

CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT

CÁC LƯỢC ĐỒ THUẬT TOÁN QUAN
TRỌNG

Các lược đồ thuật toán quan trọng

- Độ quy
- Độ quy có nhớ
- Độ quy quay lui
- Nhánh và cận
- Tham lam
- Chia để trị
- Quy hoạch động

Đệ quy

- Một chương trình con (thủ tục/hàm) đưa ra lời gọi đến chính nó nhưng với dữ liệu đầu vào nhỏ hơn
- Tình huống cơ sở
 - Dữ liệu đầu vào nhỏ đủ để đưa ra kết quả một cách trực tiếp mà không cần đưa ra lời gọi đệ quy
- Tổng hợp kết quả
 - Kết quả của chương trình còn được xây dựng từ kết quả của lời gọi đệ quy và một số thông tin khác

Ví dụ: tổng $1 + 2 + \dots + n$

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

Đệ quy

- Ví dụ: hằng số tổ hợp

$$C(k, n) = \frac{n!}{k!(n-k)!}$$

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

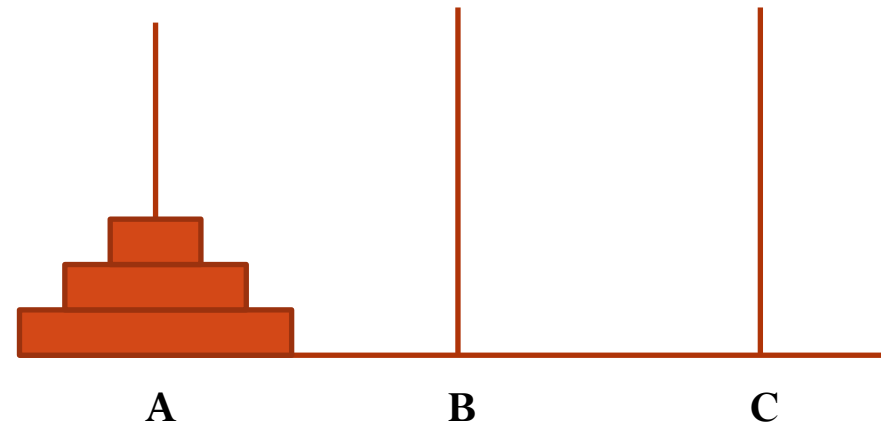
- Trường hợp cơ sở:

$$C(0, n) = C(n, n) = 1$$

```
int C(int k, int n) {  
    if (k == 0 || k == n)  
        return 1;  
    int C1 = C(k-1, n-1);  
    int C2 = C(k, n-1);  
    return C1 + C2;  
}
```

Đệ quy

- Bài toán tháp Hà Nội
 - Có n đĩa với kích thước khác nhau và 3 cọc A, B, C
 - Ban đầu n đĩa nằm ở cọc A theo thứ tự đĩa nhỏ nằm trên và đĩa lớn nằm dưới
 - Tìm cách chuyển n đĩa này từ cọc A sang cọc B, sử dụng cọc C làm trung gian theo nguyên tắc
 - Mỗi lần chỉ được chuyển 1 đĩa trên cùng từ 1 cọc sang cọc khác
 - Không được phép để xảy ra tình trạng đĩa to nằm bên trên đĩa nhỏ

