Thuật toán ứng dụng

BÀI TẬP LẬP TRÌNH

Giảng viên: Ban Hà Bằng
TA: Phan Trung Kiên

SOICT — HUST

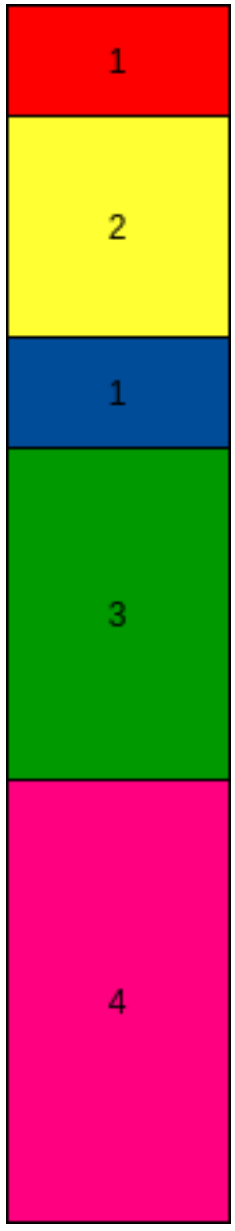
05/2020

Outline

04. BOOKS1

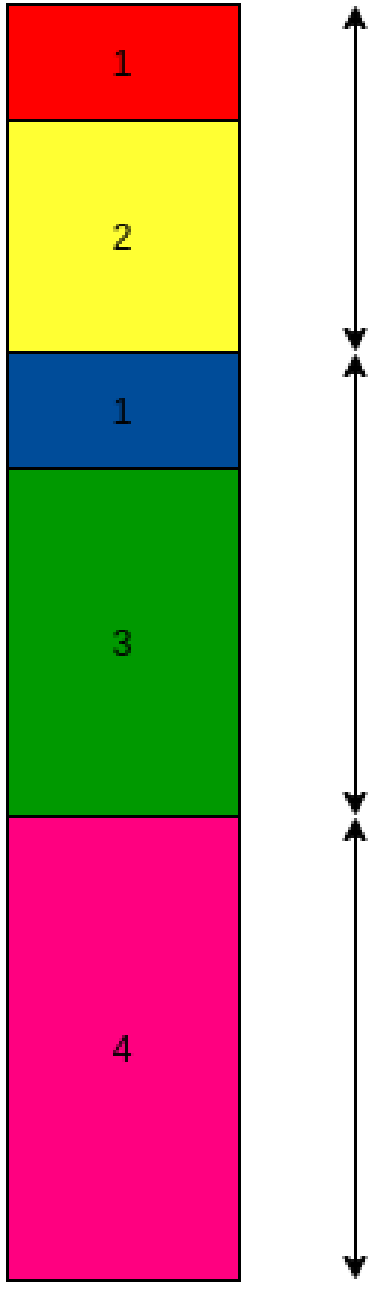
- Có m quyển sách, quyển sách thứ i dày p_i trang.
- Phải chia số sách trên cho đúng k người, mỗi người sẽ nhận được một đoạn sách liên tiếp nhau.
- In ra cách chia để số trang sách lớn nhất được nhận bởi một người là nhỏ nhất.
- Nếu có nhiều kết quả lớn nhất thì ưu tiên số sách nhận bởi người 1 là ít nhất, sau đó đến người 2, ...

Ví dụ



- Đầu vào có 5 quyển sách và phải chia số sách trên cho 3 người
- Mỗi quyển sách có độ dày như hình bên

Ví dụ



- Kết quả của bài toán là 4
- Có 2 cách chia để đạt được kết quả trên :
  1 2/1 3/4 hoặc
  1 2 1/3/4
- Cách chia như hình bên là kết quả của bài toán

Thuật toán 1

- Duyệt kết quả của bài toán từ nhỏ đến lớn, cố định số trang sách lớn nhất được chia bởi 1 người.
- Với mỗi kết quả ta đi kiểm tra có chia được cho đúng k người hay không bằng thuật toán tham lam.
- In ra kết quả ngay khi tìm được kết quả thỏa mãn
- Độ phức tạp thuật toán $O(MAX * n)$

