Cấu trúc dữ liệu và giải thuật

Một số mô hình thuật toán kinh điển

Nguyễn Văn Tiến

Nội dung

1. Mô hình thuật toán sinh (Generative Algorithm)
2. Mô hình thuật toán đệ quy (Recursion Algorithm)
3. Mô hình thuật toán quay lui (Backtrack Algorithm)
4. Mô hình thuật toán tham lam (Greedy Algorithm)
5. Mô hình thuật toán chia và trị (Divide and Conquer Algorithm)
6. Mô hình thuật toán nhánh cận (Branch and Bound Algorithm)
7. Mô hình thuật toán quy hoạch động (Dynamic programming)

Mô hình thuật toán sinh (Generative algorithm)

Bài toán duyệt (Vét cạn)

Mục tiêu: xem xét tất cả các khả năng có thể.

Áp dụng: Thường đưa về dạng bài toán tổ hợp

Ý tưởng: Sinh kế tiếp hoặc Quay lui

Một thuật toán duyệt cần thỏa mãn hai điều kiện:

- Không được lặp lại bất kỳ khả năng nào.
- Không được bỏ sót bất kỳ cấu hình nào

Mô hình thuật toán sinh (Generative algorithm)

Giới thiệu thuật toán sinh

Mô hình thuật toán sinh được dùng để giải lớp các bài toán liệt kê, bài toán đếm, bài toán tối ưu, bài toán tồn tại thỏa mãn hai điều kiện:

- Điều kiện 1: Có thể xác định được một thứ tự trên tập các cấu hình cần liệt kê của bài toán. Biết cấu hình đầu tiên, biết cấu hình cuối cùng.
- Điều kiện 2: Từ một cấu hình chưa phải cuối cùng, ta xây dựng được thuật toán sinh ra cấu hình đứng ngay sau nó.

Mô hình thuật toán sinh (Generative algorithm)

Mô hình tổng quát

Begin

Bước1 (Khởi tạo):

<Thiết lập cấu hình đầu tiên>;

Bước 2 (Bước lặp):

while (<Lặp khi cấu hình chưa phải cuối cùng>) **do**

<Đưa ra cấu hình hiện tại>;

<Sinh ra cấu hình kế tiếp>;

endwhile;

End

Mô hình thuật toán sinh (Generative algorithm)

Sinh xâu nhị phân

STT	A=(a0, a1, a2, a3)	STT	A=(a0, a1, a2, a3)
0	0 0 0 0	8	1 0 0 0
1	0 0 0 1	9	1 0 0 1
2	0 0 1 0	10	1 0 1 0
3	0 0 1 1	11	1 0 1 1
4	0 1 0 0	12	1 1 0 0
5	0 1 0 1	13	1 1 0 1
6	0 1 1 0	14	1 1 1 0
7	0 1 1 1	15	1 1 1 1

Mô hình thuật toán sinh (Generative algorithm)

Sinh xâu nhị phân

```
void generate(int n) {
    int a[n];
    for(int i = 0; i < n; i++)
        a[i] = 0;
    bool has_next = true;
    while(has_next){
        printArray(a, n);
        int i = n - 1;
        while(i >= 0 && a[i] == 1){
            a[i]=0;
            i--;
        }
        if (i>=0) {
            a[i]=1;
        }
        else has_next = false;
    }
}
```

Mô hình thuật toán sinh (Generative algorithm)

Sinh tổ hợp (Liệt kê các tập con k phần tử của 1, 2, ... n)

Ví dụ k = 3, n = 5

STT	Tập con	STT	Tập con
1	1 2 3	6	1 4 5
2	1 2 4	7	2 3 4
3	1 2 5	8	2 3 5
4	1 3 4	9	2 4 5
5	1 3 5	10	3 4 5

Mô hình thuật toán sinh (Generative algorithm)

Sinh tổ hợp (Liệt kê các tập con k phần tử của 1, 2, ... n)

```
void generate(int k, int n) {
    int a[k];
    for(int i = 0; i < k; i++)
        a[i] = i + 1;
    bool has_next = true;
    while(has_next){
        printArray(a, k);
        int i = k - 1;
        while(i >= 0 && a[i] == n - k + i + 1){
            i--;
        }
        if (i>=0) {
            a[i]= a[i]+1;
            for(int j=i+1; j<=k; j++)
                a[j] = a[i] + j - i;
        }
        else has_next = false;
    }
}
```

Mô hình thuật toán sinh (Generative algorithm)

Sinh hoán vị

Ví dụ n = 3

STT	Hoán vị
1	1 2 3
2	1 3 2
3	2 1 3
4	2 3 1
5	3 1 2
6	3 2 1

Mô hình thuật toán sinh (Generative algorithm)

Sinh hoán vị

```
void generate(int n) {
    int a[n];
    for(int i = 0; i < n; i++)
        a[i] = i + 1;
    bool has_next = true;
    while(has_next){
        printArray(a, n);
        int i = n - 2;
        while(i >= 0 && a[i] > a[i + 1]){
            i--;
        }
        if (i>=0) {
            int j = n - 1;
            while(a[i]>a[j])
                j--;
            swap(a[i], a[j]);
            int r = i+1, s = n - 1;
            while (r<=s){
                swap(a[r], a[s]);
                r++; s--;
            }
        }
        else has_next = false;
    }
}
```

Mô hình thuật toán đệ quy (Recursion algorithm)

Giới thiệu thuật toán đệ quy

Một đối tượng được định nghĩa trực tiếp hoặc gián tiếp thông qua chính nó được gọi là phép định nghĩa bằng đệ quy.

Thuật toán giải bài toán P một cách trực tiếp hoặc gián tiếp thông qua bài toán P’ giống như P được gọi là thuật toán đệ qui giải bài toán P.

Một hàm được gọi là đệ quy nếu nó được gọi trực tiếp hoặc gián tiếp đến chính nó.

Mô hình thuật toán đệ quy (Recursion algorithm)

Giới thiệu thuật toán đệ quy

Tổng quát, một bài toán có thể giải được bằng đệ qui nếu nó thỏa mãn hai điều kiện:

- Phân tích được: Có thể giải được bài toán P bằng bài toán P′ giống như P. Bài toán P′ và chỉ khác P ở dữ liệu đầu vào. Việc giải bài toán P′ cũng được thực hiện theo cách phân tích giống như P.
- Điều kiện dừng: Dãy các bài toán P′ giống như P là hữu hạn và sẽ dừng tại một bài toán xác định nào đó.

Mô hình thuật toán đệ quy (Recursion algorithm)

Mô hình tổng quát

Thuật toán Recursion (P) {

1. Nếu P thỏa mãn điều kiện dừng:

<Giải P với điều kiện dừng>;

2. Nếu P không thỏa mãn điều kiện dừng:

<Giải P’ giống như P:Recursion(P’)>;

}

Mô hình thuật toán đệ quy (Recursion algorithm)

Ví dụ

Tìm tổng của n số tự nhiên đầu tiên bằng phương pháp đệ qui.

Lời giải: Gọi Sn là tổng của n số tự nhiên. Khi đó:

- Bước phân tích: dễ dàng phận thấy tổng n số tự nhiên $S_n = n + S_{n-1}$, với $n \geq 1$.
- Điều kiện dừng: $S_0 = 0$ nếu $n = 0$;

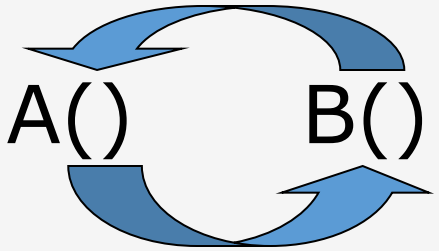
Mô hình thuật toán đệ quy (Recursion algorithm)

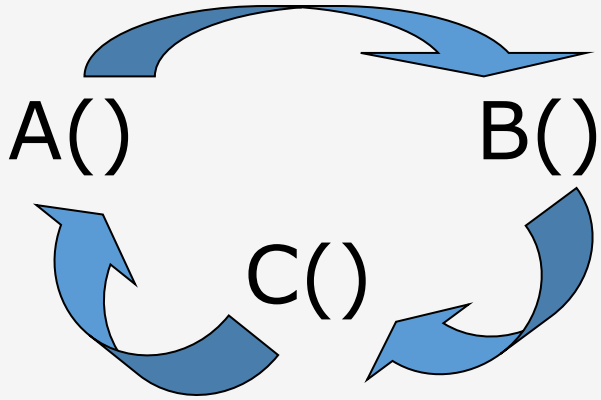
```
int sum(int n) {
    if (n == 0 ) {
        return(0);
    }
    return(n + sum(n - 1));
}
```

Mô hình thuật toán đệ quy (Recursion algorithm)

Các loại đệ quy

1. Đệ quy trực tiếp
2. Đệ quy gián tiếp





Mô hình thuật toán quay lui (Back-track algorithm)

Giới thiệu thuật toán quay lui

Quay lui là một kĩ thuật thiết kế giải thuật dựa trên đệ quy. Ý tưởng của quay lui là tìm lời giải từng bước, mỗi bước chọn một trong số các lựa chọn có thể và đệ quy.

Quay lui có thể áp dụng để giải quyết các bài toán liệt kê cấu hình

Mô hình thuật toán quay lui (Back-track algorithm)

Giới thiệu thuật toán quay lui

Giả sử ta cần xác định bộ X =(x_0, x_1,..,x_{n-1}) thỏa mãn một số ràng buộc nào đó. Ứng với mỗi thành phần x_i ta có n_i khả năng cần lựa chọn. Ứng với mỗi khả năng j thuộc n_i dành cho thành phần x_i ta cần thực hiện:

- Kiểm tra xem khả năng j có được chấp thuận cho thành phần x_i hay không? Nếu khả năng j được chấp thuận thì ta xác định thành phần x_i theo khả năng j.

Nếu i là thành phần cuối cùng (i=n-1) ta ghi nhận nghiệm của bài toán. Nếu i chưa phải cuối cùng ta xác định thành phần thứ i +1.

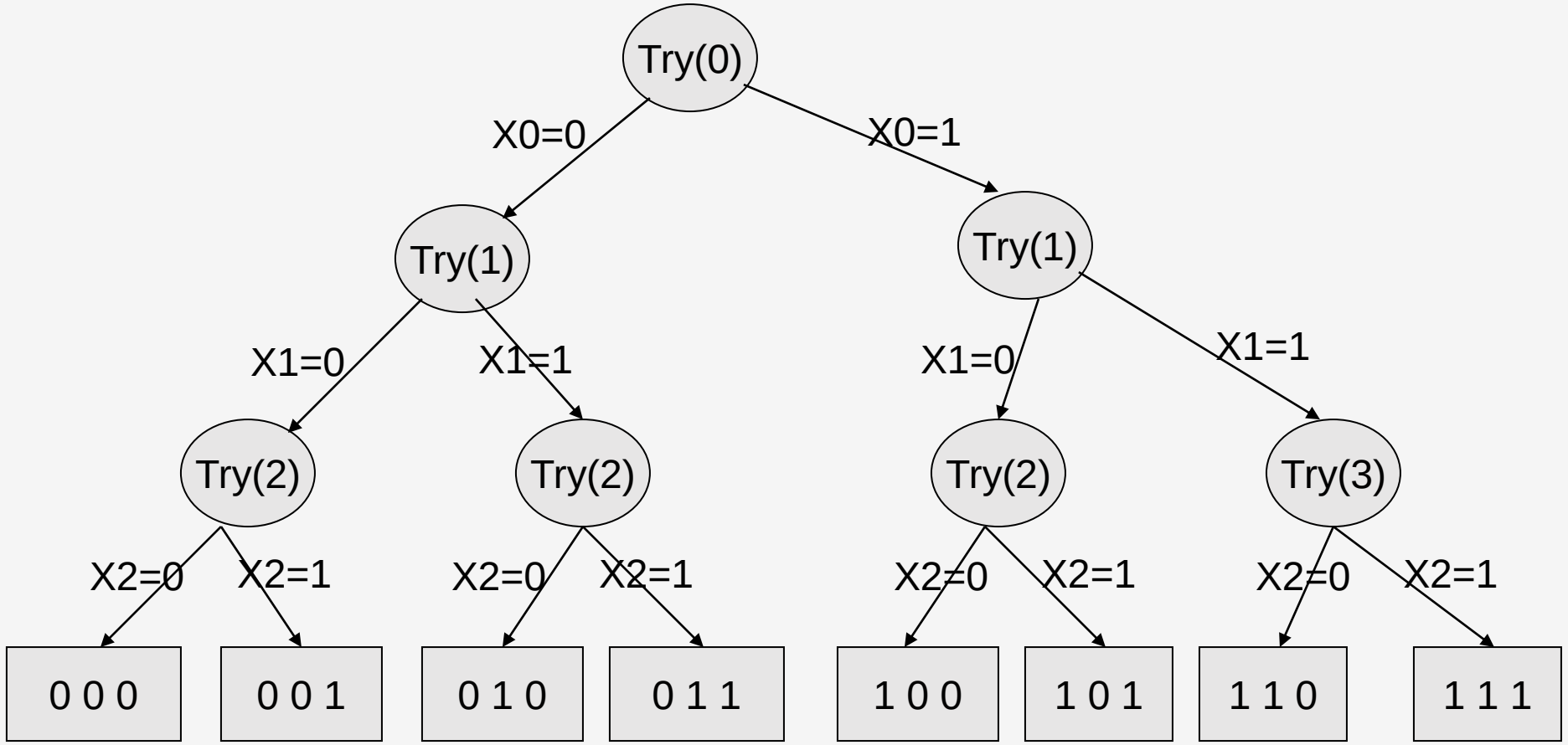
- Nếu không có khả năng j nào được chấp thuận cho thành phần xi thì ta quay lại bước trước đó (i-1) để thử lại các khả năng còn lại

