# Các nội dung tô màu phân tài liệu tham khảo là đã copy và đưa vào tai liệu chính thức,

# Code C++ cùng thư mục này là code trong tài liệu này

# Thuật toán Quy hoạch động

[Khi nào thì dùng thuật toán quy hoạch động](https://topdev.vn/blog/thuat-toan-quy-hoach-dong/" \l "_khi-nao-thi-dung-quy-hoach-dong-0)?

* 1. [Bài toán con gối nhau](https://topdev.vn/blog/thuat-toan-quy-hoach-dong/" \l "_bai-toan-con-goi-nhau-1)
  2. [Cấu trúc con tối ưu](https://topdev.vn/blog/thuat-toan-quy-hoach-dong/" \l "_cau-truc-con-toi-uu-2)

-------------------------------------------------

Gần một nửa các bài thi trong các cuộc thi code cần đến quy hoạch động. Tất nhiên, có những cách khác để giải bài toán đó. Nhưng vì các cuộc thi code đều có giới hạn về thời gian, cũng như bộ nhớ của chương trình, nên một thuật toán hiệu quả là cực kỳ cần thiết. Và trong những trường hợp như vậy, quy hoạch động là một trong những thuật toán xuất hiện nhiều nhất.

Thuật toán quy hoạch động được ưa chuộng bởi vì ban đầu, bài toán có muôn hình vạn trạng và bạn phải suy nghĩ rất nhiều mới tìm ra được lời giải. Không có một công thức chuẩn mực nào áp dụng được cho mọi bài toán. Bởi vì sự phổ biến của nó, bạn bắt buộc phải cực kỳ thuần thục thuật toán này nếu muốn có kết quả tốt trong các cuộc thi.

Cách hiệu quả nhất để tìm hiểu một thuật toán là xem xét những ví dụ cụ thể. Trong bài viết này, tôi sẽ giới thiệu vài ví dụ trong phần sau. Có thể nó chưa đầy đủ, bạn có thể đọc thêm ở các bài viết khác. Giới thiệu với các bạn một tài liệu rất hay: Dynamic Programming: From novice to advanced

**I. KHI NÀO THÌ DÙNG THUẬT TOÁN QUY HOẠCH ĐỘNG**

Khi nào thì chúng ta cần đến quy hoạch động? Đó là một câu hỏi rất khó trả lời. Không có một công thức nào cho các bài toán như vậy.

Tuy nhiên, có một số tính chất của bài toán mà bạn có thể nghĩ đến quy hoạch động. Dưới đây là hai tính chất nổi bật nhất trong số chúng:

* Bài toán có các bài toán con gối nhau.
* Bài toán có cấu trúc con tối ưu.

Thường thì một bài toán có đủ cả hai tính chất này, chúng ta có thể dùng quy hoạch động được. Một câu hỏi rất thú vị là không dùng quy hoạch động có được không? Câu trả lời là có, nhưng nếu bạn đi thi code, bạn trượt là cái chắc. Để hiểu rõ hơn, chúng ta sẽ tìm hiểu từng tính chất một trong những phần dưới đây

**1/ Bài toán con gối nhau**

Tương tự như thuật toán chia để trị, quy hoạch động cũng chia bài toán lớn thành các bài toán con nhỏ hơn. Quy hoạch động được sử dụng khi các bài toán con này được gọi đi gọi lại. Phương pháp quy hoạch động sẽ lưu kết quả của bài toán con này, và khi được gọi, nó sẽ không cần phải tính lại, do đó làm giảm thời gian tính toán.

Quy hoạch động sẽ không thể áp dụng được (hoặc nói đúng hơn là áp dụng cũng không có tác dụng gì) khi các bài toán con không gối nhau. Ví dụ với thuật toán tìm kiếm nhị phân, quy hoạch động cũng không thể tối ưu được gì cả, bởi vì mỗi khi chia nhỏ bài toán lớn thành các bài toán con, mỗi bài toán cũng chỉ cần giải một lần mà không bao giờ được gọi lại.

Một ví dụ rất điển hình của bài toán con gối nhau là bài toán tính số [**Fibonacci**](https://topdev.vn/blog/thuat-toan-chuoi-fibonacci-△/). Bài toán quá nổi tiếng rồi, chúng ta có thể tính toán số Fibonacci theo đúng công thức như sau:

int fib(n){

if (n <= 1) return 1;

else return fib(n -1) + fib(n - 2)

Nếu tính toán như trên, chúng ta rất nhiều bài toán con sẽ được tính đi tính lại, điển hình là các số fib(0) và fib(1).

Và quy hoạch động chính là một trong số những phương pháp có thể giúp chúng ta tối ưu hóa quá trình tính toán này. Mỗi bài toán con (số fib) sẽ được lưu lại trước khi tính những bài toán con lớn hơn. Nhờ đó, mà việc tính toán giảm đi đáng kể, mỗi bài toán con chỉ cần tính **đúng một lần**.

Giải pháp quy hoạch động với bài toán này như sau:

int fib(n){

int dp[100];

dp[0]=0; dp[1] = 1;

for (int i=2;i<=n;i++) dp[i] = dp[i - 1] + dp[i - 2]

return dp[n]

**Ví dụ 2:** Tính N!

Đệ qui: int gt(int n){

if (n= =0) return 1;

else return gt(n-1)\*n;

}

Việc tính các giai thừa được thực hiện lặp lại nhiều lần

Giải pháp khác: Xử lí các bài toán con và Lưu các giá trị vào mảng  việc tính các giải thừa nhỏ hơn chỉ thực hiện một lần.

1. Tính nghiệm của bài toán trong trường hợp riêng đơn giản nhất.

Gt[0]=1;

1. Tìm các công thức đệ quy biểu diễn nghiệm tối ưu của bài toán lớn thông qua nghiệm tối ưu của các bài toán con.

for (int i=1;i<=n;i++) Gt[i] = Gt[i - 1]\*i;

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | 0! | n>=1 (n!=(n-1)!\*n | N=2 | N=3 | N=4 | N=5 |
| F[] | 1 | 1 | 2 | 6 | 24 | 120 |

**Ví dụ 3:** Tính các phần tử của  số tổ hợp chập k của n phần tử, với với 0≤k ≤n ≤20.

Biết 

Giải thuật đệ qui:

int Tohop(n,k){

if (k ==0 or k==n) return 1;

else return Tohop(n -1,k-1) + Tohop(n – 1, k);

trường hợp n=5; k=3 hàm C(n,k)=C(5,3) thực hiện như mô hình sau :

**C**(5,3))

**C**(4,2)

**C**(4,3)

**C**(3,1)

**C**(3,2)

**C**(3,2))

**C**(3,3)

**C**(2,0)

**C**(2,1)

**C**(2,1)

**C**(2,2)

**C**(2,1)

**C**(2,2)

**C**(1,0)

**C**(1,1)

**C**(1,0)

**C**(1,1)

**C**(1,0)

**C**(1,1)

**C**(n,k)

**C**(n-1,k-1)

**C**(n-1,k)

Nhìn vào sơ đồ trên ta thấy : để tính C(5,3) ta phải tính:

* C(1,0) : 3 lần
* C(1,1) : 3 lần
* C(2,1) : 3 lần
* C(2,2) : 2 lần
* C(3,2) : 2 lần
* C(1,0) : 3 lần

……………..

Giải pháp Qui hoạch động

1) Tính nghiệm của bài toán trong trường hợp riêng đơn giản nhất.

for (int i=1;i<=n;i++) {

C[0, i] := 1;

C[i, i] := 1;

}

2) Tìm các công thức đệ quy biểu diễn nghiệm tối ưu của bài toán lớn thông qua nghiệm tối ưu của các bài toán con.

for (int i=2;i<=n;i++)

for (int j=1;i<=i-1;j++) C[i, j] := C[i-1,j-1] + C[i-1,j];

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| K  N | 0 | 1 | 2 | 3 | 4 | 5 |
| 1 | 1 | 1 |  |  |  |  |
| 2 | 1 | 2 | 1 |  |  |  |
| 3 | 1 | 3 | 3 | 1 |  |  |
| 4 | 1 | 4 | 6 | 4 | 1 |  |
| 5 | 1 | 5 | 10 | 10 | 5 | 1 |

3) Có thể cải tiến: dùng 2 mảng một chiều thay cho 1 mảng hai chiều.