Code 1

```
bool check(long long max_val) {
    vector < int > pos;
    long long sum = 0;
    for (int i = n; i >= 1; i--) {
        if (sum + a[i] <= max_val) {
            sum += a[i];
        } else {
            sum = a[i];
            if (a[i] > max_val) { return false; }
            pos.push_back(i);
        }
    }
    if (pos.size() >= k) { return false; }
    return true;
}
```

Thuật toán 2

- Gọi *maxVal* là số trang lớn nhất được chia bởi 1 người.
- Nhận thấy nếu với giá trị $maxVal = x$ có thể chia dãy thành $\leq k$ đoạn thì với $maxVal = x + 1$ cũng có thể chia dãy thành $\leq k$ đoạn với cách chia như cũ.
- Ta chặt nhị phân giá trị *maxVal*.
- Độ phức tạp thuật toán $O(\log MAX * n)$

Code 2

```
16 bool check(long long max_val) {
17     // Giong voi ham o Code 1
18 }
19 int main() {
20     int q; cin >> q;
21     while (q--) {
22         cin >> n >> k;
23         for (int i = 1; i <= n; i++) { cin >> a[i]; }
24         long long l = 0, r = MAX;
25         while (r - l > 1) {
26             long long mid = (l + r)/2;
27             if (check(mid)) {
28                 r = mid;
29             } else {
30                 l = mid;
31             }
32         }
33         ** In kq tuong ung voi gia tri r **
34     }
35 }
```

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

Tìm kiếm vét cạn:

- Nhà kho thứ i có thể được chọn hoặc không.
- Sử dụng 1 vector chứa danh sách các nhà kho được chọn.
- Kiểm tra 2 nhà kho liên tiếp có thoả mãn ràng buộc không.
- Tính tổng số vàng của các nhà kho được chọn.
- Độ phức tạp: $O(2^n)$.

Thuật toán 2

Quy hoạch động:

- 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: $max_i F[i]$.
- Độ phức tạp: $O(N \times (L_2 - L_1)) = O(N^2)$.

Code

```
37 int main() {
38     cin >> n >> l1 >> l2;
39     for (int i = 1; i <= n; i++) {
40         cin >> a[i];
41     }
42     int res = 0;
43     for (int i = 1; i <= n; i++) {
44         f[i] = a[i];
45         for (int j = max(1, i - l2); j <= i - l1; j++)
46             f[i] = max(f[i], f[j] + a[i]);
47         }
48         res = max(res, f[i]);
49     }
50     cout << res << endl;
51     return 0;
52 }
```

Cải tiến thuật toán

Sử dụng hàng đợi ưu tiên:

- Mỗi phần tử trong hàng đợi là một cặp giá trị $(j, F[j])$.
- Ưu tiên phần tử có $F[j]$ lớn.
- Khi xét đến nhà kho i, thêm cặp giá trị $(i - L_1, F[i - L_1])$ vào hàng đợi.
- Loại bỏ phần tử j ở đầu hàng đợi trong khi $i - j > L_2$, gán $F[i] = a[i] + F[j]$.
- Độ phức tạp: $O(n + n \times log(n)) = O(n \times log(n))$

05. 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 $x_1 < x_2 < ... < x_k$ $(1 \leq x_j \leq N, j = 1...k)$ sao cho:
  - $x_{i+1} - x_i \leq D \; \forall i \in [1, k]$.
  - $\sum_{i=1}^{k} t[x_i] \leq T$

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.
- $dp[i][k_1] <= dp[i][k_2]\ \forall(k_1 < k_2)$
- $$dp[i][k] = \begin{cases} 0 & \text{if } k < t[i] \\ \max(dp[j][k - t[i]] + a[i], j \in [i - D, i - 1]) & \text{if } k \geq t[i] \end{cases}$$
- kết quả $ans = max(dp[i][k], i \in [1, n], k \in [1, T])$

Code

```
53 for (int i = 1; i <= n; i++) {
54     for (int k = t[i]; k <= T; k++) {
55         for (int j = i-1; j >= max(0,i-D); j--)
56             dp[i][k] = max(dp[i][k],
57                 dp[j][k-t[i]] + a[i]);
58         ans = max(ans, dp[i][k]);
59     }
60 }
```

05. NURSE

- Cần sắp xếp lịch làm việc cho một y tá trong N ngày
- Lịch làm việc bao gồm các giai đoạn làm việc được xen giữa bởi các ngày nghỉ
- Các giai đoạn làm việc là các ngày làm việc liên tiếp thỏa mãn hai điều kiện sau
  - Thời gian nghỉ giữa hai giai đoạn không quá một ngày
  - Số ngày làm việc của mỗi giai đoạn lớn hơn hoặc bằng K_1 và bé hơn hoặc bằng K_2
- Tìm số phương án xếp lịch thỏa mãn.