Mô hình thuật toán quay lui (Back-track algorithm)

Mô hình tổng quát

```
Thuật toán Back-Track ( int i ) {
        for ( j =<Khả năng 1>; j <=ni; j++ ){
                if (<chấp thuận khả năng j>) {
                        X[i] = <khả năng j>;
                        if ( i ==n - 1) Result();
                        else Back-Track(i+1);
                }
        }
}
```

Mô hình thuật toán quay lui (Back-track algorithm)



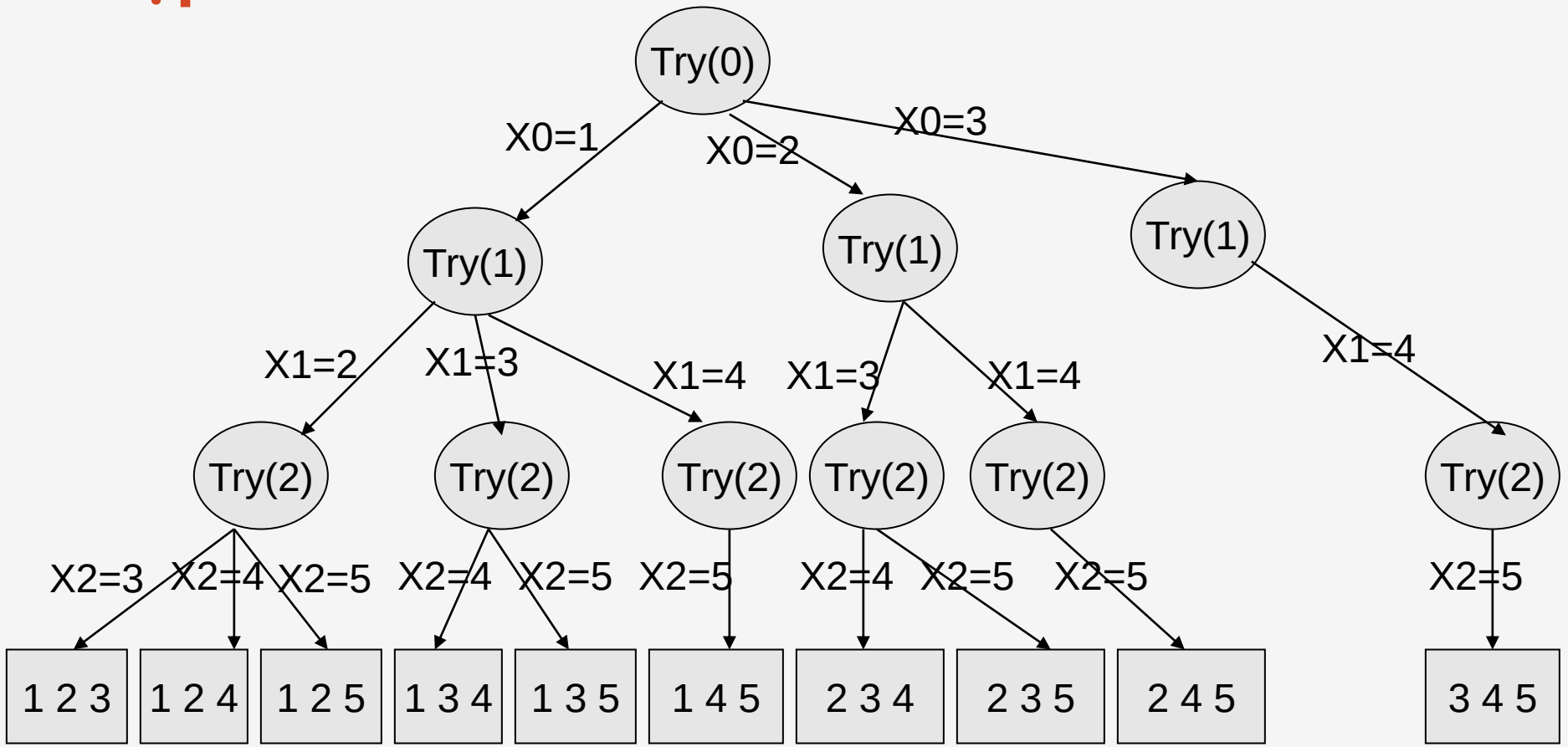
Mô hình thuật toán quay lui (Back-track algorithm)

Duyệt xâu nhị phân

```
void Try(int a[], int n, int i){
    for (int j=0; j<=1; j++){
        a[i] = j;
        if(i == n - 1) {
            printArray(a, n);
        } else {
            Try(a, n, i+1);
        }
    }
}
```

Mô hình thuật toán quay lui (Back-track algorithm)

Duyệt tổ hợp



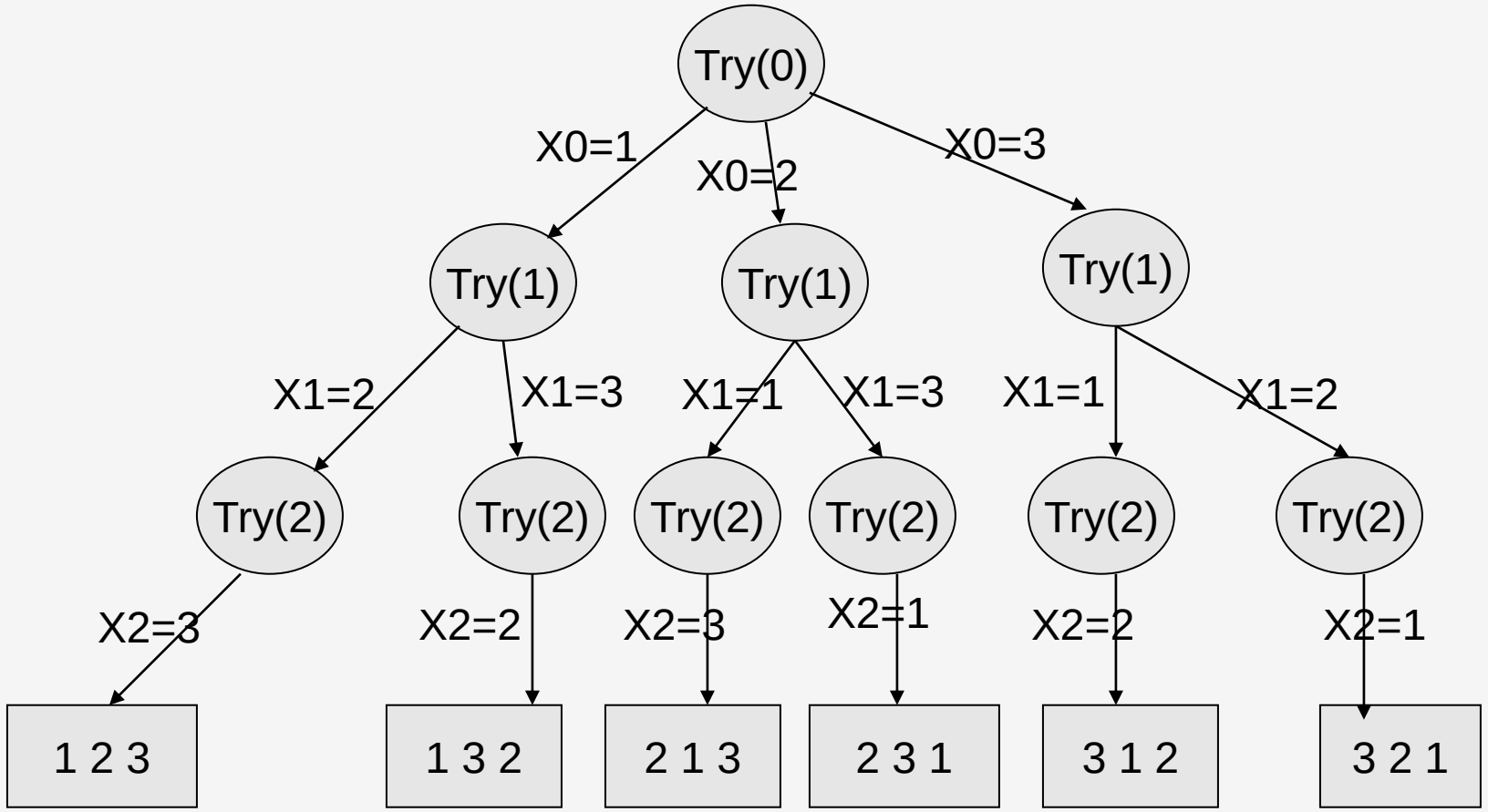
Mô hình thuật toán quay lui (Back-track algorithm)

Duyệt tổ hợp

```
void Try(int a[], int n, int k, int i){
    for (int j = (i - 1 >= 0 ? a[i-1] : 0) + 1; j <= n - k + i + 1; j++){
        a[i]=j;
        if(i==k - 1 ) {
            printArray(a, k);
        }
        else {
            Try(a, n, k, i+1);
        }
    }
}
```

Mô hình thuật toán quay lui (Back-track algorithm)

Duyệt hoán vị



Mô hình thuật toán quay lui (Back-track algorithm)

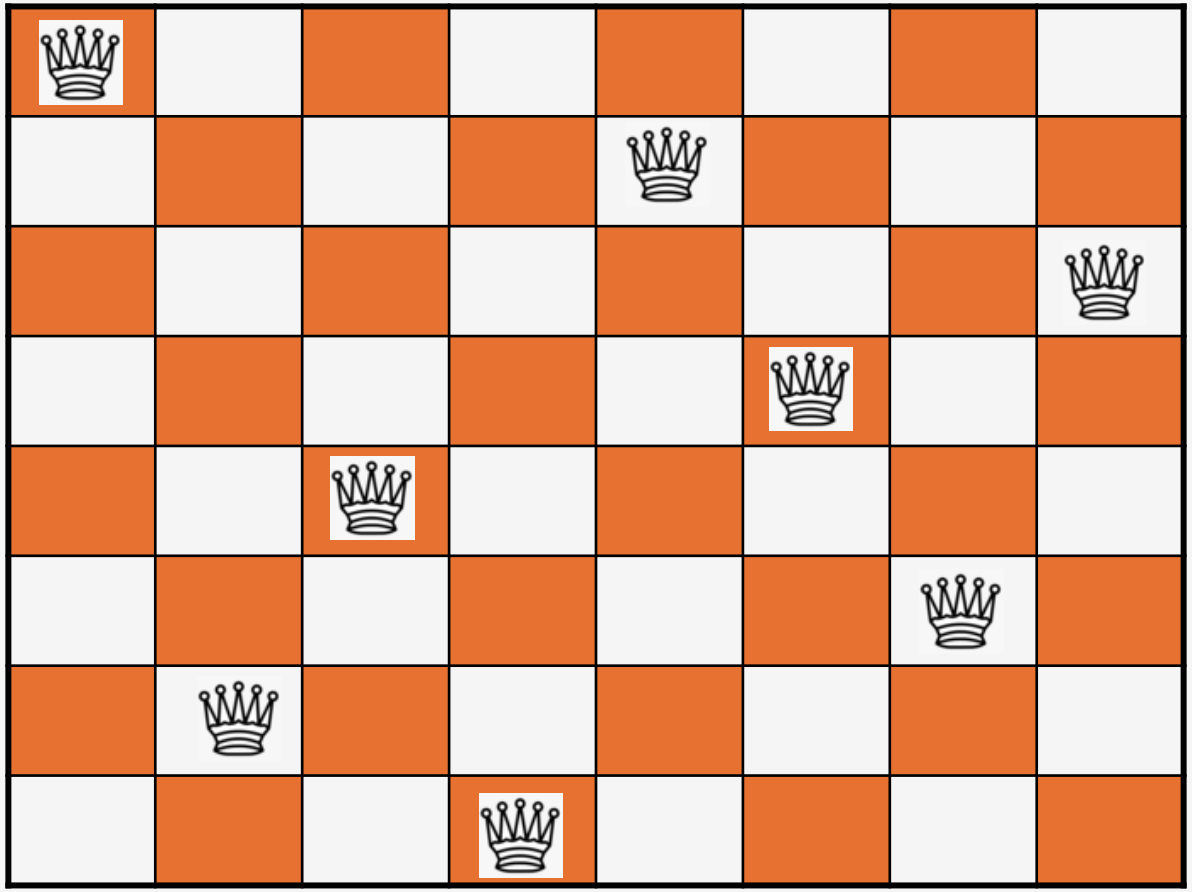
Duyệt hoán vị

```
void Try(int a[], int n, int used[], int i){
    for (int j= 1; j <= n; j++){
        if(!used[j]){
            a[i] = j;
            used[j] = 1;
            if (i == n - 1)
                printArray(a, n);
            else
                Try(a, n, used, i+1);
            used[j] = 0;
        }
    }
}
```

