|  |  |
| --- | --- |
| 28 | DP Optimization |
| Vương Hoàng Long  ĐỘI TUYỂN 2017 |
|  | |

Quy Hoạch Động là một trong những chủ đề quan trọng nhất của Tin học. Ở các kì thi lớn như APIO và IOI đều có ít nhất 1 đến 2 bài thuộc dạng này. Tuy nhiên, để đạt được điểm tối đa của các bài đó, đòi hỏi thí sinh phải có những kĩ thuật để tối ưu hàm quy hoạch động (hoặc là tăng tốc chuyển trạng thái, hoặc là giảm số trạng thái). Bài viết này sẽ giúp các bạn nắm được một số kĩ thuật tối ưu quy hoạch động tiêu biểu.

Tất cả code trong bài có thể được tìm thấy ở đây:

<https://github.com/UncleGrandpa925/TinTuyen2017>

## Quy hoạch động sử dụng bao lồi (Convex Hull Trick)

Phần này ở trên VNOI wiki có một bài dịch rất tốt nên tác giả xin phép không viết lại:

<http://vnoi.info/wiki/translate/wcipeg/Convex-Hull-Trick>

Để có thể sử dụng được bao lồi trong DP, hệ số góc của đường thẳng phải tăng dần. Nếu không thì ta có thể cài bao lồi động, nhưng kiến thức đó sẽ không được đề cập đến trong bài viết này.

Với việc add các đường thẳng theo hệ số góc tăng dần nên để tìm cực trì cho một giá trị nào đó thì chúng ta chỉ cần chặt tam phân. Tuy nhiên, nếu các truy vấn cũng có hệ số x tăng dần , thì ta có thể lợi dụng 2 con trỏ để giảm thời gian truy vấn xuống còn O(n) thay vì O(nlogn) như chặt tam phân. Với trường hợp sử dụng 2 con trỏ, khi chúng ta pop đường thẳng ra khỏi bao lồi thì con trỏ trỏ đến đường thẳng đang đạt cực trị cũng phải được lùi tương ứng.

Cũng cần lưu ý thêm, là trong một số ít bài, tồn tại một số đường thẳng có hệ số góc bằng nhau, vì vậy chúng ta cần xử lí đặc biệt. Ví dụ nếu chúng ta muốn tìm max, thì khi add các đường thẳng có A bằng nhau, chúng ta chỉ cần add đường thẳng có B lớn nhất. Nếu tìm min thì làm ngược lại.

Thêm một chú ý nữa, trong code truy vấn khi tìm bao lồi, có 1 đoạn như thế này

while (curit != A.size() - 1 && A[curit] \* 1LL \* x + B[curit] <= A[curit + 1] \* 1LL \* x + B[curit + 1]) curit++;

Ở phần này, nếu bỏ dấu = thì code vẫn sẽ cho kết quả đúng. Tuy nhiên, với một số bài yêu cầu truy vết khá đặc biệt như bài APIO2014\_sequence( sẽ đọc ở dưới) thì việc thêm dấu = là bắt buộc. Lấy ví dụ với ngay bài đó, vì đề bài yêu cầu tách đúng K lần, nên nếu không thêm dấu bằng thì sẽ dẫn đến trường hợp chỉ cần tách < K lần và in ra truy vết sai.

Một số bài tập áp dụng:

// fix tăng (giamr dần)

// thêm folder code vào đây:

// bao lồi chưa test phần không tenary search

1. ACQUIRE – USACO

Link đề bài: <http://www.spoj.com/problems/ACQUIRE/en/>

Cách giải: Trước tiên, không ảnh hưởng đến kết quả bài toán, ta sort lại các hình chữ nhật theo X, sau đó theo Y. Xét 2 hình chữ nhật, nếu Xj>=Xi && Yj >= Yi, bỏ hình chữ nhật I đi sẽ không ảnh hưởng đến kết quả của bài toán. Sau khi bỏ hết các hình chữ nhật thừa đi, ta sẽ được một dãy các hình chữ nhật thoả mãn nếu i<j thì Xi < Xj nhưng Yi > Yj. Gọi dp[i] là cost nhỏ nhất để mua được tất cả các miếng đất từ 1 đến i. Ta có công thức quy hoạch động đơn giản dp[i] = min(dp[j] + X[i] \* Y[j+1]) với các j chạy từ 1 đến i-1 (Vì các hình chữ nhật từ j+1 đến I thì hình I sẽ là hình có chỉ số X lớn nhất, và j+1 sẽ là hình có chỉ số Y lớn nhất.

