Thuật toán ứng dụng
Bài thực hành số 3

Giảng viên: TS. Đinh Viết Sang
Trợ giảng: Nguyễn Trung Hiếu
Email: hieu.nt164813@sis.hust.edu.vn

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

19/04/2020

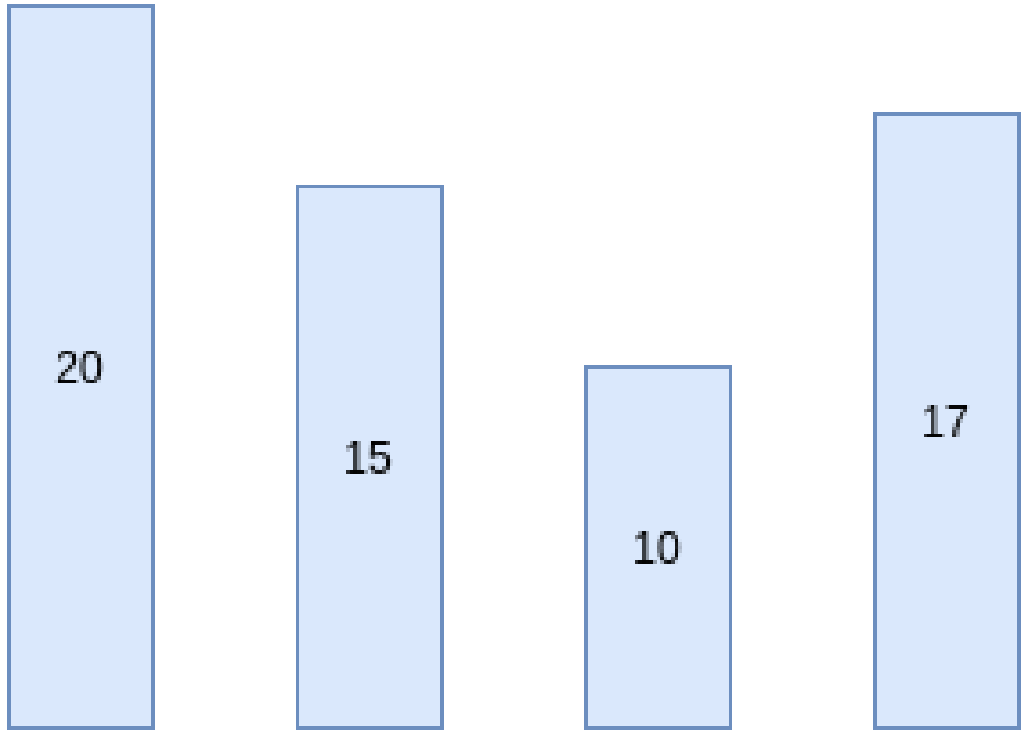
Nội dung

EKO

- 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 lớn nhất có thể để số lượng gỗ thu được tối thiểu là m.

