

Các kỹ thuật thiết kế thuật toán (Algorithm Design Techniques)

Bài giảng môn Cấu trúc dữ liệu và giải thuật

Khoa Công nghệ thông tin

Trường Đại học Thủy Lợi

Nội dung

- Vét cạn và quay lui
- Chia để trị
- Thuật toán tham lam
- Quy hoạch động

Vét cạn và quay lui

(Exhaustive search & backtracking)

Tìm kiếm vét cạn

- Xem xét mọi tổ hợp có thể lập ra được từ một tập các lựa chọn hoặc giá trị.
- Thường cài đặt bằng đệ quy.
- Ví dụ:
 - Tạo ra tất cả các hoán vị của một tập giá trị.
 - Liệt kê tất cả các tên và mật khẩu có thể.
- Không gian tìm kiếm thường bao gồm nhiều quyết định, và mỗi quyết định lại có nhiều lựa chọn.
 - Ví dụ: Khi liệt kê tất cả các xâu có 5 chữ cái tiếng Anh, mỗi trong 5 chữ cái đó là một quyết định, và mỗi quyết định như vậy có 26 lựa chọn khác nhau (a, b, c, ..., z).

Tìm kiếm vết cạn: Mã giả

Tìm-kiếm-vết-cạn:

- Nếu không còn quyết định nào nữa, dừng thuật toán.
- Ngược lại, xử lý một quyết định cụ thể, sau đó gọi đệ quy để xử lý các quyết định còn lại.

Ví dụ 1: In số nhị phân

Viết hàm đệ quy `printBinary(n)` in tất cả các số nhị phân có n chữ số.

```
printBinary(2);
```

```
00
```

```
01
```

```
10
```

```
11
```

```
printBinary(3);
```

```
000
```

```
001
```

```
010
```

```
011
```

```
100
```

```
101
```

```
110
```

```
111
```

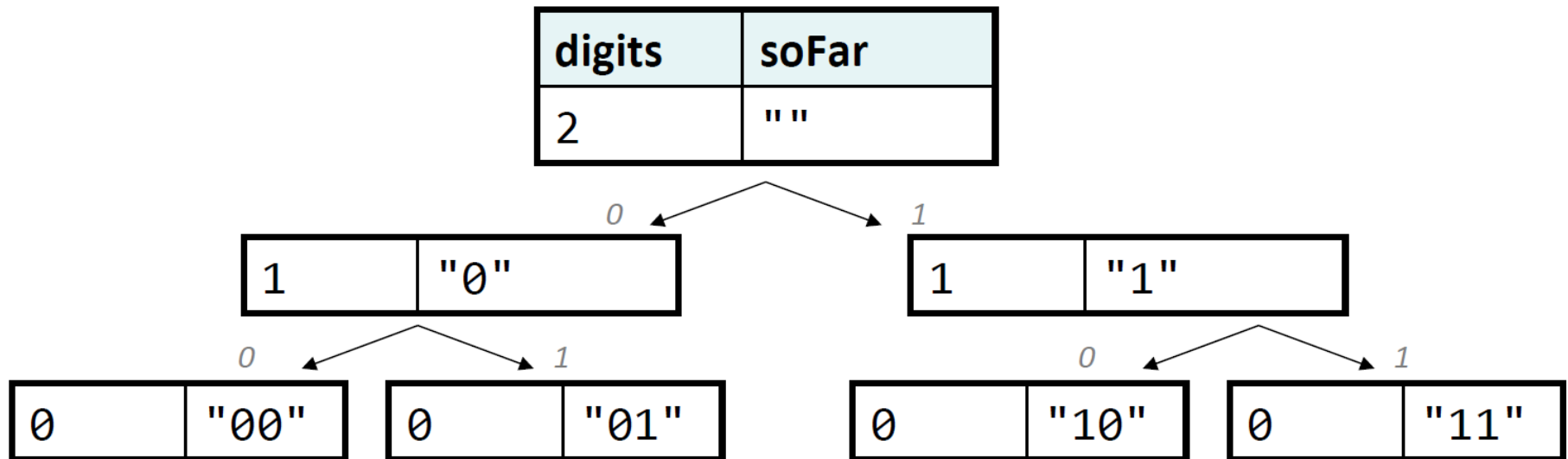
Ví dụ 1: In số nhị phân

```
void printBinary(int digits) {  
    printBinaryHelper(digits, "");  
}
```

```
void printBinaryHelper(int digits, string soFar) {  
    if (digits == 0)  
        cout << soFar << endl;  
    else {  
        printBinaryHelper(digits - 1, soFar + "0");  
        printBinaryHelper(digits - 1, soFar + "1");  
    }  
}
```

Cây quyết định

`printBinary(2)`



Ví dụ 2: In số thập phân

Viết hàm đệ quy `printDecimal(n)` in tất cả các số thập phân có n chữ số.

`printDecimal(2);`

00

01

02

..

98

99

`printDecimal(3);`

000

001

002

...


