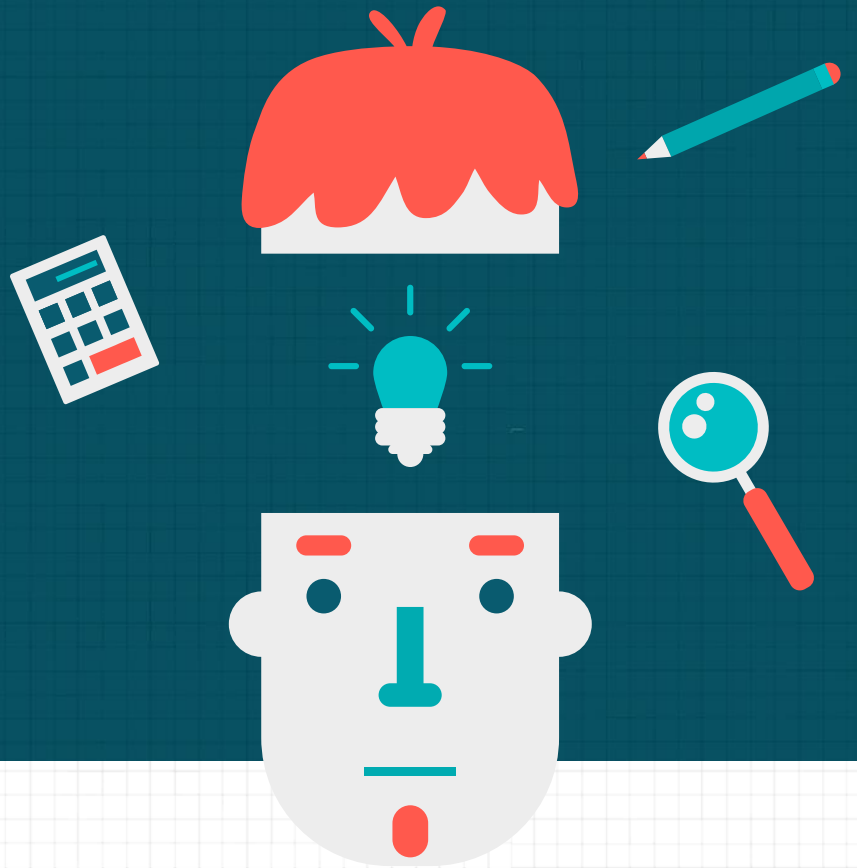
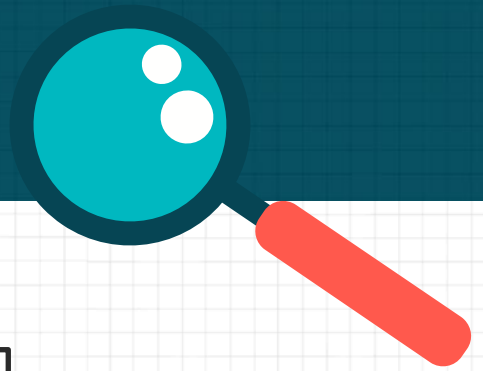
Bài 5
Thuật toán

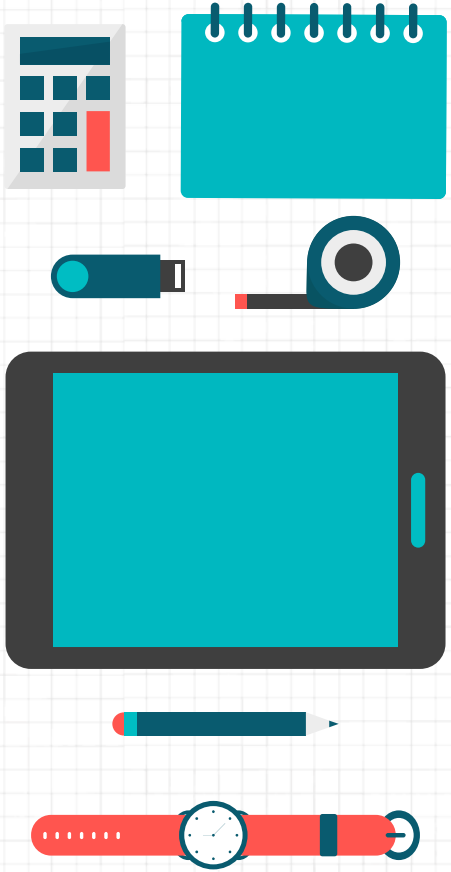


Ths. Phạm Minh Hoàng

Nội dung



- Tìm kiếm
- Sắp xếp
- Quy hoạch động



01

Tìm kiếm

Tìm kiếm

- Tìm kiếm là quá trình tìm một phần tử trong tập hợp rất nhiều phần tử dựa theo yêu cầu nào đó
- Hai phương pháp tìm kiếm
  - Tìm kiếm tuần tự
  - Tìm kiếm nhị phân

Tìm kiếm tuần tự

- Ý tưởng
  - Tìm lần lượt từng phần tử trong mảng
  - Nếu tìm thấy trả về vị trí tìm thấy trong mảng
  - Ngược lại trả về -1

```
int linearSearch (int A[], int N, int x){
    int i = 0;
    while (i < n && A[i] != x)
        i++;
    if (i == n)
        return -1;
    return i;
}
```