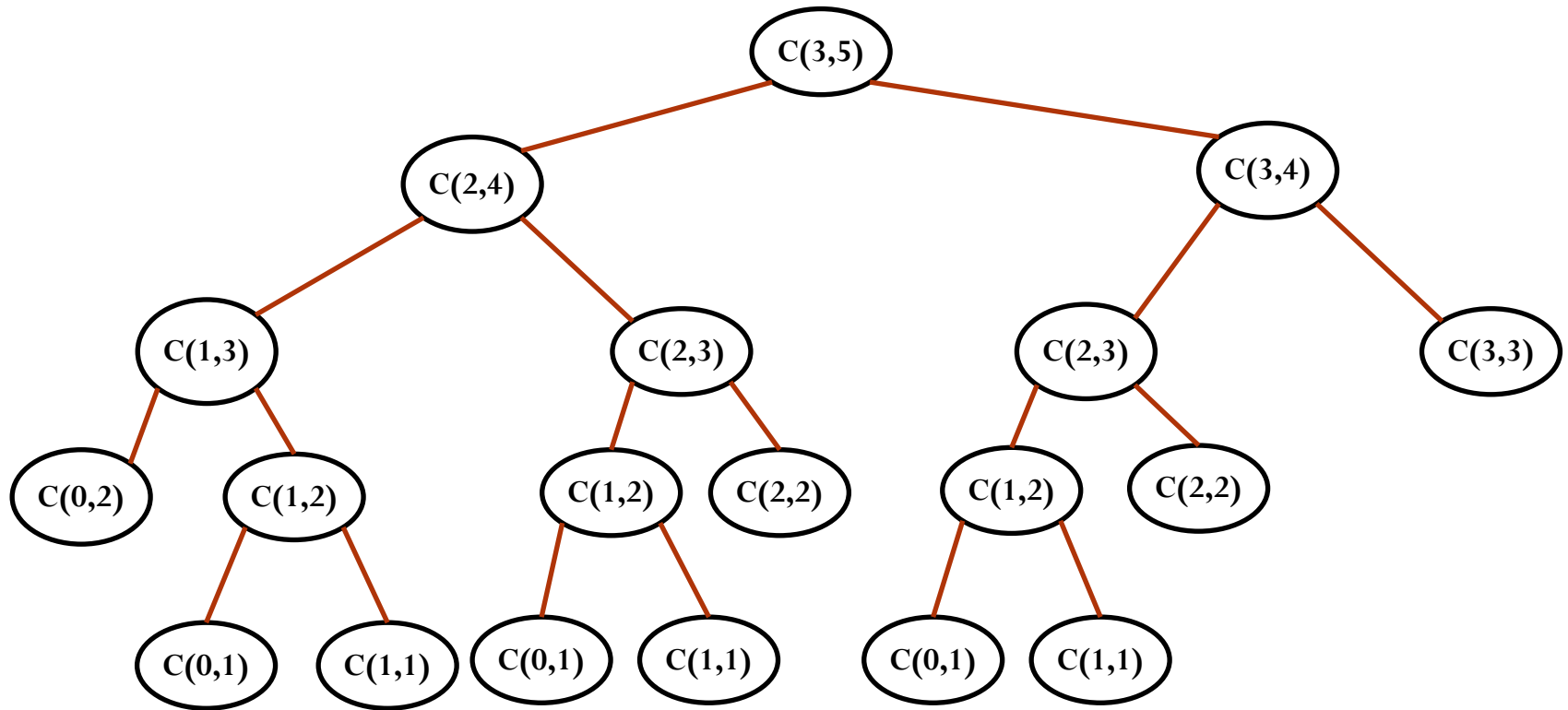


- Lời giải
- B1: $A \rightarrow B$
- B2: $A \rightarrow C$
- B3: $B \rightarrow C$
- B4: $A \rightarrow B$
- B5: $C \rightarrow A$
- B6: $C \rightarrow B$
- B7: $A \rightarrow B$

Đệ quy

```
void move(int n, char A, char B, char C) {  
    if(n == 1) {  
        printf("Move 1 disk from %c to %c", A, B)  
    }else{  
        move(n-1, A, C, B);  
        move(1, A, B, C);  
        move(n-1, C, B, A);  
    }  
}  
  
void main() {  
    int n = 3;  
    move(n, 'A', 'B', 'C');  
}
```

Đệ quy



Đệ quy có nhớ

- Khắc phục tình trạng một chương trình con với tham số xác định được gọi đệ quy nhiều lần
- Sử dụng bộ nhớ để lưu trữ kết quả của một chương trình con với tham số cố định
- Bộ nhớ được khởi tạo với giá trị đặc biệt để ghi nhận mỗi chương trình con chưa được gọi lần nào
- Địa chỉ bộ nhớ sẽ được ánh xạ với các giá trị tham số của chương trình con

```
int m[MAX][MAX];

int C(int k, int n) {
    if (k == 0 || k == n)
        m[k][n] = 1;
    if(m[k][n] < 0){
        m[k][n] = C(k-1,n-1) +
                  C(k,n-1);
    }
    return m[k][n];
}

int main() {
    for(int i = 0; i < MAX; i++)
        for(int j = 0; j < MAX; j++)
            m[i][j] = -1;
    printf("%d\n",C(16,32));
}
```


Đệ quy quay lui

- Áp dụng để giải các bài toán liệt kê, bài toán tối ưu tổ hợp
- $A = \{(x_1, x_2, \dots, x_n) \mid x_i \in A_i, \forall i = 1, \dots, n\}$
- Liệt kê tất cả các bộ $x \in A$ thoả mãn một thuộc tính P nào đó
- Thủ tục TRY(k):
 - Thử các giá trị v có thể gán cho x_k mà không vi phạm thuộc tính P
 - Với mỗi giá trị hợp lệ v :
 - Gán v cho x_k
 - Nếu $k < n$: gọi đệ quy TRY($k+1$) để thử tiếp giá trị cho x_{k+1}
 - Nếu $k = n$: ghi nhận cấu hình

Đệ quy quay lui

```
TRY( $k$ )
  Begin
    Foreach  $v$  thuộc  $A_k$ 
      if check( $v, k$ ) /* kiểm tra xem  $v$  có hợp lệ không */
        Begin
           $x_k = v$ ;
          if( $k = n$ ) ghi_nhan_cau_hinh;
          else TRY( $k+1$ );
        End
      End
    End
  End
Main()
  Begin
    TRY(1);
  End
```

