

Tối ưu quy hoạch động 1 chiều

Tối ưu quy hoạch động 1 chiều

Người viết: Nguyễn Tuấn Tài - Trường Đại học Khoa học Tự nhiên, Đại học Quốc gia TP.HCM



Giới thiệu: đây là kiến thức xuất hiện trong đề thi TST 2022, và đã lấy đi rất nhiều nước mắt của thí sinh. Nếu bạn muốn thử học một thuật toán mới lạ mà nhiều người chưa biết, thì đây chính là bài viết dành cho bạn!

Khi làm những bài toán quy hoạch động, đôi khi ta sẽ nghĩ ra những thuật toán có độ phức tạp rất lớn, ví dụ:

$$\triangleright f[i][j] = \min_{0 \leq k < j} f[i-1][k] + w(k, j)$$

Công thức trên có độ phức tạp $O(n^3)$, có thể cải tiến xuống $O(n^2)$ hoặc $O(n^2 \log n)$ bằng quy hoạch động bao lồi/chia để trị (trong một số điều kiện nhất định).

$$\triangleright f[i] = \min_{0 \leq j < i} f[j] + w(j, i)$$

Công thức trên có độ phức tạp $O(n^2)$, có thể cải tiến xuống $O(n \log n)$ hoặc $O(n)$ (trong một số điều kiện nhất định).

Ở bài viết này, chúng ta sẽ tìm hiểu về cách tối ưu công thức thứ 2, hay còn gọi là *phương pháp tối ưu quy hoạch động 1 chiều*.

Giới thiệu bài toán

Gọi $w(j, i)$ là một hàm tính cost thỏa mãn bất đẳng thức tứ giác (quadrangle inequality):

$$w(a, c) + w(b, d) \leq w(a, d) + w(b, c) \text{ với mọi } a < b \leq c < d.$$

Ta sẽ tính toán công thức quy hoạch động sau với độ phức tạp nhanh hơn $O(n^2)$:

$$f[i] = \min_{0 \leq j < i} f[j] + w(j, i)$$

Một số ví dụ về hàm w thỏa mãn bất đẳng thức tứ giác (bạn đọc có thể tự chứng minh):

$$\triangleright w(j, i) = i - j$$

- $w(j, i) = b[j] \cdot a[i]$ (với $a[j] \leq a[i]$ và $b[j] \geq b[i] \forall j < i$)
- $w(j, i) = (a[i] - a[j])^2$ (với $a[j] \leq a[i] \forall j < i$)

Thuật toán



Nhưng... làm sao để tối ưu công thức quy hoạch động trên? Có cách nào để nhanh chóng tìm được vị trí mà $f[j] + w(j, i)$ đạt giá trị nhỏ nhất không?

Ta định nghĩa mảng h như sau

$$h[i] = \arg \min_{0 \leq j < i} f[j] + w(j, i)$$

với $\arg \min_t f(t)$ là chỉ số t để $f(t)$ đạt giá trị nhỏ nhất.

Nói cách khác, $h[i]$ là vị trí j nhỏ nhất thỏa mãn $f[j] + w(j, i)$ đạt giá trị cực tiểu.

Để thuận tiện cho việc biểu diễn thuật toán, ta sẽ quy ước

$$f[0] = 0, f[1] = f[2] = \dots = f[n] = \infty.$$

Ý tưởng "ngây thơ"

Trước khi đi vào thuật toán chính, chúng ta sẽ xem xét qua thuật toán "ngây thơ" sau:

- Ở thời điểm đầu tiên,
 $h[1] = h[2] = \dots = h[n] = 0$.
- Ở thời điểm thứ i :
 - Vì $h[i]$ đã được cập nhật hoàn toàn, ta tính được $f[i] = f[h[i]] + w(h[i], i)$.
 - Sau khi tính được $f[i]$, ta sẽ cập nhật lại $h[i+1], h[i+2], \dots, h[n]$.