Tìm kiếm nhị phân

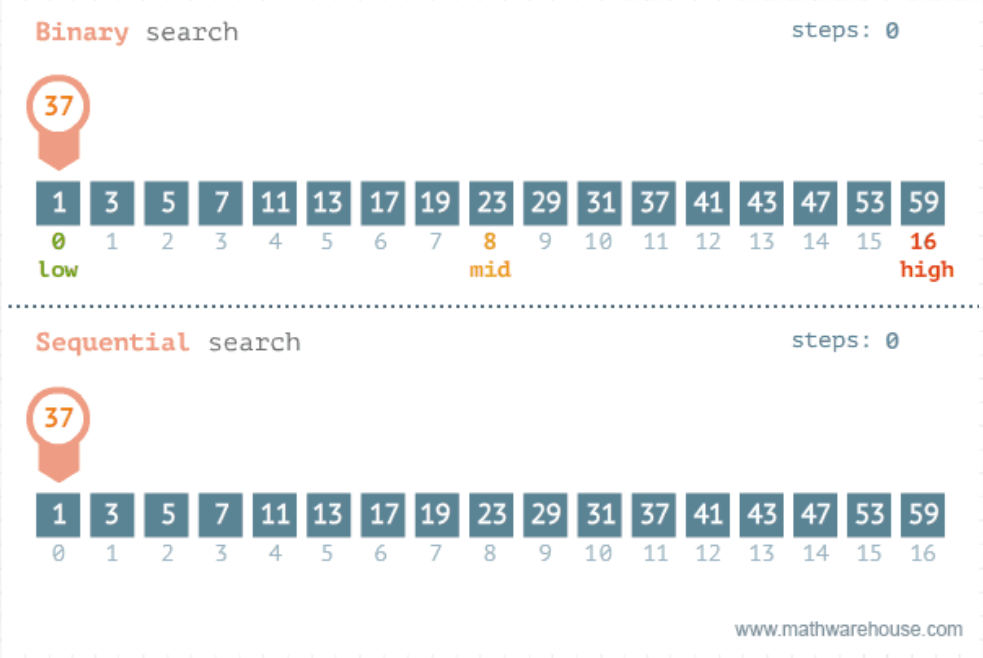
- Ý tưởng
  - Nếu mảng chứa các phần tử đã được sắp xếp
  - Chia mảng thành các phần: nửa bên trái, nửa bên phải, và phần tử chính giữa
  - Nếu phần tử cần tìm x trùng phần tử chính giữa → tìm thấy
  - Nếu x < phần tử chính giữa → chỉ cần tìm nửa mảng bên trái
  - Nếu x > phần tử chính giữa → chỉ cần tìm nửa mảng bên phải
  - Lặp lại quá trình tìm kiếm ở nửa mảng bên trái hoặc bên phải

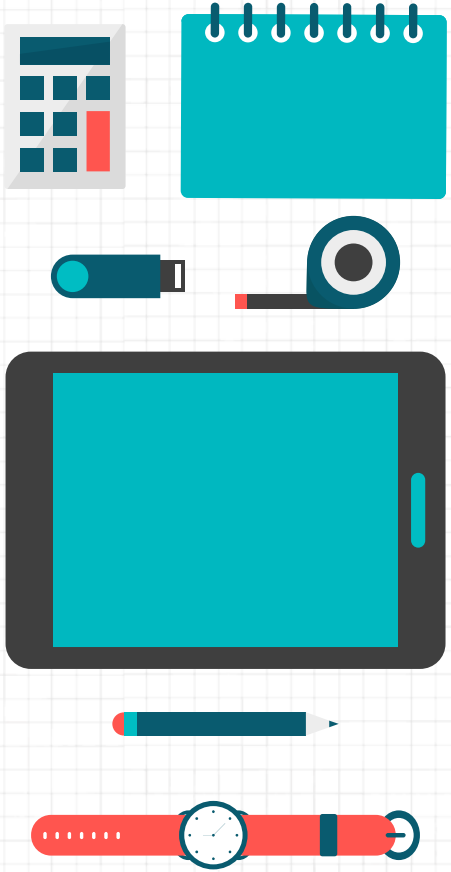
Tìm kiếm nhị phân

```
int binarySearch (int A[], int l, int r, int x){
    while (l <= r){
        int mid = (l + r)/2;
        if(A[mid] == x)
            return mid;
        if(x < A[mid])
            r = mid - 1;
        else
            l = mid + 1;
    }
    return -1;
}
```

Tìm tuần tự vs Tìm nhị phân

- Tìm tuần tự: độ phức tạp tuyến tính O(n)
- Tìm nhị phân: độ phức tạp O(logn) nhưng phải tốn chi phí sắp xếp mảng





02

Sắp xếp

Sắp xếp

- Sắp xếp là quá trình xử lý một danh sách các phần tử để đặt chúng theo một thứ tự theo yêu cầu nào đó

- Ví dụ: danh sách trước khi sắp xếp:
  {1, 25, 6, 5, 2, 37, 40}
  Danh sách sau khi sắp xếp:
  {1, 2, 5, 6, 25, 37, 40}

- Khi khảo sát các bài toán sắp xếp, ta phải làm việc nhiều với khái niệm nghịch thế

