Thuật toán ứng dụng

BÀI TẬP LẬP TRÌNH

Dành cho Trợ giảng thực hành

SOICT — HUST

02/02/2020

Lưu ý không show toàn bộ code cho sinh viên, chỉ cần chữa chi tiết phần thuật toán xử lý chính của bài

Outline

04. DIVIDE AND CONQUER

04. DIVIDE AND CONQUER

- 04. PIE
- 04. AGGRCOW
- 04. BOOKS1
- 04. EKO
- 04. FIBWORDS
- 04. CLOPAIR

04. PIE (VUONGDX)

- Có N cái bánh và $F + 1$ người.
- Mỗi cái bánh có hình tròn, bán kính r và chiều cao là 1.
- Mỗi người chỉ được nhận một miếng bánh từ một chiếc bánh.
- Cần chia bánh sao cho mọi người có lượng bánh bằng nhau (có thể bỏ qua vụn bánh).
- Tìm lượng bánh lớn nhất mỗi người nhận được.

Thuật toán

- Gọi $p[i]$ là số người ăn chiếc bánh thứ i. Lượng bánh mỗi người nhận được là $min_i\{\frac{V[i]}{p[i]}\}$ với $V[i]$ là thể tích của chiếc bánh thứ i.
- **Cách 1 - Tìm kiếm theo mảng p:** Tìm kiếm vét cạn mọi giá trị của p.
- **Cách 2 - Tìm kiếm theo lượng bánh mỗi người nhận được:** Thử từng kết quả, với mỗi kết quả, kiểm tra xem có thể chia bánh cho tối đa bao nhiêu người.
- **Tối ưu cách 2:** Sử dụng thuật toán tìm kiếm nhị phân để tìm kiếm kết quả.

Code

```
1     sort(r, r + N);
2
3         double lo = 0, hi = 4e8, mi;
4
5         for(int it = 0; it < 100; it++){
6             mi = (lo + hi) / 2;
7
8             int cont = 0;
9
10            for(int i = N - 1;
11            i >= 0 && cont <= F; --i)
12                cont += (int)
13                floor(M_PI * r[i] / mi);
14
15            if(cont > F) lo = mi;
16            else hi = mi;
17        }
```

04. AGGRCOW (quanglm)

- Có N chuồng bò và C con bò.
- Chuồng bò thứ i có tọa độ là x_i.
- Cần xếp các con bò vào các chuồng sao cho khoảng cách nhỏ nhất giữa 2 con bò bất kỳ là lớn nhất.

Thuật toán

- **Thuật toán 1:** Duyệt vét cạn từng con bò vào từng chuồng rồi tính khoảng cách ngắn nhất, $O(N^C \times C)$.
- **Thuật toán 2:** Duyệt giá trị kết quả bài toán d. Mỗi d, xếp các con bò vào chuồng 1 cách tham lam sao cho con bò sau cách con bò trước ít nhất d đơn vị. Nếu xếp đủ C con bò thì d là một giá trị hợp lệ. Tìm d lớn nhất. $O(max(x_i) \times N)$.
- **Thuật toán 3:** Tìm kiếm nhị phân với giá trị d. $O(\log max(x_i) \times N)$.

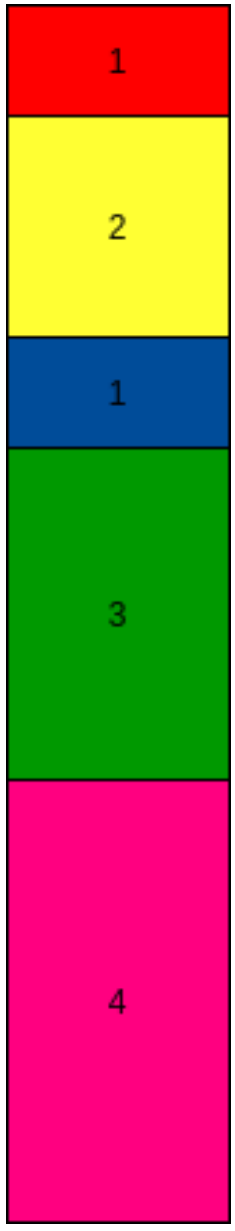
Code

```
18      sort(x + 1, x + n + 1);
19      int low = -1, high = (int)1e9 + 10;
20      while (high - low > 1) {
21          int mid = (low + high) / 2;
22          int num = 0;
23          int last = (int)-1e9;
24          for (int i = 1; i <= n; i++) {
25              if (x[i] >= x[i - 1] + last) {
26                  num++;
27                  last = x[i];
28              }
29          }
30          if (num >= C) low = mid;
31          else high = mid;
32      }
33      cout << low << endl;
```

