# Dynamic Programming

# Phương pháp quy hoạch động

Các bài toán *quy hoạch động* chiếm một vị trí khá quan trọng trong tổ chức hoạt động và sản xuất. Chính vì lẽ đó mà trong các kì thi học sinh giỏi quốc gia và quốc tế chúng ta thường gặp loại toán này.

Thông thường những bạn nào dùng phương pháp quay lui, vét cạn thì chỉ có thể vét được các tập dữ liệu nhỏ, kích thước chừng vài chục byte. Nếu tìm được đúng hệ thức thể hiện bản chất quy hoạch động của bài toán và khéo tổ chức dữ liệu thì ta có thể xử lí được những tập dữ liệu khá lớn.

Có thể tóm lược nguyên lí quy hoạch động do Bellman phát biểu như sau:

|  |
| --- |
| **Quy hoạch động** |
| *Quy hoạch động là lớp các bài toán mà quyết định ở bước thứ i phụ thuộc vào quyết định ở các bước đã xử lí trước hoặc sau bước đó.* |

Để giải các bài toán quy hoạch động, ta có thể theo sơ đồ đơn giản sau đây:

|  |
| --- |
| **Sơ đồ giải bài toán quy hoạch động** |
| 1. *Lập hệ thức:* Lập hệ thức biểu diễn tương quan quyết định của bước đang xử lí với các bước đã xử lí trước hoặc sau bước đó. Hệ thức này được gọi là *phương trình Bellman.* Khi đã có hệ thức tương quan, chúng ta đã có thể xây dựng ngay thuật giải, tuy nhiên các hệ thức này thường là các biểu thức *đệ quy*, do đó dễ gây ra hiện tượng tràn miền nhớ và tốn thời gian khi ta tổ chức chương trình trực tiếp bằng đệ quy. 2. *Tổ chức dữ liệu và chương trình:* Tổ chức dữ liệu tính toán dần theo từng bước. Nên tìm cách *khử đệ quy*. Muốn khử đệ quy ta cần lưu lại các giá trị đã tính tại mỗi bước. Trong các bài toán quy hoạch động thuộc chương trình phổ thông, thường đòi hỏi một vài mảng một hoặc hai chiều (1D, 2D) đảm nhận việc lưu trữ này. 3. *Làm tốt:* Làm tốt thuật toán bằng cách thu gọn hệ thức quy hoạch động và giảm kích thước miền nhớ: thay mảng hai chiều bằng một vài mảng một chiều. |

## Chia thưởng

Cần chia hết m phần thưởng cho n học sinh sắp theo thứ tự từ giỏi trở xuống sao cho mỗi bạn không nhận ít phần thưởng hơn bạn xếp sau mình. 1 ≤ m, n ≤ 70.

Hãy tính số cách chia?

Ví dụ, Có 11 cách chia 7 phần thưởng cho 4 học sinh theo yêu cầu của đầu bài. Đó là:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Phương án | Học sinh | | | |
| ① | ② | ③ | ④ |
| 1 | 7 | 0 | 0 | 0 |
| 2 | 6 | 1 | 0 | 0 |
| 3 | 5 | 2 | 0 | 0 |
| 4 | 5 | 1 | 1 | 0 |
| 5 | 4 | 3 | 0 | 0 |
| 6 | 4 | 2 | 1 | 0 |
| 7 | 3 | 3 | 1 | 0 |
| 8 | 3 | 2 | 2 | 0 |
| 9 | 4 | 1 | 1 | 1 |
| 10 | 3 | 2 | 1 | 1 |
| 11 | 2 | 2 | 2 | 1 |

Bài này còn có cách phát biểu khác như sau: Có bao nhiêu cách viết số tự nhiên m thành tổng của n số hạng theo trật tự giảm dần: số hạng viết sau không lớn hơn số hạng viết trước nó.

Thuật toán

Lập hệ thức quy hoạch động

Gọi Chia(*pt, hs*) là số cách chia *pt* phần thưởng cho *hs* học sinh, ta thấy:

* Nếu không có học sinh (*hs* = 0) thì không có cách chia nào (Chia(pt, 0) = 0).
* Nếu không có phần thưởng (*pt* = 0) thì chỉ có một cách chia (Chia(0, *hs*) = 1: mỗi học sinh nhận 0 phần thưởng).
* Nếu số phần thưởng ít hơn số học sinh (*pt* < *hs*) thì trong mọi phương án chia, từ học sinh thứ *pt* + 1 trở đi sẽ không được nhận phần thưởng nào:
* Chia(*pt,hs*) = Chia(*pt, pt*) nếu *pt* < *hs*.
* Ta xét tất cả các phương án chia trong trường hợp *pt* ≥ *hs*. Ta tách các phương án chia thành *hai nhóm không giao nhau* dựa trên số phần thưởng mà học sinh đứng cuối bảng thành tích, học sinh thứ *hs*, được nhận:
* Nhóm thứ nhất gồm các phương án trong đó học sinh cuối bảng không được nhận thưởng, tức là *pt* phần thưởng chỉ chia cho *hs*-1 học sinhđầu danh sách và do đó, số cách chia, tức là số phần tử của nhóm này sẽ là: Chia(*pt, hs*-1).
* Nhóm thứ hai gồm các phương án còn lại, trong đó học sinh cuối bảngcũng được nhận thưởng. Khi đó, do học sinh đứng cuối bảng thành tích được nhận thưởng, thì mọi học sinh khác cũng sẽ có thưởng. Do ai cũng được thưởng nên ta bớt của mỗi người một phần thưởng (để họ lĩnh sau), số phần thưởng còn lại (*pt*-*hs*) sẽ được chia cho *hs* học sinh. Số cách chia khi đó sẽ là Chia(*pt*-*hs, hs*).

Tổng số cách chia cho trường hợp *pt* ≥ *hs* sẽ là tổng số phần tử của hai nhóm, ta có:

Chia(*pt, hs*) = Chia(*pt, hs-1*) + Chia(*pt-hs, hs*)

Tổng hợp lại, hàm *Chia(pt,hs)* có các tính chất sau đây:

|  |  |
| --- | --- |
| *Điều kiện* | *Chia(pt, hs)* |
| *hs* = *0:* | *0* |
| *pt* = *0 and hs ≠ 0:* | *1* |
| *pt* < *hs:* | *Chia(pt, pt)* |
| *pt* ≥ *hs:* | *Chia(pt, hs–1)+Chia(pt–hs, hs)* |

Phương án đệ quy

Ta có phương án đầu tiên của giải thuật Chia như sau:

Chương trình

"""

Chia thưởng. Phương án 1: Đệ quy

"""

from time import time

def Go(msg=' ? '):

if input(msg) == '.': exit(0)

def Chia(pt, hs):

if hs > pt: hs = pt

if hs == 0: return 1 if pt == 0 else 0

# hs > 0

return 1 if pt == 0 else Chia(pt, hs-1) + Chia(pt-hs, hs)

def Run():

print(Chia(0, 0)) # 1

print(Chia(7, 0)) # 0

print(Chia(0, 4)) # 1

print(Chia(0, 0)) # 1

print(Chia(7, 4)) # 11

# APPLICATION

t = time()

Run()

print(time()-t)

print(' T h e E n d')

Phương án này chạy chậm vì phát sinh ra quá nhiều lần gọi hàm trùng lặp. Bảng dưới đây liệt kê số lần gọi hàm Chia khi giải bài toán chia thưởng với 7 phần thưởng và 4 học sinh. Ví dụ, hàm Chia(1,1) sẽ được gọi 9 lần,… Tổng số lần gọi hàm Chia là trên 70 lần cho lời giải Chia(7,4) chỉ để phat hiện ra 11 phương án là quá tốn kém.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  | hs | | | | |
|  |  | ⓿ | ❶ | ❷ | ❸ | ❹ |
|  | ⓪ | 0 | 9 | 1 | 1 | 0 |
|  | ① | 9 | 9 | 2 | 1 | 0 |
|  | ② | 6 | 6 | 1 | 0 | 0 |
|  | ③ | 5 | 5 | 2 | 1 | 1 |
| pt | ④ | 3 | 3 | 1 | 1 | 0 |
|  | ⑤ | 2 | 2 | 1 | 0 | 0 |
|  | ⑥ | 1 | 1 | 0 | 0 | 0 |
|  | ⑦ | 1 | 1 | 1 | 1 | 1 |
|  | *Số lần gọi hàm Chia cục bộ*  *khi tính hàm Chia*(⑦,❹) | | | | | |