Đệ quy quay lui: liệt kê xâu nhị phân

- Mô hình hoá cấu hình:
 - Mảng $x[n]$ trong đó $x[i] \in \{0,1\}$ là bit thứ i của xâu nhị phân
($i = 0, \dots, n-1$)

```
void printSolution(){
    for(int k = 0; k < n; k++)
        printf("%d",x[k]);
    printf("\n");
}

int TRY(int k) {
    for(int v = 0; v <= 1; v++){
        x[k] = v;
        if(k == n-1) printSolution();
        else TRY(k+1);
    }
}

int main() {
    TRY(0);
}
```

Đệ quy quay lui: liệt kê xâu nhị phân với ràng buộc

- Liệt kê các xâu nhị phân sao cho không có 2 bit 1 nào đứng cạnh nhau
- Mô hình hoá cấu hình:
 - Mảng $x[n]$ trong đó $x[i] \in \{0,1\}$ là bit thứ i của xâu nhị phân
($i = 1, \dots, n$)
 - Thuộc tính P : không có 2 bit 1 nào đứng cạnh nhau

```
int TRY(int k) {
    for(int v = 0; v <= 1; v++){
        if(x[k-1] + v < 2){
            x[k] = v;
            if(k == n)
                printSolution();
            else TRY(k+1);
        }
    }
}

int main() {
    x[0] = 0;
    TRY(1);
}
```

Đệ quy quay lui: liệt kê các tổ hợp

- Liệt kê các tổ hợp chập k của $1, 2, \dots, n$
- Mô hình hoá cấu hình:
 - Mảng $x[k]$ trong đó $x[i] \in \{1, \dots, n\}$ là phần tử thứ i của cấu hình tổ hợp ($i = 1, \dots, k$)
 - Thuộc tính P : $x[i] < x[i+1]$, với mọi $i = 1, 2, \dots, k-1$

```
int TRY(int i) {
    for(int v = x[i-1]+1; v <= n-k+i;
        v++){
        x[i] = v;
        if(i == k)
            printSolution();
        else TRY(i+1);
    }
}
```

```
int main() {
    x[0] = 0;
    TRY(1);
}
```

Đệ quy quay lui: liệt kê các hoán vị kỹ thuật đánh dấu

- Liệt kê các hoán vị của 1, 2, ..., n
- Mô hình hoá cấu hình:
 - Mảng $x[1, \dots, n]$ trong đó $x[i] \in \{1, \dots, n\}$ là phần tử thứ i của cấu hình hoán vị ($i = 1, \dots, n$)
 - Thuộc tính P :
 - $x[i] \neq x[j]$, với mọi $1 \leq i < j \leq n$
 - Mảng đánh dấu $m[v] = \text{true}$ (false) nếu giá trị v đã xuất hiện (chưa xuất hiện) trong cấu hình bộ phận, với mọi $v = 1, \dots, n$

```
void TRY(int i) {
    for(int v = 1; v <= n; v++){
        if(!m[v]) {
            x[i] = v;
            m[v] = true; // đánh dấu
            if(i == n)
                printSolution();
            else TRY(i+1);
            m[v] = false; // khôi phục
        }
    }
}

void main() {
    for(int v = 1; v <= n; v++)
        m[v] = false;
    TRY(1);
}
```

Đệ quy quay lui: bài toán xếp hậu

- Xếp n quân hậu trên một bàn cờ quốc tế sao cho không có 2 quân hậu nào ăn được nhau
- Mô hình hoá
 - $x[1, \dots, n]$ trong đó $x[i]$ là hàng của quân hậu xếp trên cột i , với mọi $i = 1, \dots, n$
 - Thuộc tính P
 - $x[i] \neq x[j]$, với mọi $1 \leq i < j \leq n$
 - $x[i] + i \neq x[j] + j$, với mọi $1 \leq i < j \leq n$
 - $x[i] - i \neq x[j] - j$, với mọi $1 \leq i < j \leq n$

	1	2	3	4
1		X		
2				X
3	X			
4			X	

Lời giải $x = (3, 1, 4, 2)$

Đệ quy quay lui: bài toán xếp hậu

```
int check(int v, int k) {  
    // kiểm tra xem v có thể gán được  
    // cho x[k] không  
    for(int i = 1; i <= k-1; i++) {  
        if(x[i] == v) return 0;  
        if(x[i] + i == v + k) return 0;  
        if(x[i] - i == v - k) return 0;  
    }  
    return 1;  
}
```

```
void TRY(int k) {  
    for(int v = 1; v <= n; v++) {  
        if(check(v,k)) {  
            x[k] = v;  
            if(k == n) printSolution();  
            else TRY(k+1);  
        }  
    }  
}  
  
void main() {  
    TRY(1);  
}
```


