên cho các lời giải bộ phận cấp i có thể xác định bởi :

$$g(x_1, \dots, x_i) = S + v_{i+1} * \left(\frac{m_i}{w_{i+1}}\right)$$

- Theo biểu thức xác định cận trên g , các giá trị có thể chấp được cho x_{j+1} là :

$$t = 0 \rightarrow \left(\frac{m_i}{w_{i+1}}\right)$$

- Thao tác ghi nhận trạng thái mới khi xác định được x_i chẳng qua là cập nhật lại giá trị thu được và giới hạn trọng lượng mới của chiếc túi :

$$S = S + x_i v_i$$

$$T = T + x_i w_i .$$

- Vì vậy, thao tác trả lại trạng thái cũ cho bài toán :

$$S = S - x_i v_i$$

$$T = T - x_i w_i .$$

- Cập nhật lời giải tối ưu :

Khi tìm được một lời giải, ta so sánh lời giải này với lời giải mà ta coi là tốt nhất vào thời điểm hiện tại để chọn lời giải tối ưu.

- Các khởi tạo giá trị ban đầu :

- $x^* = 0$; //Lời giải tối ưu của bài toán

- $f^* = f(x^*) = 0$; // Giá trị tối ưu

- $S = 0$; //Giá trị thu được từng bước của chiếc túi.

- $TL =$; //Trọng lượng xếp vào chiếc túi từng bước.

Ta viết lại thủ tục nhánh cận trên :

Input m ,

$$v=(v_1, \dots, v_n) : v_i \in \mathbb{R}, \forall i;$$

$$w=(w_1, \dots, w_n) : w_i \in \mathbb{R}, \forall i;$$

Output $x^* = (x_1, \dots, x_n) : x_i \in \mathbb{N}, \forall i;$

$$f^* = f(x^*) = \text{Max}\left\{\sum_{i=1}^n u_i v_i : \sum_{i=1}^n u_i w_i \leq m, u_i \in \mathbb{N}, \forall i \in \overline{1, n}\right\}$$

Try(i)≡

$$t = (m - TL) / w_i ;$$

for ($j = t$; $j \geq 0$; $j--$)

{

$$x_i = j;$$

$$TL = TL + w_i * x_i ;$$

$$S = S + v_i * x_i;$$

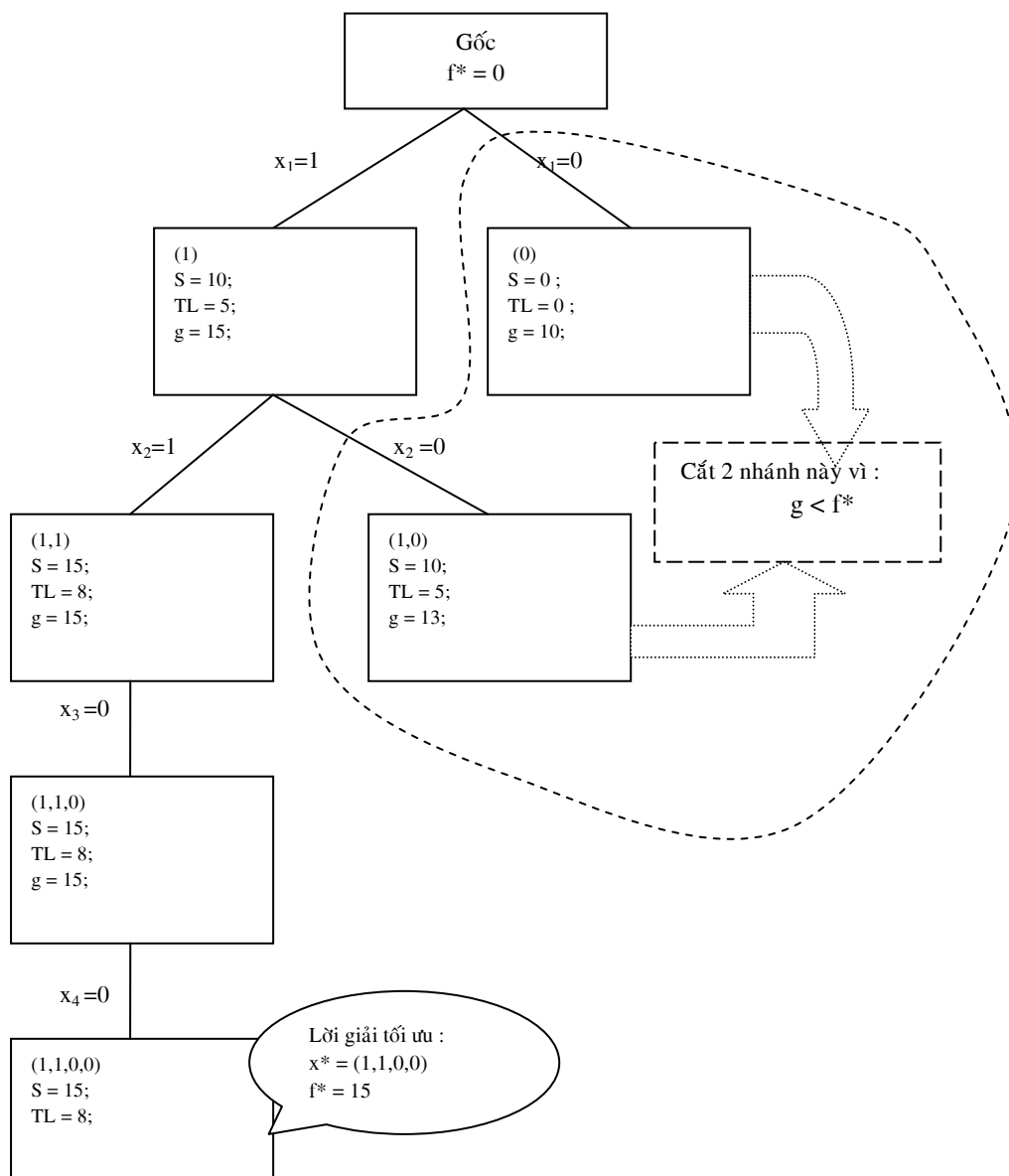
```

if(i==n)      //Cap nhat toi uu
{
    if(S > f*)
    {
        x* = x;
        f* = S;
    }
}
else
{
    g = S + vi+1*(m-TL)/wi+1; //Danh gia can
    if ( g > f*)
        Try(i+1);
}
TL = TL - wi*xi;
S = S - vi*xi;
}

```

Minh họa :

i	1	2	3	4
w	5	3	2	4
v	10	5	3	6
m = 8				



4. Cài đặt

```
void Try(int i)
{
    int j, t, g;
    t = (int)((m-TL)/w[i]);
    for (j = t; j >= 0; j--)
    {
        x[i] = j;
        TL = TL + w[i]*x[i]; //Trong lượng thu được
        S = S + v[i]*x[i]; //Giá trị thu được
        if(i==n) //Cập nhật tối ưu
        {
```

```

        if(S > Gttu)
        {
            Gan();
            Gttu = S;
        }
    }
    else
    {
        g = S + v[i+1]*(m-Tl)/w[i+1]; //Danh gia can
        if ( g > Gttu)
            Try(i+1);
    }
    Tl = Tl - w[i]*x[i];
    S = S - v[i]*x[i];
}
}
//*****
void Init()
{
    for (int i = 1; i <= n; i++)
    {
        Patu[i] = 0;
        x[i] = 0;
    }
    S = 0;
    Gttu = 0;
    Tl = 0;
}

```

BÀI TẬP

Bài 1 :

Có n đồ vật, mỗi vật i đặc trưng bởi trọng lượng w_i và giá trị sử dụng v_i , với mọi $i \in \{1, \dots, n\}$.

Cần chọn các vật này đặt vào một chiếc túi xách có giới hạn trọng lượng m , sao cho tổng giá trị sử dụng các vật được chọn là lớn nhất.

Bài 2 :

Cho 3 ký tự A, B, C và n là một số nguyên dương.

Xác định chuỗi thỏa tạo ra từ 3 ký tự trên, với chiều dài n , thỏa điều kiện không có 2 chuỗi con liên tiếp nào giống nhau và sao cho số ký tự B là ít nhất.

Bài 3 :

Tìm lời giải tối ưu bài toán :

$$\begin{cases} 10x_1 + 5x_2 + 3x_3 + 6x_4 \rightarrow \text{Max} \\ 5x_1 + 3x_2 + 2x_3 + 4x_4 \leq 8 \\ x_1, x_2, x_3, x_4 \in \mathbb{N} \end{cases}$$

Bài 4 :

Giả sử có n công việc và n thợ. Chi phí trả cho người thợ i để làm công việc j là C_{ij} . Mỗi công việc chỉ do một thợ thực hiện và ngược lại.

Tìm cách thuê các thợ làm việc sao cho tổng chi phí là nhỏ nhất.

CHƯƠNG 5: PHƯƠNG PHÁP THAM LAM

(The greedy method)

I. Mở đầu

1. Ý tưởng

Phương pháp tham lam là kỹ thuật thiết kế thường được dùng để giải các bài toán tối ưu. Phương pháp được tiến hành trong nhiều bước. Tại mỗi bước, theo một chọn lựa nào đó (xác định bằng một hàm chọn), sẽ tìm một lời giải tối ưu cho bài toán nhỏ tương ứng. Lời giải của bài toán được bổ sung dần từng bước từ lời giải của các bài toán con.

Lời giải được xây dựng như thế có chắc là lời giải tối ưu của bài toán ?

Các lời giải tìm được bằng phương pháp tham lam thường chỉ là chấp nhận được theo điều kiện nào đó, chưa chắc là tối ưu.

Cho trước một tập A gồm n đối tượng, ta cần phải chọn một tập con S của A . Với một tập con S được chọn ra thỏa mãn các yêu cầu của bài toán, ta gọi là một nghiệm chấp nhận được. Một hàm mục tiêu gắn mỗi nghiệm chấp nhận được với một giá trị. Nghiệm tối ưu là nghiệm chấp nhận được mà tại đó hàm mục tiêu đạt giá trị nhỏ nhất (lớn nhất).

Đặc trưng tham lam của phương pháp thể hiện bởi : trong mỗi bước việc xử lý sẽ tuân theo một sự chọn lựa trước, không kể đến tình trạng không tốt có thể xảy ra khi thực hiện lựa chọn lúc đầu.

2. Mô hình

Chọn S từ tập A .

Tính chất tham lam của thuật toán định hướng bởi hàm Chọn.

- Khởi động $S = \emptyset$;
- Trong khi $A \neq \emptyset$:
 - Chọn phần tử tốt nhất của A gắn vào $x : x = \text{Chọn}(A)$;
 - Cập nhật các đối tượng để chọn : $A = A - \{x\}$;
 - Nếu $S \cup \{x\}$ thỏa mãn yêu cầu bài toán thì

Cập nhật lời giải : $S = S \cup \{x\}$;

Thủ tục thuật toán tham lam có thể cài đặt như sau :

```

input A[1..n]
output S //lời giải;
greedy (A,n) ≡
  S = ∅;
  while ( A ≠ ∅)
  {
    x= Chọn(A);
    A = A - {x}
    if( S ∪ {x} chấp nhận được )
      S = S ∪ {x};
  }

```

return S;

II. Bài toán người du lịch

1. Bài toán

Một người du lịch muốn tham quan n thành phố T_1, \dots, T_n . Xuất phát từ một thành phố nào đó, người du lịch muốn đi qua tất cả các thành phố còn lại, mỗi thành phố đi qua đúng 1 lần rồi quay trở lại thành phố xuất phát.

Gọi C_{ij} là chi phí đi từ thành phố T_i đến T_j . Hãy tìm một hành trình thỏa yêu cầu bài toán sao cho chi phí là nhỏ nhất.

2. Ý tưởng

Đây là bài toán tìm chu trình có trọng số nhỏ nhất trong một đơn đồ thị có hướng có trọng số.

Thuật toán tham lam cho bài toán là chọn thành phố có chi phí nhỏ nhất tính từ thành phố hiện thời đến các thành phố chưa qua.

3. Thuật toán

Input $C = (C_{ij})$

output TOUR //Hành trình tối ưu,

COST; //Chi phí tương ứng

Mô tả :

TOUR := 0; COST := 0; $v := u$; // Khởi tạo

$\forall k := 1 \rightarrow n$://Thăm tất cả các thành phố

// Chọn cạnh kế)

- Chọn $\langle v, w \rangle$ là đoạn nối 2 thành phố có chi phí nhỏ nhất tính từ TP v đến các thành phố chưa qua.