Qua ví dụ trên, bạn đã thấy được sức mạnh vượt trội của quy hoạch động chưa? Đó cũng chính là lý do mà nó rất được ưa chuộng trong các cuộc thi lập trình, khi mà thời gian và bộ nhớ đều là hữu hạn (và thường khá nhỏ).

**2/ Cấu trúc con tối ưu**

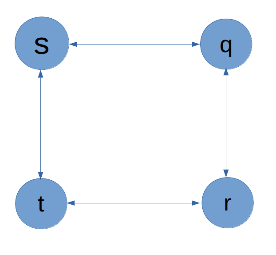
Cấu trúc con tối ưu là một tính chất là lời giải của bài toán lớn sẽ là tập hợp lời giải từ các bài toán nhỏ hơn.

Mình lấy một ví dụ cho dễ hiểu:

Trong bài toán tìm đường đi ngắn nhất trong đồ thị, nếu một node x nằm trên đường đi ngắn nhất giữa hai node u, v thì đường đi ngắn nhất từ u đến v sẽ là tổng hợp của đường đi ngắn nhất từ u đến x và đường đi ngắn nhất từ x đến v. Môt số thuật toán tìm đường trên đồ thị (nổi tiếng nhất có lẽ là [Dijkstra](https://en.wikipedia.org/wiki/Dijkstra's_algorithm)) đều dựa trên tính chất này, và nó cũng áp dụng quy hoạch động.

Tính chất cấu trúc con tối ưu rất quan trọng. Nó cho phép chúng ta giải bài toán lớn dựa vào các bài toán con đã giải được. Nếu không có tính chất này, chúng ta không thể áp dụng quy hoạch động được.

Không phải bài toán nào cũng có tính chất cấu trúc con tối ưu này. Ví dụ với đồ thị sau:



Đường đi **dài nhất** từ q -> t sẽ là q -> r -> t hoặc q -> s -> t. Nhưng không giống như bài toán tìm đường đi ngắn nhất, đường đi dài nhất không phải là tổ hợp của những đường đi thành phần, do đó, bài toán này không có cấu trúc con tối ưu.

Ví dụ, đường q -> r -> t không phải là tổ hợp của đường đi dài nhất từ q -> r và đường đi dài nhất từ r -> t. Bởi vì, đường đi dài nhất q -> rphải là q -> s -> t -> r và đường đi dài nhất từ r -> t phải là r -> q -> s -> t.

**II. CÁC BƯỚC GIẢI BÀI TOÁN QUY HOẠCH ĐỘNG**

1) Tính nghiệm tối ưu của bài toán trong trường hợp riêng đơn giản nhất.

2) Tìm các công thức đệ quy (truy hồi) biểu diễn nghiệm tối ưu của bài toán lớn thông qua nghiệm tối ưu của các bài toán con.

3) Tính nghiệm tối ưu từ dưới lên (bottom up) và ghi lại các nghiệm tối ưu của các bài toán con đã tính để sử dụng sau này.

**III. ƯU NHƯỢC ĐIỂM, PHẠM VỊ ÁP DỤNG**

**1/ Ưu điểm của phương pháp quy hoạch động:**

Chương trình chạy nhanh, thường chỉ cho kết quả là 1 giá trị tối ưu.

**2/ Hạn chế của phương pháp quy hoạch động:**

Phương pháp quy hoạch động không đem lại hiệu quả trong các trường hợp sau:

Sự kết hợp lời giải của các bài toán con chưa chắc cho ta lời giải của bài toán lớn.

Số lượng các bài toán con cần giải quyết và lưu trữ kết quả có thể rất lớn, không thể chấp nhận được.

Không tìm được công thức truy hồi.

**3/ Phạm vi áp dụng của phương pháp quy hoạch động:**

- Các bài toán tối ưu: như tìm xâu con chung dài nhất, bài toán balô(cái túi), tìm đường đi ngắn nhất, bài toán biến đổi ít nhất, …

- Các bài toán có công thức truy hồi.

**IV. MỘT SỐ BÀI TOÁN QUY HOẠCH ĐỘNG**

Trong phần này, chúng ta sẽ làm quen với quy hoạch động thông qua một số ví dụ cụ thể. Chúng ta sẽ xem xét cách quy hoạch động được áp dụng vào các bài toán cụ thể như thế nào, đồng thời qua đó, chúng ta sẽ hiểu hơn về các tính chất ở phần trước.

**1/ Ví dụ 1: Tìm dãy con không giảm dài nhất:**

**Đề bài:** Cho một dãy n số nguyên. Hãy loại bỏ khỏi dãy một số phần tử để được một dãy con thu được là dãy không giảm dài nhất. In ra dãy con đó.

*Hay phát biểu 1 cách khác cho dãy có N phần tử, tìm dãy con không giảm dài nhất.*

Input: N và dãy A

Output: Độ dài và dãy con tìm được

|  |  |
| --- | --- |
| INPUT | OUTPUT |
| 10  2 6 -7 5 8 1 -3 5 15 9 | 4  -7 -3 5 9 |

**1. Đặc trưng**:

Các phần tử trong dãy kết quả chỉ xuất hiện 1 lần. Vì vậy phương pháp làm là ta sẽ dùng vòng For duyệt qua các phần tử A trong dãy, khác với các bài toán của mô hình 4 (đặc trưng là bài toán đổi tiền các phần tử trong dãy có thể được chọn nhiều lần nên ta thực hiện bằng phương pháp cho giá trị cần quy đổi tăng dần từng đơn vị).

Thứ tự của các phần tử được chọn phải được giữ nguyên so với dãy ban đầu.

Đặc trưng này có thể mất đi trong một số bài toán khác tùy vào yêu cầu cụ thể. Chẳng hạn bài: Tam giác bao nhau.

**2. Công thức QHĐ**

Ta nhận thấy số lượng phần tử dãy a càng tăng thì độ dài dãy con tăng có khả năng tăng theo. Độ dài L của dãy con tăng biến thiên theo 1 đại lượng  ta sẽ dùng bảng phương án là mảng một chiều. Gọi **L(i)** là độ dài dãy con tăng dài nhất của đoạn phần tử từ a1 đến ai.

Nhận xét với cách làm này ta đã chia 1 bài toán lớn (dãy con của n số) thành các bài toán con cùng kiểu có kích thước nhỏ hơn (dãy con của dãy i số). Vấn đề là công thức truy hồi để phối hợp kết quả của các bài toán con.

Ta có công thức QHĐ để tính L(i) như sau:

L(1) = 1 (Hiển nhiên)

L(i) = 1 + Max(L[j]) với j=1..i-1 và aj<ai.

Tính L(i) : phần tử đang được xét là ai . Ta tìm đến phần tử aj < ai có L(j) lớn nhất. Khi đó nếu bổ sung ai vào sau dãy con ...aj ta sẽ được dãy con tăng dần dài nhất xét từ a1...ai.

Đáp án: Giá trị lớn nhất của mảng L.