998

999

Ví dụ 2: In số thập phân

```
void printDecimal(int digits) {  
    printDecimalHelper(digits, "");  
}
```

```
void printDecimalHelper(int digits, string soFar) {  
    if (digits == 0)  
        cout << soFar << endl;  
    else  
        for (int i = 0; i < 10; i++)  
            printDecimalHelper(digits - 1, soFar +  
                                to_string(i));  
}
```



Hàm này chỉ có trong C++11 trở lên

Quay lui

- Tìm lời giải bằng cách thử tất cả các con đường đi có thể (như trong tìm kiếm vét cạn);
 - nhưng sẽ hủy bỏ ngay một con đường đi nếu phát hiện nó sẽ không thể dẫn đến lời giải, tức là sẽ **quay lui** về phía trước để tìm con đường đi khác.
- Xét bài toán gieo xúc xắc để minh họa kỹ thuật quay lui (slide tiếp theo).

Bài toán gieo xúc xắc

Viết hàm `diceSum(n, sum)` xuất ra tất cả các trường hợp gieo xúc xắc n lần và thu được tổng giá trị đúng bằng `sum`.

```
diceSum(2, 7);
```

```
{1, 6}  
{2, 5}  
{3, 4}  
{4, 3}  
{5, 2}  
{6, 1}
```

```
diceSum(3, 7);
```

```
{1, 1, 5}  
{1, 2, 4}  
{1, 3, 3}  
{1, 4, 2}  
{1, 5, 1}  
{2, 1, 4}  
{2, 2, 3}  
{2, 3, 2}  
{2, 4, 1}  
{3, 1, 3}  
{3, 2, 2}  
{3, 3, 1}  
{4, 1, 2}  
{4, 2, 1}  
{5, 1, 1}
```

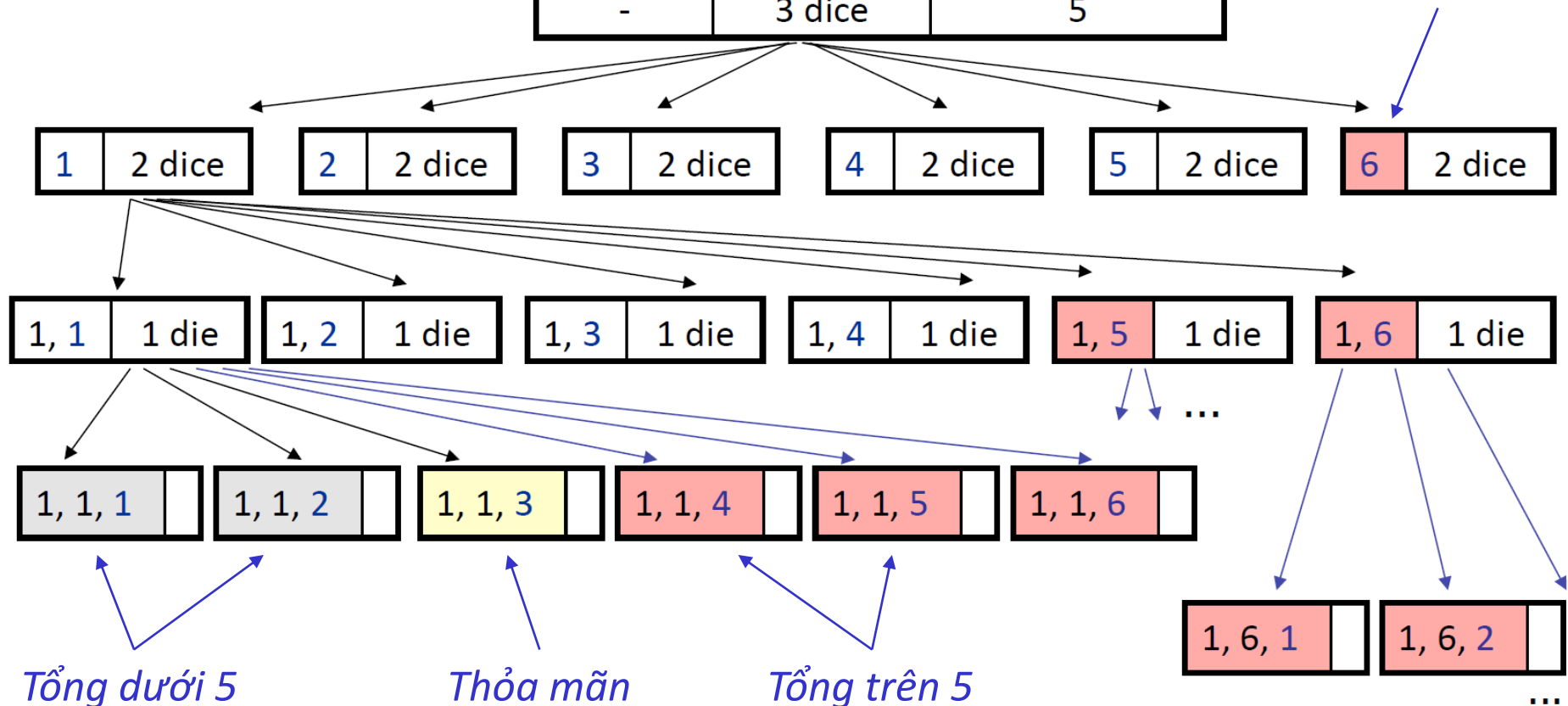


Cây quyết định

diceSum(3, 5);