Chúng ta có thể cài đặt thuật toán trên một cách đơn giản như sau:

```

1  const int N = 1e5 + 3;
2  int n, h[N];
3  long long f[N];
4
5  long long w(int j, int i) {
6      // một hàm cost bất kì thỏa mãn
7      // bất đẳng thức tứ giác
8  }
9
10 void solve() {
11     for (int i = 1; i <= n; ++i) {
```

```

12 // cập nhật f[i]
13 f[i] = f[h[i]] + w(h[i], i);
14
15 for (int j = i + 1; j <= n; ++j) {
16     // cập nhật lại h[i + 1..n]
17     if (
18         f[i] + w(i, j) < f[h[j]] + w(h[j], j)
19     ) {
20         h[j] = i;
21     }
22 }
23 }
24 }

```

Ý tưởng chính



Dễ thấy thuật toán "ngây thơ" trên có độ phức tạp $O(n^2)$. Làm sao để cải tiến thuật toán? Liệu mảng h có một tính chất đặc biệt nào có thể giúp ta dễ dàng cập nhật được không?

Nhận xét †. Ở mọi thời điểm, mảng h luôn là dãy đơn điệu tăng (tức $h[1] \leq h[2] \leq \dots \leq h[n]$). (chứng minh ở phần Phụ lục)



Việc mảng h luôn là dãy đơn điệu tăng sẽ có ý nghĩa gì?

Khi cập nhật đến $f[i]$, nếu tồn tại một vị trí j thỏa mãn $f[h[j]] + w(h[j], j) \leq f[i] + w(i, j)$, điều đó đồng nghĩa với việc $h[j]$ sẽ không thay đổi. Không chỉ thế, vì $h[i + 1] \leq h[i + 2] \leq \dots \leq h[j - 1] \leq h[j]$, nên cả đoạn $h[i + 1 \dots j]$ cũng sẽ không thay đổi.

Hệ quả. Nếu tồn tại vị trí j thỏa mãn $f[h[j]] + w(h[j], j) \leq f[i] + w(i, j)$, ta được $f[h[p]] + w(h[p], p) \leq f[i] + w(i, p)$ với mọi $i < p \leq j$.

Chính vì thế, để cập nhật mảng h , ta sẽ tìm vị trí z nhỏ nhất thỏa mãn $f[h[z]] + w(h[z], z) > f[i] + w(i, z)$, và cập nhật $h[z] = h[z + 1] = \dots = h[n] = i$. Từ đây, ta có ý tưởng thuật toán sau:

Thuật toán.

Ta sẽ biểu diễn mảng h thành m đoạn $(l[i], r[i], p[i])$ thỏa mãn:

$$\begin{cases} l[1] = 1 \\ r[m] = n \\ p[i] = h[l[i]] = h[l[i] + 1] = \dots = h[r[i]] \\ l[i + 1] = r[i] + 1, \forall 1 \leq i < m \\ p[i] < p[i + 1], \forall 1 \leq i < m \end{cases}$$

- Ở thời điểm đầu tiên, mảng h chỉ chứa đoạn $(1, n, 0)$.
- Ở thời điểm thứ i :
 - Để tính được $f[i]$, ta sẽ tìm đoạn $(l[k], r[k], p[k])$ thỏa mãn $l[k] \leq i \leq r[k]$.
 - Vì mảng h đã cập nhật đến $f[i - 1]$ nên $h[i]$ đã được cập nhật hoàn toàn (nhắc lại định nghĩa: $h[i] = \arg \min_{0 \leq j < i} f[j] + w(j, i)$).
 - Lúc này, ta sẽ cập nhật $f[i] = f[p[k]] + w(p[k], i)$.
 - Sau khi tính được $f[i]$, ta sẽ cập nhật lại mảng h . Theo nhận xét phía trên, ta sẽ tìm vị trí z nhỏ nhất thỏa mãn $f[i] + w(i, z) < f[h[z]] + w(h[z], z)$. Xét đoạn cuối cùng trong mảng h , giả sử đoạn đó là (l, r, p) .
 - Nếu $f[i] + w(i, l) < f[p] + w(p, l)$, ta sẽ xóa đoạn đó khỏi mảng h và tiếp tục xét đoạn cuối cùng tiếp theo.
 - Ngược lại, ta sẽ tìm vị trí z nhỏ nhất thỏa mãn $f[i] + w(i, z) < f[p] + w(p, z)$ bằng tìm kiếm nhị phân, sau đó cập nhật lại (l, r, p) thành $(l, z - 1, p)$, và thêm đoạn (z, n, i) vào cuối mảng h .