- TOUR := TOUR + $\langle v, w \rangle$; //Cập nhật lời giải

- COST := COST + C_{vw} ; //Cập nhật chi phí

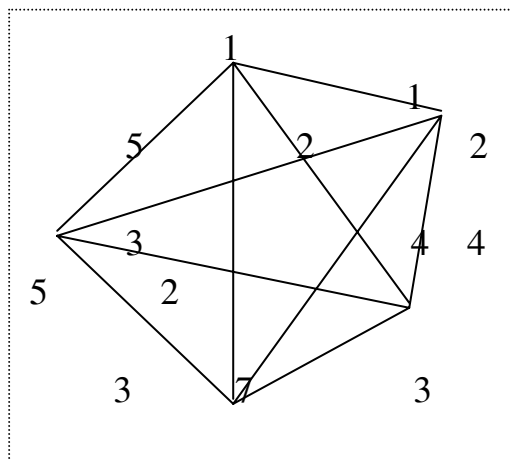
// Chuyển đi hoàn thành

TOUR := TOUR + $\langle v, u \rangle$;

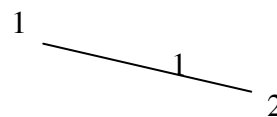
COST := COST + C_{vu} ;

Minh hoạ :

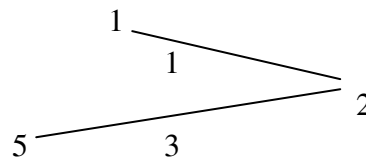
$$C = \begin{bmatrix} 0 & 1 & 2 & 7 & 5 \\ 1 & 0 & 4 & 4 & 3 \\ 2 & 4 & 0 & 1 & 2 \\ 7 & 4 & 1 & 0 & 3 \\ 5 & 3 & 2 & 3 & 0 \end{bmatrix}$$



1. TOUR := 0; COST := 0; u := 1;
 $\Rightarrow w = 2$;

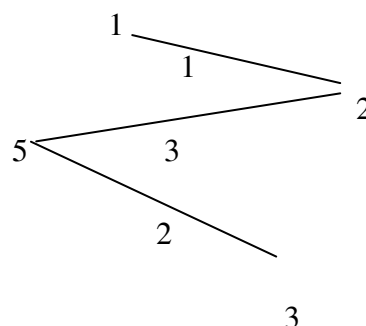


2. TOUR := <1,2>; COST := 1; u := 2;
 $\Rightarrow w = 5$;



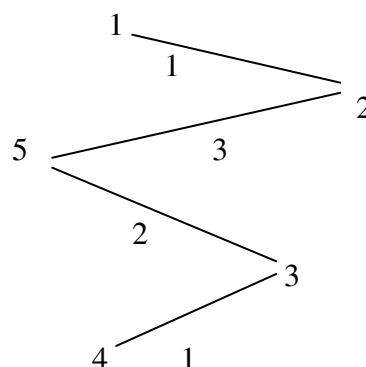
3. TOUR := {<1,2>, <2,5>}
 COST := 4; u := 5;

$\Rightarrow w = 3$;



4. TOUR := {<1,2>, <2,5>, <5,3>}
 COST := 6; u := 3;

$\Rightarrow w = 4$;



5. TOUR := {<1,2>, <2,5>, <5,3>, <3,4>}
 COST := 7; u := 1;

TOUR := {<1,2>, <2,5>, <5,3>, <3,4>, <4,1>}
 COST := 14

4. Độ phức tạp của thuật toán

Thao tác chọn đỉnh thích hợp trong n đỉnh được tổ chức bằng một vòng lặp để duyệt. Nên chi phí cho thuật toán xác định bởi 2 vòng lặp lồng nhau, nên $T(n) \in O(n^2)$.

5. Cài đặt

int GTS (mat a, int n, int TOUR[max], int Ddau)

```

{
    int    v,      //Dinh dang xet
           k,      //Duyet qua n dinh de chon
           w;      //Dinh duoc chon trong moi buoc
    int    mini;   //Chon min cac canh(cung) trong moi buoc
    int    COST;   //Trong so nho nhat cua chu trinh
    int    daxet[max]; //Danh dau cac dinh da duoc su dung
    for(k = 1; k <= n; k++)
        daxet[k] = 0; //Chua dinh nao duoc xet
    COST = 0;      //Luc dau, gia tri COST == 0

    int i; // Bien dem, dem tim du n dinh thi dung
    v = Ddau; //Chon dinh xuat phat la 1
    i = 1;
    TOUR[i] = v; //Dua v vao chu trinh
    daxet[v] = 1; //Dinh v da duoc xet

    while(i < n)
    {
        mini = VC;
        for (k = 1; k <= n; k++)
            if(!daxet[k])
                if(mini > a[v][k])
                {
                    mini = a[v][k];
                    w = k;
                }

        v = w;
        i++;
        TOUR[i] = v;
        daxet[v] = 1;
        COST += mini;
    }
    COST += a[v][Ddau];
    return COST;
}

```

III. Thuật toán Dijkstra -Tìm đường đi ngắn nhất trong đồ thị có trọng số

1. Bài toán

Cho $G = (V, E)$ là đơn đồ thị liên thông (vô hướng hoặc có hướng) có trọng số, $V = \{1, \dots, n\}$ là tập các đỉnh, E là tập các cạnh (cung).

Cho $s_0 \in E$. Tìm đường đi ngắn nhất đi từ s_0 đến các đỉnh còn lại.

Giải bài toán trên bằng thuật toán Dijkstra .

2. Ý tưởng

Thuật toán Dijkstra cho phép tìm đường đi ngắn nhất từ một đỉnh s đến các đỉnh còn lại của đồ thị và chiều dài (trọng số) tương ứng.

Phương pháp của thuật toán là xác định tuần tự đỉnh có chiều dài đến s theo thứ tự tăng dần.

Thuật toán được xây dựng trên cơ sở gán cho mỗi đỉnh các nhãn tạm thời. Nhãn tạm thời của các đỉnh cho biết cận trên của chiều dài đường đi ngắn nhất từ s đến đỉnh đó. Nhãn của các đỉnh sẽ biến đổi trong các bước lặp, mà ở mỗi bước lặp sẽ có một nhãn tạm thời trở thành chính thức. Nếu nhãn của một đỉnh nào đó trở thành chính thức thì đó cũng chính là chiều dài ngắn nhất của đường đi từ s đến đỉnh đó.

3. Mô tả thuật toán

Ký hiệu :

* $L(v)$ để chỉ nhãn của đỉnh v , tức là cận trên của chiều dài đường đi ngắn nhất từ s_0 đến v .

* $d(s_0, v)$: chiều dài đường đi ngắn nhất từ s_0 đến v .

* $m(s_0, v)$ là trọng số của cung (cạnh) (s, v) .

Thuật toán Dijkstra tìm chiều dài đường đi ngắn nhất từ đỉnh s đến $n-1$ đỉnh còn lại được mô tả như sau :

input : G, s_0

Output : $d(s_0, v), \forall v \neq s_0$;

Mô tả :

- Khởi động :

$L(v) = \infty, \forall v \neq s_0$; //Nhãn tạm thời

$S = \emptyset$; //Tập lưu trữ các đỉnh có nhãn chính thức

- Bước 0 :

$d(s_0, s_0) = L(s_0) = 0$;

$S = \{s_0\}$; // s_0 có Nhãn chính thức

- Bước 1:

- Tính lại nhãn tạm thời $L(v), v \notin S$:

Nếu v kề với s_0 thì

$L(v) = \min\{L(v), L(s_0) + m(s_0, v)\}$;

- Tìm $s_1 \notin S$ và kề với s_0 sao cho :

$L(s_1) = \min\{L(v) : v \notin S, \}$;

(Khi đó : $d(s_0, s_1) = L(s_1)$)

- $S = S \cup \{s_1\}$; // $S = \{s_0, s_1\}$; s_1 có nhãn chính thức

- Bước 2:

- Tính lại nhãn tạm thời $L(v), v \notin S$:

Nếu v kề với s_1 thì

$L(v) = \min\{L(v), L(s_1) + m(s_1, v)\}$;

- Tìm $s_2 \notin S$ và kề với s_1 hoặc s_0 sao cho :

$$L(s_2) = \min\{L(v) : \forall v \notin S\};$$

$$(\text{ Khi đó : } d(s, s_2) = L(s_2)); // 0 = d(s_0, s_1) \leq d(s_0, s_2)$$

Nếu $L(s_2) = \min\{L(s_j), L(s_j) + m(s_j, s_2)\}$ thì đường đi từ s đến s_2 đi qua đỉnh s_j là ngắn nhất, và s_j là đỉnh đứng kề trước s_2 .

- $S = S \cup \{s_2\}$; // $S = \{s_0, s_1, s_2\}$; // s_2 có nhãn chính thức

...

• Bước i:

- Tính lại nhãn tạm thời $L(v)$, $v \notin S$:

Nếu v kề với s_{i-1} thì

$$L(v) = \min\{L(v), L(s_{i-1}) + m(s_{i-1}, v)\};$$

- Tìm $s_i \notin S$ và kề với $s_j, j = \overline{0, i-1}$ sao cho :

$$L(s_i) = \min\{L(v) : \forall v \notin S\};$$

$$(\text{ d}(s, s_i) = L(s_i)); // 0 = d(s, s_1) \leq d(s, s_2) \leq \dots \leq d(s, s_i)$$

Nếu $L(s_i) = \min\{L(s_j), L(s_j) + m(s_j, s_i)\}$ thì đường đi ngắn nhất từ s đến s_i đi qua đỉnh s_j , và s_j là đỉnh đứng kề trước s_i .

- $S = S \cup \{s_i\}$; // $S = \{s_0, s_1, \dots, s_i\}$; // s_i có nhãn chính thức

Thuật toán dừng khi $i = n-1$;

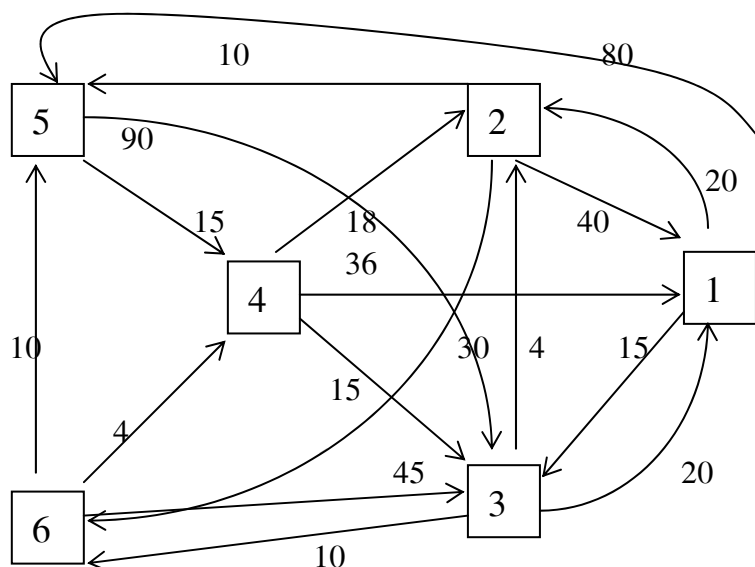
Khi thuật toán kết thúc, ta có : $0 = d(s_0, s_0) \leq d(s_0, s_1) \leq d(s_0, s_2) \leq \dots \leq d(s_0, s_{n-1})$

1)

Nếu chỉ tìm đường đi ngắn nhất từ s_0 đến t , thì thuật toán dừng khi có $t \in S$.

Tính chất tham lam của thuật toán Dijkstra là tại mỗi bước, chọn $s_i \notin S$ và s_i là đỉnh kề với s_j , với $j = \overline{0, i-1}$ sao cho $L(s_i) = \min\{L(v) : \forall v \notin S\}$.

Minh hoạ : Xét đồ thị có hướng G :



Đường đi ngắn nhất từ đỉnh $s = 1$ đến các đỉnh còn lại :

Bước 0 :

$$L(s) = \infty; \forall s$$

$$d(1,1) = L(1) = 0;$$

$$S = \{1\};$$

Bước 1 :

- Tính nhãn tạm thời $L(v)$, $v \notin S$:

* Các đỉnh $\notin S$ và kề với 1 là 2,3,5 :

$$L(2) = \text{Min}\{L(2), L(1)+m(1,2)\} = 20.$$

$$L(3) = \text{Min}\{L(3), L(1)+m(1,3)\} = 15.$$

$$L(5) = \text{Min}\{L(5), L(1)+m(1,5)\} = 80.$$

* Các đỉnh $\notin S$ và không kề với 1 là 4,6 :

$$L(4) = L(6) = \infty.$$

- Tìm $s_1 \notin S$ và kề với 1 sao cho : $L(s_1) = \text{Min}\{L(v) : \forall v \notin S\}$;

$$L(3) = \text{Min}\{L(v) : \forall v \notin S\} = 15.$$

Đường đi từ 1 đến 3 xác định bởi : $1 \rightarrow 3$ là ngắn nhất trong tất cả các đường đi từ 1 đến các đỉnh khác và $d(1,3) = L(3) = 15$.

$$- S = S \cup \{3\}; // S = \{1,3\}$$

Bước 2 :

- Tính nhãn tạm thời $L(v)$, $v \notin S$:

* Các đỉnh $\notin S$ và kề với 3 là 2,6 :

$$L(2) = \text{Min}\{L(2), L(3)+m(3,2)\} = \text{Min}\{20, 15+4\} = 19.$$

$$L(6) = \text{Min}\{L(6), L(3)+m(3,6)\} = \text{Min}\{\infty, 15+10\} = 25.$$

$$L(4) = \infty.$$

$$L(5) = 80 // \text{Đã tính ở bước 1.}$$

- Tìm $s_2 \notin S$ và kề với 1 hoặc 3 sao cho : $L(s_2) = \text{Min}\{L(v) : \forall v \notin S\}$;

$$L(2) = \text{Min}\{L(v) : \forall v \notin S\} = \text{Min}\{L(2), L(3)+m(3,2)\} = 19.$$

Đường đi từ 1 đến 2 xác định bởi : $1 \rightarrow 3 \rightarrow 2$ là ngắn nhất trong tất cả các đường đi từ 1 đến các đỉnh $j \neq 3$ và : $d(1,2) = L(2) = 19$.

$$- S = S \cup \{2\}; // S = \{1,3,2\}$$

... tương tự, ta có kết quả tính toán sau đây :

Bước lập	Đường đi ngắn nhất là đường đi từ đỉnh 1	đến đỉnh	Chiều dài của đường đi ngắn nhất từ đỉnh s (=1) đến các đỉnh khác : tsnn[]					
			1	2	3	4	5	6
Bước1	$1 \rightarrow 3$	3	-	20	15	∞	80	∞
Bước2	$1 \rightarrow 3 \rightarrow 2$	2	-	19	-			25
Bước3	$1 \rightarrow 3 \rightarrow 6$	6	-	-	-		29	25
Bước4	$1 \rightarrow 3 \rightarrow 6 \rightarrow 4$	4	-	-	-	29		-
Bước5	$1 \rightarrow 3 \rightarrow 2 \rightarrow 5$	5	-	-	-	-	29	-

4. Cài đặt

- Ta biểu diễn đơn đồ thị có hướng G bằng ma trận các trọng số của cạnh :

$$\text{trong đó : } a = (a_{uv})_{n \times n};$$

$$a_{uv} = \begin{cases} \text{trọng số của } (u,v); (u,v) \in E; \\ \infty; (u,v) \notin E; \end{cases}$$

- Dùng mảng 1 chiều để lưu trữ các nhãn tạm thời của các đỉnh, ký hiệu là $L[]$.