Duyệt với i = 1,n

Duyệt với j = 1, i-1

Tìm L[j] lớn nhất (j=1,i-1) với a[i]>a[j] và j gần i nhất

* L[i] = L[j]+1;

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| i | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| A[i] | 2 | 6 | -7 | 5 | 8 | 1 | -3 | 5 | 15 | 9 |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| i | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| L[i] | 1 | 2 | 1 | 2 | 3 | 2 | 2 | 3 | 4 | 4 |
|  |  |  |  |  |  |  |  |  |  |  |
| Luu[i] | 0 | 1 | 0 | 3 | 4 | 3 | 3 | 7 | 8 | 8 |
| Kết quả A[k] |  |  | -7 |  |  |  | -3 | 5 |  | 9 |

Dãy con không giảm dài nhất có 4 phần tử: -7 -3 5 9

**3. Cài đặt**

Bảng phương án là một mảng một chiều L để lưu trữ các giá trị của hàm QHĐ L(i). Đoạn chương trình tính các giá trị của mảng L như sau:

Trong ví dụ trên:

*void PA(){*

*//khoi tao*

*for (int i=1; i<=n;i++) L[i]=1;*

*for (int i=0; i<=n;i++) luu[i]=0;*

*//Tao bang PA*

*int i,j=0;*

*for (i=1; i<=n;i++) {*

*int cs=i-1;*

*for(j=2;j<=i-1;j++)*

*if (a[j]<=a[i] and L[cs]<=L[j]) cs=j;*

*L[i]=L[cs]+1;*

*}}*

*Cải tiến để thuật toán chạy nhanh hơn*

*//khoi tao*

*for (int i=1; i<=n;i++) L[i]=1;*

*for (int i=0; i<=n;i++) truoc[i]=0;*

*//Tao bang PA*

*int i,j,l=0,d=0;*

*for (i=1; i<=n;i++) {*

*j=i-1;*

*while (j>=1 and a[i]<a[j]) j--, d++;*

*L[i]=L[j]+1;*

*truoc[i]=j;*

*}*

***3.1 Truy vết:***

Với cách làm trên ta đã tìm được độ dài dãy con tăng dài nhất. Bây giờ yêu cầu phải đưa thêm danh sách các phần tử thuộc dãy con tăng dài nhất đó. Lưu có thể có nhiều

Ta sẽ sử dụng kỷ thuật lưu lại dấu vết trên quá trình đi. Ta thấy nếu L[i]=L[j]+1 tức là phần tử nhỏ hơn ai và trước ai chính là aj. Vậy ta sẽ sử dụng một mảng **luu** để lưu lại thông tin này:

//Truy vet de xuat ket qua

*void truyvet(){*

*//tim chi so lon cua mang luu vet luu vao j*

*int j=1;*

*for (int i=2; i<=n;i++) if (luu[i]>=luu[j]) j=i;*

*//Luu cac so theo dk vao mang kq*

*int kq[100];*

*int i=0;*

*while (j!=0){*

*i++;*

*kq[i]=a[j];*

*j=luu[j];*

*}*

*//xuat ket qua*

*for (int j=i; j>=1;j--) cout<<kq[j]<<" ";*

*}*

Như vậy chi phí không gian của bài toán là O(n), chi phí thời gian là O(n2). Áp dụng được cho bài LIQ trên trang SPOJ.

int main() {

nhap();

int len = 0;

for(int i = 1; i <= n; ++i) {

int dp = lower\_bound(L, L+ len, a[i]) - L; //cần xem thêm hàm lower\_bound

if(dp == len) ++ len, L[dp] = a[i];

else L[dp] = min(L[dp], a[i]);

}

for(int i = 0; i <= n; ++i) cout<<lst[i]<<" ";cout<<'\n';

printf("%d\n", len);

return 0;

}

**2/ Ví dụ 2: Xâu con chung dài nhất (dãy con chung dài nhất)**

Thêm một ví dụ nữa cho dễ, cũng là một bài toán rất nổi tiếng.

*Cho hai xâu ký tự. Tìm độ dài xâu con chung dài nhất giữa chúng. Ví dụ với 2 xâu “quetzalcoatl” và “tezcatlipoca” thì xâu con chung dài nhất sẽ là “ezaloa” với độ dài 6.*

Cho 2 xâu X,Y. Hãy tìm xâu con của X và của Y có độ dài lớn nhất.

**Công thức QHĐ**

Bài này làm việc với 2 xâu, nên liên qua đến 2 tham số là chỉ số các phần tử của xâu, phương án đầu tiên sẽ sử dụng làm một mảng 2 chiều.

Gọi L[i,j] là *độ dài xâu con chung dài nhất của xâu Xi gồm i kí tự phần đầu của X*( Xi=X[1..i]) và *xâu Yj gồm j kí tự phần đầu của Y*( Yj =Y[1..j] ).

Chúng ta hãy bắt đầu lần lượt các bài toán con. Đương nhiên, nếu một trong hai xâu là rỗng thì xâu con chung của chúng cũng rỗng. Vậy L(0,j)=L(i,0)=0.

Nếu cả i và j đều dương, chúng ta cần suy xét một vài trường hợp.

1. Nếu hai ký tự cuối cùng của hai xâu Xi,  Yj đều có mặt trong xâu con chung dài nhất, tức là Xi = Yj. Trong trường hợp này, khi xoá đi bất cứ một ký tự nào trong hai ký tự đó đều khiến xâu con chung dài nhất ngắn đi 1 ký tự. Vậy rõ ràng là L(i,j) = L(i−1,j−1)+1.
2. Nếu X[i]  Y[j], chứng tỏ hai phần tử này không thể là phần tử chung. Do đó dãy chung dài nhất trong trường hợp này không được thêm vào phần tử nào. Như vậy ta chọn phương án để tìm tiếp theo là 2 dãy từ X1..Xi với Y1..Yj-1 hoặc X1..Xi-1 với Y1 ..Yj ; phương án nào có độ dài đã chọn có giá trị lớn hơn sẽ đươc chọn để tính tiếp theo: Tức sẽ chọn một trong 2 giá trị L[i-1,j] và L[i,j-1]: Trường hợp này ta lấy độ dài L(i,j) lớn lớn, tức là:

L(i,j)= Max (L(i−1,j) , L(i,j-1))

Trong cả ba trường hợp trên, chúng ta phải chọn ra trường hợp nào cho kết quả là xâu con chung dài nhất (với bài toán này thì chỉ cần đưa ra độ dài đó là đủ).

Ta có công thức quy hoạch động như sau:

L[0,j]=L[i,0]=0.

L[i,j] = L[i−1,j−1]+1 nếu Xi = Yj.

L[i,j] = max( L(i−1,j), L[i,j−1] ) nếu Xi≠Yj.

Bảng phương án như như sau:

X="aggttab"

Y="gxtxayb"

Kết quả: gtab

L[i,j] = L[i−1,j−1]+1 nếu Xi = Yj.

L[i,j] = max( L(i−1,j), L[i,j−1] ) nếu Xi≠Yj.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| j  i | | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |  |
| Y= | g | x | t | x | a | y | b |  |
| 0 | X= | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |  |
| 1 | a | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 |  |
| 2 | g | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |  |
| 3 | g | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |  |
| 4 | t | 0 | 1 | 1 | 2 | 2 | 2 | 2 | 2 |  |
| 5 | a | 0 | 1 | 1 | 2 | 2 | 3 | 3 | 3 | P |
| 6 | b | 0 | 1 | 1 | 2 | 2 | 3 | 3 | 4 | L |

**Cài đặt**

Bảng phương án là một mảng 2 chiều L[0..m,0..n] để lưu các giá trị của hàm QHĐ L[i,j].