Bạn có thể kiểm tra sự tốn kém này bằng cách đặt thêm một biến đếm tổng thể call để đếm số lần gọi hàm Chia như sau:

Chương trình

""" Chia thưởng. Phương án 1: Đệ quy

"""

from time import time

def Go(msg=' ? '):

if input(msg) == '.': exit(0)

def Chia(pt, hs):

global call

call += 1

if hs > pt: hs = pt

if hs == 0: return 1 if pt == 0 else 0

# hs > 0

return 1 if pt == 0 else Chia(pt, hs-1) + Chia(pt-hs, hs)

def Run():

global call

call = 0

print(Chia(7, 4))

print('Số lần gọi hàm Chia:', call)

# APPLICATION

t = time()

Run()

print(time()-t)

print(' T h e E n d')

Cải tiến lần 1

Phương án đệ quy khá dễ triển khai nhưng chương trình sẽ chạy rất lâu, bạn hãy thử gọi Chia(66,32) để trải nghiệm được điều trên. Diễn tả đệ quy thường trong sáng, nhàn tản, nhưng khi thực hiện sẽ sinh ra hiện tượng gọi lặp lại những lần gọi hàm. Cải tiến đầu tiên là *tránh những lần gọi lặp* như vậy. Muốn thế, chúng ta tính sẵn các giá trị của hàm *Chia* theo các trị của đầu vào khác nhau và điền vào một mảng hai chiều cc theo công thức:

cc[pt][hs] = Chia(pt, hs)

Theo các tính chất của hàm hai ngôi Chia, ta có

|  |
| --- |
| cc[pt][hs] = Chia(pt, hs) |
| = 0, nếu hs = 0 và pt > 0 |
| = 1, nếu hs > 0 và pt = 0 |
| = cc[pt][pt], nếu 0 < pt < hs |
| = cc[pt][hs-1] + cc[pt-hs][hs], nếu pt ≥ hs > 0 |

Từ đó ta suy ra quy trình điền trị vào mảng hai chiều cc như sau:

* Dòng 0: Tại dòng 0 ta điền toàn 1 vì không có phần thưởng thì có 1 phương án chia là mỗi hs nhận 0 phần thưởng.
* Cột 0: Tại cột 0 ta điền toàn 0, vì không có hs mà có pt thì không thể chia hết pt được. Riêng cc[0][0] = 1 theo quy ước.
* Cột 1: Tại cột 1 ta điền toàn 1, vì chỉ có một hs nên có bao nhiêu phần thưởng đều dành cho em đó. Như vậy là chỉ có 1 cách chia.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | *0* | *1* | *…* | *j* | *…* |  | *hs* |
| *0* | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| *1* | 0 | 1 |  |  |  |  | 1 |
| *…* |  | 1 |  | \* |  |  |  |
| *...* |  | 1 |  | ... |  |  |  |
| *i* |  | 1 | \* | ? |  |  |  |
| *…* |  | 1 |  |  |  |  |  |
| *pt* | 0 | 1 |  |  |  |  |  |

Các phần tử cc[i][j]còn lại sẽ được điền theo cột, từ cột j = 2 đến j = hs. Mỗi cột j sẽ được điền trị như sau:

* Giữ nguyên các trị cc[i][j] của cột trước với i < j: cc[i][j] = cc[i][j-1] vì khi số phần thưởng i ít hơn hoặc bằng số học sinh j thì kết quả sẽ là Chia(i, j) = cc[i][i].
* Các giá trị còn lại của cột j sẽ được điền từ trên xuống theo hệ thức

cc[i][j] = cc[i][j-1] + cc[i-j][j], i = j .. m

Chương trình

"""

Chia thưởng.

Phương án 2: mảng 2D

"""

from time import time

def Go(msg=' ? '):

if input(msg) == '.': exit(0)

def Chia(pt, hs):

if hs > pt: hs = pt

if hs == 0: return 1 if pt == 0 else 0 # khong co hs

# hs > 0: Điền mảng pt+1 dòng, hs+1 cột

cc = [[1]\*(hs+1)] # dòng 0: toàn 1, cột 1: toàn 1

for j in range(pt): cc.append([0,1] + [0]\*(hs-1))

# điền theo cột 2..hs

for j in range(2, hs+1):

# bảo lưu j giá trị đầu

for i in range(0, j): cc[i][j] = cc[i][j-1]

for i in range(j, pt+1): cc[i][j] = cc[i][j-1] + cc[i-j][j]

return cc[pt][hs]

def Run():

print(Chia(7, 4)) # 11

print(Chia(70, 14)) # 1614987

print(Chia(66, 32)) # 2269557

# APPLICATION

t = time()

Run()

print(time()-t)

print(' T h e E n d')

Cải tiến lần 2

Dùng mảng hai chiều chúng ta chỉ có thể tính toán được với dữ liệu không quá lớn. Bước cải tiến sau đây khá quan trọng: chúng ta dùng mảng một chiều.

Quan sát kỹ quy trình gán trị cho mảng hai chiều theo từng cột chúng ta dễ phát hiện ra rằng cột thứ *j* có thể được tính toán từ cột thứ *j* - 1. Hơn nữa, khi tính toán tại bước j ta có, nếu i < j thì

c[i] tại bước j = cc[i][j] = cc[i][j-1] = giá trị c[i] tại bước j-1

nghĩa là giá trị c[i] tại bước j được bảo lưu khi i < j.

Ta có phương án 3: dùng một mảng một chiều c như sau:

Chương trình

""" Chia thưởng.

Phương án 3: mảng 1D

"""

from time import time

def Go(msg=' ? '):

if input(msg) == '.': exit(0)

def Chia(pt, hs):

if hs > pt: hs = pt

if hs == 0: return 1 if pt == 0 else 0 # không có hs

# hs > 0

c = [1]\*(pt+1) # cot 1 toan 1

# điền theo cột j = 2..hs

for j in range(2, hs+1):

for i in range(j, pt+1): c[i] += c[i-j]

return c[pt]

def Run():

print(Chia(7, 4)) # 11

print(Chia(70, 14)) # 1614987

print(Chia(66, 32)) # 2269557

# APPLICATION

t = time()

Run()

print(time()-t)

print(' T h e E n d')

# Luyện tập Phương pháp quy hoạch động

## Palindrome

*Olympic Tin học Quốc tế, 2000, Bắc Kinh.*

Dãy ký tự s được gọi là đối xứng (palindrome) nếu các phần tử cách đều đầu và cuối giống nhau. Cho dãy s tạo bởi n ký tự gồm các chữ cái hoa và thường phân biệt và các chữ số. Hãy cho biết cần xoá đi từ s ít nhất là bao nhiêu ký tự để thu được một dãy đối xứng. Giả thiết rằng sau khi xoá một số ký tự khỏi s thì các ký tự còn lại sẽ tự động xích lại sát nhau.

Ví dụ, với dãy s gồm 9 ký tự, s = 'baeadbadb' thì cần xoá ít nhất 4 ký tự, chẳng hạn, các ký tự thứ 4, 6, 7 và 8, để thu được dãy đối xứng chiều dài 5 là baeab:

baeadbadb → baeab

Dĩ nhiên là có nhiều cách xoá. Tuy nhiên đáp số là số ít nhất các ký tự cần loại bỏ khỏi s là duy nhất và bằng 4.

## Flowers

Olympic Tin học Quốc tế, 1999.

Cần cắm hết k bó hoa khác nhau vào n lọ xếp thẳng hàng sao cho bó hoa có số hiệu nhỏ được đặt trước bó hoa có số hiệu lớn. Với mỗi bó hoa i ta biết giá trị thẩm mĩ khi cắm bó hoa đó vào lọ j là v[i][ j].