- Dùng mảng 1 chiều $Daxet[]$ các giá trị logic để đánh dấu các đỉnh đã được đưa vào tập S (gồm các đỉnh có nhãn chính thức):

Mỗi bước, nếu xác định được đỉnh k để đưa vào tập S thì ta gán $Daxet[k] = 1$; và khi đó $L[k]$ là nhãn chính thức của k (chỉ chiều dài của đường đi nhỏ nhất của đường từ s đến k).

- Khởi động dữ liệu :

o Khởi động $Daxet[]$ là rỗng :

$$Daxet[i] = 0, \forall i.$$

o Khởi động cận trên chiều dài của đường đi ngắn nhất từ s đến đỉnh khác (đánh nhãn tạm thời) bằng ∞ .

$$L[i] = \infty; i \neq s.$$

o Khởi động trọng số nhỏ nhất đường đi từ s đến s bằng 0.

$$L[s] = 0; // d(s,s) = 0$$

- Giả sử tại mỗi bước, (với Dht là đỉnh vừa đưa được vào S , $Daxet[Dht] = 1$), các đỉnh i chưa được xét sẽ được đánh nhãn lại như sau :

Nếu $(L[i] + m(Dht,i)) < L[i]$ thì :

$$L[i] = L[Dht] + m(Dht,i);$$

Và trong trường hợp này, đường đi ngắn nhất từ s đến i sẽ đi qua đỉnh Dht (đó là đỉnh kề trước i)

- Để lưu trữ các đỉnh trên đường đi ngắn nhất từ s đến t , với $t \in S$, ta dùng mảng một chiều $Ddnn[]$, với tính chất $Ddnn[i]$ là đỉnh trước đỉnh i .

Thuật toán được cài đặt bằng hàm sau :

Input $a[n][n]$, s

Output - Xuất ra màn hình đường đi ngắn nhất từ s đến các đỉnh còn lại
- Chiều dài tương ứng

void dijkstra(int s)

```
{
    int Ddnn[max]; // Chứa đường đi ngắn nhất từ s đến đỉnh t tại mỗi bước
    int i,k,Dht,Min;
    int Daxet[max]; //Đánh dấu các đỉnh đã đưa vào S
    int L[max];
    for ( i = 1; i <= n; i++)
    {
        Daxet[i] = 0;
        L[i] = VC;
```



```

    }
    //Dua dinh s vao tap dinh S da xet
    Daxet[s] = 1;
    L[s] = 0;
    Dht = s;
    int h = 1; //đếm mỗi bước : cho đủ n-1 bước
    while (h <= n-1)
    {
        Min = VC;
        for ( i = 1; i <= n; i++)
            if(!Daxet[i])
            {
                if ( L[dht] + a[dht][i] < L[i] ) //Tính lại nhãn
                {
                    L[i] = L[dht] + a[dht][i] ;
                    Ddnn[i] = dht;
                    //gan dinh hien tai la dinh truoc dinh i tren lo
trinh
                }
                if(L[i] < Min) // Chon đỉnh k
                {
                    Min = L[i];
                    k = i;
                }
            }
        // Tại bước h : tìm được đường đi ngắn nhất từ s đến k : Ddnn[]
        xuấtdd(s,k,Ddnn);
        cout<<"\nTrong so : "<<L[k];
        dht = k;// Khoi dong lai Dht
        Daxet[dht] = 1;      //Dua nut k vao tap nut da xet
        h++;
    }
}
//*****
void xuấtdd(int s, int k, int Ddnn[max])
{
    int i;
    cout<<"\nDuong di ngan nhat tu "<<s<<" den "<<k<<" la : ";
    i = k;
    while(i != s)
    {
        cout<<i<<"<--";
        i = Ddnn[i];
    }
}

```

```

    cout<<s;
}

```

5. Độ phức tạp của thuật toán

Thuật toán mô tả bởi 2 vòng lặp lồng nhau, nên $T(n) \in O(n^2)$.

IV. Thuật toán Prim – Tìm cây bao trùm nhỏ nhất (Minimal Spanning Tree)

1. Bài toán

$G = (V, E)$ là đơn đồ thị vô hướng liên thông, có trọng số.

$V = \{1, \dots, n\}$ là tập các đỉnh. E là tập các cạnh (edge).

Một cây T gọi là cây bao trùm của G nếu T là đồ thị con của G và chứa mọi đỉnh của G .

Vấn đề là tìm cây bao trùm có trọng số nhỏ nhất : MST (Minimal Spanning Tree) của G .

Các thuật toán cơ bản giải bài toán trên là các thuật toán Prim và Kruscal. Trong phần này, ta giới thiệu thuật toán Prim

2. Ý tưởng

Thuật toán Prim xây dựng một đồ thị con T của G như sau:

Đầu tiên chọn tùy ý 1 đỉnh của G đặt vào T .

Quá trình sau còn thực hiện trong khi T chưa chứa hết các đỉnh của G :

Mỗi bước, tìm một cạnh có trọng số nhỏ nhất nối 1 đỉnh trong T với 1 đỉnh ngoài T . Thêm cạnh này vào T .

Kết thúc thuật toán Prim cho ta một MST của đồ thị G .

Tính tham lam của thuật toán Prim là tại mỗi bước thêm vào T một cạnh có trọng số nhỏ nhất nối một đỉnh trong T và một đỉnh ngoài T .

3. Mô tả thuật toán

Input $G = (V, E)$

Output $T = (V, E)$ là MST

Mô tả :

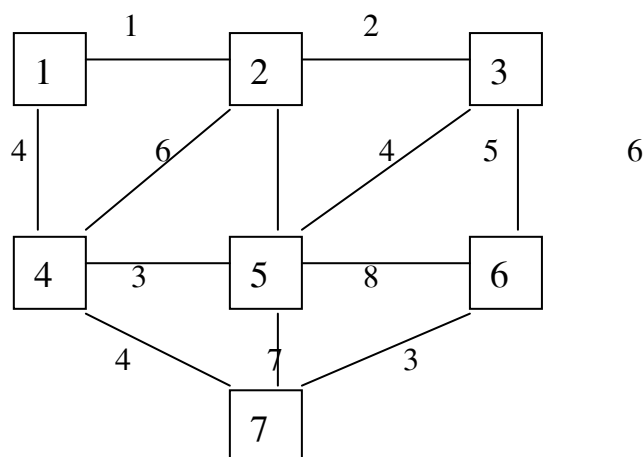
- Gọi U là tập con của V .
- Khởi động $T = (U, E) = \emptyset$; //Đồ thị con rỗng.
- Khởi động $U = \{1\}$; // Chọn đỉnh 1 đặt vào T .
- Trong khi ($U \neq V$)

Tìm cạnh (u, v) có trọng số nhỏ nhất, với $u \in U$ và $v \notin U$. Thêm đỉnh v này vào U . Thêm (u, v) vào T .

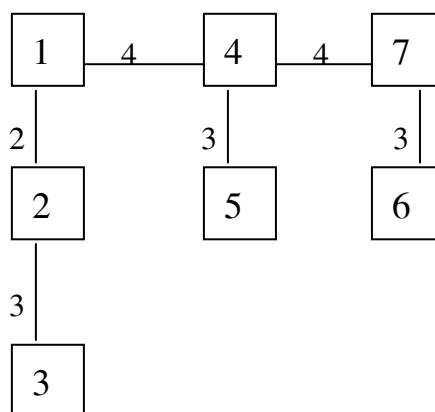
Lời giải của bài toán là lời giải tối ưu.

Minh họa :

Xét đồ thị sau :



Áp dụng thuật toán Prim, bắt đầu từ đỉnh 1, ta xây dựng dần 1 MST của đồ thị trên :



Hoạt động của thuật toán :

Bước	Các cạnh chọn	Tập U
0	-	{1}
1	(2,1)	{1,2}
2	(3,2)	{1,2,3}
3	(4,1)	{1,2,3,4}
4	(5,4)	{1,2,3,4,5}
5	(7,4)	{1,2,3,4,5,7}
6	(6,7)	{1,2,3,4,5,7,6}

4. Cài đặt

Để tiến hành cài đặt thuật toán, ta cần mô tả dữ liệu .

Đồ thị có trọng số có thể biểu diễn bởi 1 ma trận kề của nó :

$$c = (c[i][j])_{n \times n} .$$

$$c[i][j] = \begin{cases} \text{Trọng số } \overline{(i, j)}; \text{ Nếu } (i, j) \text{ tồn tại;} \\ 0; i = j \\ \infty; \text{ Nếu } (i, j) \text{ không tồn tại;} \end{cases}$$

Khi tìm cạnh có trọng số nhỏ nhất nối 1 đỉnh trong U và một đỉnh ngoài U tại mỗi bước, ta sẽ dùng 2 mảng để lưu trữ :

- Mảng `closest [] : // == U`

Với $i \in V \setminus U$ thì `closest[i] ∈ U` là đỉnh kề gần i nhất.

- Mảng `lowcost[i]` cho trọng số của cạnh $(i, \text{closest}[i])$.

Tại mỗi bước, ta duyệt mảng `lowcost` để tìm đỉnh `closest[k] ∈ U` sao cho trọng số $(k, \text{closest}[k]) = \text{lowcost}[k]$ là nhỏ nhất. Khi tìm được, Ta in cạnh $(\text{closest}[k], k)$, cập nhật vào các mảng `closest` và `lowcost`, và có k đã thêm vào U. Khi mà ta tìm được một đỉnh k cho cây bao trùm, ta cho `lowcost[k] = ∞`, là một giá trị rất lớn, lớn hơn bất kỳ trọng số của cạnh nào của đồ thị, như vậy đỉnh này sẽ không được kéo dài trong U.

`void Prim (Mat c)`

```
{
    double Lowcost[MAX];
    int Closest[MAX];
    int i,j,k,Min;
    for (i=2;i<=n;i++)
    {
        Lowcost[i] = c[1][i];
        Closest[i] = 1;
    }
    for (i=2;i<=n;i++)
    {
        Min = Lowcost[2];
        k = 2;
        for ( j=3; j<=n; j++) // Chọn k
            if (Lowcost[j] < Min )
            {
                Min = Lowcost[j];
                k = j;
            }
        cout<<endl<<k<<closest[k];
        lowcost[k] = ∞;
        // Khởi động lại Closest[], Lowcost[]
        for ( j=2; j<=n; j++)
            if ( (c[k][j] < lowcost[j]) && (lowcost[j] < ∞))
            {
                lowcost[j] = c[k][j];
                closest[j] = k;
            }
    }
}
```

}
}

5. Độ phức tạp thuật toán

Ta có thể thấy là Độ phức tạp trong thuật toán Prim là $O(n^2)$.

V. Bài toán ghi các bài hát

1. Phát biểu bài toán

Có n bài hát, dung lượng hoặc chiều dài phát được ghi trong mảng :

$$a = (a_1, a_2, \dots, a_n).$$

- Tần suất phát là như nhau, tìm và phát tuần tự.

- Gọi Σ là tập hợp các hoán vị trên tập $\{1, \dots, n\}$.

Với $K = (k_1, k_2, \dots, k_n) \in \Sigma$, ta đặt :

$$D(K) = \sum_{j=1}^n \sum_{i=1}^j a_{k_i}; \quad a_{k_i} \text{ là dung lượng bài hát thứ } k_i.$$

Tìm $b \in \Sigma$ sao cho $D(b) = \min\{D(k) : k \in \Sigma\}$

Minh họa :

Với $n = 3$; $a = (5, 10, 3)$.

i	1	2	3	Số hiệu bài hát
a[i]	5	10	3	Dung lượng

b	a_{k_1}	$a_{k_1} + a_{k_2}$	$a_{k_1} + a_{k_2} + a_{k_3}$	D(b)
(1,2,3)	5	5+10 = 15	5+10+3 = 18	38
(1,3,2)	5	5+3 = 8	5+3+10 = 18	31
(2,1,3)	10	10+5 = 15	10+5+3 = 18	43
(2,3,1)	10	10+3 = 13	10+3+5 = 18	41
(3,1,2)	3	3+5 = 8	3+5+10 = 18	29
(3,2,1)	3	3+10 = 13	3+10+5 = 18	34

Ta có : $D(b) = d((3,1,2)) = \min\{D(k) : k \in \Sigma\}$.