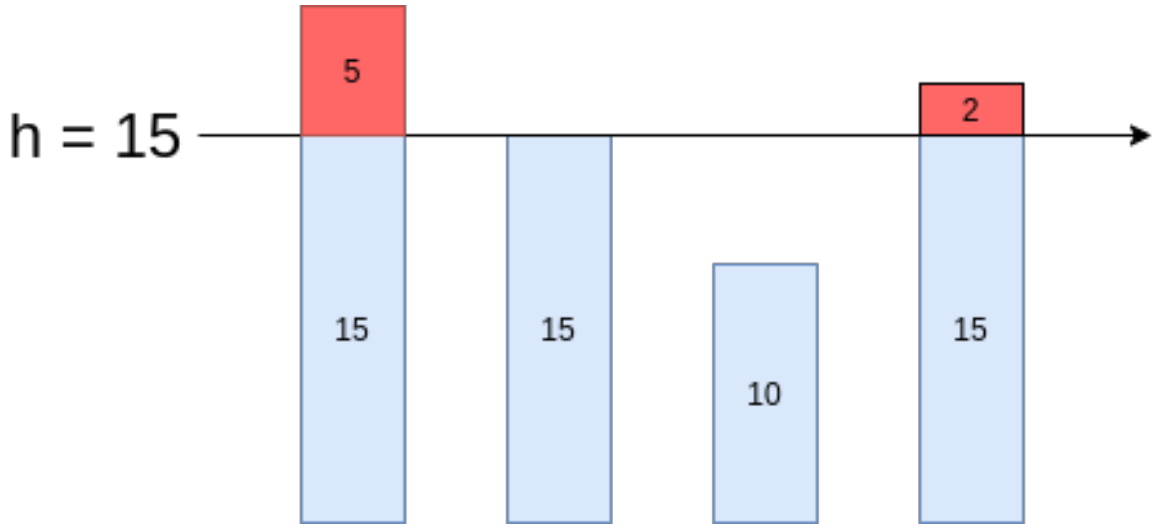
Ví dụ

- Có 4 cây $20, 15, 10, 17$, lượng gỗ tối thiểu cần cắt là 7.



Ví dụ

- Chọn $h = 15 \rightarrow$ số lượng gỗ thu được ở mỗi cây là $5, 0, 0, 2$. Tổng lượng gỗ thu được là 7.
- Vậy chiều cao lớn nhất có thể cắt được là 15



Thuật toán

- **Thuật toán 1:** Duyệt tất cả các giá trị $h \in \{0, max(a_i)\}$.
  Độ phức tạp: $O(max(a_i) \times n)$.
- **Thuật toán 2:** Tìm kiếm nhị phân giá trị h.
  Độ phức tạp $O(log(max(a_i)) \times n)$

Code

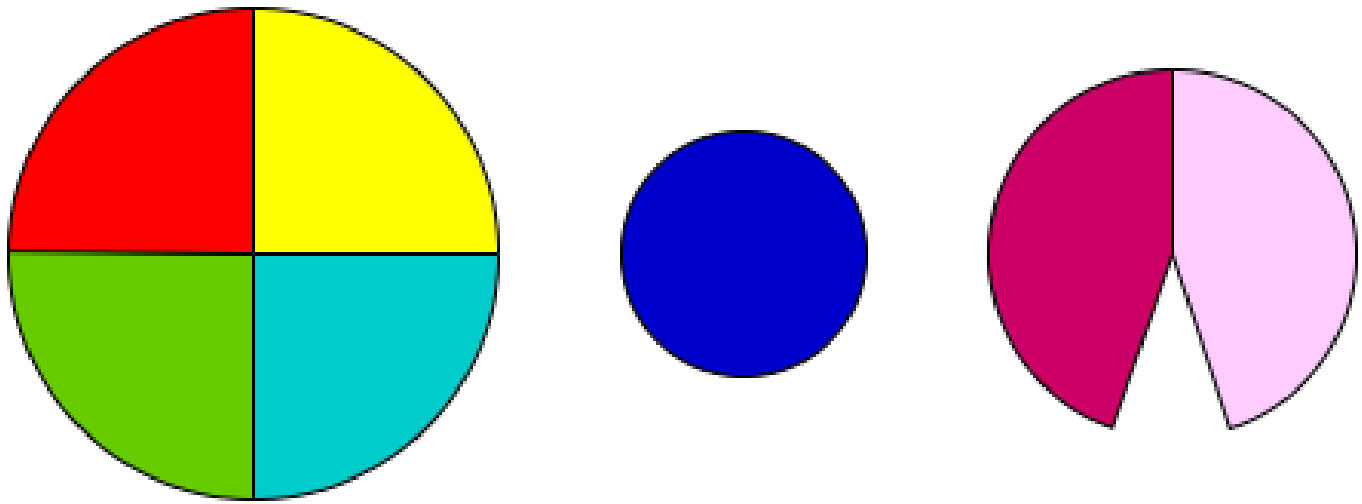
```
long long count_wood(int height) {
    long long sum = 0;
    for (int i = 1; i <= n; i++)
        if ( a[i] > height )
            sum += a[i] - height;
    return sum;
}

int main {
    int l = 0, r = max(r,a[i]);

    while (r - l > 1) {
        int mid = (l + r) / 2;
        if (count_wood(mid) >= m ) l = mid;
        else r = mid;
    }
    cout << l;
}
```

PIE

- Có N cái bánh và $F + 1$ người.
- Bánh thứ i có bán kính r_i 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.