Yêu cầu: Xác định một phương án cắm hoa sao cho tổng giá trị thẩm mĩ là lớn nhất.

Dữ liệu vào ghi trong tệp văn bản FLOWER.INP:

Dòng đầu tiên là hai số k và n.

Từ dòng thứ hai trở đi là các giá trị nguyên v[i][ j] trong khoảng 0..10, với i = 1..k và j = 1..n; 1 ≤ k ≤ n ≤ 100.

Dữ liệu ra ghi trong tệp văn bản FLOWER.OUT: dòng đầu tiên là tổng giá trị thẩm mĩ của phương án cắm hoa tối ưu. Từ dòng thứ hai là dãy k số hiệu lọ được chọn cho mỗi bó hoa.

Các số liệu vào và ra đều là số tự nhiên và được ghi cách nhau trên mỗi dòng.

Ví dụ

|  |  |
| --- | --- |
| FLOWER.INP | FLOWER.OUT |
| 4 6  1 1 3 4 3 10  9 1 4 7 2 7  7 2 6 10 2 3  6 10 7 1 3 9 | 24  1 3 4 6 |

*Kết quả cho biết tổng giá trị thẩm mĩ sẽ đạt là 24 (điểm)*

*nếu cắm hoa như sau:*

*- Bó hoa 1 cắm vào lọ 1, đạt độ thẩm mĩ 1*

*- Bó hoa 2 cắm vào lọ 3, đạt độ thẩm mĩ 4*

*- Bó hoa 3 cắm vào lọ 4, đạt độ thẩm mĩ 10*

*- Bó hoa 4 cắm vào lọ 6, đạt độ thẩm mĩ 9*

## Bố trí phòng học

Câu lạc bộ - Học sinh giỏi Tin học, Hà Nội, năm 2000

Cần bố trí hết k nhóm học sinh vào k trong số n phòng học chuyên đề sao cho nhóm có số hiệu nhỏ được xếp vào phòng có số hiệu nhỏ hơn phòng chứa nhóm có số hiệu lớn. Với mỗi phòng có xếp học sinh, các ghế thừa phải được chuyển ra hết, nếu thiếu ghế thì phải lấy từ kho vào cho đủ mỗi học sinh một ghế. Biết số học sinh trong mỗi nhóm và số ghế trong mỗi phòng. Hãy chọn phương án bố trí sao cho tổng số lần chuyển ghế ra và chuyển ghế vào là ít nhất.

## Bống

Tấm được Bụt trao cho k chú bống. Nhiệm vụ của Tấm là phải thả k bống này vào k trong số n giếng để chăm nuôi, Các giếng được xếp thẳng hàng theo thứ tự 1..n. Bống có số hiệu nhỏ phải được thả vào giếng số hiệu nhỏ. Nếu nuôi bống b trong giếng g thì sau này bống sẽ sinh ra số quà tặng là v(b, g).

Bạn hãy giúp Tấm thả cá vào các giếng để sau này nhận được số quà nhiều nhất.

A fish and a roll of toilet paper

Description automatically generated

Dữ liệu vào ghi trong tệp văn bản BONG.INP:

Dòng đầu tiên là hai trị k và n.

Từ dòng thứ hai trở đi là các giá trị v(b, g) trong khoảng 0..10, với b = 1..k

và g = 1..n; 1 ≤ b ≤ g ≤ 100.

Kết quả hiển thị trên màn hình một phương án thả bống và tổng giá trị quà của phương án tối ưu.

Ví dụ

|  |
| --- |
| BONG.INP |
| 4 5  1 1 6 4 10  9 1 4 7 7  7 2 6 10 3  6 10 7 1 9 |
|  |

Kết quả cho biết tổng giá trị quà sẽ đạt là 24 (điểm)

nếu thả cá như sau:

* bống 1 vào giếng số 1;
* bống 2 vào giếng số 3;
* bống 3 vào giếng số 4;
* bống 4 vào giếng số 5.

A fish and a roll of toilet paper

Description automatically generated

## Tìm các đường ngắn nhất

Cho một đồ thị có hướng gồm n đỉnh mã số từ 1..n với các cung (u, v) có hướng đi từ đỉnh u đến đỉnh v và có chiều dài thể hiện đường đi nối từ đỉnh u đến đỉnh v. Viết chương trình tìm mọi đường đi ngắn nhất từ một đỉnh s cho trước tới các đỉnh còn lại của đồ thị.

Dữ liệu vào được ghi trong file văn bản tên DIJ.INP có cấu trúc như sau:

Dòng đầu ghi hai số tự nhiên n và s cách nhau, trong đó n là số lượng đỉnh của đồ thị, s là số hiệu của đỉnh xuất phát 2 ≤ n ≤ 100, 1 ≤ s ≤ n.

Từ dòng thứ hai ghi lần lượt độ dài đường đi từ đỉnh i đến đỉnh j. Giá trị 0 cho biết không có cung nối hai đỉnh tương ứng. Với mọi đỉnh i = 1..n, cung (i, i) được hiểu là không tồn tại và đường đi từ đỉnh i tới chính nó có chiều dài là 0. Các số cùng dòng được ghi cách nhau. Dạng dữ liệu cho như vậy được gọi là ma trận kề của đồ thị.

Ví dụ sau đây cho biết đồ thị có bảy đỉnh, cần tìm các đường đi ngắn nhất từ đỉnh s = 2 tới các đỉnh còn lại của đồ thị.

Đỉnh 1 không nối với mọi đỉnh còn lại

Cung (2, 1) có chiều dài 4, cung (2,3) dài 1, …, cung (2,7) dài 5.

…

|  |  |
| --- | --- |
| *DIJ.INP* | *A picture containing text, clock, watch  Description automatically generated* |
| *7 2*  *0 0 0 0 0 0 0*  *4 0 1 0 0 0 5*  *0 0 0 0 0 0 1*  *0 0 0 0 0 2 0*  *0 0 0 3 0 0 0*  *1 0 0 0 0 0 5*  *0 0 0 1 0 0 0* |

Dữ liệu ra cần ghi trong file văn bản DIJ.OUT gồm n dòng. Thông tin về mỗi đường đi ngắn nhất từ đỉnh s đến mỗi đỉnh còn lại được ghi trên 1 dòng. Số đầu tiên của dòng là chiều dài đường đi. Nếu không tồn tại đường đi thì ghi giá trị 0. Tiếp đến, trong trường hợp có đường đi từ đỉnh s đến đỉnh i thì ghi dãy đỉnh xuất hiện lần lượt trên đường đi, đỉnh đầu tiên là s, đỉnh cuối cùng là i. Đường đi từ đỉnh i tới chính đỉnh đó được coi là không tồn tại, i = 1..n. Thí dụ trên cho ta kết quả

|  |
| --- |
| *DIJ.OUT* |
| *4 2 1*  *0*  *1 2 3*  *3 2 3 7 4*  *0*  *5 2 3 7 4 6*  *2 2 3 7* |

*- Đường ngắn nhất từ đỉnh 2 đến đỉnh 1 có chiều dài 4: 2 → 1.*

*- Đường ngắn nhất từ đỉnh 2 đến đỉnh 2: không có (thực ra, theo lẽ thường là có đường chiều dài 0).*

*- Đường ngắn nhất từ đỉnh 2 đến đỉnh 3 có chiều dài 1: 2 → 3.*

*- Đường ngắn nhất từ đỉnh 2 đến đỉnh 4 có chiều dài 3: 2 → 3 → 7 → 4.*

*- Đường ngắn nhất từ đỉnh 2 đến đỉnh 5: không có.*

*- Đường ngắn nhất từ đỉnh 2 đến đỉnh 6 có chiều dài 5: 2→3→7→4→6.*

*- Đường ngắn nhất từ đỉnh 2 đến đỉnh 7 có chiều dài 2: 2→3→7.*

# Bài giải Phương pháp quy hoạch động

## Palindrome

Thuật toán

Bài toán này đã được nhiều bạn đọc công bố lời giải với một mảng hai chiều kích thước *n*×*n* hoặc vài ba mảng một chiều kích thước *n*, trong đó *n* là chiều dài của dữ liệu vào.

