



TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI  
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG



## Thuật toán ứng dụng

Nguyễn Khánh Phương

Computer Science department  
School of Information and Communication technology  
E-mail: phuongnk@soict.hust.edu.vn

### Nội dung khóa học

Chương 1. Các cấu trúc dữ liệu và thư viện

**Chương 2. Kỹ thuật đệ quy và nhánh cận**

Chương 3. Chia để trị

Chương 4. Quy hoạch động

Chương 5. Các thuật toán trên đồ thị và ứng dụng

Chương 6. Các thuật toán xử lý xâu và ứng dụng

Chương 7. Lớp bài toán NP-đầy đủ

NGUYỄN KHÁNH PHƯƠNG  
KHMT – SOICT- DHBK HN

### Nội dung chương 2

1. Duyệt toàn bộ
2. Thuật toán quay lui
3. Thuật toán nhánh cận

NGUYỄN KHÁNH PHƯƠNG  
KHMT – SOICT- DHBK HN

### Nội dung chương 2

- 1. Duyệt toàn bộ**
2. Thuật toán quay lui
3. Thuật toán nhánh cận

NGUYỄN KHÁNH PHƯƠNG  
KHMT – SOICT- DHBK HN

## 1. Duyệt toàn bộ

Duyệt toàn bộ (Brute force – Exhaustive search):

- Khi bài toán yêu cầu tìm đối tượng thỏa mãn tính chất nào đó trong tập các đối tượng đã cho, ta có thể áp dụng phương pháp duyệt toàn bộ:
  - Duyệt qua tất cả các đối tượng, với mỗi đối tượng, tiến hành kiểm tra xem nó có thỏa mãn tính chất yêu cầu hay không, nếu có thì đối tượng đó là lời giải cần tìm, nếu không thì tiếp tục tìm.

Ví dụ: Bài toán người du lịch (Traveling Salesman Problem): Một người du lịch muốn di tham quan  $n$  thành phố  $T_1, T_2, \dots, T_n$ . *Hành trình là cách di xuất phát từ một thành phố nào đó đi qua tất cả các thành phố còn lại, mỗi thành phố dừng một lần, rồi quay trở lại thành phố xuất phát.* Biết  $d_{ij}$  là chi phí đi từ thành phố  $T_i$  đến thành phố  $T_j$  ( $i, j = 1, 2, \dots, n$ ). Tìm hành trình với tổng chi phí là nhỏ nhất.

Giải: Duyệt toàn bộ cần tính toán tổng cộng  $n!$  hành trình có thể có, mỗi hành trình tính chi phí đường đi tương ứng, và so sánh chi phí  $n!$  hành trình này với nhau để đưa ra được hành trình có chi phí nhỏ nhất.

- Duyệt toàn bộ: đơn giản, nhưng thời gian tính không hiệu quả.

NGUYỄN KHÁNH PHƯƠNG  
KHMT - SOICT - ĐHQGHN

## Ví dụ: bài toán stock

Bài toán thường được hỏi trong các cuộc phỏng vấn của Google và Amazon:

Given a list of prices of a single stock for  $N$  number of days, find stock span for each day. Stock span is defined as a number of consecutive days prior to the current day when the price of a stock was less than or equal to the price at current day.

Ví dụ:

Giả stock trong 6 ngày là  $\{100, 60, 70, 65, 80, 85\}$ , khi đó span =  $\{0, 0, 1, 0, 3, 4\}$ .

Giải bằng phương pháp duyệt toàn bộ:

Duyệt lần lượt từng ngày  $i$  (từ trái sang phải `for i = 0, ..., 5`):

- `span[i] = 0;`
- quét lần lượt các ngày  $j$  trước nó (`for j = i-1, ..., 0`):
  - `if price[j] <= price[i] then span[i]++;`
  - `else break;`

Độ phức tạp:  $O(n^2)$  trong đó  $n$  là số ngày.

Bài tập: đưa ra thuật toán với thời gian tính  $O(n)$  và bộ nhớ  $O(n)$

6

## Ví dụ: Một dạng phát biểu khác của bài toán stock

Various signal towers are present in a city. Towers are aligned in a straight horizontal line (from left to right) and each tower transmits a signal in the right to left direction. Tower A shall block the signal of Tower B if Tower A is present to the left of Tower B and Tower A is taller than Tower B. So, the range of a signal of a given tower can be defined as :

((the number of contiguous towers just to the left of the given tower whose height is less than or equal to the height of the given tower) + 1).

You need to find the range of each tower.

INPUT

First line contains an integer  $T$  specifying the number of test cases.

Second line contains an integer  $n$  specifying the number of towers.

Third line contains  $n$  space separated integers ( $H[i]$ ) denoting the height of each tower.

OUTPUT

Print the range of each tower (separated by a space).

Constraints

$1 \leq T \leq 10$

$2 \leq n \leq 10^4$

$1 \leq H[i] \leq 10^4$



A      B

SAMPLE INPUT	SAMPLE OUTPUT
1 7 100 80 60 70 60 75 85	1 1 1 2 1 4 6

7

## Ví dụ: Một dạng phát biểu khác của bài toán stock

### Training 2 - DST - 20192

#### A. 02. HIST

time limit per test: 1 second  
memory limit per test: 256 megabytes  
input: standard input  
output: standard output

Lô Ban là một vị thư moe nổi tiếng bậc nhất thời Tống của Trung Quốc với đôi bàn tay cực kỳ tài hoa khéo léo. Ngưỡng mài tài năng của Lô Ban, Vua Tống mời Lô Ban vào triều làm quan chuyên quản lý việc thiết kế cung điện và chế tác vật dụng linh xảo. Vua Tống trong một chuyến du ngoạn ở núi Ngũ Nhạc tình cờ phát hiện một phiến đá ngũ sắc tuyệt đẹp. Vua Tống này ra ý định đem phiến đá này về cung để chế tác thành một bản cò. Nhưng Vua Tống nhanh chóng phát hiện ra rằng, phiến đá này có hình dạng kỳ lạ rất khó để có thể cắt ra phần diện tích vuông vắn dù lớn cho bản cò. Vậy là Vua Tống triệu Lô Ban vào triều để thương lượng:

Bé mặt phiến đá có thể được mô tả như là một hình đa giác được ghép thành từ nhiều phiến đá nhỏ hình chữ nhật có chung nhau một mép, có các chiều dài khác nhau nhưng giống nhau về chiều rộng và bằng 1 đơn vị. Trong hình vẽ dưới đây, phiến đá đa giác gồm các hình chữ nhật có chiều cao lần lượt từ trái qua phải là 2,1,4,5,1,3,3 và chiều rộng đều bằng 1.



Yêu cầu: Bạn cần giúp Lô Ban tìm ra hình chữ nhật chung mép với các hình chữ nhật nhỏ và có diện tích lớn nhất nằm trong phiến đá đa giác nói trên. Ở hình vẽ dưới, hình chữ nhật lớn nhất là hình được gạch chéo.

NGUYỄN KHÁNH PHƯƠNG 8  
KHMT - SOICT - ĐHQGHN

## Amortized time: Phân tích thời gian khấu trừ

- Giả sử thuật toán để xuất cần phải thực hiện một dãy các thao tác (operation), thời gian khấu trừ của thuật toán là thời gian tính trung bình của mỗi thao tác trong dãy các thao tác cần thực hiện trong thuật toán.

$$\text{Amortized time} = \frac{O(\text{Total cost})}{\text{number of operations}}$$

NGUYỄN KHÁNH PHƯƠNG 9  
KHMT - SOICT - DHBK HN

10

## Ví dụ 1: Binary counter

### Amortized time: Phân tích thời gian khấu trừ

Xét bài toán lưu một biến đếm nhị phân rất lớn. Giả sử ta sẽ sử dụng mảng A, với A[i] = bit thứ i của biến đếm nhị phân.

Câu hỏi:

- Mỗi khi biến đếm tăng thêm 1, ta sẽ phải làm gì ?
- Giả sử cần thực hiện liên tiếp  $n$  lần thao tác tăng biến đếm lên 1, khi đó thời gian tính của dãy  $n$  thao tác này là bao nhiêu ?

## Amortized time: Phân tích thời gian khấu trừ

**Question:** Tại sao cần phân tích thời gian tính khấu trừ

**Answer:**

Khi phân tích thời gian tính trong tình huống tồi nhất của thuật toán (worst case analysis) cho ta thấy thời gian tính của thuật toán rất lớn (ví dụ: hàm mũ), ta sẽ dùng phân tích thời gian khấu trừ để hiểu rõ hơn thời gian chạy của thuật toán. Khi một thuật toán áp dụng trên một cấu trúc dữ liệu nào đó thực hiện một dãy các thao tác, trong đó có những thao tác thời gian tính toán nhanh, nhưng có những thao tác thời gian tính toán chậm. Vậy muốn đánh giá tất cả các thao tác này trong thuật toán, tính xem trung bình một thao tác trong thuật toán có thời gian tính là bao nhiêu, ta sẽ dùng phân tích thời gian khấu trừ

**Chú ý:** Phân tích thời gian khấu trừ (amortized time analysis) không phải là phân tích thời gian tính trung bình (average case analysis) vì phân tích thời gian khấu trừ không đưa ra bất kỳ giả thiết nào về tính phân phối của giá trị dữ liệu đầu vào như khi phân tích thời gian tính trung bình.

11

## Ví dụ 1: Binary counter

### Amortized time: Phân tích thời gian khấu trừ

Binary number	Chi phí tăng biến đếm lên 1
0000	–
0001	1
0010	2
0011	1
0100	3
0101	1
0110	2
0111	1
1000	4
1001	1
1010	2
1011	1
1100	3
1101	1
1111	1

Xét trường hợp  $k = 4$  bits  
Khởi tạo: biến đếm ban đầu bởi mảng gồm dãy 4 bit đều = 0, tại mỗi bước ta tăng biến đếm lên 1, tức là cần “flip” một số bit 0 thành 1 và một số bit 1 thành 0. Giả sử mỗi thao tác “flip” bit, ta mất chi phí là 1.

Chi phí thực hiện  $n$  thao tác liên tiếp tăng biến đếm lên 1 là bao nhiêu ?

- Worst case analysis: thao tác tốn nhất mất  $O(k)$ . Tổng cộng  $n$  thao tác, nên chi phí là  $O(n*k)$

Hỏi: rõ ràng thời gian tính thực sự không lên đến  $O(n*k)$  vì không phải thao tác nào cũng tốn  $O(k)$ . Vậy làm thế nào để đưa ra được thời gian tính nhỏ hơn  $O(n*k)$  ?

Trả lời : dùng Amortized analysis

### Ví dụ 1: Binary counter

#### Amortized time: Phân tích thời gian khấu trừ

Binary number	Chi phí tăng biến đếm lên 1	
0000	–	Nhận thấy, trong dãy $n$ thao tác liên tiếp tăng biến đếm lên 1:
0001	1	<ul style="list-style-type: none"> <li>Cứ mỗi thao tác tăng: bit thứ 0 flip</li> <li>Cứ 2 thao tác tăng: bit thứ 1 flip</li> <li>Cứ 4 thao tác tăng: bit thứ 2 flip</li> <li>Cứ 8 thao tác tăng: bit thứ 3 flip</li> <li>.....</li> </ul>
0100	3	Do đó, tổng số lượng bit flip được thực hiện trong $n$ thao tác liên tiếp tăng biến đếm là:
0101	1	$n + \frac{n}{2} + \frac{n}{4} + \dots + \frac{n}{2^{k-1}} < n \left( \frac{1}{1 - \frac{1}{2}} \right)$
0110	2	$= 2n$
0111	1	Tổng chi phí của dãy $n$ thao tác là $O(n)$
1000	4	Thời gian khấu trừ trên mỗi thao tác là $O(n)/n = O(1)$
1001	1	
1010	2	
1011	1	
1100	3	
1101	1	
1111	1	

13

### Ví dụ 2: Growing an array

#### Amortized time: Phân tích thời gian khấu trừ

Ta dùng cấu trúc dữ liệu mảng để cài đặt ngăn xếp, lưu trữ thông tin giải quyết bài toán. Tuy nhiên trong quá trình thực hiện thuật toán, nhận thấy cần tăng kích thước mảng so với dự kiến ban đầu.

Giả sử:

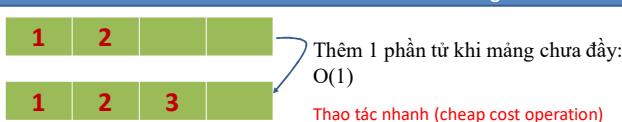
- thêm 1 phần tử vào mảng: tốn chi phí 1,
- xóa 1 phần tử khỏi mảng: tốn chi phí 1,
- chi phí thay đổi kích thước của mảng = số lượng phần tử cần di chuyển từ mảng cũ sang mảng mới

Câu hỏi: Nếu mỗi lần tăng, ta tăng kích thước mảng lên gấp đôi. (Giả sử kích thước ban đầu của mảng là  $k=1$ ). Tính thời gian khấu trừ của thuật toán khi thực hiện  $n$  lần thao tác thêm một phần tử vào mảng.

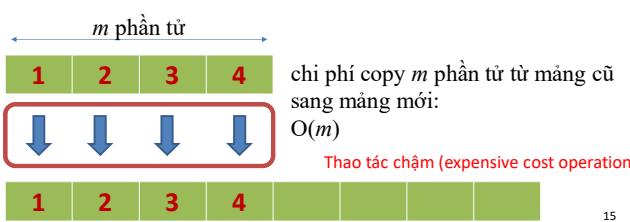
NGUYỄN KHÁNH PHƯƠNG 14  
KHMT – SOICT- DHBK HN

### Ví dụ 2: Growing an array

#### Amortized time: Phân tích thời gian khấu trừ



Mảng đầy, cần tăng kích thước mảng lên gấp đôi:



### Ví dụ 2: Growing an array

#### Amortized time: Phân tích thời gian khấu trừ

Chi phí chèn  $n$  phần tử vào mảng ( $n=32$ ): mỗi lần chèn 1 phần tử

$$O(\sum \text{chi phí các thao tác nhanh})$$

+

$$\Sigma \text{chi phí các thao tác chậm}$$

Item Number	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Operations	1	1	1	4	1	1	1	8	1	1	1	1	1	1	1	16
Item Number	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
Operations	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	32

### Ví dụ 2: Growing an array

Amortized time: Phân tích thời gian khâu trù

Chi phí chèn  $n$  phần tử vào mảng ( $n=32$ ): mỗi lần chèn 1 phần tử

- Tổng chi phí thao tác nhanh:
  - Mỗi thao tác:  $O(1)$
  - Có khoảng gần  $n$  thao tác
$$\Rightarrow \sum \text{chi phí các thao tác nhanh} = O(n)$$

Item Number	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Operations	1	1	1	4	1	1	1	8	1	1	1	1	1	1	1	16
Item Number	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
Operations	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	32

### Ví dụ 2: Growing an array

Amortized time: Phân tích thời gian khâu trù

Chi phí chèn  $n$  phần tử vào mảng ( $n=32$ ): mỗi lần chèn 1 phần tử vào mảng

- Tổng chi phí thao tác chậm (nếu tính kích thước mảng khởi tạo ban đầu = 1):
  - Khi mảng có 1 phần tử: tổng chi phí thao tác chậm = 1
  - Khi mảng có 2 phần tử: tổng chi phí thao tác chậm =  $1+2=3 < 2*2$
  - Khi mảng có 4 phần tử: tổng chi phí thao tác chậm =  $3+4=7 < 2*4$
  - Khi mảng có 8 phần tử: tổng chi phí thao tác chậm =  $7+8=15 < 2*8$
  - Khi mảng có 16 phần tử: tổng chi phí thao tác chậm =  $15+16=31 < 2*16$
  - Khi mảng có 32 phần tử: tổng chi phí thao tác chậm =  $31+32=63 < 2*32$
$$\Rightarrow \sum \text{chi phí các thao tác chậm} = 63 \text{ nếu } n=32$$

Item Number	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Cost	1	2	1	4	1	1	1	8	1	1	1	1	1	1	1	16
Item Number	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
Cost	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	32

18

### Ví dụ 2: Growing an array

Amortized time: Phân tích thời gian khâu trù

Chi phí chèn  $n$  phần tử vào mảng ( $n=32$ ): mỗi lần chèn 1 phần tử vào mảng

- Tổng chi phí thao tác chậm (nếu tính kích thước mảng khởi tạo ban đầu = 1):
  - Khi mảng có 1 phần tử: tổng chi phí thao tác chậm = 1
  - Khi mảng có 2 phần tử: tổng chi phí thao tác chậm =  $1+2=3 < 2*2$
  - Khi mảng có 4 phần tử: tổng chi phí thao tác chậm =  $3+4=7 < 2*4$
  - Khi mảng có 8 phần tử: tổng chi phí thao tác chậm =  $7+8=15 < 2*8$
  - Khi mảng có 16 phần tử: tổng chi phí thao tác chậm =  $15+16=31 < 2*16$
  - Khi mảng có 32 phần tử: tổng chi phí thao tác chậm =  $31+32=63 < 2*32$
$$\Rightarrow \sum \text{chi phí các thao tác chậm} < 2 * \text{tổng số thao tác} = 2n$$

$$\Rightarrow \sum \text{chi phí các thao tác chậm} = O(2n) = O(n)$$

Item Number	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Cost	1	2	1	4	1	1	1	8	1	1	1	1	1	1	1	16
Item Number	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
Cost	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	32

19

### Ví dụ 2: Growing an array

Amortized time: Phân tích thời gian khâu trù

Chi phí chèn  $n$  phần tử vào mảng ( $n=32$ ): mỗi lần chèn 1 phần tử vào mảng

$$O(\sum \text{chi phí các thao tác nhanh}) + O(\sum \text{chi phí các thao tác chậm}) = O(n+n) = O(n)$$

$$\text{Amortized time} = \frac{O(\text{Total cost})}{\text{number of operations}} = \frac{O(n)}{n} = O(1)$$

20

## Ví dụ 2: Growing an array

Amortized time: Phân tích thời gian khai trừ

Ta dùng cấu trúc dữ liệu mảng để cài đặt ngăn xếp, lưu trữ thông tin giải quyết bài toán. Tuy nhiên trong quá trình thực hiện thuật toán, nhận thấy cần tăng kích thước mảng so với ban đầu.

