TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI

VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

———————— * ————————

Bài tập lớn môn học:

Phân tích và thiết kế thuật toán

Một số kĩ thuật tối ưu hóa quy hoạch động

Sinh viên thực hiện:
Nguyễn Quang Quý - 20153108

Giáo viên hướng dẫn:
TS. Đỗ Phan Thuận

Hà Nội, Ngày 20 tháng 12 năm 2018

Mục lục

Mở đầu

Trong một số bài toán quy hoạch động, việc tìm được công thức quy hoạch động tối ưu vẫn chưa đảm bảo rằng kết quả của bài được thông qua do phần lớn sẽ bị vượt quá về mặt thời gian. Do đó cần thay đổi cách cài đặt để có thể được kết quả tối ưu hơn về mặt thời gian. Những bài toán này thường được gặp trong kì thi ACM, ICPC, Codefore (với phân ban div 1), ..., đòi hỏi người tham gia phải biết được những kĩ thuật tối ưu nếu muốn được trọn vẹn điểm của bài toán. Trong tài liệu này sẽ trình bày hai kĩ thuật tối ưu trong quy hoạch động hiệu quả đó là sử dụng bao lồi và chia để trị, đồng thời là hai ví dụ cho từng bài toán.

1 Bao lồi đường thẳng

Kĩ thuật bao lồi được sử dụng để có thể cải tiến một lớp các bài toán quy hoạch động với độ phức tạp thời gian từ $O(n^2)$ về $O(nlogn)$ thậm chí còn có thể là $O(n)$. Lớp bài toán được áp dụng kĩ thuật này có dạng như sau:

$$dp[i] = min_{j<i}(dp[j] + b[j] * a[i])$$

Điều kiện để áp dụng đó là: $b[j] \geq b[j+1]$.

1.1 Cơ sở kĩ thuật bao lồi đường thẳng

Bài toán cơ sở

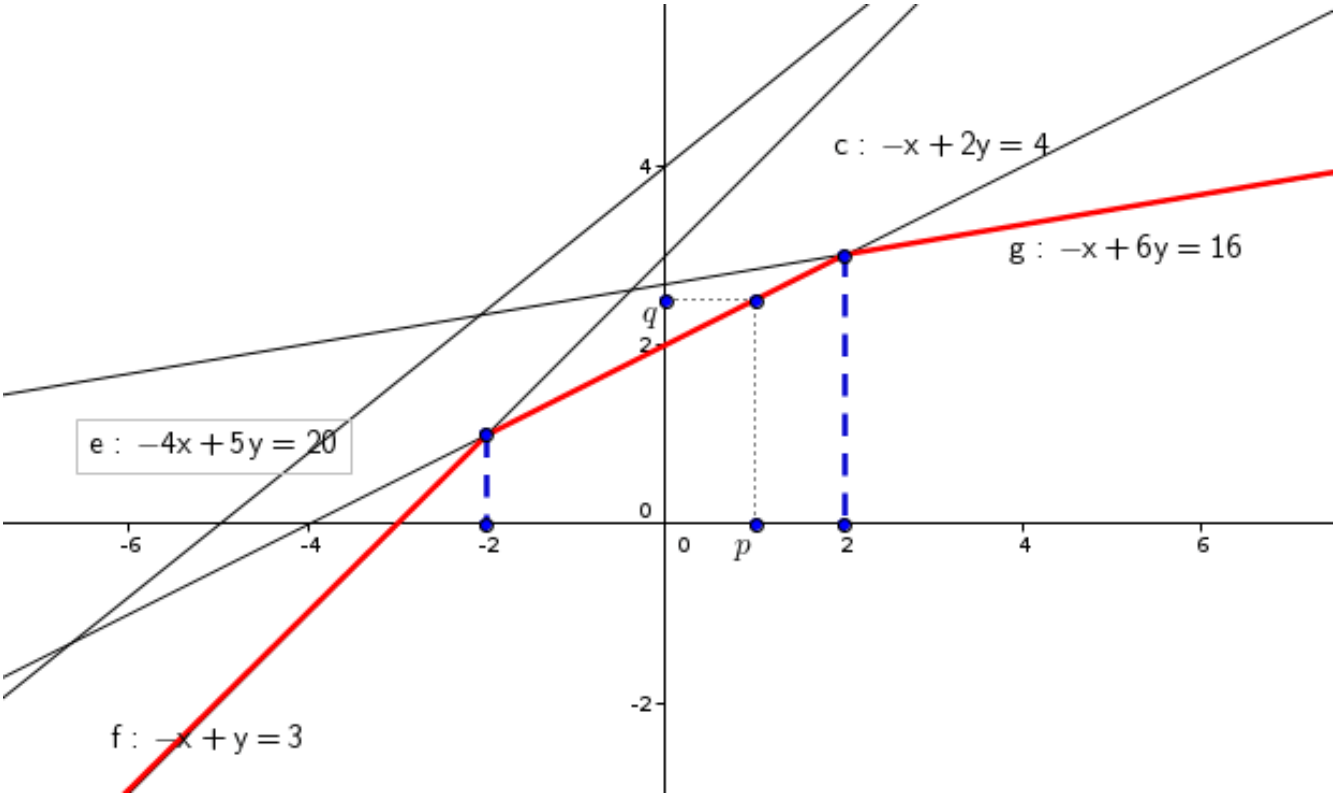
Cơ sở của kĩ thuật bao lồi được xem xét thông qua bài toán sau:

Bài toán: Cho một tập n đường thẳng $D = \{d_1, d_2, \ldots, d_n\}$ trong đó $(d_i) \quad y_i = a_i x + b_i, a_i \geq 0$ và k điểm trên trục Ox: $p_1, p_2, \ldots, p_k$. Tìm $q_1, q_2, \ldots, q_k$ trong đó mỗi giá trị q_ℓ được định nghĩa như sau:

$$q_\ell = \min_{1 \leq i \leq n}(a_i \cdot p_\ell + b_i)$$

Phân tích Với mỗi điểm p_ℓ, bằng cách duyệt qua tất cả các đường trong D, ta có thể tính được q_ℓ trong thời gian $O(n)$. Tuy nhiên, nếu các đường thẳng đã được sắp xếp giảm dần theo độ dốc thì ta có thể tính q_ℓ trong thời gian $O(logn)$ bằng kĩ thuật bao lồi đường thẳng. Ta xem xét một ví dụ sau để thu được một số nhận xét quan trọng:
Giả sử D gồm:

- $(d_1) \quad -x + y = 3$
- $(d_2) \quad -x + 2y = 4$
- $(d_3) \quad -x + 6y = 16$
- $(d_4) \quad -4x + 5y = 20$