Một cách đơn giản để tìm lời giải trên là vét cạn các hoán vị, nhưng khi đó chi phí thời gian là quá lớn. Ta sẽ xác định trước một trật tự và xử lý dữ liệu theo trật tự này.

2. Thiết kế

Input : (a_1, a_2, \dots, a_n)

Output : Hoán vị $b = (k_1, \dots, k_n)$,

$\min = D(b) = \min\{D(K) : K \in \Sigma\}$;

Ta có nhận xét rằng $D(b)$ đạt min nếu $T_j = \sum_{i=1}^j a_{k_i}$; $\forall j \in \overline{1..n}$ đạt min.

Và T_j đạt Min nếu trong mỗi bước i , a_{k_i} được chọn vào là giá trị nhỏ nhất của dãy con còn lại của a .

Định hướng cho thuật toán tham lam trong trường hợp này là các T_j được tính theo trật tự tăng dần của a , phần tử a_i được chọn trong mỗi bước thêm vào chính là min của $\{a_i, \dots, a_n\}$. Khi đó, lời giải tìm được sẽ là lời giải tối ưu. Thuật toán tham lam có thể mô tả như sau:

`record_greedy(a,b,n) ≡`

* Khởi động $b[i] = i, \forall i$; $\text{min} = 0; t = 0$;

* for ($i = 1 \rightarrow n$)

- Chọn $j = \text{arcmin}(a,n,i)$; // $a[j] = \min\{a[i], \dots, a[n]\}$;

- $b[i] \leftrightarrow b[j]$; // Đổi chỗ

- $a[i] \leftrightarrow a[j]$;

- Cập nhật lại giá trị min;

$t = t + a[i]$;

$\text{min} = \text{min} + t$;

* return min;

3. Độ phức tạp của thuật toán

Thuật toán chọn min được sử dụng chính là chọn trực tiếp, Ta dễ thấy độ phức tạp của thuật toán trong các trường hợp là $O(n^2)$.

4. Cài đặt

```
int record_greedy(int a[max],int b[max],int n)
{
    int i,t= 0,min = 0;
    int j,k,x;
    for ( i = 1; i <= n; i++)        // Khởi động b
        b[i] = i;
    for (i = 1;i <=n; i++)
    {
        j = arcmin(a,n,i);
        Doich(b[i],b[j]);        // Chính xác lại b tại mỗi bước
        Doicho(a[i],a[j]);
        t += a[i];
        min += t;        // Chính xác dần min trong mỗi bước
    }
    return min;
}
```

VI. Bài toán chiếc túi xách (Knapsack)

1. Phát biểu bài toán

Có n vật, mỗi vật i , $i \in \{1, \dots, n\}$ được đặc trưng bởi trọng lượng w_i (kg) và giá trị v_i (US). Có một chiếc túi xách có khả năng mang m kg. Giả sử $w_i, v_i, m \in \mathbb{N}^*$,

$\forall i \in \{1, \dots, n\}$.

Hãy chọn vật xếp vào ba lô sao cho ba lô thu được có giá trị nhất.

Các trọng lượng w_i của n vật có thể biểu diễn bởi mảng :

$$w = (w_1, w_2, \dots, w_n);$$

Và giá trị tương ứng của các vật :

$$v = (v_1, v_2, \dots, v_n);$$

Các vật chọn được được lưu trữ trong mảng $\varepsilon[]$, với qui ước :

$$\varepsilon[i] = 1 \Leftrightarrow \text{Vật } i \text{ được chọn.}$$

Bài toán trở thành :

$$\begin{cases} \sum_{i=1}^n \varepsilon_i v_i \rightarrow \max \\ \sum_{i=1}^n \varepsilon_i w_i \leq m \\ \varepsilon_i \in \{0, 1\}; \forall i = \overline{1, n} \end{cases}$$

2. Thiết kế thuật toán

Thuật toán tham lam cho bài toán chọn vật có giá trị giảm dần (theo đơn giá).

Input

$w = (w_1, w_2, \dots, w_n)$; // Trọng lượng

$v = (v_1, v_2, \dots, v_n)$; // Giá trị

m = Sức chứa chiếc ba lô.

Output

$\varepsilon[1..n]$; // Đánh dấu các vật được chọn

V_{\max} : Giá trị lớn nhất của ba lô.

Mô tả :

Knap_Greedy(w,v,Chon, n, m) \equiv

* Khởi động $b[i] = i$, $\forall i = \overline{1, n}$; //Lưu trữ chỉ số làm cho mảng đơn giá giảm dần.

* Khởi động $\varepsilon[i] = 0$, $\forall i = \overline{1, n}$; Khởi động $V_{\max} = 0$;

* Tính đơn giá : $d_i = \frac{w_i}{v_i}$; $\forall i \in \{1, \dots, n\}$

* for ($i = 1$; $i \leq n$ && $m > 0$; $i++$)

{

- Chọn $j = \text{arcmax}(d, n, i)$; // $d[j] = \text{Max}\{d[i], \dots, d[n]\}$;

```

- b[i] ↔ b[j] ;
- // Cập nhật lại Vmax, Chon[ ], m;
  if (m > w[b[i]])
  {
    Vmax += v[b[i]];
    ε[b[i]] = 1;
    m -= w[b[i]];
  }
- d[i] ↔ d[j] ;
}
* return max;

```

Minh họa :

Với $n = 4; m = 17$

i	1	2	3	4	
w[i]	8	10	9	5	
v[i]	8	12	10	4	
Đơn giá d[i]	1	6/5	10/9	4/5	
Vật chọn theo thuật toán ε[i]		x		x	Ttl : 15/17 Vmax = 16
Phương án tối ưu	x		x		Ttl : 17 Vmax = 18

3. Độ phức tạp của thuật toán

Thuật toán chọn Max được sử dụng chính là chọn trực tiếp, nên độ phức tạp của thuật toán trong các trường hợp là $O(n^2)$.

4. Cài đặt

```
long MAX_GREEDY(long w[], long v[], int C[], int n, long m)
```

```

{
    int i, j, k, b[max];
    long Vmax = 0;
    double d[max]; // Mang đơn giá
    for (i = 1; i <= n; i++)
    {
        b[i] = i;
        C[i] = 0;
        d[i] = (double)w[i]/v[i];
    }
    for (i = 1; i <= n && m > 0; i++)
    {
        j = arcmax(d, n, i);
        dcu(b[i], b[j]); // Đổi chỗ
    }
}

```



```

        if (m > w[b[i]])
        {
            Vmax += v[b[i]];
            C[b[i]] = 1;
            m -= w[b[i]];
        }
        det(d[i],d[j]); //Đổi chỗ
    }
    return Vmax;
}

```

VII. Phương pháp tham lam và Heuristic

Trong khi thiết kế giải các bài toán ta có thể cố thử theo mọi phương án để tìm lời giải tối ưu. Nhưng không phải lúc nào cũng được như vậy, vì có rất nhiều trường hợp tốn phí rất nhiều thời gian. Nên thay vì tìm lời giải tối ưu, ta tìm một lời giải tốt theo nghĩa :

- Nó đáp ứng được yêu cầu, trong một thời gian mà thực tế chấp nhận được.

Một thuật toán “tốt” như vậy (không phải là tối ưu) gọi là thuật toán

Heuristic.

Thuật toán Heuristic thường được thể hiện trong phương pháp tham lam. Ta cố gán cho một trật tự nào đó rồi xử lý theo trật tự đã cho.

Ta xét bài toán “ Tô màu đồ thị “ sau :

“ Tô màu cho một đồ thị với số màu ít nhất có thể.”

Tô màu cho đồ thị là gán màu cho mỗi đỉnh của đồ thị sao cho không có 2 đỉnh kề nhau cùng một màu.

Bài toán tô màu đồ thị được nghiên cứu trong nhiều thập kỷ nay, nó thuộc vào một lớp khá rộng bài toán, được gọi là “ bài toán N-P đầy đủ “, mà đối với chúng thì những lời giải hiện có chủ yếu thuộc loại “cố hết mọi khả năng”.

Nếu đồ thị nhỏ ta có thể cố thử mọi phương án, để có thể đi tới một lời giải tối ưu. Nhưng với đồ thị lớn thì cách làm này là không thể. Một lời giải “tốt” có được từ thuật toán Heuristic là cách tiếp cận của ta cho trường hợp này.

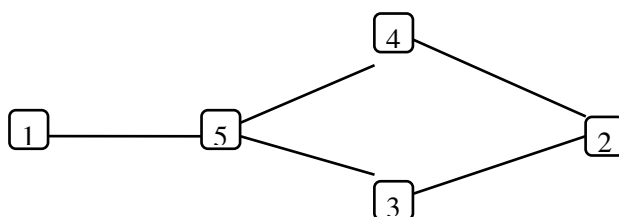
Thuật toán Heuristic hợp lý cho bài toán tô màu đồ thị được thể hiện bởi cách thiết kế tham lam :

- Ta cố tô màu cho các đỉnh, trước hết bằng một màu, không thể được nữa mới dùng tới màu thứ hai, thứ ba . . .

Thuật toán được mô tả như sau :

1. Chọn một đỉnh chưa được tô màu, và tô nó bằng màu mới.
2. Tìm trong các đỉnh chưa được tô màu, với mỗi đỉnh đó xác định xem có phải là đỉnh kề của 1 đỉnh đã được tô màu mới chưa. Nếu chưa thì tô đỉnh đó bằng màu mới.

minh họa :



- Tô xanh cho đỉnh (1), theo thứ tự đó tô xanh cho (2).
- Khi đó, (3) và (4) phải tô khác màu, chẳng hạn đỏ.
- Khi đó, (5) lại phải tô một màu thứ 3, chẳng hạn vàng.

Cách tiếp cận này thể hiện rõ ý tham lam. Nó thực hiện tô màu một đỉnh nào đó mà nó có thể tô được, không hề chú ý đến tình huống bất lợi có thể xảy ra (khi theo trật tự đã xác định trước).

Cần nhắc hơn, với đồ thị trên ta chỉ cần 2 màu để tô, chẳng hạn :

- Tô xanh cho (1), (3) và (4).
- Tô đỏ cho (2) và (5).

BÀI TẬP

Bài 1 :

Cho một lưới hình vuông cấp n , mỗi ô được gán với một số tự nhiên.

Tại một ô có thể di chuyển đến ô khác theo các hướng : lên trên, xuống dưới, rẽ trái, rẽ phải (4 ô kề cạnh).

Tìm đường đi từ ô đầu tiên (1,1) đến ô (m, m) sao cho tổng các ô đi qua là nhỏ nhất. ($1 \leq m \leq n$).

Bài 2 :

Cho n thiết bị $(p_i)_{1 \leq i \leq n}$ và m công việc $(w_i)_{1 \leq i \leq m}$.

Các thiết bị có thể làm việc đồng thời và làm việc nào cũng được. Mỗi việc đã làm ở thiết bị nào thì làm đến cùng. Thời gian làm công việc w_i là t_i , $i \in \{1, \dots, m\}$.

Cần xây dựng một lịch biểu là thứ tự thực hiện các công việc sao cho tổng thời gian hoàn thành là nhanh nhất.

Bài 3 :

Cho m công việc $(w_i)_{1 \leq i \leq m}$ tương ứng thời gian thực hiện $(t_i)_{1 \leq i \leq m}$ và tập các thiết bị cùng chức năng.

Với thời gian T_0 cho trước cố định, để hoàn thành m công việc thì cần bố trí các công việc trên các thiết bị sao cho số thiết bị đạt min.

Bài 4 :

Giải bài toán :

$$\begin{cases} \sum_{i=1}^n \varepsilon_i v_i \rightarrow \max \\ \sum_{i=1}^n \varepsilon_i w_i \leq m \\ 0 \leq \varepsilon_i \leq 1; \forall i = \overline{1, n} \end{cases}$$

Bài 5 :

Có n loại đồ vật, mỗi loại có số lượng không hạn chế. Đồ vật loại i , đặc trưng bởi trọng lượng w_i và giá trị sử dụng v_i , với mọi $i \in \{1, \dots, n\}$.

Cần chọn các vật này đặt vào một chiếc túi xách có giới hạn trọng lượng m , sao cho tổng giá trị sử dụng các vật được chọn là lớn nhất.

Bài 6 :

Cho $G = (V, E)$ là một đơn đồ thị liên thông. $V = \{1, \dots, n\}$ là tập các đỉnh, E là tập các cạnh. Thuật toán Kruscal xây dựng tập cạnh T của cây bao trùm nhỏ nhất

$H = (V, T)$ theo từng bước :

- Sắp E theo thứ tự không giảm.
- Khởi đầu $T = \emptyset$;
- Trong khi ($|T| < n - 1$)

{

Chọn e là cạnh có trọng số nhỏ nhất trong E ;

$E = E \setminus \{e\}$;

if ($T \cup \{e\}$ không chứa chu trình)

$T = T \cup \{e\}$;

}

Bài 7 :

Cài đặt thuật toán tô màu đồ thị.