Với một nhận xét nhỏ ta có thể phát hiện ra rằng chỉ cần dùng một mảng một chiều kích thước *n* và một vài biến đơn là đủ.

Lập hệ thức quy hoạch động

Gọi *p*(*i*, *j*) là chiều dài của dãy con đối xứng dài nhất thu được khi giải bài toán với dữ liệu vào là đoạn *s*[*i*..*j*]. Khi đó *p*(0, *n-*1) là chiều dài của dãy con đối xứng dài nhất trong dãy s và do đó, số ký tự cần loại bỏ khỏi dãy *s* sẽ là *n-p(0,n-*1*).* Đó chính là đáp số của bài toán.

Ví dụ, với dãy s = baeadbadb, n = 9 thì p(0,8) = 5 và số ký tự cần xóa sẽ là n – p(0,8) = 9–5 = 4.

Ta liệt kê một số tính chất quan trọng của hàm hai biến *p*(*i, j*). Ta có:

* Nếu *i* > *j,* tức là chỉ số đầu trái lớn hơn chỉ số đầu phải, ta quy ước *p*(*i, j*) = 0.
* Nếu *i* = *j* thì *p*(*i, i*) = 1 vì dãy khảo sát chỉ chứa duy nhất một ký tự s[i] nên nó là đối xứng.
* Nếu *i* < *j* và *s*[*i*] = *s*[*j*] thì *p*(*i, j*) = *p*(i+*1*, *j*–1)+2. Vì hai ký tự đầu và cuối dãy *s*[*i..j*] giống nhau nên chỉ cần xác định chiều dài của khúc giữa rồi cộng thêm 2 đơn vị ứng với hai ký tự đầu và cuối dãy.
* Nếu *i* < *j* và *s*[*i*] ≠ *s*[*j*], tức là hai ký tự đầu và cuối của dãy con *s*[*i*..*j*] là khác nhau thì ta khảo sát hai dãy con là *s*[*i*..*j*–1] và *s*[*i*+1..*j*] để lấy chiều dài của dãy con đối xứng dài nhất trong hai dãy này làm kết quả:

p(i,j) = max(p(i,j-1),p(i+1,j))

Vấn đề đặt ra là cần tính *p*(0, *n-1*). Mà muốn tính được *p*(0, *n-*1) ta phải tính được các *p*(*i, j*) với mọi *i, j* = 0..*n-*1*.*

Phương án đệ quy

Chương trình

# Palindrome. Phương án 1. Đệ quy

from time import time

def Go(msg = ' ? '):

if input(msg) == '.': exit(0)

# số kí tự nhiều nhất còn lại

def P(i, j):

if i > j: return 0

if i == j: return 1

return P(i+1,j-1)+2 if s[i]==s[j] else max(P(i,j-1),P(i+1,j))

# số kí tự cần xóa

def Pal(inps):

global s

s = inps

print(s)

return len(s)-P(0,len(s)-1)

def Run():

print(' Xóa ', Pal("abcvcbaabcvcba")) # xoa 0

print(' Xóa ', Pal("ab1cvcbaa2bcvc3ba")) # xoa 3

print(' Xóa ', Pal("ab1cvcba2abcvc3ba")) # xoa 2

# APPLICATION

t = time()

Run()

print(time()-t)

print(' T h e E n d')

Cải tiến 1: dùng mảng 2 chiều

Phương án đệ quy sẽ phát sinh các lời gọi hàm trùng lặp như đã phân tích trong bài toán trước.

Ta khắc phục điều này bằng cách sử dụng một mảng hai chiều v để tính trước các giá trị của hàm *p*(*i, j*), mỗi giá trị được tính tối đa một lần :

v[i][j] = p(i,j)

Ta có, theo hệ thức tính p(i,j),

v[i][j] = 0, nếu i > j: nửa tam giác dưới đường chéo chính toàn 0

v[i][i] = 1: đường chéo chính toàn 1

Nếu i < j

v[i][j] = v[i+1][j-1]+2, nếu s[i] = s[j]

v[i][j] = max(v[i][j-1],v[i+1][j]), nếu s[i] ≠ s[j]

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 |  |  |  |  |  |  | ? ô v[i][j] cần tính  + ô v[i+1][j-1] (chéo trái)  \* ô v[i][j-1] (kề trái)  # ô v[i+1][j] (kề dưới)  *Điền trị cho ma trận v* |
| 0 | 1 |  |  | \* | ? |  |
| 0 | 0 | 1 |  | + | # |  |
| 0 | 0 | 0 | 1 |  |  |  |
| 0 | 0 | 0 | 0 | 1 |  |  |
| 0 | 0 | 0 | 0 | 0 | 1 |  |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 |

Như vậy, ô v[i][j] sẽ được điền trị khi ta biết trị của ba ô bao quanh nó là

+ ô chéo trái v[i+1][j-1]

\* ô kề trái v[i][j-1]

# ô kề dưới v[i+1][j]

Từ đây ta suy ra quy trình điền mảng hai chiều v như sau:

Điền 1 tại đường chéo chính,

Điền 0 vào tam giác dưới đường chéo chính,

Điền các dòng i từ dưới lên trên: i = n-2..0.

Trên mỗi dòng, ta điền trị từ trái qua phải: j = i+1..n-1.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | b | a | e | a | d | b | a | d | b |
|  |  | *0* | *1* | *2* | *3* | *4* | *5* | *6* | *7* | *8* |
| b | *0* | 1 | 1 | 1 | 3 | 3 | 5 | 5 | 5 | 5 |
| a | *1* | 0 | 1 | 1 | 3 | 3 | 3 | 3 | 3 | 3 |
| e | *2* | 0 | 0 | 1 | 1 | 1 | 1 | 3 | 3 | 3 |
| a | *3* | 0 | 0 | 0 | 1 | 1 | 1 | 3 | 3 | 3 |
| d | *4* | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 3 | 3 |
| b | *5* | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 3 |
| a | *6* | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 |
| d | *7* | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| b | *8* | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| *Gía trị của hàm p(i,j) đối với dãy baeadbadb*  *i,j=0..8* | | | | | | | | | | |

Chương trình

# Palindrome. Phương án 2: dùng mảng 2D

from time import time

def Go(msg = ' ? '):

if input(msg) == '.': exit(0)

# số kí tự cần xóa

def Pal(s):

print(s)

n = len(s)

v = [[0]\*n for i in range(n)] # mang v toan 0

v[n-1][n-1] = 1

for i in range(n-2, -1, -1):

v[i][i] = 1

for j in range(i+1, n):

if s[i] == s[j]: v[i][j] = v[i+1][j-1]+2

else: v[i][j] = max(v[i][j-1],v[i+1][j])

return n-v[0][n-1]

def Run():

print(' Xóa ', Pal("abcvcbaabcvcba")) # xoa 0

print(' Xóa ', Pal("ab1cvcbaa2bcvc3ba")) # xoa 3

print(' Xóa ', Pal("ab1cvcba2abcvc3ba")) # xoa 2

# APPLICATION

t = time()

Run()

print(time()-t)

print(' T h e E n d')

Cải tiến 2: dùng hai mảng 1 chiều

Ta sẽ không theo đuổi phương án dùng mảng hai chiều mà hãy căn cứ vào quy luật điền mảng hai chiều để vận dụng cho hai mảng một chiều là *a và b*, trong đó *a* là mảng ứng với dòng dưới v[i+1][\*], *b* là mảng ứng với dòng trên cần điền trị v[i][\*]*,* i = n-2..0. Cụ thể là

a[j] = v[i+1][j] # dòng i+1

b[j] = v[i][j]) # dòng i

Sau mỗi dòng ta hoán vị a và b như hai con trỏ mảng.