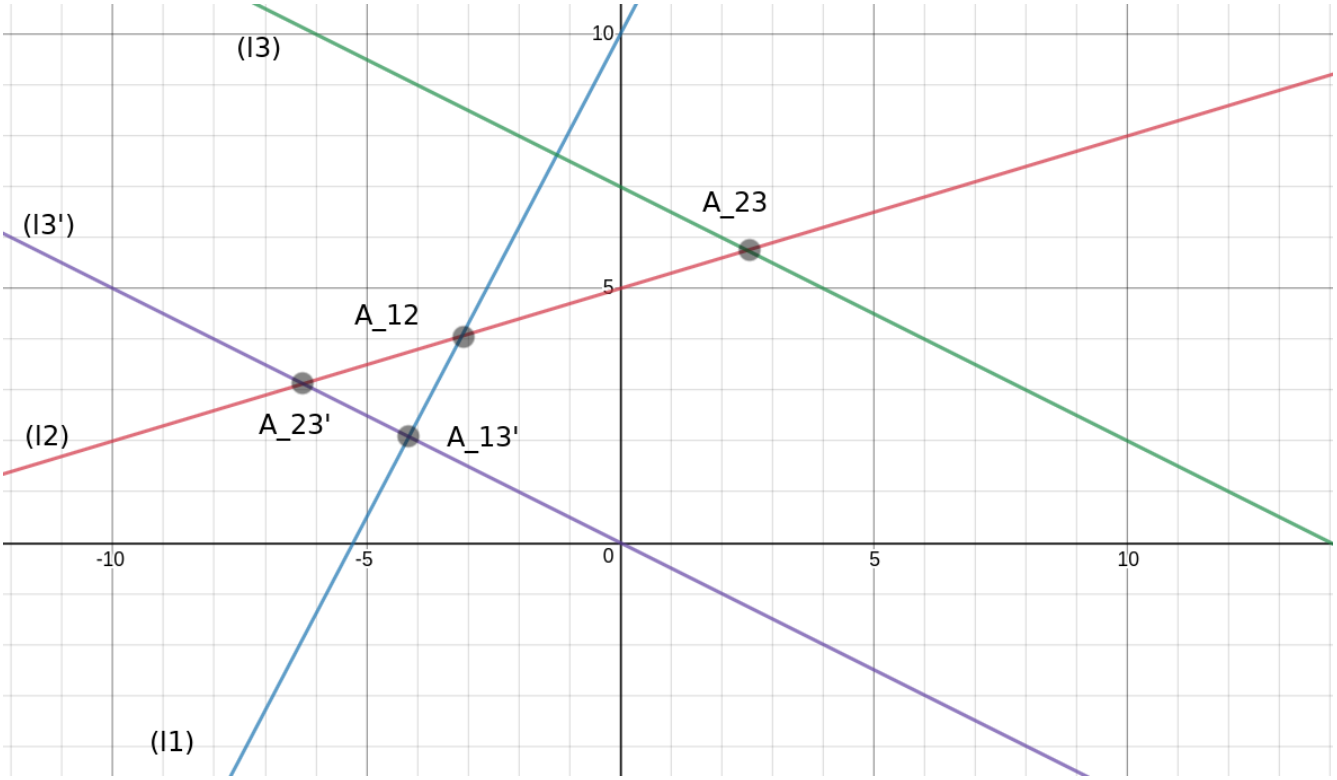
Ta thấy với mỗi điểm $(p, 0) \in Ox$ giá trị nhỏ nhất tương ứng $(0, q)$ sẽ thỏa mãn (p, q) thuộc phần gạch đỏ. Phần gạch đỏ đó gọi là bao lồi. Dựa vào hình vẽ ta thấy đường thẳng d_4 trong D là đường dư thừa (tung độ của điểm có hoành độ p trên đường thẳng này luôn lớn hơn tung độ của điểm có cùng hoành độ trên một đường thẳng khác của D). Dựa vào ví dụ này, ta có một số nhận xét như sau:

- Tồn tại một số đường thẳng dư thừa trong D
- Dọc theo đường bao lồi từ hoành độ $-\infty$ tới $+\infty$, các đường thẳng thuộc bao lồi có hệ số góc giảm dần
- Đường bao lồi có thể được biểu diễn dưới dạng các khoảng giá trị (interval) và đường thẳng tương ứng với interval đó. Trong ví dụ này, ta có 3 interval đó là $I_1 = [-\infty, -2], I_2 = [-2, 2], I_3 = [2, +\infty]$ và các đường thẳng tương ứng lần lượt là d_1, d_2 và d_3.

Từ những nhận xét này, ta xây dựng thuật toán tìm bao lồi của D (Giả sử trong D không tồn tại hai đường thẳng song song với nhau, với trường hợp song song ta sẽ xem xét sau) như sau:

```
FindHull(d_1, d_2, ..., d_n):
  sort {d_1, d_2, ..., d_n} by decreasing order of slope
  Stack S ← ∅
  S.push(d_1)
  I_1 ← [−∞, +∞]
  m ← 1
  for i ← 2 to n
    d ← S.peek()          [[examine the top element]]
    x_p ← FindIntersectionX(d, d_i)
    while x_p ≤ left(I_m)          [[found a redundant line]]
      S.pop()
      m ← m − 1
      d ← S.peek()
      x_p ← FindIntersectionX(d_i, d)
    S.push(d_i)
    I_m ← [left(I_m), x_p]
    I_{m+1} ← [x_p, +∞]
    associate I_{m+1} with d_i
    m ← m + 1
```

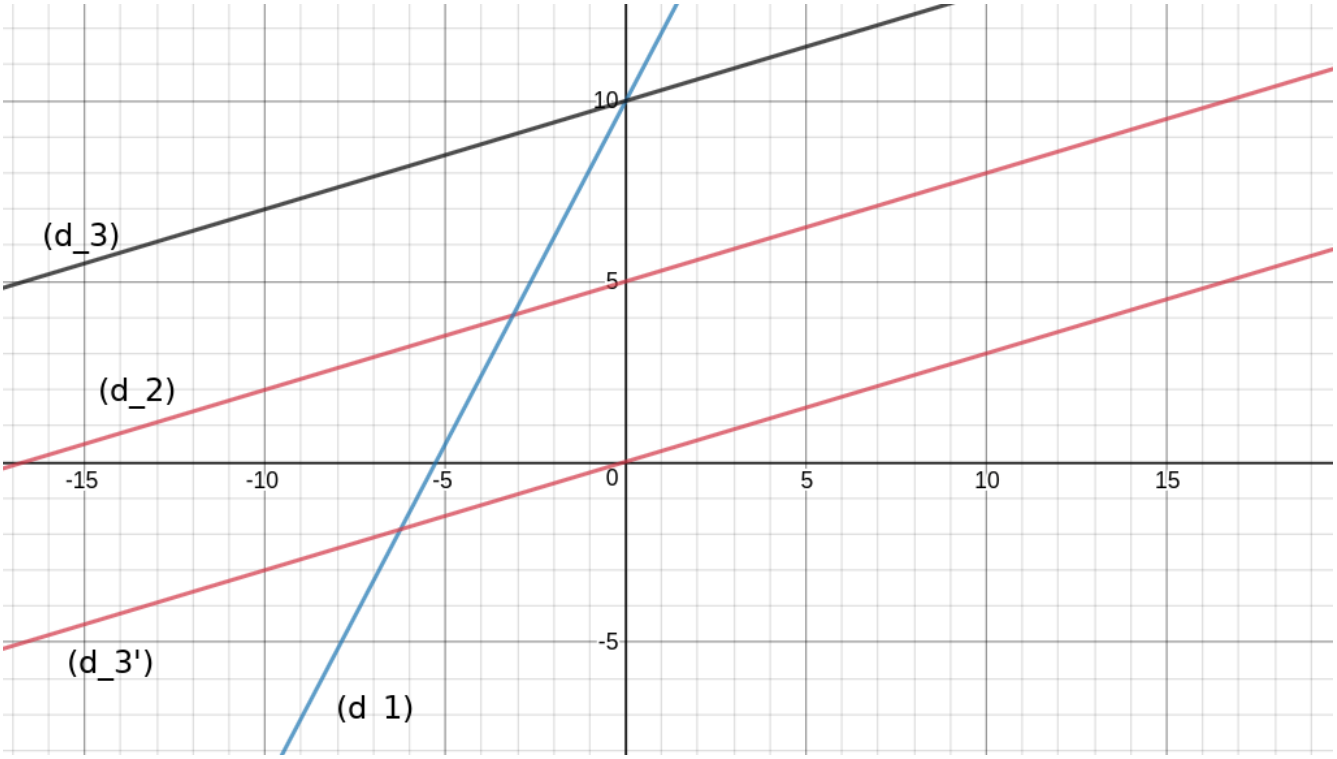
Ở đây, ta sử dụng stack S để lưu các đường thẳng có ích, cụ thể ta cho từng đường thẳng vào S. Với mỗi đường thẳng cho vào ta xem xét khả năng thay thế của nó với các đường sẵn có trong S, bất cứ đường nào được thay thế sẽ bị loại khỏi S và không thể trở lại S lần thứ 2. Để xem xét khả năng thay thế, ta thực hiện như sau: Giả sử ℓ_1, ℓ_2, ℓ_3 là các đường nằm ngay sau đỉnh của S, nằm tại đỉnh của S và đường được đưa vào. Đoạn ℓ_2 không quan trọng (không có giá trị cực tiểu ở điểm nào) khi và chỉ khi giao điểm của ℓ_1 và ℓ_3 nằm bên trái giao điểm của ℓ_1 và ℓ_2 (Đoạn mà ℓ_3 nhận giá trị cực tiểu đã nằm đè lên đoạn của ℓ_2).
Một cách trực quan, ta quan sát hình dưới đây. Nếu đường ℓ_3 là đường được thêm vào thì giao điểm giữa ℓ_3 và ℓ_2 là A_{23} không làm ℓ_2 và ℓ_1 trở thành vô ích. Ngược lại, nếu đường $\ell_{3'}$ là đường được thêm vào thì ta thấy ngay trên hình ℓ_2 chính là đường dư thừa và do đó nó sẽ bị loại bỏ.



