Training 1: Introduction and Data Structure

Hanoi University of Science and Technology

Hanoi, 2020

Outline

Outline

01. ADD

- 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 $a = ha \times 10 + ta$
2. Tách $b = hb \times 10 + tb$
3. Tách $r = hr \times 10 + tr$
4. $tr = (ta + tb) \quad mod \quad 10$
5. $hr = (ha + hb) + (ta + tb)/10$
6. In ra liên tiếp hr và tr

Code

```cpp
int main()
{
  // freopen("in.txt","r",stdin);
  unsigned long long a, b, ha, hb, ta, tb, hr, tr;
  cin >> a;
  cin >> b;
  ha = a / 10, ta = a % 10;
  hb = b / 10, tb = b % 10;
  tr = (ta + tb) % 10;
  hr = ha + hb + (ta + tb)/10;
  if ( hr != 0 ) cout << hr;
  cout << tr;


  return 0;
}
```

Outline

01. SUBSEQMAX (Thuận)

- 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
- `http://www.spoj.com/problems/MAXSUMSU/`

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 $1 \leq i \leq j \leq n$
- $\binom{n}{2} + n$
- Thuật toán trực tiếp!

Thuật toán trực tiếp — $\mathcal{O}(n^3)$

- Duyệt qua tất cả $\binom{n}{2} + n = \frac{n^2+n}{2}$ dãy con

```
public long algo1(int[] a){
  int n = a.length;
  long max = a[0];
  for(int i = 0; i < n; i++){
    for(int j = i; j < n; j++){
      int s = 0;
      for(int k = i; k <= j; k++)
        s = s + a[k];
      max = max < s ? s : max;
    }
  }
  return max;
}
```

Thuật toán tốt hơn — $\mathcal{O}(n^2)$

- Quan sát: $\sum_{k=i}^{j} a[k] = a[j] + \sum_{k=i}^{j-1} a[k]$

```
public long algo2(int[] a){
  int n = a.length;
  long max = a[0];
  for(int i = 0; i < n; i++){
    int s = 0;
    for(int j = i; j < n; j++){
      s = s + a[j];
      max = max < s ? s : max;
    }
  }
  return max;
}
```

Thuật toán Chia để trị

- Chia dãy thành 2 dãy con tại điểm giữa $s = s_1 :: s_2$
- Dãy con có trọng số cực đại có thể
  - nằm trong s_1 hoặc
  - nằm trong s_2 hoặc
  - bắt đầu tại một vị trí trong s_1 và kết thúc trong s_2
- Code Java:

```
private long maxSeq(int i, int j){
  if(i == j) return a[i];
  int m = (i+j)/2;
  long ml = maxSeq(i,m);
  long mr = maxSeq(m+1,j);
  long maxL = maxLeft(i,m);
  long maxR = maxRight(m+1,j);
  long maxLR = maxL + maxR;
  long max = ml > mr ? ml : mr;
  max = max > maxLR ? max : maxLR;
  return max;
}
public long algo3(int[] a){
  int n = a.length;
  return maxSeq(0,n-1);
```

Chia để trị — $\mathcal{O}(n \log n)$

```
private long maxLeft(int i, int j){
  long maxL = a[j];
  int s = 0;
  for(int k = j; k >= i; k--){
    s += a[k];
    maxL = maxL > s ? maxL : s;
  }
  return maxL;
}
private long maxRight(int i, int j){
  long maxR = a[i];
  int s = 0;
  for(int k = i; k <= j; k++){
    s += a[k];
    maxR = maxR > s ? maxR : s;
  }
  return maxR;
}
```

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:
  - $s_1 = a_1$
  - $s_i = \max\{s_{i-1} + a_i, a_i\}, \forall i = 2, \ldots, n$
  - Đáp án là $\max\{s_1, \ldots, s_n\}$
- Độ phức tạp thuật toán là n (thuật toán tốt nhất!)