chosen	available	desired sum
-	3 dice	5

*Cần cắt tỉa
(hủy bỏ) sớm*



Nhận xét

- Không cần thăm mọi nhánh trên cây quyết định.
 - Một số nhánh rõ ràng sẽ không dẫn đến thành công, ví dụ khi tổng của những lần gieo xúc xắc cho đến hiện tại đã quá cao hoặc quá thấp rồi.
 - Ta có thể cắt tỉa những nhánh như vậy.

```
void diceSum(int dice, int desiredSum) {
    vector<int> chosen;
    diceSumHelper(dice, 0, desiredSum, chosen);
}

void diceSumHelper(int dice, int sum, int desiredSum,
                   vector<int>& chosen) {
    if (dice == 0) {
        if (sum == desiredSum) { // tìm thấy lời giải
            for (int i = 0; i < chosen.size(); i++) // in lời giải
                cout << chosen[i] << " ";
            cout << endl;
        }
    } else if (sum + 1*dice > desiredSum || sum + 6*dice < desiredSum)
        return; // cắt tỉa nhánh sẽ thất bại
    else
        for (int i = 1; i <= 6; i++) {
            chosen.push_back(i); // thử nhánh này
            diceSumHelper(dice - 1, sum + i, desiredSum, chosen);
            chosen.pop_back();   // chuẩn bị thử nhánh tiếp theo
        }
}
```

Chia để trị

(Divide and Conquer)

Chia để trị

- Các bước:
 - Chia bài toán thành một số bài toán con.
 - Giải mỗi bài toán con theo kiểu đệ quy.
 - Kết hợp lời giải của các bài toán con thành lời giải tổng thể.
- Ví dụ, thuật toán sắp xếp trộn gồm các bước:
 - Chia dãy n phần tử thành 2 nửa, mỗi nửa có $n/2$ phần tử.
 - Sắp xếp mỗi nửa dùng thuật toán sắp xếp trộn.
 - Trộn 2 nửa đã sắp xếp thành dãy tổng thể sao cho dãy đó cũng được sắp xếp.

Đếm số nghịch đảo

- Một ứng dụng âm nhạc muốn giới thiệu cho bạn các bài hát mà bạn chưa nghe bằng cách so sánh sở thích nghe nhạc của bạn với những người khác.
 - Bạn xếp hạng n bài hát.
 - Ứng dụng tra cứu cơ sở dữ liệu để tìm những người có sở thích tương tự với bạn.
 - Ứng dụng giới thiệu cho bạn những bài hát bạn chưa nghe nhưng một người có sở thích tương tự với bạn đã nghe và thích bài hát đó.
- Độ đo tương tự: số lượng ***nghịch đảo*** (inversions) giữa hai danh sách xếp hạng bài hát (của bạn và của tôi).

Số nghịch đảo giữa hai xếp hạng

- Xếp hạng của tôi: 1, 2, ..., n
- Xếp hạng của bạn: a_1, a_2, \dots, a_n ($a_i \in \{1, 2, \dots, n\}$)
- Hai bài hát i và j bị đảo ngược nếu $i < j$ nhưng $a_i > a_j$

	A	B	C	D	E
me	1	2	3	4	5
you	1	3	4	2	5

2 inversions: 3-2, 4-2

Đếm số nghịch đảo: chia để trị

- Chia: Tách danh sách thành hai nửa A và B.
- Trị: Đếm số nghịch đảo trong mỗi danh sách theo kiểu đệ quy.
- Hợp: Đếm số nghịch đảo (a, b) với $a \in A$ và $b \in B$.
- Trả về tổng của ba lượng đếm được.

input

1	5	4	8	10	2	6	9	3	7
---	---	---	---	----	---	---	---	---	---

count inversions in left half A

1	5	4	8	10
---	---	---	---	----

5-4

count inversions in right half B

2	6	9	3	7
---	---	---	---	---

6-3 9-3 9-7

count inversions (a, b) with $a \in A$ and $b \in B$

1	5	4	8	10
---	---	---	---	----

2	6	9	3	7
---	---	---	---	---

4-2 4-3 5-2 5-3 8-2 8-3 8-6 8-7 10-2 10-3 10-6 10-7 10-9

output $1 + 3 + 13 = 17$

Đếm số nghịch đảo: cách kết hợp hai bài toán con

Đếm số nghịch đảo (a, b) với $a \in A$ và $b \in B$, giả sử A và B đã sắp xếp.

- Quét A và B từ trái sang phải.
- So sánh hai phần tử hiện hành a_i và b_j .
- Nếu $a_i < b_j$, thì a_i không đảo ngược với bất kì phần tử nào còn lại trong B .
- Nếu $a_i > b_j$, thì b_j đảo ngược với mọi phần tử còn lại trong A .
- Thêm phần tử nhỏ hơn vào danh sách C đã sắp xếp.

count inversions (a, b) with $a \in A$ and $b \in B$

3	7	10	a_i	18
---	---	----	-------	----



2	11	b_j	17	23
---	----	-------	----	----

5

2



merge to form sorted list C

2	3	7	10	11					
---	---	---	----	----	--	--	--	--	--



Đếm số nghịch đảo: thuật toán

- Đầu vào: Danh sách L
- Đầu ra: Số nghịch đảo trong L và danh sách đã sắp xếp L'

Sort-and-Count (L)

IF list L has one element

RETURN $(0, L)$.

DIVIDE the list into two halves A and B .

$(r_A, A) \leftarrow \text{Sort-and-Count}(A)$.

$(r_B, B) \leftarrow \text{Sort-and-Count}(B)$.

$(r_{AB}, L') \leftarrow \text{Merge-and-Count}(A, B)$.