Giả sử:

- thêm 1 phần tử vào mảng: tốn chi phí 1,
- xóa 1 phần tử khỏi mảng: tốn chi phí 1,
- chi phí thay đổi kích thước của mảng = số lượng phần tử cần di chuyển

Câu hỏi 1: Nếu mỗi lần tăng, ta tăng kích thước mảng lên gấp đôi. (Giả sử kích thước ban đầu của mảng là  $k = 1$ ). Tính thời gian khai trừ của thuật toán khi thực hiện  $n$  lần thao tác thêm một phần tử vào mảng.

**Câu hỏi 2 (BTVN): Nếu mỗi lần tăng, ta tăng kích thước mảng lên 1. (Giả sử kích thước ban đầu của mảng là  $k = 1$ ). Tính thời gian khai trừ của thuật toán khi thực hiện  $n$  lần thao tác thêm một phần tử vào mảng.**

21

## Bài tập: Vito's family

The world-known gangster Vito Deadstone is moving to New York. He has a very big family there, all of them living in Lamafia Avenue. Since he will visit all his relatives very often, he is trying to find a house close to them.

Vito wants to minimize the total distance to all of them and has blackmailed you to write a program that solves his problem.

### Input

The input consists of several test cases. The first line contains the number of test cases.

For each test case you will be given the integer number of relatives  $r$  ( $0 < r < 500$ ) and the street numbers (also integers)  $s_1, s_2, \dots, s_r$ , where they live ( $0 < s_i < 30000$ ). Note that several relatives could live in the same street number.

### Output

For each test case your program must write the minimal sum of distances from the optimal Vito's house to each one of his relatives. The distance between two street numbers  $s_i$  and  $s_j$  is  $d_{ij} = |s_i - s_j|$ .

#### Sample Input

```
2  
2 2 4  
3 2 4 6
```

Trong số  $r$  nhà của họ hàng, tìm xem Vito nên ở nhà của họ hàng nào để tổng quãng đường mà Vito sẽ đi thăm cả  $r$  họ hàng này là nhỏ nhất

#### Sample Output

```
2  
4
```

22

## Bài tập: Vito's family

### • Phương pháp 1: Duyệt toàn bộ (brute force): $O(r^2)$

- Khởi tạo: `min_Distance = 0;`
- Với mỗi phần tử  $i$  trong mảng (for  $i = 0$  to  $r-1$ ) coi đó là nơi Vito sẽ ở
  - Tính khoảng cách từ  $i$  đến tất cả các phần tử  $j$  của mảng:  
`for i = 0 to r-1`  
`{`  
 `distance = 0;`  
 `for j = 0 to r-1`  
 `distance += |si - sj|;`
  - So sánh để tìm ra `min_Distance`:  
`if (min_Distance < distance) min_Distance = distance;`

➔ Xảy ra **time limit exceeded** khi kích thước dữ liệu đầu vào lớn

Cải tiến:  $O(r\log r)$  hoặc  $O(r)$  ???

23

## Nội dung chương 2

### 1. Duyệt toàn bộ

### 2. Thuật toán quay lui

### 3. Thuật toán nhánh cận

NGUYỄN KHÁNH PHƯƠNG  
KHMT – SOICT- ĐHBK HN

## 2. Thuật toán quay lui

### 2.1. Sơ đồ thuật toán quay lui

### 2.2. Một số ví dụ minh họa

NGUYỄN KHÁNH PHƯƠNG 25  
KHMT – SOICT - DHBK HN

## 2. Thuật toán quay lui

### 2.1. Sơ đồ thuật toán quay lui

### 2.2. Một số ví dụ minh họa

NGUYỄN KHÁNH PHƯƠNG 26  
KHMT – SOICT - DHBK HN

### 2.1. Sơ đồ thuật toán quay lui

- Bài toán liệt kê (Q):** Cho  $A_1, A_2, \dots, A_n$  là các tập hữu hạn. Ký hiệu  $A = A_1 \times A_2 \times \dots \times A_n = \{(a_1, a_2, \dots, a_n) : a_i \in A_i, i=1, 2, \dots, n\}$ .

Giả sử  $P$  là tính chất cho trên  $A$ . Vấn đề đặt ra là liệt kê tất cả các phần tử của  $A$  thoả mãn tính chất  $P$ :

$$D = \{ a = (a_1, a_2, \dots, a_n) \in A : a \text{ thoả mãn tính chất } P \}.$$

- Các phần tử của tập  $D$  được gọi là các **lời giải chấp nhận được**.

NGUYỄN KHÁNH PHƯƠNG  
KHMT – SOICT - DHBK HN

### 2.1. Sơ đồ thuật toán quay lui

Tất cả các bài toán liệt kê tổ hợp cơ bản đều có thể phát biểu dưới dạng bài toán liệt kê (Q).

Ví dụ:

- Bài toán liệt kê xâu nhị phân độ dài  $n$  dẫn về việc liệt kê các phân tử của tập

$$B^n = \{(a_1, \dots, a_n) : a_i \in \{0, 1\}, i=1, 2, \dots, n\}.$$

- Bài toán liệt kê các tập con  $m$  phần tử của tập  $N = \{1, 2, \dots, n\}$  đòi hỏi liệt kê các phân tử của tập:

$$S(m,n) = \{(a_1, \dots, a_m) \in N^m : 1 \leq a_1 < \dots < a_m \leq n\}.$$

- Tập các hoán vị của các số tự nhiên  $1, 2, \dots, n$  là tập

$$\Pi_n = \{(a_1, \dots, a_n) \in N^n : a_i \neq a_j ; i \neq j\}.$$

NGUYỄN KHÁNH PHƯƠNG  
KHMT – SOICT - DHBK HN

## Lời giải bộ phận

Lời giải của bài toán là bộ có thứ tự gồm  $n$  thành phần  $(a_1, a_2, \dots, a_n)$ , trong đó  $a_i \in A_i$ ,  $i = 1, 2, \dots, n$ .

**Định nghĩa.** Ta gọi lời giải bộ phận cấp  $k$  ( $0 \leq k \leq n$ ) là bộ có thứ tự gồm  $k$  thành phần

$$(a_1, a_2, \dots, a_k),$$

trong đó  $a_i \in A_i$ ,  $i = 1, 2, \dots, k$ .

- Khi  $k = 0$ , lời giải bộ phận cấp 0 được ký hiệu là () và còn được gọi là lời giải rỗng.
- Nếu  $k = n$ , ta có lời giải đầy đủ hay đơn giản là một lời giải của bài toán.

NGUYỄN KHÁNH PHƯƠNG  
KHMT – SOICT - DHBK HN

## 2.1. Sơ đồ thuật toán quay lui

Thuật toán quay lui được xây dựng dựa trên việc xây dựng dần từng thành phần của lời giải bằng cách **thứ tự các khía cạnh có thể**.

- Thuật toán bắt đầu từ lời giải rỗng () .
- Trên cơ sở tính chất  $P$  ta xác định được những phần tử nào của tập  $A_1$  có thể chọn vào vị trí thứ nhất của lời giải. Những phần tử như vậy ta sẽ gọi là những ứng cử viên (viết tắt là UCV) vào vị trí thứ nhất của lời giải. Ký hiệu tập các UCV vào vị trí thứ nhất của lời giải là  $S_1$ . Lấy  $a_1 \in S_1$ , bổ sung nó vào lời giải rỗng đang có ta thu được lời giải bộ phận cấp 1:  $(a_1)$ .

NGUYỄN KHÁNH PHƯƠNG  
KHMT – SOICT - DHBK HN

## 2.1. Sơ đồ thuật toán quay lui

- Bước tổng quát: Giả sử ta đang có lời giải bộ phận cấp  $k-1$ :

$$(a_1, a_2, \dots, a_{k-1}),$$

cần xây dựng lời giải bộ phận cấp  $k$ :

$$(a_1, a_2, \dots, a_{k-1}, a_k)$$

- Trên cơ sở tính chất  $P$ , ta xác định được những phần tử nào của tập  $A_k$  có thể chọn vào vị trí thứ  $k$  của lời giải.
- Những phần tử như vậy ta sẽ gọi là những ứng cử viên (viết tắt là UCV) vào vị trí thứ  $k$  của lời giải khi  $k-1$  thành phần đầu của nó đã được chọn là  $(a_1, a_2, \dots, a_{k-1})$ . Ký hiệu tập các ứng cử viên này là  $S_k$ .
- Xét 2 tình huống:
  - $S_k \neq \emptyset$
  - $S_k = \emptyset$

NGUYỄN KHÁNH PHƯƠNG  
KHMT – SOICT - DHBK HN

## 2.1. Sơ đồ thuật toán quay lui

- $S_k \neq \emptyset$ : Lấy  $a_k \in S_k$  bổ sung nó vào lời giải bộ phận cấp  $k-1$  đang có  $(a_1, a_2, \dots, a_{k-1})$  ta thu được lời giải bộ phận cấp  $k$   $(a_1, a_2, \dots, a_{k-1}, a_k)$ . Khi đó
  - Nếu  $k = n$  thì ta thu được một lời giải,
  - Nếu  $k < n$ , ta tiếp tục đi xây dựng thành phần thứ  $k+1$  của lời giải.

- $S_k = \emptyset$ : Điều đó có nghĩa là lời giải bộ phận  $(a_1, a_2, \dots, a_{k-1})$  không thể tiếp tục phát triển thành lời giải đầy đủ. Trong tình huống này ta **quay trở lại** tìm ứng cử viên mới vào vị trí thứ  $k-1$  của lời giải (chú ý: ứng cử viên mới này nằm trong tập  $S_{k-1}$ )
  - Nếu tìm thấy UCV như vậy, thì bổ sung nó vào vị trí thứ  $k-1$  rồi lại tiếp tục đi xây dựng thành phần thứ  $k$ .
  - Nếu không tìm được thì ta lại **quay trở lại** thêm một bước nữa tìm UCV mới vào vị trí thứ  $k-2$ , ... Nếu quay lại tận lời giải rỗng mà vẫn không tìm được UCV mới vào vị trí thứ 1, thì thuật toán kết thúc.

NGUYỄN KHÁNH PHƯƠNG  
KHMT – SOICT - DHBK HN

Cây liệt kê lời giải theo thuật toán quay lui	Thuật toán quay lui (đệ qui)	Thuật toán quay lui (không đệ qui)
<p>Tập UCV <math>S_1</math></p> <p>Tập UCV <math>S_2</math> khi đã có <math>(a_1)</math></p> <p>Tập UCV <math>S_3</math> khi đã có <math>(a_1, a_2)</math></p> <p>Tập UCV <math>S_4</math> khi đã có <math>(a_1, a_2, a_3)</math></p> <p><math>a_4</math></p> <p><math>a'_4</math></p> <p><math>(a_1, a_2, a_3, a_4)</math></p> <p><math>(a_1, a_2, a_3, a')</math></p> <p>Gốc (lời giải rỗng)</p>	<pre> void Try(int k) {     &lt;Xây dựng <math>S_k</math> là tập chứa các ứng cử viên cho vị trí     thứ <math>k</math> của lời giải&gt;;     for <math>y \in S_k</math> //Với mỗi UCV <math>y</math> từ <math>S_k</math>     {         <math>a_y = y</math>;         if <math>(k == n)</math> then &lt;Ghi nhận lời giải <math>(a_1, a_2, \dots, a_k)</math>&gt;;         else Try(<math>k+1</math>);         Tra các biến về trạng thái cũ;     } } </pre> <p>Lệnh gọi để thực hiện thuật toán quay lui là: Try(1);</p> <ul style="list-style-type: none"> <li>Nếu chỉ cần tìm một lời giải thì cần tìm cách chấm dứt các thủ tục gọi đệ qui lồng nhau sinh bởi lệnh gọi Try(<math>i</math>) sau khi ghi nhận được lời giải đầu tiên.</li> <li>Nếu kết thúc thuật toán mà ta không thu được một lời giải nào thì điều đó có nghĩa là bài toán không có lời giải.</li> </ul>	<pre> void Backtracking () {     k=1;     &lt;Xây dựng <math>S_1</math>&gt;;     while (<math>k &gt; 0</math>)     {         while (<math>S_k \neq \emptyset</math>)         {             <math>a_k \leftarrow S_k</math>; // Lấy <math>a_k</math> từ <math>S_k</math>             if <math>(k == n)</math> then &lt;Ghi nhận <math>(a_1, a_2, \dots, a_k)</math>&gt;;             else {                 <math>k = k+1</math>;                 &lt;Xây dựng <math>S_k</math>&gt;;             }         }         k = k - 1; // Quay lui     } } </pre> <p>Lệnh gọi để thực hiện thuật toán quay lui là: Backtracking();</p>

Hai vấn đề mấu chốt
<ul style="list-style-type: none"> <li>Để cài đặt thuật toán quay lui giải các bài toán tổ hợp cụ thể ta cần giải quyết hai vấn đề cơ bản sau:       <ul style="list-style-type: none"> <li>Tìm thuật toán xây dựng các tập UCV <math>S_k</math></li> <li>Tìm cách mô tả các tập này để có thể cài đặt thao tác liệt kê các phần tử của chúng (cài đặt vòng lặp qui ước <math>\text{for } y \in S_k</math>).</li> </ul> </li> <li>Hiệu quả của thuật toán liệt kê phụ thuộc vào việc ta có xác định được chính xác các tập UCV này hay không.</li> </ul>

NGUYỄN KHÁNH PHƯƠNG  
KHMT – SOICT- DHBK HN

Chú ý
<ul style="list-style-type: none"> <li>Nếu độ dài của lời giải là không biết trước và các lời giải cũng không nhất thiết phải có cùng độ dài.</li> </ul> <pre> void Try(int k) {     &lt;Xây dựng <math>S_k</math> là tập chứa các ứng cử viên cho vị trí     thứ <math>k</math> của lời giải&gt;;     for <math>y \in S_k</math> //Với mỗi UCV <math>y</math> từ <math>S_k</math>     {         <math>a_y = y</math>;         if <math>(k == n)</math> then &lt;Ghi nhận lời giải <math>(a_1, a_2, \dots, a_k)</math>&gt;;         else Try(<math>k+1</math>);         Tra các biến về trạng thái cũ;     } } </pre> <ul style="list-style-type: none"> <li>Khi đó chỉ cần sửa lại câu lệnh       <pre> if <math>(k == n)</math> then &lt;Ghi nhận lời giải <math>(a_1, a_2, \dots, a_k)</math>&gt;; else Try(<math>k+1</math>);       </pre>       thành       <pre> if &lt;<math>(a_1, a_2, \dots, a_k)</math> là lời giải&gt; then &lt;Ghi nhận <math>(a_1, a_2, \dots, a_k)</math>&gt;; else Try(<math>k+1</math>);       </pre> </li> </ul> <p>⇒ Cần xây dựng hàm nhận biết <math>(a_1, a_2, \dots, a_k)</math> đã là lời giải hay chưa.</p>

NGUYỄN KHÁNH PHƯƠNG  
KHMT – SOICT- DHBK HN

## 2. Thuật toán quay lui

2.1. Sơ đồ thuật toán quay lui

### 2.2. Một số ví dụ minh họa

NGUYỄN KHÁNH PHƯƠNG 37  
KHMT – SOICT - DHBK HN

## 2.2. Một số ví dụ minh họa

1. Liệt kê xâu nhị phân độ dài  $n$
2. Liệt kê các  $m$ -tập con của  $n$ -tập
3. Liệt kê hoán vị
4. Bài toán xếp hậu
5. Bài toán tìm nghiệm nguyên

NGUYỄN KHÁNH PHƯƠNG 38  
KHMT – SOICT - DHBK HN

## 2.2. Một số ví dụ minh họa

### 1. Liệt kê xâu nhị phân độ dài $n$

2. Liệt kê các  $m$ -tập con của  $n$ -tập

3. Liệt kê hoán vị

4. Bài toán xếp hậu

5. Bài toán tìm nghiệm nguyên

NGUYỄN KHÁNH PHƯƠNG 39  
KHMT – SOICT - DHBK HN

### Ví dụ 1: Liệt kê xâu nhị phân độ dài $n$

- Bài toán liệt kê xâu nhị phân độ dài  $n$  dẫn về việc liệt kê các phần tử của tập $B^n = \{(b_1, \dots, b_n) : b_i \in \{0, 1\}, i=1, 2, \dots, n\}.$
- Ta xét cách giải quyết hai vấn đề cơ bản để cài đặt thuật toán quay lui:
  - Xây dựng tập ứng cử viên  $S_k$ : Rõ ràng ta có  $S_1 = \{0, 1\}$ . Giả sử đã có xâu nhị phân cấp  $k-1$   $(b_1, \dots, b_{k-1})$ , khi đó rõ ràng  $S_k = \{0, 1\}$ . Như vậy, tập các UCV vào các vị trí của lời giải đã được xác định.
  - Cài đặt vòng lặp liệt kê các phần tử của  $S_k$ : ta có thể sử dụng vòng lặp for