Quy hoạch động — $\mathcal{O}(n)$

```java
public long algo4(int[] a){
  int n = a.length;
  long max = a[0];
  int[] s = new int[n];
  s[0] = a[0];
  max = s[0];
  for(int i = 1; i < n; i++){
    if(s[i-1] > 0) s[i] = s[i-1] + a[i];
    else s[i] = a[i];
    max = max > s[i] ? max : s[i];
  }
  return max;
}
```

Thuật toán sử dụng kỹ thuật mảng tiền tố

- $w(i,j) = \sum_{k=i}^{j} a_k$
- Đặt tổng tiền tố (prefix sum) $ps(i) = \sum_{k=1}^{i}$
- $w(i,j) = ps(j) - ps(i-1)$
- Bài toán con j:
  - cho mảng $a_1, ...a_j$
  - tìm i để $w(i,j)$ đạt giá trị lớn nhất.
  - $max_i\{w(i,j)\} = max_{i=1}^{j-1}\{ps(j) - ps(i-1)\} = ps(j) - min_{i=1}^{j-1}\{ps(i-1)\}$

Thuật toán sử dụng kỹ thuật mảng tiền tố

Giải bài toán con j bằng bài toán con $j-1$

- Giả sử lời giải bài toán con $j-1$ đã biết
  $\rightarrow res(j-1) = ps(j-1) - min_{i=1}^{j-2}\{ps(i-1)\}$
- Cần tính $res(j)$ từ $res(j-1)$
- $res(j) = ps(j-1) + a_j - min(min_{i=1}^{j-2}\{ps(i-1)\}, a_{j-1})$

Mảng tiền tố — $\mathcal{O}(n)$

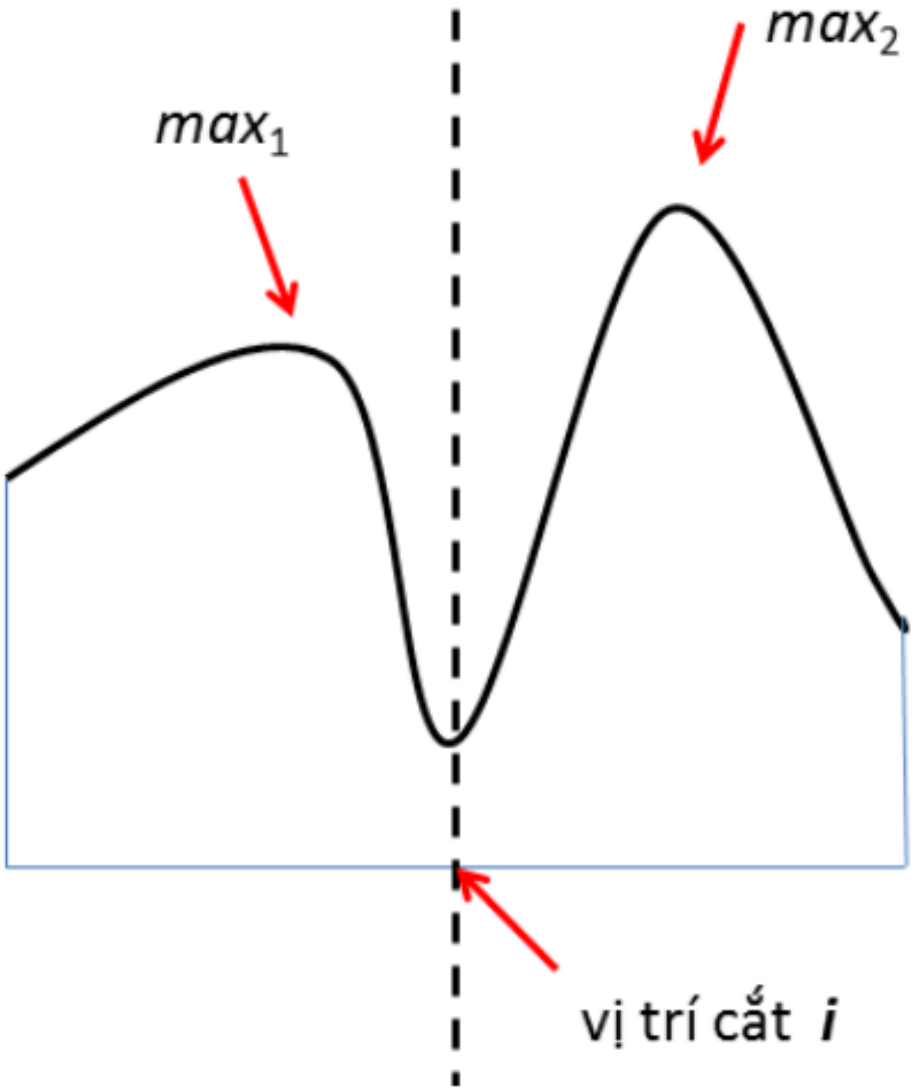
```
int ans = -999999999;
int psj = 0, min_psj = 0;

for(int j = 0; j < n; j++) {
  psj += a[j];
  ans = max(ans, ps - min_psj);
  min_psj = min(min_psj, ps);
}
cout << ans;
```