RETURN $(r_A + r_B + r_{AB}, L')$.

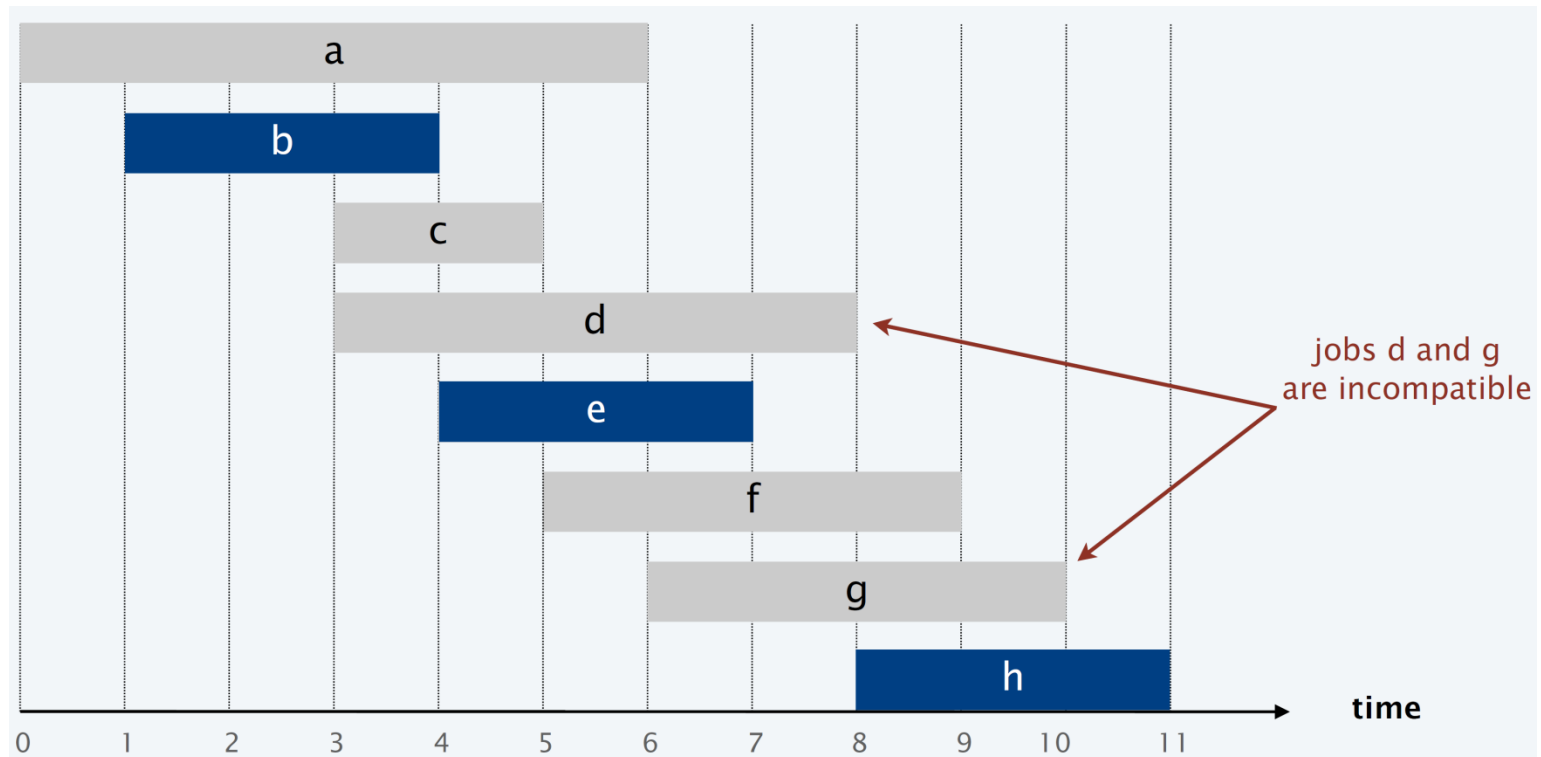
Thuật toán tham lam (Greedy Algorithms)

Thuật toán tham lam

- Thuật toán tham lam xây dựng lời giải trong các bước nhỏ, mỗi bước ra một quyết định nông cạn (tham lam) để tối ưu hóa một tiêu chuẩn cục bộ nào đó.
- Ví dụ, trong thuật toán Dijkstra tìm đường đi ngắn nhất trên đồ thị, mỗi bước chọn một đỉnh kề với đám mây (tập S) gần với đỉnh nguồn s nhất để thêm vào đám mây.
- Trong một số trường hợp, thuật toán tham lam tìm ra lời giải tối ưu.
- Trong các trường hợp khác, thuật toán tham lam chỉ tìm ra lời giải gần đúng.

Bài toán lập lịch khoảng

- Công việc j bắt đầu tại thời điểm s_j và kết thúc tại thời điểm f_j .
- Hai công việc **tương thích** với nhau nếu chúng không chồng lên nhau.
- Mục tiêu: Tìm ra tập con lớn nhất bao gồm các công việc tương thích lẫn nhau.



Lập lịch khoảng: các thuật toán tham lam (1)

Kiểu tham lam: Xét các công việc theo một trình tự tự nhiên nào đó. Chọn một công việc nếu nó tương thích với các công việc đã được chọn trước đó rồi.