Đệ quy quay lui: bài toán sudoku

- Điền các chữ số từ 1 đến 9 vào các ô trong bảng vuông 9x9 sao cho trên mỗi hàng, mỗi cột và mỗi bảng vuông con 3x3 đều có mặt đầy đủ 1 chữ số từ 1 đến 9

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

Đệ quy quay lui: bài toán sudoku

- Mô hình hoá
 - Mảng 2 chiều $x[0..8, 0..8]$
 - Thuộc tính P
 - $x[i, j_2] \neq x[i, j_1]$, với mọi $i = 0, \dots, 8$, và $0 \leq j_1 < j_2 \leq 8$
 - $x[i_1, j] \neq x[i_2, j]$, với mọi $j = 0, \dots, 8$, và $0 \leq i_1 < i_2 \leq 8$
 - $x[3I+i_1, 3J+j_1] \neq x[3I+i_2, 3J+j_2]$, với mọi $I, J = 0, \dots, 2$, và $i_1, j_1, i_2, j_2 \in \{0, 1, 2\}$ sao cho $i_1 \neq i_2$ hoặc $j_1 \neq j_2$

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

Đệ quy quay lui: bài toán sudoku

- Thứ tự duyệt: từ ô (0,0), theo thứ tự từ trái qua phải và từ trên xuống dưới

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

Đệ quy quay lui: bài toán sudoku

```
bool check(int v, int r, int c){
    for(int i = 0; i <= r-1; i++)
        if(x[i][c] == v) return false;
    for(int j = 0; j <= c-1; j++)
        if(x[r][j] == v) return false;
    int I = r/3;
    int J = c/3;
    int i = r - 3*I;
    int j = c - 3*J;
    for(int i1 = 0; i1 <= i-1; i1++)
        for(int j1 = 0; j1 <= 2; j1++)
            if(x[3*I+i1][3*J+j1] == v)
                return false;
    for(int j1 = 0; j1 <= j-1; j1++)
        if(x[3*I+i][3*J+j1] == v)
            return false;
    return true;
}
```

```
void TRY(int r, int c){
    for(int v = 1; v <= 9; v++){
        if(check(v,r,c)){
            x[r][c] = v;
            if(r == 8 && c == 8){
                printSolution();
            }else{
                if(c == 8) TRY(r+1,0);
                else TRY(r,c+1);
            }
        }
    }
}

void main(){
    TRY(0,0);
}
```

Đệ quy quay lui: bài tập

Cho số nguyên dương M , N và N số nguyên dương A_1, A_2, \dots, A_N . Liệt kê các nghiệm nguyên dương của phương trình

$$A_1X_1 + A_2X_2 + \dots + A_NX_N = M$$

Đệ quy quay lui: bài tập

1. Liệt kê tất cả các cách chọn ra k phần tử từ $1, 2, \dots, n$ sao cho không có 2 phần tử nào đứng cạnh nhau cũng được chọn
2. Liệt kê tất cả các cách chọn ra k phần tử từ $1, 2, \dots, n$ sao cho không có 3 phần tử nào đứng cạnh nhau cùng đồng thời được chọn
3. Liệt kê tất cả các xâu nhị phân độ dài n trong đó không có 3 bit 1 nào đứng cạnh nhau
4. Liệt kê tất cả các cách phân tích số N thành tổng của các số nguyên dương
5. Giải bài toán Sudoku, xếp hậu sử dụng kỹ thuật đánh dấu
6. Giải bài toán Sudoku với 1 số ô đã được điền từ trước

Thuật toán nhánh và cận

- Bài toán tối ưu tổ hợp
 - Phương án $x = (x_1, x_2, \dots, x_n)$ trong đó $x_i \in A_i$ cho trước
 - Phương án thoả mãn ràng buộc C
 - Hàm mục tiêu $f(x) \rightarrow \min (\max)$

Thuật toán nhánh và cận

- Bài toán người du lịch
 - Có n thành phố $1, 2, \dots, n$. Chi phí đi từ thành phố i đến thành phố j là $c(i, j)$. Hãy tìm một hành trình xuất phát từ thành phố thứ 1, đi qua các thành phố khác, mỗi thành phố đúng 1 lần và quay về thành phố 1 với tổng chi phí nhỏ nhất
- Mô hình hoá
 - Phương án $x = (x_1, x_2, \dots, x_n)$ trong đó $x_i \in \{1, 2, \dots, n\}$
 - Ràng buộc C : $x_i \neq x_j, \forall 1 \leq i < j \leq n$
 - Hàm mục tiêu

$$f(x) = c(x_1, x_2) + c(x_2, x_3) + \dots + c(x_n, x_1) \rightarrow \min$$

Thuật toán nhánh và cận

- Duyệt toàn bộ:
 - Liệt kê tất cả các phương án bằng phương pháp đệ quy quay lui
 - Với mỗi phương án, tính toán hàm mục tiêu
 - Giữ lại phương án có hàm mục tiêu nhỏ nhất

```
TRY( $k$ )
  Begin
    Foreach  $v$  thuộc  $A_k$ 
      if check( $v, k$ )
        Begin
           $x_k = v$ ;
          if( $k = n$ )
            ghi_nhan_cau_hinh;
            cập nhật kỷ lục  $f^*$ ;
          else TRY( $k+1$ );
        End
      End
    End
Main()
  Begin
    TRY(1);
  End
```