Outline

02. SIGNAL (TungTT)

- 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_2
max_1
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).

Cài đặt

```
int main() {
  const int MAX = 1e9 + 5;
  int n, b;

  cin >> n >> b;
  vector < int > a(n + 2, 0);
  vector < int > max_prefix(n + 2, 0);
  vector < int > max_suffix(n + 2, 0);
  for (int i = 1; i <= n; i++) {
    cin >> a[i];
  }

  // khoi tao gia tri max o bien
  max_prefix[0] = -MAX; max_suffix[n + 1] = -MAX;

  // tinh max_prefix
  for (int i = 1; i <= n; i++) {
    max_prefix[i] = max(max_prefix[i - 1], a[i]);
  }
```

Cài đặt

```
    // tinh max_suffix
  for (int i = n; i >= 1; i--) {
    max_suffix[i] = max(max_suffix[i + 1], a[i]);
  }

  // tinh ket qua
  int ans = -1;
  for (int i = 2; i < n; i++) {
    // Kiem tra vi tri i
    if (max_prefix[i - 1] - a[i] >= b &&
        max_suffix[i + 1] - a[i] >= b) {
      // lay ket qua
      ans = max(ans, max_prefix[i - 1] - a[i] +
                     max_suffix[i + 1] - a[i]);
    }
  }
  cout << ans << endl;
```

Outline

02. REROAD (QuangLM)

- 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.

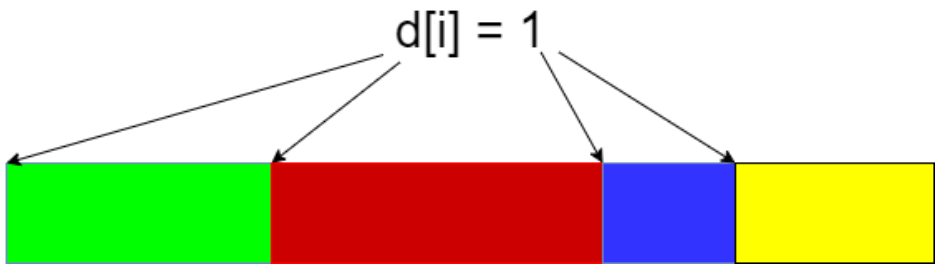
Ví dụ

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 $d[i+1]$



Code

```
int main()
{
  // freopen("in.txt","r",stdin);

  cin >> n;
  t[0] = t[n+1] = INF, res = 0;
  for (int i = 1; i <= n; i++)
    cin >> t[i], res += (t[i] == t[i-1]) ? 0 : 1;

  cin >> q;
  for (int i = 1; i <= q; i++) {
    cin >> p >> c;
    res += (t[p-1] == t[p]) + (t[p] == t[p+1]);
    res -= (t[p-1] == c) + (c == t[p+1]);
    t[p] = c;
    cout << res << endl;
  }
  return 0;
}
```

Outline

02. LOCATE

- Cho T test, mỗi test gồm 2 bản đồ kích thước LxC, thể hiện cùng một địa điểm tại 2 thời điểm khác nhau.
- Mỗi bản đồ biểu diễn bởi các số 0, 1. 1 ứng với vị trí có vật thể bay (có thể là chim hoặc chiến đấu cơ), 0 là vị trí không có vật thể nào.
- Biết rằng tất cả các chiến đấu cơ trên bản đồ di chuyển theo cùng một quy luật.
- Tính số chiến đấu cơ tối đa có thể xuất hiện trong cả hai bản đồ.
- Biết rằng: $1 \leq L, C \leq 1000$. Tổng số các số 1 không quá 10000.

