Thuật toán ứng dụng
Bài thực hành số 1

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

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

15/03/2021

Nội dung

01. ALICEADD

- Cho hai số a và b, hãy viết chương trình bằng C/C++ tính số $c = a + b$
- Lưu ý giới hạn: $a, b < 10^{19}$ dẫn đến c có thể vượt quá khai báo `long long`

Thuật toán

- Chỉ cần khai báo a, b, c kiểu `unsigned long long`, trường hợp tràn số chỉ xảy ra khi a, b có 19 chữ số và c có 20 chữ số

1. Tách $a = a_1 \times 10 + a_0$
2. Tách $b = b_1 \times 10 + b_0$
3. Tách $a_0 + b_0 = c_1 \times 10 + c_0$
4. In ra liên tiếp $a_1 + b_1 + c_1$ và c_0

Code

```
int main() {
    unsigned long long a, b, a0, b0, a1, b1, c0, c1;
    cin >> a >> b;
    a0 = a % 10;
    a1 = (a - a0) / 10;
    b0 = b % 10;
    b1 = (b - b0) / 10;
    c0 = (a0 + b0) % 10;
    c1 = (a0 + b0 - c0) / 10;
    c1 = a1 + b1 + c1;
    if (c1 > 0) cout << c1;
    cout << c0;
    return 0;
}
```

01. SUBSEQMAX

- Cho dãy số $s = \langle a_1, \ldots, a_n \rangle$
- một dãy con từ i đến j là $s(i,j) = \langle a_i, \ldots, a_j \rangle$, $1 \leq i \leq j \leq n$
- với trọng số $w(s(i,j)) = \sum_{k=i}^{j} a_k$
- Yêu cầu: tìm dãy con có trọng số lớn nhất

Ví dụ

- dãy số: -2, 11, -4, 13, -5, 2
- Dãy con có trọng số cực đại là 11, -4, 13 có trọng số 20

Có bao nhiêu dãy con?

- Số lượng cặp (i,j) với $1 \leq i \leq j \leq n$
- $\frac{n(n+1)}{2}$
- Thuật toán trực tiếp!

Thuật toán trực tiếp

- Duyệt qua tất cả $\frac{n^2+n}{2}$ dãy con
- Với mỗi dãy con ta tính tổng của dãy
- Lưu lại tổng lớn nhất
- Độ phức tạp: $O(n^3)$

```
int algo1(int a[], int n) {
  int maxs = a[0];
  for(int i = 1; i <= n; i++){
    for(int j = i; j <= n; j++){
      int s = 0;
      for(int k = i; k <= j; k++)
        s = s + a[k];
      maxs = maxs < s ? s : maxs;
    }
  }
  return maxs;
}
```

Thuật toán tốt hơn

- Quan sát: $\sum_{k=i}^{j} a[k] = a[j] + \sum_{k=i}^{j-1} a[k]$
- Độ phức tạp: $O(n^2)$

```
int algo2(int[] a, int n){
  int maxs = a[0];
  for(int i = 1; i <= n; i++){
    int s = 0;
    for(int j = i; j <= n; j++){
      s = s + a[j];
      maxs = maxs < s ? s : maxs;
    }
  }
  return maxs;
}
```

Thuật toán Quy hoạch động

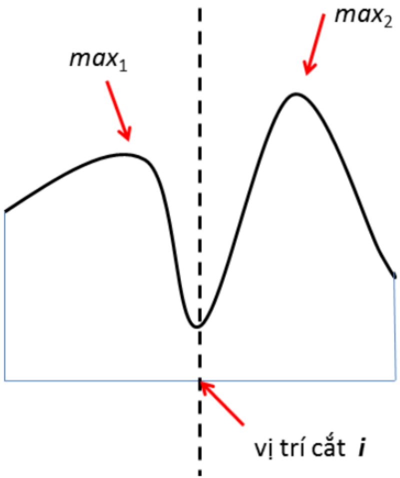
- Thiết kế hàm tối ưu:
  - Đặt s_i là trọng số của dãy con có trọng số cực đại của dãy $a_1, \ldots, a_i$ mà kết thúc tại a_i
- Công thức Quy hoạch động:
  - $s_1 = a_1$
  - $s_i = \max\{s_{i-1}, 0\} + a_i, \forall i = 2, \ldots, n$
  - Đáp án là $\max\{s_1, \ldots, s_n\}$
- Độ phức tạp thuật toán: $O(n)$ (thuật toán tốt nhất!)