Mô hình thuật toán quay lui (Back-track algorithm)

Bài toán N quân hậu

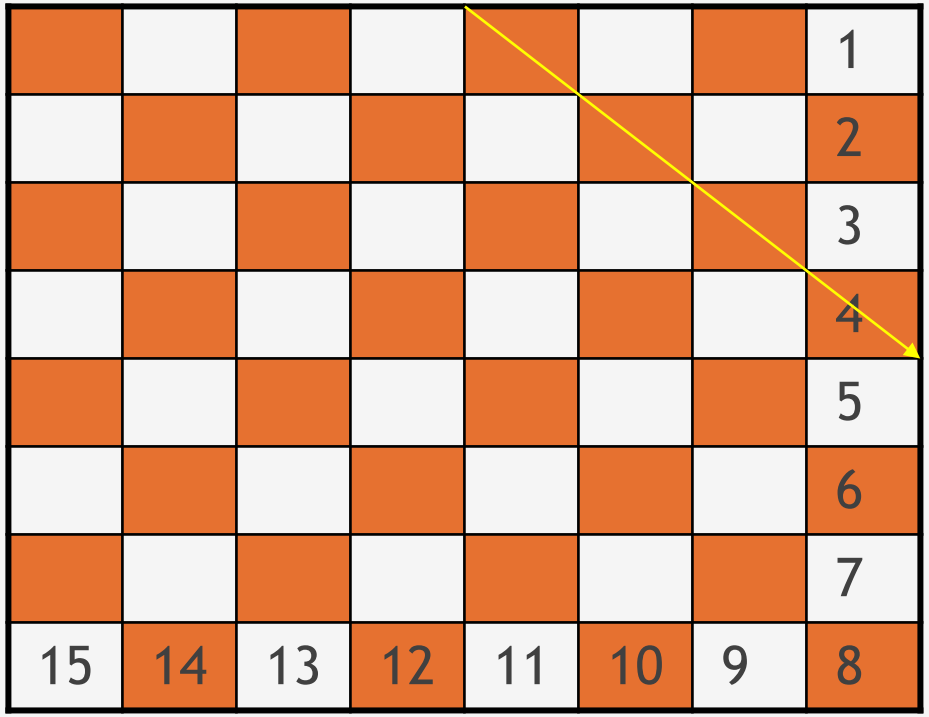
1 5 8 6 3 7 2 4



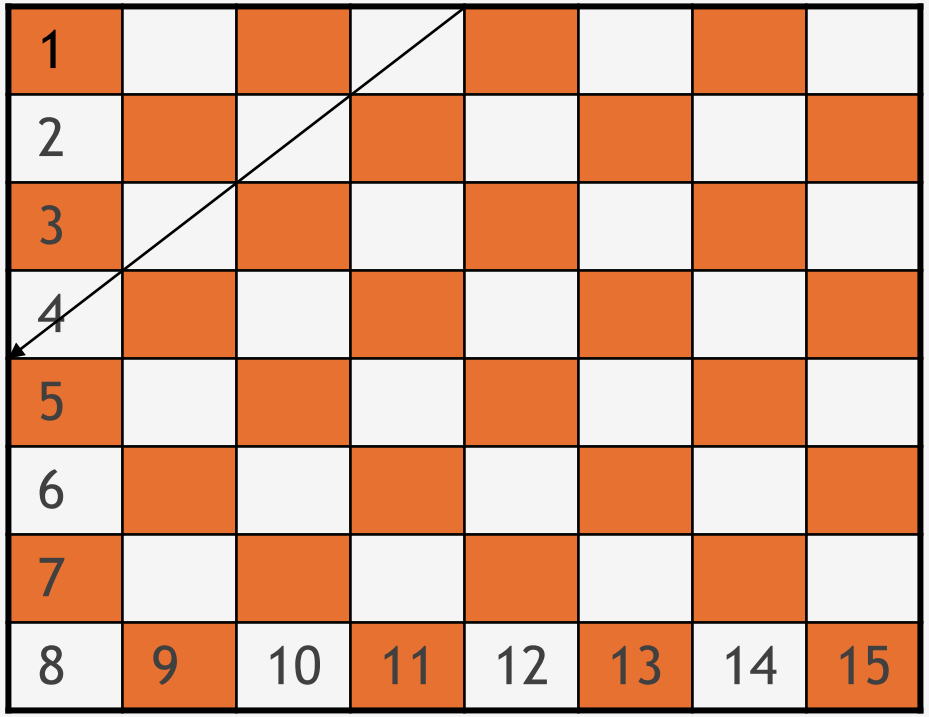
Mô hình thuật toán quay lui (Back-track algorithm)

Bài toán quân hậu

Đường chéo xuôi: Used1 [i – j + n]



Đường chéo ngược: Used2 [i + j -1]



Mô hình thuật toán quay lui (Back-track algorithm)

Bài toán quân hậu

Bài toán: Trên bàn cờ kích cỡ N×N, hãy đặt N quân hậu mỗi quân trên 1 hàng sao cho tất cả các quân hậu đều không ăn được lẫn nhau.

Phân tích. Gọi X =(x_1, x_2,..,x_n) là một nghiệm của bài toán. Khi đó, x_i = j được hiểu là quân hậu hàng thứ i đặt ở cột j.

Để các quân hậu khác không thể ăn được, quân hậu thứ i cần:

- Không được lấy trùng với bất kỳ cột nào,
- Không được cùng đường chéo xuôi,
- Không được cùng đường chéo ngược.

Ta có n cột Used = (a_1,..a_n), có **Used1[2*n-1]** đường chéo xuôi, **Used2[2*n-1]** đường chéo ngược.

- Nếu x_i =j thì
  - Used[j] = True
  - Used1[i-j+n] = True
  - Used2[i + j -1] = True.

Mô hình thuật toán tham lam (Greedy Algorithm)

Giới thiệu thuật toán tham lam

Thuật toán tham lam (Greedy Algorithm) xây dựng một chiến lược "tham từng miếng", miếng dễ tham nhất thường là một giải pháp **tối ưu cục bộ** nhưng lại luôn có mặt trong cấu trúc tối ưu toàn cục. Tiếp tục phát triển chiến lược "tham từng miếng" cho các bước tiếp theo để đạt được kết quả tối ưu toàn cục

Bài toán có thể sử dụng tham lam khi thỏa mãn 2 điều kiện:

- Ở mỗi bước ta luôn tạo ra một lựa chọn tốt nhất tại thời điểm đó.
- Việc liên kết lại kết quả mỗi bước sẽ cho ta kết quả tối ưu toàn cục.

Mô hình thuật toán tham lam (Greedy Algorithm)

Giới thiệu thuật toán tham lam

Thuật toán tham lam gồm 5 thành phần chính:

1) Một tập các ứng viên (candidate members) mà giải pháp có thể tham lam.

2) Một hàm lựa chọn (selection function) để chọn ứng viên tốt nhất cho giải pháp tham lam cục bộ.

3) Một hàm thực thi (feasibility function) được sử dụng để quyết định xem một ứng viên có được dùng để xây dựng lời giải hay không?

4) Một hàm mục tiêu (objective function) dùng để xác định giá trị của lời giải hoặc một phần của lời giải.

5) Một hàm giải pháp (solution function) dùng để xác định khi nào giải pháp hoàn chỉnh.

Mô hình thuật toán tham lam (Greedy Algorithm)

Ví dụ bài toán tìm số lớn nhất

Số lớn nhất gồm các chữ số 3, 9, 5, 9, 7, 1 là số nào?

359179, 537991, 913579, . . .?

997531

Mô hình thuật toán tham lam (Greedy Algorithm)

Ví dụ bài toán tìm số lớn nhất

Chiến lược

- Tìm chữ số lớn nhất
- Thêm chữ số đó vào kết quả
- Xóa chữ số đó khỏi dãy ban đầu
- Lặp lại quá trình khi dãy ban đầu vẫn còn chữ số

Mô hình thuật toán tham lam (Greedy Algorithm)

Ví dụ bài toán tìm số lớn nhất

3 9 5 9 7 1 -> 9

3 5 9 7 1 -> 99

3 5 7 1 -> 997

3 5 1 -> 9975

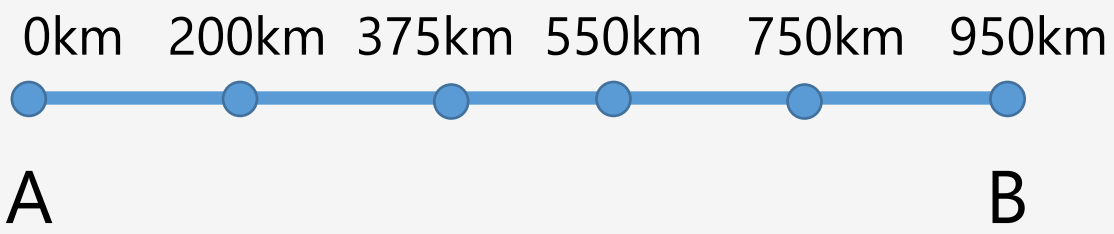
3 1 -> 99753

1 -> 997531

Mô hình thuật toán tham lam (Greedy Algorithm)

Bài toán đổ xăng

Khoảng cách tối đa xe có thể đi được khi đổ đầy bình xăng là 400km



Tìm phương án đi có số lần đổ xăng ít nhất

Mô hình thuật toán tham lam (Greedy Algorithm)

Bài toán đổ xăng

Input: Một chiếc ô tô có thể đi được nhiều nhất L km với đầy bình, đi từ điểm A, đến điểm B và n trạm xăng ở khoảng cách $x_1 \leq x_2 \leq x_3 \leq \cdots \leq x_n$ tính bằng km tính từ A theo đường đi từ A đến B.

Output: Số lần đổ xăng tối thiểu để đi từ A đến B, ngoài việc đổ đầy xăng tại điểm A

Mô hình thuật toán tham lam (Greedy Algorithm)

Bài toán đổ xăng

Có ba chiến lược:

- Đi đến điểm dừng gần nhất và đổ đầy xăng
- Đổ xăng tại trạm xa nhất có thể đi đến được
- Cứ đi cho đến khi hết xăng và hy vọng sẽ có cây xăng ở vị trí đó

Chiến lược nào là đúng?

Mô hình thuật toán tham lam (Greedy Algorithm)

Bài toán đổ xăng

Tham lam:

- Điểm bắt đầu là A, điểm B là điểm đến với số lần đổ xăng tối thiểu
- Gọi G là trạm xăng xa nhất có thể tới được
- Giảm bài toán ban đầu thành một bài toán tương tự bằng cách biến G thành A mới

Mô hình thuật toán tham lam (Greedy Algorithm)

Bài toán đổ xăng

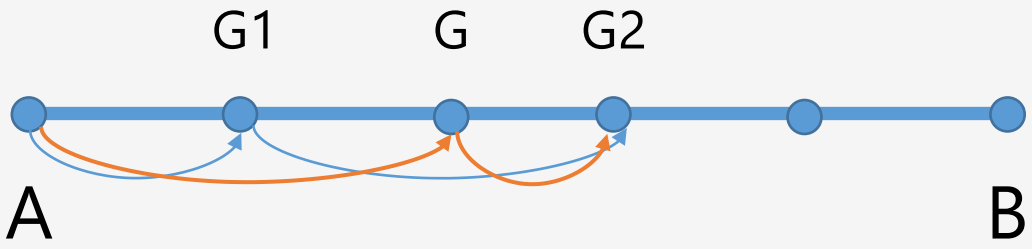
Chứng minh chiến lược thứ 2 là đúng

- Gọi G1 là trạm xăng đầu tiên trên đường từ A đến B
- Gọi G là trạm xăng xa nhất có thể đến được từ A
    + G trùng G1 (Chiến lược đúng)
    + G1 gần A hơn G, Xét G2 là điểm từng tiếp theo:
        * G gần A hơn G2, tốn 1 lần đổ xăng để tới G2 => (Đúng)
        * G2 gần A hơn G, tốn 1 lần đổ xăng chưa tới được G. (Đúng)

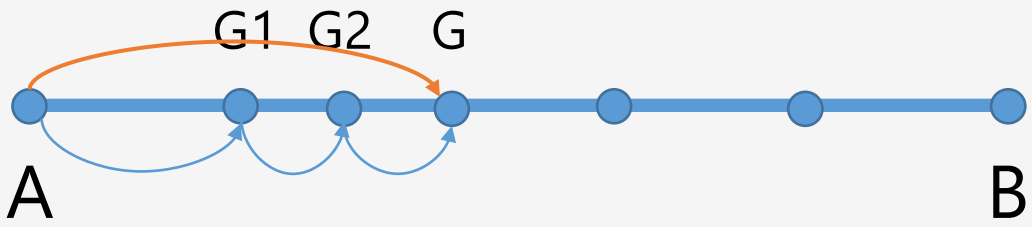
Mô hình thuật toán tham lam (Greedy Algorithm)

Bài toán đổ xăng

TH1



TH2



Mô hình thuật toán tham lam (Greedy Algorithm)

Bài toán đổ xăng

A = x0 <= x1 <= x2 <= ... <= xn <= xn + 1 = B

```
int min_refills(int dist, int tank, vector<int> stops) {
    int numR = 0, currR = 0, lastR, n = stops.size();
    n = n - 2;
    while (currR <= n) {
        lastR = currR;
        while(currR <= n && stops[currR + 1] - stops[lastR] <= tank)
            currR += 1;
        if(currR == lastR)
            return -1;
        if(currR <= n)
            numR += 1;
    }
    return numR;
}
```

Mô hình thuật toán tham lam (Greedy Algorithm)

Chia nhóm trẻ em

Mời rất nhiều trẻ em tham gia bữa tiệc và thuê một vài giáo viên trông trẻ.