Chương trình

# Palindrome. Phương án 3. mảng 1D

from time import time

def Go(msg = ' ? '):

if input(msg) == '.': exit(0)

# số kí tự cần xóa

def Pal(s):

print(s)

n = len(s)

if n == 1: return 0

a, b = [0]\*n, [0]\*n

a[n-1] = 1

for i in range(n-2, -1, -1):

# a -> b

b[i] = 1

for j in range(i+1,n):

if s[i] == s[j]: b[j] = a[j-1] + 2

else: b[j] = max(b[j-1],a[j])

a, b = b, a

return n - a[n-1]

def Run():

print(' Xóa ', Pal("abcvcbaabcvcba")) # xoa 0

print(' Xóa ', Pal("ab1cvcbaa2bcvc3ba")) # xoa 3

print(' Xóa ', Pal("ab1cvcba2abcvc3ba")) # xoa 2

# APPLICATION

t = time()

Run()

print(time()-t)

print(' T h e E n d')

Nếu muốn, bạn có thể dùng một mảng một chiều với hai biến đơn c và d ghi nhận lại giá trị của dòng chéo dưới (c) và dòng dưới (d).

Chương trình

# Palindrome. Phương án 3B

from time import time

def Go(msg = ' ? '):

if input(msg) == '.': exit(0)

# số kí tự cần xóa

def Pal(s):

print(s)

n = len(s)

if n == 1: return 0

a = [1]\*n

for i in range(n-2, -1, -1):

c = 0

for j in range(i+1,n):

d = a[j] # cất tạm phần tử j của dòng dưới

a[j] = c+2 if s[i]==s[j] else max(a[j-1],a[j])

c = d

return n - a[n-1]

def Run():

print(' Xóa ', Pal("abcvcbaabcvcba")) # xoa 0

print(' Xóa ', Pal("ab1cvcbaa2bcvc3ba")) # xoa 3

print(' Xóa ', Pal("ab1cvcba2abcvc3ba")) # xoa 2

# APPLICATION

t = time()

Run()

print(time()-t)

print(' T h e E n d')

## Flowers

Thuật toán

Trước hết ta thử giải bài này bằng Greedy Method. Ta sẽ lấy lần lượt các bó hoa b = 1..4 rồi xác định lọ h cần cắm bó hoa b nhằm đạt tổng độ thẩm mỹ max. Do phải cắm hết 4 bó hoa vào các lọ, nên với bó hoa b = 1 ta phai dành 3 lọ cuối cho 3 bó hoa còn lại, do đó ta chỉ có thể chọn các lọ h = 1..3 cho bó hoa thứ nhất b = 1. Trong ba lọ này ta chọn lọ có độ thẩm mỹ cao nhất để cắm bó hoa b = 1. Gọi t(b,h) là độ thẩm mỹ khi cắm bó hoa b vào lọ hoa h, ta có max{t(1,1), t(1,2), t(1,3)} = t(1,3), tức là ta cắm bó hoa 1 vào lọ 3 để đạt độ thẩm mỹ là 3. Lúc này chỉ còn lại 3 lọ hoa chưa dùng là 4, 5 và 6, do đó ta buộc phải cắm bó hoa 2 vào lọ 4, bó hoa 3 vào lọ 5 và bó hoa 4 vào lọ 6. Độ thẩm mỹ tổng cộng khi đó sẽ là:

T = t(1,3) + t(2,4) + t(3,5) + t(4,6) = 3 + 7 + 2 + 9 = 21 < 24

Lý do phương pháp tham không vận dụng được cho tình huống này là khi ta chọn tối ưu cục bộ cho một đối tượng sẽ làm ảnh hưởng đến các đối tượng còn lại.

Với số lọ là ít, ví dụ n < 10 thì bạn có thể vét cạn bằng tổ hợp chặp k của n phần tử, tuy nhiên bài này cho maxn = 100.

Ta dùng phương pháp quy hoạch động.

Lập hệ thức quy hoạch động

Gọi T(*i*, *j*) là tổng giá trị thẩm mỹ khi giải bài toán với *i* bó hoa mã số 1.*.i* và *j* lọ mã số 1..*j*, tức là độ thẩm mỹ thu được khi cắm hết *i* bó hoa đầu tiên vào *i* trong số *j* lọ đầu tiên, ta thấy:

* Nếu số bó hoa nhiều hơn số lọ, *i* > *j* thì không có cách cắm nào vì đầu bài yêu cầu phải cắm hết *i* bó hoa, mỗi bó vào đúng 1 lọ.

T(*i, j*) = 0, nếu *i > j*

* Nếu số bó hoa bằng số lọ (*i* = *j*) thì chỉ có một cách cắm là bó nào vào lọ đó.

T(i, j) = sum {v(k,k) | k = 1..i}, nếu i = j

* Ta xét trường hợp số bó hoa ít hơn hẳn số lọ (*i* < *j*). Có hai tình huống: lọ cuối cùng, tức lọ thứ *j* được chọn cho phương án tối ưu và lọ thứ *j* không được chọn.
* Nếu lọ cuối cùng, lọ thứ *j* được chọn để cắm bó hoa (cuối cùng) *i* thì *i* -1 bó hoa đầu tiên sẽ được phân phối vào *j*-1 lọ đầu tiên. Tổng giá trị thẩm mỹ khi đó sẽ là

T(i-1, j-1) + v[i][j].

* Nếu lọ thứ *j* không được chọn cho phương án tối ưu thì *i* bó hoa phải được cắm vào *j*-1 lọ đầu tiên và do đó tổng giá trị thẩm mỹ sẽ là T(i,j-1).

Tổng hợp lại ta có giá trị tối ưu khi cắm *i* bó hoa vào *j* lọ là:

T(i,j) = max {T(i-1,j-1)+v[i][j],T(i,j-1)}

Phương án dưới đây chỉ tìm giá trị thẩm mỹ tối ưu chứ chưa xác định được lọ cắm mỗi bó hoa.

Chương trình

# Flowers. PA 1: Đệ quy

from time import time

FN = "FLOWER.INP"

def Go(msg = ' ? '):

if input(msg) == '.': exit(0)

def ReadInput():

global k, n, v

with open(FN) as f:

k, n = map(int, f.readline().split())

# độ thẩm mĩ v[hoa][loj]

v = [[0]\*(n+1)] # dong 0

for hoa in range(1, k+1):

v.append([0]+list(map(int,f.readline().split())))

# hoa = lo: hoa nao vao lo do

def SumDiagonal(k):

return sum([v[i][i] for i in range(1, k+1)])

# Recursive

def T(hoa, lo):

if hoa == 0 or lo == 0 or hoa > lo: return 0

if hoa == lo: return SumDiagonal(hoa)

# hoa < lo

return max(T(hoa-1,lo-1)+v[hoa][lo], T(hoa,lo-1))

def Run():

ReadInput()

print(k, 'bó hoa,', n , 'lọ')

print(' Độ thẩm mĩ:')

for i in range(1, k+1): print(v[i][1:])

print(' Result: ', T(k,n))

# APPLICATION

t = time()

Run()

print(time()-t)

print(' T h e E n d')

Output

4 bó hoa, 6 lọ

Độ thẩm mĩ:

[1, 1, 3, 4, 3, 10]

[9, 1, 4, 7, 2, 7]

[7, 2, 6, 10, 2, 3]

[6, 10, 7, 1, 3, 9]

Result: 24

0.0009999275207519531

T h e E n d

Cải tiến1. Dùng mảng 2 chiều

Để khử đệ quy, ta dùng một mảng hai chiều tt để lưu các giá trị của hàm T(i,j):

tt[i][j] = T(i,j) # Cắm hết i bó hoa vào j lọ

Nếu số bó hoa i = 0 ta có dòng 0 chứa toàn 0,

Nếu số lọ hoa j = 0 ta có cột 0 chứa toàn 0,

Nếu số bó hoa = số lọ hoa , i = j, ta gán trị trên đường chéo chính

tt[j][j] = sum {v[j][j], j = 1..k (số bó hoa)}

Nếu số bó hoa > số lọ hoa: i > j, ta gán trị cho tam giác dưới toàn 0,

Ta xét trường hợp số bó hoa i < sô lọ j. Ta có

tt[i][j] = T(i,j) = max{tt[i-1][j-1] + v[i][j], tt[i][j-1]}

Tóm lại, muốn tính tt[i][j] cho trường hợp i > j ta phải biết trị của các ô tt[i-1][j-1] và tt[i][j-1]. Như vậy, các phần tử phía trên đường chéo chính sẽ được điền trị theo cột, từ trái qua phải. Trên mỗi cột ta điền các ô từ dưới lên trên.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| tt |  |  |  | j-1 | j |  |
|  |  |  |  |  |  |  |
| i-1 |  |  |  | \* |  |  |
| i |  |  |  | \* | ? |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |

Ta chia quy trình điền trị cho mảng hai chiều tt thành các pha sau đây cho dễ hiểu:

Pha 1. Khởi trị tt toàn 0

Pha 2. Điền trị cho đường chéo chính tt[i][i] = sum(v[j][j]), i = 1..k, j = 1..i. Giá trị tt[i][i] cho biết khi có i bó hoa và i lọ hoa thì ta cắm bó hoa j vào lọ j và thu được tổng độ thẩm mỹ sum(t[j][j]), j = 1..i.