Thuật toán 1

- Mỗi cách xếp lịch tương ứng với một dãy nhị phân độ dài n. Bit thứ i là 0/1 tương ứng là ngày đó y tá được nghỉ hoặc phải đi làm
- Xét hết các xâu nhị phân độ dài n và tìm số lượng xâu thỏa mãn điều kiện

Thuật toán 2

- Gọi $F[x, i]$ là số cách xếp lịch thỏa mãn cho đến ngày thứ i và x là trạng thái nghỉ hoặc làm việc của ngày đó.
- Ta có: $F[0, 1] = F[1, 1] = 1$
- Với i là ngày nghỉ, ta có: $F[0, i] = F[1, i - 1]$
- Với i là ngày làm việc, ta có: $F[1, i] = \sum_{k=i-K_2}^{i-K_1} F[0, k]$
- Kết quả của bài toán là: $F[0, n] + F[1, n]$

Code

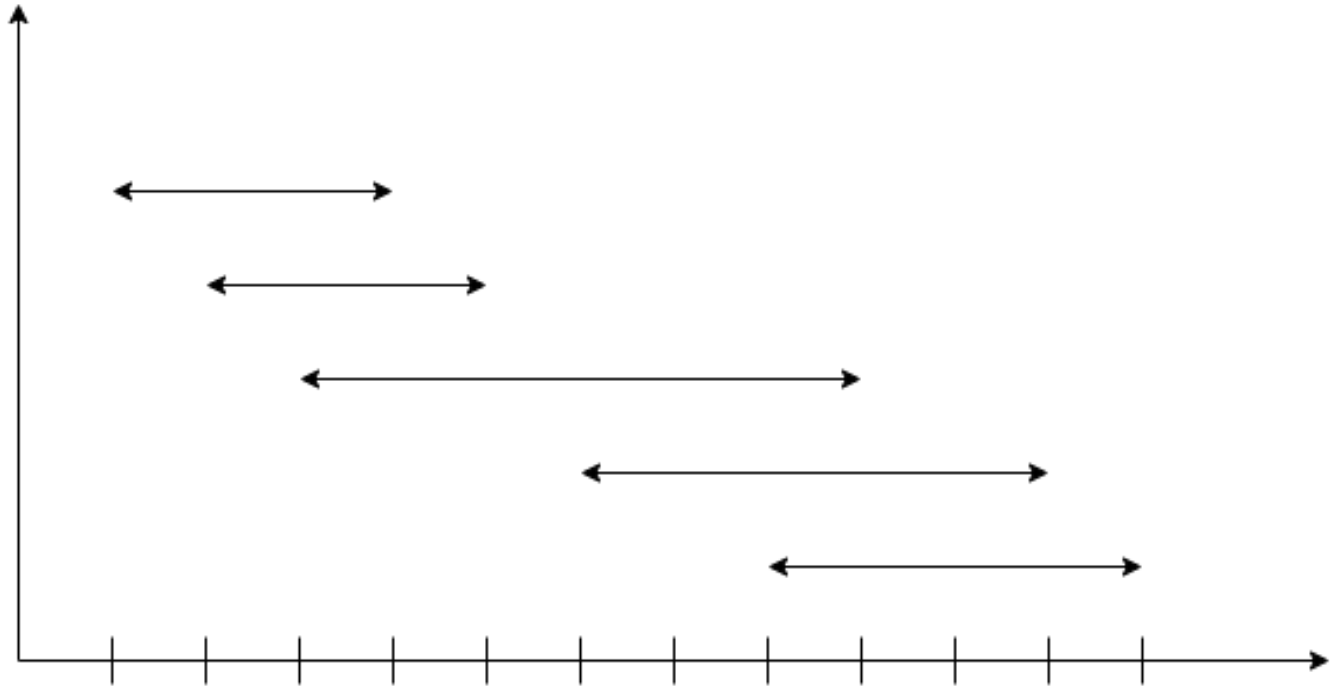
```
61  int main()
62  {
63      cin >> n >> k1 >> k2;
64      S[0, 1] = S[1, 1] = 1;
65      for (int i = 2; i <= n; i++)
66      {
67          for (int j = min(0,i-k2); j <= i-k1; j++)
68              S[1, i] += S[0, j];
69          S[0, i] = S[1, i-1];
70      }
71      cout << S[0, n] + S[1, n];
72      return 0;
73  }
```