Đoạn chương trình cài đặt công thức QHĐ trên như sau:

for (int i=1; i<=n;i++) L[i][0]=0;

for (int j=1; j<=m;j++) L[0][j]=0;

//Tao bang PA

for (int i=1; i<=n;i++)

for(int j=1;j<=m;j++)

if (X[i]==Y[j]) L[i][j]=L[i-1][j-1]+1;

else L[i][j]=max(L[i-1][j],L[i][j-1]);

Kết quả : cout<<L[n][m];

Truy vết:

//Truy vet de xuat ket qua

void truyvetPA1(){

string S="";

int i=n; int j=m;

while (i>0 or j>0){

if (X[i]==Y[j]){

S=X[i]+S; //cout<<X[i];

i--;

j--;

}

else

if (L[i-1][j]==L[i][j]) i--;

else j--;

}

cout<<S;

}

Như vậy chi phí không gian của bài toán là O(n.m), chi phí thời gian là O(n.m). Có một phương pháp cài đặt tốt hơn, chỉ với chi phí không gian O(n) dựa trên nhận xét sau: *để tính ô L[i,j] của bảng phương án, ta chỉ cần 3 ô L[i–1,j–1],L[i–1,j] và L[i,j–1]. Tức là để tính dòng L[i] thì chỉ cần dòng L[i–1]*. Do đó ta chỉ cần 2 mảng 1 chiều để lưu dòng vừa tính (P) và dòng đang tính (L) mà thôi. Cách cài đặt mới như sau:

void PA2(){

//khoi tao

for (int j=0; j<=m;j++) LL[j]=0;

//Tao bang PA

for (int i=1;i<=n;i++){

for (int k=1; k<=m;k++) P[k]=LL[k];// luu lai mang LL o buoc truoc

for (int j=1; j<=m;j++)

if (X[i]==Y[j]) LL[j]=P[j-1]+1;

else LL[j]=ma(LL[j-1],P[j]);

//xuat bang pt

for(int k=1;k<=m;k++) cout<<P[k]<<" "; cout<<'\n';

}

}

**(Chưa có mã lệnh truy vết)**

**3/ Ví dụ 3: Đồng xu – Đổi tiền (Mô hình Bài toán cái túi mỗi vật được chọn nhiều lần)**

Đây là một ví dụ rất kinh điển khi học về quy hoạch động. Có thể có nhiều cách phát biểu khác nhau nhưng về cơ bản, nội dung của nó sẽ tương tự như sau:

Cho số tiền M đồng, và *n* loại tiền (đồng xu) với các mệnh giá *w*[1], w[2],…, w[*n*], hãy tìm phương án đổi số tiền M (đồng) sao cho số tờ tiền (đồng xu) dùng ít nhất. Tất nhiên, số lượng tờ tiền (đồng xu) là không giới hạn. ***(hiểu n loại tiền phải có mệnh giá khác nhau)***

*Giả sử chúng ta có n đồng xu nặng lần lượt là W1, W2, ..., Wn, và bài toán đặt ra là tìm số lượng đồng xu nhỏ nhất để tổng khối lượng của chúng là một giá trị M. Tất nhiên, số lượng đồng xu là không giới hạn.*

Với bài toán này, chúng ta cần xây dựng và giải các bài toán con gối nhau. Với ví dụ của chúng ta, mỗi bài toán con F(j) với j <= M là bài toán tìm số đồng xu nhỏ nhất để khối lượng của chúng là j. và F(j) = k chính là số lượng đồng xu nhỏ nhất đó.

Chúng ta sẽ áp dụng phương pháp quy hoạch động bằng cách bắt đầu từ bài toán con F(0)=0 sau đó tiếp tục với các bài toán con lớn hơn. Lời giải của các bài toán con sẽ được xây dựng lần lượt cho đến chúng ta xây dựng đến bài toán F(M) và đó chính là kết quả của bài toán lớn. Một điều cần lưu ý với kỹ thuật này là bài toán con tiếp theo sẽ không thể giải được nếu chúng ta chưa giải bài toán con trước đó.

Cuối cùng là phần khó nhất của mọi bài toán quy hoạch động, đó là trả lời câu hỏi: cấu trúc con tối ưu của bài toán này ở đâu. Hay nói một cách khác, làm thế nào để từ những bài toán nhỏ hơn có thể tổ hợp ra lời giải cho bài toán lớn. Với ví dụ kinh điển này, mọi thứ sẽ tương đối đơn giản, nhưng với những bài toán phức tạp hơn, chúng ta cần suy nghĩ và tính toán nhiều hơn.

Quay trở lại với bài toán của chúng ta. Giả sử j là tổng khối lượng của các đồng xu nặng lần lượt là V1, V2, ..., Vj. Để có được khối lượng j, chúng ta cần thêm đúng 1 đồng xu nặng U vào khối lượng Q sao cho Q + U = j. Tất nhiên, bài toán con F(Q) chúng ta đã có lời giải nên chúng ta sẽ biết được cần bao nhiêu đồng xu cho F(j). Và vì có nhiều đồng xu U(nhiều nhưng hữu hạn) nên chúng ta có thể cần đến nhiều bài toán con trước đó, và F(j) là giá trị nhỏ nhất sau khi tổng hợp những bài toán con đó.

Ví dụ với n = 3, M = 11, W = [1, 3, 5].

* Bắt đầu với bài toán con 0 ta có F(0) = 0
* Với bài toán con 1, có 1 đồng xu (nặng 1) có thể thêm vào từ 0 đồng xu nào cả. Vậy F(1) = F(0) + 1 = 1.
* Với bài toán con 2, cũng chỉ có 1 đồng xu (nặng 1) có thể thêm vào từ 1 đồng xu. Vậy F(2) = F(1) + 1 = 2.
* Với bài toán con 3, chúng ta có thể thêm 1 đồng xu 3 vào 0 đồng xu hoặc thêm 1 đồng xu 1 vào 2 đồng xu. Rõ ràng là cách đầu tiên cho kết quả nhỏ hơn. Vậy F(3) = min(F(2) + 1, F(0) + 1) = min(3, 1) = 1
* …
* Cứ tiếp tục như vậy cho đến bài toán M chính là đáp án chúng ta cần tìm.

Về mặt cài đặt, quy hoạch động thường lưu kết quả vào một mảng. Có thể cài mảng 1 chiều hoặc 2 chiều.

**\* Cài đặt bằng mảng 2 chiều (đổi tiền):**

Gọi C[i][j] là số tờ tiền ít nhất khi đổi j đồng, với i tờ tiền có mệnh giá từ w1,w2… wi.

Khi xét đến đồng thứ i có mệnh giá w[i] để đổi số tiền j đồng, Ta có các trường hợp sau:

TH1: Nếu i=0, j<=m nghĩa là không có tờ tiền nào để đổi cho j đồng, nên số cách đổi ở đây là vô cùng tức không đổi được(không có tiền không thể đổi): ***C[0][j] = vô cùng.***

TH2: Nếu j=0; tức có i tờ tiền, đổi cho 0 đồng (có tiển để đổi nhưng không có cách nào để đổi) nên số cách là 0: ***C[i][0] = 0;***

TH3: Nếu j < w[i] tức số tiền cần đổi j nhỏ hơn tờ tiền đang xét, nghĩa là đồng thứ i này không có tác dụng để đổi, tức không thêm được đồng có mệnh giá w[i] vào danh sách được chọn, vậy nó sẽ chấp nhận cách đổi trước đó, tức là như đổi j đồng với số đồng được lựa chọn để lấy là w1, w2, … , wi-1:

Nếu (j<w[i]) thì ***C[i][j] = C[i-1][j] ; (1)***

TH4: j>=w[i], có nghĩa số tiền cần đổi j lớn hơn mệnh giá tờ đang xét w[i], nên có thể nhận W[i] (vì j-W[i]>=0 nên đổi được), có 2 trường hợp để lựa chọn, ta chọn trường hợp nào có số tờ ít hơn thì lấy:

Một là, không chọn đồng thứ i (đồng có mệnh giá W[i]) mà chọn theo cách cũ như khi đổi j đồng với i -1 mệnh giá w1, w2, … , wi-1.

tức ***là: C[i][j] = C[i-1][j] ; (2)***

Hai là, Chọn đồng thứ i với mệnh giá w[i], Khi số tiền cần đổi j-w[i] có cách đổi, khi đó số cách đổi sẽ cộng thêm 1, tức ***C[i][j] = C[i-1][j-w[i]] +1 ; (3)***