CHƯƠNG 6 : PHƯƠNG PHÁP QUY HOẠCH ĐỘNG (Dynamic Programming)

I. Phương pháp tổng quát

Đối với nhiều thuật toán, phương pháp chia để trị thường đóng vai trò chủ đạo trong việc thiết kế thuật toán. Trong phương pháp quy hoạch động lại càng tận dụng phương pháp này : Khi không biết cần phải giải bài toán con nào, ta giải tất cả các bài toán con và lưu trữ những lời giải này (để khỏi phải tính toán lại) nhằm sử dụng lại chúng để giải bài toán lớn hơn.

Phương pháp này tổ chức tìm kiếm lời giải theo kiểu từ dưới lên (bottom up). Xuất phát từ các bài toán con nhỏ và đơn giản nhất, tổ hợp các lời giải của chúng để có lời giải của bài toán con lớn hơn...và cứ như thế để tìm lời giải của bài toán ban đầu.

Khi sử dụng phương pháp quy hoạch động để giải quyết vấn đề, ta có thể gặp 2 khó khăn sau :

1. Số lượng lời giải của các bài toán con có thể rất lớn không chấp nhận được .

2. Không phải lúc nào sự kết hợp lời giải của các bài toán con cũng cho ra lời giải của bài toán lớn hơn.

Để giải quyết những trường hợp như vậy, phương pháp quy hoạch động dựa vào một nguyên lý, gọi là nguyên lý tối ưu (The principle of optimality) của Bellman :

“ Nếu lời giải của bài toán là tối ưu thì lời giải của các bài toán con cũng tối ưu ”.

Trong thuật toán quy hoạch động thường dùng các thao tác :

- Xây dựng một hàm quy hoạch động (hoặc phương trình quy hoạch động).
- Lập bảng lưu lại các giá trị của hàm.
- Truy xuất lời giải tối ưu của bài toán từ bảng lưu.

...

Trong chương này ta giới thiệu một số bài toán có thể dùng quy hoạch động giải quyết một cách hiệu quả. Những vấn đề này đều liên quan đến bài toán tìm phương án tối ưu để thực hiện một công việc nào đó, và chúng có chung một tính chất là đáp án tốt nhất cho một bài toán con vẫn được duy trì khi bài toán đó trở thành một phần trong một bài toán lớn.

II. Thuật toán Floyd -Tìm đường đi ngắn nhất giữa các cặp đỉnh

1. Bài toán

Cho $G = (V, E)$ là một đồ thị có hướng có trọng số. $V = \{1, \dots, n\}$ là tập các đỉnh. E là tập các cung. Tìm đường đi ngắn nhất giữa các cặp đỉnh của đồ thị .

2. Ý tưởng

Thuật toán Floyd được thiết kế theo phương pháp quy hoạch động. Nguyên lý tối ưu được vận dụng cho bài toán này là :

“ Nếu k là đỉnh nằm trên đường đi ngắn nhất từ i đến j thì đoạn đường từ i đến k và từ k đến j cũng phải ngắn nhất. “

3. Thiết kế

Đồ thị được biểu diễn bởi ma trận kề các trọng số của cung : $a = (a_{ij})_{n \times n}$.

$$\forall i, j \in \{1, \dots, n\} : a_{ij} = \begin{cases} \text{Trọng số } (i, j); (i, j) \in E \\ 0; i = j \\ \infty; (i, j) \notin E \end{cases}$$

Ta ký hiệu :

- Ma trận trọng số đường đi ngắn nhất giữa các cặp đỉnh : $d = (d_{ij})$
 d_{ij} : Trọng số của đường đi ngắn nhất từ i đến j .
- Ma trận xác định các đỉnh trung gian của đường đi ngắn nhất từ i đến j : $p = (p_{ij})$
 p_{ij} : đường đi ngắn nhất từ i đến j có đi qua đỉnh trung gian p_{ij} hay không ?
 $\begin{cases} p_{ij} = 0; \text{ đường đi ngắn nhất từ } i \text{ đến } j \text{ không có đi qua đỉnh trung gian } p_{ij}. \\ p_{ij} \neq 0; \text{ đường đi ngắn nhất từ } i \text{ đến } j \text{ đi qua đỉnh trung gian } p_{ij}. \end{cases}$
- Ở bước k :
 - Ký hiệu ma trận d là d^k cho biết chiều dài nhỏ nhất của đường đi từ i đến j .
 - Ký hiệu ma trận p là p^k cho biết đường đi ngắn nhất từ i đến j có đi qua đỉnh trung gian thuộc tập đỉnh $\{1, \dots, k\}$.

Input a

Output d, p ;

Mô tả :

Bước 0 :

- Khởi động d : $d = a$; ($= d^0$)
- Khởi động p : $p_{ij} = 0$;

Bước 1 :

Kiểm tra mỗi cặp đỉnh i, j : Có/không một đường đi từ i đến j đi qua đỉnh trung gian 1, mà có trọng số nhỏ hơn bước 0 ? Trọng số của đường đi đó là :

$$d^1_{ij} = \min\{d^0_{ij}, d^0_{i1} + d^0_{1j}\}$$

Nếu $d^1_{ij} = d^0_{i1} + d^0_{1j}$ thì $p^1_{ij} = 1$, tức là đường đi tương ứng đi qua đỉnh 1.

Bước 2 :

Kiểm tra mỗi cặp đỉnh i, j : Có/không một đường đi từ i đến j đi qua đỉnh trung gian 2, mà có trọng số nhỏ hơn bước 1 ? Trọng số của đường đi đó là :

$$d^2_{ij} = \min\{d^1_{ij}, d^1_{i2} + d^1_{2j}\}$$

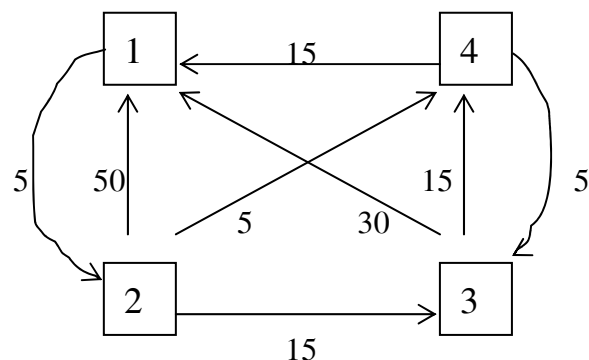
Nếu $d^2_{ij} = d^1_{i2} + d^1_{2j}$ thì $p^2_{ij} = 2$: tức là đường đi tương ứng đi qua đỉnh 2.

...

Cứ tiếp tục như vậy, thuật toán kết thúc sau bước n , ma trận d xác định trọng số đường đi ngắn nhất giữa 2 đỉnh bất kỳ i, j . Ma trận p cho biết đường đi ngắn nhất từ i đến j có đi qua đỉnh trung gian p_{ij} .

Minh hoạ :

Tìm đường đi ngắn nhất giữa các cặp đỉnh của đồ thị :



Hoạt động của thuật toán Floyd :

b1		1	2	3	4
d^1	1	0	5	∞	∞
	2	50	0	15	5
	3	30	35	0	15
	4	15	20	5	0

p^1		1	2	3	4
1	1	0	0	0	0
	2	0	0	0	0
	3	0	1	0	0
	4	0	1	0	0

b2		1	2	3	4
d^2	1	0	5	20	10
	2	50	0	15	5
	3	30	35	0	15
	4	15	20	5	0

p^2		1	2	3	4
1	1	0	0	2	2
	2	0	0	0	0
	3	0	1	0	0
	4	0	1	0	0

b3		1	2	3	4
d^3	1	0	5	20	10
	2	45	0	15	5
	3	30	35	0	15
	4	15	20	5	0

p^3		1	2	3	4
1	1	0	0	2	2
	2	3	0	0	0
	3	0	1	0	0
	4	0	1	0	0

b4		1	2	3	4
$d^4 = d$	1	0	5	15	10
	2	20	0	10	5
	3	30	35	0	15
	4	15	20	5	0

$p^4 = p$		1	2	3	4
1	1	0	0	4	2
	2	4	0	4	0
	3	0	1	0	0
	4	0	1	0	0

Căn cứ vào ma trận d , ta chỉ ra khoảng cách đường đi ngắn nhất từ i đến j , và dựa vào p có thể xác định các đỉnh nằm trên đường đi ngắn nhất này.

Chẳng hạn, với $i = 1, j = 3$.

Theo d , $d_{13} = 15$. Nên đường đi ngắn nhất từ 1 đến 3 có khoảng cách là 15.

Theo p , đường đi ngắn nhất từ đỉnh 1 đến đỉnh 3 đi qua đỉnh trung gian $p_{13} = 4$, đường đi ngắn nhất từ đỉnh 1 đến đỉnh 4 đi qua đỉnh trung gian $p_{14} = 2$, đường đi ngắn nhất từ đỉnh 1 đến đỉnh 2 không đi qua đỉnh trung gian nào ($p_{12} = 0$).

Vậy đường đi ngắn nhất từ đỉnh 1 đến đỉnh 3 là : $1 \rightarrow 2 \rightarrow 4 \rightarrow 3$.

4. Cài đặt

```
void floyd()
{
    int i, j, k;
    // Khởi động ma trận d và p
    for (i = 1; i <= n; i++)
        for (j = 1; j <= n; j++)
        {
            d[i][j] = a[i][j];
            p[i][j] = 0;
        }
    for (k = 1; k <= n; k++) // Tính ma trận d và p ở bước lặp k
        for (i = 1; i <= n; i++)
            if (d[i][k] > 0 && d[i][k] < vc)
                for (j = 1; j <= n; j++)
                    if (d[k][j] > 0 && d[k][j] < vc)
                        if (d[i][k] + d[k][j] < d[i][j])
                        {
                            d[i][j] = d[i][k] + d[k][j];
                            p[i][j] = k;
                        }
}
```

Hàm xuất đường đi ngắn nhất từ x đến y cài đặt như sau :

```
void xuấtdd(int x, int y)
{
    int r;
    if (p[x][y] == 0)
    {
        cout << y << " -> ";
        return;
    }
    else
    {
        r = p[x][y];
        xuấtdd(x, r);
        xuấtdd(r, y);
    }
}
```

}
}

5. Độ phức tạp của thuật toán

$$T(n) \in O(n^3).$$

III. Nhân tổ hợp nhiều ma trận

1. Bài toán

Xét tích các ma trận : $A = A_1 \times \dots \times A_n$, với giả thiết phép nhân có nghĩa.

Do tính kết hợp của phép nhân ma trận, các ma trận A_i có thể nhóm lại theo nhiều cách khác nhau, mà ma trận kết quả A không đổi. Tuy nhiên có sự khác biệt về chi phí khi thay đổi các tổ hợp các ma trận A_i . Ta lưu ý rằng tích 2 ma trận cấp $(m \times n)$ và $(n \times p)$ sẽ có chi phí là $m \times n \times p$.

Vấn đề là tìm trình tự thực hiện các ma trận sao cho có chi phí ít nhất.

Cho các ma trận, với các kích thước tương ứng :

$$\begin{array}{ccccccc} A & \times & B & \times & C & \times & D \\ 30 \times 1 & & 1 \times 40 & & 40 \times 10 & & 10 \times 25 \end{array}$$

Nhân các ma trận trên với các thứ tự sau :

Thứ tự	Chi phí
$((AB)C)D$	$30 \times 1 \times 40 + 30 \times 40 \times 10 + 30 \times 10 \times 25 = 20700$
$(A(B(CD)))$	$40 \times 10 \times 25 + 1 \times 40 \times 25 + 30 \times 1 \times 25 = 11750$
$(AB)(CD)$	$30 \times 1 \times 40 + 40 \times 10 \times 25 + 30 \times 40 \times 25 = 41200$
$A((BC)D)$	$1 \times 40 \times 10 + 1 \times 10 \times 25 + 30 \times 1 \times 25 = 1400$

Có thể thấy chi phí cho phép nhân các ma trận phụ thuộc vào cách tổ hợp các ma trận.

2. Ý tưởng

Ta giải bài toán bằng cách tiếp cận từ dưới lên. Ta sẽ tính toán và lưu trữ lời giải tối ưu cho từng phần nhỏ để tránh tính toán lại cho bài toán lớn hơn.

Trước hết là cho các bộ 2 ma trận, các bộ 3 ma trận . . .

Chẳng hạn, để tính $A \times B \times C$ ta có thể tính theo các cách : $(A \times B) \times C$ hoặc $A \times (B \times C)$.

Nên chi phí để tính $A \times B \times C$ là chi phí tính được từ 2 phần :

Phần một là chi phí $kq1 \times C$, với $kq1 = A \times B$ (chi phí này đã tính và được lưu trữ)

Phần hai là chi phí $A \times kq2$, với $kq2 = B \times C$ (chi phí này đã được lưu trữ)

So sánh 2 phần trên và lưu trữ chi phí nhỏ hơn. . .

3. Thiết kế

Mấu chốt là tính chi phí nhân bộ các ma trận : $A_i \times \dots \times A_j$, với $1 \leq i < j \leq n$, trong đó các bộ nhỏ hơn đã được tính và lưu trữ kết quả.