Cài đặt mẫu

Ta có thể cài đặt thuật toán trên bằng cách sử dụng [deque](#). Để thuận tiện cho việc cài đặt, ta sẽ không lưu lại các giá trị $h[i]$ đã qua sử dụng.

```

1 | struct item {
2 |     int l, r, p;
3 | };
4 |
5 | const int N = 1e5 + 3;
6 | int n;
7 | long long f[N];
8 |
9 | long long w(int j, int i) {
10 |     // một hàm cost bất kì thỏa mãn
11 |     // bất đẳng thức tứ giác
12 | }
13 |
14 | void solve() {
15 |     deque<item> dq;
16 |     dq.push_back({1, n, 0});
17 |

```

```

17
18 for (int i = 1; i <= n; ++i) {
19     f[i]=f[dq.front().p]+w(dq.front().p,i);
20     // deque chỉ lưu giá trị từ h[i + 1]
21     // tới h[n]
22     ++dq.front().l;
23
24     // nếu l > r, ta loại đoạn này khỏi deque
25     if (dq.front().l > dq.front().r) {
26         dq.pop_front();
27     }
28
29     while (!dq.empty()) {
30         auto [l, r, p] = dq.back();
31         if (f[i] + w(i, l) < f[p] + w(p, l)) {
32             dq.pop_back();
33             // p không còn là giá trị của
34             // h[l], h[l + 1], ..., h[r]
35             // lúc này, h[l]=h[l+1]=...=h[r]=i.
36         }
37         else break;
38     }
39
40     if (dq.empty()) {
41         dq.push_back({i + 1, n, i});
42         // h[i+1]=h[i+2]=...=h[n]=i
43     }
44     else {
45         // tìm nhị phân vị trí pos nhỏ nhất
46         // thỏa mãn h[pos] = i
47         auto& [l, r, p] = dq.back();
48         int low = l, high = r;
49         int pos = r + 1, mid;
50         while (low <= high) {
51             mid = (low + high) / 2;
52             if (f[i] + w(i, mid) < f[p] + w(p, mid)) {
53                 pos = mid, high = mid - 1;
54             }
55             else {
56                 low = mid + 1;
57             }
58         }
59
60         // cập nhật đoạn (l,r,p) thành (l,pos-1,p)
61         r = pos - 1;
62         if (pos <= n) {
63             dq.push_back({pos, n, i});
64             // h[pos]=h[pos+1]=...=h[n]=i
65         }

```

```

    ~
66  |   }
67  |   }
    |   }
    }

```

Vì số đoạn tối đa được thêm vào deque trong cả quá trình là n , kết hợp với tìm kiếm nhị phân ở cuối mỗi thời điểm, ta được độ phức tạp cuối cùng là $O(n \log n)$.

Bài toán 1

[Codeforces - 319C](#) [↗](#)

Tóm tắt

Cho n cây được đánh số hiệu từ 1 tới n , mỗi cây có độ cao lần lượt là a_1, a_2, \dots, a_n . Alob và Bice muốn cưa đổ tất cả n cây sao cho tốn ít chi phí nhất.

Alob và Bice có một cái cưa máy, mỗi lần sử dụng cưa có thể giảm độ cao của một cây bất kì xuống 1. Tuy nhiên, sau mỗi lần sử dụng, cưa máy cần được sạc lại. Chi phí để sạc phụ thuộc vào những cây đã được chặt hoàn toàn (những cây đã được giảm độ cao về 0): trong những cây đã được chặt hoàn toàn, giả sử cây có số hiệu lớn nhất là i , chi phí để sạc cưa máy là b_i . Nếu không có cây nào đã được chặt hoàn toàn, ta không thể sạc lại cưa máy.