Lấy số cách nhỏ nhất trong 2 cách (2) và (3):

***C[i,j] = min(C[i-1,j] , C[i][j-w[i] + 1);***

**Cài đặt:**

*int C[100][100];*

*for( int j=1;j<=M;j++) C[0][j] = M+1;//vô cùng (không đổi được)*

*for(int i=1;i<=n;i++) C[i][0] = 0;*

*for(int i=1;i<=n;i++)*

*for (int j=1;j<=M;j++)*

*if (j<w[i]) C[i][j] = C[i-1][j] ;//nếu j<Wi ko đổi được nên chọn cách đổi trước đó (i-1tờ)*

*else //Có thể chọn cách ở bước trước hoặc cách mới, cách nào nhỏ hơn thì lấy.*

*C[i][j] = min(C[i-1][j], 1+ C[i][j-w[i]]) ;*

# Nếu đầu vào như sau: n = 3, M = 11, w = [3, 1, 5]

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| j | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
| F[i,j] | 0 | 12 | 12 | 12 | 12 | 12 | 12 | 12 | 12 | 12 | 12 | 12 |
| W[1]=3; F[i,j] | 0 | 12 | 12 | 1 | 12 | 12 | 2 | 12 | 12 | 3 | 12 | 12 |
| W[2]=1; F[i,j] | **0** | **1** | **2** | 1 | 2 | 3 | 2 | 3 | 4 | 3 | 4 | 5 |
| W[3]=5;F[i,j] | **0** | **1** | **2** | **1** | **2** | 1 | 2 | 3 | 2 | 3 | 2 | 3 |

***Truy vết để tìm ra nghiệm thông thường nghiệm ko tối ưu, cần sắp xếp để đạt pa tối ưu:***

Bắt đầu từ ô C[n][m] kiểm tra nếu số tiền j> mệnh giá w[i] (i đầu tiên bằng N)

Khởi tạo i=n; j=m;

* Lặp khi (j>0)

Nếu (j-w[i]<0) giảm i--; // Không chọn tờ thứ i

Nếu (j-w[i]>=0)

Nếu C[i][j]==C[i-1][j] thì dự kiến chọn tờ i-1

Nếu C[i][j]=C[i][j-w[i]]+1 thì chọn tờ thứ i (lưu )

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| i=n | J=m | Wi | C[i,j] | J=j-wi | Thoả ĐK | Chọn tờ thứ i | Mệnh giá |
| 3 | 11 | 5 | 3 | 6 | C[i][j]==C[i][j-w[i]] +1 | 3 | 5 |
| 3 | 6 | 5 | 2 | 1 | C[i][j]==C[i][j-w[i]] +1 | 3 | 5 |
| 3 | 1 | 5 | 1 | -4<0 |  |  |  |
| 2 | 1 | 1 | 1 | 0 | C[i][j]==C[i][j-w[i]] +1 | 2 | 1 |
| 2 | 0 |  |  |  |  |  |  |

**Cài đặt truy vết trên mảng 2 chiều.**

//--------------TRUY VET va LUU VAO MANG dd -------------

if (C[n][M]==M+1) cout<<"Khong doi duoc";

else{

cout<<C[n][M]<<endl;

int maxx=w[1];

for (int i=2;i<=n;i++) if (maxx<w[i]) maxx=w[i];

int dd[maxx+1];

memset(dd,0,sizeof(dd));

int i=n; int j=M;

int k=0;

while (j>0){

if (j-w[i]<0) i--; //khong chon menh gia nay

else {

if (C[i][j]!=C[i][j-w[i]]+1) i--; //(C[i][j]==C[i-1][j]) dự kiến chọn tờ i-1

else j=j-w[i], dd[i]++; //chọn tờ thứ i

}

}

//-------------------XUAT KET QUA TRU VET TU MANG dd-------------

cout<<'\n';

for (int i=1;i<=n;i++)

if (dd[i]>0) cout<<"to tien "<<w[i]<<" duoc chon "<<dd[i]<<" lan"<<'\n';

}

**Lưu ý:** *Để tìm được số tờ ít nhất trước khi lập bảng phương án, có thể xem xét sắp xếp mảng w[i] (với i=1..n) tăng dần* ***Sort(w+1,w+n,td);***

**\* Cài đặt bằng mảng 1 chiều:**

Trong ví dụ của chúng ta, mảng F[0..M] sẽ lưu kết quả cho từng bài toán con. Nói cách khác, F[j] = i nghĩa là cần ít nhất i đồng xu để có khối lượng là j. Toàn bộ mảng này sẽ được tính bằng vòng lặp.

Khi ta code thì chỉ cần sử dụng một mảng một chiều để tính lần lượt các hàng (hàng sau quy hoạch dựa trên hàng trước) còn vô cùng sẽ dùng là m+1

Gọi F[j] là số tờ tiền cần dùng ít nhất để đổi số tiền j đổng với i tờ tiền mệnh giá w*i*. Vậy kết quả bài toán cần tìm là F[m].

Bài toán con: F[0]=0; với mọi m,n.

Khi không có cách chọn, số cách chọn j đồng là vô cùng (Vô cùng bằng m+1)

Tương tự mảng hai chiều, ta chỉ một cần mảng một chiều để lưu hàng đang xét i, và chỉ cần tính các cột F[j].

- Khi *j*>= w[*i*] :

+ TH1: Có chọn tờ tiền w[*i*] để đổi. Khi đó, rõ ràng là *F*[*j*] <= *F*[j-w[i]]+1.

Suy ra *F*[*j*] = 1 + F[*j*-*w*[i]] *(với i=1, …, n & j >=w[i])*

+ TH2: Không chọn tờ tiền w[*i*] để đổi. Khi đó, rõ ràng là F[j] không đổi so với khi đổi j đồng với i-1 mệnh giá w1, w2, … , wi-1

*Vì vậy F*[*j*]trong trường hợp này là Min của F[j] cũ và 1 + F[*j*-*w*[i]], C*ông thức chọn là:*

*F*[*j*] = min(F[j], 1 + F[*j*-*w*[i]] ) | *i*=1, …, *n* & *j* >=*w*[*i]*

Để truy xuất lại kết quả ta cần lưu vết cách chọn bằng cách dùng mảng ***tr (trước)*** để lưu lại chỉ số j khi chọn như sau ***tr[j]=j-w[i];***

Bảng phướng án sau mô tả toàn bộ quá trình này.

# Nếu đầu vào như sau: n = 3, M = 11, w = [1, 3, 5]

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| j | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
| F[j] | 0 | 12 | 12 | 12 | 12 | 12 | 12 | 12 | 12 | 12 | 12 | 12 |
| W[1]=1 |  | 1+f[1-1]= | 1+f[2-1]= | 1+f[3-1]= | 1+f[4-1]= | 1+f[5-1]= | 1+f[6-1]= | 1+f[7-1]= | 1+f[8-1]= | 1+f[9-1]= | 1+f[10-1]= | 1+f[11-1]= |
| F[j] | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
| Trước[j] | 0 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| W[2]=3 |  |  |  | 1+f[3-3]= | 1+f[4-3]= | 1+f[5-3]= | 1+f[6-3]= | 1+f[7-3]= | 1+f[8-3]= | 1+f[9-3]= | 1+f[10-3]= | 1+f[11-3]= |
| F[j] | **0** | **1** | **2** | 1 | 2 | 3 | 2 | 3 | 4 | 3 | 4 | 5 |
| Trước[j] | **0** | **0** | **1** | 0 | 1 | 2 | 3 | 4 | 8 | 6 | 7 | 8 |
| W[3]=5 |  |  |  |  |  | 1+f[5-5]= | 1+f[6-5]= | 1+f[7-5]= | 1+f[8-5]= | 1+f[9-5]= | 1+f[10-5]= | 1+f[11-5]= |
| F[j] | **0** | **1** | **2** | **1** | **2** | 1 | 2 | 3 | 2 | 3 | 2 | 3 |
| Trước[j] | **0** | **0** | **1** | **0** | **1** | 0 | 1 | 2 | 3 | 4 | 5 | 6 |

**Cài đặt bảng phương án với mảng một chiều**

Tương tự, có thể dùng một mảng phụ P[] giống như bài xâu con đổi tiền để lưu lại các giá trị ở bước trước (i-1) khi xử lí đến đồng mệnh giá thứ i. Tuy nhiên bài này không cần vì không dùng đến phần tử thứ [i-1,j-1].