Với một cách tổ hợp các ma trận :

$$A_i \times \dots \times A_j = (A_i \times \dots \times A_k) \times (A_{k+1} \times \dots \times A_j)$$

Chi phí để nhân các ma trận A_i, \dots, A_j sẽ bằng tổng : Chi phí để nhân $A_i \times \dots \times A_k$ ($= kq_1$), chi phí để nhân $A_{k+1} \times \dots \times A_j$ ($= kq_2$), và chi phí $kq_1 \times kq_2$.

Nếu gọi M_{ij} là chi phí nhỏ nhất để nhân bộ các ma trận $A_i \times \dots \times A_j$, $1 \leq i < j \leq n$, thì:

* M_{ik} là chi phí nhỏ nhất để nhân bộ các ma trận $A_i \times \dots \times A_k$

* $M_{k+1,j}$ là chi phí nhỏ nhất để nhân bộ các ma trận $A_{k+1} \times \dots \times A_j$

Vì ma trận kq_1 cỡ $d_{i-1} \times d_k$ và kq_2 có cỡ $d_k \times d_j$, nên chi phí để nhân $kq_1 \times kq_2$ là $d_{i-1}d_kd_j$.

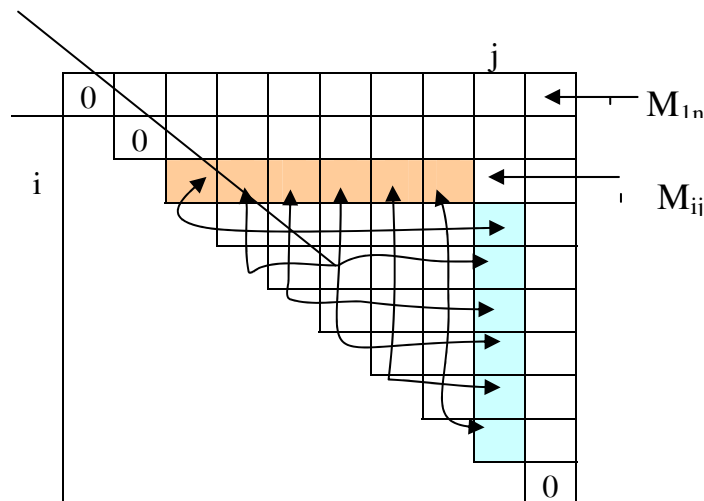
Vậy ta có :

$$\begin{cases} M_{ij} = \min_{i \leq k \leq j-1} \{M_{ik} + M_{k+1,j} + d_{i-1}d_kd_j\}, 1 \leq i < j \leq n \\ M_{ii} = 0 \end{cases}$$

Ta có thể xem M là ma trận tam giác trên : $(M_{ij})_{1 \leq i < j \leq n}$. Ta cần tính và làm đầy các phần tử của ma trận này cho đến khi xác định được M_{1n} . Đường chéo chính bằng 0 tất cả.

Tính M_{ij} , ta cần biết M_{ik} , $M_{k+1,j}$. Ta tính bằng dọc theo các đường chéo bắt đầu từ đường chéo kế trên đường chéo chính và thay đổi về hướng góc phải trên.

Ta muốn biết thứ tự tốt nhất để nhân dãy các ma trận (theo nghĩa chi phí nhỏ nhất). Mỗi lần ta xác định được tổ hợp tốt nhất, ta dùng biến k để lưu trữ thứ tự này. Đó là $O_{ij} = k$, với $M_{ik} + M_{k+1,j} + d_{i-1}d_kd_j$ đạt min.



(Bảng tính M_{ij})

Các M_{ij} ta lưu trữ trong mảng 2 chiều M .

Các chỉ số k để xác định được M_{ij} ta lưu trữ trong mảng 2 chiều O .

Kích thước của các ma trận ta lưu trữ trong mảng 1 chiều d : A_i là ma trận có d_{i-1} hàng, d_i cột. Thuật toán có thể viết như sau :

Input d = (d_0, d_1, \dots, d_n) ;

Output M = (M_{ij}) ,

O = (O_{ij});

Mô tả :

MO(d,n,O,M) \equiv

int i, j, k, diag;

* for (i = 1; i <= n; i++)

M[i][i] = 0;

* for (diag = 1; diag <= n-1; diag++)

for (i = 1; i <= n - diag; i++)

+ j = i + diag;

+ $M_{ij} = \underset{i \leq k \leq j-1}{\text{Min}} \{M_{ik} + M_{k+1,j} + d_{i-1}d_kd_j\}$;

+ O[i][j] = k; // với M_{ij} đạt min

* return m[1][n];

Kết quả của thuật toán, với $d_0 = 30$; $d_1 = 1$; $d_2 = 40$; $d_3 = 10$; $d_4 = 25$:

$$M = \begin{bmatrix} 0 & 1200 & 700 & 1400 \\ - & 0 & 400 & 650 \\ - & - & 0 & 10000 \\ - & - & - & 0 \end{bmatrix} ; \quad O = \begin{bmatrix} - & 1 & 1 & 1 \\ - & - & 2 & 3 \\ - & - & - & 3 \\ - & - & - & - \end{bmatrix}$$

4. Độ phức tạp của thuật toán

$$T(n) \in O(n^3)$$

5. Cài đặt

long MO(int d[max],int n, mat O, mat M)

{

int i, j, k, diag,min,csm;

for (i = 1; i <= n; i++)

M[i][i] = 0;

for (diag = 1; diag <= n-1; diag++)

for (i = 1; i <= n - diag; i++)

{

j = i + diag;

csm = i;

min = M[i][i]+M[i+1][j]+d[i-1]*d[i]*d[j];

for (k= i; k <= j - 1;k++)

if (min > (M[i][k]+M[k+1][j]+d[i-1]*d[k]*d[j]))

```

        {
            min = M[i][k]+M[k+1][j]+d[i-1]*d[k]*d[j];
            csm = k;
        }
        M[i][j] = min;
        O[i][j] = csm;
    }
    return M[1][n];
}

```

Hàm xuất trình tự tổ hợp các ma trận :

```

void MOS(int i, int j, mat O)
{
    int k;
    if (i == j)
        cout<<'A'<<i;
    else
    {
        k = O[i][j];
        cout<<'(';
        MOS(i,k,O);
        cout<<'*';
        MOS (k+1,j,O);
        cout<<')';
    }
}

```

IV. Cây nhị phân tìm kiếm tối ưu (Optimal Binary Search Tree)

Ta thường tổ chức cây nhị phân tìm kiếm trên giả thiết là các khóa tìm kiếm có đồng khả năng về truy xuất. Tuy nhiên có những trường hợp mà ta có thông tin về xác suất truy xuất các khóa. Chẳng hạn quá trình phân tích của trình biên dịch để xác định xem một từ có phải là từ khóa hay không ? Trong trường hợp này, việc thống kê hàng trăm chương trình (được biên dịch) có thể cho thông tin khá chính xác về tần số xuất hiện tương đối của các khóa.

Giả sử trong cây nhị phân tìm kiếm, xác suất truy xuất của khóa K_i là p_i :

$$P\{X = K_i\} = p_i.$$

Bây giờ ta muốn tổ chức cây nhị phân tìm kiếm sao cho tổng số các bước tìm kiếm là nhỏ nhất. Chi phí tìm kiếm đặc trưng bởi số lượng các phép toán so sánh cần thiết khi tìm kiếm trên cây, nên ta phải chú ý đến độ dài đường đi trên cây. Ở đây ta cũng dựa vài khái niệm

này, nhưng phải thay đổi chút ít để phản ánh được tính chất của cây nhị phân tìm kiếm bây giờ : Đối với các khóa mà xác suất tìm kiếm lớn hơn phải tương ứng với đường đi ngắn hơn.

Muốn thế, ta gán cho mỗi khóa (nút) một số dương mà ta gọi là trọng số, đó là xác suất của khóa này trong tìm kiếm. Từ đây dẫn tới khái niệm độ dài đường đi có trọng số của cây (Weighted path tree):

“ Độ dài đường đi (trong) có trọng số của cây là tổng độ dài đường đi có trọng số ứng với mỗi nút trên cây”.

Đó chính là giá trị :

$$P = \sum_{i=1}^n p_i h_i$$

Với p_i là xác suất để khóa K_i xuất hiện. h_i là mức của nút tương ứng K_i .

Mục đích của bài toán ta muốn đặt ra là : Cực tiểu hóa độ dài đường đi có trọng số với phân phối xác suất cho trước. Nói cách khác, Xác định cây nhị phân tìm kiếm sao cho P có giá trị nhỏ nhất.

Một cách tự nhiên ta có thể thay xác suất truy xuất của các khóa bằng tần suất của các khóa, bài toán có thể phát biểu lại như sau :

1. Phát biểu bài toán

Cho trước tập các khóa K_i , $i \in \overline{1, n}$, sao cho : $K_1 < K_2 < \dots < K_n$.

Xác định cây nhị phân tìm kiếm với tập các khóa trên sao cho biểu thức P sau đây có giá trị nhỏ nhất :

$$P = \sum_{i=1}^n a_i h_i$$

Trong đó :

- h_i là mức của nút trong thứ i ; $i \in \overline{1, n}$.
- a_i là tần suất xuất hiện của khóa k_i ; $i \in \overline{1, n}$.

Cây nhị phân tìm kiếm thỏa mãn yêu cầu này gọi là cây nhị phân tìm kiếm tối ưu.

2. Ý tưởng

Người ta chứng minh được rằng số lượng các cây nhị phân tìm kiếm n nút có dạng khác nhau, là :

$$\frac{1}{n+1} C_{2n}^n \cong \frac{4^n}{\sqrt{\pi} \cdot n^{\frac{3}{2}}}; \text{ (Khi } n \text{ khá lớn).}$$

Do đó việc chọn cây nhị phân tìm kiếm tối ưu bằng cách lựa chọn trong các cây đó một cây có độ dài đường đi có trọng số nhỏ nhất, là khó thực hiện khi n lớn. Ta có thể áp dụng phương pháp qui hoạch động cho bài toán này, vì ta có thể sử dụng được nguyên lý tối ưu. Đó là vì cây tối ưu có tính chất đáng chú ý sau đây :

Một cây là tối ưu thì các cây con cũng là tối ưu. Tính chất này gợi lên một thuật toán :

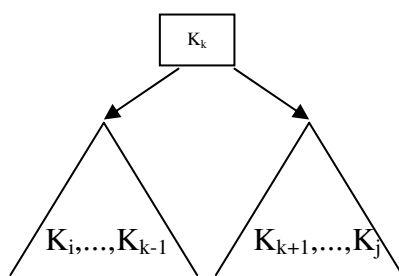
Xuất phát từ từng nút được xem như cây con nhỏ nhất, tìm một cách có hệ thống các cây lớn hơn . Như thế cây lớn lên “ từ lá tới gốc “.

3. Thiết kế thuật toán

Cách tiếp cận quy hoạch động để giải quyết bài toán này là tìm nghiệm tối ưu theo cách phát triển cây lớn lên từ lá tới gốc, tức là tìm kiếm phương án tối ưu xây dựng cây con gồm các khóa K_i, \dots, K_j và lưu trữ lại đáp án.

Nếu gọi K_k là gốc của cây T_{ij} tương ứng với các khóa liền nhau K_i, \dots, K_j thì các nút K_i, \dots, K_{k-1} phải nằm trên cây con trái và các nút K_{k+1}, \dots, K_j phải nằm trên cây con phải, và ta phải sắp tối ưu cho 2 cây con này. Vì ta không biết phải chọn nút nào làm gốc cho tốt nhất nên ta phải chọn thử tất cả các nút và cực tiểu hóa các cách chọn này.

Ta có thể thấy độ dài đường đi có trọng số của cây T_{ij} sẽ bằng tổng độ dài đường đi có trọng số của 2 cây con $T_{i,k-1}$ và $T_{k+1,j}$ và số lần một phép tìm kiếm duyệt qua các nút từ lá đến gốc (đó chính là độ dài đường đi có trọng số của cây T_{ij}).



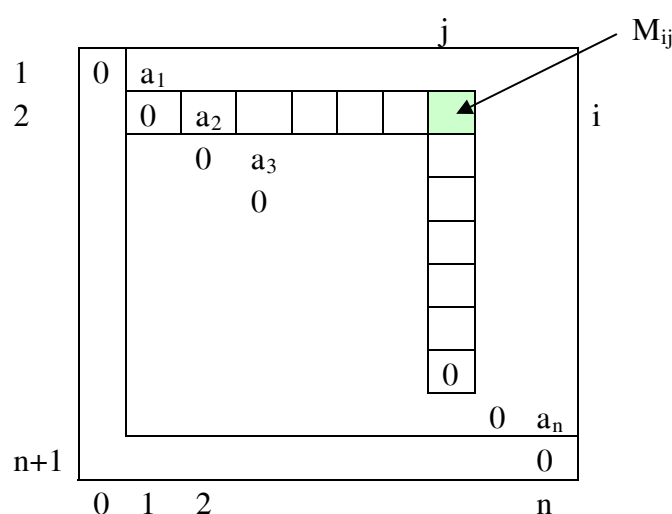
Đặt :

M_{ij} = Độ dài đường đi có trọng số của cây nhị phân tìm kiếm tối ưu tương ứng với các khóa $K_i < \dots < K_j$, với $1 \leq i \leq j \leq n$.