- [Thời gian bắt đầu sớm nhất] Xét các công việc theo thứ tự tăng dần của s_j .
- [Thời gian kết thúc sớm nhất] Xét các công việc theo thứ tự tăng dần của f_j .
- [Khoảng ngắn nhất] Xét các công việc theo thứ tự tăng dần của $f_j - s_j$.
- [Ít đụng độ nhất] Đối với mỗi công việc j , đếm số công việc c_j đụng độ với nó. Lập lịch theo thứ tự tăng dần của c_j .

Lập lịch khoảng: các thuật toán tham lam (2)

Kiểu tham lam: Xét các công việc theo một trình tự tự nhiên nào đó. Chọn một công việc nếu nó tương thích với các công việc đã được chọn trước đó rồi.

counterexample for earliest start time



counterexample for shortest interval



counterexample for fewest conflicts



Lập lịch khoảng: thuật toán theo thời gian kết thúc sớm nhất

EARLIEST-FINISH-TIME-FIRST ($n, s_1, s_2, \dots, s_n, f_1, f_2, \dots, f_n$)

SORT jobs by finish time so that $f_1 \leq f_2 \leq \dots \leq f_n$

$A \leftarrow \phi$  set of jobs selected

FOR $j = 1$ **TO** n

IF job j is compatible with A

$A \leftarrow A \cup \{j\}$

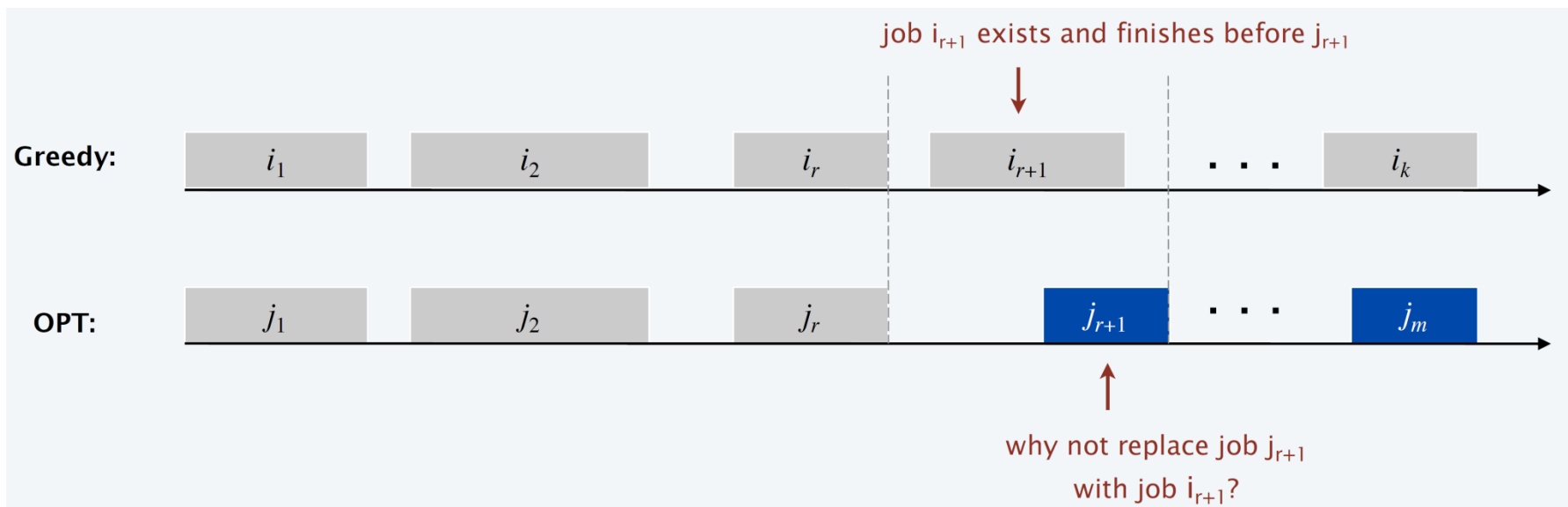
RETURN A

Lập lịch khoảng: phân tích thuật toán theo thời gian kết thúc sớm nhất (1)

Định lý: Thuật toán theo thời gian kết thúc sớm nhất trả về lời giải tối ưu.

Chứng minh (bằng phản chứng):

- Giả sử thuật toán không tối ưu.
- Gọi i_1, i_2, \dots, i_k là tập công việc được chọn bởi thuật toán.
- Gọi j_1, j_2, \dots, j_m là tập công việc trong lời giải tối ưu với $i_1 = j_1, i_2 = j_2, \dots, i_r = j_r$ với r là giá trị lớn nhất có thể.