Ví dụ

- Giả sử có 3 cái bánh có bán kính lần lượt là $2; 1; 1.5$
- Cần chia 3 cái bánh trên cho 7 người
- Cách chia sau là tối ưu, mỗi người nhận được một phần là π



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

```
19
20  int count_pie(int m) {
21      int cnt = 0;
22
23      for(int i = 1; i <= n; i++)
24          cnt += (int)floor(PI * r[i] * r[i] / m);
25      return cnt;
26  }
27
28  int main() {
29      double l = 0, r = 4e8;
30      double eps = 0.000001;
31      while(r - l > eps)
32          double m = (l + r) / 2;
33          if(count_pie(m) > F) l = m;
34          else r = m;
35      }
36  }
```

FIBWORDS

- 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}.

Example

standard input	standard output
6 10	Case 1: 5

- $F_6 = 10_1110_210_3110_4110_5$

Thuật toán trực tiếp

- Sinh xâu $F(n)$ ($O(n)$).
- Duyệt qua toàn bộ vị trí i của xâu F_n. Coi vị trí i là vị trí bắt đầu của xâu p. So khớp xâu p với xâu $F_n[i, i + len(p) - 1]$.
- ĐPT: $O(len(F_n) * len(p))$

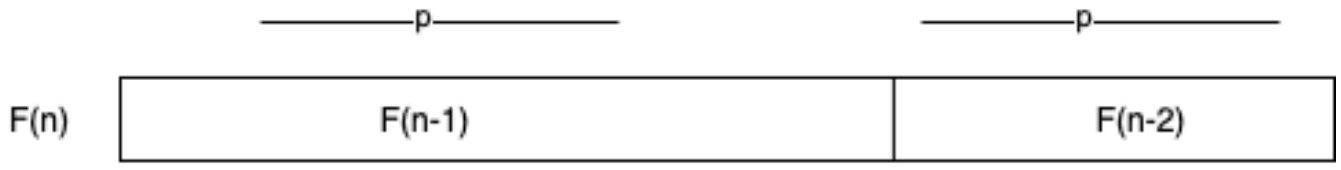
Thuật toán trực tiếp

- Sinh xâu $F(n)$ ($O(n)$).
- Duyệt qua toàn bộ vị trí i của xâu F_n. Coi vị trí i là vị trí bắt đầu của xâu p. So khớp xâu p với xâu $F_n[i, i + len(p) - 1]$.
- ĐPT: $O(len(F_n) * len(p))$
- Vấn đề:
  - $n \to 100 \Rightarrow len(F_n) \to 354224848179261915075$
  - Không đủ bộ nhớ để lưu.
  - Quá thời gian.