Chia trẻ em thành các nhóm, mỗi giáo viên sẽ phụ trách 1 nhóm.

Để quản lý hiệu quả thì ở mỗi nhóm, trẻ em nên cùng độ tuổi hoặc tương đương (tuổi của 2 trẻ bất kỳ không chênh lệch quá 1 tuổi).

Để tiết kiệm chi phí, cần tối ưu để số nhóm được chia ít nhất có thể.

Ví dụ: Có 4 trẻ, 1: 3 tuổi 2 tháng; 2: 3 tuổi 8 tháng, 3: 4 tuổi 6 tháng, 4: 5 tuổi. Số nhóm ít nhất để đảm bảo trong mỗi nhóm trẻ không chênh lệch quá 1 tuổi là bao nhiêu?

Mô hình thuật toán tham lam (Greedy Algorithm)

Chia nhóm trẻ em – tham lam

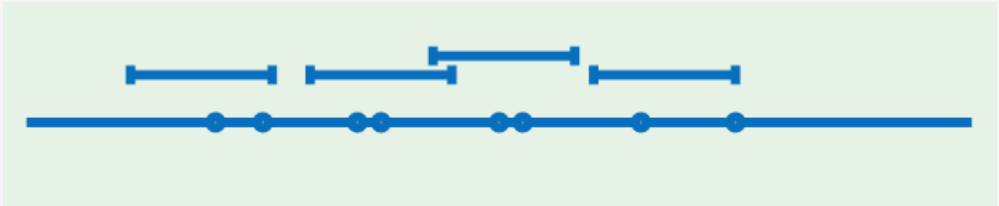
Biểu diễn các điểm trên đường thẳng thay cho trẻ em

VD: 3,5 trên trục số tương đương trẻ em 3 tuổi 6 tháng.

Mô hình thuật toán tham lam (Greedy Algorithm)

Chia nhóm trẻ em – tham lam

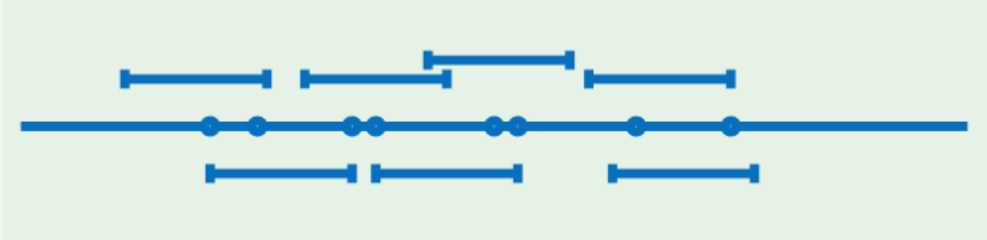
Ví dụ: 4 đoạn thẳng tương đương với 4 nhóm.



Mô hình thuật toán tham lam (Greedy Algorithm)

Chia nhóm trẻ em – tham lam

Ví dụ: 3 đoạn thẳng tương đương với 3 nhóm.



Mô hình thuật toán tham lam (Greedy Algorithm)

Chia nhóm trẻ em – tham lam

Giả sử $x_1 \le x_2 \le \ldots \le x_n$

PointsCoverSorted($x_1, \ldots, x_n$)

$R \leftarrow \{\}$, $i \leftarrow 1$

while $i \le n$:

    $[\ell, r] \leftarrow [x_i, x_i + 1]$

    $R \leftarrow R \cup \{[\ell, r]\}$

    $i \leftarrow i + 1$

    while $i \le n$ and $x_i \le r$:

        $i \leftarrow i + 1$

return R

Mô hình thuật toán tham lam (Greedy Algorithm)

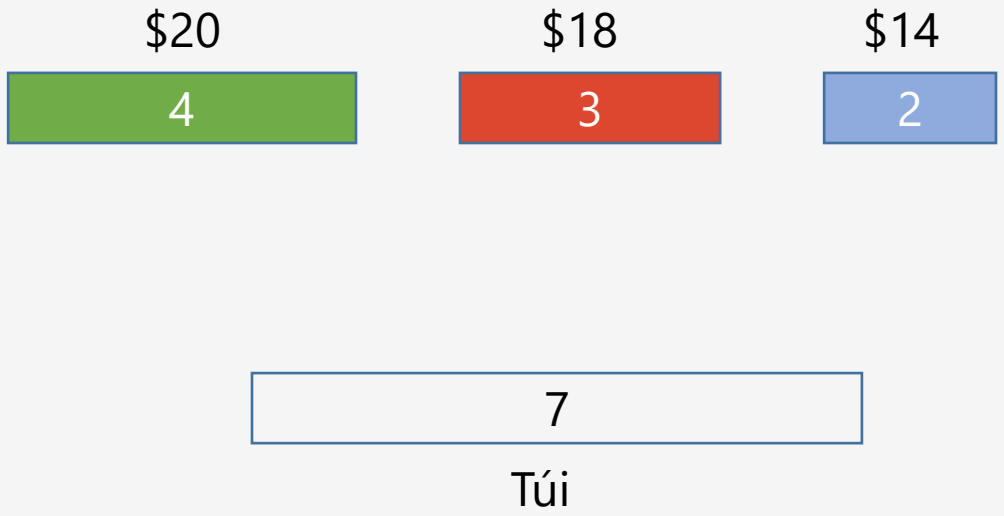
Bài toán cái túi

Input: Trọng lượng w1, . . . , wn và giá trị v1, . . . , vn của n thực phẩm; Tổng trọng lượng của túi là: W.

Output: Tổng giá trị lớn nhất của các phần nhỏ của vật phẩm có trong một túi có trọng lượng W.

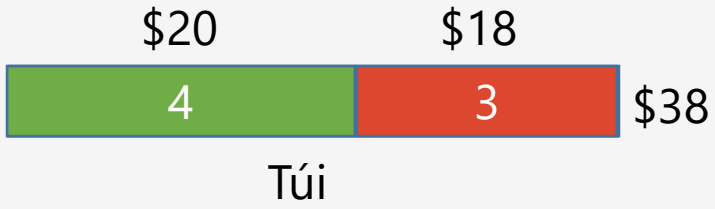
Mô hình thuật toán tham lam (Greedy Algorithm)

Bài toán cái túi



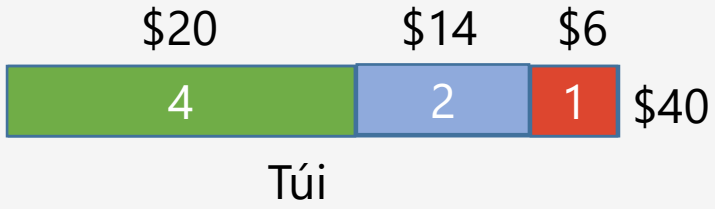
Mô hình thuật toán tham lam (Greedy Algorithm)

Bài toán cái túi



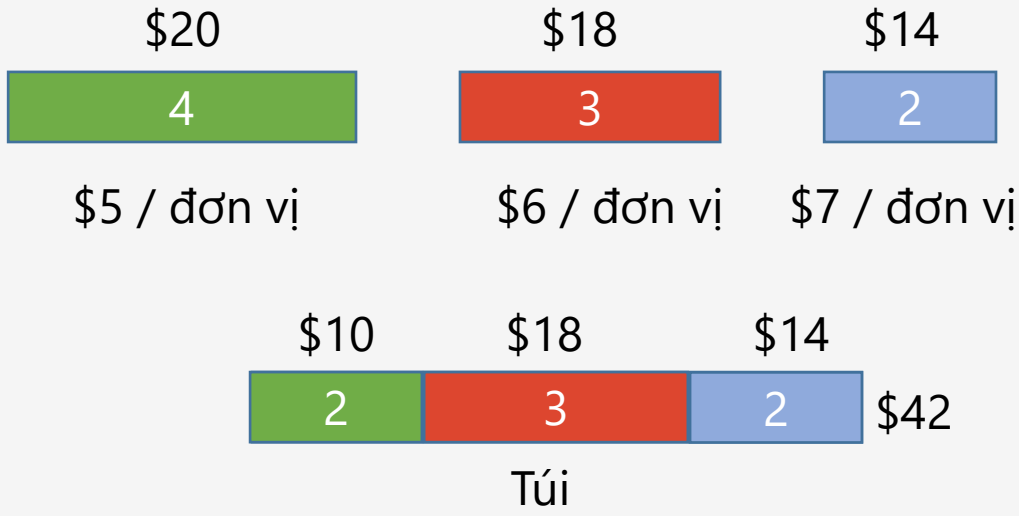
Mô hình thuật toán tham lam (Greedy Algorithm)

Bài toán cái túi



Mô hình thuật toán tham lam (Greedy Algorithm)

Bài toán cái túi



Tồn tại một giải pháp tối ưu sử dụng càng nhiều càng tốt một mặt hàng có giá trị lớn nhất trên một đơn vị trọng lượng.

Mô hình thuật toán tham lam (Greedy Algorithm)

Bài toán cái túi

Knapsack(W , w_1, v_1, . . . , w_n, v_n)

A ← [0, 0, . . . , 0], V ← 0

repeat n times:

    if W = 0:

        return (V , A)

    select i with $w_i > 0$ and max $\frac{v_i}{w_i}$

    a ← min(w_i , W)

    V ← V + a$\frac{v_i}{w_i}$

    $w_i \leftarrow w_i - a$, A[i] ← A[i] + a, W ← W − a

return (V , A)

Mô hình thuật toán tham lam (Greedy Algorithm)

Bài toán cái túi – tối ưu

Giả sử $\frac{v_1}{w_1} \geq \frac{v_2}{w_2} \geq \cdots \geq \frac{v_n}{w_n}$

Knapsack(W , w_1, v_1, . . . , w_n, v_n)

$A \leftarrow [0, 0, \ldots, 0]$, $V \leftarrow 0$

for i from 1 to n:

    if $W = 0$:

        return (V , A)

    $a \leftarrow \min(w_i , W)$

    $V \leftarrow V + a\frac{v_i}{w_i}$

    $w_i \leftarrow w_i - a$, $A[i] \leftarrow A[i] + a$, $W \leftarrow W - a$

return (V , A)

Mô hình thuật toán tham lam (Greedy Algorithm)

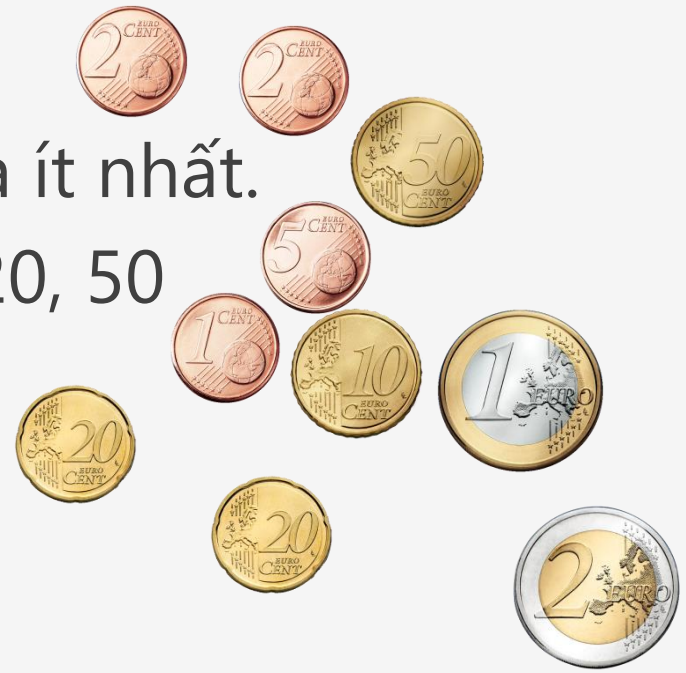
Bài toán đổi tiền

Mô tả bài toán:

Cho trước một số tiền và tập đồng xu.