Thời gian thực hiện thuật toán Ta có thể thấy thời gian thực hiện vòng lặp **for** của thuật toán tìm bao lồi có thời gian chạy $O(n)$ vì mỗi đường thẳng sẽ được xem xét tối đa hai lần: khi đưa vào stack và khi lấy ra khỏi stack. Nếu một đường thẳng bị lấy ra khỏi stack, nó sẽ không bao giờ được kiểm tra lại nữa (do nó không thể trở thành một phần của đường bao lồi nữa). Do đó, thời gian của thuật toán tìm bao lồi chủ yếu dành để sắp xếp các đường thẳng theo độ dốc và mất $O(nlogn)$.
Giả sử ta đã tìm được bao lồi của D, khi đó với mỗi điểm p_ℓ thực hiện tìm kiếm nhị phân để tìm $I \in I_1, I_2, \ldots, I_m$ sao cho $p_\ell \in I$, khi đó ta sẽ tìm được q_ℓ bằng cách thay $_\ell$ vào đường thẳng tương ứng với I. Do vậy, để tìm q_ℓ ta sẽ chỉ mất thời gian là $O(logn)$.

Xử lý trường hợp song song Trong trường hợp tập D tồn tại những cặp đường thẳng song song, khi đó ta cần thay đổi một chút trong thao tác loại bỏ đường dư thừa. Giả sử, đường thẳng cận đỉnh stack là d_1, đỉnh stack là d_2 và đường thẳng được thêm vào là d_3 song song với d_2. Khi đó sẽ có hai khả năng đối với d_3: (hình vẽ)

- Nếu d_3 nằm trên hoặc trùng với d_2 (giao điểm với Oy lớn hơn) thì d_3 sẽ không gây ảnh hưởng tới tập interval hiện tại, do đó d_3 chính là đường dư thừa và sẽ không được thêm vào stack.
- Nếu d_3 nằm dưới d_2 thì khi đó d_2 sẽ trở nên dư thừa, do đó cần được loại bỏ khỏi stack. Sau đó ta tiếp tục tiến hành thao tác loại bỏ đường dư thừa có thể có trong stack.



1.2 Bài toán 1: Phân nhóm

Giới thiệu

- File dữ liệu vào: group.inp
- File kết quả: group.out
- Hạn chế thời gian: 0.661 giây
- Hạn chế bộ nhớ: 1536Mb

Đề bài: Cho $1 \leq n \leq 300000$ cặp số (x, y) $(1 \leq x, y \leq 1000000)$. Ta có thể nhóm một vài cặp số lại thành một nhóm. Việc nhóm này sẽ mất một chi phí. Giả sử một nhóm gồm các cặp số $a_1, a_2, \ldots, a_m$ thì chi phí cho nhóm này sẽ là $max(x_{a_1}, x_{a_2}, \ldots, x_{a_m}) * max(y_{a_1}, y_{a_2}, \ldots, y_{a_m})$.
Yêu cầu: Tìm cách phân nhóm sao cho tổng chi phí là nhỏ nhất.

Dữ liệu vào

- Dòng đầu tiên là số nguyên n
- n dòng tiếp theo mỗi dòng tương ứng với một cặp (x, y).

Kết quả In ra chi phí tối ưu của bài toán.

Ví dụ

group.inp	group.out
4 100 1 15 15 20 5 1 100	500

Phân tích

Nhận xét 1: Với hai cặp số (x_1, y_1) và (x_2, y_2) mà $x_1 > x_2$ và $y_1 > y_2$ thì ta nói (x_2, y_2) là không "tiềm năng" vì ta có thể thêm nó vào nhóm với cặp (x_1, y_1) mà không làm tăng thêm chi phí.
Như vậy có thể sắp xếp lại các cặp số tăng dần theo x sau đó sử dụng Stack để loại bỏ đi những cặp số không "tiềm năng", cuối cùng sẽ chỉ còn lại một dãy các cặp có x tăng dần và y giảm dần.

Nhận xét 2: Sau khi sắp xếp, giả sử ta chọn (x_1, y_1) và (x_2, y_2) là một nhóm, khi đó ta có thể thêm các cặp (x, y) mà $x_1 \leq x \leq x_2$ và $y_1 \geq y \geq y_2$ vào nhóm đó mà không tốn thêm chi phí nào. Vậy ta có thể thấy rằng cách phân hoạch tối ưu là một cách phân dãy thành các đoạn liên tiếp và chi phí của một đoạn là bằng tích của thành phần y của cặp đầu tiên và thành phần x của cặp cuối cùng.

Công thức qui hoạch động Gọi $C[i]$ là chi phí cực tiểu để phân nhóm được i cặp số đầu tiên. Ta có công thức đệ qui của $C[i]$ như sau:

$$C[i] = min_{0 \leq j < i}(C[j] + x_i * y_{j+1})$$

Đặt $(d_j) \quad y = y_{j+1} * x_i + C[j]$ khi đó $C[i]$ chính là tung độ q nhỏ nhất thuộc một trong các đường thẳng $d_1, d_2, \ldots, d_{i-1}$ tương ứng với hoành độ $p = x_i$. Do đó chúng ta có thể sử dụng kĩ thuật bao lồi để giải quyết bài toán này.