04. BOOKS1 (TungTT)

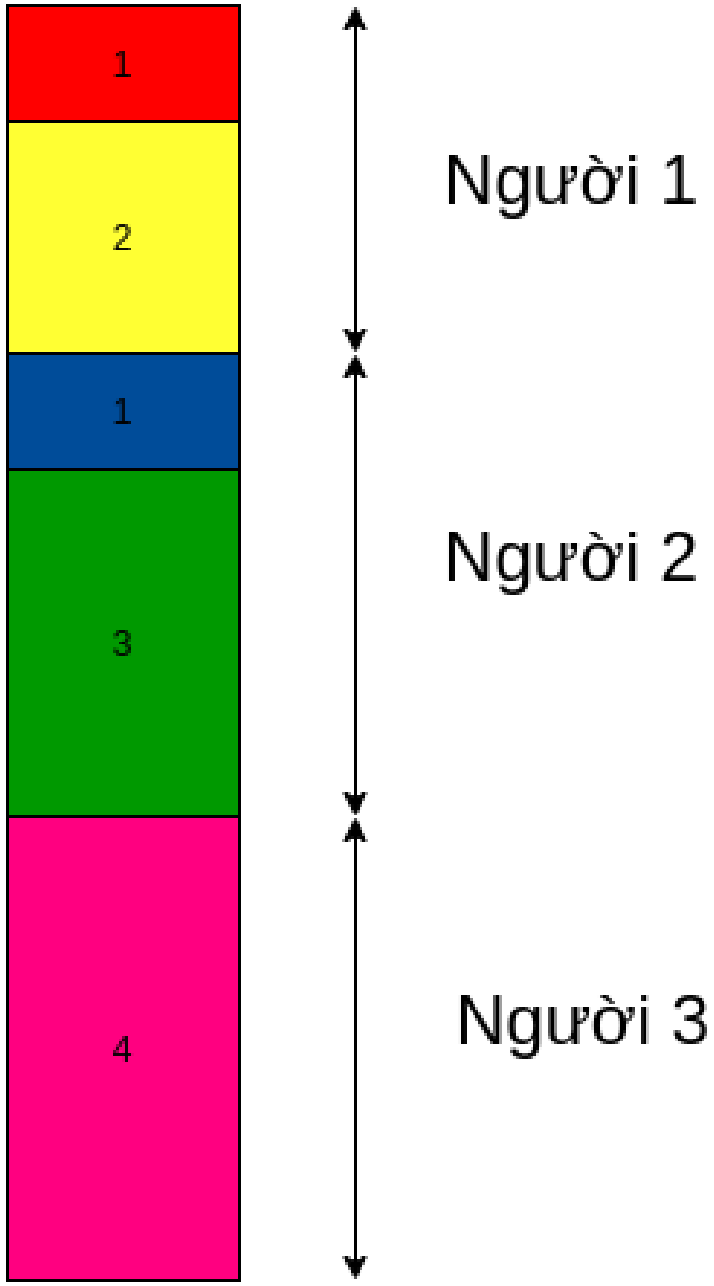
- Có m quyển sách, quyển sách thứ i dày p_i trang.
- Phải chia số sách trên cho đúng k người, mỗi người sẽ nhận được một đoạn sách liên tiếp nhau.
- In ra cách chia để số trang sách lớn nhất được nhận bởi một người là nhỏ nhất.
- Nếu có nhiều kết quả lớn nhất thì ưu tiên số sách nhận bởi người 1 là ít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

```
34 bool check(long long max_val) {
35     vector < int > pos;
36     long long sum = 0;
37     for (int i = n; i >= 1; i--) {
38         if (sum + a[i] <= max_val) {
39             sum += a[i];
40         } else {
41             sum = a[i];
42             if (a[i] > max_val) { return false; }
43             pos.push_back(i);
44         }
45     }
46     if (pos.size() >= k) { return false; }
47     In kq
48     return true;
49 }
```

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 $\leq k + 1$ đoạn với cách chia như cũ.
- Ta chặt nhị phân giá trị *maxVal*.
- Độ phức tạp thuật toán $O(\log MAX * n)$

Code 2

```
50 bool check(long long max_val) {
51     // Giong voi ham o Code 1
52 }
53 int main() {
54     int q; cin >> q;
55     while (q--) {
56         cin >> n >> k;
57         for (int i = 1; i <= n; i++) { cin >> a[i]; }
58         long long l = 0, r = MAX;
59         while (r - l > 1) {
60             long long mid = (l + r) >> 1;
61             if (check(mid)) {
62                 r = mid;
63             } else {
64                 l = mid;
65             }
66         }
67         ** In kq tuong ung voi gia tri r **
68     }
69 }
```

04. EKO (ngocbh)

- Cho n cái cây có chiều cao khác nhau $a_1, a_2, ...a_n$
- Có thể thực hiện một phát cắt độ cao h với tất cả các cây.
- Số lượng gỗ thu được là phần chóp của các cây cao hơn h.
- Tìm h nhỏ nhất có thể để số lượng gỗ thu được lớn hơn m.
- VD:
  - có 4 cây $20, 15, 10, 17$.
  - chọn $h = 15 \rightarrow$ số lượng gỗ thu được ở mỗi cây là $5, 0, 0, 2$. tổng là 7.
  - vậy ta thu được 7 mét gỗ.