Thuật toán nhánh và cận

- Duyệt nhánh và cận:
 - Phương án bộ phận (a_1, \dots, a_k) trong đó a_1 gán cho x_1, \dots, a_k gán cho x_k
 - Phương án $(a_1, \dots, a_k, b_{k+1}, \dots, b_n)$ là một phương án đầy đủ được phát triển từ (a_1, \dots, a_k) trong đó b_{k+1} gán cho x_{k+1}, \dots, b_n được gán cho x_n
 - Với mỗi phương án bộ phận (x_1, \dots, x_k) , hàm cận dưới $g(x_1, \dots, x_k)$ có giá trị không lớn hơn giá trị hàm mục tiêu của phương án đầy đủ phát triển từ (x_1, \dots, x_k)
 - Nếu $g(x_1, \dots, x_k) \geq f^*$ thì không phát triển lời giải từ (x_1, \dots, x_k)

```
TRY(k) {  
    Foreach v thuộc  $A_k$   
        if check(v,k) {  
             $x_k = v$ ;  
            if(k = n) {  
                ghi_nhan_cau_hinh;  
                cập nhật kỷ lục  $f^*$ ;  
            } {  
                if  $g(x_1, \dots, x_k) < f^*$   
                    TRY(k+1);  
            }  
        }  
    }  
Main()  
{  $f^* = \infty$ ;  
    TRY(1);  
}
```

Thuật toán nhánh và cận giải bài toán người du lịch

- c_m là chi phí nhỏ nhất trong số các chi phí đi giữa 2 thành phố khác nhau
- Phương án bộ phận (x_1, \dots, x_k)
 - Chi phí bộ phận $f = c(x_1, x_2) + c(x_2, x_3) + \dots + c(x_{k-1}, x_k)$
 - Hàm cận dưới
$$g(x_1, \dots, x_k) = f + c_m \times (n - k + 1)$$

Thuật toán nhánh và cận giải bài toán người du lịch

```
void TRY(int k){
    for(int v = 1; v <= n; v++){
        if(marked[v] == false){
            a[k] = v;
            f = f + c[a[k-1]][a[k]];
            marked[v] = true;
            if(k == n){
                process();
            }else{
                int g = f + cmin*(n-k+1);
                if(g < f_min)
                    TRY(k+1);
            }
            marked[v] = false;
            f = f - c[a[k-1]][a[k]];
        }
    }
}
```

```
void process() {
    if(f + c[x[n]][x[1]] < f_min){
        f_min = f + c[x[n]][x[1]];
    }
}

void main() {
    f_min = 9999999999;
    for(int v = 1; v <= n; v++){
        marked[v] = false;
    }
    x[1] = 1; marked[1] = true;
    TRY(2);
}
```

Thuật toán tham lam

- Được ứng dụng để giải một số bài toán tối ưu tổ hợp
- Đơn giản và tự nhiên
- Quá trình tìm lời giải diễn ra qua các bước
- Tại mỗi bước, ra quyết định dựa trên các thông tin hiện tại mà không quan tâm đến ảnh hưởng của nó trong tương lai
- Dễ đề xuất, cài đặt
- Thường không tìm được phương án tối ưu toàn cục

Thuật toán tham lam

- Lời giải được biểu diễn bởi tập S
- C biểu diễn các ứng cử viên
- $\text{select}(C)$: chọn ra ứng cử viên tiềm năng nhất
- $\text{solution}(S)$: trả về true nếu S là lời giải của bài toán
- $\text{feasible}(S)$: trả về true nếu S không vi phạm ràng buộc nào của bài toán

```
Greedy() {  
    S = {};  
    while C  $\neq$   $\emptyset$  and  
        not solution(S){  
        x = select(C);  
        C = C \ {x};  
        if feasible(S  $\cup$  {x}) {  
            S = S  $\cup$  {x};  
        }  
    }  
    return S;  
}
```

Thuật toán tham lam

- Bài toán tập đoạn không giao nhau
 - Đầu vào: cho tập các đoạn thẳng $X = \{(a_1, b_1), \dots, (a_n, b_n)\}$ trong đó $a_i < b_i$ là tọa độ đầu mút của đoạn thứ i trên đường thẳng, với mọi $i = 1, \dots, n$.
 - Đầu ra: Tìm tập con các đoạn đôi một không giao nhau có số đoạn lớn nhất