Để ý một chút nữa, ta thấy rằng các giá trị của x là tăng dnầ, nên khi tìm kiếm interval chứa x_i, chúng ta không cần phải xét các interval nằm bên trái x_{i-1}. Do đó thay vì thực hiện tìm kiếm nhị phân, ta tìm kiếm tuyến tính với điểm tìm kiếm bắt đầu là interval chứa x_{i-1}.

Phân tích độ phức tạp

Thuật toán được thực hiện gồm 2 giai đoạn:

Tiền xử lý: Ở bước này, ta tiến hày xử lý đầu vào theo như nhận xét 1. Ta thấy rằng, sau khi sắp xếp, mỗi cặp số sẽ được xem xét duy nhất một lần và mỗi lần xem xét nó sẽ có hai khả năng: giữ lại hoặc bỏ đi. Vậy nên, thao tác loại bỏ cặp không "tiềm năng" sẽ lã $O(n)$. Do đó độ phức tạp tính toán trong pha này sẽ là chủ yếu là do theo tác sắp xếp đó là $O(nlogn)$.

Thực hiện quy hoạch động: Thuật toán dùng để cài đặt dựa trên mã giả như sau (Với B là mảng lưu các hệ số góc, A là mảng lưu các điểm truy vấn (trong trường hợp bài toán này đó là x_i):

```
FastFastDynamic(A[1,2,...,n], B[1,2,...,n]):
    d_1 ← B[1]x + C[1]
    Stack S ← ∅
    S.push(d_1)
    I_1 ← [−∞, +∞]
    associate I_1 with A[1]
    m ← 1
    for i ← 2 to n
        I_j ← the interval associated with A[i − 1]
        I ← ∅
        ℓ ← j − 1
        while I = ∅
            ℓ ← ℓ + 1
            if A[i] ∈ I[ℓ]
                I ← I[ℓ]
        d ← the line associated with I
        C[i] ← a · A[i] + b              [[assuming (d)   y = ax + b]]
        d_i ← B[i]x + C[i]
        d ← S.peek()                [[examine the top element]]
        x_p ← FindIntersectionX(d, d_i)
        while x_p ≤ left(I_m)               [[found a redundant line]]
            S.pop()
            m ← m − 1
            d ← S.peek()
            x_p ← FindIntersectionX(d_i, d)
        S.push(d_i)
        I_m ← [left(I_m), x_p]
        I_{m+1} ← [x_p, +∞]
        if ℓ < m        [[the line associated with I_ℓ is not redundant]]
            associate I_ℓ with A[i]
        else
            if A[i] ∈ I_m
                associate I_m with A[i]
            else
                associate I_{m+1} with A[i]
        associate I_{m+1} with d_i
        m ← m + 1
    return C[n]
```

Dựa vào mã giả trên, mỗi vòng lặp thực hiện hai khâu:

- Tìm kiếm interval: Trong bài toán này, các điểm truy vấn x_i được sắp xếp theo thứ tự tăng dần, do đó ta để tìm kiếm interval cho x_i ta chỉ cần tìm kiếm kể từ interval chứa x_{i-1}, và có thể thực hiện bằng tìm kiếm tuyến tính thay vì sử dụng binary search.
- Loại bỏ đường thẳng dư thừa: Thực hiện như trong bài toán cơ sở.

Tổng thời gian cho thao tác loại bỏ đường thẳng dư thừa vẫn là $O(n)$ như đã pân tích trong bài toán cơ sở.

Tổng thời gian cho thao tác tìm kiếm interval là $O(n)$. Thật vậy, dựa trên mã giả trong thủ tục tìm kiếm I_ℓ sao cho $A[i] \in I_\ell$trường hợp tồi nhất chúng ta phải duyệt qua toàn bộ tập các interval. Tuy nhiên, nếu đã duyệt qua K interval, thì các vòng lặp sau chúng ta không phải xét tới interval này và các đường thẳng tương ứng với chúng nữa (do chúng ta chỉ cần phải xét các interval kể từ interval chứa điểm truy vấn ngay trước điểm truy vấn hiện tại).

Do đó tổng thời gian thực hiện của phần quy hoạch động là $O(n)$.

Cuối cùng ta được: tổng thời gian thực hiện thuật toán $O(nlogn)$ (thời gian sắp xếp và loại bỏ cặp không "tiềm năng").

Một số chú ý khi cài đặt

- Trong công thứ qui hoạch động, ta để ý thấy rằng, ngoài các đường thẳng $(d_j) \quad y = y_{j+1} * x + C[j]$ đã được sắp xếp theo thứ tự giảm dần về độ dốc thì có một đường thẳng đặc biệt song song với trục $0x$ đó là $(d) \quad y = y_1 * x_i$ (ứng với vòng lặp thứ i). Do đó trong thủ tục tìm I_ℓ chứa điểm truy vấn x_i ta cần xét thêm sự ảnh hưởng của d đối với tập interval đã được tạo từ (d_j). Tuy nhiên, mục tiêu của chúng ta khi đi tìm I_ℓ đó là tìm được đường thẳng ứng với I_ℓ, từ đó tính toán được $C[i]$, do vậy thay ta có thể xem xét ảnh hưởng của d bằng cách như sau: ta sẽ tính toán $C[i]$ dựa trên đường thẳng ứng với I_ℓ (d) sau đó so sánh với giá trị của $C[i]$ thu được khi tính toán từ d. Khi đó kết quả của $C[i]$ sẽ là giá trị nhỏ nhất trong hai giá trị tìm được.
- Tại vòng lặp thứ i, sau khi tìm được interval ứng với điểm truy vấn x_i ta có thể lưu lại chỉ số của interval đó để sử dụng cho bước lặp thứ $i + 1$, do vậy không cần thiết phải có một mảng để lưu trữ interval của mỗi điểm truy vấn.

Kết quả thử nghiệm

Mô tả bộ thử nghiệm Ta nhắc lại các ràng buộc của bài toán đó là:

- Ràng buộc cho số cặp: $1 \leq n \leq 300000$
- Ràng buộc cho cặp (x, y): $1 \leq x, y \leq 1000000$

