# Thuật toán ứng dụng
# Bài thực hành số 4

Giảng viên: TS. Đinh Viết Sang
Trợ giảng: Nguyễn Trung Hiếu

Viện Công nghệ thông tin & truyền thông
Đại học Bách khoa Hà Nội

04/2021

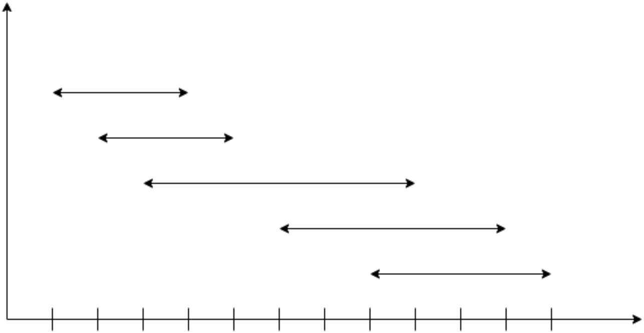
# Nội dung

# MACHINE

- Cho n đoạn, đoạn thứ $i$ bắt đầu từ $s_i$ đến $t_i$.
- Số tiền nhận được khi chọn đoạn thứ $i$ là $t_i - s_i$.
- 2 đoạn $i, j$ được gọi là tách biệt nếu $t_i < s_j$ hoặc $t_j < s_i$.
- Cần chọn 2 đoạn tách biệt sao cho số tiền nhận được là lớn nhất.
- In ra số tiền nhận được

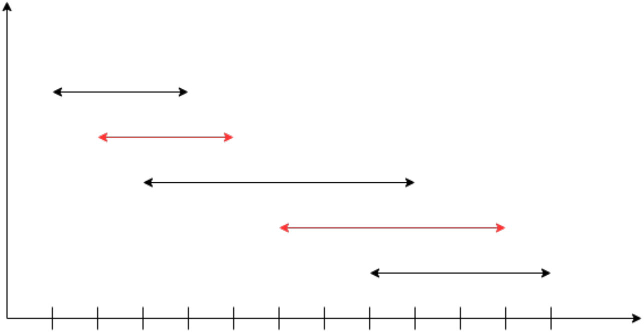
# Ví dụ

- Có 5 đoạn thẳng $[8, 12]; [6, 11]; [3, 9]; [2, 5]; [1, 4]$



# Ví dụ

- Cách chọn tối ưu : chọn 2 đoạn $[6, 11]$ và $[1, 4]$.
- Số tiền nhận được : 8



# Thuật toán 1

- Duyệt toàn bộ $\frac{n(n-1)}{2}$ cách chọn, mỗi cách chọn kiểm tra điều kiện và lấy kết quả tối ưu.
- Độ phức tạp $O(n^2)$

# Thuật toán 2

- Sử dụng quy hoạch động.
- Gọi $maxs[x]$ là giá trị đoạn lớn nhất có điểm cuối $\leq x$
- Giả sử đoạn $i$ là một đoạn được chọn và có điểm cuối $t_i$ lớn hơn đoạn còn lại thì giá trị lớn nhất mà ta có thể nhận được là $t_i - s_i + maxs[s_i - 1]$
- Lấy giá trị max $t_i - s_i + maxst[s_i - 1]$ của tất cả các vị trí $i$
- Độ phức tạp : $O(n)$

# Code

```
int main() {
    const int N = 2e6 + 5;
    for (int i = 1; i <= n; i++) {
        maxs[t[i]] = max(maxs[t[i]], t[i] - s[i]);
    }

    for (int i = 1; i < N; i++) {
        maxs[i] = max(maxs[i - 1], maxs[i]);
    }

    int ans = -1;
    for (int i = 1; i <= n; i++) {
        if (maxs[s[i] - 1] > 0) {
            ans = max(ans,
                    maxs[s[i] - 1] + t[i] - s[i]);
        }
    }
    cout << ans << endl;
}
```

# WAREHOUSE

- $N$ nhà kho được đặt tại các vị trí từ $1...N$. Mỗi nhà kho có:
  - $a_i$ là số lượng hàng.
  - $t_i$ là thời gian lấy hàng.
- Một tuyến đường lấy hàng đi qua các trạm $x_1 < x_2 < ... < x_k$ $(1 \leq x_j \leq N, j = 1...k)$ thỏa mãn:
  - $x_{i+1} - x_i \leq D \; \forall i \in [1, k]$.
  - $\sum_{i=1}^{k} t[x_i] \leq T$
- Cần tìm tuyến đường có tổng số lượng hàng trên đường đi là tối đa.