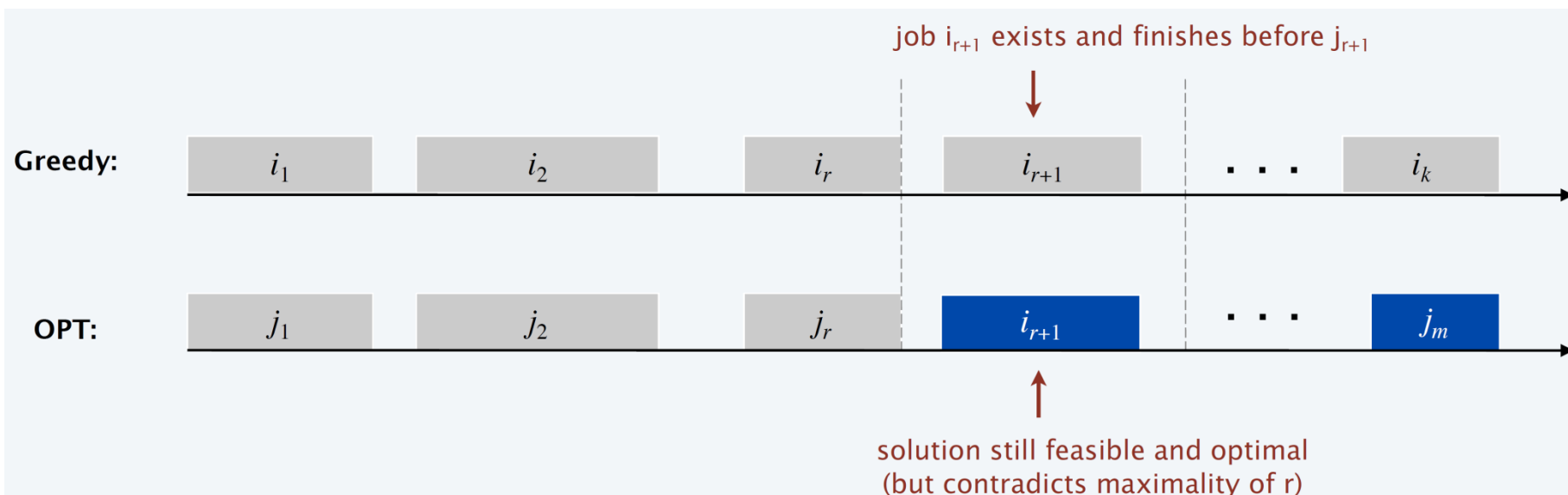


Lập lịch khoảng: phân tích thuật toán theo thời gian kết thúc sớm nhất (2)

Định lý: Thuật toán theo thời gian kết thúc sớm nhất trả về lời giải tối ưu.

Chứng minh (bằng phản chứng):

- Giả sử thuật toán không tối ưu.
- Gọi i_1, i_2, \dots, i_k là tập công việc được chọn bởi thuật toán.
- Gọi j_1, j_2, \dots, j_m là tập công việc trong lời giải tối ưu với $i_1 = j_1, i_2 = j_2, \dots, i_r = j_r$ với r là giá trị lớn nhất có thể.



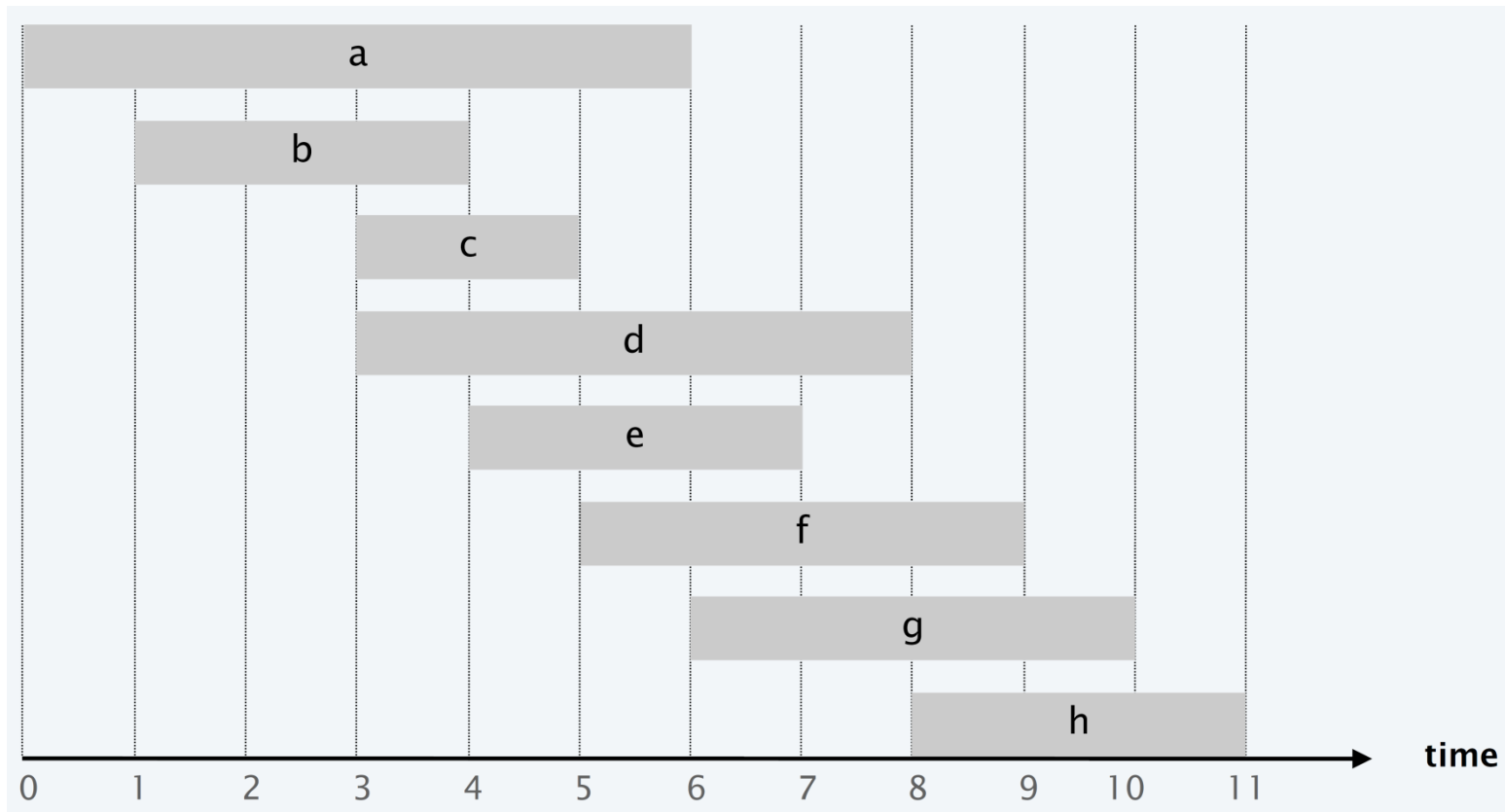
Quy hoạch động (Dynamic Programming)