`for (y=0;y<=1;y++)`

NGUYỄN KHÁNH PHƯƠNG  
KHMT – SOICT - DHBK HN

## Chương trình trên C++ (Đệ qui)

```
#include <iostream>
using namespace std;
int n, count;
int b[100];

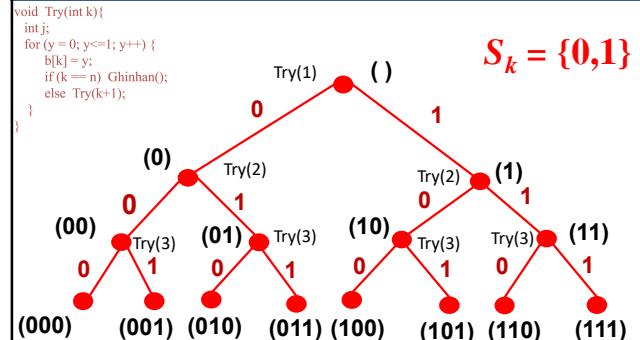
void Ghinhan() {
    int i, j;
    count++;
    cout<<"Xau thu " << count <<": ";
    for (i=1 ; i<= n ; i++) {
        j=b[i];
        cout<<j<< " ";
    }
    cout<<endl;
}

void Try(int k) {
    int j;
    for (j = 0; j<=1; j++) {
        b[k] = j;
        if (k == n) Ghinhan();
        else Try(k+1);
    }
}

int main() {
    cout<<"Nhập n = "; cin>>n;
    count = 0; Try(1);
    cout<<"Số luồng xau = "<<count<<endl;
}
```

NGUYỄN KHÁNH PHƯƠNG  
KHMT – SOICT - DHBK HN

## Cây liệt kê dãy nhị phân độ dài 3



## Chương trình trên C++ (không đệ qui)

```
#include <iostream>
using namespace std;
int n, count;
int b[100], s[100];

void Ghinhan() {
    int i, j;
    count++;
    cout<<"Xau thu " << count <<": ";
    for (i=1 ; i<= n ; i++) {
        j=b[i];
        cout<<j<< " ";
    }
    cout<<endl;
}

void Xau() {
    k=1; s[k]=0;
    while (k > 0) {
        while (s[k] <= 1) {
            b[k]=s[k];
            s[k]=s[k]+1;
            if (k==n) Ghinhan();
            else {
                k++;
                s[k]=0;
            }
        }
        k--; // Quay lui
    }
}
```

NGUYỄN KHÁNH PHƯƠNG  
KHMT – SOICT - DHBK HN

## Chương trình trên C++ (không đệ qui)

```
int main() {
    cout<<"Nhập n = "; cin>>n;
    count = 0; Xau();
    cout<<"Số luồng xau = "<<count<<endl;
}
```

NGUYỄN KHÁNH PHƯƠNG  
KHMT – SOICT - DHBK HN

## 2.2. Một số ví dụ minh họa

1. Liệt kê xâu nhị phân độ dài  $n$

### 2. Liệt kê các $m$ -tập con của $n$ -tập

3. Liệt kê hoán vị

4. Bài toán xếp hậu

5. Bài toán tìm nghiệm nguyên

NGUYỄN KHÁNH PHƯƠNG 45  
KHMT – SOICT - DHBK HN

## Ví dụ 2. Liệt kê các $m$ -tập con của $n$ -tập

**Bài toán:** Liệt kê các tập con  $m$  phần tử của tập  $N = \{1, 2, \dots, n\}$ .

**Ví dụ:** Liệt kê các tập con 3 phần tử của tập  $N = \{1, 2, 3, 4, 5\}$

**Giải:**  $(1, 2, 3), (1, 2, 4), (1, 2, 5), (1, 3, 4), (1, 3, 5), (1, 4, 5), (2, 3, 4), (2, 3, 5), (2, 4, 5), (3, 4, 5)$

➔ **Bài toán dẫn về:** Liệt kê các phần tử của tập:

$$S(m, n) = \{(a_1, \dots, a_m) \in N^m : 1 \leq a_1 < \dots < a_m \leq n\}$$

NGUYỄN KHÁNH PHƯƠNG  
KHMT – SOICT - DHBK HN

## Ví dụ 2. Liệt kê các $m$ -tập con của $n$ -tập

Ta xét cách giải quyết 2 vấn đề cơ bản để cài đặt thuật toán quay lui:

### • Xây dựng tập ứng cử viên $S_k$ :

– Từ điều kiện:  $1 \leq a_1 < a_2 < \dots < a_m \leq n$

suy ra  $S_1 = \{1, 2, \dots, n-(m-1)\}$ .

– Giả sử có tập con  $(a_1, \dots, a_{k-1})$ . Từ điều kiện  $a_{k-1} < a_k < \dots < a_m \leq n$ , ta suy ra:

$S_k = \{a_{k-1}+1, a_{k-1}+2, \dots, n-(m-k)\}$ .

### • Cài đặt vòng lặp liệt kê các phần tử của $S_k$ :

**for** ( $y=a[k-1]+1; y \leq n-m+k; y++$ ) ...

NGUYỄN KHÁNH PHƯƠNG  
KHMT – SOICT - DHBK HN

## Thuật toán quay lui (đệ qui)

```
void Try(int k)
{
    <Xây dựng  $S_k$  là tập chứa các ứng cử viên cho vị trí
    thứ  $k$  của lời giải>;
    for  $y \in S_k$  //Với mỗi UCV y từ  $S_k$ 
    {
         $a_k = y$ ;
        if ( $k == n$ ) then<Ghi nhận lời giải  $(a_1, a_2, \dots, a_k)$ >;
        else Try( $k+1$ );
        Trả các biến về trạng thái cũ;
    }
}
```

Lệnh gọi để thực hiện thuật toán quay lui là:

Try(1);

- Nếu chỉ cần tìm một lời giải thì cần tìm cách chấm dứt các thủ tục gọi đệ qui lồng nhau sinh bởi lệnh gọi Try( $k+1$ ) sau khi ghi nhận được lời giải đầu tiên.
- Nếu kết thúc thuật toán mà ta không thu được một lời giải nào thì điều đó có nghĩa là bài toán không có lời giải.

```
void Try(int k)
{
    for ( $y=a[k-1]+1; y \leq n-m+k; y++$ )
    {
         $a[k] = y$ ;
        if ( $k == m$ ) GhiNhan( );
        else Try( $k+1$ );
    }
}
Try(1);
```

NGUYỄN KHÁNH PHƯƠNG  
KHMT – SOICT - DHBK HN

## Chương trình trên C++ (Đệ qui)

```
#include <iostream>
using namespace std;

int n, m, count;
int a[100];
void Ghinhhan() {
    int i;
    count++;
    cout<<"Tap con thu " <<count<<" ";
    for (i=1 ; i<= m ;i++)
        cout<<a[i]<<" ";
    cout<<endl;
}
```

```
void Try(int k){
    int y;
    for (y = a[k-1] +1; y<= n-m+k; y++) {
        a[k] = y;
        if (k==m) Ghinhhan();
        else Try(k+1);
    }
}

int main() {
    cout<<"Nhập n, m = "; cin>>n; cin>>m;
    a[0]=0; count = 0; Try(1);
    cout<<"So tap con "<<m<<" phan tu cua tap
    "<<n<<" phan tu = "<<count<<endl;
}
```

## Chương trình trên C++ (không đệ qui)

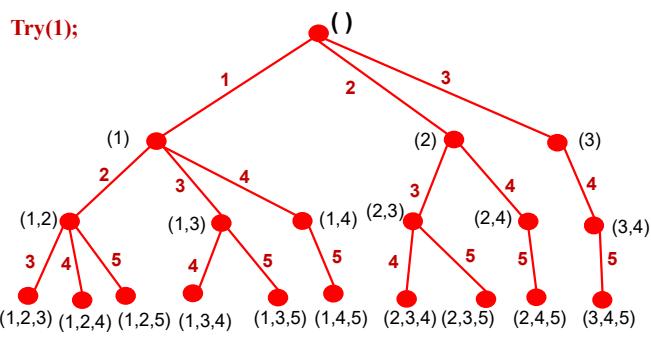
```
#include <iostream>
using namespace std;

int n, m, count,k;
int a[100], s[100];
void Ghinhhan() {
    int i;
    count++;
    cout<<"Tap con thu " <<count<<" ";
    for (i=1 ; i<= m ;i++)
        cout<<a[i]<<" ";
    cout<<endl;
}

void MSet(){
    k=1; s[k]=1;
    while(k>0){
        while (s[k]<= n-m+k) {
            a[k]=s[k]; s[k]=s[k]+1;
            if (k==m) Ghinhhan();
            else { k++; s[k]=a[k-1]+1; }
        } k--;
    }
}

int main() {
    cout<<"Nhập n, m = "; cin>>n; cin>>m;
    a[0]=0; count = 0; MSet();
    cout<<"So tap con "<<m<<" phan tu cua tap
    "<<n<<" phan tu = "<<count<<endl;
}
```

## Cây liệt kê S(5,3)



## 2.2. Một số ví dụ minh họa

- Liệt kê xâu nhị phân độ dài  $n$
- Liệt kê các  $m$ -tập con của  $n$ -tập
- Liệt kê hoán vị**
- Bài toán xếp hậu
- Bài toán tìm nghiệm nguyên

### Ví dụ 3. Liệt kê hoán vị

Tập các hoán vị của các số tự nhiên  $1, 2, \dots, n$  là tập:

$$\Pi_n = \{(x_1, \dots, x_n) \in N^n : x_i \neq x_j, i \neq j\}.$$

**Bài toán: Liệt kê tất cả các phần tử của  $\Pi_n$**

NGUYỄN KHÁNH PHƯƠNG  
KHMT – SOICT - DHBK HN

### Ví dụ 3. Liệt kê hoán vị

• Xây dựng tập ứng cử viên  $S_k$ :

- Rõ ràng  $S_1 = N$ . Giả sử ta đang có hoán vị bộ phận  $(a_1, a_2, \dots, a_{k-1})$ , từ điều kiện  $a_i \neq a_j$ , với mọi  $i \neq j$  ta suy ra

$$S_k = N \setminus \{a_1, a_2, \dots, a_{k-1}\}.$$

NGUYỄN KHÁNH PHƯƠNG  
KHMT – SOICT - DHBK HN

### Mô tả $S_k$

Xây dựng hàm nhận biết UCV:

```
bool UCV(int y, int k)
{
    //UCV nhận giá trị true khi và chỉ khi y ∈ S_k
    int i;
    for (i=1; i++ ; i<=k-1)
        if (y == a[i]) return false;
    return true;
}
```

NGUYỄN KHÁNH PHƯƠNG  
KHMT – SOICT - DHBK HN

### Liệt kê hoán vị

• Cài đặt vòng lặp liệt kê các phần tử của  $S_k$ :

```
for (j=1; j++; j<=n)
    if (UCV(y, k))
    {
        //y là UCV vào vị trí k
        ...
    }
```

## Chương trình trên C++

```
#include <iostream>
using namespace std;

int n, m, count;
int a[100];

int Ghinhhan() {
    int i, j;
    count++;
    cout << "Hoan vi thu " << count << ": ";
    for (i=1 ; i<= n ; i++)
        cout << a[i] << " ";
    cout << endl;
}
```

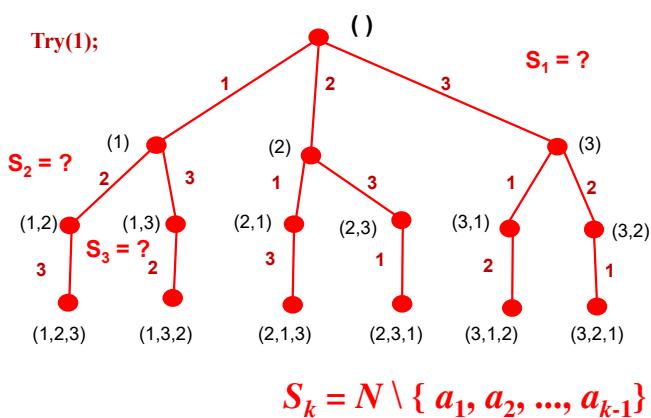
```
bool UCV(int y, int k)
{
    int i;
    for (i=1; i<=k-1; i++)
        if (y == a[i]) return false;
    return true;
}
```

NGUYỄN KHÁNH PHƯƠNG  
KHMT – SOICT - DHBK HN

```
void Try(int k)
{
    int y;
    for (y = 1; y<=n; y++)
        if (UCV(y,k))
            { a[k] = y;
              if (k==n) Ghinhhan();
              else Try(k+1);
            }
}
```

```
int main() {
    cout << ("Nhập n = "); cin >> n;
    count = 0; Try(1);
    cout << "So hoan vi = " << count;
}
```

## Cây liệt kê hoán vị của 1, 2, 3



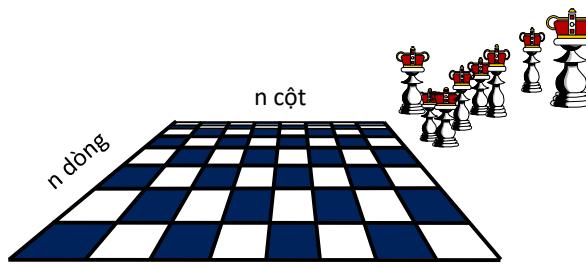
## 2.2. Một số ví dụ minh họa

- Liệt kê xâu nhị phân độ dài  $n$
- Liệt kê các  $m$ -tập con của  $n$ -tập
- Liệt kê hoán vị
- Bài toán xếp hậu**
- Bài toán tìm nghiệm nguyên

NGUYỄN KHÁNH PHƯƠNG 60  
KHMT – SOICT - DHBK HN

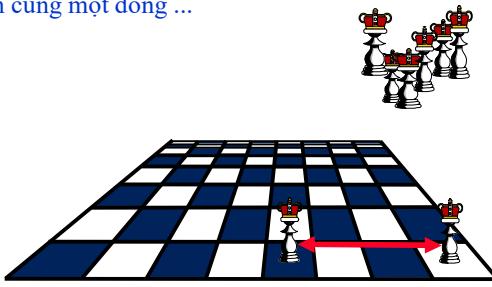
#### Ví dụ 4. Bài toán xếp hậu

- Liệt kê tất cả các cách xếp  $n$  quân Hậu trên bàn cờ  $n \times n$  sao cho chúng không ăn được lẫn nhau, nghĩa là sao cho không có hai con nào trong số chúng nằm trên cùng một dòng hay một cột hay một đường chéo của bàn cờ.



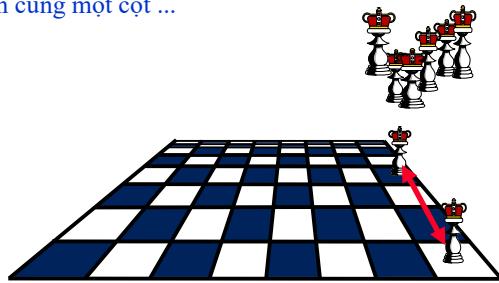
#### The n-Queens Problem

Hai con hậu bất kỳ không được xếp trên cùng một dòng ...



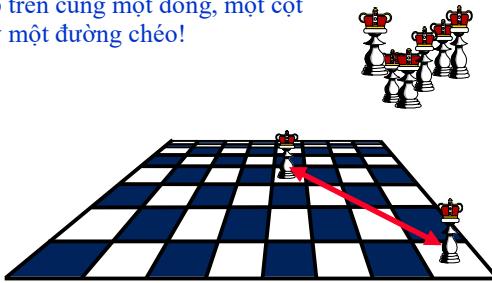
#### The n-Queens Problem

Hai con hậu bất kỳ không được xếp trên cùng một cột ...



#### The n-Queens Problem

Hai con hậu bất kỳ không được xếp trên cùng một dòng, một cột hay một đường chéo!



## Bài toán xếp hậu: Duyệt toàn bộ

- Cần xếp  $N$  quân hậu lên bàn cờ có tất cả  $N \times N$  ô, hỏi có tất cả bao nhiêu cách xếp?
  - Quân thứ 1 có thể đặt vào 1 trong số  $N \times N$  ô  $\rightarrow$  có  $N^2$  cách
  - Sau khi đặt quân thứ 1, tiến hành đặt quân thứ 2: bỏ qua ô đã đặt quân thứ 1  $\rightarrow$  chỉ còn  $N^2 - 1$  ô có thể đặt quân thứ 2  $\rightarrow$  có  $N^2 - 1$  cách
  - ...

$\Rightarrow$  tổng cộng có tất cả  $(N^2)!$  cách đặt. Với mỗi cách ta kiểm tra xem có thỏa mãn điều kiện không có quân nào cùng dòng/cột/đường chéo; nếu thỏa mãn thì thu được lời giải tương ứng.

## Bài toán xếp hậu: Duyệt toàn bộ

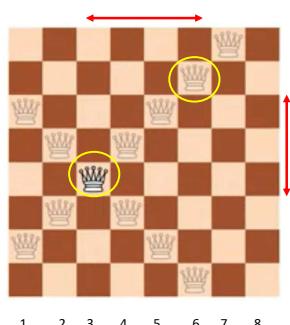
- Cần xếp  $N$  quân hậu lên bàn cờ có tất cả  $N \times N$  ô, hỏi có tất cả bao nhiêu cách xếp?
  - Quân thứ 1 có thể đặt vào 1 trong số  $N \times N$  ô  $\rightarrow$  có  $N^2$  cách
  - Sau khi đặt quân thứ 1, tiến hành đặt quân thứ 2: bỏ qua ô đã đặt quân thứ 1  $\rightarrow$  chỉ còn  $N^2 - 1$  ô có thể đặt quân thứ 2  $\rightarrow$  có  $N^2 - 1$  cách
  - ...