Sắp xếp

- Xét một mảng các số $a_0, a_1, \ldots, a_n$
  Nếu $i<j$ và $a_i > a_j$ thì ta gọi đó là một nghịch thế
- Nhận xét:
  - Mảng chưa sắp xếp sẽ có nhiều nghịch thế
  - Mảng đã sắp xếp không chứa nghịch thế
  - Việc sắp xếp mảng là nhằm tìm cách giảm số nghịch thế bằng cách hoán vị các cặp nghịch thế (a_i, a_j)

Sắp xếp

- Nhóm thuật toán có độ phức tạp bình phương $O(n^2)$
  - Sắp xếp chọn (Selection sort)
  - Sắp xếp chèn (Insertion sort)
  - Sắp xếp đổi chỗ (Interchange sort)
- Nhóm thuật toán có độ phức tạp O(nlogn)
  - Sắp xếp trộn (Merge sort)
  - Sắp xếp nhanh (Quick sort)

Sắp xếp đổi chỗ (interchange sort)

- Ý tưởng
  - Duyệt tất cả các phần tử
  - Tìm các cặp nghịch thế (a_i, a_j)
  - Đổi chỗ nếu phát hiện nghịch thế

```
void interchangeSort(int A[], int N){
    for(int i = 0; i < N; i++){
        for(int j = i+1; j < N; j++){
            if(a[i] > a[j])
                swap(a[i], a[j]);
        }
    }
}
```

Sắp xếp chọn (selection sort)

- Ý tưởng
  - Tìm phần tử nhỏ nhất đưa lên đầu mảng
  - Lặp lại với các phần tử còn lại

```
void selectionSort(int A[], int N){
   for (int i = 0; i < N - 1; i++){
      int min = i;
      for (int j = i + 1; j < N; j++){
         if (a[j] < a[min])
            min = j;
      }
      swap(a[i], a[min]);
}
```

Sắp xếp chèn (insertion sort)

- Ý tưởng
  - Chia mảng thành 2 dãy con: bên trái và bên phải
  - Ban đầu xem dãy con bên trái gồm k = 1 phần tử, dãy bên phải gồm N – 1 phần tử
  - Duyệt từng phần tử của mảng bên phải và chèn vào mảng bên trái sao cho mảng bên trái luôn có thứ tự tăng

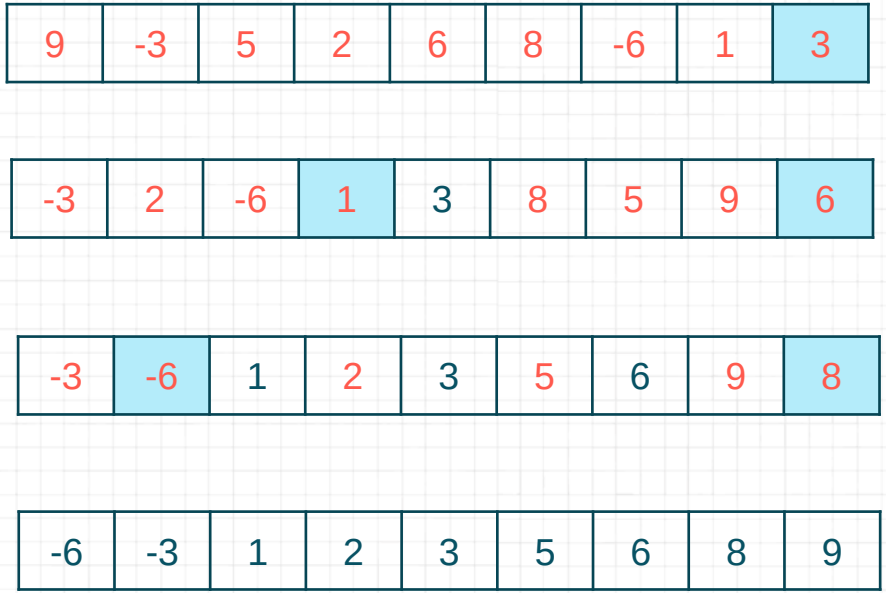
Sắp xếp chèn (insertion sort)

```
void insertionSort(int A[], int N){
    for(int i = 1; i < N; i++){
        int x = A[i];
        int j = i - 1;
        while(j >= 0 && x < A[j]){
            A[j+1] = A[j];
            j = j - 1;
        }
        A[j+1] = x;
    }
}
```

Sắp xếp nhanh (quick sort)

- Áp dụng kỹ thuật chia để trị
  - Mảng có 1 phần tử → mảng đã sắp xếp
  - Ngược lại
    - Chọn 1 phần tử làm pivot
    - Chia mảng thành các phần:
      - Nửa bên trái nhỏ hơn pivot
      - Phần tử bằng pivot → đúng vị trí cần sắp xếp
      - Nửa bên phải lớn hơn pivot
    - Gọi đệ quy sắp xếp nửa bên trái
    - Gọi đệ quy sắp xếp nửa bên phải

Sắp xếp nhanh (quick sort)



Sắp xếp nhanh (quick sort)

```
void quickSort(int A[], int l, int r){
    if(l < r){
        int p = (l + r)/2;
        int k = partition(a, l, r, p);
        quickSort(A, l, k-1);
        quickSort(A, k+1, r);
    }
}
```

Sắp xếp nhanh (quick sort)

```
int partition(int a[], int l, int r, int p)
{
   swap(a[p], a[r]);
   int i = l, j = r - 1;
   while (i <= j) {
        while(a[i] < a[r]) i++;
        while(a[j] > a[r]) j++;
        if(i < j)
           swap(a[i], [j]);
   }
   swap(a[i], a[r]);
   return i;
}
```