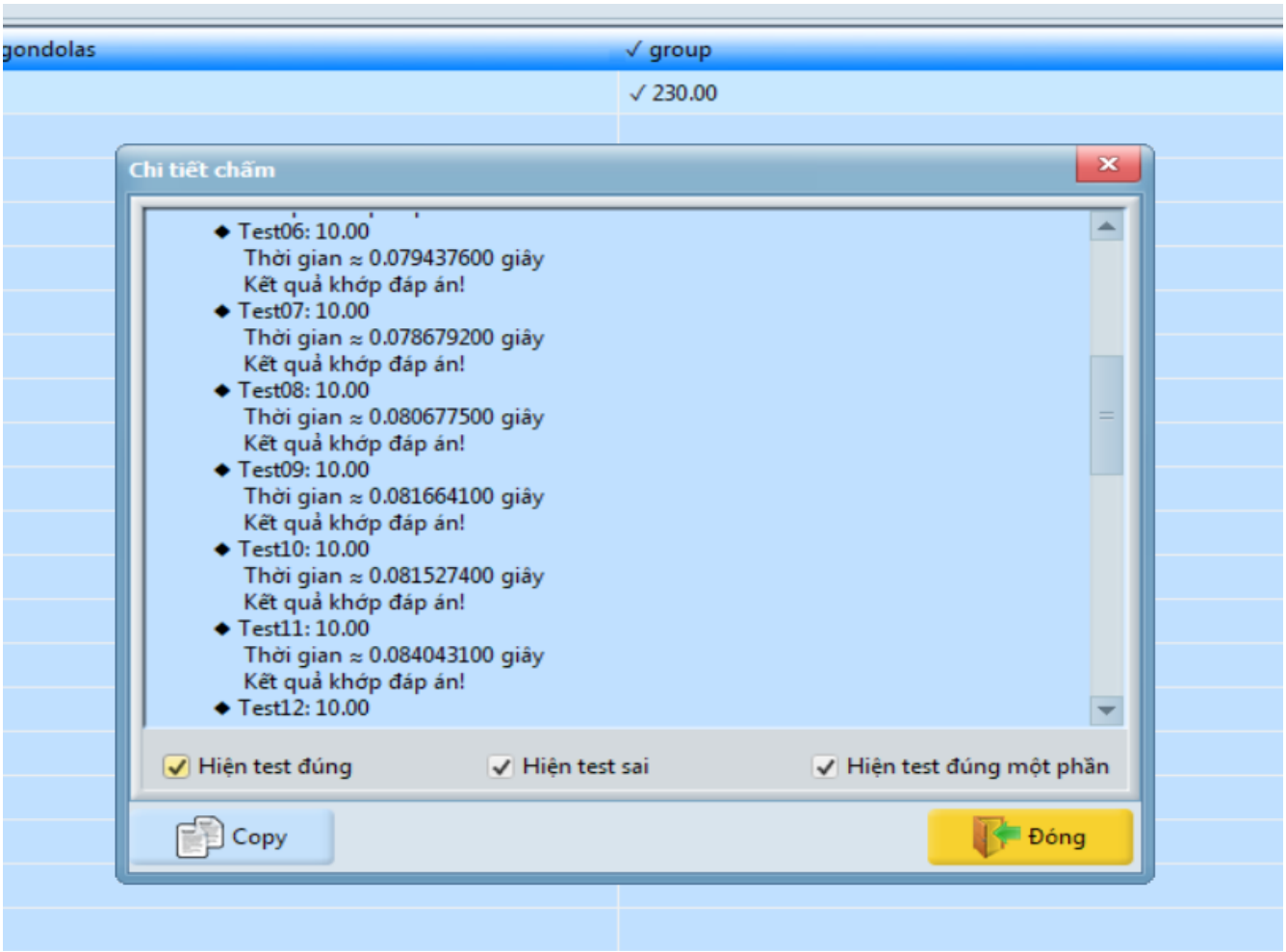
Trong bộ thử nghiệm được sinh ra, chúng ta có những test sau:

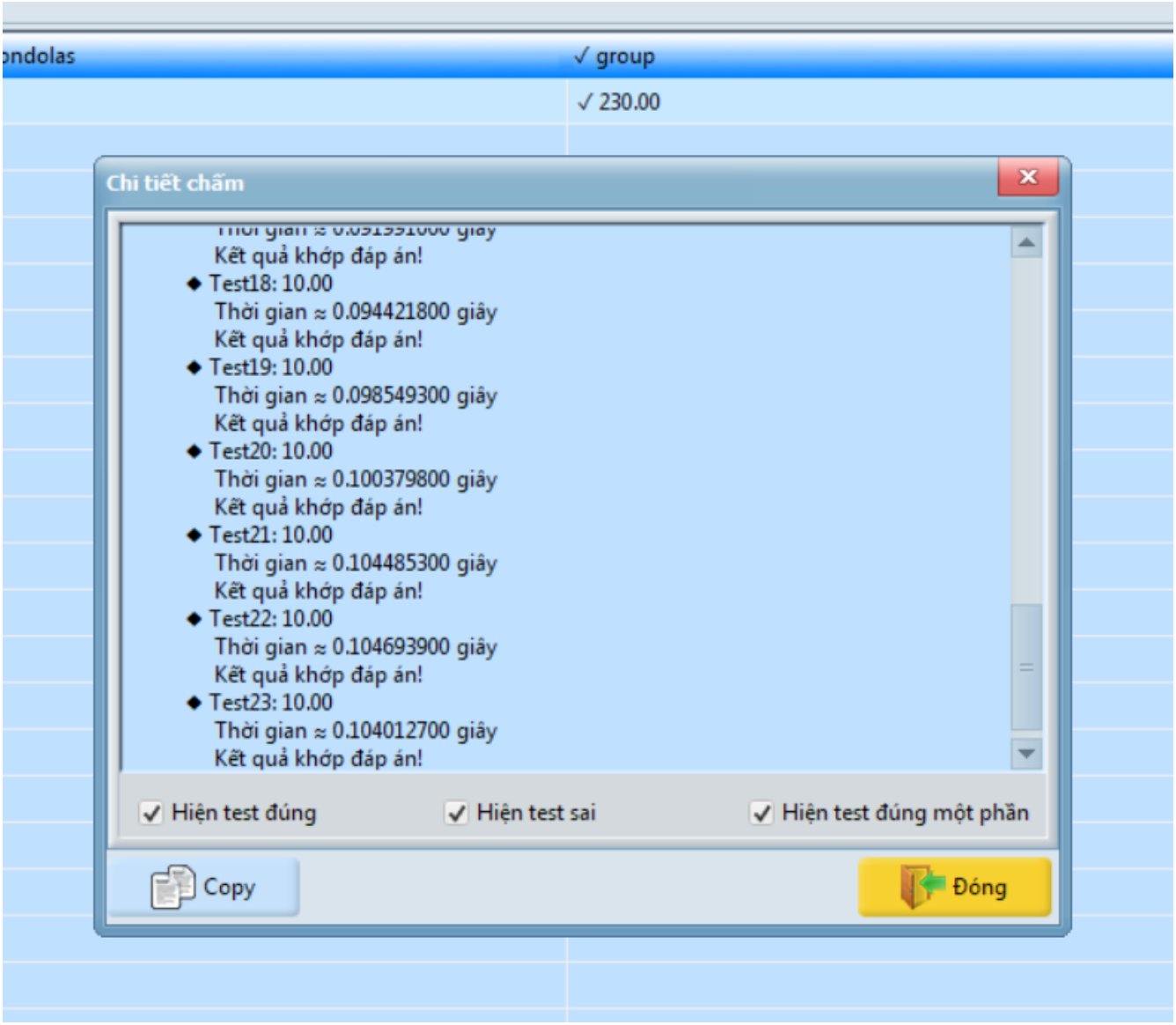
- 1 test với n = 5000 cho trường hợp các cặp số có cùng giá trị y, còn gía trị x được sinh ngẫu nhiên (trường hợp tất cả các đường thẳng đều song song với nhau)
- 1 test với n = 20 cho trường hợp tất cả các cặp số đều giống nhau
- 1 test với n = 30000, trong đó các cặp số được sắp xếp giảm dần theo x
- 5 test với n nằm trong khoảng 3000 → 5000
- 5 test với n nằm trong khoảng 6000 → 10000
- 10 test với n nằm trong khoảng 10000 → 29999

Trong đó, hai test đầu sẽ kiểm tra hai trường hợp đặc biệt đó là kết hợp đường thẳng song song với hoặc điểm truy vấn trùng nhau, hoặc điêm truy vấn khác nhau. Test thứ 3 kiểm tra thời gian cho thủ tục sắp xếp. Các test sau chủ yếu dùng để kiểm tra về tốc độ tăng thời gian chạy của thuật toán trong các khoảng của n nhận giá trị lớn.