$\Rightarrow$  tổng cộng có tất cả  $(N^2)!$  cách đặt. Với mỗi cách ta kiểm tra xem có thỏa mãn điều kiện không có quân nào cùng dòng/cột/đường chéo; nếu thỏa mãn thì thu được lời giải tương ứng.
- Có một cách khác: giảm không gian xét duyệt từ  $(N^2)!$  xuống còn  $N!$

## Bài toán xếp hậu: Duyệt toàn bộ

Kiểm tra hai quân hậu đặt ở ô  $(i_1, j_1)$  và ô  $(i_2, j_2)$

- không thuộc cùng một dòng:  $i_1 \neq i_2$
- không thuộc cùng một cột:  $j_1 \neq j_2$
- không thuộc cùng một đường chéo:  $|i_1 - i_2| \neq |j_1 - j_2|$



NGUYỄN KHÁNH PHƯƠNG  
KHMT – SOICT- ĐHBK HN

## Biểu diễn lời giải

- Đánh số các cột và dòng của bàn cờ từ 1 đến  $n$ .
- Một cách xếp hậu có thể biểu diễn bởi bộ có  $n$  thành phần  $(a_1, a_2, \dots, a_n)$ , trong đó  $a_i$  là toạ độ cột của con hậu ở dòng  $i$ .
- Các điều kiện đặt ra đối với bộ  $(a_1, a_2, \dots, a_n)$ :

  - $a_i \neq a_j$ , với mọi  $i \neq j$  (nghĩa là hai con hậu ở hai dòng  $i$  và  $j$  không được nằm trên cùng một cột);
  - $|a_i - a_j| \neq |i - j|$ , với mọi  $i \neq j$  (nghĩa là hai con hậu ở hai ô  $(i, a_i)$  và  $(j, a_j)$  không được nằm trên cùng một đường chéo).

NGUYỄN KHÁNH PHƯƠNG  
KHMT – SOICT- ĐHBK HN

## Phát biểu bài toán

- Như vậy bài toán xếp Hậu dẫn về bài toán liệt kê các phần tử của tập:

$$D = \{(a_1, a_2, \dots, a_n) \in N^n : a_i \neq a_j \text{ và } |a_i - a_j| \neq |i - j|, i \neq j\}$$

NGUYỄN KHÁNH PHƯƠNG  
KHMT – SOICT - ĐH BKHN

## Bài toán xếp hậu: Duyệt toàn bộ

- Có một cách khác: giảm không gian xét duyệt từ  $(N^2)$  xuống còn  $N!$ 
  - Mỗi cách xếp  $N$  quân hậu lên bàn cờ sao cho không có quân nào cùng dòng, cùng cột ~ mỗi hoán vị  $N$  phần tử



Mảng a gồm 4 phần tử  
a[1]=3; a[2]=1; a[3]=4; a[4]=2  
Đòng 1: xếp quân hậu ở cột 3  
Đòng 2: xếp quân hậu ở cột 1  
Đòng 3: xếp quân hậu ở cột 4  
Đòng 4: xếp quân hậu ở cột 2

- Chú ý: không phải hoán vị nào cũng là lời giải của bài toán xếp hậu:



Hoán vị: (3, 1, 4, 2)  
=> lời giải



Hoán vị: (3, 2, 1, 4)  
=> Không phải là lời giải

Thuật toán: Liệt kê tất cả  $N!$  hoán vị;  
kiểm tra điều kiện không cùng đường  
chéo tại mỗi hoán vị. Nếu hoán vị nào  
thỏa mãn điều kiện  $\Rightarrow$  cho ta 1 lời giải

? Làm thế nào để kiểm tra được điều kiện  
Không cùng đường chéo

Vì còn phải kiểm tra điều kiện: không có quân hậu nào cùng đường chéo

## Bài toán xếp hậu: Thuật toán quay lui

**Thuật toán:** Xếp từng quân hậu lên lần lượt từng dòng của bàn cờ: tại bước lập thứ  $k$  ( $k=1, \dots, n$ ): ta cần tìm tọa độ cột  $a_k$  để đặt quân cờ lên dòng  $k$  [tức là đặt lên ô (dòng  $k$ , cột  $a_k$ )]

- Giả sử đã xây dựng được lời giải bộ phận  $(a_1, a_2, \dots, a_{k-1})$ : tức là đã xếp được  $(k-1)$  quân hậu lần lượt vào các ô  $(1, a_1), (2, a_2), \dots, (k-1, a_{k-1})$  thỏa mãn điều kiện đề bài.
- Giờ tiếp tục xây dựng thành phần thứ  $k$  của lời giải: tức là cần tìm tọa độ cột để xếp quân hậu lên dòng thứ  $k$  của bàn cờ. Ta tiến hành như sau:

- Duyệt lần lượt từng cột  $j = 1, 2, \dots, n$ :
  - Kiểm tra xem có thể xếp quân hậu lên ô  $(k, j)$  - dòng  $k$  cột  $j$  hay không: bằng cách dùng hàm nhận biết ứng cử viên
- `UCVh(int j, int k) //check xếp quân hậu vào ô (k, j)`
- `// UCVh nhận giá trị 1 khi và chỉ khi j ∈ Sk`
- `for (i=1; i<k; i++) //duyệt qua lần lượt từ dòng 1..(k-1) là những dòng đã xếp quân hậu`
- `if ((j == a[i]) || (fabs(j-a[i]) == k-i)) return 0;`
- `return 1;`
- `}`
- `void Try(int k) //tim a[k]: cột xếp quân hậu thứ k lên dòng k`
- `{`
- `for (int j=1; j<n; j++) //duyệt qua lần lượt từ cột 1..n: xem có xếp được lên ô (k, j) không`
- `if (UCVh(j,k)) //nếu xếp được quân hậu lên ô (k, j)`
- `{`
- `a[k]=j;`
- `if (k==n) Ghinhhan(); //nếu đã xếp được đủ cả n quân hậu thì in ra lời giải`
- `else Try(k+1); //nếu ko thi tiếp tục tìm vị trí xếp quân hậu thứ k+1 lên dòng thứ k+1`
- `}`
- `}`

## Thuật toán quay lui: Hàm nhận biết ứng cử viên

```
int UCVh(int j, int k) //check xếp quân hậu vào ô (k, j)
{
    // UCVh nhận giá trị 1 khi và chỉ khi j ∈ Sk
    for (i=1; i<k; i++) //duyệt qua lần lượt từ dòng 1..(k-1) là những dòng đã xếp quân hậu
        if ((j == a[i]) || (fabs(j-a[i]) == k-i)) return 0;
    return 1;
}

void Try(int k) //tim a[k]: cột xếp quân hậu thứ k lên dòng k
{
    for (int j=1; j<n; j++) //duyệt qua lần lượt từ cột 1..n: xem có xếp được lên ô (k, j) không
        if (UCVh(j,k)) //nếu xếp được quân hậu lên ô (k, j)
    {
        a[k]=j;
        if (k==n) Ghinhhan(); //nếu đã xếp được đủ cả n quân hậu thì in ra lời giải
        else Try(k+1); //nếu ko thi tiếp tục tìm vị trí xếp quân hậu thứ k+1 lên dòng thứ k+1
    }
}
```

```

#include <bits/stdc++.h>
using namespace std;

int n, dem;
int a[20];

void Ghinhau() {
    int i;
    dem = 0;
    cout<<den<<" ";
    for (i=1; i<=n; i++) cout<<a[i]<<" ";
    cout<<endl;
}

int UCVH(int j, int k) {
    int i;
    for (i=1; i<k; i++)
        if ((j == a[i]) || (fabs(j-a[i]) == k-1)) return 0;
    return 1;
}

void Try (int k)
{
    for (int j=1; j<=n; j++)
        if (UCVH(j, k))
    {
        a[k] = j;
        if (k == n) Ghinhau();
        else Try (k+1);
    }
}

int main()
{
    cout<<"Nhap kich thuoc ban co: n "; cin>>n;
    cout<<"\n==== CAC CACH XEP QUAN HAU =====\n";
    dem = 0;
    Try (1);
    if (dem == 0) cout<<"Khong co cach xep nao!\n";
}

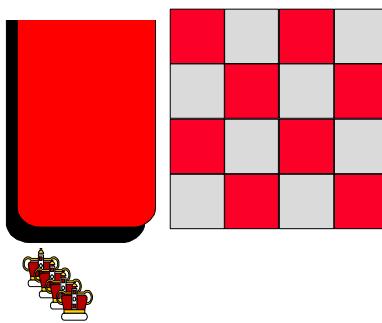
```

73

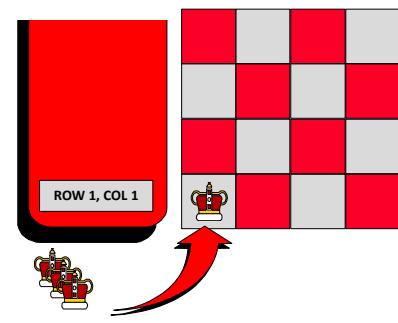
## Chú ý

- Rõ ràng là bài toán xếp hậu không phải là luôn có lời giải, chẳng hạn bài toán không có lời giải khi  $n = 2, 3$ . Do đó điều này cần được thông báo khi kết thúc thuật toán.

## Thuật toán làm việc như thế nào

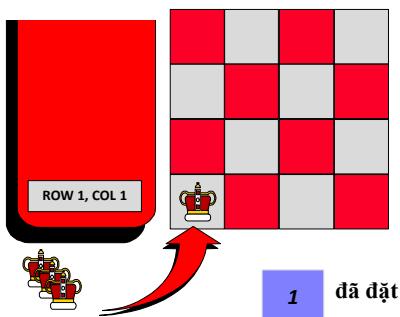


## Thuật toán làm việc như thế nào



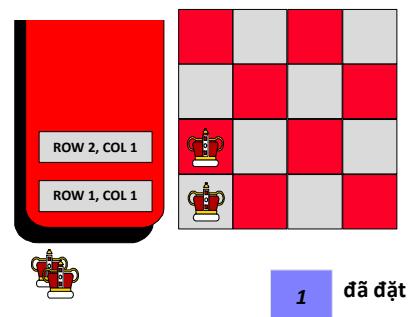
### Thuật toán làm việc như thế nào

Xếp con hậu ở dòng 1  
vào vị trí cột 1



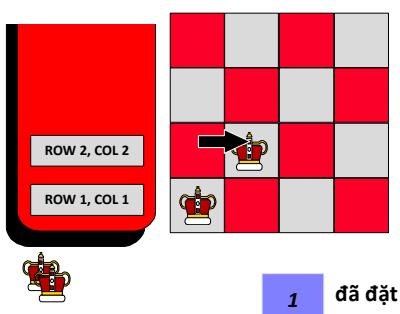
### Thuật toán làm việc như thế nào

Thử xếp con hậu ở dòng 2  
vào vị trí cột 1



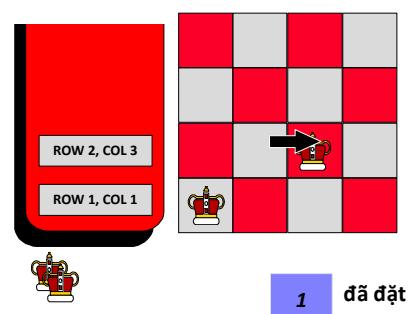
### Thuật toán làm việc như thế nào

Thử xếp con hậu ở dòng 2  
vào vị trí cột 2



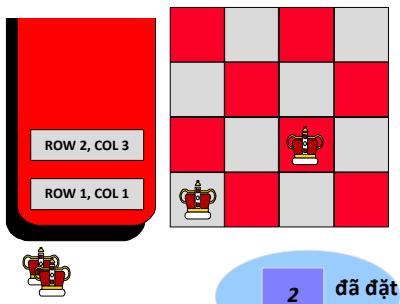
### Thuật toán làm việc như thế nào

Thử xếp con hậu ở dòng 2  
vào vị trí cột 3



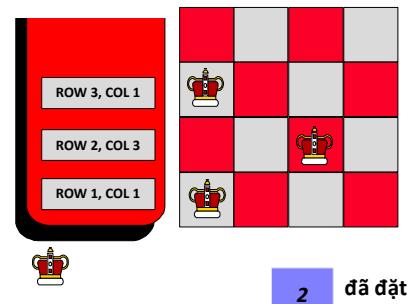
### Thuật toán làm việc như thế nào

Chấp nhận xếp con hậu ở dòng 2  
vào vị trí cột 3



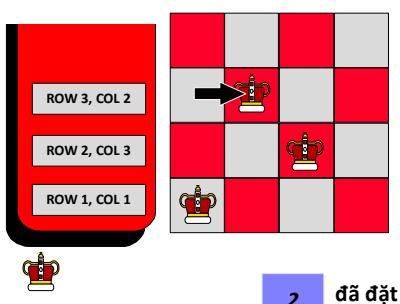
### Thuật toán làm việc như thế nào

Thử xếp con hậu ở dòng 3  
vào cột đầu tiên



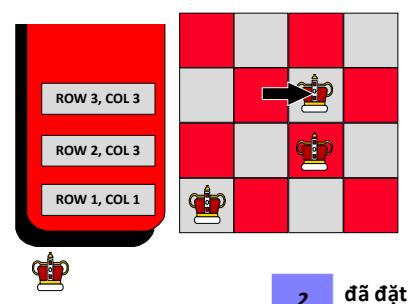
### Thuật toán làm việc như thế nào

Thử cột tiếp theo



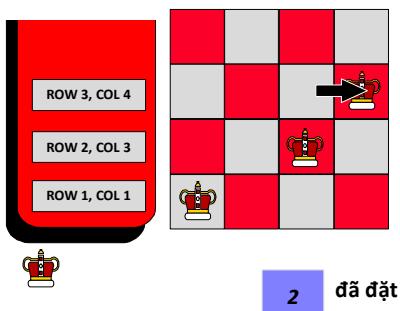
### Thuật toán làm việc như thế nào

Thử cột tiếp theo



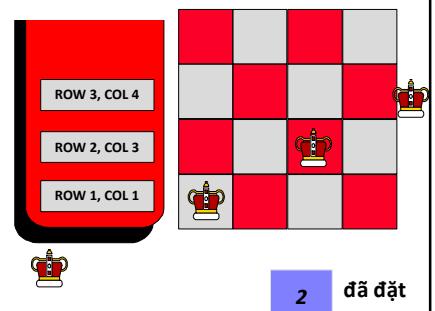
## Thuật toán làm việc như thế nào

Thứ cột tiếp theo



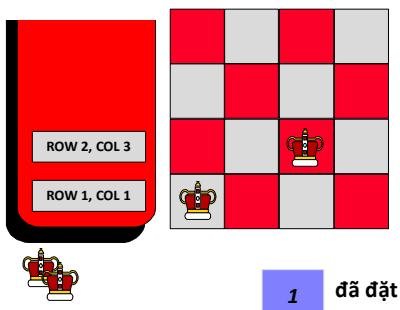
## Thuật toán làm việc như thế nào

...không có vị trí đặt  
con hậu ở dòng 3.



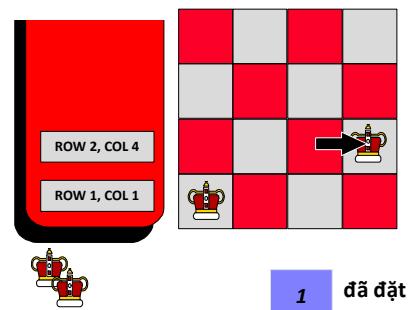
## Thuật toán làm việc như thế nào

Quay lại dịch  
chuyển con hậu ở  
dòng 2



## Thuật toán làm việc như thế nào

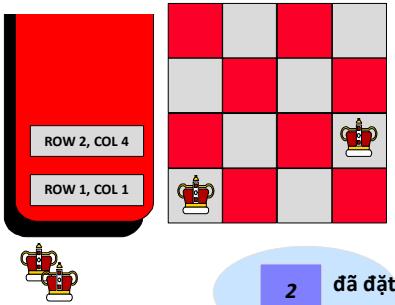
Đẩy con hậu ở dòng  
2 sang cột thứ 4.



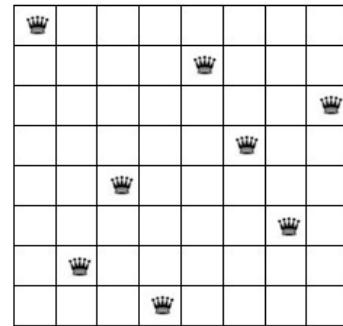
## Thuật toán làm việc như thế nào

Xếp được con hậu ở  
dòng 2 ta tiếp tục xếp  
con hậu ở dòng 3

...



## Một lời giải của bài toán xếp hậu khi $n = 8$



## 2.2. Một số ví dụ minh họa

1. Liệt kê xâu nhị phân độ dài  $n$
2. Liệt kê các  $m$ -tập con của  $n$ -tập
3. Liệt kê hoán vị
4. Bài toán xếp hậu

### **Bài toán tìm nghiệm nguyên**

91

## Ví dụ 5. Bài toán nghiệm nguyên

Bài toán chia kẹo: Cần chia  $n$  cái kẹo cho  $k$  em bé  $B_1, B_2, \dots, B_k$  (có thể có em bé không có cái kẹo nào). Hỏi có bao nhiêu cách chia khác nhau?