Hãy tìm cách đổi tiền sao cho số đồng xu là ít nhất.

Trường hợp 1: xét tập đồng xu 1, 2, 5, 10, 20, 50



Mô hình thuật toán tham lam (Greedy Algorithm)

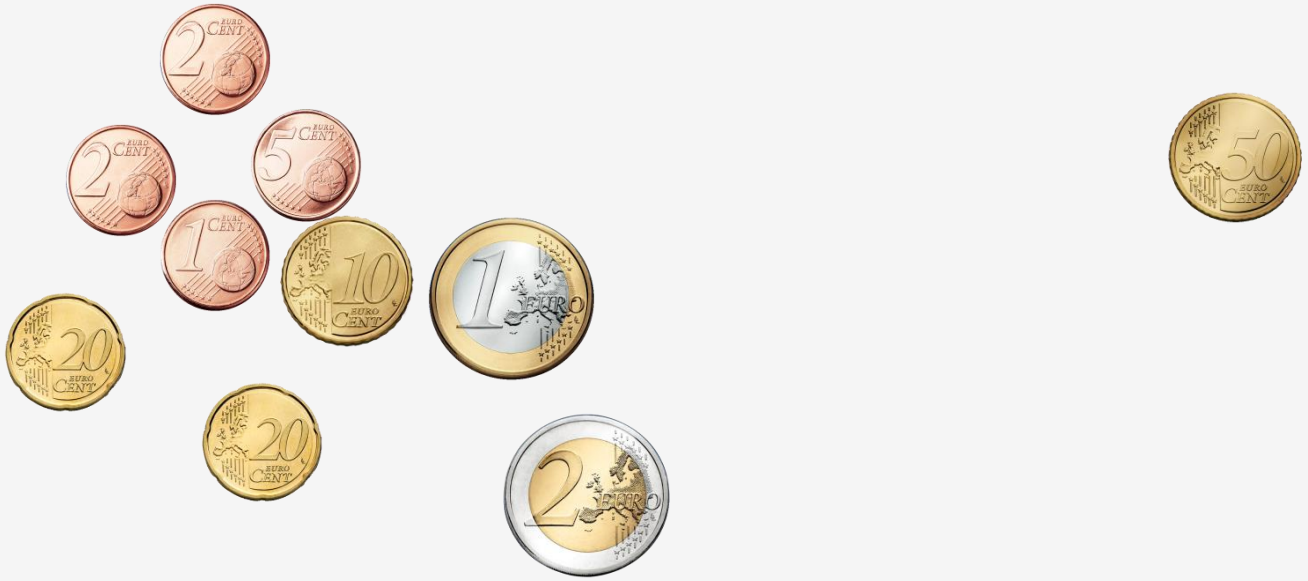
Bài toán đổi tiền

Trường hợp 1: xét tập đồng xu 1, 2, 5, 10, 20, 50

Giải pháp tham lam:

- Chọn các đồng xu từ giá trị lớn đến nhỏ

Cần đổi 72 xu



Mô hình thuật toán tham lam (Greedy Algorithm)

Bài toán đổi tiền

Trường hợp 1: xét tập đồng xu 1, 2, 5, 10, 20, 50

Giải pháp tham lam:

- Chọn các đồng xu từ giá trị lớn đến nhỏ

Cần đổi 72 xu

Mô hình thuật toán tham lam (Greedy Algorithm)

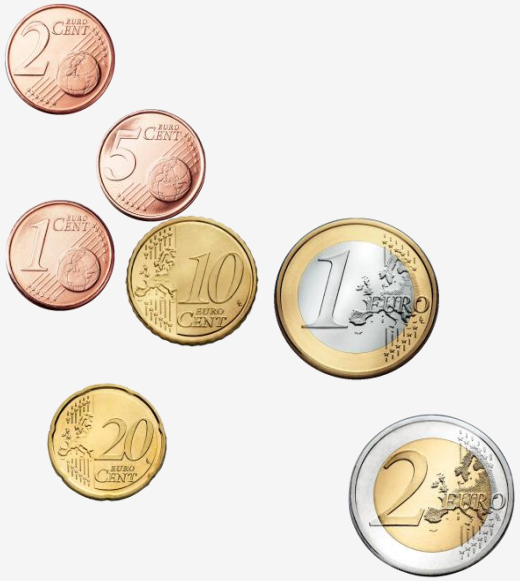
Bài toán đổi tiền

Trường hợp 1: xét tập đồng xu 1, 2, 5, 10, 20, 50

Giải pháp tham lam:

- Chọn các đồng xu từ giá trị lớn đến nhỏ

Cần đổi 72 xu



Kết quả tối ưu

Mô hình thuật toán tham lam (Greedy Algorithm)

Bài toán đổi tiền

Mô tả bài toán:

Cho trước một số tiền và tập đồng xu.

Hãy tìm cách đổi tiền sao cho số đồng xu là ít nhất.

Trường hợp 2: xét tập đồng xu 1, 4, 9, 16, 25, 36, 49

Mô hình thuật toán tham lam (Greedy Algorithm)

Bài toán đổi tiền

Trường hợp 2: xét tập đồng xu 1, 4, 9, 16, 25, 36, 49

Kết quả sử dụng tham lam: 72 = 49 + 16 + 4 + 1 + 1 + 1

Mô hình thuật toán tham lam (Greedy Algorithm)

Bài toán đổi tiền

Trường hợp 2: xét tập đồng xu 1, 4, 9, 16, 25, 36, 49

Kết quả sử tối ưu: 72 = 36 + 36

Mô hình thuật toán tham lam (Greedy Algorithm)

Bài toán đổi tiền

Câu hỏi: Vì sao trường hợp 1 thì giải thuật tham lam cho kết quả tối ưu, trường hợp 2 thì không?

```
void greedy(int value, int coins[], int ans[], int n) {
    for ( int i = n - 1; i >= 0; --i ) {
        ans[i] = 0;
        while ( coins[i] <= value ) {
            value -= coins[i];
            ans[i]++;
        }
    }
}
```

Mô hình thuật toán tham lam (Greedy Algorithm)

Bài toán sudoku

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

Mô hình thuật toán tham lam (Greedy Algorithm)

Bài toán sudoku

Hãy xét cách tham lam theo giá trị từ nhỏ đến lớn.

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

Mô hình thuật toán tham lam (Greedy Algorithm)

Bài toán sudoku

Hãy xét cách tham lam theo giá trị từ nhỏ đến lớn.

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

Mô hình thuật toán tham lam (Greedy Algorithm)

Bài toán sudoku

Hãy xét cách tham lam theo giá trị từ nhỏ đến lớn.

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

Mô hình thuật toán tham lam (Greedy Algorithm)

Bài toán sudoku

Hãy xét cách tham lam theo giá trị từ nhỏ đến lớn.

8	3	1	6	4	?			2
	4			5			1	
			7					3
	9				4			6
2								8
7				1			5	
3					9			
	1			8			9	
4					2			5

Mô hình thuật toán tham lam (Greedy Algorithm)

Bài toán sudoku

Không thể giải bài toán này theo giải thuật tham lam

8	3	1	6	4	?			2
	4			5			1	
			7					3
	9				4			6
2								8
7				1			5	
3					9			
	1			8			9	
4					2			5

Mô hình thuật toán tham lam (Greedy Algorithm)

Tham lam gần đúng

Xét bài toán sau (một phiên bản của bài toán cái túi)

Một dự án sản xuất có tổng thời gian thực hiện là 26 tuần

Mỗi sản phẩm (từ A đến J) có thời gian thực hiện và giá trị cho trước

Hãy sắp xếp thứ tự sản xuất các sản phẩm sao cho tổng giá trị là lớn nhất.

Mô hình thuật toán tham lam (Greedy Algorithm)

Tham lam gần đúng

Product ID	Completion Time (wks)	Expected Revenue (1000 $)
A	15	210
B	12	220
C	10	180
D	9	120
E	8	160
F	7	170
G	5	90
H	4	40
I	3	60
J	1	10

Mô hình thuật toán tham lam (Greedy Algorithm)

Tham lam gần đúng

Giải pháp Tham lam theo thời gian:

Project A: 15 wks

Project C: 10 wks

Project J: 1 wk

Tổng thời gian: 26 wks

Giá trị:

$400 000

Product ID	Completion Time (wks)	Expected Revenue (1000 $)
A	**15**	**210**
B	12	220
C	**10**	**180**
D	9	120
E	8	160
F	7	170
G	5	90
H	4	40
I	3	60
J	**1**	**10**

Mô hình thuật toán tham lam (Greedy Algorithm)

Tham lam gần đúng

Giải pháp tham lam theo giá trị:

Project B: $220K

Project C: $180K

Project H: $ 60K

Project K: $ 10K

Tổng thời gian: 26 wks

Giá trị:

$470 000

Product ID	Completion Time (wks)	Expected Revenue (1000 $)
B	12	220
A	15	210
C	10	180
F	7	170
E	8	160
D	9	120
G	5	90
I	3	60
H	4	40
J	1	10

Mô hình thuật toán tham lam (Greedy Algorithm)

Tham lam gần đúng

Product ID	Completion Time (wks)	Expected Revenue (1000 $)	Revenue Density ($ / wk)
A	15	210	**14 000**
B	12	220	**18 333**
C	10	180	**18 000**
D	9	120	**13 333**
E	8	160	**20 000**
F	7	170	**24 286**
G	5	90	**18 000**
H	4	40	**10 000**
I	3	60	**20 000**
J	1	10	**10 000**

Mô hình thuật toán tham lam (Greedy Algorithm)

Tham lam gần đúng

Giải pháp tham lam theo tỉ lệ giá trị trên thời gian:

Project F: $24 286/wk

Project E: $20 000/wk

Project I: $20 000/wk

Project G: $18 000/wk

Project J: $10 000/wk

Tổng thời gian: 24 wks

Giá trị:

$490 000

Product ID	Completion Time (wks)	Expected Revenue (1000 $)	Revenue Density ($/wk)
F	7	170	24 286
E	8	160	20 000
I	3	60	20 000
B	12	220	18 333
C	10	180	18 000
G	5	90	18 000
A	15	210	14 000
D	9	120	13 333
H	4	40	10 000
J	1	10	10 000

Mô hình thuật toán tham lam (Greedy Algorithm)

Tham lam gần đúng

Giải pháp duyệt vét cạn:

Project C: $180 000

Project E: $170 000

Project F: $150 000

Project J: $ 10 000

Tổng thời gian: 26 wks

Giá trị:

$520 000

Product ID	Completion Time (wks)	Expected Revenue (1000 $)	Revenue Density ($/wk)
A	15	210	14 000
B	12	220	18 333
C	10	180	18 000
D	9	120	13 333
E	8	160	20 000
F	7	170	24 286
G	5	90	18 000
H	4	40	10 000
I	3	60	20 000
J	1	10	10 000

Mô hình thuật toán tham lam (Greedy Algorithm)

Tham lam gần đúng

Các giải pháp tham lam chỉ cho giá trị gần tối ưu.

Algorithm	Expected Revenue
Theo thời gian	$400 000
Theo giá trị	$470 000
Theo tỉ lệ giá trị/thời gian	$490 000
Vét cạn	$520 000

Mô hình thuật toán tham lam (Greedy Algorithm)

Nhận xét chung về giải thuật tham lam

- Cùng một dạng bài toán nhưng các trường hợp dữ liệu khác nhau dẫn tới có thể giải bằng tham lam được hay không.
- Có những bài toán không thể giải bằng tham lam
- Có những bài toán có thể chấp nhận tham lam cho ra kết quả gần đúng.
- Với các bài toán áp dụng được tham lam: cần chứng minh về mặt toán học.

Yêu cầu cơ bản: hãy cài đặt các giải thuật tham lam đã được chứng minh tính đúng đắn.

Mô hình thuật toán tham lam (Greedy Algorithm)

Các bài toán có thể áp dụng giải thuật tham lam

- Trong lý thuyết đồ thị:
    + Tìm cây khung nhỏ nhất: KRUSKAL, PRIM, BORUVKA
    + Tìm đường đi ngắn nhất DIJKSTRA

- Một số bài toán khác:
    + Bài toán đổi tiền
    + Bài toán sắp xếp công việc
    + Bài toán nối dây
    + Bài toán sắp đặt xâu ký tự

Mô hình thuật toán tham lam (Greedy Algorithm)

Bài toán sắp xếp công việc

Cho tập gồm n công việc, mỗi công việc được biểu diễn bởi cặp thời gian bắt đầu si và thời gian kết thúc fi (i=1, 2, .., n).