Kết quả

- Trình chấm Themis





- SPOJ

ID	GIỜ NỘP	Tài khoản:	PROBLEM	RESULT	TIME	MEM	NGÔN NGỮ
22907082	2018-12-19 11:39:08	Zoe2i	Phân nhóm	accepted edit run	0.20	29M	CPP14

2 Kĩ thuật chia để trị

Kĩ thuật chia để trị được áp dụng với lớp bài toán có dạng như sau:

$$dp[i][j] = min_{k<j}(dp[i-1][k] + C[k][j])$$

Với điều kiện để áp dụng đó là điểm chia phải thỏa mãn điều kiện như sau:

$$R[i][j] \leq R[i][j+1]$$

Nhờ vào tính chất đơn điệu của điểm chia, ta có thể giảm được độ phức tạp của một số thuật toán từ dạng đa thức về dạng logarit. Sau đây là ví dụ để thấy rõ điều này:

2.1 Bài toán 2: Ciel and Gondolas

- File dữ liệu vào: ciel_and_gondolas.inp
- File kết quả: ciel_and_gondolas.out
- Hạn chế thời gian: 4 giây
- Hạn chế bộ nhớ: 512Mb

Đề bài: Có n người đang đợi trong một hàng đợi để đi lên một máy bay có k khoang. Nhân viên máy bay muốn xếp n người này vào khoang theo thứ tự từ khoang thứ nhất đến khoang thứ k. Thứ tự người vào khoang máy bay phải tuân theo thứ tự hàng đợi. Ví dụ: bạn có thể xếp q_1 người liên tiếp đầu tiên trong hàng đợi vào khoang thứ nhất, q_2 người liên tiếp sau q_1 người đầu tiên trong hàng đợi vào khoang thứ hai, ..., q_k người cuối cùng vào khoang thứ k.
Như vậy $\sum_{i=1}^{k}(q_i) = n$. Tuy nhiên, trong những người đó, có những người khó chịu với nhau. Gọi A là mảng $n * n$ số nguyên trong đó A[i][j] thể hiện mức độ khó chịu của người thứ i với thứ j. Tìm cách xếp sao cho:

- Mỗi khoang có ít nhất một người
- Tổng mức độ khó chịu là nhỏ nhất, trong đó mức độ khó chịu của một khoang được định nghĩa là tổng mức độ khó chịu đôi một của những người trong khoang đó.

Dữ liệu vào Dòng đầu là hai số nguyên n và k ($1 \leq n \leq 4000$ và $1 \leq k \leq min(n, 800)$) tương ứng với số lượng khách và số khoang.
n dòng tiếp theo, mỗi dòng gồm n số nguyên tương ứng với ma trận A ($0 \leq A_{ij} \leq 9$, $A_{ij} = A_{ji}$ và $A_{ii} = 0$)

Kết quả In ra tổng mức độ khó chịu nhỏ nhất có thể đạt được.

Ví dụ

ciel_and_gondolas.inp	ciel_and_gondolas.out
3 2 0 2 0 2 0 3 0 3 0	2

2.2 Phân tích

Chi phí của bài toán ở đây chính là tổng mức độ khó chịu ứng với một cách sắp xếp xác định. Gọi $C[i][j]$ là chi phí để xếp j người đầu tiên của hàng đợi vào i khoang đầu tiên. Khi đó ta có công thức:

$$C[i][j] = \begin{cases} 0, & \text{i} = \text{j} \\ \min_{i \le \ell < j}(C[i-1,\ell] + D[\ell+1,j]), & \text{i} < \text{j} \end{cases} \qquad (1)$$

Trong đó: $D[i][j]$ là chi phí khi xếp từ người thứ i tới người thứ j vào cùng một khoang. Khi đó công thức (1) được hiểu như sau:

- Nếu xếp i người vào i khoang, mà mỗi khoang cần có tối thiểu một người. Do vậy mỗi người sẽ được xếp vào một khoang nên chi phí bằng 0.
- Nếu j > i, chi phí để xếp j người đầu vào i khoang đầu sẽ bằng chi phí để xếp ℓ người đầu vào i - 1 khoang đầu cộng với chi phí để xếp j - ℓ người còn lại vào khoang thứ i.

Dựa vào định nghĩa của D ta có công thức tính D như sau:

$$D[i][j] = \sum_{i \le p < q \le j} A[p,q] \qquad (2)$$

Từ đây ta có thể tính D trong thời gian $O(n^2)$ với thủ tục như sau:

```
ComputeDissimilarity(A[1,2,...,n][1,2,...,n]):
  for i ← 1 to n
    D[i,i] ← 0
    B[i,i] ← 0
  for d ← 1 to n − 1
    for i ← 1 to n − d
      B[i,i+d] ← B[i+1,i+d] + A[i,i+d]   [[B[i,j] = Σ_{i≤ℓ<j} A[ℓ,j]]]
      D[i,i+d] ← D[i,i+d−1] + B[i,i+d]
```

Cài đặt không sử dụng chi để trị

Dựa vào công thức (1) ta tính C theo cách thông thường trong thời gian $O(kn^2)$ theo thủ tục:

```
SlowDynamicProgram(A[1,2,...,n][1,2,...,n]):
  ComputeDisimilarity(A[1,2,...,n][1,2,...,n])
  for i ← 1 to k
    for j ← i + 1 to n
      tmp ← +∞
      for ℓ ← i to j
        tmp ← min{ tmp, C[i − 1, ℓ] + D[ℓ + 1, j]}
      C[i, j] ← tmp
  return C[k, n]
```

Cài đặt theo kĩ thuật chia để trị

Khi áp dụng kĩ thuật chi để trị, ta có thể giảm được độ phức tạp của thuật toán thành $O(n^2 + knlogn)$.
Trước tiên, ta cần kiểm tra điều kiện để có thể áp dụng được chia để trị. Ta có $R[i][j]$ được định nghĩa như sau:

$$R[i][j] = argmin_{i \le l < j}(C[i-1,\ell] + D[\ell+1, j])$$

Ta cần chứng minh:

$$R[i][j] \le R[i][j+1] \ (3)$$

Chứng minh: Ta sẽ sử dụng phương pháp phản chứng để chứng minh (3). Với $i \le r < j$ ta định nghĩa $C_r[i,j] = C[i-1,r] + D[r+1,j]$. Đặt $r_1 = R[i,j]$ và $r_2 = R[i,j+1]$. Theo định nghĩa của r_2, ta có:

$$C_{r_2}[i,j+1] \le C_{r_1}[i,j+1] \qquad (*)$$

Giả sử $r_2 < r_1$. Khi đó từ (*) ta có:

$$C[i-1,r_2] + D[r_2+1,j+1] \le C[i-1,r_1] + D[r_1+1,j+1]$$

$$\Leftrightarrow C[i-1,r_2] + D[r_2+1,j] + \sum_{k=r_2+1}^{j} A[j+1,k] \le C[i-1,r_1] + D[r_1+1,j] + \sum_{k=r_1+1}^{j} A[j+1,k]$$

$$\Leftrightarrow C_{r_2}[i,j] + \sum_{k=r_2+1}^{j} A[j+1,k] \le C_{r_1}[i,j] + \sum_{k=r_1+1}^{j} A[j+1,k]$$

Do $\sum_{k=r_2+1}^{j} A[j+1,k] \ge \sum_{k=r_1+1}^{j} A[j+1,k]$ nên:

$$C_{r_2}[i,j] \le C_{r_1}[i,j]$$

Bất đẳng thức này trái với định nghĩa của r_1 do đó $r_1 \le r_2$ (dpcm).