Cách dưới đây không dùng mảng phụ:

B1. Khởi tạo: F[0]=0; for (int i=0; i<=M;i++) F[i]=M+1;

B2. //Tạo bảng PÁ

for (int i=1; i<=n;i++) {

for (int j=w[i];j<=M;j++){ //F[j]=mi(F[j],1+F[j-w[i]])

if ( F[j] >= 1+F[j-w[i]]) {

F[j]= 1+F[j-w[i]];

tr[j]=j-w[i];

}

}

}

**Truy vết để đưa ra kết quả**

B1. Tìm Max của Wi để làm chỉ số biến đếm dd

B2. Khai báo mảng dd để tính số lượng các đồng được chọn (***int dd[Max+1]***)

B3. Bắt đầu từ j=m.

B4. Khi j>0 lặp

Tính K=j-trước[j]

Đếm số k lưu vào mảng dd

J=trước[j]

Ví dụ: # Nếu đầu vào như sau: n = 3, M = 11, w = [1, 3, 5]

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| j | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|  |  |  |  |  |  |  |  |  |  |  |  |  |
| F[j] | **0** | **1** | **2** | **1** | **2** | 1 | 2 | 3 | 2 | 3 | 2 | 3 |
| Trước[j] | **0** | **0** | **1** | **0** | **1** | 0 | 1 | 2 | 3 | 4 | 5 | 6 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| J | truoc[j] | K=j-truoc[j] | Ghi nhận K | J = truoc[j] |  |
| 11 | 6 | 11-6=5 | 5 | 6 |  |
| 6 | 1 | 6-1=5 | 5 | 1 |  |
| 1 | 0 | 1-0=1 | 1 | 0 |  |
| 0 | Dừng | | | | |

if (F[M]==M+1) cout<<"Khong doi duoc";

else{

int maxx=w[1];

for (int i=2;i<=n;i++) if (maxx<w[i]) maxx=w[i];

int dd[maxx+1];

memset(dd,0,sizeof(dd));

int j=M,i=n;cout<<'\n';

int k=0;

// luu vet vao bien tr

while(j>0){

k=j-tr[j];cout<<"k= "<<k<<", j = "<<j<<" "<<endl;

if (k>0) dd[k]++;

j=tr[j];

}

//Xuat truy vet trong bien dem

for (int i=1;i<=n;i++)

if (dd[w[i]]>0) cout<<"to tien "<<w[i]<<" duoc chon "<<dd[w[i]]<<" lan"<<'\n';

}

}

Nhận xét: Như vậy chi phí không gian của bài toán là O(n), chi phí thời gian là O(n2).

**4/ Ví dụ 4: VALI (A) Một vật chọn nhiều lần**

**1/ Mô hình (giống bài toán đồng xu)**

Cho n vật, vật i nặng Ai và có giá trị Bi. Hãy chọn ra một số vật để cho vào balô sao cho tổng khối lượng không vượt quá K và tổng giá trị là lớn nhất. Chú ý rằng mỗi vật có thể được chọn nhiều lần.

**Có thể phát biểu bài toán:**

Trong siêu thị có n đồ vật (n≤1000), đồ vật thứ i có trọng lượng là W[i]≤10­00 và giá trị V[i] ≤1000. Một tên trộm đột nhập vào siêu thị, tên trộm mang theo một cái túi có thể mang được tối đa trọng lượng M (M≤1000). Hỏi tên trộm sẽ lấy đi những đồ vật nào để được tổng giá trị lớn nhất.

Giải quyết bài toán trong các trường hợp sau:

* Mỗi vật chỉ được chọn một lần.
* Mỗi vật được chọn nhiều lần (không hạn chế số lần)

**InputData:** file văn bản **Bag.inp**

* Dòng 1: n, M cách nhau ít nhất một dấu cách
* n dòng tiếp theo: Mỗi dòng gồm 2 số Wi, Vi, là chi phí và giá trị đồ vật thứ i.

**OutputData:** file văn bản **bag.out**: Ghi giá trị lớn nhất tên trộm có thể lấy

**Example**

|  |  |
| --- | --- |
| Input | Ouput |
| 5 15  12 4  2 2  1 1  1 2  4 10 | 36 |

**2. Công thức**

Ta nhận thấy rằng: Giá trị của cái túi phụ thuộc vào 2 yếu tố: Có bao nhiêu vật đang được xét và trọng lượng còn lại cái túi có thể chứa được, do vậy chúng ta có 2 đại lượng biến thiên. Cho nên hàm mục tiêu sẽ phụ thuộc vào hai đại lượng biến thiên. Do vậy bảng phương án của chúng ta sẽ là bảng 2 chiều.

Gọi F[i][j] là tổng giá trị lớn nhất của cái túi khi xét từ vật 1 đến vật i và trọng của cái túi chưa vượt quá j. Với giới hạn j, việc chọn tối ưu trong số các vật {1,2,…,i-1,i} để có giá trị lớn nhất sẽ có hai khả năng:

Nếu không chọn vật thứ i thì F[i][j] là giá trị lớn nhất có thể chọn trong số các vật {1,2,…,i-1} với giới hạn trọng lượng là j, tức là:

**F[i][j] =F[i-1][j].**

Nếu có chọn vật thứ i (phải thỏa điều kiện W[i] ≤ j) thì F[i,j] bằng giá trị vật thứ i là V[i] cộng với giá trị lớn nhất có thể có được bằng cách chọn trong số các vật đã xét, (tức i vật){1,2,…,i} (vì vật i vẫn có thể được chọn tiếp) với giới hạn trọng lượng j-W[i] tức là về mặt giá trị thu được:

**F[i][j]=V[i]+F[i][j-W[i]]**

F[n,m] sẽ là đáp số của bài toán (là giá trị lớn nhất có được nếu chọn n vật và tổng khối lượng không vượt quá M).

Do vậy chúng ta có **công thức truy hồi** như sau:

* **F[0][j] = 0** (hiển nhiên) – Bài toán con nhỏ nhất.
* **F[i,j]= max(F[i-1][j], V[i]+F[i][j-W[i]]**
* Trong đó: F[i–1][j] là giá trị có được nếu không đưa vật i vào balô (j>=Wi), F[i,j–Wi]+Vi  là giá trị có được nếu chọn vật i.

F[i,0]=0;

F[0,j]=0.

**Bảng phương án:**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| i/j |  |  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| 0; | Wi | Vi | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1; | 12 | 4 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 4 | 4 | 4 | 4 |
| 2; | 2 | 2 | 0 | 0 | 2 | 2 | 4 | 4 | 6 | 6 | 8 | 8 | 10 | 10 | 12 | 12 | 14 | 14 |
| 3; | 1 | 1 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| 4; | 1 | 2 | 0 | 2 | 4 | 6 | 8 | 10 | 12 | 14 | 16 | 18 | 20 | 22 | 24 | 26 | 28 | 30 |
| 5; | 4 | 10 | 0 | 2 | 4 | 6 | 10 | 12 | 14 | 16 | 20 | 22 | 24 | 26 | 30 | 32 | 34 | 36 |

* Chúng ta có thể chọn vật 4 (3 lần) và vật 5 (3 lần).