Hãy lựa chọn nhiều nhất các công việc có thể thực hiện tuần tự bởi một máy hoặc một cá nhân mà không xảy ra tranh chấp.

Mỗi công việc chỉ thực hiện đơn lẻ tại một thời điểm.

Ví dụ: Số lượng công việc: 6

Thời gian bắt đầu Start []= { 1, 3, 0, 5, 8, 5}

Thời gian kết thúc Finish[]= { 2, 4, 6, 7, 9, 9}

Output:

OPT[] = {1, 2, 4, 5 }

Mô hình thuật toán tham lam (Greedy Algorithm)

Bài toán sắp xếp công việc

Sắp xếp công việc (N,S[], F[]):

Input:

N là số lượng công việc.

S[] thời gian bắt đầu.

F[] thời gian kết thúc.

Output: Danh sách thực thi nhiều nhất.

Mô hình thuật toán tham lam (Greedy Algorithm)

Bài toán sắp xếp công việc

Bước 1 Sắp xếp thứ tự tăng dần của thời gian kết thúc.

Bước 2 (Khởi tạo) Lựa chọn công việc đầu tiên làm phương án tối ưu (OPT=1). N = N\{1};

Bước 3 (Lặp).

Với mỗi công việc $j \in N$

if (S[j] >=F[i]) {OPT = OPT $\cup$ j; i = j; N = N\{i} }

Bước 4 (Trả lại kết quả)

Mô hình thuật toán tham lam (Greedy Algorithm)

Bài toán sắp xếp công việc

Ví dụ thực nghiệm thuật toán:

Số lượng hành động n = 8.

Thời gian bắt đầu S[] = {1, 3, 0, 5, 8, 5, 9, 14}.

Thời gian kết thúc F[] = {2, 4, 6, 7, 9, 9, 12, 18}.

Mô hình thuật toán tham lam (Greedy Algorithm)

Bài toán sắp xếp công việc

Bước	i= ? j =?	(S[j] >=F[i]) ? i=?	OPT=?
			OPT = OPT$\cup${1}.
1	i=1; j=2;	(3>=2): Yes; i=2.	OPT = OPT$\cup${2}.
2	i=2; j=3;	(0>=4): No; i=2.	OPT = OPT$\cup\phi$.
3	i=2; j=4;	(5>=4): Yes; i=4.	OPT = OPT$\cup${4}.
4	i=4; j=5;	(8>=7): Yes; i=5.	OPT = OPT$\cup${5}.
5	i=5; j=6;	(5>=9): No; i=5.	OPT = OPT$\cup\phi$.
6	i=5; j=7;	(9>=9): Yes; i=7.	OPT = OPT$\cup${7}.
7	i=7; j=8;	(14>=12): Yes; i=8.	OPT = OPT$\cup${8}.
OPT = { 1, 2, 4, 5, 7, 8}			

Mô hình thuật toán tham lam (Greedy Algorithm)

Bài toán nối dây

Cho n dây với chiều dài khác nhau. Cần phải nối các dây lại với nhau thành một dây.

Chi phí nối hai dây lại với nhau được tính bằng tổng độ dài hai dây.

Nhiệm vụ của bài toán là tìm cách nối các dây lại với nhau thành một dây sao cho chi phí nối các dây lại với nhau là ít nhất.

Mô hình thuật toán tham lam (Greedy Algorithm)

Bài toán nối dây

Số lượng dây: 4; Độ dài dây L[]={4, 3, 2, 6}

Chi phí nối dây nhỏ nhất: OPT = 29

Cách làm:

Dây số 3 nối với dây số 2 => 3 dây với độ dài 4, 5, 6.

Dây độ dài 4 nối với dây độ dài 5 => 2 dây với độ dài 6, 9.

Nối hai dây còn lại 6+9 =15.

Tổng chi phí nhỏ nhất là 5 + 9 + 15 = 29.

Mô hình thuật toán tham lam (Greedy Algorithm)

Bài toán nối dây - tham lam sử dụng hàng đợi ưu tiên

Input: n: số lượng dây; L[] : chi phí nối dây.

Output: Chi phí nối dây nhỏ nhất.

Bước 1. Tạo pq là hàng đợi ưu tiên lưu trữ độ dài n dây.

Bước 2 (Lặp).

```
OPT = 0;
while (pq.size>1)
    First = pq.top; pq.pop();
    Second = pq.top; pq.pop();
    OPT = First + Second;
    Pq.push(First + Second);
```

Bước 3 (Trả lại kết quả). Return(OPT);

Mô hình thuật toán tham lam (Greedy Algorithm)

Bài toán nối dây - tham lam sử dụng hàng đợi ưu tiên

Input: Output: Chi phí nối dây nhỏ nhất.

Số lượng dây n = 8.

Chi phí nối dây:

L[] = { 9, 7, 12, 8, 6, 5, 14, 4}.

Bước	Giá trị First, Second	OPT=?	Trạng thái hàng đợi ưu tiên.
		0	4, 5, 6, 7, 8, 9, 12, 14
1	First=4; Second=5	9	6, 7, 8, 9, 9, 12, 14
2	First=6; Second=7	22	8, 9, 9, 12, 13, 14
3	First=8; Second=9	39	9, 12, 13, 14, 17
4	First=9; Second=12	60	13, 14, 17, 21
5	First=13; Second=14	87	17, 21, 27
6	First=17; Second=21	125	27, 38
7	First=27; Second=38	190	65
OPT = 190			

Mô hình thuật toán tham lam (Greedy Algorithm)

Bài toán sắp đặt xâu ký tự

Bài toán

Cho xâu ký tự s[] độ dài n và số tự nhiên d.

Hãy sắp đặt lại các ký tự trong xâu s[] sao cho hai ký tự giống nhau đều cách nhau một khoảng là d.

Nếu bài toán có nhiều nghiệm, đưa ra một cách sắp đặt đầu tiên tìm được. Nếu bài toán không có lời giải hãy đưa ra thông báo "Vô nghiệm".

Mô hình thuật toán tham lam (Greedy Algorithm)

Bài toán sắp đặt xâu ký tự

Input:

Xâu ký tự S[] =“ABB”;

Khoảng cách d = 2.

Output: BAB

Input:

Xâu ký tự S[] =“AAA”;

Khoảng cách d = 2.

Output: Vô nghiệm.

Input:

Xâu ký tự S[] =“GEEKSFORGEEKS”;

Khoảng cách d = 3.

Output: EGKEGKESFESOR

Mô hình thuật toán tham lam (Greedy Algorithm)

Bước 1. Tìm Freq[] là số lần xuất hiện mỗi ký tự trong xâu.

Bước 2. Sắp xếp theo thứ tự giảm dần theo số xuất hiện ký tự.

Bước 3. (Lặp).

```
i = 0; k =<Số lượng ký tự trong Freq[]>;
while (i < k ) {
        p = Max(Freq);
        for ( t = 0; t<p; t++ )
                if (i+(t*d)>n ) { < Không có lời giải>; return;>}
                KQ[i + (t*d)]= Freq[i].kytu;
        }  i++;
}
```

Bước 4(Trả lại kết quả): Return(KQ);

Mô hình thuật toán tham lam (Greedy Algorithm)

Bài toán sắp đặt xâu ký tự

Freq[]	
G	2
E	4
K	2
S	2
F	1
O	1
R	1

Sắp xếp theo thứ tự giảm dần của số lần xuất hiện.

Freq[]	
E	4
G	2
K	2
S	2
F	1
O	1
R	1

Mô hình thuật toán tham lam (Greedy Algorithm)

Bài toán sắp đặt xâu ký tự

	KQ[I + p*d]												
I =0	E			E			E			E			
i=1	E	G		E	G		E			E			
i=2	E	G	K	E	G	K	E			E			
i=3	E	G	K	E	G	K	E	S		E	S		
i=4	E	G	K	E	G	K	E	S	F	E	S		
i=5	E	G	K	E	G	K	E	S	F	E	S	O	
i=6	E	G	K	E	G	K	E	S	F	E	S	O	R

Mô hình thuật toán chia để trị (Divide and Conquer)

Chia để trị

Mục tiêu: Giảm thời gian chạy

Ý tưởng: Áp dụng đệ quy theo 3 bước

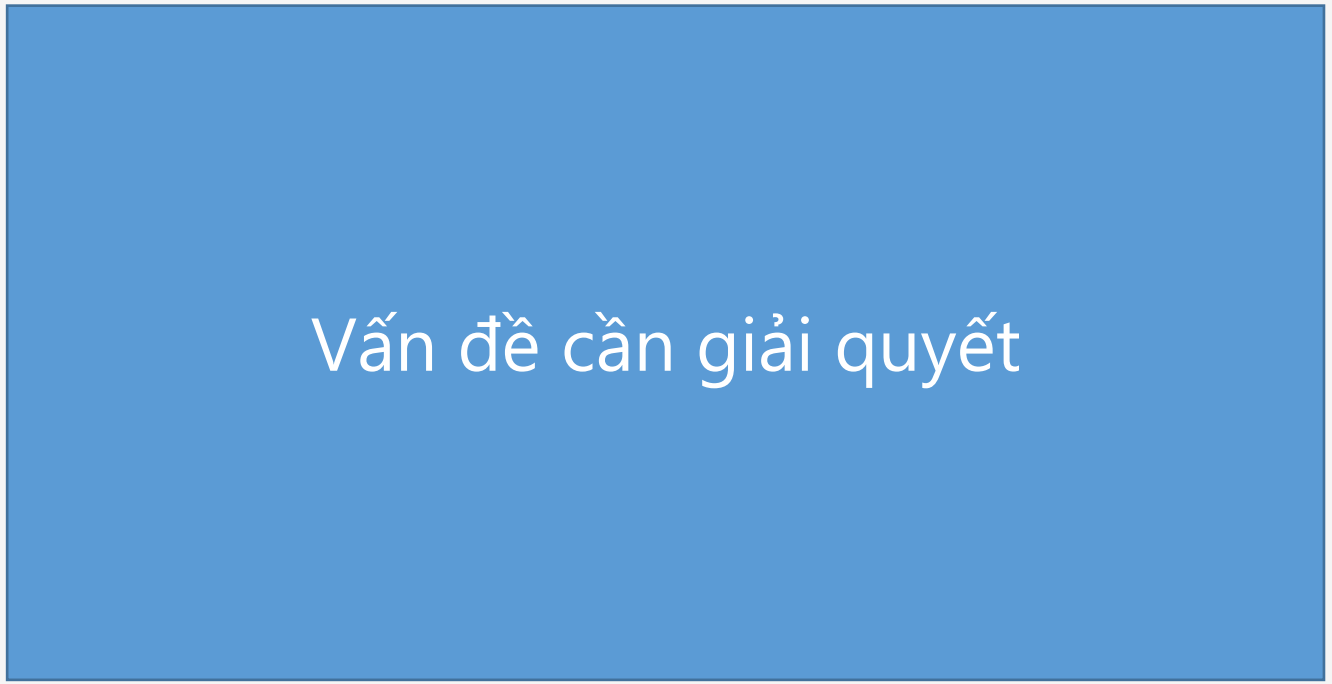
+ Divide (Chia): Chia bài toán lớn thành những bài toán con có cùng kiểu với bài toán lớn.

+ Conquer (Trị): Giải các bài toán con. Thông thường các bài toán con chỉ khác nhau về dữ liệu vào nên ta có thể thực hiện bằng một thủ tục đệ quy.

+ Combine (Tổng hợp): Tổng hợp lại kết quả của các bài toán con để nhận được kết quả của bài toán lớn.