Từ đây, ta thấy R[i] là mảng một chiều với các giá trị tăng dần. Ta có thể tìm được $R[i,i] = i$ và $R[i,n]$ trong thời gian $O(n)$ bằng cách thay đổi một chút trong thủ tục tính C không sử dụng chia để trị ở trên. Như vậy, ta biết các điểm chia trong đối với hàng i của mảng C sẽ nằm trong khoảng $[i, R[i,n]]$. Dựa trên ý tưởng của tìm kiếm nhị phân ta có thể tính $C[i,m]$ (với $m = (i+n)/2$) và $R[i,m]$ Dựa vào bất đẳng thức (3) ta biết rằng $R[i,i+1], \ldots, R[i,m-1]$ nằm trong khoảng $[R[i,i], R[i,m]$ và $R[i,m+1], \ldots, R[i,n]$ nằm trong khoảng $[R[i,m], R[i,n]]$. Do đó ta có thể đệ qui hai dãy con này. Chi tiết như sau:

```
FastDynamicProgram(A[1,2,...,n][1,2,...,n]):
  ComputeDisimilarity(A[1,2,...,n][1,2,...,n])
  for i ← 1 to k
    R[i,i] ← i
    R[i,n] ← ComputeR(i,n,i,n)
    DivConDynamic(i,i+1,n-1,i,R[i,n])
  return C[k,n]
```

```
DivConDynamic(i,x,y,L,R):
  if y - x ≤ 2
    use bruteforce
  else
    m ← ⌊(x+y)/2⌋
    R[i,m] ← ComputeR(i,m,L,R)
    DivConDynamic(i,x,m-1,L,R[i,m])
    DivConDynamic(i,m+1,y,R[i,m],R)
```

```
ComputeR(i,j,L,R):
  tmp ← +∞
  p ← L
  for ℓ ← L to R
    tmp ← min{ tmp, C[i-1,ℓ] + D[ℓ+1,j]}
    p ← ℓ
  C[i,j] ← tmp
  return p
```

Thủ tục $DivConDynamic(i, x, y, L, R)$ dùng để tính $C[i, x], C[i, x+1], \ldots, C[i, y]$ nếu biết điểm chia nằm trong khoảng $[L, R]$. Thủ tục $ComputeR(i, j, L, R)$ dùng để tính $C[i, j]$ và $R[i, j]$ biết điểm chia đó nằm trong khoảng $[L, R]$.

Phân tích độ phức tạp thuật toán

- Về mặt bộ nhớ: Chúng ta sử dụng mảng C có kích thước kxn, mảng A, D, B có kích thước $n * n$. Do đó sẽ là $O(n^2)$.
- Về mặt thời gian: Trước hết ta phân tích thời gian của thủ tục đệ quy DivConDynamic. Gọi $T(n, m)$ là thời gian tính của $DivConDynamic(i, x, y, L, R)$ trong đó $n = y-x$ và $m = R-L$. Thủ tục $ComputeR(i, j, L, R)$ được thực hiện trong thời gian $O(R - L) = O(m)$. Nếu:
  - $m = O(1)$ thì $T(n, m) = T(n) = 2T(n/2) + O(1) = O(n)$
  - $n = O(1)$ thì $T(n, m) = O(m)$ (trường hợp sử dụng Brute force)
  - $T(n, m) = T(n/2, k) + T(n/2, m - k) + O(m)$ với k = R[i, m] - L

  Ta sẽ tìm T(n, m) bằng dự đoán $T(n, m) = O((m + n)logn)$.
  Giả sử $T(n, m) \leq a(m + n)logn$ với $a > 0$. Khi đó $T(n/2, k) + T(n/2, m - k) + cm \leq a(n/2+k)log(n/2)+a(n/2+m-k)log(n/2)+cm = a(m+n)log(n/2)+cm < a(m+n)logn$ với $a > c/log2$
  Do đó, thời gian tính tủa thuật toán là $O(n^2 + knlogn)$ với $O(n^2)$ là thao tác tính D và $O(knlogn)$ là thao tác tính C.

Một số chú ý khi cài đặt

- R[i, j] chỉ được sử dụng làm điểm chia, không được sử dụng để tính các giá trị khác trong bảng R, do vậy ta chỉ cần sử dụng R[i, j] một lần. Do đó khi cài đặt ta có thể sử dụng một biến nhớ R để lưu lại giá trị trả về của thủ tục ComputeR được sử dụng trong thủ tục FastDynamicProgram và DivConDynamic.
- Một chú ý nữa khi submit trên Codeforce đó là chỉnh sửa một chút cách đọc input để có thể được Accept. Cụ thể do các phần tử của mảng A lớn nhất là 9 nên ta có thể đọc từng phần tử vào một kí tự c, sau đó thự hiện $c -' 0'$ để tính ra giá trị một phần tử của mảng A.

2.3 Kết quả thử nghiệm

Mô tả bộ thử ngiệm

Ràng buộc của bài toán:

- $1 \leq n \leq 4000$
- $1 \leq k \leq min(n, 800)$
- $0 \leq A_{ij} = A_{ji} \leq 9$