Điều kiện bài toán:

$$\begin{cases} 1 \leq n \leq 10^5 \\ 1 = a_1 < a_2 < \dots < a_n \leq 10^9 \\ 10^9 \geq b_1 > b_2 > \dots > b_n = 0 \end{cases}$$

Ý tưởng

Vì $b_n = 0$, ta sẽ tìm chi phí nhỏ nhất để chặt hoàn toàn cây n (sau đó, ta có thể chặt bất kì cây nào mà không tốn chi phí).

Gọi $f[i]$ là chi phí nhỏ nhất để chặt hoàn toàn cây thứ i . Nếu cây gần nhất được chặt hoàn toàn trước đó là j , chi phí nhỏ nhất để chặt hoàn toàn cây thứ i sẽ là $f[j] + b_j \cdot a_i$. Vì vậy, ta có được công thức quy hoạch động sau:

$$f[i] = \min_{1 \leq j < i} f[j] + b_j \cdot a_i$$

Nếu đặt $w(j, i) = b_j \cdot a_i$, hàm w là một hàm thỏa mãn bất đẳng thức tứ giác.

Chứng minh.

Xét 4 điểm $x < y \leq z < t$, ta có:

$$\begin{aligned}
& w(x, z) + w(y, t) - w(x, t) - w(y, z) \\
= & b_x \cdot a_z + b_y \cdot a_t - b_x \cdot a_t - b_y \cdot a_z \\
= & (b_x - b_y)(a_z - a_t) \leq 0
\end{aligned}$$

Vì vậy,

$$w(x, z) + w(y, t) \leq w(x, t) + w(y, z)$$

Từ đây, ta có thể áp dụng thuật toán đã nêu trong bài.

Cài đặt mẫu

```

1  #include <bits/stdc++.h>
2  using namespace std;
3
4  struct item {
5      int l, r, p;
6  };
7
8  const int N = 1e5 + 3;
9  int n, a[N], b[N];
10 long long f[N];
11
12 long long w(int j, int i) {
13     return 1LL * b[j] * a[i];
14 }
15
16 void solve() {
17     deque<item> dq;
18     dq.push_back({2, n, 1});
19     for (int i = 2; i <= n; ++i) {
20         f[i] = f[dq.front().p] + w(dq.front().p, i);
21         ++dq.front().l;
22         if (dq.front().l > dq.front().r) {
23             dq.pop_front();
24         }
25
26         while (!dq.empty()) {
27             auto [l, r, p] = dq.back();
28             if (f[i] + w(i, l) < f[p] + w(p, l)) {
29                 dq.pop_back();
30             }
31             else break;
32         }
33
34         if (dq.empty()) {
35

```

```

36     dq.push_back({i + 1, n, i});
37 }
38 else {
39     auto& [l, r, p] = dq.back();
40     int low = l, high = r, pos = r + 1, mid;
41     while (low <= high) {
42         mid = (low + high) / 2;
43         if (f[i] + w(i, mid) < f[p] + w(p, mid)) {
44             pos = mid, high = mid - 1;
45         }
46         else {
47             low = mid + 1;
48         }
49     }
50
51     r = pos - 1;
52     if (pos <= n) {
53         dq.push_back({pos, n, i});
54     }
55 }
56 }
57 }
58
59 int main() {
60     cin >> n;
61     for (int i = 1; i <= n; ++i) {
62         cin >> a[i];
63     }
64     for (int i = 1; i <= n; ++i) {
65         cin >> b[i];
66     }
67     solve();
68     cout << f[n];
69 }

```

Bài toán 2

VNOJ - Lễ hội [🔗](#)

Tóm tắt

Ở ngôi làng nọ, có n ngôi nhà nằm trên một đường thẳng. Biết trưởng làng sống ở nhà thứ 1, nhà thứ i nằm cách nhà trưởng làng đúng a_i km về phía đông. Trưởng làng muốn chọn ra một số địa điểm trên đường thẳng để chuẩn bị tổ chức lễ hội, biết chi phí tổ chức lễ hội cho một địa điểm là k .