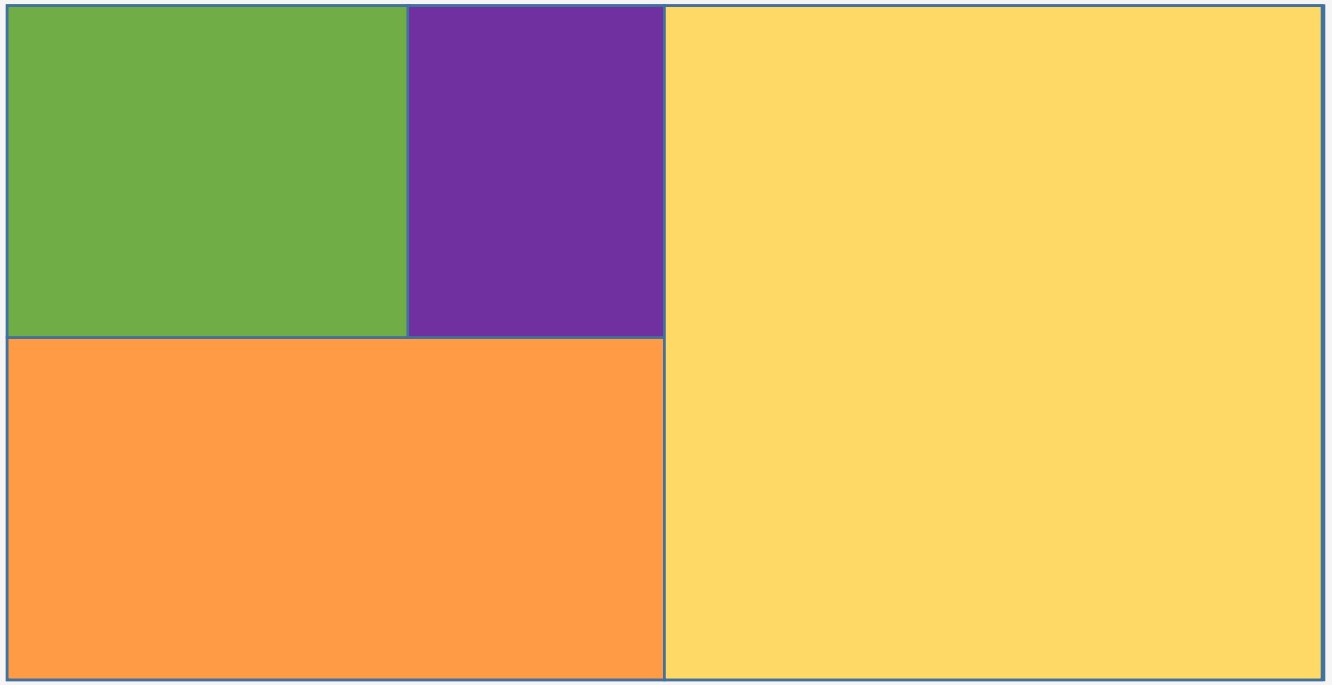
Mô hình thuật toán chia để trị (Divide and Conquer)

Chia để trị



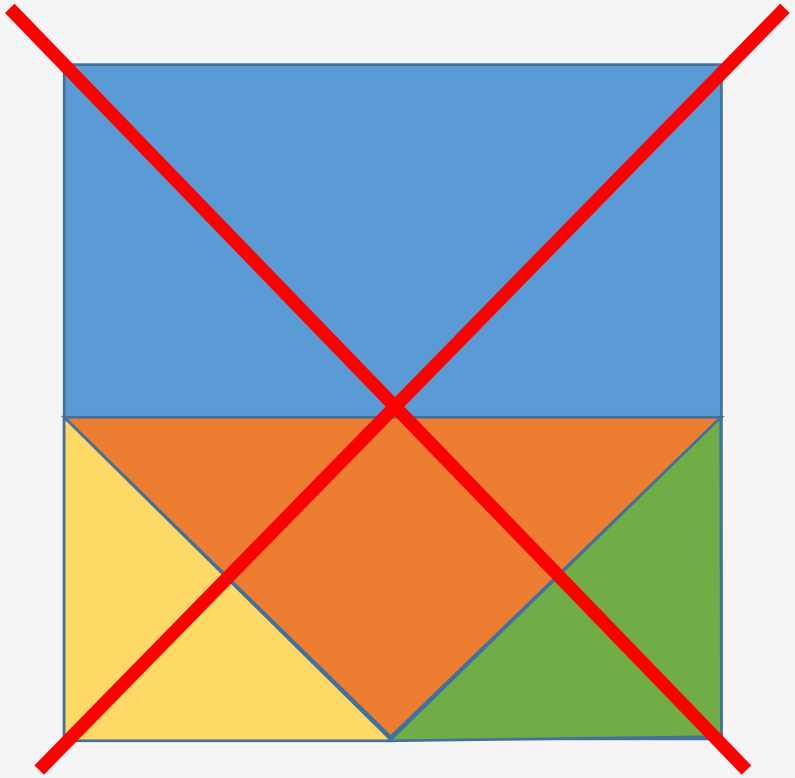
Mô hình thuật toán chia để trị (Divide and Conquer)

Chia để trị



Mô hình thuật toán chia để trị (Divide and Conquer)

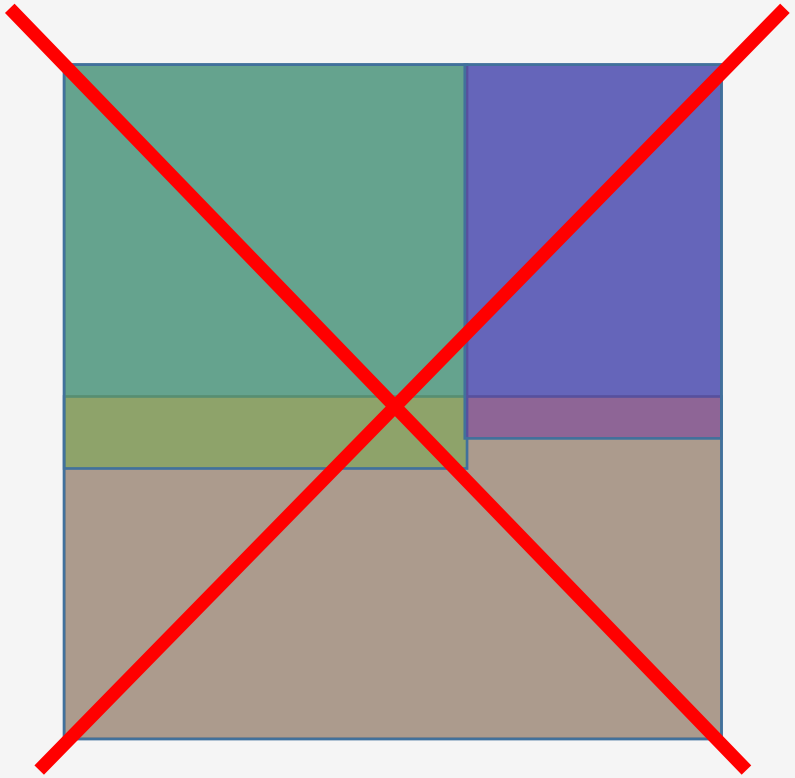
Chia để trị



Không đồng dạng

Mô hình thuật toán chia để trị (Divide and Conquer)

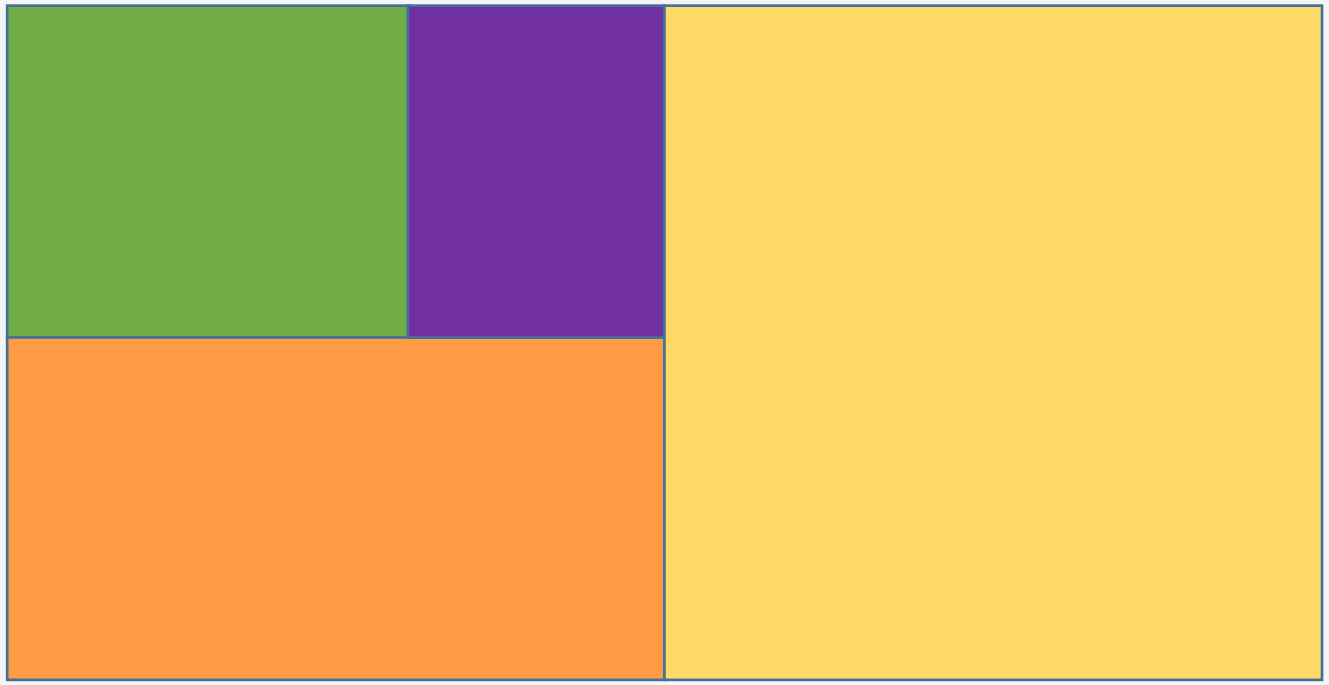
Chia để trị



Trùng lặp nhau

Mô hình thuật toán chia để trị (Divide and Conquer)

Chia để trị



Tổng hợp

Chia

Trị

Mô hình thuật toán chia để trị (Divide and Conquer)

Chia để trị

Áp dụng khi nào?

Bài toán có thể được chia thành các bài toán con giống bài toán gốc nhưng với kích thước nhỏ hơn

Ứng dụng trong xử lý song song ...

Một số thuật ngữ liên quan

Chia và trị (Divide and Conquer)

Giảm và trị (Decrease and Conquer)

Chặt nhị phân

Chặt tam phân

Mô hình thuật toán chia để trị (Divide and Conquer)

Chia để trị

```
void solve(int n) {
  if (n == 0)
     return;

  solve(n/2);
  solve(n/2);

  for (int i = 0; i < n; i++) {
     // some constant time operations
  }
}
```

Mô hình thuật toán chia để trị (Divide and Conquer)

Chia để trị

Gọi T(n) là độ phức tạp tính toán, n0 là trường hợp dừng.

Theo thuật toán tổng quát

$T(n) = O(1)$ nếu $n = n0$

$T(n) = aT(n/b) + D(n) + C(n)$ nếu $n > n0$

Trong đó

a: Số bài toán con

n/b: Kích thước bài toán con

D(n): Độ phức tạp thời gian của phần Divide

C(n): Độ phức tạp thời gian của phần Combine

Ví dụ với b=2.

Theo định lý Master: $T(n) = 2T(n/2) + n$ thì độ phức tạp sẽ là $O(n \log n)$

Mô hình thuật toán chia để trị (Divide and Conquer)

Các bài toán áp dụng chia để trị

- Tìm kiếm nhị phân
- Nhân số nguyên lớn
- Tính tổng dãy con liên tục lớn nhất
- Sắp xếp nhanh, sắp xếp trộn
- Tính lũy thừa
- Dãy xâu Fibonacci
- Tính số Fibonacci thứ N với N lớn

Mô hình thuật toán chia để trị (Divide and Conquer)

Tìm kiếm nhị phân

Bài toán: Cho số x và mảng A[] các số nguyên được sắp xếp theo thứ tự không giảm. Tìm i sao cho A[i] = x.

Phân tích bài toán:

Số x cho trước:

Hoặc là bằng phần tử nằm ở vị trí giữa mảng A

Hoặc là nằm ở nửa bên trái (x < phần tử ở giữa mảng A)

Hoặc là nằm ở nửa bên phải (x > phần tử ở giữa mảng A)

Mô hình thuật toán chia để trị (Divide and Conquer)

Tìm kiếm nhị phân

```
int BinarySearch(int A[], int n, int x) {
    int  low = 0;
    int  high = n - 1;
    int  mid = (low+high) / 2;
    while (low <= high) {
        if (x > A[mid] )
            low = mid + 1;
        else if (x < A[mid])
            high = mid -1;
        else
            return mid;
        mid = (low + high) / 2;
    }
    return -1;
}
```

Mô hình thuật toán chia để trị (Divide and Conquer)

Tìm kiếm nhị phân

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

$$T(n) = O(\log_2 n)$$

Mô hình thuật toán chia để trị (Divide and Conquer)

Nhân số nguyên lớn

Thuật toán cổ điển

Nhân hai số nguyên có n chữ số có độ phức tạp $O(n^2)$.