1. **Cài đặt**

void PAM2C(){

//khoi tao

memset(S,0,sizeof(S));

//tao bang phuong an

for (int i=1; i<=n;i++) {

for (int j=1;j<=m;j++)

if(j<w[i]) S[i][j]=S[i-1][j];

else S[i][j]=ma(S[i-1][j],S[i][j-w[i]] + v[i]) ;

}

}

**Truy vết để đưa ra nghiệm**

**j=m; i=n**

* Lặp khi (j>0 )

Nếu (j-w[i]<0) giảm i--; //không có vật i

Nếu (j-w[i]>=0)

Nếu F[i][j]==F[i-1][j] i--; //thì dự kiến chọn vật thứ i-1

Nếu F[i][j]=F[i][j-w[i]]+ V[i] thì chọn vật thứ i (lưu )

**Cài đặt truy vết**

//--------------------- TRUY VET - GHI KET QUA---------------------

int maxx=w[1]; //tìm max

for (int i=2;i<=n;i++) if (maxx<w[i]) maxx=w[i];

int dd[maxx+10];

memset(dd,0,sizeof(dd));

int i=n; int j=m;

while (j>0 and i>0){

if (j<w[i]) i-- ,cout<<" chuyen sang vat moi"<<endl;//khong chon vat nao

else{ // j>=w[i]

if (S[i][j]==S[i-1][j]) i--; //du kien Chon vat i-1

else // if (S[i][j]==S[i][j-w[i]]+v[i])

j=j-w[i], dd[i]++; //chon vat thu i

}

}

//--------- Xuat truy vet trong bien dem---------------------------

for (int i=1;i<=n;i++)

if (dd[i]>0)

cout<<"vat "<<i<<"khoi luong "<<w[i]<<";giatri"<<v[i]<<"duoc chon "<<dd[i]<<" lan"<<"; gia tri "<<dd[i]\*v[i]<<'\n';

}

**Cải tiến thành mảng một chiều**

Ta có thể dùng một mảng 2 chiều để lưu bảng phương án, tuy nhiên dựa trên nhận xét rằng để tính dòng i của bảng phương án chỉ cần dòng i–1, ta chỉ cần dùng 2 mảng một chiều F và P có chỉ số từ 0 đến m để lưu 2 dòng đó. Đoạn chương trình con tính bảng phương án như sau:

memset(P,0,sizeof(P));

memset(F,0,sizeof(F));//F[0,j]=0;voi moi j

for (int i=1; i<=n;i++) {

for (int k=1; k<=m;k++) P[k]=F[k]; //Luu mang F o hang thu i-1

for (int j=1;j<=m;j++)

if (j<w[i]) F[j]=P[j];

else F[j]= ma(F[j],F[j-w[i]] + v[i]) ;

//for (int i=1; i<=m;i++)cout<<F[i]<<" "; cout<<'\n';

}

Nếu để ý kĩ bạn sẽ thấy rằng đoạn trình trên chỉ viết giống công thức QHĐ chứ chưa tối ưu. Chẳng hạn đã có lệnh gán P=F, sau đó lại có gán F[j] =P[j] với các giá trị j<w[i] là không cần thiết. Có thể tự cải tiến để chương trình tối ưu hơn. Chi phí không gian của cách cài đặt trên là O(m) và chi phí thời gian là O(n.m).

for (int i=0; i<=m;i++) F[i]=0;

F[0]=0;

//tao bang phuong an

for (int i=1; i<=n;i++) {

for (int j=w[i];j<=m;j++)

F[j]=max(F[j],F[j-w[i]] + v[i]) ;

for (int i=1; i<=m;i++)cout<<F[i]<<" ";cout<<'\n';

}

**Truy vết**

**4/ Ví dụ 4: VALI (B) Một vật chọn 1 lần**

**1/ Mô hình (giống bài toán vali (A) nhưng các vật chỉ được chọn 1 lân**

Cho n vật, vật i nặng Ai và có giá trị Bi. Hãy chọn ra một số vật để cho vào balô sao cho tổng khối lượng không vượt quá K và tổng giá trị là lớn nhất. Chú ý rằng mỗi vật có thể được chọn 1 lần.

**Có thể phát biểu bài toán:**

Trong siêu thị có n đồ vật (n≤1000), đồ vật thứ i có trọng lượng là W[i]≤10­00 và giá trị V[i] ≤1000. Một tên trộm đột nhập vào siêu thị, tên trộm mang theo một cái túi có thể mang được tối đa trọng lượng M (M≤1000). Hỏi tên trộm sẽ lấy đi những đồ vật nào để được tổng giá trị lớn nhất.

Giải quyết bài toán trong các trường hợp sau:

* Mỗi vật chỉ được chọn một lần.
* Mỗi vật được chọn nhiều lần (không hạn chế số lần)

**InputData:** file văn bản **Bag.inp**

* Dòng 1: n, M cách nhau ít nhất một dấu cách
* n dòng tiếp theo: Mỗi dòng gồm 2 số Wi, Vi, là chi phí và giá trị đồ vật thứ i.

**OutputData:** file văn bản **bag.out**: Ghi giá trị lớn nhất tên trộm có thể lấy

**Example**

|  |  |
| --- | --- |
| Input | Ouput |
| 5 15  12 4  2 2  1 1  1 2  4 10 | 15 |

**2. Công thức**

**Trường hợp mỗi vật được chọn 1 lần**

Ta nhận thấy rằng: Giá trị của cái túi phụ thuộc vào 2 yếu tố: Có bao nhiêu vật đang được xét và trọng lượng còn lại cái túi có thể chứa được, do vậy chúng ta có 2 đại lượng biến thiên. Cho nên hàm mục tiêu sẽ phụ thuộc vào hai đại lượng biến thiên. Do vậy bảng phương án của chúng ta sẽ là bảng 2 chiều.

Gọi F[i,j] là tổng giá trị lớn nhất của cái túi khi xét từ vật 1 đến vật i và trọng của cái túi chưa vượt quá j. Với giới hạn j, việc chọn tối ưu trong số các vật {1,2,…,i-1,i} để có giá trị lớn nhất sẽ có hai khả năng:

Nếu không chọn vật thứ i thì F[i,j] là giá trị lớn nhất có thể chọn trong số các vật {1,2,…,i-1} với giới hạn trọng lượng là j, tức là:

**F[i,j]:=F[i-1,j].**

Nếu có chọn vật thứ i (phải thỏa điều kiện W[i] ≤ j) thì F[i,j] bằng giá trị vật thứ i là V[i] cộng với giá trị lớn nhất có thể có được bằng cách chọn trong số các vật đã chọn, tức i-1 vật đã chọn {1,2,…,i-1} với giới hạn trọng lượng j-W[i] tức là về mặt giá trị thu được:

**F[i,j]:=V[i]+F[i-1,j-W[i]]**

F[n,m] sẽ là đáp số của bài toán (là giá trị lớn nhất có được nếu chọn n vật và tổng khối lượng không vượt quá M).

Do vậy chúng ta có **công thức truy hồi** như sau:

* **F[0,j] = 0** (hiển nhiên) – Bài toán con nhỏ nhất.
* **F[i,j]= max(F[i-1,j], V[i]+F[i-1,j-W[i]]**
* Trong đó: F[i–1][j] là giá trị có được nếu không đưa vật i vào balô (j>=Wi),

F[i-1,j–Wi]+Vi  là giá trị có được nếu chọn vật i.

F[i,0]=0;

F[0,j]=0.