Khi đó M_{ij} được xác định theo công thức :

$$\begin{aligned}
 M_{ij} &= \min_{i \leq k \leq j} \left\{ M_{i,k-1} + \sum_{q=i}^{k-1} a_q + M_{k+1,j} + \sum_{q=k+1}^j a_q + a_k \right\} \\
 &= \min_{i \leq k \leq j} \left\{ M_{i,k-1} + M_{k+1,j} + \sum_{q=i}^j a_q \right\} \\
 &= \min_{i \leq k \leq j} \left\{ M_{i,k-1} + M_{k+1,j} \right\} + \sum_{q=i}^j a_q; \text{ với } i < j. \\
 M_{ii} &= a_i.
 \end{aligned}$$

Ta tính M_{ij} bằng cách lập bảng như thuật toán nhân tổ hợp nhiều ma trận. Để giữ vết khóa được chọn làm gốc trong mỗi bước ta dùng một bảng (ma trận) $root$; $root[i][j]$ là khóa chọn làm gốc của cây con chứa K_i, \dots, K_j .



Thuật toán có thể mô tả như sau

input $a[1..n]$, //chứa tần suất các khóa

n ; //số khóa

output $root[][]$ // bảng các khóa của cây nhị phân tìm kiếm tối ưu.

$minOBST$; //trọng số của cây nhị phân tìm kiếm tối ưu.

$obst(a, n, root, M) \equiv$

double $m[][]$;

* for ($i=1 \rightarrow n$) //Khởi động

{

$M[i][i] = p[i]$;

$M[i][i-1] = 0$;

$root[i][i] = i$;

}

$M[n+1][n] = 0$;

* for ($diag = 1 \rightarrow n-1$)

for ($i=1 \rightarrow n-diag$)

{

$j = i + diag$;

$$M[i][j] = \min_{i \leq k \leq j} (M[i][k-1] + M[k+1][j]) + \sum_{q=i}^j a[q];$$

$root[i][j] = k$; //Chỉ số giá trị nhỏ nhất $m[i][j]$

}

* $minOBST = M[1][n]$;

* return $minOBST$;

4. Độ phức tạp của thuật toán

$T(n) \in O(n^3)$

5. Cài đặt

```
double obst(double a[max],int n, mat root, mat M)
{
    int i, j, k, diag,csm;
    double min;
    for (i = 1; i <= n; i++)
    {
        M[i][i] = p[i];
        M[i][i-1] = 0;
        root[i][i] = i;
    }
    M[n+1][n] = 0;
    for (diag = 1; diag <= n-1; diag++)
        for (i = 1; i <= n - diag; i++)
        {
            j = i + diag;
            csm = i;
            min = M[i][i-1] + M[i+1][j];
            for (k= i; k <= j; k++)
                if ( min > (M[i][k-1]+M[k+1][j]) )
                {
                    min = M[i][k-1]+M[k+1][j];
                    csm = k;
                }
            root[i][j] = csm;
            for (int q = i; q <= j; q++)
                min += a[q];
            M[i][j] = min;
        }
    return M[1][n];
}
```

V. Dãy chung dài nhất của 2 dãy số

1. Bài toán

Cho các dãy số nguyên : $a = (a_1, \dots, a_m)$, $b = (b_1, \dots, b_n)$.

Tìm một dãy dài nhất c nhận được từ a và b như sau : trong a xóa đi một số phần tử, trong b xóa đi một số phần tử (các phần tử bị xóa đi trong a và b không nhất thiết là trùng chỉ số).

Ta nói c là dãy con chung dài nhất nhận được từ a và b .

Ví dụ :

Với các dãy a , b cho như sau :

i	1	2	3	4	5	6	7
a	3	5	1	3	5	5	3
b	1	5	3	5	3	1	

Dãy con chung dài nhất c là :

i	1	2	3	4	5	6	7
	1	5	5	3			
c	1	3	5	3			
	5	3	5	3			

2. Ý tưởng

$\forall i \in \{1, \dots, m\}, \forall j \in \{1, \dots, n\}$, nếu ta gọi L_{ij} là độ dài của dãy con dài nhất của $a[1\dots i]$ và $b[1\dots j]$ thì khi đó L_{mn} chính là độ dài của dãy con dài nhất của a và b .

Vậy ta cần tính các L_{ij} .

Một cách tiếp cận tự nhiên của thuật toán là từ dưới lên. Lời giải tối ưu của các dãy có độ dài ngắn hơn sẽ được tính toán và lưu trữ lại để sử dụng lại việc xét các dãy có độ dài dài hơn.

3. Thuật toán

Mấu chốt của thuật toán là tính các L_{ij} .

- Đầu tiên ta xét trong trường hợp dãy a có 1 phần tử, hoặc dãy b có 1 phần tử.

Dễ thấy là:

$$L_{1j} = \begin{cases} 0; & \text{Nếu } a_1 \text{ không có trong } b[1\dots j] \\ 1; & \text{Ngược lại} \end{cases}$$

Tương tự :

$$L_{i1} = \begin{cases} 0; & \text{Nếu } b_1 \text{ không có trong } a[1\dots i] \\ 1; & \text{Ngược lại} \end{cases}$$

- Tổng quát, với $i > 1, j > 1$:
Xét 2 trường hợp sau :

TH1 : $a_i = b_j$.

Suy ra rằng $L_{ij} = L_{i-1,j-1} + 1$;

TH1 : $a_i \neq b_j$.

* Nếu $a_i \in b[1\dots j]$ thì rõ ràng là $a_i \in b[1\dots j-1]$, nên : $L_{ij} = L_{i,j-1}$.

* Nếu $b_j \in a[1\dots i]$ thì rõ ràng là $b_j \in a[1\dots i-1]$, nên : $L_{ij} = L_{i-1,j}$.

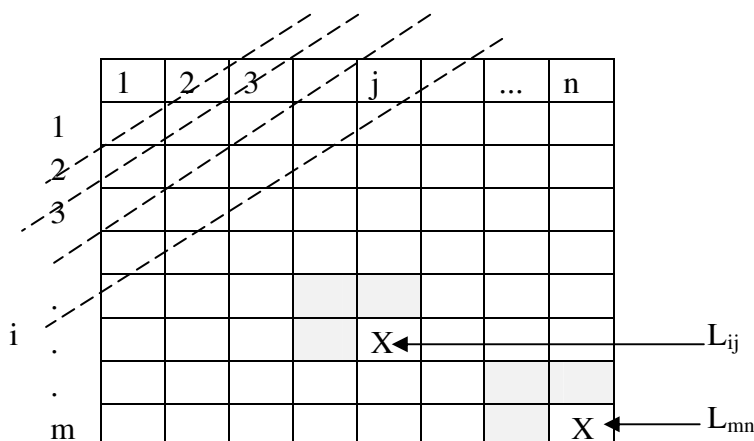
Vậy ta có :

$$L_{ij} = \text{Max}\{L_{i,j-1}, L_{i-1,j}, L_{i-1,j-1} + 1\}$$

$$\text{trong đó : } x = \begin{cases} 1; & \text{Nếu } a_i = b_j \\ 0; & \text{Ngược lại} \end{cases}$$

Ta có thể xem L là một $m \times n$ ma trận : $(L_{ij})_{m \times n}$. Ta tính và làm đầy các phần tử của ma trận này có thể xem

Tính L_{ij} , cần phải biết $L_{i,j-1}$, $L_{i-1,j}$, $L_{i-1,j-1}$. Ta tính các phần tử của ma trận L từ góc bên trái lần lượt theo các đường chéo song song với đường chéo ngược



Input a, b, m, n

Output L

Qhd(a, m, b, n, L) \equiv

for ($i = 1$; $i \leq m$; $i++$)

if ($b_1 \in a[1..i]$)

$L_{i1} = 1$;

else

$L_{i1} = 0$;

for ($j = 1$; $j \leq n$; $j++$)

if ($a_1 \in b[1..j]$)

$L[1][j] = 1$;

else

$L[1][j] = 0$;

for ($i = 2$; $i \leq m$; $i++$)

for ($j = 2$; $j \leq n$; $j++$)

+ if ($a[i] == b[j]$)

$x = 1$;

else

$x = 0$;

+ $L_{ij} = \text{Max}\{L_{i,j-1}, L_{i-1,j}, L_{i-1,j-1} + x\}$;

return L_{mn} ;

Ghi chú :

Để tìm dãy c từ ma trận L, ta xuất phát từ ô L_{mn} .

Giả sử đang ở ô L_{ij} , và cần xác định c_i . ($1 \leq i \leq L_{mn}$) .

Nếu $a_i = b_j$ thì $c_i = a_i$, còn ngược lại thì $L_{ij} = L_{i,j-1}$ hoặc $L_{ij} = L_{i-1,j}$.

Nếu $L_{ij} = L_{i,j-1}$ ta đi đến ô $L_{i,j-1}$, còn nếu $L_{ij} = L_{i-1,j}$ thì đi đến ô $L_{i-1,j}$

Minh họa :

Với dữ liệu a, b như trên, ta có :

$$L = (L_{ij})_{7 \times 6} = \begin{bmatrix} 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 2 & 2 & 2 \\ 1 & 1 & 1 & 2 & 2 & 3 \\ 1 & 1 & 2 & 2 & 3 & 3 \\ 1 & 2 & 2 & 3 & 3 & 3 \\ 1 & 2 & 2 & 3 & 3 & 3 \\ 1 & 2 & 3 & 3 & 4 & 4 \end{bmatrix} ; \quad c = (1, 5, 5, 3)$$

4. Độ phức tạp của thuật toán

$T(n) \in O(n^2)$.

5. Cài đặt

```
int Bt_Dcnd(int a[MAX], int m, int b[MAX], int n, int L[MAX][MAX])
{
    int i, j, x;
    for (i = 1; i <= m; i++)
        if( Thuoc(a,i,b[1] ))
            L[i][1] = 1;
        else
            L[i][1] = 0;
    for (j = 1; j <= n; j++)
        if( Thuoc(b,j,a[1] ))
            L[1][j] = 1;
        else
            L[1][j] = 0;
    for (i = 2; i <= m; i++)
        for(j = 2; j <= n; j++)
        {
            if(a[i] == b[j])
                x = 1;
```

```

        else
            x = 0;
            L[i][j] = Max(L[i][j-1], L[i-1][j], L[i-1][j-1] + x);
        }
    int k = L[m][n];
    return k;
}
//*****
void Dcdn(int a[MAX],int b[MAX],int m,int n,int L[MAX][MAX],int c[MAX], int
&l)
{
    int i = m, j = n;
    l = 0;
    while ( i > 0 && j > 0)
    {
        if( a[i] == b[j])
        {
            l++;
            c[l] = a[i];
            i--; j--;
        }
        else
            if(L[i][j] == L[i][j-1])
                j--;
            else
                i--;
    }
    for(i = 1; i <= l/2; i++)
        Hv(c[i], c[l-i+1]); //Đổi chỗ.
}

```

VI. Bài toán người du lịch

Bài toán người du lịch ta đã giải bằng các phương pháp :

- Tham lam : Lời giải tìm được không chắc tối ưu.
- Nhánh cận : Lời giải tìm được tối ưu.

Trong phần này, ta tiếp cận cách giải bài toán này bằng phương pháp quy hoạch động.

n thành phố được đánh số từ 1 đến n . Đường đi từ thành phố i đến thành phố j xem như là cung đi từ đỉnh i đến đỉnh j của đơn đồ thị có hướng. Chi phí đi từ i đến j là trọng số $m(i,j)$ của cung (i,j) .

Vậy bài toán người du lịch có thể xem là tìm một chu trình xuất phát từ đỉnh i nào đó của đơn đồ thị có hướng có trọng số $G=(V,E)$, đi qua mỗi đỉnh đúng 1 lần sao cho có trọng số nhỏ nhất.

1. Ý tưởng

Giả sử có một chu trình thỏa yêu cầu bài toán và bắt đầu từ đỉnh 1 về đỉnh 1. Khi đó chu trình này bao gồm cung $(1, k)$, với $k \in V \setminus \{1\}$, và đường đi từ k đến 1, đi qua mỗi đỉnh còn lại thuộc $V \setminus \{1, k\}$, mỗi đỉnh đúng 1 lần.

Nếu chu trình này có trọng số nhỏ nhất (tối ưu), thì theo nguyên lý tối ưu đường đi từ k đến 1 cũng có trọng số nhỏ nhất (tối ưu).

2. Thiết kế thuật toán

Biểu diễn G bằng ma trận kề : $C = (C_{ij})_{n \times n}$, với :

$$C_{ij} = \begin{cases} m(i, j) & ; (i, j) \in E \\ 0 & ; i = j \\ \infty & ; (i, j) \notin E \end{cases}$$

Xét tập $S \subset V \setminus \{1\}$ và $i \in (V \setminus \{1\}) \setminus S$. Ta gọi :

$d(i, S)$ = Trọng số của đường đi ngắn nhất đi từ đỉnh i đến đỉnh 1 đi qua mỗi đỉnh trong S đúng 1 lần.