Nhận thấy công thức quy hoạch động có dáng y = ax+b, ta sẽ nghĩ đến việc tối ưu bằng bao lồi. Đường thằng trong trường hợp này sẽ có a = Y[j+1] và b = dp[j]. Chý ý rằng ở vị trí n thì không add đường thẳng nữa để tránh tràn mảng.

1. Product Sum – Codeforces 631E

Link đề bài: <http://codeforces.com/problemset/problem/631/E>

Cách giải: Bài này là một bài toán rất hay, áp dụng tư tưởng đổi hệ số X để có thể tránh phải sử dụng bao lồi động. Trước tiên thì ta cần có cách giải N^2 cho bài này. Gọi mảng simpleSum[i] là tổng các số từ a[1] đến a[i], complexSum[i] là tổng của a[1] \* 1 + a[2] \* 2 + … + a[n] \* n. Tạm gọi 1 giá trị a[i] \* i là complex[i]

Nếu chưa có phép chuyển nào, giá trị hiện tại của dãy là complexSum[n].

Ta sẽ chia phép chuyển trong bài thành 2 trường hợp, chuyển 1 số từ trái qua phải và ngược lại. Với mỗi phép chuyển ta sẽ tính giá trị dãy mới. Với trường hợp chuyển số từ trái qua phải, gọi vị trí số bị chuyển là J, vị trị mà nó đến là I. Ta thấy giá trị complex của các số từ 1 đến J – 1 không đổi, và các số từ I+1 đến N cũng không đổi. Còn các số từ J+1 đến I thì mỗi số sẽ bị giảm 1 lần trong độ đóng góp cho giá trị cuối cùng của dãy (vì tất cả các số này đều bị lùi sang trái 1 đơn vị). Trong đó, số ở vị trí thứ J sẽ nhảy từ vị trí J lên vị trí I, tức là tăng thêm (I-J) lần xuất hiện. Vậy giá trị dãy mới sẽ là valLeft = complexSum[n] + simpleSum[j] – a[j]\*j + a[j] \* i – simpleSum[i]. Tương tự với trường hợp chuyển từ phải qua trái, vẫn gọi J là vị trí ban đầu và I là vị trí đến. Ta có giá trị dãy mới là valRight = complexSum[n] + simpleSum[j - 1] - a[j] \* j + a[j] \* i – simpleSum[i-1]. Vậy với 2 vòng for ta đã có thể dễ dàng giải được bài toán với độ phức tạp O(n^2).

Để AC được bài này, ta sẽ phải tối ưu sử dụng bao lồi. Mình sẽ xét trường hợp từ trái qua phải, trường hợp từ phải qua trái thì tương tự. Với một số bạn (trong đó có mình) ở công thức N^2 để tiện for thì mình sẽ for I trước, và for J sau. Tức là for vị trí chuyển đến trước và for số nào chuyển đến số đó. Tuy nhiên, cách for này sẽ khiến I giữ nguyên, và a[j] thay đổi. Lúc này, các đường thẳng sẽ đều có hệ số là a[j], và sẽ không thể làm bao lồi tĩnh bình thường được ( do hệ số góc không tăng(hoặc giảm) dần). Vậy để ý rằng nếu ta giữ J và for I thì I sẽ là tăng dần và có thể sử dụng bao lồi tĩnh. Vậy ta viết lại công thức tính giá trị dãy mới khi đổi từ trái qua phải thành valLeft = complexSum[n] + simpleSum[j] – a[j]\*j + i\*a[j] – simpleSum[i]. Lúc này thì hệ số góc của các đường thẳng đã đổi thành I, và vì truy vấn không tăng (hoặc giảm) dần nên ta sẽ cần dùng chặt tam phân để tìm cực trị. Cách này sẽ dễ code và nhanh hơn so với cài bao lồi động. Chi tiết các bạn có thể xem trong code.