Pha 3. Điền trị cho các cột tt[i][1..kj]. tt[i][j] là giá trị tối ưu khi giải bài toán con: cắm i bó hoa vào j lọ, i < j, khi số lọ hoa không lớn hơn số bó hoa.

Pha 4. Điền trị cho các cột tt[\*][k+1..n]. tt[i][j] là giá trị tối ưu khi giải bài toán con: cắm i bó hoa vào j lọ, i < j khi số lọ hoa n nhiều hơn hẳn số bó hoa k.

Chương trình chỉ tạm hiển thị độ thẩm mỹ tối ưu chứ chưa tổ chức ghi file.

Chương trình

# Flowers. PA 2: mảng 2D

from time import time

FN = "FLOWER.INP"

def Go(msg = ' ? '):

if input(msg) == '.': exit(0)

def ReadInput():

global k, n, v

with open(FN) as f:

k, n = map(int, f.readline().split())

# độ thẩm mĩ v[hoa][loj]

v = [[0]\*(n+1)] # dong 0

for hoa in range(1, k+1):

v.append([0]+list(map(int,f.readline().split())))

# cắm hết k bó hoa vào n lọ

def Flowers():

if k == 0 or n == 0 or k > n: return 0

# k <= n: Khởi trị tt

tt = [[0]\*(n+1) for i in range(k+1)]

for hoa in range(1,k+1):

tt[hoa][hoa] = tt[hoa-1][hoa-1]+v[hoa][hoa]

# Pha 3: k bo hoa, k lo

for lo in range(2, k+1):

for hoa in range(lo-1,0,-1):

tt[hoa][lo]=max(tt[hoa-1][lo-1]+v[hoa][lo],tt[hoa][lo-1])

# Pha 4:

for lo in range(k+1, n+1):

for hoa in range(k, 0, -1):

tt[hoa][lo]=max(tt[hoa-1][lo-1]+v[hoa][lo],tt[hoa][lo-1])

print('tt:')

for i in range(1, k+1): print(tt[i][1:])

return tt[k][n]

def Run():

ReadInput()

print(k, 'bó hoa,', n , 'lọ')

print(' Độ thẩm mĩ:')

for i in range(1, k+1):

print(v[i][1:])

c = Flowers()

print(' Result: ', c)

# APPLICATION

t = time()

Run()

print(time()-t)

print(' T h e E n d')

Output

4 bó hoa, 6 lọ

Độ thẩm mĩ:

[1, 1, 3, 4, 3, 10]

[9, 1, 4, 7, 2, 7]

[7, 2, 6, 10, 2, 3]

[6, 10, 7, 1, 3, 9]

tt:

[1, 1, 3, 4, 4, 10]

[0, 2, 5, 10, 10, 11]

[0, 0, 8, 15, 15, 15]

[0, 0, 0, 9, 18, 24]

Result: 24

T h e E n d

Cải tiến2. Dùng mảng 1 chiều

Sau khi biết trật tự điền trị cho mảng hai chiều, ta suy ra ngay là có thể chỉ dùng một mảng một chiều t và dịch chuyển dần theo từng cột. Ngoài ra, ta còn cần đặt trong mỗi ô của bảng một mảng dữ liệu gồm *n* phần tử để đánh dấu lọ hoa nào được chọn cho mỗi tình huống. Gọi mảng dữ liệu đó là b, ta dễ thấy là nên điền bảng lần lượt theo từng cột, tại mỗi cột ta điền bảng từ dưới lên theo hai pha như sau:

Pha 1. Cập nhật mảng t và b với số lọ hoa j = 1..k (k là số bó hoa)

Tại pha này với mỗi số lọ hoa j ta cập nhật t và b theo hai bước:

Bước 1. tính trị t[j] ứng với hàm T(j,j) cắm bó hoa nào vào lọ ấy.

Bước 2. tính các trị t[i] và b[i] ứng với hàm

T(i,j) = max{T(i-1,j-1)+v[i][j], T(i,j-1)}, i = j-1🡨1

Pha 2. Cập nhật mảng t và b với số lọ hoa j = k+1..n (n là số lọ hoa)

Tại pha này ta chỉ cần thực hiện bước 2:

Bước 2. tính các trị t[i] và b[i] ứng với hàm

T(i,j) = max{T(i-1,j-1)+v[i][j], T(i,j-1)}, i = k🡨1

* Nếu t[*i*-1] + v[*i*][ *j*] > t[i] thì ta phải thực hiện hai thao tác:
  + Đặt lại trị t[i] = t[*i*-1] + v[*i*][ *j*]
  + Ghi nhận việc chọn lọ hoa *j* trong phương án mới, cụ thể lấy phương án cắm hoa (*i*-1, *j*-1) rồi bổ sung thêm thông tin chọn lọ hoa j như sau: đặt *b*[i] = *b*[*i*-1] và đánh dấu phần tử *j* trong mảng *b*[*i*][*j*]: *b*[*i*][*j*] = 1.

Chương trình

# Flowers. PA 3: mảng 1D

from time import time

FN = "FLOWER.INP"

GN = "FLOWER.OUT"

def Go(msg = ' ? '):

if input(msg) == '.': exit(0)

def ReadInput():

global k, n, v

with open(FN) as f:

k, n = map(int, f.readline().split())

# độ thẩm mĩ v[hoa][loj]

v = [[0]\*(n+1)] # dong 0

for hoa in range(1, k+1):

v.append([0]+list(map(int,f.readline().split())))

# cắm hết k bó hoa vào n lọ

def Flowers():

global b, t

if k == 0 or n == 0 or k > n: return 0

# k <= n: Khoi tri cot t

t = [0]\*(k+1)

# khoi tri cac mang b toan 0

b = [bytearray([0]\*(n+1)) for i in range(k+1)]

"""

k <= n

t[i] tai buoc j la tri toi uu

khi giai bai cam het i bo hoa vao j lo

"""

for j in range(1,k+1): # j lo

# Xet truong hop j hoa, j lo

t[j] = t[j-1] + v[j][j]

b[j][j] = 1

# duyet cot voi so bo hoa i tu duoi len

# so hoa i < so lo j

for i in range(j-1, 0, -1):

tt = t[i-1] + v[i][j]

if tt > t[i]:

t[i] = tt

b[i] = b[i-1].copy()

b[i][j] = 1

# Pha 4

for j in range(k+1, n+1):

# duyet cot voi so bo hoa i tu tren xuong

# so hoa i < so lo j

for i in range(k, 0, -1):

tt = t[i-1] + v[i][j]

if tt > t[i]:

t[i] = tt

b[i] = b[i-1].copy()

b[i][j] = 1

return t[k]

def WriteResult():

with open(GN, 'w') as g:

g.write(str(t[k])+'\n')

bh = 0

s = ''

for j in range(1, n+1):

if b[k][j]:

bh += 1

print(' Cắm bó hoa',bh,' vào lọ',j)

s += str(j) + ' '

g.write(s+'\n')

def Run():

ReadInput()

print(k, 'bó hoa,', n , 'lọ')

print(' Độ thẩm mĩ:')

for i in range(1, k+1):

print(v[i][1:])

c = Flowers()

print(' Result: ', c)

WriteResult()

# APPLICATION

tg = time()

Run()

print(time()-tg)

print(' T h e E n d')

Output

4 bó hoa, 6 lọ

Độ thẩm mĩ:

[1, 1, 3, 4, 3, 10]

[9, 1, 4, 7, 2, 7]

[7, 2, 6, 10, 2, 3]

[6, 10, 7, 1, 3, 9]

Result: 24

Cắm bó hoa 1 vào lọ 1

Cắm bó hoa 2 vào lọ 3

Cắm bó hoa 3 vào lọ 4

Cắm bó hoa 4 vào lọ 6

0.002994537353515625

T h e E n d

Output file FLOWER.OUT

24

1 3 4 6

Độ phức tạp

Điền mảng tt kích thứơc k×n đòi hỏi *kn* thao tác, trong đó *k* là số bó hoa, n là số lọ hoa, *kn* là chiều dài của dữ liệu vào. Như vậy thuật toán quy hoạch động có độ phức tạp tuyến tính theo chiều dài dữ liệu vào.