Sau khi chuẩn bị xong, tất cả người dân sẽ di chuyển đến địa điểm gần nhà mình nhất để tham gia lễ hội. Nếu một người dân ở cách địa điểm tổ chức x km, chi phí để di chuyển sẽ là x .

Hãy tìm cách chọn một số địa điểm sao cho tổng chi phí tổ chức lễ hội và di chuyển là nhỏ nhất.

Nói cách khác, nếu như ta chọn m địa điểm, địa điểm thứ i nằm cách nhà trưởng làng đúng s_i km về phía đông, tổng chi phí tổ chức lễ hội và di chuyển sẽ là $k \cdot m + \sum_{i=1}^n \min_{j=1}^m \|a_i - s_j\|$.

Điều kiện bài toán:

$$\begin{cases} 1 \leq n \leq 2 \cdot 10^5, 1 \leq k \leq 10^9 \\ 0 = a_1 < a_2 < \dots < a_n \leq 10^9 \end{cases}$$

Ý tưởng

Ta có nhận xét sau: tất cả người dân nằm trên một đoạn liên tiếp sẽ đến cùng một địa điểm, vì thế bài toán có thể viết lại thành: chia n người dân thành các đoạn liên tiếp sao cho tổng chi phí là nhỏ nhất, biết chi phí mỗi đoạn gồm chi phí tổ chức k và chi phí di chuyển của người dân trong đoạn.

Gọi $f[i]$ là chi phí nhỏ nhất để chia i người dân thành các đoạn sao cho tổng chi phí là nhỏ nhất. Ta có công thức quy hoạch động sau:

$$f[i] = k + \min_{0 \leq j < i} f[j] + w(j, i)$$

với $w(j, i)$ là chi phí di chuyển của người dân nằm trong đoạn $j + 1$ tới i .

Ta sẽ chứng minh hàm w thỏa mãn bất đẳng thức tứ giác.

Chứng minh.

Đặt số người dân trong đoạn $[l + 1, r]$ là $t = r - l$.

Đặt $p[i] = a_1 + a_2 + \dots + a_i$. Ta có 2 trường hợp sau:

- Nếu t chẵn, phương án đặt địa điểm tập trung tối ưu nhất là ở giữa người thứ $\frac{t}{2}$ và người thứ $\frac{t}{2} + 1$.

Chi phí di chuyển trong trường hợp này là $\sum_{i=r-\frac{t}{2}+1}^r a_i - \sum_{i=l+1}^{l+\frac{t}{2}} a_i$, hay $(p[r] - p[r - \frac{t}{2}]) - (p[l + \frac{t}{2}] - p[l])$.

- Ngược lại, phương án đặt địa điểm tập trung tối ưu nhất là ở nhà của người thứ $\frac{t+1}{2}$.

Chi phí di chuyển trong trường hợp này là $\sum_{i=r-\frac{t-1}{2}+1}^r a_i - \sum_{i=l+1}^{l+\frac{t-1}{2}} a_i$, hay $(p[r] - p[r - \frac{t-1}{2}]) - (p[l + \frac{t-1}{2}] - p[l])$.

Xét 4 điểm $x < y \leq z < t$. Để thuận tiện cho việc chứng minh, ta sẽ giả sử độ dài các đoạn đều là chẵn. Các trường hợp còn lại cũng có thể chứng minh tương tự.

Với $r - l$ chẵn:

$$w(l, r) = (p[r] - p[r - \frac{k}{2}]) - (p[l + \frac{k}{2}] - p[l]) = p[l] + p[r] - 2 \cdot p[\frac{l+r}{2}].$$

Đặt $b = \frac{x+z}{2}, c = \frac{y+t}{2}, d = \frac{x+t}{2}, e = \frac{y+z}{2}$. Ta có:

$$\begin{aligned} & w(x, z) + w(y, t) - w(x, t) - w(y, z) \\ = & 2 \cdot (-p[\frac{x+z}{2}] - p[\frac{y+t}{2}] + p[\frac{x+t}{2}] + p[\frac{y+z}{2}]) \\ = & 2 \cdot (-p[b] - p[c] + p[d] + p[e]) \\ = & 2 \cdot (p[d] - p[b]) - 2 \cdot (p[c] - p[e]) \\ = & 2 \cdot (a_{b+1} + a_{b+2} + \dots + a_d) \\ & - 2 \cdot (a_{e+1} + a_{e+2} + \dots + a_c) \\ = & 2 \cdot (a_{b+1} - a_{e+1}) + 2 \cdot (a_{b+2} - a_{e+2}) + \dots \\ & + 2 \cdot (a_d - a_c) \leq 0 \end{aligned}$$