Thuật toán tham lam

- Tham lam 1
 - Sắp xếp các đoạn theo thứ tự không giảm của a_i được danh sách L
 - Lặp lại thao tác sau cho đến khi L không còn phần tử nào
 - Chọn đoạn (a_c, b_c) đầu tiên trong L và loại nó ra khỏi L
 - Nếu (a_c, b_c) không có giao nhau với đoạn nào trong lời giải thì đưa (a_c, b_c) vào lời giải

```
Greedy1() {  
    S = {};  
    L = Sắp xếp các đoạn trong X  
        theo thứ tự không giảm của  $a_i$ ;  
    while (X  $\neq \emptyset$ ) {  
         $(a_c, b_c)$  = chọn đoạn đầu tiên  
            trong L;  
        Loại bỏ  $(a_c, b_c)$  khỏi L;  
        if feasible( $S \cup \{(a_c, b_c)\}$ ) {  
            S = S  $\cup \{(a_c, b_c)\}$ ;  
        }  
    }  
    return S;  
}
```


Thuật toán tham lam

- Tham lam 1 không đảm bảo cho lời giải tối ưu, ví dụ

$$X = \{(1,11), (2,5), (6,10)\}$$

- Greedy 1 sẽ cho lời giải là $\{(1,11)\}$ trong khi lời giải tối ưu là $\{(2,5), (6,10)\}$

Thuật toán tham lam

- Tham lam 2
 - Sắp xếp các đoạn theo thứ tự không giảm của độ dài b_i - a_i được danh sách L
 - Lặp lại thao tác sau cho đến khi L không còn phần tử nào
 - Chọn đoạn (a_c, b_c) đầu tiên trong L và loại nó ra khỏi L
 - Nếu (a_c, b_c) không có giao nhau với đoạn nào trong lời giải thì đưa (a_c, b_c) vào lời giải

```
Greedy2() {  
    S = {};  
    L = Sắp xếp các đoạn trong X  
        theo thứ tự không giảm của  
         $b_i - a_i$ ;  
    while (X  $\neq \emptyset$ ) {  
         $(a_c, b_c)$  = chọn đoạn đầu tiên  
            trong L;  
        Loại bỏ  $(a_c, b_c)$  khỏi L;  
        if feasible( $S \cup \{(a_c, b_c)\}$ ) {  
             $S = S \cup \{(a_c, b_c)\}$ ;  
        }  
    }  
    return S;  
}
```

Thuật toán tham lam

- Tham lam 2 không đảm bảo cho lời giải tối ưu, ví dụ

$$X = \{(1,5), (4,7), (6,11)\}$$

- Greedy 1 sẽ cho lời giải là $\{(4,7)\}$ trong khi lời giải tối ưu là $\{(1,5), (6,11)\}$

Thuật toán tham lam

- Tham lam 3
 - Sắp xếp các đoạn theo thứ tự không giảm của b_i được danh sách L
 - Lặp lại thao tác sau cho đến khi L không còn phần tử nào
 - Chọn đoạn (a_c, b_c) đầu tiên trong L và loại nó ra khỏi L
 - Nếu (a_c, b_c) không có giao nhau với đoạn nào trong lời giải thì đưa (a_c, b_c) vào lời giải
- Tham lam 3 đảm bảo cho lời giải tối ưu

```
Greedy3() {  
    S = {};  
    L = Sắp xếp các đoạn trong X  
        theo thứ tự không giảm của  $b_i$ ;  
    while (X  $\neq \emptyset$ ) {  
         $(a_c, b_c)$  = chọn đoạn đầu tiên  
            trong L;  
        Loại bỏ  $(a_c, b_c)$  khỏi L;  
        if feasible( $S \cup \{(a_c, b_c)\}$ ) {  
            S = S  $\cup \{(a_c, b_c)\}$ ;  
        }  
    }  
    return S;  
}
```

Thuật toán chia để trị

- Sơ đồ chung
 - Chia bài toán xuất phát thành các bài toán con độc lập nhau
 - Giải các bài toán con (đệ quy)
 - Tổng hợp lời giải của các bài toán con để xây dựng lời giải của bài toán xuất phát

Thuật toán chia để trị

- Bài toán tìm kiếm nhị phân: cho dãy $x[1..n]$ được sắp xếp theo thứ tự tăng dần và 1 giá trị y . Tìm chỉ số i sao cho $x[i] = y$

```
bSearch(x, start, finish, y) {  
    if(start == finish) {  
        if(x[start] == y)  
            return start;  
        else return -1;  
    }else{  
        m = (start + finish)/2;  
        if(x[m] == y) return m;  
        if(x[m] < y)  
            return bSearch(x, m+1,finish,y);  
        else  
            return bSearch(x,start,m-1,y);  
    }  
}
```