05. 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

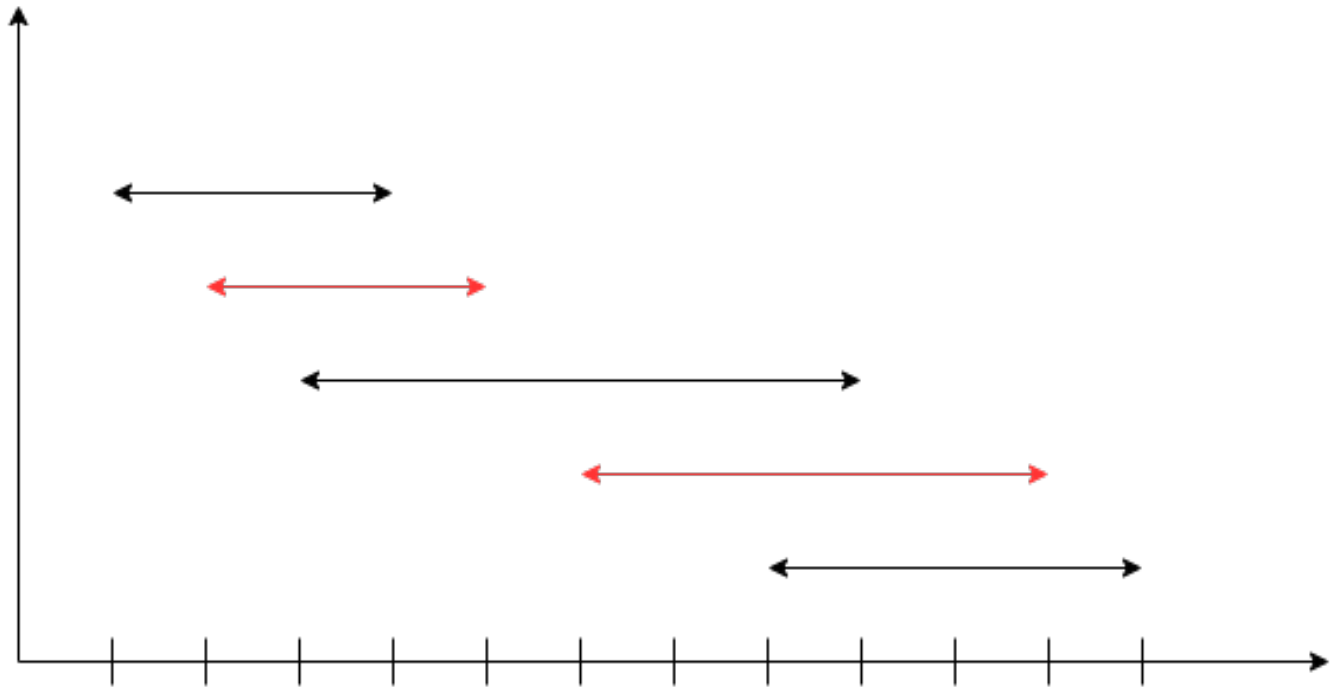
Ví dụ

- 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

```
74  int main() {
75      const int N = 2e6 + 5;
76      for (int i = 1; i <= n; i++) {
77          maxs[t[i]] = max(maxs[t[i]], t[i] - s[i]);
78      }
79
80      for (int i = 1; i < N; i++) {
81          maxs[i] = max(maxs[i - 1], maxs[i]);
82      }
83
84      int ans = -1;
85      for (int i = 1; i <= n; i++) {
86          if (maxs[s[i] - 1] > 0) {
87              ans = max(ans,
88                        maxs[s[i] - 1] + t[i] - s[i]);
89          }
90      }
91      cout << ans << endl;
92  }
```