Gọi  $x_j$  là số kẹo chia cho em bé  $B_j, j=1, \dots, k$ . Khi đó, vấn đề đặt ra dẫn đến bài toán:

Cho  $k$  và  $n$  là các số nguyên không âm. Hỏi phương trình sau đây có bao nhiêu nghiệm nguyên không âm?

$$x_1 + x_2 + x_3 + \dots + x_k = n$$

$$x_1, x_2, \dots, x_k \geq 0$$

Số nghiệm nguyên không âm của phương trình là:

$$C(n+k-1, n) = C(n+k-1, k-1) = (n+k-1)! / n!(k-1)!$$

92

## Bài toán chia kẹo

Khi chọn ra  $n$  đối tượng từ tập X, ta đặt chúng vào  $k$  hộp sao cho đối tượng loại  $i$  sẽ được cho vào hộp  $i$ ,  $1 \leq i \leq k$ .



Vì đối tượng thuộc cùng 1 loại là giống nhau, ta dùng “0” kí hiệu cho các đối tượng trong hộp, các đối tượng ở các hộp khác nhau được phân cách bởi vách ngăn “1”.

Khi đó ta thu được dãy 0-1 độ dài  $n+(k-1)$  với đúng  $n$  số 0 và  $(k-1)$  số 1.

Ví dụ:  $k = 4, n = 7$

$$\{a, a, b, c, c, c, d\} \longrightarrow \boxed{\begin{array}{|c|c|c|c|c|} \hline 0 & 0 & 0 & 0 & 0 \\ \hline 1 & 2 & 3 & 4 & \\ \hline \end{array}} \longrightarrow 0010100010$$

4 loại đối tượng: a, b, c, d

$$\{b, b, b, b, d, d, d\} \longrightarrow \boxed{\begin{array}{|c|c|c|c|c|} \hline 0 & 0 & 0 & 0 & 0 \\ \hline 1 & 2 & 3 & 4 & \\ \hline \end{array}} \longrightarrow 1000011000$$

Bài toán chuyển thành: đếm số dãy 0-1 độ dài  $n+k-1$  với đúng  $n$  số 0 và  $(k-1)$  số 1  
 $= C(n+k-1, n) = C(n+k-1, k-1)$

93

## Ví dụ 5. Bài toán nghiệm nguyên

Cho  $k$  và  $n$  là các số nguyên không âm. Hỏi phương trình sau đây có bao nhiêu nghiệm nguyên không âm?

$$\begin{aligned} x_1 + x_2 + x_3 + \cdots + x_k &= n \\ x_1, x_2, \dots, x_k &\geq 0 \end{aligned}$$

Số nghiệm nguyên không âm của phương trình là:

$$C(n+k-1, n) = C(n+k-1, k-1) = (n+k-1)! / n!(k-1)!$$



Cho  $k$  và  $n$  là các số nguyên không âm. Hỏi phương trình sau đây có bao nhiêu nghiệm nguyên dương?

$$\begin{aligned} x_1 + x_2 + x_3 + \cdots + x_k &= n \\ x_1, x_2, \dots, x_k &> 0 \end{aligned}$$

Đặt  $y_i = x_i - 1 \geq 0 \Rightarrow y_1 + y_2 + y_3 + \cdots + y_k = n - k$

$$y_1, y_2, \dots, y_k \geq 0$$

Số nghiệm nguyên dương của phương trình là:  $(n-1)!/(n-k)!(k-1)!$

94

## Ví dụ 5. Bài toán nghiệm nguyên

### Liệt kê tất cả các nghiệm nguyên dương của phương trình

$$x_1 + x_2 + x_3 + \cdots + x_k = n$$

$$x_1, x_2, \dots, x_k > 0$$

**Thuật toán quay lui:** Tìm giá trị cho từng biến  $x_i$  của phương trình: tại bước lặp thứ  $i$  ( $i=1, \dots, n$ ): ta cần tìm giá trị của biến  $x_i$

- Giả sử đã xây dựng được lời giải bộ phận ( $x_1, x_2, \dots, x_{i-1}$ ): tức là đã biết được giá trị của ( $i-1$ ) biến đầu tiên lần lượt là  $x_1, x_2, \dots, x_{i-1}$
- Giờ tiếp tục xây dựng thành phần thứ  $i$  của lời giải: tức là cần tìm giá trị cho biến  $x_i$ . Ta tiến hành như sau:
  - Tính tổng của ( $i-1$ ) biến đầu tiên đã biết giá trị:  $M = x_1 + x_2 + \dots + x_{i-1}$
  - Tổng ( $k-i$ ) các biến từ  $x_{i+1}, \dots, x_k$  ít nhất phải =  $k-i$
  - Giá trị lớn nhất mà  $x_i$  có thể nhận:  $U = n - M - (k-i)$
  - biến  $x_i$  chỉ có thể nhận giá trị trong khoảng:  $1 \leq x_i \leq U$

## Thuật toán quay lui: Hàm nhận biết ứng cử viên

```
void Try(int i) //tim x[i]: gia tri cho bien x_i
{
  if (i==k) //chi con bien cuoi cung x_k can xác định giá trị
  {
    U = n - M; L = U;
  }
  else
  {
    U = n - M - (k-i); L = 1;
  }
  for (j = L; j <= U; j++)
  {
    x[i] = j;
    M = M + j;
    if (i == k) Ghinhanh(); //neu da co duoc toan bo k giá trị bien x_i thi in ra lời giải
    else Try(i+1); //tiếp tục xác định giá trị cho x_{i+1}
    M = M - j;
  }
}
void nghiemnguyen(int k, int n)
{
  M = 0; //luu tru tong cac bien da xác định được giá trị
  Try(1);
}
```

NGUYỄN KHÁNH PHƯƠNG  
KHMT - SOICT - ĐHBKHN

## Thuật toán quay lui: Hàm nhận biết ứng cử viên

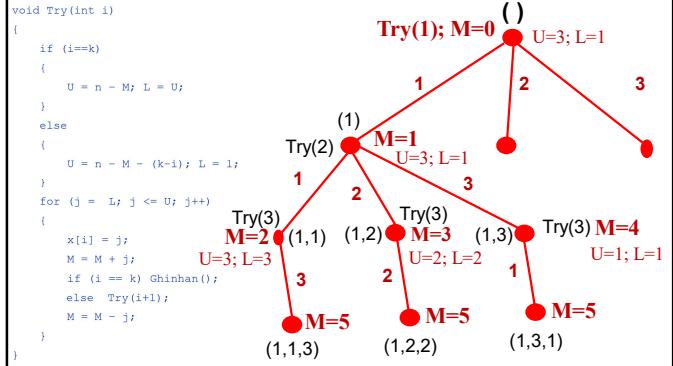
```

void Try(int i)
{
    if (i==k)
    {
        U = n - M; L = U;
    }
    else
    {
        U = n - M - (k-i); L = 1;
    }
    for (j = L; j <= U; j++)
    {
        x[i] = j;
        M = M + j;
        if (i == k) Ghinhan();
        else Try(i+1);
        M = M - j;
    }
}
void nghiemnguyen(int k, int n)
{
    M = 0; //hàu trữ tổng các biến đã xác định được giá trị
    Try(1);
}

```

NGUYỄN KHÁNH PHƯƠNG  
KHMT – SOICT - DHBK HN

## Cây liệt kê lời giải $n=5, k=3$



NGUYỄN KHÁNH PHƯƠNG  
KHMT – SOICT - DHBK HN

## Bài tập 1: The Hamming Distance problem

The Hamming distance between two strings of bits (binary integers) is the number of corresponding bit positions that differ. This can be found by using XOR on corresponding bits or equivalently, by adding corresponding bits (base 2) without a carry. For example, in the two bit strings that follow:

```

A      0 1 0 0 1 0 1 0 0 0
B      1 1 0 1 0 1 0 1 0 0
A XOR B = 1 0 0 1 1 1 1 0 0

```

The Hamming distance ( $H$ ) between these 10-bit strings is 6, the number of 1's in the XOR string.

### Input

Input consists of several datasets. The first line of the input contains the number of datasets, and it's followed by a blank line. Each dataset contains  $N$ , the length of the bit strings and  $H$ , the Hamming distance, on the same line. There is a blank line between test cases.

### Output

For each dataset print a list of all possible bit strings of length  $N$  that are Hamming distance  $H$  from the bit string containing all 0's (origin). That is, all bit strings of length  $N$  with exactly  $H$  1's printed in ascending lexicographical order.

The number of such bit strings is equal to the combinatorial symbol  $C(N, H)$ . This is the number of possible combinations of  $N - H$  zeros and  $H$  ones. It is equal to

$$\frac{N!}{(N-H)!H!}$$

This number can be very large. The program should work for  $1 \leq H \leq N \leq 16$ . Print a blank line between datasets.

### Sample Input

1

4 2

### Sample Output

```

0011
0101
0110
1001
1010
1100

```

## Bài tập 2: Optimal Slots

- <https://codeforces.com/gym/102219/problem/E>

### E. Optimal Slots

time limit per test: 2 seconds  
memory limit per test: 256 megabytes  
input: standard input  
output: standard output

The main hall of your residency is open for use by the local community and public. Since it was built on public donations, there is no charge of using it. Every weekend particularly on public holidays there are up to 50 reservations to use it for multiple events with different durations.

You have been assigned by the residents to develop a program in choosing events to get most out of allocation time per weekend and have as short unused time as possible. Program should find the event(s) which fill(s) the allocation time best and print it in the same sequence as appears in the reservation list.

### Input

Each test case consist of a single line.

The line starts with two integer  $T$  and  $N$  which is the time allocated for the hall to be used in the particular weekend and the number of events. The next  $N$  integer are the durations of the events (as appear in the reservation list). For example from the first line in sample data:  $T = 5$  hours,  $N$ , number of events = 5, first event lasts for 1 hour, second is 2 hours, third is 3 hours, fourth is 4 hours, and the last one is 5 hours.

The input process will be terminated by a line containing 0.

### Output

For each line of input value, in a single line, first, output a list of integers which are the selected events duration and another integer which is the sum of the selected events duration.

If there are multiple possible list of events, events that appear earlier in the list takes priority.

100

## Bài tập 2: Optimal Slots

- <https://codeforces.com/gym/102219/problem/E>

### Example

#### Input

```
5 5 1 2 3 4 5  
10 9 11 9 3 5 8 4 9 3 2  
16 8 12 6 11 11 13 1 10 7  
13 5 10 12 2 13 10  
28 14 18 19 26 15 18 24 7 21 14 25 2 12 9 6  
0
```

#### Output

```
1 4 5  
3 5 2 10  
6 10 16  
13 13  
19 7 2 28
```

NGUYỄN KHÁNH PHƯƠNG 101  
KHMT – SOICT - DHBK HN

## 3. Thuật toán nhánh cận

- Bài toán tối ưu tổ hợp
- Một số ví dụ về bài toán tối ưu tổ hợp
- Thuật toán nhánh cận (Branch and bound algorithm)

NGUYỄN KHÁNH PHƯƠNG  
KHMT – SOICT - DHBK HN

## 3. Thuật toán nhánh cận

- Bài toán tối ưu tổ hợp**
- Một số ví dụ về bài toán tối ưu tổ hợp
- Thuật toán nhánh cận (Branch and bound algorithm)

## 3.1. Bài toán tối ưu tổ hợp

- Trong rất nhiều vấn đề ứng dụng thực tế của tổ hợp, các cấu hình tổ hợp được gán cho một giá trị bằng số đánh giá giá trị sử dụng của cấu hình đối với mục đích sử dụng cụ thể nào đó.
- Khi đó xuất hiện bài toán: Hãy lựa chọn trong số các cấu hình tổ hợp chấp nhận được cấu hình có giá trị sử dụng tốt nhất. Các bài toán như vậy chúng ta sẽ gọi là bài toán tối ưu tổ hợp.

NGUYỄN KHÁNH PHƯƠNG  
KHMT – SOICT - DHBK HN

## Phát biểu bài toán tối ưu tổ hợp

Dưới dạng tổng quát bài toán tối ưu tổ hợp có thể phát biểu như sau:

Tìm cực tiểu (hay cực đại) của hàm  
 $f(x) \rightarrow \min (\max)$ ,  
với điều kiện  
 $x \in D$ ,  
trong đó  $D$  là tập hữu hạn phân tử.

Các thuật ngữ:

- $f(x)$  - hàm mục tiêu của bài toán,
- $x \in D$  - phương án
- $D$  - tập các phương án của bài toán.
- Thông thường tập  $D$  được mô tả như là tập các cấu hình tổ hợp thỏa mãn một số tính chất cho trước nào đó.
- Phương án  $x^* \in D$  đem lại giá trị nhỏ nhất (lớn nhất) cho hàm mục tiêu được gọi là **phương án tối ưu**, khi đó giá trị  $f^* = f(x^*)$  được gọi là **giá trị tối ưu** của bài toán.

## 3. Thuật toán nhánh cận

3.1. Bài toán tối ưu tổ hợp

### 3.2. Một số ví dụ về bài toán tối ưu tổ hợp

3.3. Thuật toán nhánh cận (Branch and bound algorithm)

NGUYỄN KHÁNH PHƯƠNG  
KHMT – SOICT - ĐHBK HN

## 3.2. Một số ví dụ về bài toán tối ưu tổ hợp

1. **Bài toán người du lịch (Traveling Salesman Problem)**
2. Bài toán cái túi (Knapsack Problem)
3. Bài toán đóng thùng (Bin Backing)
4. Bài toán định tuyến xe (Vehicle Routing Problem)
5. Bài toán lập lịch (Scheduling)
6. Bài toán xếp thời khóa biểu (Timetabling)
7. Bài toán phân bổ tài nguyên (Resource allocations)

NGUYỄN KHÁNH PHƯƠNG  
KHMT – SOICT - ĐHBK HN

Bài toán người du lịch  
(Traveling Salesman Problem – TSP)

- Một người du lịch muốn đi thăm quan  $n$  thành phố  $T_1, T_2, \dots, T_n$ .
- *Hành trình là cách đi xuất phát từ một thành phố nào đó đi qua tất cả các thành phố còn lại, mỗi thành phố đúng một lần, rồi quay trở lại thành phố xuất phát.*
- Biết  $c_{ij}$  là chi phí đi từ thành phố  $T_i$  đến thành phố  $T_j$  ( $i, j = 1, 2, \dots, n$ ),
- Tìm hành trình với tổng chi phí là nhỏ nhất.

NGUYỄN KHÁNH PHƯƠNG  
KHMT – SOICT - ĐHBK HN

### 3.2. Một số ví dụ về bài toán tối ưu tổ hợp

1. Bài toán người du lịch (Traveling Saleman Probem)
- 2. Bài toán cái túi (Knapsack Problem)**
3. Bài toán đóng thùng (Bin Backing)
4. Bài toán định tuyến xe (Vehicle Routing Problem)
5. Bài toán lập lịch (Scheduling)
6. Bài toán xếp thời khóa biểu (Timetabling)
7. Bài toán phân bổ tài nguyên (Resource allocations)

NGUYỄN KHÁNH PHƯƠNG  
KHMT – SOICT - DHBK HN

### Bài toán cái túi (Knapsack Problem)

- Một nhà thám hiểm cần đem theo một cái túi có trọng lượng không quá  $b$ .
- Có  $n$  đồ vật có thể đem theo. Đồ vật thứ  $j$  có
  - trọng lượng là  $a_j$  và
  - giá trị sử dụng là  $c_j$  ( $j = 1, 2, \dots, n$ ).
- Hỏi rằng nhà thám hiểm cần đem theo các đồ vật nào để cho tổng giá trị sử dụng của các đồ vật đem theo là lớn nhất?

NGUYỄN KHÁNH PHƯƠNG  
KHMT – SOICT - DHBK HN

### 3.2. Một số ví dụ về bài toán tối ưu tổ hợp

1. Bài toán người du lịch (Traveling Saleman Probem)
2. Bài toán cái túi (Knapsack)
- 3. Bài toán đóng thùng (Bin Backing)**
4. Bài toán định tuyến xe (Vehicle Routing Problem)
5. Bài toán lập lịch (Scheduling)
6. Bài toán xếp thời khóa biểu (Timetabling)
7. Bài toán phân bổ tài nguyên (Resource allocations)

NGUYỄN KHÁNH PHƯƠNG  
KHMT – SOICT - DHBK HN

### Bài toán đóng thùng (Bin packing)

- Có  $n$  đồ vật với trọng lượng là  $w_1, w_2, \dots, w_n$ . Cần tìm cách xếp các đồ vật này vào các cái thùng có cùng dung lượng là  $b$  sao cho số thùng cần sử dụng là nhỏ

NGUYỄN KHÁNH PHƯƠNG  
KHMT – SOICT - DHBK HN

### 3.2. Một số ví dụ về bài toán tối ưu tổ hợp

1. Bài toán người du lịch (Traveling Salesman Problem)
2. Bài toán cái túi (Knapsack)
3. Bài toán đóng thùng (Bin Backing)
- 4. Bài toán định tuyến xe (Vehicle Routing Problem)**
5. Bài toán lập lịch (Scheduling)
6. Bài toán xếp thời khóa biểu (Timetabling)
7. Bài toán phân bổ tài nguyên (Resource allocations)

NGUYỄN KHÁNH PHƯƠNG  
KHMT – SOICT - DHBK HN

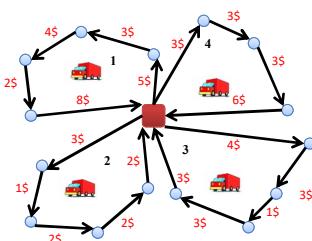
### Bài toán định tuyến xe (Vehicle Routing Problem)

- Mở rộng của bài toán Traveling Salesman Problem
- Công ty vận chuyển có một tập  $k$  xe tải đậu tại kho hàng của công ty. Mỗi ngày, công ty cần lên lịch vận chuyển một lượng hàng cho  $n$  khách hàng: mỗi khách hàng  $i$  ( $1 \leq i \leq n$ ) có lượng hàng cần giao là  $d_i$ .
  - $V = \text{kho hàng} \cup \{\text{tập } n \text{ khách hàng}\}$
  - $c_{ij}$  là chi phí vận chuyển từ  $i$  đến  $j$ , với  $i, j \in V$
  - Dung lượng hàng hóa tối đa mà mỗi xe có thể chứa là  $Q$
  - Hành trình của mỗi chiếc xe: xuất phát từ kho hàng, chất lượng hàng hóa của các khách hàng mà nó sẽ đem giao trong hành trình của nó, lần lượt đem giao hàng cho các khách hàng, sau đó quay về kho hàng để kết thúc hành trình làm việc.