Thuật toán chia để trị

- Bài toán dãy con cực đại
 - Đầu vào: dãy số nguyên a_1, a_2, \dots, a_n
 - Đầu ra: tìm dãy con bao gồm các phần tử liên tiếp của dãy đã cho có tổng cực đại

```
int maxSeq(int* a, int l, int r){  
    if(l == r) return a[l];  
    int max;  
    int mid = (l+r)/2;  
    int mL = maxSeq(a,l,mid);  
    int mR = maxSeq(a,mid+1,r);  
    int mLR = maxLeft(a,l,mid) +  
               maxRight(a,mid+1,r);  
  
    max = mL;  
    if(max < mR) max = mR;  
    if(max < mLR) max = mLR;  
    return max;  
}
```

Thuật toán chia để trị

```
int maxLeft(int* a, int l, int r){
    int max = -99999999;
    int s = 0;
    for(int i = r; i >= l; i--){
        s += a[i];
        if(s > max) max = s;
    }
    return max;
}
```

```
int maxRight(int* a, int l, int r){
    int max = -99999999;
    int s = 0;
    for(int i = l; i <= r; i++){
        s += a[i];
        if(s > max) max = s;
    }
    return max;
}
```

```
void main() {
    readData();
    int rs = maxSeq(a,0,n-1);
    printf("result = %d",rs);
}
```


Thuật toán chia để trị (định lý thợ)

- Chia bài toán xuất phát thành a bài toán con, mỗi bài toán con kích thước n/b
- $T(n)$: thời gian của bài toán kích thước n
- Thời gian phân chia (dòng 4): $D(n)$
- Thời gian tổng hợp lời giải (dòng 6): $C(n)$
- Công thức truy hồi: $\boxed{?}$

$$T(n) = \begin{cases} \Theta(1), & n \leq n_0 \\ aT\left(\frac{n}{b}\right) + C(n) + D(n), & n > n_0 \end{cases}$$

```
procedure D-and-C( $n$ ) {  
  1.  if( $n \leq n_0$ )  
  2.    xử lý trực tiếp  
  3.  else{  
  4.    chia bài toán xuất phát  
    thành  $a$  bài toán con kích thước  
     $n/b$   
  5.    gọi đệ quy  $a$  bài toán con  
  6.    tổng hợp lời giải  
  7.  }  
}
```

Thuật toán chia để trị

- Độ phức tạp của thuật toán chia để trị (định lý thợ)
- Công thức truy hồi:

$$T(n) = aT(n/b) + cn^k, \text{ với các hằng số } a \geq 1, b > 1, c > 0$$

- Nếu $a > b^k$ thì $T(n) = \Theta(n^{\log_b a})$
- Nếu $a = b^k$ thì $T(n) = \Theta(n^k \log n)$ với $\log n = \log_2 n$
- Nếu $a < b^k$ thì $T(n) = \Theta(n^k)$

Thuật toán quy hoạch động

- Sơ đồ chung
 - Chia bài toán xuất phát thành các bài toán con không nhất thiết độc lập với nhau
 - Giải các bài toán con từ nhỏ đến lớn, lời giải được lưu trữ lại vào 1 bảng
 - Bài toán con nhỏ nhất phải được giải 1 cách trực tiếp
 - Xây dựng lời giải của bài toán lớn hơn từ lời giải đã có của các bài toán con nhỏ hơn (truy hồi)
 - Số lượng bài toán con cần được bị chặn bởi đa thức của kích thước dữ liệu đầu vào
 - Phù hợp để giải hiệu quả một số bài toán tối ưu tổ hợp

Thuật toán quy hoạch động

- Bài toán dãy con cực đại
 - Đầu vào: dãy số nguyên a_1, a_2, \dots, a_n
 - Đầu ra: tìm dãy con bao gồm các phần tử liên tiếp của dãy đã cho có tổng cực đại
- Phân chia
 - Ký hiệu P_i là bài toán tìm dãy con bao gồm các phần tử liên tiếp có tổng cực đại mà phần tử cuối cùng là a_i , với mọi $i = 1, \dots, n$
 - Ký hiệu S_i là tổng các phần tử của lời giải của P_i , $\forall i = 1, \dots, n$
 - $S_1 = a_1$
 - $$S_i = \begin{cases} S_{i-1} + a_i, & \text{nếu } S_{i-1} > 0 \\ a_i, & \text{nếu } S_{i-1} \leq 0 \end{cases}$$
 - Tổng các phần tử của dãy con cực đại của bài toán xuất phát là $\max\{S_1, S_2, \dots, S_n\}$