Cải tiến

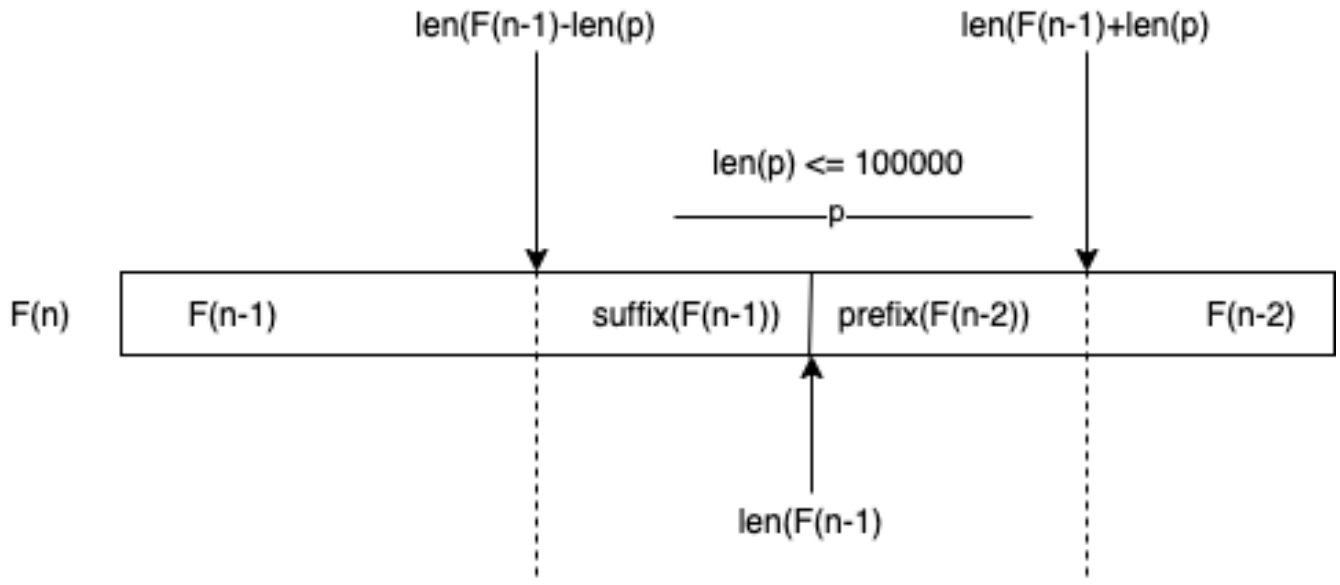
- Nhận xét: Phân loại 3 trường hợp p xuất hiện trong F_n:
  - TH1: Nằm hoàn toàn trong F_{n-1}.
  - TH2: Nằm hoàn toàn trong F_{n-2}.
  - TH3: Nằm ở giữa nửa đầu trong F_{n-1} và nửa sau trong F_{n-2}.

Cải tiến



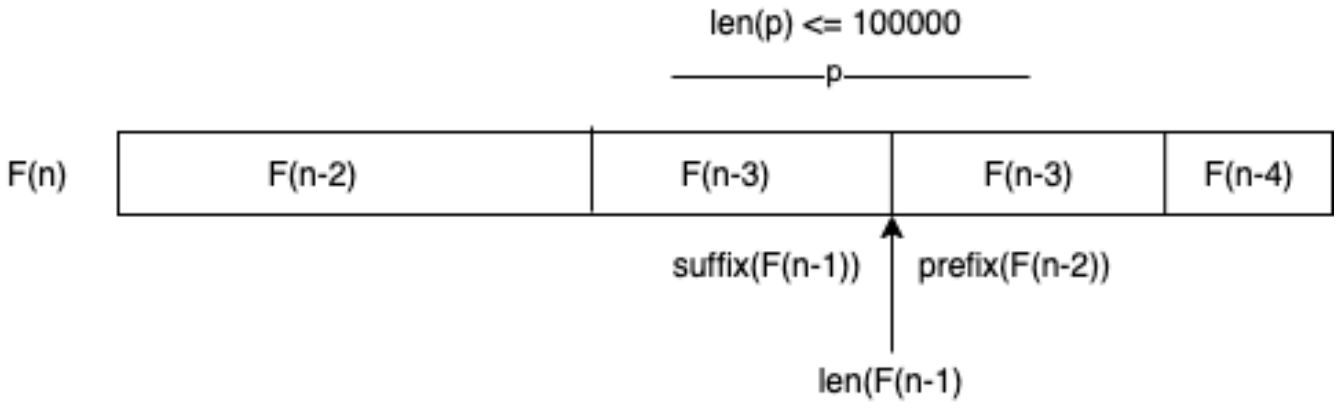
- Tính số lượng p xuất hiện trong trường hợp 1 (F_{n-1}) và 2 (F_{n-2}).
- $\rightarrow$ Một bài toán con của bài toán tính số lần xuất hiện p trong F_n. Ta có thể giả sử đã giải quyết các bài toán con này rồi. Sau đó nếu tìm được công thức tính tổng quát cho F_n. Ta có thể quay lui để áp dụng công thức đó trên F_{n-1} và F_{n-2}