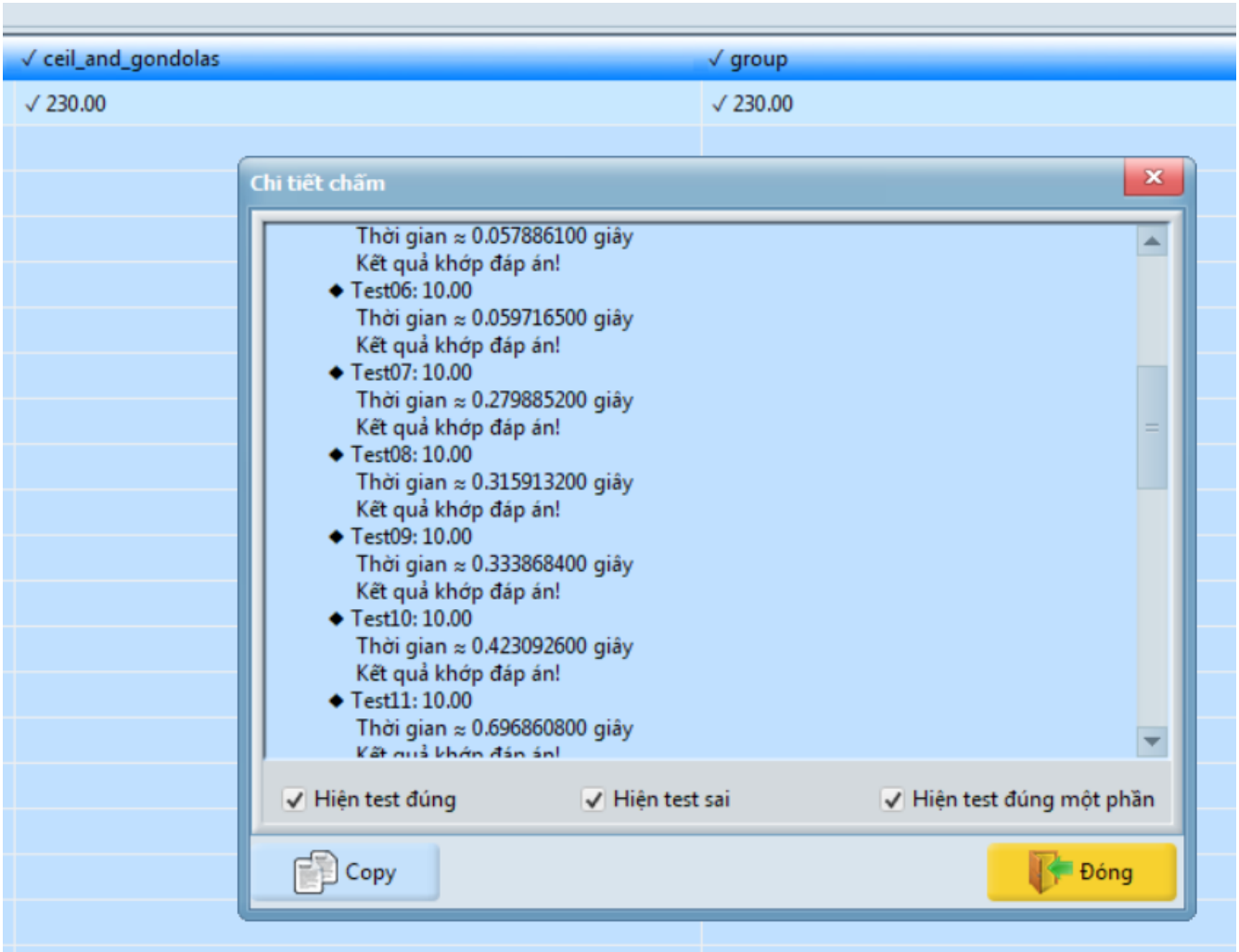
Các test có trong bộ test:

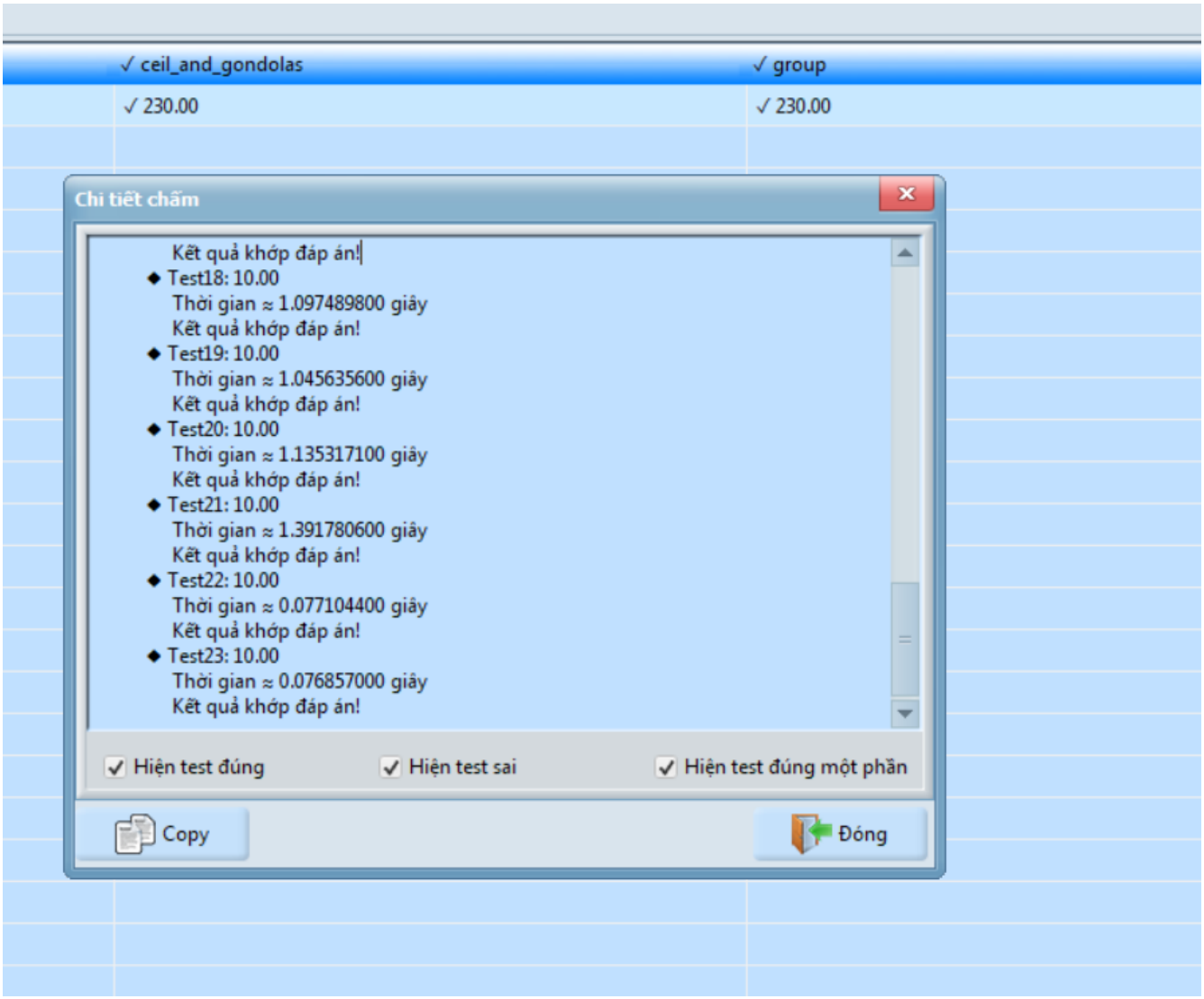
- 1 test với n trong khoảng $10 \rightarrow 50$ và k = n+1

- 5 test với n trong khoảng $100 \to 500$ và k trong khoảng $1 \to 20$
- 5 test với n trong khoảng $300 \to 500$ và k trong khoảng $100 - \frac{n}{2}$
- 5 test với n trong khoảng $1000 \to 2000$, k trong khoảng $1 \to 200$
- 5 test với n trong khoảng $2000 \to 3999$, k trong khoảng $200 \to 800$
- 2 test với $n = 4000$ với k trong khoảng $1 \to 100$ và k trong khoảng $500 \to 800$

Kết quả

- Trình chấm Themis





- Codeforce

General

#	Author	Problem	Lang	Verdict	Time	Memory	Sent	Judged		
47160065	Practice: zoe2i	321E - 19	GNU C++11	Accepted	872 ms	200964 KB	2018-12-17 12:16:52	2018-12-17 12:16:52		Compare

Tài liệu tham khảo

[1] Kĩ thuật nâng cao trong quy hoạch động (Nguồn giaithuatlaptrinh)

[2] Một số kĩ thuật tối ưu hóa quy hoạch động (Nguồn vnoi)

[3] Kĩ thuật bao lồi (Nguồn vnoi)