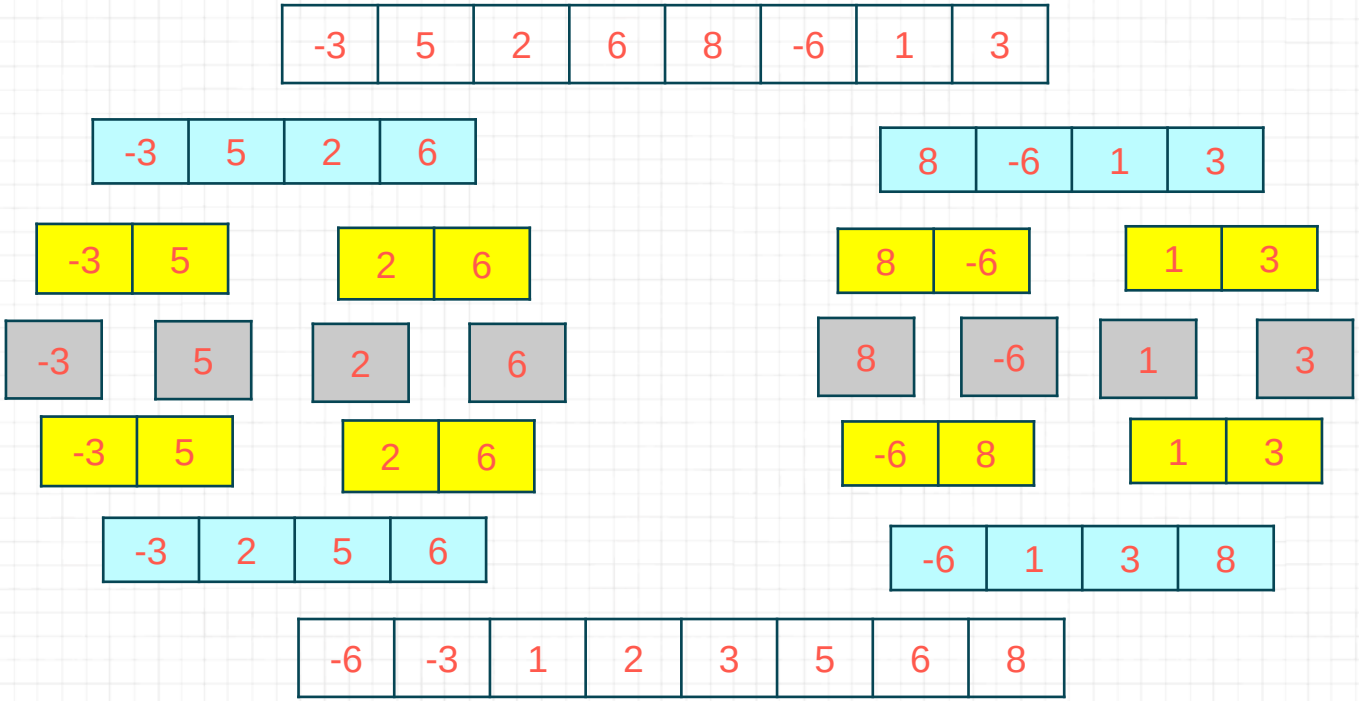
Sắp xếp nhanh (quick sort)

- Cài đặt quick sort bằng DSLK
  - Nếu DSLK chỉ có 1 node → coi như đã sắp xếp
  - Ngược lại
    - Khởi tạo hai DSLK rỗng L1, L2
    - Chọn node đầu tiên làm pivot
    - Duyệt qua các node của DSLK, chia ds làm 2:
      - Các node nhỏ hơn pivot thêm vào L1
      - Các node lớn hơn pivot thêm vào L2
    - Gọi đệ quy sắp xếp L1 và L2
    - Nối L1 vào pivot, rồi nối tiếp vào L2

Sắp xếp trộn (merge sort)

- Áp dụng kỹ thuật chia để trị
  - Mảng có 1 phần tử → mảng đã sắp xếp
  - Ngược lại
    - Chia mảng làm 2 nửa bên trái và bên phải
    - Gọi đệ quy sắp xếp nửa bên trái
    - Gọi đệ quy sắp xếp nửa bên phải
    - Trộn 2 mảng đã sắp xếp thành mảng đã sắp xếp

Sắp xếp trộn (merge sort)



Sắp xếp trộn (merge sort)

```
void mergeSort(int a[], int l, int r){
    if(l < r) {
        int mid = (l+r)/2;
        mergeSort(a, l, mid);
        mergeSort(a, mid+1, r);
        merge(a, l, m, r);
    }
}
```

Sắp xếp trộn (merge sort)

```
void merge(int A[], int l, int r, int m){
   int nL = m - l + 1, nR = r - m;
   int* L = new int[nL];
   int* R = new int[nR];
   for(int i = 0; i < nL; i++)
       L[i] = A[l + i];
   for(int j = 0; j < nR; j++)
       R[j] = A[m + 1 + j];
   int iL = 0, iR = 0, iA = l;
   while(iL < nL && iR < nR){
       if(L[iL] <= R[iR])
          A[iA++] = L[iL++];
       else
          A[iA++] = R[iR++];
   }
   //…continue
}
```