Thuật toán quy hoạch động

- Bài toán dãy con tăng dần cực đại
 - Đầu vào: dãy số nguyên $a = a_1, a_2, \dots, a_n$ (gồm các phần tử đôi một khác nhau)
 - Đầu ra: tìm dãy con (bằng cách loại bỏ 1 số phần tử) của dãy đã cho tăng dần có số lượng phần tử là lớn nhất (gọi là dãy con cực đại)

Thuật toán quy hoạch động

- Phân chia
 - Ký hiệu P_i là bài toán tìm dãy con cực đại mà phần tử cuối cùng là a_i , với mọi $i = 1, \dots, n$
 - Ký hiệu S_i là số phần tử của lời giải của P_i , $\forall i = 1, \dots, n$
 - $S_1 = 1$
 - $S_i = \max\{1, \max\{S_j + 1 / j < i \wedge a_j < a_i\}\}$
 - Số phần tử của dãy con cực đại của bài toán xuất phát là
 $\max\{S_1, S_2, \dots, S_n\}$

```
void solve(){
    S[0] = 1;
    rs = S[0];
    for(int i = 1; i < n; i++){
        S[i] = 1;
        for(int j = i-1; j >= 0; j--){
            if(a[i] > a[j]){
                if(S[j] + 1 > S[i])
                    S[i] = S[j] + 1;
            }
        }
        rs = S[i] > rs ? S[i] : rs;
    }
    printf("rs = %d\n",rs);
}
```

Thuật toán quy hoạch động

- Bài toán dãy con chung dài nhất
 - Ký hiệu $X = \langle X_1, X_2, \dots, X_n \rangle$, một dãy con của X là dãy được tạo ra bằng việc loại bỏ 1 số phần tử nào đó của X đi
 - Đầu vào
 - Cho 2 dãy $X = \langle X_1, X_2, \dots, X_n \rangle$ và $Y = \langle Y_1, Y_2, \dots, Y_m \rangle$
 - Đầu ra
 - Tìm dãy con chung của X và Y có độ dài lớn nhất

Thuật toán quy hoạch động

- Bài toán dãy con chung dài nhất

- Phân rã

- Ký hiệu $S(i, j)$ là độ dài dãy con chung dài nhất của dãy $\langle X_1, \dots, X_i \rangle$ và $\langle Y_1, \dots, Y_j \rangle$, với $\forall i = 1, \dots, n$ và $j = 1, \dots, m$

- Bài toán con nhỏ nhất

- $\forall j = 1, \dots, m: S(1, j) = \begin{cases} 1, \text{ nếu } X_1 \text{ xuất hiện trong } Y_1, \dots, Y_j \\ 0, \text{ ngược lại} \end{cases}$
 - $\forall i = 1, \dots, n: S(i, 1) = \begin{cases} 1, \text{ nếu } Y_1 \text{ xuất hiện trong } X_1, \dots, X_i \\ 0, \text{ ngược lại} \end{cases}$

- Tổng hợp lời giải

$$S(i, j) = \begin{cases} S(i-1, j-1) + 1, & \text{nếu } X_i = Y_j \\ \max\{S(i-1, j), S(i, j-1)\} \end{cases}$$

Thuật toán quy hoạch động

X	3	7	2	5	1	4	9
---	---	---	---	---	---	---	---

Y	4	3	2	3	6	1	5	4	9	7
---	---	---	---	---	---	---	---	---	---	---

	1	2	3	4	5	6	7	8	9	10
1	0	1	1	1	1	1	1	1	1	1
2	0	1	1	1	1	1	1	1	1	2
3	0	1	2	2	2	2	2	2	2	2
4	0	1	2	2	2	2	3	3	3	3
5	0	1	2	2	2	3	3	3	3	3
6	1	1	2	2	2	3	3	4	4	4
7	1	1	2	2	2	3	3	4	5	5

Thuật toán quy hoạch động

```
void solve(){
    rs = 0;
    for(int i = 0; i <= n; i++) S[i][0] = 0;
    for(int j = 0; j <= m; j++) S[0][j] = 0;

    for(int i = 1; i <= n; i++){
        for(int j = 1; j <= m; j++){
            if(X[i] == Y[j]) S[i][j] = S[i-1][j-1] + 1;
            else{
                S[i][j] = S[i-1][j] > S[i][j-1] ?
                    S[i-1][j] : S[i][j-1];
            }
            rs = S[i][j] > rs ? S[i][j] : rs;
        }
    }
    printf("result = %d\n",rs);
}
```

Thuật toán quy hoạch động – Bài tập

- Cho dãy $a = a_1, a_2, \dots, a_n$. Một dãy con của dãy a là một dãy thu được bằng cách loại bỏ 1 số phần tử khỏi a . Tìm dãy con của a là một cấp số cộng với bước nhảy bằng 1 có độ dài lớn nhất