Quy hoạch động

- Tham lam: Xây dựng lời giải lớn dần, tối ưu hóa (nông cạn) một tiêu chuẩn cục bộ nào đó.
- Chia để trị: Phân rã bài toán thành các bài toán con **độc lập**, giải mỗi bài toán con, và kết hợp lời giải của các bài toán con để lập thành lời giải cho bài toán tổng thể ban đầu.
- Quy hoạch động: Phân rã bài toán thành một chuỗi các bài toán con **chồng lên nhau**, và xây dựng lời giải cho các bài toán con ngày càng lớn dần.

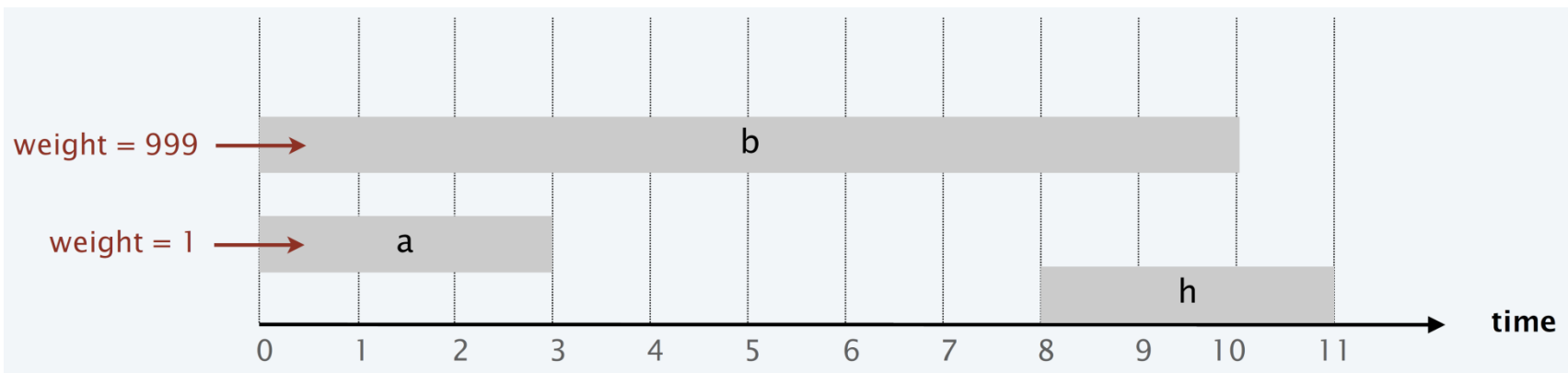
Bài toán lập lịch khoảng có trọng số

- Công việc j bắt đầu tại s_j , kết thúc tại f_j , và có trọng số v_j .
- Hai công việc tương thích với nhau nếu chúng không chồng lên nhau.
- Mục tiêu: Tìm ra tập con có trọng số lớn nhất bao gồm các công việc tương thích lẫn nhau.