Sắp xếp trộn (merge sort)

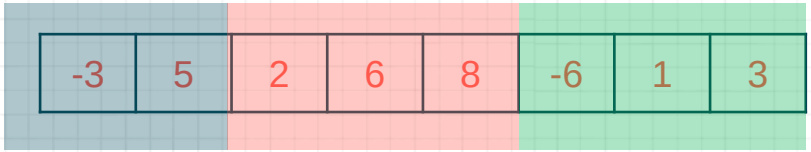
```
void merge(int A[], int l, int r, int m){
    //…continue
    while(iL < nL)
        A[iA++] = L[iL++];
    while(iR < nR)
        A[iA++] = R[iR++];

    delete[] L;
    delete[] R;
}
```

Sắp xếp trộn (merge sort)

- Cài đặt merge sort bằng DSLK → sử dụng natural merge sort
- Khái niệm đường chạy (run): là dãy con không giảm nằm liên tiếp nhau

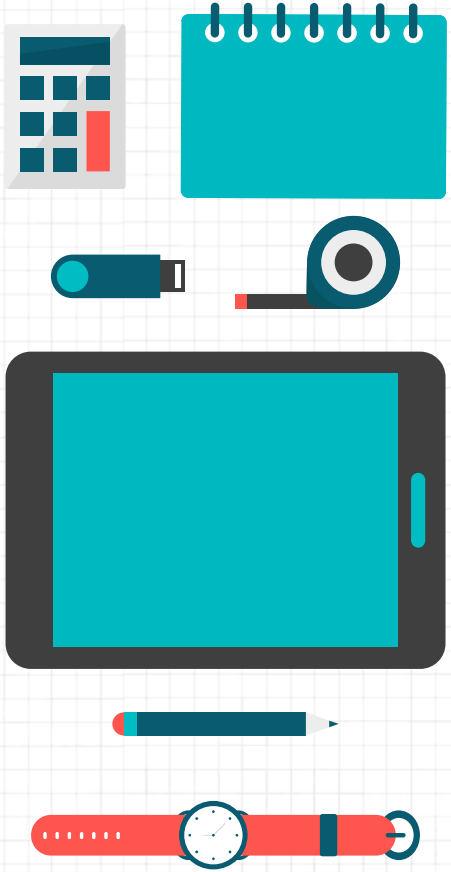
$$\begin{cases} a_k \le a_{k+1} \forall k \in [i, j) \\ a_i < a_{i-1} \\ a_j > a_{j+1} \end{cases}$$

- Ví dụ:



Sắp xếp trộn (merge sort)

- Trộn tự nhiên (natural merge sort)
  - Nếu DSLK L chỉ có 1 đường chạy → được sắp xếp
  - Tách danh sách liên kết L thành hai danh sách liên kết L_1, L_2 theo nguyên tắc luân phiên từng đường chạy
  - Gọi đệ quy trên DSLK L1, L2
  - Trộn DSLK L1, L2 được sắp xếp thành DSLK L được sắp xếp