Yêu cầu: Hãy lên hành trình vận chuyển cho  $k$  chiếc xe của công ty (chú ý: có thể dùng ít hơn  $k$  xe), để giao hàng cho  $n$  khách hàng này, sao cho tổng chi phí vận chuyển là nhỏ nhất

NGUYỄN KHÁNH PHƯƠNG  
KHMT – SOICT - DHBK HN

### Bài toán định tuyến xe (Vehicle Routing Problem)



#### Input bài toán:

- Mỗi khách hàng  $i$  yêu cầu cần giao một lượng hàng kí hiệu là  $d_i$  (kg)
- Chi phí di từ điểm  $i$  đến điểm  $j$  là  $c_{ij}$
- Thời gian di từ điểm  $i$  đến điểm  $j$  là  $t_{ij}$
- Trọng tải mỗi xe:  $Q$  (ví dụ: 3000kg)
- Thời gian hoạt động của mỗi xe từ lúc rời khỏi depot đến lúc quay về không được vượt quá  $D$  (ví dụ: 4 tiếng).
- Mỗi khách chỉ được phục vụ 1 lần và bởi duy nhất 1 xe.

#### Output bài toán:

Lời giải có chi phí vận chuyển nhỏ nhất

Giả sử có lời giải như trên hình, khi đó tổng chi phí vận chuyển là:

- Xe 1:  $5 + 3 + 4 + 2 + 8 = 22\$$
- Xe 2:  $3 + 1 + 2 + 2 + 2 = 10\$$
- Xe 3:  $4 + 3 + 1 + 3 + 3 = 14\$$
- Xe 4:  $3 + 3 + 3 + 6 = 15\$$

Tổng 4 xe =  $22 + 10 + 14 + 15 = 71\$$

115

### 3.2. Một số ví dụ về bài toán tối ưu tổ hợp

1. Bài toán người du lịch (Traveling Salesman Problem)
2. Bài toán cái túi (Knapsack)
3. Bài toán đóng thùng (Bin Backing)
4. Bài toán định tuyến xe (Vehicle Routing Problem)
- 5. Bài toán lập lịch (Scheduling)**
6. Bài toán xếp thời khóa biểu (Timetabling)
7. Bài toán phân bổ tài nguyên (Resource allocations)

NGUYỄN KHÁNH PHƯƠNG  
KHMT – SOICT - DHBK HN

## Bài toán lập lịch (Scheduling problem)

- Tập công việc cần thực hiện
- Tập máy/người thực hiện công việc.
- Căn lê lịch máy/người thực hiện các công việc đã cho sao cho một số điều kiện ràng buộc được thỏa mãn, đồng thời tối ưu hóa chi phí/thời gian...

NGUYỄN KHÁNH PHƯƠNG  
KHMT – SOICT - DHBK HN

## Duyệt toàn bộ

- Một trong những phương pháp hiển nhiên nhất để giải bài toán tối ưu là hợp đặt ra **duyệt toàn bộ**: Trên cơ sở các thuật toán liệt kê toàn bộ ta tiến hành duyệt từng phương án của bài toán, đối với mỗi phương án ta đều tính giá trị hàm mục tiêu tại nó, sau đó so sánh giá trị hàm mục tiêu tại tất cả các phương án được liệt kê để tìm ra phương án tối ưu.

NGUYỄN KHÁNH PHƯƠNG  
KHMT – SOICT - DHBK HN

## Ví dụ: Giải bài toán cái túi

- Xét bài toán cái túi biến 0-1:

$$\max \{f(x) = \sum_{j=1}^n c_j x_j : x \in D\},$$

trong đó  $D = \{x = (x_1, x_2, \dots, x_n) \in B^n : \sum_{j=1}^n a_j x_j \leq b\}$

➢  $c_j, a_j, b$  là các số nguyên dương,  $j=1, 2, \dots, n$ .

➢ Cần có thuật toán liệt kê các phần tử của  $D$

NGUYỄN KHÁNH PHƯƠNG  
KHMT – SOICT - DHBK HN

## Thuật toán quay lui liệt kê các phương án chất đòn

- **Xây dựng  $S_k$ :**

- $S_1 = \{0, t_1\}$ , với  $t_1 = 1$  nếu  $b \geq a_1$ ;  $t_1 = 0$ , nếu trái lại
- Giả sử đã có phương án  $(x_1, \dots, x_{k-1})$ . Khi đó:

- Dung lượng còn lại là:

$$b_{k-1} = b - a_1 x_1 - \dots - a_{k-1} x_{k-1}$$

- Giá trị của các đồ vật chất vào túi là

$$f'_{k-1} = c_1 x_1 + \dots + c_{k-1} x_{k-1}$$

Do đó:  $S_k = \{0, t_k\}$ , với  $t_k = 1$  nếu  $b_{k-1} \geq a_k$ ;  $t_k = 0$ , nếu trái lại

- **Mô tả  $S_k$ ?**

for (y = 0; y++ ; y <= t<sub>k</sub>)

NGUYỄN KHÁNH PHƯƠNG  
KHMT – SOICT - DHBK HN

## Chương trình trên C++

```

int x[20], xopt[20], c[20], w[20];
int n, b, bk, fk, fopt;

void Nhaphd() {
    <Nhập vào n, c, w, b>;
}

void Inkq() {
    <Phương án tối ưu: xopt;
    Giá trị tối ưu: fopt>;
}

int main()
{
    Nhaphd();
    bk=b;
    fk= 0;
    fopt= 0;
    Try(1);
    Inkq();
}

```

NGUYỄN KHÁNH PHƯƠNG  
KHMT – SOICT - DHBK HN

```

void Try(int k)
{
    int j, t;
    if (bk >= a[k]) t=1 else t=0; //if: Trọng lượng còn lại của túi >= trọng lượng đồ vật k
    for (j= t; j>=0; j--)
    {
        x[k] = j; //j = 1 → x[k] = 1 tức là cho đồ vật k vào túi; j = 0 → x[k] = 0 tức là ko cho đồ vật k vào túi
        bk= bk-a[k]*x[k]; //bk: trọng lượng còn lại của túi
        fk= fk + c[k]*x[k]; //fk: giá trị hiện tại của túi
        if (k == n) //nếu tất cả n đồ vật đều đã được xét
        {
            if (fk>fopt) { //nếu giá trị hiện tại của túi > giá trị tốt nhất của túi hiện có
                xopt=x; fopt=fk; //Cập nhật giá trị tốt nhất
            }
        }
        else Try(k+1); //xét tiếp đồ vật thứ k+1
        bk= bk+a[k]*x[k];
        fk= fk - c[k]*x[k];
    }
}

```

NGUYỄN KHÁNH PHƯƠNG  
KHMT – SOICT - DHBK HN

## Bình luận

- Duyệt toàn bộ là khó có thể thực hiện được ngay cả trên những máy tính điện tử hiện đại nhất. Ví dụ để liệt kê hết

$$15! = 1\ 307\ 674\ 368\ 000$$

hoán vị trên máy tính điện tử với tốc độ tính toán 1 tỷ phép tính một giây, nếu để liệt kê một hoán vị cần phải làm 100 phép tính, thì ta cần một khoảng thời gian là 130767 giây > 36 tiếng đồng hồ!

$$20! \implies 7645 \text{ năm}$$

NGUYỄN KHÁNH PHƯƠNG  
KHMT – SOICT - DHBK HN

## Bình luận

- Vì vậy cần phải có những biện pháp nhằm hạn chế việc tìm kiếm thi mới có hy vọng giải được các bài toán tối ưu tố hợp thực tế. Tất nhiên để có thể đề ra những biện pháp như vậy cần phải nghiên cứu kỹ tính chất của bài toán tối ưu tố hợp cụ thể.
- Nhờ những nghiên cứu như vậy, trong một số trường hợp cụ thể ta có thể xây dựng những thuật toán hiệu quả để giải bài toán đặt ra.

NGUYỄN KHÁNH PHƯƠNG  
KHMT – SOICT - DHBK HN

## Bình luận

- Tuy nhiên phải nhấn mạnh rằng trong nhiều trường hợp (ví dụ trong các bài toán người du lịch, bài toán cái túi, bài toán đóng thùng) chúng ta chưa thể xây dựng được phương pháp hữu hiệu nào khác ngoài phương pháp duyệt toàn bộ.
- Khi đó, một vấn đề đặt ra là trong quá trình liệt kê lời giải ta cần tận dụng các thông tin đã tìm được để loại bỏ những phương án chắc chắn không phải là tối ưu.
- Trong mục tiếp theo chúng ta sẽ xét một sơ đồ tìm kiếm như vậy để giải các bài toán tối ưu tổ hợp mà trong tài liệu tham khảo được biết đến với tên gọi: thuật toán nhánh cận.

NGUYỄN KHÁNH PHƯƠNG  
KHMT – SOICT - DHBK HN

## 3. Thuật toán nhánh cận

- Bài toán tối ưu tổ hợp
- Một số ví dụ về bài toán tối ưu tổ hợp

### 3.3. Thuật toán nhánh cận (Branch and bound algorithm)

NGUYỄN KHÁNH PHƯƠNG  
KHMT – SOICT - DHBK HN

## 3.3. Thuật toán nhánh cận

### 3.3.1. Sơ đồ chung

#### 3.3.2. Ví dụ

- Bài toán người du lịch
- Bài toán cái túi
- Bài toán giao việc

NGUYỄN KHÁNH PHƯƠNG  
KHMT – SOICT - DHBK HN

### 3.3.1. Sơ đồ chung

- Thuật toán bao gồm hai thủ tục:
  - Phân nhánh (Branching Procedure)
  - Tính cận (Bounding Procedure)
- Phân nhánh:** Quá trình phân hoạch tập các phương án ra thành các tập con với kích thước càng ngày càng nhỏ cho đến khi thu được phân hoạch tập các phương án ra thành các tập con một phần tử
- Tính cận:** Cân đếm ra cách tính cận cho giá trị hàm mục tiêu của bài toán trên mỗi tập con A trong phân hoạch của tập các phương án.

NGUYỄN KHÁNH PHƯƠNG  
KHMT – SOICT - DHBK HN

### 3.3.1. Sơ đồ chung

- Ta sẽ mô tả tư tưởng của thuật toán trên mô hình bài toán tối ưu tổ hợp tổng quát sau

$$\min \{f(x) : x \in D\},$$

trong đó  $D$  là tập hữu hạn phần tử.

- Giả thiết rằng tập  $D$  được mô tả như sau

$$D = \{x = (x_1, x_2, \dots, x_n) \in A_1 \times A_2 \times \dots \times A_n : \\ x \text{ thoả mãn tính chất } P\},$$

với  $A_1, A_2, \dots, A_n$  là các tập hữu hạn, còn  $P$  là tính chất trên tích Đề các  $A_1 \times A_2 \times \dots \times A_n$ .

NGUYỄN KHÁNH PHƯƠNG  
KHMT – SOICT - ĐHBK HN

### Nhận xét

- Yêu cầu về mô tả của tập  $D$  là để có thể sử dụng thuật toán quay lui để liệt kê các phương án của bài toán.

- Bài toán

$$\max \{f(x) : x \in D\}$$

là tương đương với bài toán

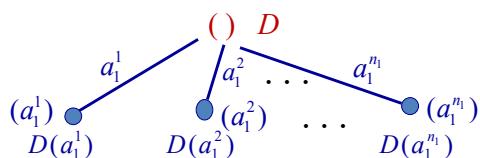
$$\min \{g(x) : x \in D\}, \text{ trong đó } g(x) = -f(x)$$

Do đó ta có thể hạn chế ở việc xét bài toán min.

NGUYỄN KHÁNH PHƯƠNG  
KHMT – SOICT - ĐHBK HN

### Phân nhánh

Quá trình phân nhánh được thực hiện nhờ thuật toán quay lui:



trong đó  $D(a_1^i) = \{x \in D : x_1 = a_1^i\}, i = 1, 2, \dots, n_1$

là tập các phương án có thể phát triển từ pabp ( $a_1^i$ )

Ta có phân hoạch:

$$D = D(a_1^1) \cup D(a_1^2) \cup \dots \cup D(a_1^{n_1})$$

NGUYỄN KHÁNH PHƯƠNG  
KHMT – SOICT - ĐHBK HN

### Phân nhánh

- Như vậy ta có thể đặt tương ứng mỗi phương án bộ phận  $(a_1, a_2, \dots, a_k)$  với một tập con các phương án của bài toán:

$$D(a_1, \dots, a_k) = \{x \in D : x_i = a_i, i = 1, \dots, k\}.$$

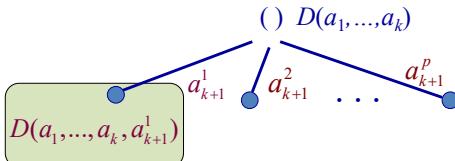
- Ở bước tổng quát của thuật toán quay lui ta sẽ làm việc với phương án bộ phận  $(a_1, a_2, \dots, a_k)$  và xét các cách tiếp tục phát triển phương án này.

- Điều đó tương đương với việc phân hoạch tập  $D$  ra thành các tập con nhỏ hơn.

NGUYỄN KHÁNH PHƯƠNG  
KHMT – SOICT - ĐHBK HN

## Phân nhánh

- Quá trình phân nhánh có thể diễn tả như sau:



➤ Ta có phân hoạch:

$$D(a_1, \dots, a_k) = \bigcup_{i=1}^p D(a_1, \dots, a_k, a_{k+1}^i)$$

NGUYỄN KHÁNH PHƯƠNG  
KHMT – SOICT - ĐHBKHN

## Tính cận

- Cần có hàm  $g$  xác định trên tập tất cả các phương án bộ phận của bài toán thỏa mãn bất đẳng thức sau:

Giá trị hàm mục tiêu của mọi lời giải có  $k$  thành phần đầu tiên là  $(a_1, a_2, \dots, a_k)$

$$g(a_1, \dots, a_k) \leq \min \{f(x): x \in D(a_1, \dots, a_k)\} \quad (*)$$

với mỗi phương án bộ phận cấp  $k$  ( $a_1, a_2, \dots, a_k$ ), và với mọi  $k = 1, 2, \dots$

- Bất đẳng thức  $(*)$  có nghĩa là giá trị của hàm  $g$  tại phương án bộ phận  $(a_1, a_2, \dots, a_k)$  là không vượt quá giá trị nhỏ nhất của hàm mục tiêu của bài toán trên tập con các phương án

$$D(a_1, \dots, a_k) = \{x \in D: x_i = a_i, i = 1, \dots, k\},$$

hay nói một cách khác,  $g(a_1, a_2, \dots, a_k)$  là **cận dưới** của giá trị hàm mục tiêu trên tập  $D(a_1, a_2, \dots, a_k)$ .

NGUYỄN KHÁNH PHƯƠNG  
KHMT – SOICT - ĐHBKHN

## Cắt nhánh nhờ sử dụng cận dưới

- Giả sử đã có hàm  $g$ . Ta xét cách sử dụng hàm này để giám sát khối lượng duyệt trong quá trình duyệt tất cả các phương án theo thuật toán quay lui.
- Trong quá trình liệt kê các phương án có thể đã thu được một số phương án của bài toán. Gọi  $x^*$  là phương án với giá trị hàm mục tiêu nhỏ nhất trong số các phương án đã tìm được, ký hiệu  $f^* = f(x^*)$
- Ta sẽ gọi
  - $x^*$  là phương án **tốt nhất hiện có** (phương án kỷ lục),
  - $f^*$  là **giá trị tốt nhất hiện có** (giá trị kỷ lục).

NGUYỄN KHÁNH PHƯƠNG  
KHMT – SOICT - ĐHBKHN

## Cắt nhánh nhờ sử dụng cận dưới

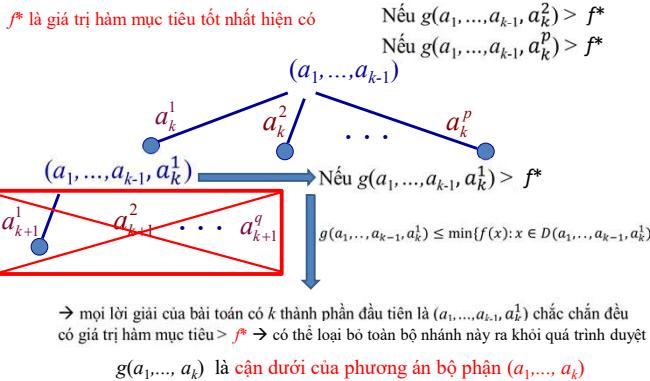
- Giả sử đã có  $f^*$  (**giá trị tốt nhất hiện có**), khi đó nếu  $g(a_1, a_2, \dots, a_k) > f^*$  thì từ bất đẳng thức  $(*)$  suy ra

$$f^* < g(a_1, \dots, a_k) \leq \min \{f(x): x \in D(a_1, \dots, a_k)\},$$

Vì thế tập  $D(a_1, \dots, a_k)$  chắc chắn không chứa phương án tối ưu và có thể loại bỏ khỏi quá trình duyệt.