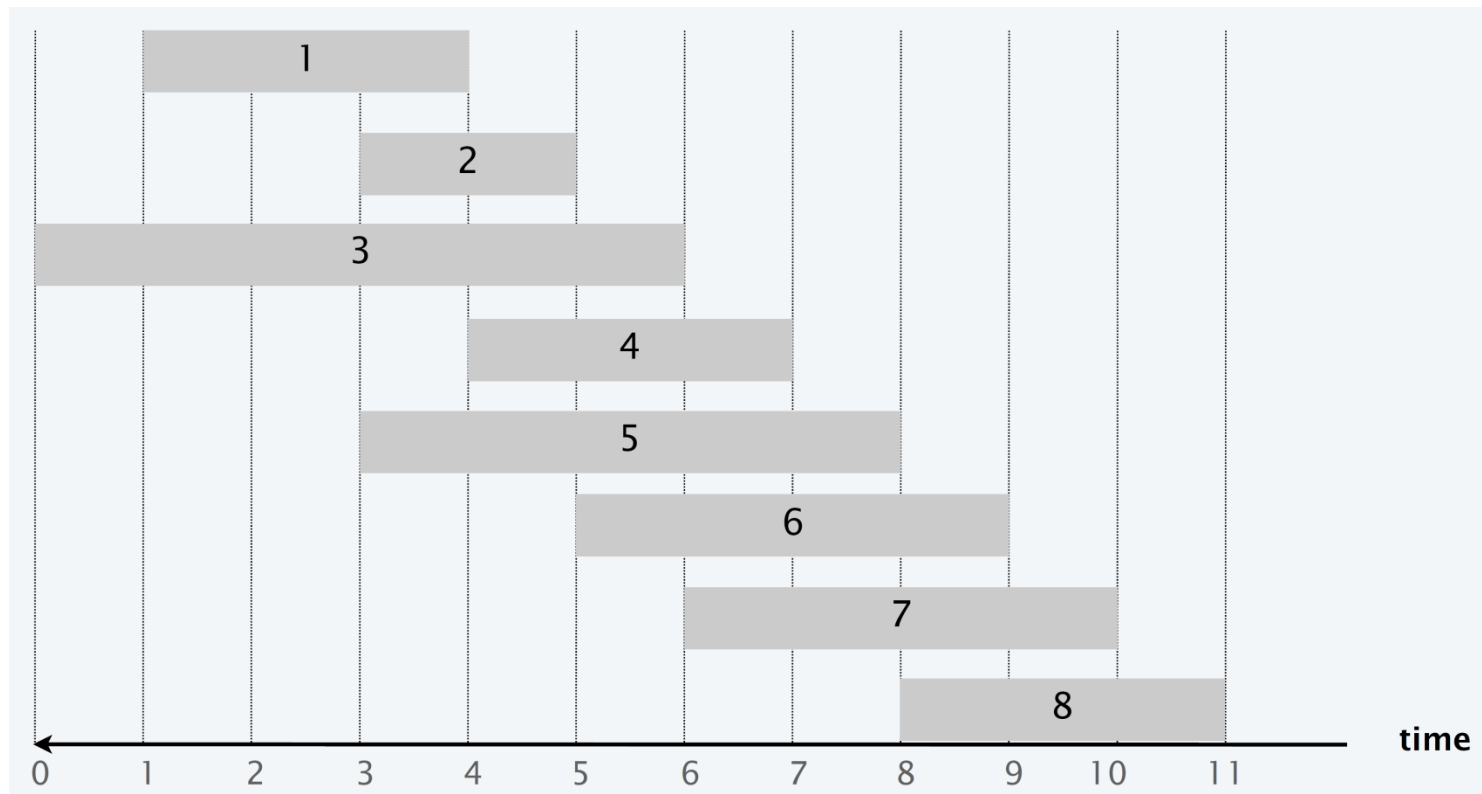
Thuật toán theo thời gian kết thúc sớm nhất

- Thuật toán tham lam:
 - Xét các công việc theo thứ tự tăng dần của thời gian kết thúc.
 - Thêm công việc vào tập con nếu nó tương thích với các công việc đã được chọn trước đó.
- Nhắc lại: Thuật toán tham lam đúng nếu tất cả các trọng số bằng 1.
- Nhận xét: Thuật toán tham lam thất bại với phiên bản có trọng số.



Lập lịch khoảng có trọng số

- Gắn nhãn các công việc theo thời gian kết thúc: $f_1 \leq f_2 \leq \dots \leq f_n$
- Gọi $p(j)$ là chỉ số i lớn nhất nhỏ hơn j sao cho công việc i tương thích với công việc j .
- Ví dụ: $p(8) = 5$, $p(7) = 3$, $p(2) = 0$



Quy hoạch động: hai lựa chọn

- Kí hiệu $OPT(j)$ là giá trị của lời giải tối ưu cho bài toán con bao gồm các công việc 1, 2, ..., j.
- Trường hợp 1: OPT chọn công việc j
 - Thu thập giá trị v_j .
 - Không thể dùng những việc không tương thích $\{p(j) + 1, p(j) + 2, \dots, j - 1\}$.
 - Phải bao gồm lời giải tối ưu cho bài toán con bao gồm các công việc còn lại 1, 2, ..., $p(j)$.
- Trường hợp 2: OPT không chọn công việc j
 - Phải bao gồm lời giải tối ưu cho bài toán con bao gồm các công việc còn lại 1, 2, ..., $j - 1$.

$$OPT(j) = \begin{cases} 0 & \text{nếu } j = 0 \\ \max \{v_j + OPT(p(j)), OPT(j - 1)\} & \text{ngược lại} \end{cases}$$

Lập lịch khoảng có trọng số: tìm giá trị tối ưu

- Lưu trữ lại kết quả của các bài toán con
- Tra cứu khi cần

Input: $n, s[1..n], f[1..n], v[1..n]$

Sort jobs by finish time so that $f[1] \leq f[2] \leq \dots \leq f[n]$.

Compute $p[1], p[2], \dots, p[n]$.

for $j = 1$ to n

$M[j] \leftarrow \text{empty}.$

$M[0] \leftarrow 0.$

M-Compute-Opt(j)

if $M[j]$ is empty

$M[j] \leftarrow \max(v[j] + \text{M-Compute-Opt}(p[j]), \text{M-Compute-Opt}(j - 1)).$

return $M[j].$

Lập lịch khoảng có trọng số: tìm lời giải

Find-Solution(j)

if $j = 0$

 return \emptyset .

else if ($v[j] + M[p[j]] > M[j-1]$)

 return $\{j\} \cup \text{Find-Solution}(p[j])$.

else

 return Find-Solution($j-1$).