Vậy với $2 \leq i \leq n$:

- Nếu $S = \emptyset$, rõ ràng là :

$$d(i, \emptyset) = C_{i1};$$

- Nếu $S \neq \emptyset$:

$$d(i, S) = \min_{k \in S} \{C_{ik} + d(k, S \setminus \{k\})\}$$

Khi đó, trọng số của chu trình ngắn nhất đi từ 1 đến 1 sẽ là :

$$d(1, V \setminus \{1\}) = \min_{2 \leq k \leq n} \{C_{1k} + d(k, V \setminus \{1, k\})\}$$

Để tính $d(1, V \setminus \{1\})$ ta cần có $d(k, V \setminus \{1, k\})$, với $2 \leq k \leq n$.

Tổng quát, ta cần tính các $d(i, S)$, $S \subset V \setminus \{1\}$ và $i \in (V \setminus \{1\}) \setminus S$.

Đầu tiên ta tính và lưu trữ $d(i, \emptyset)$; $d(i, S)$ với S chỉ có 1 phần tử; $d(i, S)$ với S có 2 phần tử, ... cho đến khi tính được các $d(k, V \setminus \{1, k\})$ với $2 \leq k \leq n$.

Input : C

Output : $d(1, V \setminus \{1\})$

Mô tả:

Bước 0 :

- Khởi tạo : $d(i, \emptyset) = C_{i1}$; $2 \leq i \leq n$

Bước 1 :

- Với $S \subset V \setminus \{1\}$ và $|S| = 1$; $\forall i \neq 1$ và $i \notin S$:

$$d(i, S) = \min_{k \in S} \{C_{ik} + d(k, S \setminus \{k\})\}$$

.

Bước n-2 :

- Với $S \subset V \setminus \{1\}$ và $|S| = n-2$; $\forall i \neq 1$ và $i \notin S$:

$$d(i, S) = \min_{k \in S} \{C_{ik} + d(k, S \setminus \{k\})\}$$

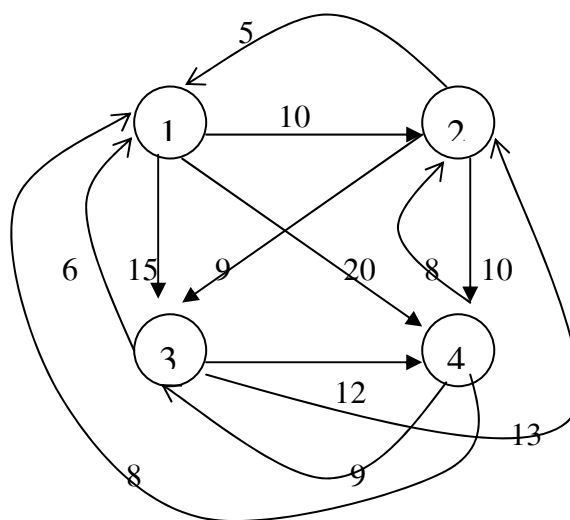
Bước n-1 :

$$d(1, V \setminus \{1\}) = \min_{2 \leq k \leq n} \{C_{1k} + d(k, V \setminus \{1, k\})\}$$

Minh họa :

Xét đồ thị sau :

$$C = \begin{bmatrix} 0 & 10 & 15 & 20 \\ 5 & 0 & 9 & 10 \\ 6 & 13 & 0 & 12 \\ 8 & 8 & 9 & 0 \end{bmatrix}$$



* Khi $S = \emptyset$:

$$d(2, \emptyset) = C_{21} = 5$$

$$d(3, \emptyset) = C_{31} = 6$$

$$d(4, \emptyset) = C_{41} = 8$$

* Khi S là tập chỉ có 1 phần tử $\neq 1$ và $i \in (V \setminus \{1\}) \setminus S$

$$d(2, \{3\}) = C_{23} + d(3, \emptyset) = C_{23} + C_{31} = 15$$

$$d(2, \{4\}) = C_{24} + d(4, \emptyset) = 18$$

$$d(3, \{2\}) = C_{32} + d(2, \emptyset) = 18$$

$$d(3, \{4\}) = C_{34} + d(4, \emptyset) = 20$$

$$d(4, \{2\}) = C_{42} + d(2, \emptyset) = 13$$

$$d(4, \{3\}) = C_{43} + d(3, \emptyset) = 15$$

* Khi S là tập có 2 phần tử $\neq 1$ và $i \in (V \setminus \{1\}) \setminus S$

$$d(2, \{3, 4\}) = \min \{ C_{23} + d(3, \{4\}), C_{24} + d(4, \{3\}) \} = 25$$

$$d(3, \{2, 4\}) = \min \{ C_{32} + d(2, \{4\}), C_{34} + d(4, \{2\}) \} = 25$$

$$d(4, \{2, 3\}) = \min \{ C_{42} + d(2, \{3\}), C_{43} + d(3, \{2\}) \} = 23$$

* Cuối cùng ta có:

$$\begin{aligned} d(1, \{2, 3, 4\}) &= \min \{ C_{12} + d(2, \{3, 4\}), C_{13} + d(3, \{2, 4\}), C_{14} + d(4, \{2, 3\}) \} \\ &= \min \{ 10 + 25, 15 + 25, 20 + 23 \} \\ &= \min \{ 35, 40, 43 \} = 35. \end{aligned}$$

Cách tìm các đỉnh nằm trên chu trình có trọng số nhỏ nhất tương ứng như sau :

- Gọi $J(i,S)$ là đỉnh làm cho $\min_{k \in S} \{C_{ik} + d(k, S \setminus \{k\})\}$ đạt min.

Ta có :

$$J(1, \{2,3,4\}) = 2$$

$$J(2, \{3, 4\}) = 4$$

$$J(4, \{3\}) = 3$$

Vậy chu trình ngắn nhất là : $1 \rightarrow 2 \rightarrow 4 \rightarrow 3 \rightarrow 1$

3. Độ phức tạp của thuật toán

Ta xét thời gian thực hiện $T(n)$ của $d(i,S)$:

- i có $n-1$ lựa chọn.
- $\forall i=2,...,n$: Số các tập S có k phần tử khác $1, i$ là C_{n-2}^k .

Do đó :

$$T(n) = \sum_{k=0}^{n-2} (n-1)C_{n-2}^k = (n-1)2^{n-2}$$

Mặt khác khi tính $d(i,S)$ với S gồm k phần tử, ta cần thực hiện $k-1$ phép so sánh để xác định min. Nên thời gian thực hiện của thuật toán là $n^2 2^n$.

BÀI TẬP

Bài 1 :

Cho một bảng chữ nhật m hàng, n cột. Mỗi ô của bảng chứa một số nguyên dương. Hãy tìm một đường đi từ cột 1 đến cột m , đi qua đúng m ô sao cho tổng giá trị các ô là nhỏ nhất.

Bài 2 :

Cho một dãy số nguyên a_1, a_2, \dots, a_n . Hãy xóa một số lượng ít nhất các số trong dãy sao cho dãy còn lại (vẫn giữ nguyên thứ tự) là một dãy không giảm.

Bài 3 :

Cho n loại tiền xu tương ứng với các giá trị k_1, k_2, \dots, k_n xu. Cần đổi T đồng (tiền giấy) ra tiền xu sao cho số xu cần dùng là ít nhất. Cho 1 đồng bằng 100 xu.

Bài 4 :

Cho n loại đồ vật, mỗi loại có số lượng không hạn chế. Trong mỗi loại, các đồ vật có trọng lượng như nhau và có giá trị như nhau.

Với mọi $i \in \{1, \dots, n\}$, đồ vật loại thứ i có trọng lượng là w_i và giá trị là p_i .

Có một chiếc túi xách với giới hạn trọng lượng là m .

Cần chọn các vật từ n loại đồ vật trên để đặt vào chiếc túi xách sao cho thu được chiếc túi xách có giá trị nhất.

Bài 5 :

Cài đặt hàm Ackerman :

$$A(m, n) = \begin{cases} n + 1; m = 0, n \in N; \\ A(m - 1, 1); m \in N^*, n = 0; \\ A(m - 1, A(m, n - 1)); m, n \in N^* \end{cases}$$

Bài 6 :

Tính :

$$p[i][j] = \begin{cases} 1; i = 0, j > 0 \\ 0; i > 0, j = 0 \\ \frac{1}{2} \cdot (p[i - 1][j] + p[i][j - 1]); i > 0, j > 0 \end{cases}$$

Bài 7 :

Tính các số Catalan :

$$T(n) = \sum_{i=1}^{n-1} T(i)T(n-i); n \in N^*$$

Bài 8 :

Cho a là một dãy các số nguyên dương.

Tìm trong a một dãy con giảm dần dài nhất.

Bài 9 :

Cài đặt hàm :

$f(x, k)$ = Số cách phân tích x thành tổng các số nguyên tố
mà mỗi số nguyên tố xuất hiện trong tổng không quá k lần.

Bài 10 :

Có n người xếp hàng mua vé xe. Trong hàng, ta đánh số theo thứ tự từ 1 đến n. Thời gian bán vé cho người thứ i là t_i . Mỗi người cần mua 1 vé nhưng có thể được mua tối đa 2 vé. Một người có thể mua hộ cho người đứng sau mình. Người thứ i mua vé hộ cho người thứ i+1 thì thời gian mua vé cho 2 người là r_i .

Xác định phương án sao cho n người đều có vé với thời gian ít nhất.

PHỤ LỤC

Dữ liệu sử dụng trong các thuật toán được trình bày trong giáo trình thường được lưu trữ trong các mảng 1 chiều, mảng 2 chiều vuông. Để tránh việc nhập liệu nhiều lần làm mất thời gian trong khi thực hành, đồng thời chuẩn bị trước dữ liệu để kiểm tra kết quả thuật toán, các dữ liệu đầu vào sẽ được tổ chức và lưu trữ trong các tệp văn bản. Trong chương trình thể hiện thuật toán, chỉ cần viết thêm một hàm chuyển dữ liệu từ tệp vào mảng.

I. Mảng 1 chiều :

1. Định dạng :

- Dòng 1 : n

(là một số nguyên dương, chỉ kích thước sử dụng của mảng)

- Dòng 2 :

Các số chỉ các phần tử của mảng. Hai số tách biệt bằng 1 khoảng trắng.

2. Soạn thảo tệp dữ liệu :

Có thể sử dụng các phần mềm soạn thảo văn bản trong chế độ không định dạng, như NC, NOTEPAD, hoặc cài đặt thành một hàm riêng thực hiện việc nhập liệu từ bàn phím rồi ghi vào tệp.

3. Hàm chuyển dữ liệu từ tệp văn bản vào mảng 1 chiều :

//Đọc dữ liệu từ tệp f, rồi ghi vào dãy a.

void Tep_Day(char *f, Day a, int &n)

```
{
    ifstream in(f);
    if(!in)
    {
        cout<<"\nKhong mo duoc tep "<<f;
        getch();
        exit(1) ;
    }
    in>>n;
    for( int i = 1; i <= n; i++)
        in>>a[i];
    in.close();
}
```

II. Mảng 2 chiều vuông (ma trận vuông):

1. Định dạng :

- Dòng 1 : n

(là một số nguyên dương, chỉ kích thước của ma trận vuông)

- n dòng tiếp theo, mỗi dòng n số .

Các số chỉ các phần tử của ma trận . Hai số tách biệt bằng 1 khoảng trắng.

2. Soạn thảo tệp dữ liệu :

Như mảng 1 chiều.

3. Hàm chuyển dữ liệu từ tệp văn bản vào ma trận vuông.

//Đọc dữ liệu từ tệp f, rồi ghi vào ma trận a.

```
void Tep_Mat(char *f, mat a, int &n)
{
    ifstream in(f);
    if(!in)
    {
        cout<<"\nKhong mo duoc tep "<<f;
        getch();
        exit(1) ;
    }
    in>>n ;
    int i, j;
    for(i = 1; i <= n; i++)
        for(j = 1; j <= n; j++)
            in>>a[i][j];
    in.close();
}
```

TÀI LIỆU THAM KHẢO

ALFRED V. AHO & JOHN E.HOPCROFT & JOHN D. ULMANN
“Data structures and algorithms”, Addison Wesley, 1983.

C.FROIDEVAUX & M-C GAUDEL & M. SORIA
“Types de données et algorithmes “, Ediscience, 1994

D. BEAUQUIER & J.BERSTEL & Ph.CHRÉTIENNE,
“Elément d’algorithmique”., Masson, 1992.

DONALD KNUTH, “The art of computer programming”,
vol 1 : Fundamental algorithms;
vol 3 : Sorting and searching ,
Addition Wesley Publishing company,1973.

ELLIS HOROWITZ & SARTAJ SAHANI:
“Fundamentals of computer algorithms”,
computer Science Press INC, 1978.

G. BRASSARD & P. BRATLEY ,
“Algorithmique - conception et analyse”, Masson, Paris , 1987.

J.P. BARTHÉLEMY & G. COHEN & A . LOBSTEIN ,
“ Complexité algorithmique et problèmes de communications “
Masson, Paris , 1992.

NGUYỄN XUÂN HUY , “Thuật toán “, Nhà xuất bản Thống kê,
Hà Nội, 1988

NIKLAUS WIRTH , “Algorithms + data structures = Programs”,
Prentice-Hall INC,1976

S.E.GOODMAN & S.T. HEDETNIEMI ,
“Introduction to the design and analysis of algorithms”,
Mcgraw-Hill.1977.

TRƯỜNG CHÍ TÍN, giáo trình “Cấu trúc dữ liệu và thuật toán 1”,
Đại học Đà lạt, 2002.