Code

```
int algo3(int a[], int n){
  int maxs = a[1];
  s[1] = a[1];
  maxs = s[1];
  for(int i = 2; i <= n; i++) {
    // tinh s[i]
    s[i] = max(s[i-1], 0) + a[i];
    maxs = maxs > s[i] ? maxs : s[i];
  }
  return maxs;
}
```

02. SIGNAL

- Cho một dãy tín hiệu độ dài n có độ lớn lần lượt là $a_1, a_2, ..., a_n$ và một giá trị phân tách b.
- Một tín hiệu được gọi là phân tách được khi tồn tại một vị trí i $(1 < i < n)$ sao cho $max\{a_1, .., a_{i-1}\} - a_i \geq b$ và $max\{a_{i+1}, .., a_n\} - a_i \geq b$
- Tìm vị trí i phân tách được sao cho $max\{a_1, .., a_{i-1}\} - a_i + max\{a_{i+1}, .., a_n\} - a_i$ đạt giá trị lớn nhất.
- In ra giá trị lớn nhất đó. Nếu không tồn tại vị trí phân tách được thì in ra giá trị -1.


max_1
max_2
vị trí cắt *i*

Thuật toán

- Chuẩn bị mảng $maxPrefix[i] = max\{a_1, .., a_i\}$.
- Chuẩn bị mảng $maxSuffix[i] = max\{a_i, .., a_n\}$
- Duyệt qua hết tất cả các vị trí i $(1 < i < n)$. Với mỗi vị trí kiểm tra xem liệu đó có phải là vị trí phân tách được hay không bằng cách kiểm tra $maxPrefix[i-1] - a[i] >= b$ và $maxSuffix[i+1] - a[i] >= b$.
- Lấy max của giá trị $maxPrefix[i-1] - a_i + maxSuffix[i+1] - a_i$ tại các vị trí i thoả mãn.
- Độ phức tạp thuật toán O(n).

Code

```
//tinh maxPre[]
maxPre[1] = a[1];
for(int i = 2; i <= n; i++)
  maxPre[i] = max(maxPre[i - 1], a[i]);
//tinh maxSuf[]
maxSuf[n] = a[n];
for(int i = n - 1; i >= 1; i--)
  maxSuf[i] = max(maxSuf[i + 1], a[i]);

result = -1;
for(int i = 2; i < n; i++){
  int valPre = maxPre[i - 1] - a[i];
  int valSub = maxSuf[i + 1] - a[i];
  if(valPre >= b && valSub >= b) {
    result = max(result, valPre + valSub);
  }
}
```

02. REROAD

- Cho N đoạn đường, đoạn thứ i có loại nhựa đường là t_i.
- Định nghĩa một phần đường là một dãy liên tục các đoạn đường được phủ cùng loại nhựa phủ t_k và bên trái và bên phải phần đường đó là các đoạn đường (nếu tồn tại) được phủ loại nhựa khác.
- Độ gập ghềnh của đường bằng tổng số lượng phần đường.
- Mỗi thông báo bao gồm 2 số là số thứ tự đoạn đường được sửa và mã loại nhựa được phủ mới.
- Sau mỗi thông báo, cần tính độ gập ghềnh của mặt đường hiện tại.

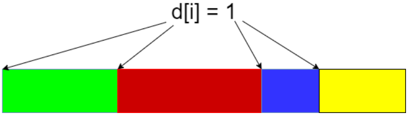
Ví dụ

Đoạn đường ban đầu với độ gập ghềnh là 4

Đoạn đường sau khi update với độ gập ghềnh là 6

Thuật toán

- Gọi d[i] là mảng nhận giá trị 1 nếu $a[i] \neq a[i-1]$ và giá trị 0 trong trường hợp ngược lại
- Nhận thấy mỗi phần đường có một và chỉ một phần tử bắt đầu, số lượng phần đường (hay độ gập ghềnh) chính là số lượng phần tử bắt đầu.
- Nói cách khác thì độ gập ghềnh $= \sum_{i=1}^{n} d[i]$
- Nhận thấy với mỗi lần đổi 1 phần tử i trong mảng a thì ta chỉ thay đổi giá trị của nhiều nhất là 2 phần tử trong mảng d đó là $d[i]$ và $d[i+1]$
- Độ phức tạp thuật toán O(n).



Code

```
int start(int u) {
    if (u == 1) return 1;
    return a[u] != a[u - 1];
}
int main() {
    cin >> n;
    for (int i = 1; i <= n; i++) cin >> a[i];
    int ans = 0;
    for (int i = 1; i <= n; i++) ans += start(i);
    cin >> q;
    for (int i = 1; i <= q; i++) {
        cin >> p >> c;
        ans -= start(p);
        if (p < n) ans -= start(p + 1);
        a[p] = c;
        ans += start(p);
        if (p < n) ans += start(p + 1);
        cout << ans << endl;
    }
}
```