Cải tiến



- Trong trường hợp này p sẽ nằm trong khoảng $(len(F_{n-1}) - len(p), len(F_{n-1}) + len(p))$.
- Do vậy điểm đầu của p chỉ phải xét từ $len(F_{n-1}) - len(p) + 1 \rightarrow len(F_{n-1}) - 1$.
- Câu hỏi là làm sao để ta xây dựng được khoảng $(len(F_{n-1}) - len(p), len(F_{n-1}) + len(p))$

Cải tiến



- $suffix(F_{n-1}) = suffix(F_{n-3}) = suffix(F_{n-5}) = ...$
- $prefix(F_{n-2}) = prefix(F_{n-3}) = prefix(F_{n-4}) = ...$
- Chúng ta chỉ cần lưu 2 suffix (một cho n lẻ một cho n chẵn) có độ dài lớn hơn p và một prefix có độ dài lớn hơn p.

Cải tiến

Độ phức tạp thuật toán

- TH3 ta mất $O(p^2)$ để tính tất cả các trường hợp.
- Để tính F_n ta cần tính $F_{n-1}, F_{n-2}, ...F_1$. Nếu ta dùng mảng lưu lại giá trị tính được thì đpt sẽ là $O(n)$.
- $\rightarrow$ ĐPT tổng thể là : $O(n * p^2)$.
- Với $n \leq 100, |q| \leq 10^5$ thì phương pháp này chưa đủ để full. Nhưng với mục đích training thì $O(n * p^2)$ đủ để full bài trên Codeforces.

Cải tiến

- Để full cả trên lý thuyết, chúng ta cần tính nhanh số lần xuất hiện của p trong trường hợp 3.
- Đưa độ phức tạp bài toán về $O(n * p)$.
- Gợi ý: KMP algorithm, Z algorithm.

Code

```
37 long long calc(int n, string p)
38 {
39     if (n < M) {
40     return count_p(fibo[n], p);
41     }
42     long long ret = dp[n];
43     if (ret != -1)
44         return ret;
45
46     ret = calc(n - 1, p) + calc(n - 2, p);
47     int l = p.size();
48     string mid = suffix(fibo[M + (n % 2)], l - 1)
49     + prefix(fibo[M], l - 1);
50     ret += count_p(mid ,p);
51     dp[n] = ret;
52     return ret;
53 }
```

Code

```
54 long long count_p(string st, string p) {
55   long long ans = 0;
56   for (int i = 0; i + p.size() - 1 < st.size(); i++) {
57     bool ok = true;
58     for (int j = 0; j < p.size(); j++)
59       if (p[j] != st[i + j]){
60         ok = false;
61         break;
62       }
63       ans += ok ? 1 : 0;
64   }
65   return ans;
66 }
67 string suffix(string st, int l) {
68   return st.substr(st.size() - l);
69 }
70 string prefix(string st, int l) {
71   return st.substr(0, l);
72 }
```

Code

```
73  const int M = 25;
74
75  string fibo[M + 5];
76  long long dp[105];
77
78  int main(){
79    int n;
80    string p;
81    int ntest = 1;
82    while (cin >> n >> p) {
83      fibo[0] = "0";
84      fibo[1] = "1";
85      for(int i = 2; i <= M + 1; i++)
86          fibo[i] = fibo[i - 1] + fibo[i - 2];
87      memset(dp, -1, sizeof(dp));
88      cout << "Case " << ntest << ": "
89          << calc(n, p) << endl;
90      ntest++;
91    }
92    return 0;
```