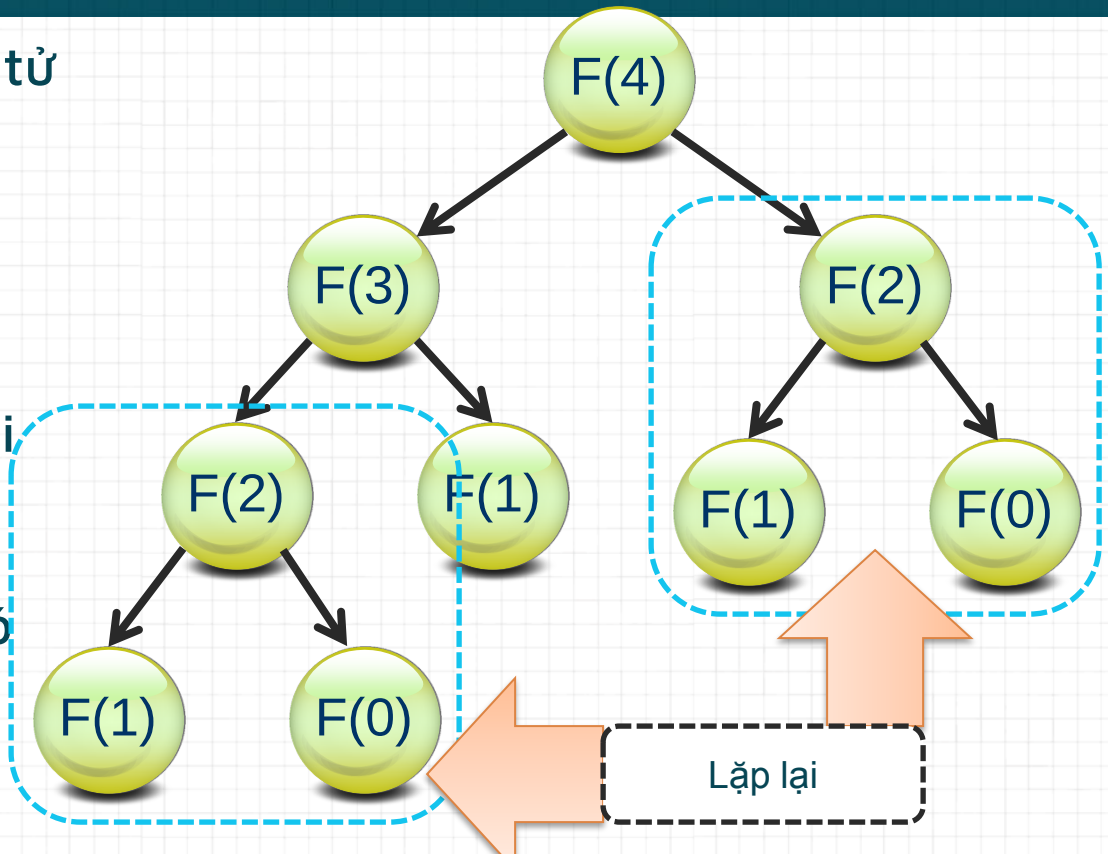


03

Quy hoạch động

Khái niệm quy hoạch động

- Bài toán dẫn nhập: Tìm phần tử thứ n của dãy Fibonacci
  - $F_0 = 0, F_1 = 1$
  - $F_n = F_{n-1} + F_{n-2}$ (n > 1)
- Giải bằng đệ quy: bắt đầu từ bài toán lớn phân rã thành bài toán con
- Có những bài toán con được giải lại cho dù đã giải trước đó rồi



Khái niệm quy hoạch động

- Ý tưởng quy hoạch động
  - Theo hướng bottom-up (từ dưới lên)
  - Đầu tiên giải các bài toán nhỏ (bài toán cơ sở) và lưu lại lời giải
  - Giải bài toán lớn hơn bằng cách phối hợp lời giải của những bài toán nhỏ (dựa vào công thức truy hồi), sau đó lưu lại lời giải của các bài toán này
  - Tiếp tục lặp lại cho đến khi giải được bài toán lớn nhất (bài toán ban đầu)

Khái niệm quy hoạch động

- Các khái niệm
  - Bài toán giải theo phương pháp quy hoạch động được gọi là **bài toán quy hoạch động**
  - Công thức phối hợp lời giải của các bài toán con để có lời giải của bài toán lớn là **công thức truy hồi của quy hoạch động**
  - Tập các bài toán nhỏ nhất có ngay lời giải để từ đó giải quyết các bài toán lớn hơn gọi là **cơ sở quy hoạch động**
  - Không gian lưu trữ lời giải các bài toán con để tìm cách phối hợp chúng gọi là **bảng phương án của quy hoạch động**

Điều kiện cho quy hoạch động

- Các điều kiện để giải được bài toán bằng quy hoạch động
  - Phải phân rã được bài toán lớn thành các bài toán con
  - Phải tìm được công thức truy hồi để phối hợp lời giải bài toán con thành lời giải của bài toán lớn
  - Phải đủ không gian vật lý để lưu trữ lời giải của các bài toán con
  - Quá trình đi từ giải bài toán cơ sở đến giải bài toán lớn ban đầu phải qua hữu hạn bước

Các bước quy hoạch động

- Bước 1: Phát biểu các bài toán con
- Bước 2: Lập công thức truy hồi
- Bước 3: Xây dựng bảng phương án
  - Lựa chọn cấu trúc dữ liệu lưu bảng phương án
  - Giải các bài toán cơ sở và lưu lời giải vào bảng phương án
  - Dùng công thức truy hồi phối hợp lời giải những bài toán nhỏ để tìm lời giải của những bài toán lớn hơn và lưu chúng vào bảng phương án (từ dưới lên). Lặp lại cho tới khi tìm được lời giải của bài toán ban đầu
- Bước 4: Dựa vào bảng phương án, truy vết tìm ra lời giải cho bài toán ban đầu

Ví dụ 1

- Tìm phần tử thứ n trong dãy Fibonacci
- Bước 1: Phát biểu các bài toán con.
  - Tìm phần tử thứ i của dãy Fibonacci (0 <= i <= n)
- Bước 2: Lập công thức truy hồi
  - $F_0 = 0, F_1 = 1$
  - $F_i = F_{i-1} + F_{i-2}$ (2 <= i <= n)
- Bước 3: Xây dựng bảng phương án
  - Cấu trúc dữ liệu: Bảng phương án là mảng F gồm n+1 phần tử mà phần tử F[i] = F_i (0 <= i <= n)
  - Giải bài toán cơ sở F[0] = F_0 = 0, F[1] = F_1 = 1

Ví dụ 1

- Bước 3: Xây dựng bảng phương án
  - Phối hợp lời giải các bài toán con thành lời giải bài toán lớn và lưu vào bảng phương án. F[i] = F[i-1] + F[i-2] (2 <= i <= n)

- Bước 4: Truy vết tìm lời giải bài toán lớn ban đầu
  - Khi i = n → truy xuất phần tử F[n] trong bảng phương án

Ví dụ 1

```
int fibo(int n){
    if(n == 0)
        return 0;
    if(n == 1)
        return 1;
    int* F = new int[n+1];
    F[0] = 0, F[1] = 1;
    for(int i = 2; i <= n; i++)
        F[i] = F[i-1] + F[i-2];
    int fn = F[n];
    delete[] F;
    return fn;
}
```