Thuật toán

- Tổng số các số 1 không quá 10000 nên có thể lưu toạ độ các đỉnh 1 của mỗi trạng thái vào 2 mảng.
- So sánh từng cặp đỉnh của mỗi mảng để đếm số khoảng cách có thể.

Code

```
vector<ii> rada1,rada2;
int cur[2*N][2*N]; // thay vi goi cur[i][j] ta goi cur[i+N][j+N];
int main()
{
  int t;
  cin >> t;
  while ( t-- ) {
    cin >> L >> C;
    rada1.clear(), rada2.clear();
    for (int i = 1; i <= L; i++)
      for (int j = 1; j <= C; j++){
        cin >> u;
        if ( u ) rada1.push_back(ii(i,j));
      }
    for (int i = 1; i <= L; i++)
      for (int j = 1; j <= C; j++) {
        cin >> u;
        if ( u ) rada2.push_back(ii(i,j));
      }
    memset(cur,0,sizeof(cur));
    for (auto e1 : rada1) {
      for (auto e2: rada2)
        cur[e1.first - e2.first + N][e1.second - e2.second + N]++;
    }
    int res = 0;
        for (int i = 0; i < 2*N; i++)
            for (int j = 0; j < 2*N; j++)
                res = max(res,cur[i][j]);
        cout << res << endl;
  }
}
```

Outline

02. POSTMAN (HieuNT)

- Một nhân viên giao hàng cần nhận các kiện hàng tại trụ sở công ty ở vị trí $x = 0$, và chuyển phát hàng đến n khách hàng, được đánh số từ 1 đến n.
- Người khách thứ i ở vị trí x_i và cần nhận m_i kiện hàng.
- Nhân viên giao hàng chỉ có thể mang theo tối đa k kiện hàng mỗi lần.
- Nhân viên giao hàng xuất phát từ trụ sở, nhận một số kiện hàng và di chuyển theo đại lộ để chuyển phát cho một số khách hàng. Khi giao hết các kiện hàng mang theo, nhân viên lại quay trở về trụ sở và lặp lại công việc nói trên cho đến khi chuyển phát hết tất cả các kiện hàng.
- Sau khi giao xong, nhân viên cần quay lại công ty để nộp hóa đơn của ngày hôm đó.
- Giả thiết là: tốc độ di chuyển là 1 đơn vị khoảng cách trên một đơn vị thời gian. Thời gian nhận hàng ở trụ sở công ty và thời gian bàn giao hàng cho khách được coi là bằng 0.
- Giả sử thời điểm nhân viên giao hàng bắt đầu công việc là 0.
- Tìm cách hoàn thành công việc tại thời điểm sớm nhất.

Thuật toán

- Nhận xét: Vì công ty nằm ở vị trí $x = 0$ và thời gian nhận hàng ở công ty bằng 0 nên ta có thể chia khách hàng thành 2 tập: $x < 0$ và $x > 0$. Kết quả bằng tổng thời gian chuyển trong 2 tập
- Thuật toán
  1. Phân chia khách hàng thành 2 tập: $x < 0$ và $x > 0$.
  2. Với mỗi tập khách hàng, ta sắp xếp các khách hàng theo khoảng cách từ vị trí của họ đến trụ sở công ty.
  3. Nhân viên giao hàng sẽ phát từ khách hàng xa nhất trong tập, nếu còn dư số kiện hàng sẽ phát tiếp cho khách hàng liền kề đó.
- Độ phức tạp: $O(n)$

Code

```
long long calSegment(pair<int, int> p[], int np)
{
  long long res = 0;
  int cur = 0;
  for(int i = 1; i <= np; i++) {
    if(p[i].second > 0) {
      if(cur >= p[i].second){
        // Du so kien de phat
        cur -= p[i].second;
      } else {
        // Khong du so kien de phat
        p[i].second -= cur;
        int times = (p[i].second - 1)/ k + 1;
        res += 2ll * abs(p[i].first) * times;
        cur = times * k - p[i].second;
      }
    }
  }
  return res;
```