# Thuận toán

- Gọi $dp[i][k]$ là số lượng hàng tối đa thu được khi xét các nhà kho từ 1...$i$, lấy hàng ở kho $i$ và thời gian lấy hàng không quá $k$.
- Công thức
  - $dp[i][k] = 0$ nếu $k < t[i]$
  - $dp[i][k] = max(dp[j][k - t[i]] + a[i], j \in [i - D, i - 1])$ nếu $k \geq t[i]$
- Kết quả $max(dp[i][k], i \in [1, n], k \in [1, T])$
- Độ phức tạp: $O(n \times T \times D)$

# Code

```
int main() {
    for (int i = 1; i <= n; i++) {
        for (int k = t[i]; k <= T; k++) {
            for (int j = i-1; j >= max(0,i-D); j--)
                dp[i][k] = max(dp[i][k],
                    dp[j][k-t[i]] + a[i]);
            ans = max(ans, dp[i][k]);
        }
    }
    cout << ans << endl;
}
```

# 05. GOLD MINING

- Có $n$ nhà kho nằm trên một đoạn thẳng.
- Nhà kho $i$ có toạ độ là $i$ và chứa lượng vàng là $a_i$.
- Chọn một số nhà kho sao cho:
  - Tổng lượng vàng lớn nhất.
  - 2 nhà kho liên tiếp có khoảng cách nằm trong đoạn $[L_1, L_2]$.

# Thuật toán 1

- Gọi $F[i]$ là tổng số vàng nếu nhà kho $i$ là nhà kho cuối cùng được chọn.
- Khởi tạo: $F[i] = a[i], \forall i < L_1$.
- Công thức truy hồi:

$$F[i] = max_{j \in [i-L_2, i-L_1]}(a[i] + F[j]), \forall i \in [L_1, N) \qquad (1)$$

- Kết quả: $max(F[i]), \forall i \in [1, N]$.
- Độ phức tạp: $O(N \times (L_2 - L_1)) = O(N^2)$.

# Code thuật toán 1

```
int main() {
    ...
    for (int i = 0; i < n; i++) {
        F[i] = a[i];
    }
    for (int i = l1; i < n; i++) {
        for (int j = i - l2; j <= i - l1; j++) {
            F[i] = max(F[i], F[j] + a[i]);
        }
    }
    ...
}
```

# Cải tiến

- Sử dụng dequeue để tìm phần tử $F[i]$ lớn nhất trong đoạn $[i - L2, i - L1]$.
- Hàng đợi 2 đầu (dequeue) là cấu trúc dữ liệu cho phép chèn/xóa phần tử có thể được ở đầu hoặc cuối dequeue.
- **Điều kiện 1:** Dequeue này sẽ lưu giá trị của $F[i]$ giảm dần và đảm bảo các phần tử ở sau luôn có chỉ số lớn hơn phần tử trước.

Thuật toán cải tiến:

- Tại mỗi $i$, ta lưu các giá trị $F[j], \forall j \in [i - L2, i - L1]$ vào dequeue sao cho đảm bảo **điều kiện 1**.
- Lúc này, $F[j]$ cần tìm chính là phần tử đầu tiên của dequeue.
- $F[i] = a[i] + F[j]$
- Kết quả: $max(F[i]), \forall i \in [1, N]$.
- Độ phức tạp: $O(N)$.

# Code

```
deque <pair<long long, int> >dq;

int main(){
    dp[1] = a[1];
    while(!dq.empty()) dq.pop_back();
    for(int i = 2; i <= n; i++){
      pushPop(i);
      dp[i] = 1ll*a[i];
      if(!dq.empty()){
        dp[i] += dq.front().first;
      }
    }
    long long ans = *max_element(dp+1, dp+n+1);
    cout << ans << endl;
}
```

# Code

```
void pushPop(int id)
{
  int x = id - L1;
  int y = id - L2;
  if(x > 0 && x <= n){
    while(!dq.empty() && dp[x] >= dq.back().first) {
      dq.pop_back();
    }
    dq.push_back(make_pair(dp[x], x));
  }
  if(y > 0 && y <= n){
    while(!dq.empty() && dq.front().second < y) {
      dq.pop_front();
    }
  }
}
```

# Phân tích độ phức tạp

- Tại sao độ phức tạp lại là $O(n)$?
- Mỗi phần tử chỉ được push dequeue vào 1 lần
- Mỗi phần tử cũng pop ra khỏi dequeue 1 lần
- Tổng các thao tác với dequeue là $O(2 \times N)$, do đó, độ phức tạp của thuật toán chỉ là $O(N + 2 \times N) = O(3 \times N)$.