Thuật toán ứng dụng

Giảng viên : Đỗ Tuấn Anh, Lê Quốc Trung
TA: Trần Thanh Tùng

Viện Công Nghệ Thông Tin và Truyền Thông
Trường Đại học Bách Khoa Hà Nội

Tháng 5, năm 2020

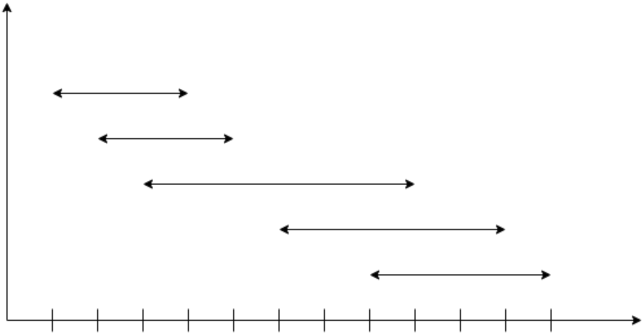
Mục lục

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

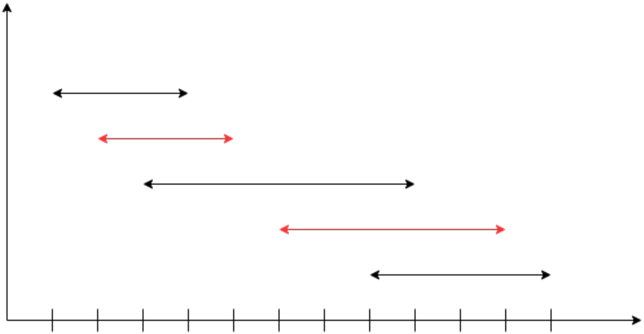
Ví dụ

- 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 $maxAmount[x]$ 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 $t_i - s_i + maxAmount[s_i - 1]$
- Lấy giá trị max $t_i - s_i + maxAmount[s_i - 1]$ 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;
}
```

MARBLE

- Có một tấm đá có kích thước $W \times H$.
- Cần cắt tấm đá thành các miếng có kích thước nằm trong $W_1 \times H_1, W_2 \times H_2, \ldots, W_n \times H_n$.
- Tấm đá có vân nên không thể xoay, có nghĩa là miếng đá $A \times B$ khác miếng đá $B \times A$.
- Các lát cắt phải thẳng và được cắt tại các điểm nguyên theo cột hoặc theo hàng, và phải cắt hết hàng hoặc hết cột.
- Các miếng đá không có kích thước như trên sẽ bị bỏ đi.
- Tìm cách cắt sao cho diện tích bỏ đi là ít nhất.

Thuật toán

Thuật toán 1 : Duyệt hết tất cả các cách cắt.

Thuật toán 2 : Quy hoạch động : Gọi $dp_{i,j}$ là phần diện tích bỏ đi ít nhất khi miếng đá có kích thước là $i \times j$.

- Ta sẽ tính $dp_{i,j}$ dựa trên các giá trị của $dp_{i',j'}$ với $i' \leq i$ và $j' \leq j$ đã được tính từ trước.
- $dp_{i,j} = 0$ nếu $\exists k (1 \leq k \leq n) : (i,j) = (W_k, H_k)$.
- Nếu cắt theo chiều ngang, ta có :

$$dp_{i,j} = \min_{i_0=1}^{i-1} (dp_{i_0,j} + dp_{i-i_0,j})$$

- Nếu cắt theo chiều dọc, ta có :

$$dp_{i,j} = \min_{j_0=1}^{j-1} (dp_{i,j_0} + dp_{i,j-j_0})$$

- Kết quả là $dp_{W,H}$. ĐPT thuật toán $O(WH(N+W+H))$.

Code

```
int main() {
    for (int i = 1; i <= W; i++) {
        for (int j = 1; j <= H; j++) {
            dp[i][j] = i * j;
            for (int k = 1; k <= n; k++) {
                if (i == w[k] && j == h[k]) {
                    dp[i][j] = 0;
                    break;
                }
            }
            for (int k = 1; k < i; k++) {
                dp[i][j] = min(dp[i][j],
                               dp[k][j] + dp[i - k][j]);
            }
            for (int k = 1; k < j; k++) {
                dp[i][j] = min(dp[i][j],
                               dp[i][k] + dp[i][j - k]);
            }
        }
    }
}
```

WAREHOUSE

- N nhà kho được đặt tại các vị trí từ $1 \dots N$. 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 $x_1 < x_2 < \dots < x_k$ $(1 \leq x_j \leq N, j = 1 \dots k)$ 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 $1 \ldots i$, 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 $max(dp[i][k], i \in [1, n], k \in [1, T])$

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;
}
```