Ví dụ 2

- Tìm phần tử nhỏ nhất trong mảng số nguyên n phần tử
- Bước 1: Phát biểu các bài toán con.
  - Tìm phần tử nhỏ nhất T_i trong (i + 1) phần tử đầu tiên của mảng a[0], a[1],...,a[i] (0 <= i <= n-1)
- Bước 2: Lập công thức truy hồi
  - T_0 = a[0]
  - $T_i = \min(T_{i-1}, a[i])$ (1 <= i <= n-1)
- Bước 3: Xây dựng bảng phương án
  - Cấu trúc dữ liệu: Bảng phương án là mảng T gồm n phần tử mà phần tử T[i] = T_i (0 <= i <= n-1)
  - Giải bài toán cơ sở T[0] = T_0 = a[0]

Ví dụ 2

- Bước 3: Xây dựng bảng phương án
  - Phối hợp lời giải các bài toán con thành lời giải bài toán lớn và lưu vào bảng phương án. T[i] = min(T[i-1], a[i]) (1 <= i <= n-1)

- Bước 4: Truy vết tìm lời giải bài toán lớn ban đầu
  - Khi i = n-1 → truy xuất phần tử T[n-1] trong bảng phương án

Ví dụ 2

```
int getMin(int a[], int n){
    if(n == 1)
        return a[0];
    int* T = new int[n];
    T[0] = a[0];
    for(int i = 1; i < n; i++)
        T[i] = min(T[i-1], a[i]);
    int min = T[n-1];
    delete[] T;
    return min;
}
```

Ví dụ 3

- Tìm chiều dài dãy con tăng dài nhất (LIS) trong mảng số nguyên n phần tử
- Ví dụ dãy A= {5, 7, 4, -3, 9, 1, 10, 4, 5, 8, 9, 3} có dãy LIS = {-3, 1, 4, 5, 8, 9}

- Bước 1: Phát biểu các bài toán con.
  - Gọi L_i là chiều dài của dãy LIS có chứa phần tử a[i] trong (i + 1) phần tử đầu tiên của mảng a[0], a[1],…,a[i] (0 <= i <= n-1)
- Bước 2: Lập công thức truy hồi
  - $L_0 = 1$
  - $L_i = 1 + \max_j\{L_j : a[j] < a[i] \ \& \ j < i\}$ (1 <= i <= n-1)
- Bước 3: Xây dựng bảng phương án

Ví dụ 3

- Bước 3: Xây dựng bảng phương án
  - Cấu trúc dữ liệu Bảng phương án là mảng L gồm n phần tử mà phần tử L[i] = L_i (1 <= i <= n-1)
  - Giải bài toán cơ sở L[0] = L_0 = 1
  - Phối hợp lời giải bài toán thành bài toán lớn và lưu vào bảng phương án.
    - Trong các phần tử a[0],...a[i-1], tìm phần tử a[j] < a[i] có L[j] lớn nhất
    - L[i] = 1 + L[j] (1 <= i <= n-1)
- Bước 4: Truy vết tìm lời giải bài toán lớn ban đầu
  - Tìm giá trị lớn nhất trong mảng L

Ví dụ 3

```
int LIS(int a[], int n){
    if(n == 1)
        return 1;
    int* L = new int[n];
    L[0] = 1;
    for(int i = 1; i < n; i++){
        L[i] = 1;
        for(int j = i-1; j>=0; j--)
            if(a[j] < a[i])
                L[i] = 1 + max(L[i], L[j]);
    }
    int lis = L[0];
    for(int i = 0; i < n; i++)
        lis = max(lis, L[i]);
    delete[] L;
    return lis;
}
```

Ví dụ 4

- Bài toán balô: Cho N gói hàng, gói hàng thứ i có khối lượng w_i và giá trị v_i. Cần chọn những gói hàng nào bỏ vào balô sao cho tổng giá trị các gói hàng được chọn là lớn nhất và tổng khối lượng không vượt quá khối lượng tối đa M mà balô có thể chứa. Mỗi gói chỉ được chọn duy nhất 1 lần hoặc không chọn

- Ví dụ N = 5 và M = 13.

	1	2	3	4	5
w	3	4	5	2	1
v	4	5	6	3	1

Ví dụ 4

- Bước 1: Phát biểu các bài toán con.
  - Gọi F(i,j) là tổng giá trị lớn nhất của các gói hàng được chọn từ i gói hàng sao cho tổng khối lượng không vượt quá j (0 <= i <= N, 0<=j<=M)
- Bước 2: Lập công thức truy hồi
  - F(0,j) = 0 và F(i,0) = 0
  - Trường hợp w_i > j → F(i,j) = F(i-1,j)
  - Trường hợp w_i <= j
    - Nếu gói hàng thứ i không được chọn → F(i,j) = F(i-1,j)
    - Nếu gói hàng thứ i được chọn → F(i,j) = F(i-1,j-w_i) + v_i.
    - Tổng hợp các trường hợp: F(i,j) = max{F(i-1,j), F(i-1,j-w_i) + v_i}

Ví dụ 4

- Bước 3: Xây dựng bảng phương án
  - Cấu trúc dữ liệu:
    - Lưu các giá trị w và v bằng 2 mảng w và v có N + 1 phần tử với w[0] = 0, v[0] = 0 và w[i] = w_i, v[i] = v_i (0 <= i <= N)
    - Bảng phương án là mảng hai chiều F gồm N + 1 dòng và M + 1 cột mà mỗi phần tử F[i][j] = F(i,j)
  - Giải bài toán cơ sở F[0][j] = 0 và F[i][0]=0 → cho toàn bộ dòng đầu tiên và cột đầu tiên của mảng F bằng 0