1. NKLEAVES – SPOJ

Link đề bài: <http://vn.spoj.com/problems/NKLEAVES/>

Cách giải: Đây là một bài DP tối ưu bằng bao lồi khá cơ bản. Ta dễ dàng nhận ra

1. Split The Sequence – APIO 2014

Link đề bài: <https://oj.uz/problem/view/APIO14_sequence>

Cách giải: Để có thể giải được bất kì subtask nào của bài, ta cũng cần có 1 nhận xét rất quan trọng: thứ tự tách dãy không quan trọng đến kết quả cuối cùng. Để làm dễ hơn, ta sẽ coi là ta tách dãy từ phải qua trái. Gọi dp[i][j] là đến vị trí thứ I và đã tách thành j đoạn thì giá trị max là bao nhiêu. Sau đó ta sẽ có công thức quy hoạch động N^2 như sau: dp[i][j] = max(dp[k][j-1] + sum(1..k) \* sum(k+1..i)) với k chạy từ 1 đến i-1. Tách công thức ra thành dp[i][j] = max(dp[k][j-1] + sum(1..k) \* sum(1..i) - sum(1..k)^2) hay viết cách khác là dp[i][j] = max(sum(1..k) \* sum(1..i) - sum(1..k)^2 + dp[k][j-1]) thì ta nhận thấy ngay phương trình có dạng y = ax + b với a là sum(1…k) còn b là sum(1..k)^2 + dp[k][j-1]. Nhận xét là a tăng dần (nhưng không tăng nghiêm ngặt, nên phải lưu ý xử lí như đã nói ở đầu bài viết), sum(1…i) cũng tăng dần nên có thể dùng 2 con trỏ thay cho chặt tam phân. Chú ý rằng có thể có nhiều cách tách để nhận được dãy có giá trị lớn nhất, nhưng vì yêu cầu tách đúng k lần nên mỗi khi truy vấn ta cần tìm k lớn nhất có thể để tránh trường hợp truy vết thiếu.

Bài này mình cũng chỉ được 71 điểm trên oj.uz, tuy nhiên khi tải test để chấm thì đã AC.

Code: // insert code here

<http://codeforces.com/problemset/problem/631/E>

<http://codeforces.com/problemset/problem/319/C>

<http://codeforces.com/problemset/problem/455/E>

<https://www.codechef.com/problems/JUMP>

<http://codeforces.com/contest/536/problem/C>

<https://www.codechef.com/problems/CYCLRACE>

<http://codeforces.com/problemset/problem/377/E>

<http://codeforces.com/contest/91/problem/E>

<https://oj.uz/problem/view/APIO14_sequence>

<http://www.spoj.com/problems/APIO10A/>

<http://codeforces.com/problemset/problem/660/F>

## II) Divide and Conquer

Divide and conquer(Gọi ngắn là D&C) là một trong những tối ưu rất mạnh, dạng thức tối ưu của nó như sau: dp[i] = max(dp[j] + cost(j…i)) = O(n^2) -> O(nlogn) với 1 điều kiện duy nhất. Gọi F[i] là vị trí J mà I truy về, ta có F[x] <= F[y] nếu x<=y. Trong một số bài, việc chứng minh được điều kiện ở trên là khá dễ dàng, tuy nhiên trong một số bài khác thì lại không được như vậy. Có một số bài toán có thể giải được bằng cả 2 cách là QHĐ Bao lồi hoặc dùng D&C, và mỗi cách đều có ưu điểm riêng. Với D&C, ưu điểm lớn nhất của nó là độ dễ cài đặt. Tuy nhiên nó lại chạy chậm hơn QHĐ Bao lồi trong trường hợp QHĐ Bao lồi có độ phức tạp là O(n) và việc chứng minh điều kiện đúng của D&C không phải lúc nào cũng dễ dàng.

Bài codeforces, codechef