NGUYỄN KHÁNH PHƯƠNG  
KHMT – SOICT - ĐHBKHN

## Cắt nhánh nhờ sử dụng cận dưới



## Thuật toán nhánh cận

```

void Try(int k)
{
    // Xây dựng  $x_k$  từ phương án bộ phận  $(x_1, x_2, \dots, x_{k-1})$ 
    for  $a_k \in A_k$ 
        if ( $a_k \in S_k$ )
        {
             $x_k = a_k$ ;
            if ( $k == n$ ) then < Cập nhật ký lục>;
            else if ( $g(x_1, \dots, x_k) \leq f^*$ ) Try(k+1);
        }
    }

void BranchAndBound()
{
     $f^* = +\infty$ ;
    // Nếu biết p/án  $x^*$  nào đó thì đặt  $f^* = f(x^*)$ 
    Try(1);
    if ( $f^* < +\infty$ )
        < $f^*$  là giá trị tối ưu,  $x^*$  là p/án tối ưu>;
    else < bài toán không có phương án>;
}

```

Chú ý rằng nếu thủ tục **Try** ta thay câu lệnh  
**if** ( $k == n$ ) then < Cập nhật ký lục>;  
**else if** ( $g(x_1, \dots, x_k) \leq f^*$ ) Try(k+1);  
bởi  
**if** ( $k == n$ ) then < Cập nhật ký lục>;  
**else** Try(k+1);

thì ta thu được thuật toán duyệt toàn bộ.

NGUYỄN KHÁNH PHƯƠNG  
KHMT – SOICT - DHBK HN

## Chú ý:

$$g(a_1, \dots, a_k) \leq \min\{f(x) : x \in D(a_1, \dots, a_k)\} \quad (*)$$

- Việc xây dựng hàm  $g$  phụ thuộc vào từng bài toán tối ưu tổ hợp cụ thể. Thông thường ta cố gắng xây dựng nó sao cho:
  - Việc tính giá trị của  $g$  phải đơn giản hơn việc giải bài toán tối ưu tổ hợp ở về phái của (\*).
  - Giá trị của  $g(a_1, \dots, a_k)$  phải sát với giá trị của về phái của (\*).
- Rất tiếc là hai yêu cầu này trong thực tế thường đối lập nhau.

NGUYỄN KHÁNH PHƯƠNG  
KHMT – SOICT - DHBK HN

## 3.3. Thuật toán nhánh cận

### 3.3.1. Sơ đồ chung

### 3.3.2. Ví dụ

#### 3.3.2.1. Bài toán người du lịch

#### 3.3.2.2. Bài toán cái túi

#### 3.3.2.3. Bài toán giao việc



Sir William Rowan Hamilton  
1805 - 1865

NGUYỄN KHÁNH PHƯƠNG  
KHMT – SOICT - DHBK HN

### Bài toán người du lịch (Traveling Salesman Problem – TSP)

- Một người du lịch muốn đi tham quan  $n$  thành phố  $1, 2, \dots, n$ .
- Hành trình là cách đi xuất phát từ thành phố 1 đi qua tất cả các thành phố còn lại, mỗi thành phố đúng một lần, rồi quay trở lại thành phố xuất phát 1.
- Biết  $c_{ij}$  là chi phí đi từ thành phố  $i$  đến thành phố  $j$  ( $i, j = 1, 2, \dots, n$ ),
- Tìm hành trình với tổng chi phí là nhỏ nhất.

NGUYỄN KHÁNH PHƯƠNG  
KHMT – SOICT - DHBK HN

### 3.3.2.1. Bài toán người du lịch

Bài toán yêu cầu:

- Tìm cực tiểu của hàm

$$f(1, x_2, \dots, x_n) = c[1, x_2] + c[x_2, x_3] + \dots + c[x_{n-1}, x_n] + c[x_n, 1]$$

với điều kiện

$(1, x_2, x_3, \dots, x_n)$  là hoán vị của các số  $1, 2, \dots, n$ .

NGUYỄN KHÁNH PHƯƠNG  
KHMT – SOICT - DHBK HN

### Hàm cận dưới

- Ký hiệu

$$c_{min} = \min \{ c[i, j], i, j = 1, 2, \dots, n, i \neq j \}$$

là chi phí đi lại nhỏ nhất giữa các thành phố.

- Cần đánh giá cận dưới cho phương án bộ phận  $(1, x_2, \dots, x_k)$  tương ứng với hành trình bộ phận đã đi qua  $k$  thành phố:

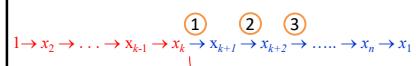
$$1 \rightarrow x_2 \rightarrow \dots \rightarrow x_{k-1} \rightarrow x_k.$$

NGUYỄN KHÁNH PHƯƠNG  
KHMT – SOICT - DHBK HN

### Hàm cận dưới

Phương án bộ phận  $(1, x_2, \dots, x_k)$ :

- Chi phí phải trả theo hành trình bộ phận này là  $\sigma = c[1, x_2] + c[x_2, x_3] + \dots + c[x_{k-1}, x_k]$ .
- Để phát triển thành hành trình đầy đủ:



ta còn phải đi qua  $n-k+1$  đoạn đường nữa, mỗi đoạn có chi phí không ít hơn  $c_{min}$ , nên cận dưới cho phương án bộ phận  $(1, x_2, \dots, x_k)$  có thể tính theo công thức:

$$g(1, x_2, \dots, x_k) = \sigma + (n-k+1) c_{min}.$$

NGUYỄN KHÁNH PHƯƠNG  
KHMT – SOICT - DHBK HN

## Cài đặt

```

void BranchAndBound()
{
    f* = +∞; cmin = min{cij : 1 ≤ i, j ≤ n}
    for (v = 1; v <= n; v++) visited[v] = FALSE;
    f=0; x1 = 1; visited[x1] = TRUE;
    Try(2);
    return f*;
}

void BranchAndBound ()
{
    f* = +∞;
    // Nếu biết p/án x* nào đó thì đặt f* = f(x*)
    Try(1);
    if (f* < +∞)
        < f* là giá trị tối ưu, x* là p/án tối ưu >
    else < bài toán không có phương án >
}

NGUYỄN KHÁNH PHƯƠNG
KHMT – SOICT - ĐHBKHN

```

## Cài đặt

```

void Try(int k)
{
    for (int v = 1; v <= n; v++) {
        if (visited[v] == FALSE) {
            xk = v; visited[v] = TRUE;
            f = f + c(xk-1, xk);
            if (k == n) //Cập nhật kí lục:
            { if (f + c(xn, x1) < f*) f* = f + c(xn, x1); }
            else {
                g = f + (n-k+1)*cmin; //tính cận
                if (g < f*) Try(k+1);
            }
            f = f - c(xk-1, xk);
            visited[v] = FALSE;
        } //end if
    } //end for
}

void Try(int k)
{
    //Xây dựng xk từ phương án bộ phận (x1, x2, ..., xk-1)
    for sk ∈ Sk
    {
        xk = sk;
        if (k == n) then < Cập nhật kí lục >
        else if (g(x1, ..., xk) < f*) Try(k+1);
    }
}

```

## 3.3 Thuật toán nhánh cận

### 3.3.1. Sơ đồ chung

### 3.3.2. Ví dụ

#### 3.3.2.1. Bài toán người du lịch

#### **3.3.2.2. Bài toán cái túi**

#### 3.3.2.3. Bài toán giao việc

NGUYỄN KHÁNH PHƯƠNG  
KHMT – SOICT - ĐHBKHN

### 3.3.2.2. Bài toán cái túi (Knapsack)

- Có  $n$  loại đồ vật.
- Đồ vật loại  $j$  có
  - trọng lượng  $a_j$  và
  - giá trị sử dụng là  $c_j$  ( $j = 1, 2, \dots, n$ ).
- Cần chất các đồ vật này vào một cái túi có trọng lượng là  $b$  sao cho tổng giá trị sử dụng của các đồ vật chất trong túi là lớn nhất.



NGUYỄN KHÁNH PHƯƠNG  
KHMT – SOICT - ĐHBKHN

### 3.3.2.2. Bài toán cái túi (Knap sack)

- Đưa vào biến số

$x_j$  – số lượng đồ vật loại  $j$  được chất vào túi,  $j=1,2, \dots, n$

- Mô hình toán học của bài toán có dạng sau: Tìm

$$f^* = \max \{ f(x) = \sum_{j=1}^n c_j x_j : \sum_{j=1}^n a_j x_j \leq b, x_j \in \{0,1\}, j = 1, 2, \dots, n \}$$

Bài toán cái túi biến nguyên

- Kí hiệu  $D$  là tập các phương án của bài toán:

$$D = \{x = (x_1, \dots, x_n) : \sum_{j=1}^n a_j x_j \leq b, x_j \in \{0,1\}, j = 1, 2, \dots, n\}$$

### Xây dựng hàm cận trên

- Giả thiết rằng các đồ vật được đánh số sao cho bất đẳng thức sau được thoả mãn:

$$c_1/a_1 \geq c_2/a_2 \geq \dots \geq c_n/a_n.$$

(có nghĩa là các đồ vật được xếp theo thứ tự không tăng của giá trị một đơn vị trọng lượng)

- Để xây dựng hàm tịnh cận trên, cùng với bài toán cái túi (KPC) ta xét bài toán cái túi biến liên tục (KPC) sau đây: Tìm

$$g^* = \max \{ f(x) = \sum_{j=1}^n c_j x_j : \sum_{j=1}^n a_j x_j \leq b, x_j \geq 0, j = 1, 2, \dots, n \}$$

NGUYỄN KHÁNH PHƯƠNG  
KHMT – SOICT- DHBK HN

### Xây dựng hàm cận trên

- Mệnh đề.** Phương án tối ưu của bài toán KPC là vector  $x^* = (x_1^*, x_2^*, \dots, x_n^*)$  với các thành phần được xác định bởi công thức:

$$x_1^* = b/a_1$$

$$x_2^* = x_3^* = \dots = x_n^* = 0$$

với giá trị tối ưu là  $g^* = c_1 x_1^* = c_1 b/a_1$

NGUYỄN KHÁNH PHƯƠNG  
KHMT – SOICT- DHBK HN

**Chứng minh.** Xét  $x = (x_1, \dots, x_n)$  là một phương án tuỳ ý của bài toán KPC. Khi đó

$$c_j \leq (c_1/a_1) a_j, j = 1, 2, \dots, n$$

do  $x_j \geq 0$ , ta suy ra

$$c_j x_j \leq (c_1/a_1) a_j x_j, j = 1, 2, \dots, n.$$

- Từ đó ta có

$$\begin{aligned} \sum_{j=1}^n c_j x_j &\leq \sum_{j=1}^n (c_1/a_1) a_j x_j \\ &= (c_1/a_1) \sum_{j=1}^n a_j x_j \\ &\leq (c_1/a_1)b = g^* \end{aligned}$$

Mệnh đề được chứng minh.

NGUYỄN KHÁNH PHƯƠNG  
KHMT – SOICT- DHBK HN

## Tính cận trên

- Bây giờ, giả sử ta có phương án bộ phận cấp  $k$ :  $(u_1, u_2, \dots, u_k)$ . Khi đó giá trị sử dụng của các đồ vật đang có trong túi là

$$\sigma_k = c_1 u_1 + c_2 u_2 + \dots + c_k u_k$$

và trọng lượng còn lại của cái túi là

$$b_k = b - (a_1 u_1 + a_2 u_2 + \dots + a_k u_k)$$

- Ta có:

$$\begin{aligned} & \max \{f(x) : x \in D, x_j = u_j, j = 1, 2, \dots, k\} \\ &= \max \left\{ \sigma_k + \sum_{j=k+1}^n c_j x_j : \sum_{j=k+1}^n a_j x_j \leq b_k, x_j \in \{0, 1\}, j = k+1, k+2, \dots, n \right\} \\ &\leq \sigma_k + \max \left\{ \sum_{j=k+1}^n c_j x_j : \sum_{j=k+1}^n a_j x_j \leq b_k, x_j \geq 0, j = k+1, k+2, \dots, n \right\} \\ &= \sigma_k + \boxed{c_{k+1} b_k / a_{k+1}}. \end{aligned}$$

Với trọng lượng còn lại  $b_k$ : lấy toàn bộ là đồ vật  $(k+1)$  là đồ vật có giá trị nhất để cho vào túi

- Vậy ta có thể tính cận trên cho phương án bộ phận  $(u_1, u_2, \dots, u_k)$  bởi công thức

$$g(u_1, u_2, \dots, u_k) = \sigma_k + c_{k+1} b_k / a_{k+1}.$$

NGUYỄN KHÁNH PHƯƠNG  
KHMT – SOICT - DHBKHN

## Tính cận trên

- Chú ý: Khi tiếp tục xây dựng thành phần thứ  $k+1$  của lời giải, các ứng cử viên cho  $x_{k+1}$  sẽ là  $0, 1, \dots, [b_k / a_{k+1}]$ .
- Do có kết quả của mệnh đề, khi chọn giá trị cho  $x_{k+1}$  ta sẽ duyệt các ứng cử viên theo thứ tự giảm dần

NGUYỄN KHÁNH PHƯƠNG  
KHMT – SOICT - DHBKHN

## Cài đặt

```
void BranchAndBound()
{
    f* = +∞;
    Sắp xếp các đồ vật theo thứ tự  $c_1/a_1 \geq c_2/a_2 \geq \dots \geq c_n/a_n$ 
    bk=b; //Trọng lượng còn lại của túi
    fk=0; //Giá trị sử dụng của túi
    Try(1);
    return f*;
}

void BranchAndBound()
{
    f* = +∞;
    // Nếu biết p/án  $x^*$  nào đó thì đặt  $f^* = f(x^*)$ 
    Try(1);
    if (f* < +∞)
        < $x^*$  là giá trị tối ưu,  $x^*$  là p/án tối ưu>;
    else < bài toán không có phương án >;
}
```

NGUYỄN KHÁNH PHƯƠNG  
KHMT – SOICT - DHBKHN

void Try(int k)

```
{
    int j, t;
    if (bk >= a[k]) t=1 else t=0; //if: Trọng lượng còn lại của túi >= trọng lượng đồ vật k
    for (j=t; j>=0;j--)
    {
        x[k] = j; //j=1 → x[k] = 1 tức là cho đồ vật k vào túi; j=0 → x[k] = 0 tức là ko cho đồ vật k vào túi
        bk= bk-a[k]*x[k]; //bk: trọng lượng còn lại của túi
        fk= fk + c[k]*x[k]; //fk: giá trị hiện tại của túi
        if (k == n) //nếu tất cả n đồ vật đều đã được xét
        {
            if (fk>f*) { //nếu giá trị hiện tại của túi > giá trị tốt nhất của túi hiện có
                x*=x; f*=fk; //Cập nhật giá trị tốt nhất
            }
        }
        else Try(k+1); //xét tiếp đồ vật thứ k+1
        bk= bk+a[k]*x[k];
        fk= fk - c[k]*x[k];
    }
}
```

### 3.3 Thuật toán nhánh cận

#### 3.3.1. Sơ đồ chung

#### 3.3.2. Ví dụ

3.3.2.1. Bài toán người du lịch

3.3.2.2. Bài toán cái túi

#### 3.3.2.3. Bài toán giao việc (Assignment problem)

NGUYỄN KHÁNH PHƯƠNG  
KHMT – SOICT - DHBK HN

### 3.3.2.3. Bài toán giao việc (Assignment Problem)

- Phát biểu bài toán: có  $n$  công việc và  $n$  công nhân. Biết chi phí phải trả cho công nhân  $i$  làm việc  $j$  là  $c_{ij}$ . Cần giao  $n$  việc này cho  $n$  công nhân sao cho mỗi công nhân chỉ làm 1 việc và mỗi việc chỉ làm bởi 1 công nhân. Hãy gán mỗi công việc cho mỗi công nhân sao cho tổng chi phí phải trả là ít nhất.