Ví dụ 4

- Bước 3: Xây dựng bảng phương án
  - Phối hợp lời giải bài toán nhỏ thành lời giải bài toán lớn và lưu vào bảng phương án.
    - Dùng giá trị của dòng (i-1) để tính giá trị dòng i
    - TH w[i] > j: F[i][j] = F[i-1][j]
    - TH w[i] <= j: F[i][j] = max{F[i-1][j], F[i-1][j-w[j]] + v[j]

Ví dụ 4 (N=5, M=13)

w[i] > j: F[i][j] = F[i-1][j]
w[i] <= j: F[i][j] = max{F[i-1][j], F[i-1][j-w[i]] + v[j]}

v	w	i \ j	0	1	2	3	4	5	6	7	8	9	10	11	12	13
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	3	1	0	0	0	4	4	4	4	4	4	4	4	4	4	4
5	4	2	0	0	0	4	5	5	5	9	9	9	9	9	9	9
6	5	3	0	0	0	4	5	6	6	9	10	11	11	11	15	15
3	2	4	0	0	3	4	5	7	8	9	10	12	13	14	15	15
1	1	5	0	1	3	4	5	7	8	9	10	12	13	14	15	16

Ví dụ 4

- Bước 4: Truy vết tìm lời giải bài toán lớn ban đầu
  - Bắt đầu từ phần tử cuối cùng F[N][M] lần ngược về dòng 0 theo nguyên tắc
  - F[i][j] $\neq$ F[i-1][j] → gói thứ i được chọn → truy tiếp ô F[i-1][j-w[i]]
  - F[i][j] = F[i-1][j] → gói thứ i không chọn → truy tiếp về ô F[i-1][j]

Ví dụ 4 (N=5, M=13)

F[i][j] ≠ F[i-1][j] → gói thứ i được chọn → truy tiếp ô F[i-1][j-w[i]]

F[i][j] = F[i-1][j] → gói thứ i không chọn → truy tiếp về ô F[i-1][j]

v	w		0	1	2	3	4	5	6	7	8	9	10	11	12	13
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	3	1	0	0	0	4	4	4	4	4	4	4	4	4	4	4
5	4	2	0	0	0	4	5	5	5	9	9	9	9	9	9	9
6	5	3	0	0	0	4	5	6	6	9	10	11	11	11	15	15
3	2	4	0	0	3	4	5	7	8	9	10	12	13	14	15	15
1	1	5	0	1	3	4	5	7	8	9	10	12	13	14	15	16

Bài tập

- Bài 1: Có một dự án kéo dài trong T tháng, người quản lý cần phải lập lịch sử dụng công nhân mỗi tháng cho dự án. Biết rằng, số công nhân tối thiểu cần trong tháng thứ i là S(i); tiền dịch vụ khi thuê 1 công nhân mới là DV; tiền đền bù khi sa thải một công nhân là ST; lương tháng mỗi công nhân phải trả là LT. Cần phải thuê hay sa thải bao nhiêu công nhân mỗi tháng để tổng chi phí nhân công của dự án là nhỏ nhất

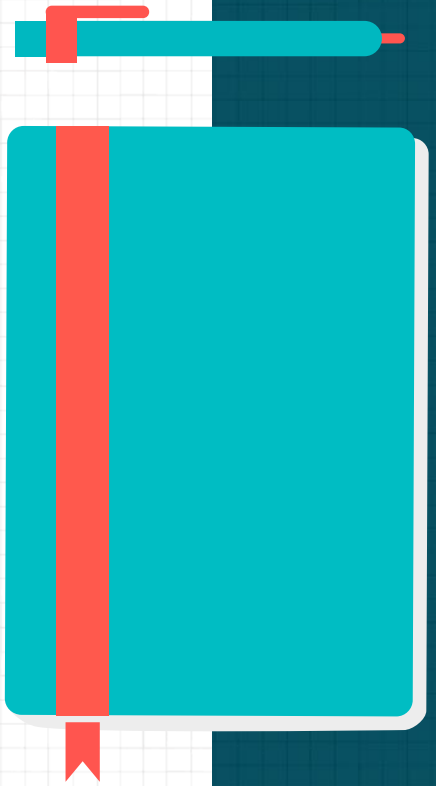
- Bài 2: Có N gói kẹo, gói thứ i có a_i cái kẹo. Không được bóc bất kỳ một gói kẹo nào, cần chia N gói kẹo thành hai phần sao cho độ chênh lệch số kẹo giữa hai gói là ít nhất

Bài tập

Bài 3: Một lâu đài cổ có M tầng lầu, mỗi tầng lầu có N phòng. Tại mỗi phòng của lâu đài đều có 3 cái thang thông với 3 phòng của tầng lầu phía trên gồm: phòng ngay bên trên, phòng trên bên trái và phòng trên bên phải. Trong mỗi phòng có một rương tiền với giá trị nhất định.

Một người săn kho báu xuất phát từ tầng trệt của lâu đài. Anh ta vào một phòng bất kỳ rồi từ đó đi lên các phòng ở tầng trên. Mỗi tầng anh ta chỉ ghé qua một phòng.

Hãy tìm lộ trình phòng ở mỗi tầng mà người săn kho báu ghé qua để thu được số tiền nhiều nhất.



The End!

Do you have any questions?