## Bố trí phòng học

Tương tự bài Flowers.

## Bống

Tương tự bài Flowers.

## Tìm các đường ngắn nhất

Thuật toán Dijkstra

Thuật giải quy hoạch động được trình bày dưới đây mang tên E. Dijkstra, nhà tin học lỗi lạc người Hà Lan. Bản chất của thuật toán là *sửa đỉnh*, chính xác là *sửa trọng số* của mỗi đỉnh.

|  |
| --- |
| A person with a beard and glasses  Description automatically generated with low confidence |
| Edsger Wybe Dijkstra  1930-2002 |

Theo sơ đồ giải các bài toán quy hoạch động, trước hết ta xây dựng *hệ thức quy hoạch động* (*phương trình Bellman*) cho bài toán.

+Hệ thức quy hoạch động

Gọi *p*(*i*) là độ dài đường ngắn nhất từ đỉnh *s* đến đỉnh *i*, 1 ≤ *i* ≤ *n*. Ta thấy, hàm *p*(*i*) phải thoả các tính chất sau:

*a*) *p*(*s*) = 0: đường ngắn nhất từ đỉnh xuất phát *s* đến chính đỉnh đó có chiều dài 0.

*b*) Với *i* ≠ *s*, muốn đến được đỉnh *i* ta phải đến được một trong các đỉnh sát trước đỉnh *i*. Nếu *j* là một đỉnh sát trước đỉnh *i*, theo điều kiện của đầu bài ta phải có *c(j,i) > 0,* trong đó *c*(*j*, *i*) chính là chiều dài cung (*j* → *i*). Khi đó đường từ s đến i sẽ được chia làm hai đoạn: đường từ s đến đỉnh j là đỉnh sát trước đỉnh i và cung (j → i)*: p(i) = p(j) + c(j,i).*

Do *p*(*i*) và *p*(*j*) phải là ngắn nhất, ta suy ra điều kiện để chọn đỉnh *j* sát trước đỉnh *i* là tổng chiều dài đường từ *s* đến *j* và chiều dài cung (*j* → *i*) là ngắn nhất. Ta thu được hệ thức sau:

p(i) = min {p(j)+c(j,i) | c(j,i) > 0, j = 1..n }

Nhắc lại rằng điều kiện *c*(*j*, *i*) > 0 cho biết *j* là đỉnh sát trước đỉnh *i*.

Điều tài tình là Dijkstra đã cung cấp thuật toán tính đồng thời mọi đường đi ngắn nhất từ đỉnh s đến các đỉnh còn lại của đồ thị. Thuật toán đó như sau.

1. Khởi trị. Gán trọng số vô cùng lớn cho mọi đỉnh, riêng đỉnh xuất phát s được gán trọng  
 số 0:

p[i] = ∞; i = 1..n; p[s] = 0;

2. Lặp n-1 lần

2.1 Lấy đỉnh i có trọng số min trong số các đỉnh chưa xử lý:

p[i] = min {p[j]: j = 1..n; j chưa xử lý}

2.2 Đánh dấu đỉnh i là đã đã xử lý

i = đã xử lý

2.3 Cập nhật trọng số p[j] của các đỉnh j chưa xử lý và kề với đỉnh i:

Nếu đỉnh j chưa xử lý và c(i,j) > 0 và p[j] > p[i]+c(i,j)

thì đặt lại p[j] = p[i]+c(i,j)

Ý nghĩa của bước này là: nếu có đường i → j ngắn hơn thì đi theo khúc i → j.

Tổ chức dữ liệu

Mảng hai chiều c[i][j] cho biết độ dài của đoạn đường (cung) từ đỉnh i đến đỉnh kề j

Mảng trọng số đỉnh p[i]

Giá trị ∞ được chọn trong chương trình là:

vocung = tổng độ dài các cung trong đồ thị + 10

Mảng rest đánh dấu đỉnh còn lại (chưa xử lý)

Mảng pred[i] = j cho biết đoạn đường cần đi là i → j.

Thuật toán chứa hai vòng for lồng nhau, do đó có độ phức tạp là .

Sau khi hoàn thành thuật toán Dijkstra ta cần gọi thủ tục Path để ghi lại kết quả theo yêu cầu của đầu bài như sau.

Với mỗi đỉnh *i* = 1..*n* ta cần ghi vào output file chiều dài đường đi từ *s* đến *i* bao gồm giá trị *p*[*i*] và các đỉnh nằm trên đường đó.

Chú ý rằng khi thuật toán kết thúc, nếu *p*[*i*] vẫn nhận giá trị khởi đầu là vocung (∞) thì tức là không có đường đi từ *s* đến *i*.

Về ý nghĩa, mảng pred chứa các con trỏ ngược từ mỗi đỉnh *i* đến đỉnh sát trước đỉnh *i* trên đường đi ngắn nhất, do đó ta phải lần ngược bằng thủ tục đệ quy path(i) để ghi vào output file *g* vết của đường đi theo trật tự từ *s* đến *i*.

|  |
| --- |
| Thuật toán Dijkstra |
| Input: đồ thị G; đỉnh xuất phát s  Output: mọi đường ngắn nhất từ s  đến các đỉnh của G |
| Begin  Gán các đỉnh của G trọng số p[i] = ∞  p[s] = 0 # đỉnh xuất phát s có trọng số 0  lặp n-1 lần với các đỉnh chưa xử lý  Tìm đỉnh i có trọng số min  Đánh dấu đỉnh i : đã xử lý  Xét các đỉnh j kề với đỉnh i  if p[i] + c[i][j] < p[j] then  p[j] = p[i] + c[i][j] # cập nhật j  pred[j] = i;  end if  kết thúc lặp  Giải trình kết quả  End Dijkstra |

Ta minh hoạ tiến trình hoạt động của thuật toán Dijkstra qua ví dụ đã cho.

Sau khi đọc dữ liệu từ file f DIJ.INP ta có *n* = 7, *s* = 2, vocung = 33 (∞) . Đồ thị có 7 đỉnh, đỉnh xuất phát là 2. Ma trận kề c thu được như sau:

Diagram

Description automatically generated

Khởi trị

p[i] = ∞, 1 ≤ i ≤ 7, p[2] = 0

Thực hiện n-1 = 6 bước lặp.

Bước lặp k = 1

Table

Description automatically generated

*i =* min *=* 2với *p*[2] = 0*.*

Các đỉnh chưa xử lý và kề với đỉnh 2 sẽ được sửa trọng số là 1, 3, 7 (có dấu 🗸).

Vì p[2] + c[2][1] = 0 + 4 = 4 < p[1] = ∞ nên p[1] được sửa thành 4 và pred[1] được sửa thành 2.

Vì p[2] + c[2][3] = 0 + 1 = 1 < p[3] = ∞ nên p[3] được sửa thành 1 và pred[3] được sửa thành 2.