Ví dụ:  $n = 4$ . Lời giải tối ưu của bài toán là (A:job2, B: job1, C: job3, D:job4), và tổng chi phí =  $2 + 6 + 1 + 4 = 13$

	Job 1	Job 2	Job 3	Job 4
A	9	2	7	8
B	6	4	3	7
C	5	8	1	8
D	7	6	9	4

NGUYỄN KHÁNH PHƯƠNG  
KHMT – SOICT - DHBK HN

### 3.3.2.3. Bài toán giao việc (Assignment Problem)

- Duyệt toàn bộ:  $n!$
  - Nhánh cận:
    - Tính cận dưới (lower bound):
      - Cách 1: Mỗi công nhân được gán việc chưa có ai làm có chi phí nhỏ nhất (mỗi dòng tìm phần tử chưa gán cho ai có giá trị nhỏ nhất)
- Ví dụ: (A: job2, B: job 3, C: job1, D: job4): là 1 lời giải của bài toán
- Cách 2: Mỗi công việc được gán cho công nhân có chi phí nhỏ nhất (Mỗi cột tìm phần tử chưa gán cho ai có giá trị nhỏ nhất)
- Ví dụ: (Job1: C, Job2: A, Job3: B, Job4: D): là 1 lời giải của bài toán

	Job 1	Job 2	Job 3	Job 4
A	9	2	7	8
B	6	4	3	7
C	5	8	1	8
D	7	6	9	4

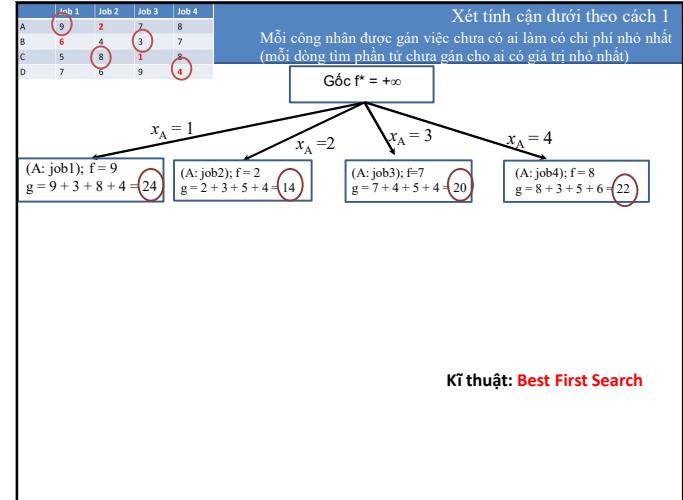
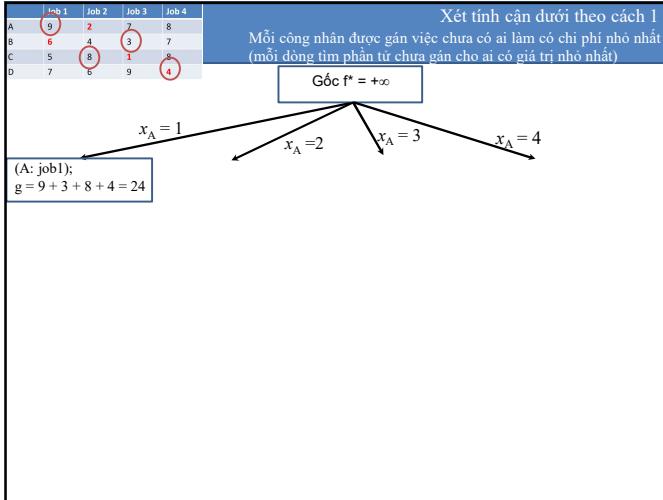
NGUYỄN KHÁNH PHƯƠNG  
KHMT – SOICT - DHBK HN

### 3.3.2.3. Bài toán giao việc (Assignment Problem)

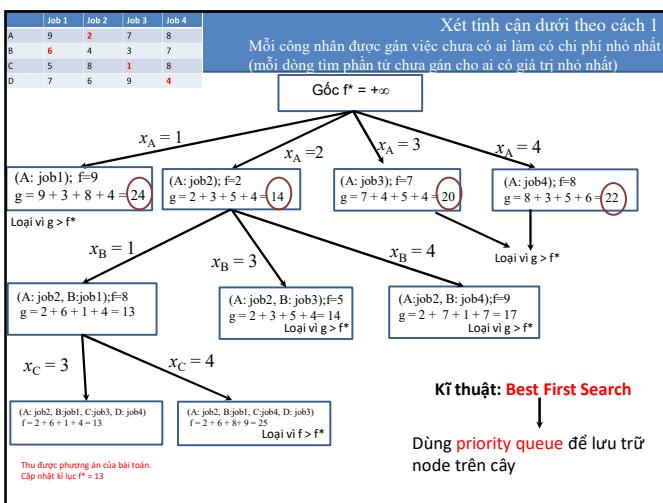
- Duyệt toàn bộ:  $n!$
  - Nhánh cận:
    - Tính cận dưới (lower bound):
      - Cách 3: Mỗi công nhân được gán việc có chi phí nhỏ nhất (mỗi dòng tìm phần tử có giá trị nhỏ nhất)
    - Ví dụ: (A: job2, B: job 3, C: job3, D: job4): không phải lời giải vì A và C cùng làm job3
    - Cách 4: Mỗi công việc được gán cho công nhân có chi phí nhỏ nhất (Mỗi cột tìm phần tử có giá trị nhỏ nhất)
- Ví dụ: (Job1: C, Job2: A, Job3: C, Job4: D): không phải lời giải vì C làm 2 job, B ko làm gì

	Job 1	Job 2	Job 3	Job 4
A	9	2	7	8
B	6	4	3	7
C	5	8	1	8
D	7	6	9	4

NGUYỄN KHÁNH PHƯƠNG  
KHMT – SOICT - DHBK HN



Kỹ thuật: Best First Search



### 3.3.2.3. Bài toán giao việc (Assignment Problem)

```
#include <bits/stdc++.h>
using namespace std;
#define N 4

// state space tree node
struct Node
{
    // stores parent node of current node
    // helps in tracing path when answer is found
    Node* parent;
    // Thông tin tại nut hien tai:
    int f; // Chi phi tinh den nut hien tai
    int lowerbound; // Can duoi cho nut hien tai
    int workerID; // workerID gan tai nut hien tai
    int jobID; // jobID gan tai nut hien tai
    bool assigned[N]; /*tai nut hien tai neu job i chua duoc gan
    cho worker nao thi assigned[i] = true; nguoc lai = false*/
};

// Thong tin tai nut hien tai:
int f; // Chi phi tinh den nut hien tai
int lowerbound; // Can duoi cho nut hien tai
int workerID; // workerID gan tai nut hien tai
int jobID; // jobID gan tai nut hien tai
bool assigned[N]; /*tai nut hien tai neu job i chua duoc gan
    cho worker nao thi assigned[i] = true; nguoc lai = false*/
};
```

### 3.3.2.3. Bài toán giao việc (Assignment Problem)

```
/* Tao nut moi tren cay: gan cong nhan worker cho cong viec job
Node* newNode(int worker, int job, bool assigned[], Node* parent)
{
    Node* node = new Node;
    for (int j = 0; j < N; j++) node->assigned[j] = assigned[j];
    node->assigned[job] = true;

    node->parent = parent;
    node->workerID = worker;
    node->jobID = job;

    return node;
}
```

165

### 3.3.2.3. Bài toán giao việc (Assignment Problem)

```
/*Tính Lowerbound cho nut hien tai (sau khi da gan cong nhan worker lam cv job):
of node after worker x is assigned to job y. */
int calculateLowerbound(int costMatrix[N][N], int worker,
                        int job, bool assigned[])
{
    //Mang assigned danh dau nhung cong viec nao chua duoc thuc hien o nut hien tai:
    int lowerbound = 0;
    bool available[N] = {true}; // to store unavailable jobs
    // start from next worker
    for (int i = worker + 1; i < N; i++)
    {
        //Voi moi cong nhan i: gan cho no cong viec chua duoc ai thuc hien ma co chi phi nho nhat
        int min = INT_MAX, minIndex = -1;
        // do for each job
        for (int j = 0; j < N; j++)
        {
            // If job is unassigned
            if (!assigned[j] && available[j] && costMatrix[i][j] < min)
            {
                minIndex = j; // store job number
                min = costMatrix[i][j]; // store cost
            }
        }
        lowerbound += min; // add cost of next worker
        available[minIndex] = false; // job becomes unavailable
    }
    return lowerbound;
}
```

166

### 3.3.2.3. Bài toán giao việc (Assignment Problem)

```
/* Ham so sanh su dung tren priority queue:
nut nao co lowerbound nho hon thi duoc uu tien hon */
struct comp
{
    bool operator()(const Node* lhs,
                     const Node* rhs) const
    {
        return lhs->lowerbound > rhs->lowerbound;
    }
};

// In loi giai
void printSolution(Node *min)
{
    if(min->parent==NULL) return;

    printSolution(min->parent);
    cout << "Gan cong nhan " << char(min->workerID + 'A')
        << " thuc hien cong viec " << min->jobID << endl;
}
```

167

### 3.3.2.3. Bài toán giao việc (Assignment Problem)

```
// Finds minimum cost using Branch and Bound.
int BranchAndBound(int costMatrix[N][N])
{
    /*Tao priority queue de luu tru cac nut tren cay
    (Create a priority queue to store live nodes of search tree)*/
    priority_queue<Node*, std::vector<Node*>, comp> pq;

    // initialize heap to dummy node with cost 0
    bool assigned[N] = {false};
    Node* root = new Node(-1, -1, assigned, NULL);
    root->f = root->lowerbound = 0;
    root->workerID = -1;

    // Add dummy node to list of live nodes;
    pq.push(root);

    // Finds a Live node with least cost,
    // add its childrens to list of live nodes and
    // finally deletes it from the list.
    while (!pq.empty())
    {
        //Lay khỏi queue nut co Lowerbound min
        Node* min = pq.top();
        //Xoa nut do khỏi queue:
        pq.pop();

        // i stores next worker
        int i = min->workerID + 1;
```

168

### 3.3.2.3. Bài toán giao việc (Assignment Problem)

```
// Neu tat ca cac cong nhan deu da duoc gan viec thi in ket qua:
if (i == N)
{
    printSolution(min);
    return min->lowerbound;
}

// do for each job
for (int j = 0; j < N; j++)
{
    // If unassigned
    if (!min->assigned[j])
    {
        // create a new tree node
        Node* child = newNode(i, j, min->assigned, min);

        // cost for ancestors nodes including current node
        child->f = min->f + costMatrix[i][j];

        // calculate its lower bound
        child->lowerbound = child->f +
            calculateLowerbound(costMatrix, i, j, child->assigned);

        // Add child to list of live nodes;
        pq.push(child);
    }
}
}
```

169

### 3.3.2.3. Bài toán giao việc (Assignment Problem)

```
int main()
{
    // x-coordinate represents a worker
    // y-coordinate represents a Job
    int costMatrix[N][N] =
    {
        {9, 2, 7, 8},
        {6, 4, 3, 7},
        {5, 8, 1, 6},
        {7, 6, 9, 4}
    };
    cout << "\n Chi phi tot uu = " << BranchAndBound(costMatrix);
    return 0;
}
```

170

### Training 3 - EXHSEARCH – 20192: TAXI

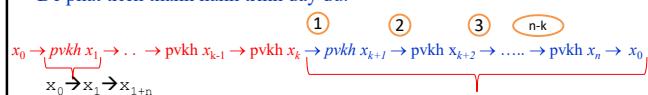
Có  $n$  hành khách  $1, 2, \dots, n$  và một chiếc taxi. Hành khách  $i$  muốn di chuyển từ điểm  $i$  đến điểm  $i + n$  ( $i = 1, 2, \dots, n$ ). Cho ma trận khoảng cách  $c_{2n+1} \times c_{2n+1}$  với  $c(i, j)$  là khoảng cách từ điểm  $i$  đến điểm  $j$  ( $i, j = 0, 1, \dots, 2n$ ). Hãy xác định độ dài đường đi ngắn nhất mà taxi đang đỗ ở điểm 0 sẽ phải đi để có thể phục vụ cả  $n$  hành khách này, rồi cuối cùng quay trở lại điểm xuất phát 0, sao cho tại bất kể thời điểm nào luôn chỉ có  $\leq 1$  hành khách trên taxi và không có điểm nào bị đón qua nhiều hơn 1 lần (ngoại trừ điểm 0 là điểm xuất phát ban đầu của taxi)

- **Input**
  - Dòng 1 chứa  $n$  ( $1 \leq n \leq 11$ )
  - Dòng  $i+1$  ( $i = 1, \dots, 2n, 2n+1$ ) chứa dòng thứ  $i$  của ma trận  $c_{2n+1 \times 2n+1}$
- **Output**
  - Độ dài đường đi ngắn nhất của taxi
  - Lời giải được biểu diễn bởi một mảng  $n+2$  số:  $0, x[1], x[2], \dots, x[n], 0$  trong đó  $x[i] = 1, \dots, n$
  - Mảng bool `visited[0...n]`: `visited[i] = true` nếu đã phục vụ khách hàng  $i$

### Hàm cận dưới

Giả sử đang có hành trình bộ phận là  $(0, x_1, x_2, \dots, x_k)$

- Chi phí phải trả theo hành trình bộ phận này là  $f = c[0, x_1] + c[x_1, x_{1+n}] + c[x_{1+n}, x_2] + \dots + c[x_{k-1+n}, x_k] + c[x_k, x_{k+n}]$
- Để phát triển thành hành trình đầy đủ:



ta còn phải phục vụ  $(n-k)$  khách hàng, tức là phải qua  $2*(n-k)+1$  đoạn đường nữa, mỗi đoạn có chi phí không ít hơn  $c_{min}$ , nên cận dưới cho phương án bộ phận  $(0, x_1, \dots, x_k)$  có thể tính theo công thức:

$$g(0, x_1, \dots, x_k) = f + [2*(n-k)+1] c_{min}$$

## TAXI: thuật toán nhánh cận

```

void BranchAndBound()
{
    c_min = INFINITY;
    cin >> n;
    int N = 2*n;
    for (int i = 0; i <= N; i++)
    {
        for (int j = 0; j <= N; j++)
        {
            cin >> c[i][j];
            if (i != j)
                if (c_min < c[i][j]) c_min = c[i][j];
        }
    }
    for (int i = 0; i <= n; i++) visited[i] = 0;

    fopt = INFINITY;
    f=0;
    x[0] = 0; visited[0] = 1;
    Try(1);
    cout << fopt;
}

void BranchAndBound ()
{
    f* = +oo;
    // Nếu biết p/án x* nào đó thi đặt f* = f(x*)
    Try(1);
    if (f* < +oo)
        <f* là giá trị tối ưu, x* là p/án tối ưu>
    else < bài toán không có phương án >
}

```

## TAXI: thuật toán nhánh cận

```

void Try(int k)
{
    for(int v = 1; v <= n; v++)
    {
        if(!visited[v]) //chưa phục vụ khách hàng v
        {
            x[k] = v; visited[v] = 1;
            f = f + c[x[k - 1]+n][v] + c[v][v + n];
            if(k == n) //Cap nhât ki luc
            {
                int temp = f + c[v+n][0];
                if (temp < fopt) fopt = temp;
            }
            else {
                g = f + (2 * (n - k) + 1) * c_min; //Tính can duoi
                if (g < fopt) Try(k+1);
            }
            visited[v] = 0;
            f = f - c[x[k - 1]+n][v] - c[v][v + n];
        }
    } //end for
}

void Try(int k)
{
    //Xây dựng x_k từ phương án bộ phận (x_1, x_2, ..., x_{k-1})
    for a_k in A_k
    {
        x_k = a_k;
        if (k == n) then < Cập nhật ki luc>
        else if (g(x_1, ..., x_k) ≤ f*) Try(k+1);
    }
}

```

## CBUS

Có  $n$  hành khách  $1, 2, \dots, n$  và một chiếc bus. Hành khách  $i$  muốn di chuyển từ điểm  $i$  đến điểm  $i+n$  ( $i = 1, 2, \dots, n$ ). Cho ma trận khoảng cách  $c_{2n+1} \times c_{2n+1}$  với  $c(i, j)$  là khoảng cách từ điểm  $i$  đến điểm  $j$  ( $i, j = 0, 1, \dots, 2n$ ). Hãy xác định độ dài đường đi ngắn nhất mà xe bus đang dỗ ở điểm 0 sẽ phải đi để có thể phục vụ cả  $n$  hành khách này, rồi cuối cùng quay trở lại điểm xuất phát 0, sao cho tại bất kỳ thời điểm nào luôn chỉ có  $\leq k$  hành khách trên bus và không có điểm nào bị đi qua nhiều hơn 1 lần (ngoại trừ điểm 0 là điểm xuất phát ban đầu của bus)

- Input**
  - Dòng 1 chứa  $n$  và  $k$  ( $1 \leq n \leq 11, 1 \leq k \leq 10$ )
  - Dòng  $i+1$  ( $i = 1, \dots, 2n, 2n+1$ ) chứa dòng thứ  $i$  của ma trận  $c_{2n+1 \times 2n+1}$
- Output**
  - Độ dài đường đi ngắn nhất của taxi

Lời giải được biểu diễn bởi một mảng  $2n+2$  số:  $0, x[1], x[2], \dots, x[2n], 0$  trong đó  $x[i] = 1, \dots, 2n$

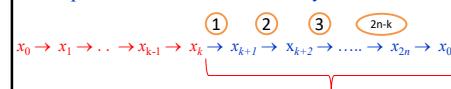
Mảng bool visited[0...2n]: visited[i] = true nếu xe đã đi qua điểm  $i$

## Hàm cận dưới

Giả sử đang có hành trình bộ phận là  $(0, x_1, x_2, \dots, x_k)$

- Chi phí phải trả theo hành trình bộ phận này là  

$$f = c[0, x_1] + c[x_1, x_2] + c[x_2, x_3] + \dots + c[x_{k-1}, x_k]$$
- Để phát triển thành hành trình đầy đủ:



ta còn phải đi qua  $(2n-k)+1$  đoạn đường nữa, mỗi đoạn có chi phí không ít hơn  $c_{min}$ , nên cận dưới cho phương án bộ phận  $(0, x_1, \dots, x_k)$  có thể tính theo công thức:

$$g(0, x_1, \dots, x_k) = f + [(2n-k)+1] c_{min}.$$

## CBUS

```
Try (int k) //xác định bên thứ k của xe
{
    for (i = 1; i<=2*n; i++)
        if (xe có thè đèn bên i)
        {
            x[k] = i; visited[i] = 0;
            f = f + c[x_{k-1}][i];
            if ( k == 2*n) Ghinhankiluc( );
            else {
                g = f + (2*n+k-1)cmin;
                if (g < fopt) Try(k+1);
            }
            visited[i] = 1; f -= c[x_{k-1}][i];
        } //end if
}
```