Cải tiến 1:

Rút gọn phép nhân hai số nguyên n chữ số còn bốn phép nhân hai số nguyên n/2

Cải tiến 2 (thuật toán Karatsuba)

Giảm xuống còn 3 phép nhân

Mô hình thuật toán chia để trị (Divide and Conquer)

Nhân số nguyên lớn

Input: $x = x_{n-1}x_{n-2}\ldots x_1x_0$ và $y = y_{n-1}y_{n-2}\cdots y_1y_0$

Output: $z = x * y = z_{2n-1}z_{2n-2}\ldots z_1z_0$

Phân tích:

$$x = \sum_{i=0}^{n-1} x_i * 10^i = x_{n-1} * 10^{n-1} + x_{n-2} * 10^{n-2} + \ldots + x_1 * 10^1 + x_0 * 10^0$$

$$y = \sum_{i=0}^{n-1} y_i * 10^i = y_{n-1} * 10^{n-1} + y_{n-2} * 10^{n-2} + \ldots + y_1 * 10^1 + y_0 * 10^0$$

$$\Rightarrow z = x * y = \sum_{i=0}^{2n-1} z_i * 10^i = (\sum_{i=0}^{n-1} x_i * 10^i) * (\sum_{i=0}^{n-1} y_i * 10^i)$$

Mô hình thuật toán chia để trị (Divide and Conquer)

Nhân số nguyên lớn – cải tiến 1

Đặt các biến phụ a,b,c,d:

$$a = x_{n-1}x_{n-2}\ldots x_{n/2} \qquad b = x_{(n/2)-1}x_{(n/2)-2}\ldots x_0$$

$$c = y_{n-1}y_{n-2}\ldots y_{n/2} \qquad d = y_{(n/2)-1}y_{(n/2)-2}\ldots y_0$$

Khi đó:

$$x = a*10^{n/2} + b \qquad y = c*10^{n/2} + d$$

$$\Rightarrow z = x*y = (a*10^{n/2}+b)(c*10^{n/2}+d) = (a*c)*10^{n} + (a*d+b*c)*10^{n/2} + (b*d)$$

Nhân số nguyên lớn – cải tiến 2

Cải tiến để còn lại 3 phép nhân

Đặt $U = a \times c; V = b \times d; W = (a+b) \times (c+d)$

$$\Rightarrow a \times d + b \times c = W - U - V$$

$$\Rightarrow Z = U \times 10^{n} + (W - U - V) \times 10^{n/2} + V$$

Mô hình thuật toán chia để trị (Divide and Conquer)

Nhân số nguyên lớn

```
Large_integer Karatsuba(x, y, n){
    If  n == 1  Return x*y
   Else  {
    a = x[n-1]. . . x[n/2]; b = x[n/2-1] . . . x[0];
    c = y[n-1]. . . y[n/2]; d = y[n/2-1] . . . y[0];
    U = Karatsuba(a, c, n/2);
    V = Karatsuba(b, d, n/2);
    W = Karatsuba(a+b, c+d, n/2);
    Return U*10n + (W-U-V)*10n/2 + V
   }
}
```

Mô hình thuật toán chia để trị (Divide and Conquer)

Nhân số nguyên lớn

Độ phức tạp thuật toán:

Gọi T(n) là thời gian cần thiết để thực hiện thuật toán. Khi đó:

$$T(n) = \begin{cases} 1 & n = 1 \\ 3T(\frac{n}{2}) + cn & n > 1 \end{cases}$$

$T(n) = O(n^{\log 3}) \approx O(n^{1.58})$

Mô hình thuật toán chia để trị (Divide and Conquer)

Tổng dãy con liên tục lớn nhất

Bài toán: Cho dãy số nguyên bao gồm cả số âm lẫn số dương. Nhiệm vụ của ta là tìm dãy con liên tục có tổng lớn nhất.

Ví dụ. Với dãy số A = {-2, -5, **6, -2, -3, 1, 5**, -6}

Tổng lớn nhất của dãy con liên tục ta nhận được là 7 tương ứng với dãy con {6, -2, -3, 1, 5}.

Mô hình thuật toán chia để trị (Divide and Conquer)

Tổng dãy con liên tục lớn nhất

```
Max = Arr[0];
for (i=1; i<n; i++) {
        S = 0;
        for ( j =i+1; j<=n; j++) {
                S = S + Arr[j];
                if (Max < S )
                        Max = S;
        }
}
Return(Max).
```

Mô hình thuật toán chia để trị (Divide and Conquer)

Tổng dãy con liên tục lớn nhất

Ý tưởng chia và trị

Với một vị trí mốc:

- Dãy con tổng lớn nhất có thể nằm hoàn toàn bên trái mốc.
- Dãy con tổng lớn nhất có thể nằm hoàn toàn bên phải mốc.
- Dãy con tổng lớn nhất có thể có một phần ở bên trái và một phần ở bên phải mốc.
- Thực hiện đệ quy để tính tổng bên trái và bên phải với mốc là vị trí ở giữa.

Kết quả là max của ba giá trị: đệ quy bên trái, đệ quy bên phải, hoặc ở cả hai bên.

Mô hình thuật toán chia để trị (Divide and Conquer)

Tổng dãy con liên tục lớn nhất

```
int maxCrossingSum(int arr[], int l, int m, int h) {
    int sum = 0; int left_sum = INT_MIN;
    for (int i = m; i >= l; i--) {
        sum = sum + arr[i];
        if (sum > left_sum)
            left_sum = sum;
    }
    sum = 0; int right_sum = INT_MIN;
    for (int i = m + 1; i <= h; i++) {
        sum = sum + arr[i];
        if (sum > right_sum)
            right_sum = sum;
    }

    return max(left_sum + right_sum, left_sum, right_sum);
}
```

Mô hình thuật toán chia để trị (Divide and Conquer)

Tổng dãy con liên tục lớn nhất

```
int maxSubArraySum(int arr[], int l, int h) {
    if (l == h) return arr[l];
    int m = (l + h)/2;
    return max(maxSubArraySum(arr, l, m),
        maxSubArraySum(arr, m+1, h),
        maxCrossingSum(arr, l, m, h));
}
```

Mô hình thuật toán chia để trị (Divide and Conquer)

Bài toán sắp xếp

Cho A[] là một mảng n phần tử. Hãy sắp xếp các phần tử của A theo thứ tự không giảm.

Giải thuật theo mô hình chia và trị (đã trình bày trong bài Sắp xếp):

MergeSort

QuickSort

Độ phức tạp chung: O(nlog(n))

Mô hình thuật toán chia để trị (Divide and Conquer)

Bài toán lũy thừa

Tính a^n

Điều kiện: a, n là các số nguyên và n không âm.

Thuật toán lặp:

```
int expose(a,n) {
        int result = 1;
        for (int i = 1; i <= n; ++i)
                result *= a;
}
```

Độ phức tạp O(n)

Mô hình thuật toán chia để trị (Divide and Conquer)

Bài toán lũy thừa

Cải tiến hàm *expose*

a^n tính theo $(a^{n/2})^2$ khi n chẵn.

$$a^n = \begin{cases} 1 & , n = 0 \\ (a^2)^{\lfloor \frac{n}{2} \rfloor}, & n\ chẵn \\ a(a^2)^{\lfloor \frac{n}{2} \rfloor}, & n\ lẻ \end{cases}$$

Mô hình thuật toán chia để trị (Divide and Conquer)

Bài toán lũy thừa

```
int power(int a, int n) {
        if (n == 0)  return 1;
        int x = power(a,n/2)
        if (n %2 == 0)
                return x*x;
        return a*x*x;
}
```

T(n) = O(logn).

Vấn đề: Kiểu dữ liệu -> Chia dư để tránh tràn số

Mô hình thuật toán chia để trị (Divide and Conquer)

Tính chất đồng dư

Với một giá trị m, các phép toán khi chia modulo cho m sẽ có tính chất sau

Phép cộng: (A+B) mod m = A mod m + B mod m

Phép nhân: (A*B) mod m = (A mod m * B mod m) mod m

Vì sao thường chọn cơ số m = 10^9+7?

Mô hình thuật toán chia để trị (Divide and Conquer)

Bài toán Fibonacci word

Dãy Fibonacci

F[1] = 1 F[2] = 1 F[n] = F[n-1] + F[n-2]

Các phần tử đầu tiên của dãy:1; 1; 2; 3; 5; 8; 13; 21; 34; 55; …

Bài toán Fibonacci Word

Cho trước hai xâu g1 và g2. Quy tắc tạo xâu tiếp theo như sau (với dấu + là nối xâu)

g(1) = A

g(2) = B

g(n) = g(n-2) + g(n-1)

Mô hình thuật toán chia để trị (Divide and Conquer)

Bài toán Fibonacci word

Các xâu đầu tiên trong dãy G

A
B
AB
BAB
ABBAB
BABABBAB
ABBABBABABBAB
BABABBABABBABBABABBAB

Độ dài của xâu thứ i chính là F[i].

Cho số tự nhiên N không quá 92 và vị trí K. Xác định ký tự thứ K trong xâu G(n)

Mô hình thuật toán chia để trị (Divide and Conquer)

Bài toán Fibonacci word

Ý tưởng Chia và Trị

Sinh dãy Fibonacci

So sánh vị trí K với F[n-2]

Nhỏ hơn hoặc bằng: gọi đệ quy để tìm bên trái

Lớn hơn: gọi để quy để tìm bên phải.

Thuật toán dừng khi gặp vị trí 1 hoặc 2 (đã biết giá trị)

Mô hình thuật toán chia để trị (Divide and Conquer)

Tính số Fibonacci thứ N

Sử dụng vòng lặp tính dãy Fibonacci

Độ phức tạp O(N)

Làm thế nào để tính F[N] với độ phức tạp O(logN)?

Ý tưởng

Nhận xét:

$$\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^n = \begin{pmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{pmatrix}.$$

Tính lũy thừa ma trận theo cách tính lũy thừa O(logN)

Mô hình thuật toán nhánh cận (Branch and Bound)

Bài toán tối ưu

Tìm min { f(X) : X ∈ D } hoặc tìm max { f(X) : X ∈ D }.

D là tập hữu hạn các phần tử thỏa mãn tính chất P nào đó.

D = { X =(x1, x2,..,xn) : X thỏa mãn tính chất P }

X ∈ D được gọi là một phương án cần duyệt.

Hàm f(X) được gọi là hàm mục tiêu của bài toán.

Miền D được gọi là tập phương án của bài toán.

X ∈ D thỏa mãn tính chất P được gọi là tập các ràng buộc của bài toán.

Mô hình thuật toán nhánh cận (Branch and Bound)

Thuật toán nhánh cận tổng quát

```
Branch_And_Bound (k) {
        for ak ∈ Ak do {
                if (<chấp nhận ak>){
                        xk = ak;
                        if ( k==n )
                                <Cập nhật kỷ lục>;
                        else if ( g(a1, a2,..,ak) ≤ f*)
                                Branch_And_Bound (k+1)  ;
                }
        }
}
```

Mô hình thuật toán nhánh cận (Branch and Bound)

Giải bài toán cái túi bằng thuật toán nhánh cận:

Tìm giá trị lớn nhất của hàm mục tiêu f(X) với X ∈ D. Trong đó, f(X) được xác định như dưới đây:

$$f^* = \max\left\{ f(X) = \sum_{i=1}^{n} c_i x_i : \sum_{i=1}^{n} a_i x_i \le b, x_i \in Z_+, i = 1,2,...,n \right\}$$

$$D = \left\{ X = (x_1, x_2, ..., x_n) : \sum_{i=1}^{n} a_i x_i \le b, x_i \in Z_+, i = 1,2,...,n \right\}$$

Mô hình thuật toán nhánh cận (Branch and Bound)

Giải bài toán cái túi bằng thuật toán nhánh cận:

Giả sử $\frac{c_1}{a_1} \geq \frac{c_2}{a_2} \geq \ldots \geq \frac{c_n}{a_n}$

Lặp trên các bài toán bộ phận cấp k =1, 2,..,n:

Giá trị sử dụng của k đồ vật trong túi: $\sigma_k = \sum_{i=1}^{k} c_k x_k$

Trọng lượng còn lại của túi: $b_k = b - \sum_{i=1}^{k} a_k x_k$

Cận trên của phương án bộ phận cấp k: $g(x_1, x_2, .., x_k) = \sigma_k - b_k . \frac{c_{k+1}}{a_{k+1}}$

Trả về phương án tối ưu.

Mô hình thuật toán nhánh cận (Branch and Bound)

Giải bài toán cái túi bằng thuật toán nhánh cận:

Thuật toán Branch_And_Bound (k) {

for (j = 1; j>=0; j--){

x[k] = j;

$\sigma_k = \sigma_k + c_i x_i;\ b_k = b_k + a_k x_k;$

If (k==n) <Ghi nhận kỷ lục>;

else if (σk + (c_{k+1}*b_k)/a_{k+1} >FOPT)

Branch_And_Bound(k+1);

$\sigma_k = \sigma_k - c_k x_k;\quad b_k = b_k - a_k x_k;$

}

}

Mô hình thuật toán nhánh cận (Branch and Bound)

Giải bài toán cái túi bằng thuật toán nhánh cận:

```
void Back_Track(int i){
    int j, t = ((b-weight)/A[i]);
    for(int j= t; j>=0; j--){
        X[i] = j;  weight = weight+A[i]*X[i];
        cost = cost + C[i]*X[i];
        if (i==n) Update();
        else if ( cost + ( C[i+1]*((b- weight)/A[i+1]))>FOPT)
            Back_Track(i+1);
        weight = weight-A[i]*X[i];
        cost = cost - C[i]*X[i];
    }
}
```