Thuật toán

- **Thuật toán 1:** tìm tất cả các giá trị $h \in \{0, max(a[i])\}$. Với mỗi h, tính số lượng gỗ thu đượ. ĐPT: $O(max(a[i]) * n)$.
- **Thuật toán 2:** chặt nhị phân giá trị h.

Code

```
71      long long count_wood(int height) {
72          long long ret = 0;
73          for (int i = 1; i <= n; i++)
74              if ( a[i] > height )
75                  ret += a[i] - height;
76          return ret;
77      }
78
79      int l = 0, r = max(r,a[i]);
80
81      while (l < r-1) {
82          int mid = (l+r)/2;
83          if (count_wood(mid) >= m ) l = mid;
84          else r = mid;
85      }
86      cout << l;
```

04. FIBWORDS (vuongdx)

- Dãy Fibonacci Words của xâu nhị phân được định nghĩa như sau:

$$F(n) = \begin{cases} 0, & \text{if } n = 0 \\ 1, & \text{if } n = 1 \\ F(n-1) + F(n-2), & \text{if } n \geq 2 \end{cases}$$

- Cho n và một xâu nhị phân p. Đếm số lần p xuất hiện trong $F(n)$ (các lần xuất hiện này có thể chồng lên nhau).
- Giới hạn: $0 \leq n \leq 100$, p có không quá 100000 ký tự, kết quả không vượt quá 2^{63}.

Thuật toán I

- **Thuật toán 1 - Vét cạn:** So sánh xâu p với mọi xâu $f(n)[i..(i + len(p)]$.
- **Thuật toán 2 - Chia để trị:** Xâu $f(n)$ gồm 2 xâu con là $f(n-1)$ và $f(n-2)$.
  - Đếm số lần p xuất hiện trong $f(n-1)$, $f(n-2)$.
  - Đếm số lần p xuất hiện ở đoạn giữa của xâu $f(n)$ (đoạn đầu của p là đoạn cuối của $f(n-1)$, đoạn cuối của p là đoạn đầu của $f(n-2)$).

Thuật toán II

- Đếm số lần p xuất hiện trong $f(i)$ với i nhỏ: Sử dụng **thuật toán 1**.
- Đếm số lần p xuất hiện ở đoạn giữa xâu $f(n)$:
  - Giả sử 2 xâu $f(i-1)$ và $f(i)$ có độ dài lớn hơn độ dài xâu p, $f(i-1)$ có dạng $x..a$, $f(i)$ có dạng $y..b$, trong đó x, y, a, b có độ dài bằng độ dài của p (x và a hay y và b có thể chồng lên nhau).
  - **Nhận xét 1:** $x = y$.
  - **Nhận xét 2:** Nếu $n \equiv i (mod 2)$ thì đoạn giữa của $f(n)$ là $..ax..$, ngược lại, đoạn giữa của $f(n)$ là $..bx...$

Thuật toán III

- Cài đặt:
  - **void preprocessing()**: Tính trước các xâu fibonacci word, 2 xâu cuối cùng có độ dài không nhỏ hơn 10^5.
  - **long long count(string s, string p)**: Đếm số lần p xuất hiện trong s theo thuật toán 1.
  - **long long count(int n, string p)**: Đếm số lần p xuất hiện trong $f(n)$ theo thuật toán 2.
  - **long long solve(int n, string p)**:
    - Xử lý trường hợp $f(n)$ có độ dài nhỏ hơn độ dài của p.
    - Khởi tạo mảng c - $c[i]$ là số lần xuất hiện của p trong $f(i)$.
    - Sử dụng hàm count(s, p) để đếm số lần xuất hiện của p trong $f(i)$ và $f(i-1)$ với $f(i-1)$ là fibonacci word đầu tiên có độ dài không nhỏ hơn độ dài của p rồi lưu vào mảng c.
    - Sử dụng hàm count(s, p) để đếm số lần xuất hiện của p trong ax và bx, lưu vào mảng mc.
    - Sử dụng hàm count(n, p) để đếm số lần xuất hiện của p trong $f(n)$.

Code

```
87  long long solve(int n, string p) {
88      int lp = p.size();
89      if (n < n_prepare && l[n] < lp) {return 0;}
90      for (int j = 0; j <= n; j++) {c[j] = -1;}
91      int i = 1;
92      while (l[i - 1] < lp) {i++;}
93      c[i - 1] = count(f[i - 1], p);
94      c[i] = count(f[i], p);
95      string x = f[i].substr(0, lp - 1);
96      string a =
97      f[i - 1].substr(f[i - 1].size() - (lp - 1));
98      string b =
99      f[i].substr(f[i].size() - (lp - 1));
100     mc[i % 2] = count(a + x, p);
101     mc[(i + 1) % 2] = count(b + x, p);
102     return count(n, p);
103 }
```

Code

```
104 long long count(int n, string p) {
105     if (c[n] < 0) {
106         c[n] = count(n - 1, p)
107         + count(n - 2, p)
108         + mc[n % 2];
109     }
110     return c[n];
111 }
```

04. CLOPAIR ()