Vì vậy,

$$w(x, z) + w(y, t) \leq w(x, t) + w(y, z)$$

Cài đặt mẫu

```

1  #include <bits/stdc++.h>
2  using namespace std;
3
4  struct item {
5      int l, r, p;
6  };
7
8  const int N = 2e5 + 3;
9  int n, k, a[N];
10 long long p[N], f[N];
11
12 long long w(int l, int r) {
13     int t = (r - l) / 2;
14     return (p[r] - p[r - t]) - (p[l + t] - p[l]);
15 }
16
17 void solve() {
18     deque<item> dq;
19     dq.push_back({1, n, 0});
20     for (int i = 1; i <= n; ++i) {
21         f[i] = k + f[dq.front().p] + w(dq.front().l, i);
22         ++dq.front().l;
23         if (dq.front().l > dq.front().r) {

```

```

24     dq.pop_front();
25 }
26
27 while (!dq.empty()) {
28     auto [l, r, p] = dq.back();
29     if (f[i] + w(i, l) < f[p] + w(p, l)) {
30         dq.pop_back();
31     }
32     else break;
33 }
34
35 if (dq.empty()) {
36     dq.push_back({i + 1, n, i});
37 }
38 else {
39     auto& [l, r, p] = dq.back();
40     int low = l, high = r, pos = r + 1, mid;
41     while (low <= high) {
42         mid = (low + high) / 2;
43         if (f[i] + w(i, mid) < f[p] + w(p, mid)) {
44             pos = mid, high = mid - 1;
45         }
46         else {
47             low = mid + 1;
48         }
49     }
50
51     r = pos - 1;
52     if (pos <= n) {
53         dq.push_back({pos, n, i});
54     }
55 }
56 }
57 }
58
59 int main() {
60     cin >> n >> k;
61     for (int i = 1; i <= n; ++i) {
62         cin >> a[i];
63         p[i] = p[i - 1] + a[i];
64     }
65     solve();
66     cout << f[n];
67 }

```

Phụ lục

Chứng minh nhận xét †.

Ta sẽ chứng minh bằng cách phản chứng: giả sử tồn tại vị trí i thỏa mãn $h[i] > h[i + 1]$. Để thuận tiện cho việc chứng minh, ta sẽ đặt $a = h[i]$, $b = h[i + 1]$ ($a > b$). Điều này tương đương với:

$$\begin{cases} f[a] + w(a, i) < f[b] + w(b, i) \\ f[a] + w(a, i + 1) > f[b] + w(b, i + 1) \end{cases}$$

Trừ hai bất đẳng thức theo vế, ta được:

$$w(a, i) - w(a, i + 1) < w(b, i) - w(b, i + 1)$$



$$\Leftrightarrow w(a, i) + w(b, i + 1) < w(a, i + 1) + w(b, i)$$

Tuy nhiên, theo tính chất của hàm w , xét bộ số $b < a < i < i + 1$, ta có:


$$w(b, i) + w(a, i + 1) \leq w(b, i + 1) + w(a, i)$$

Điều này là vô lý. Vì vậy, ta có điều phải chứng minh.

Ứng dụng

Kĩ thuật này thường được sử dụng cùng với kĩ thuật [Aliens' Trick](#) , kĩ thuật từng xuất hiện trong đề thi [IOI 2016](#) . Ngoài ra, hầu hết các bài toán có thể giải bằng QHĐ bao lồi/QHĐ chia để trị đều có thể giải bằng kĩ thuật này.

Bạn đọc có thể luyện tập thêm về kĩ thuật ở những bài sau:

- [VJudge - Batch Scheduling](#) 
- [Codeforces - 321E](#) 