Vì p[2] + c[2][7] = 0 + 5 = 5 < p[7] = ∞ nên p[7] được sửa thành 5 và pred[7] được sửa thành 2.

Bước lặp k = 2

*i* = min = 3 với *p*[3] = 1.

Đỉnh chưa xử lý và kề với đỉnh 3 sẽ được sửa trọng số là đỉnh 7.

Vì *p*[3] + *c*[3][7] = 1 + 1 = 2 < *p*[7] = 5 nên *p*[7] được sửa thành 2 và pred[7] được sửa thành 3.

Table

Description automatically generated

Bước lặp k = 3

*i* = min = 7 với *p*[7] = 2

Đỉnh chưa xử lý và kề với đỉnh 7 sẽ được sửa trọng số là đỉnh 4.

Vì *p*[7] + *c*[7][4] = 2 + 1 = 3 < *p*[4] = ∞ nên *p*[4] được sửa thành 3 và pred[4] được sửa thành 7.

Table

Description automatically generated

Bước lặp k = 4

*i* = min = 4 với *p*[4] = 3.

Đỉnh chưa xử lý và kề với đỉnh 4 sẽ được sửa trọng số là đỉnh 6.

Vì *p*[4] + *c*[4][6] = 3 + 2 = 5 < *p*[6] = ∞ nên *p*[6] được sửa thành 5 và pred[6] được sửa thành 4.

Table

Description automatically generated

Bước lặp k = 5

*i* = min = 1 với *p*[1] = 4.

Không có đỉnh chưa xử lý nào kề với đỉnh 1.

Table

Description automatically generated

Bước lặp k = 6

*i* = min = 6 với *p*[6] = 5.

Không có đỉnh chưa xử lý nào kề với đỉnh 6. Chú ý rằng đỉnh 1 và đỉnh 7 kề với đỉnh 6 nhưng hai đỉnh này đã xử lý rồi.

Table

Description automatically generated

Thuật toán dừng sau 6 bước lặp.

Lưu ý rằng đỉnh xuất phát cho bài toán này là *s* = 2. Ta minh hoạ giải trình kết quả cho ba ví dụ sau.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Đỉnh |  | p | dinhTruoc | Ba ví dụ |
| 1 |  | 4 | 2 | *Giải trình đương ngắn nhất* 2 → 4? |
| 2 |  | 0 | 0 | *Giải trình đương ngắn nhất* 2 → 5 ? |
| 3 |  | 1 | 2 | *Giải trình đương ngắn nhất* 2 → 2 ? |
| 4 |  | 3 | 7 |  |
| 5 |  | ∞ | 0 |  |
| 6 |  | 5 | 4 |  |
| 7 |  | 2 | 3 |  |

Đường đi ngắn nhất từ đỉnh s = 2 đến đỉnh t = 4 ?

Vì p[4] = 3 nên độ dài đường đi là 3.

Để giải trình vết của đường đi từ 2 đến 4 ta dựa vào mảng pred[1..7] như sau:

Vì pred[4] = 7, tức là trước khi đến đỉnh 4 phải qua đỉnh 7 nên ta có

7 → 4

Vì pred[7] = 3, tức là trước khi đến đỉnh 7 phải qua đỉnh 3 nên ta có

3 → 7 → 4

Vì dinh pred[3] = 2, tức là trước khi đến đỉnh 3 phải qua đỉnh 2 nên ta có

2 → 3 → 7 → 4

Kết quả này được ghi ở dòng thứ tư của file DIJ.OUT như sau:

3 2 3 7 4

trong đó số đầu tiên 3 cho biết chiều dài đường đi, dãy số còn lại giải trình vết của đường đi từ đỉnh 2 đến đỉnh 4.

Đường đi ngắn nhất từ đỉnh s = 2 đến đỉnh t = 5 ?

Vì *p*[5] = ∞ ứng với giá trị dương vô cùng ∞ khởi trị lúc đầu nên không có đường đi từ đỉnh 2 đến đỉnh 5.

Ta ghi kết quả 0 tại dòng 5 của file DIJ.OUT.

Đường đi ngắn nhất từ đỉnh s = 2 đến đỉnh t = 2 ?

Vì *s* = *t* nên ta coi như không có đường đi từ đỉnh 2 đến đỉnh 2.

Ta ghi kết quả 0 tại dòng 5 của file DIJ.OUT.

Chương trình

# Dijkstra

from time import time

FN = "DIJ.INP"

GN = "DIJ.OUT"

def Go(msg = ' ? '):

if input(msg) == '.': exit(0)

def ReadInput():

global n, s, vocung, c

with open(FN) as f:

n, s = map(int, f.readline().split())

print(n, 'đỉnh. Đỉnh xuất phát:',s)

vocung = 10

c = [[0]\*(n+1)] # dòng 0 của ma trận kề c

for i in range(1, n+1):

d = [0]+list(map(int, f.readline().split()))

vocung += sum(d)

print(d)

c.append(d)

print(' vô cùng =', vocung)

def Init():

global p, rest, pred

# gan trong so cac dinh la vo cung

p = [vocung]\*(n+1)

p[s] = 0

# danh dau cac dinh con lai

rest = bytearray([1]\*(n+1))

pred = [0]\*(n+1) # cac dinh sat truoc

# Dinh trong so min trong cac dinh chua tham

def Min():

imin = 0

for i in range(1, n+1):

if rest[i] and p[i] < p[imin]: imin = i

return imin

def Dijkstra():

global rest, p

for k in range(1,n): # lap n-1 lan

i = Min() # tim dinh trong so min trong cac dinh chua tham

print(' imin = ', i)

rest[i] = 0 # da tham dinh i

# sua lai trong so cac dinh chua tham va ke dinh i

for j in range(1,n+1):

if rest[j] == 0 or c[i][j] == 0: continue

# co duong tu i -> j va dinh j chua xu ly

if p[i]+c[i][j] < p[j]:

p[j] = p[i]+c[i][j]

pred[j] = i

def WriteResult():

global s

with open(GN, 'w') as g:

for i in range(1, n+1):

if p[i] == vocung: p[i] = 0

s = str(p[i]) + ' '

if p[i] > 0: Path(i)

g.write(s + '\n')

def Path(i):

global s

if i == 0: return

Path(pred[i])

s += str(i) + ' '

def Run():

ReadInput()

Init()

Dijkstra()

WriteResult()

# APPLICATION

t = time()

Run()

print(time()-t)

print(' T h e E n d')

Output

7 đỉnh. Đỉnh xuất phát: 2

[0, 0, 0, 0, 0, 0, 0, 0]

[0, 4, 0, 1, 0, 0, 0, 5]

[0, 0, 0, 0, 0, 0, 0, 1]

[0, 0, 0, 0, 0, 0, 2, 0]

[0, 0, 0, 0, 3, 0, 0, 0]

[0, 1, 0, 0, 0, 0, 0, 5]

[0, 0, 0, 0, 1, 0, 0, 0]

vô cùng = 33

imin = 2

imin = 3

imin = 7

imin = 4

imin = 1

imin = 6

0.001998424530029297

T h e E n d

Output file DiJ.OUT

4 2 1

0

1 2 3

3 2 3 7 4

0

5 2 3 7 4 6

2 2 3 7

Các dạng khác của bài toán Dijkstra

Lưu ý rằng ma trận kề có thể chứa các giá trị thực, tuy nhiên cần giả thiết rằng mọi giá trị trong ma trận kề phải là các số không âm. Với các số âm bài toán sẽ phức tạp hơn.

Dưới đây liệt kê hai dạng khá thông dụng P1, và P2 của bài toán tìm đường đi trong đồ thị.

* P1. Nếu đồ thị đã cho là vô hướng ta giải như trên, chỉ lưu ý đến tính đối xứng khi đọc dữ liệu vào ma trận kề a.
* P2. Nếu đề bài chỉ yêu cầu tìm một đường đi từ đỉnh s đến đỉnh t ta thực hiện các bước sau:
* Đọc dữ liệu.
* Gọi thuật toán Dijkstra.
* Ghi kết quả p[t] và giải trình một đường theo thuật toán path(t).