Code

```
const int N = 1002;
typedef pair<int, int> pii;
int n, nn, np, k, x, m;
long long ans = 0;
pii negCus[N], posCus[N];

int main()
{
   cin >> n >> k;
   nn = np = 0;
   for(int i = 1; i <= n; i++) {
      cin >> x >> m;
      // Chia thanh 2 tap khach hang
      if(x < 0) negCus[++nn] = make_pair(x, m);
      else posCus[++np] = make_pair(x, m);
}
   ...
```

Code

```
    ...
    // Sap xep khach hang trong tap theo khoang cach
    sort(negCus + 1, negCus + nn + 1);
    sort(posCus + 1, posCus + np + 1, greater<pii>());

    // Tinh khoang thoi gian nho nhat voi moi tap
    long long negSeg = calSegment(negCus, nn, k);
    long long posSeg = calSegment(posCus, np, k);
    ans = negSeg + posSeg;
    cout << ans;
    return 0;
}
```

Outline

02. HIST (DucLA)

- Có N cột với độ cao l_1, l_2, ...
- Tìm diện tích hình chữ nhật lớn nhất nằm gọn trong N cột này

Thuật toán

- Nhận xét: Một hình chữ nhật với hai biên là các cột thứ i và j có diện tích là $(j - i + 1) * min(l_i, ..., l_j)$
- Dễ dàng nhận thấy thuật toán $O(N^3)$: thử hết các cặp i và j và tính min 1 đoạn trong $O(N)$
- Nhận xét $min(l_i, ..., l_j) = min(min(l_i, ..., l_{j-1}), l_j)$
- Vậy min đoạn i đến j có thể cập nhật trong $O(1)$ khi i giữ nguyên và j tăng lên 1, ta có thuật toán $O(N^2)$

Thuật toán

- Góc nhìn khác: thay vì đi thử các bộ i và j, ta thử các chiều cao của hình chữ nhật, tức là nhân tử $min(l_i, ..., l_j)$
- Cụ thể: với mỗi cột i, ta xét xem nếu nó là cột có độ cao thấp nhất của một hình chữ nhật nào đó, thì chiều rộng của hình chữ nhật đó tối đa là bao nhiêu
- Ta đi tính các giá trị $left_i$ và $right_i$. Trong đó $left_i$ là vị trí của cột gần nhất bên trái i mà có độ cao nhỏ hơn cột i, tương tự đối với $right_i$ nhưng là ở bên phải
- Kết quả bài toán là max của các giá trị $(right_i - left_i - 1) * l_i$ với mọi i

Thuật toán

- Vấn đề còn lại là làm sao tính nhanh được các giá trị $left_i$ và $right_i$
- => Sử dụng stack
- Ví dụ với *left*, ta duyệt i tăng dần, luôn duy trì một stack chứa vị trí trước i và có độ cao nhỏ hơn cột i, trong đó các vị trí tăng dần và top của stack lưu vị trí lớn nhất. Như vậy $left_i$ chính là giá trị trên đỉnh stack khi ta duyệt đến i.
- Cập nhật stack: khi nào mà $l_i <= l_{stack_{top}}$ thì pop phần tử đỉnh stack. Sau đó push i vào stack
- Độ phức tạp $O(N)$, vì một phần tử chỉ vào và ra stack tối đa 1 lần.

Code

```
int a[N], l[N], r[N];
int main()
{
  while ( true ) {
    cin >> n;
    if ( n == 0 ) break;
    for (int i = 1; i <= n; i++)
      cin >> a[i];
    a[0] = a[n+1] = -1;

    stack<int> s;
    // calc left
    s.push(0);
    for (int i = 1; i <= n; i++) {
      while (a[s.top()] >= a[i]) s.pop();
      l[i] = s.top(), s.push(i);
    }

    // empty stack
    while (!s.empty()) s.pop();
    // calc right
    s.push(n+1);
    for (int i = n; i >= 1; i--) {
      while (a[s.top()] >= a[i]) s.pop();
      r[i] = s.top(), s.push(i);
    }

    long long res = 0;
    for (int i = 1; i <= n; i++)
      res = max(res, 1ll*a[i]*(r[i]-l[i]-1));
    cout << res << endl;
  }
```