**Bảng phương án:**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| i/j |  |  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| 0; | Wi | Vi | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1; | 12 | 4 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 4 | 4 | 4 | 4 |
| 2; | 2 | 2 | 0 | 0 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 4 | 4 | 6 | 6 |
| 3; | 1 | 1 | 0 | 1 | 2 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 4 | 5 | 6 | 7 |
| 4; | 1 | 2 | 0 | 2 | 3 | 4 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 6 | 7 | 8 |
| 5; | 4 | 10 | 0 | 2 | 3 | 4 | 10 | 12 | 13 | 14 | 15 | 15 | 15 | 15 | 15 | 15 | 15 | 15 |

* Vậy chúng ta có thể chọn vật 2, 3, 4, 5.
* **Cài đặt tương tự bài Vali A**

void PAM2C(){

//khoi tao

memset(S,0,sizeof(S));

//tao bang phuong an

for (int i=1; i<=n;i++) {

for (int j=1;j<=m;j++)

if(j>=w[i]) S[i][j]=ma(S[i-1][j],S[i-1][j-w[i]] + v[i]) ;

else S[i][j]=S[i-1][j];

}

//------------- Xuat bang phuong an---------------

for (int i=1; i<=n;i++){

for (int j=1; j<=m;j++) cout<<S[i][j]<<" "; cout<<'\n';

}

//--------------------- TRUY VET - GHI KET QUA---------------------

int maxx=w[1];

for (int i=2;i<=n;i++) if (maxx<w[i]) maxx=w[i];

int dd[maxx+10];

memset(dd,0,sizeof(dd));

int i=n; int j=m;

while (j>0 and i>0){

if (j<w[i]) i-- ,cout<<" chuyen sang vat moi"<<endl;//khong chon vat nao else{ // j>=w[i]

if (S[i][j]==S[i-1][j]) i--; //du kien Chon vat i-1

else // if (S[i][j]==S[i-1][j-w[i]]+v[i])

j=j-w[i], dd[i]++,i--; //chon vat thu i }

}

//--------- Xuat truy vet trong bien dem---------------------------

for (int i=1;i<=n;i++)

if (dd[i]>0) cout<<"vat "<<i<<" khoi luong "<<w[i]<<"; gia tri"<<v[i]<<" duoc chon "<<dd[i]<<" lan"<<"; gia tri "<<dd[i]\*v[i]<<'\n';

}

***Hoặc một cách truy vết khác***

// ----------- Truy vet bang cach khac -------------------------------------------

i=n; j=m;int k=0, luu2[n+1];

memset(luu2,0,sizeof(luu2));

for (int i=n; i>0; i--){

if (S[i][j]!=S[i-1][j]){

luu2[++k]=i;//cout<<"vat "<<i<<"duoc chon "<<endl;

j=j-w[i];

}

}

for (int i=1; i<=k;i++) cout<<"vat "<<luu2[i]<<"duoc chon "<<endl;

**Cách cài đặt mảng một chiều:**

**…………..**

**5/ Ví dụ 5:**

Có n phần thưởng được chia cho m học sinh giỏi được xếp hạng từ 1..n ( học sinh có thứ hạng nhỏ hơn học giỏi hơn). Hỏi có bao nhiêu cách chia phần thưởng trên sao cho thoả mãn 2 điều kiện sau:

1. Học sinh A giỏi hơn học sinh B thì số phần thưởng của học sinh A  phần thưởng của học sinh B.
2. Tất cả phần thưởng phải chia hết cho học sinh.

***Ví dụ:***

|  |  |  |
| --- | --- | --- |
| ***N=7; M=5;Kq=13*** | ***N=5; M=5; Kq=7*** | ***N=5; M=7; Kq=7*** |
| 7 0 0 0 0  6 1 0 0 0  5 2 0 0 0  5 1 1 0 0  4 3 0 0 0  4 2 1 0 0  4 1 1 1 0  3 3 1 0 0  3 2 2 0 0  3 2 1 1 0  3 1 1 1 1  2 2 2 1 0  2 2 1 1 1 | 5 0 0 0 0  4 1 0 0 0  3 2 0 0 0  3 1 1 0 0  2 2 1 0 0  2 1 1 1 0  1 1 1 1 1 | 5 0 0 0 0 0 0  4 1 0 0 0 0 0  3 2 0 0 0 0 0  3 1 1 0 0 0 0  2 2 1 0 0 0 0  2 1 1 1 0 0 0  1 1 1 1 1 0 0 |

** Nhận xét :**

-Độ phức tạp của bài toán phụ thuộc vào 2 giá trị n và m.

-Gọi Ti (i=1..n) là số phần thưởng của học sinh i nhận được. Số phần thưởng của các học sinh nhận được phải thoả mãn 2 điều kiện sau :

T1  T2  T3 …. Tn  0 (1)

T1 + T2 + T3 +….+ Tn = m (2)

Ta cần xây dựng một mảng 2 chiều f.

Với f[i,j] là số cách chia j phần thưởng cho i học sinh. Ta lần lượt cho i đạt đến n và j đạt đến m sẽ thu được số cách chia m phần thưởng cho n học sinh giỏi.

+ ***Khởi tạo :***

f[i,0] – Có i học sinh giỏi được xếp hạng từ 1 đến i, nhưng không có phần thưởng nào (vì j=0) do đó ta có 1 cách chia duy nhất là mỗi học sinh nhận 0 phần thưởng . Điều đó đồng nghĩa với việc f[i, 0]=1 (i=1..n) .

f[0,j] – Có j phần thưởng nhưng không có học sinh nào do đó ta không thể chia được. Điều này đồng nghĩa với f[0,j]=0 (j=1..m)

Với i, j  0 : Ta xét f[i, j] là số cách chia j phần thưởng cho i học sinh giỏi. có 2 trường hợp sau :

* **Trường hợp 1:** Nếu i>j ( số phần thưởng ít hơn số học sinh giỏi) do vậy i-j học sinh ở sau cùng sẽ không nhận được phần thưởng nào. Điều này cũng có nghĩa j phần thưởng này chỉ được chia cho j học sinh đầu tiên ( còn học sinh thứ j+1,j+2,..i không có phần thưởng nào). Và hiển nhiên số cách chia j phần thưởng cho i học sinh trong thường hợp này cũng bằng số cách chia j phần thưởng cho j học sinh. Do đó ta có : f[i,j]=f[j,j]
* **Trường hợp 2 :** Nếu i≤ j ( số phần thưởng lớn hơn số học sinh)

Theo đề bài thì số phần thưởng phải chia hết và phần thưởng của học sinh i phải  phần thưởng của học sinh i+1. Do đó trong trường hợp này ta thấy học sinh giỏi ở cuối cùng sẽ có 1 trong 2 khả năng (do i≤ j) sau :

\* Nhận được phần thưởng ( (1) xảy ra trường hợp Ti>0 )

\* Không nhận được phần thưởng nào ( (1) xảy ra trường hợp Ti= 0 )

***a) Không nhận được phần thưởng nào :*** đồng nghĩa với j phần thưởng này chỉ chia cho i-1 học sinh. Như nhận xét ở trường hợp 1 ta có số cách chia phần thưởng trong trường hợp này : ***f[i,j]=f[i-1,j]***

***b) Nhận được phần thưởng :*** Người cuối cùng nhận được phần thưởng đều đó có nghĩa rằng tất cả học sinh đều nhận được phần thưởng – Tương ứng với việc ta phát đồng đều cho mỗi học sinh 1 phần thưởng. Do vậy số phần thưởng còn lại sẽ là j-i . Bài toán trong trường hợp này trở thành bài toán nhỏ hơn : Chia j-i phần thưởng còn lại cho i học sinh. Do đó trong trường hợp này : f[i,j]=f[i,j-i]

***Tóm lại :*** Với j  i ta có số cách chia phần thưởng là tổng số cách của 2 trường hợp a) và b) do đó : ***f[i,j]=f[i-1,j] + f[i,j-i].***

***Cài đặt chương trình***

*void Chia(){*

*//khoi tao*

*memset(F,0,sizeof(F));*

*for (int i=1; i<=n;i++) F[i][0]=1;*

*for (int j=1; j<=m;j++) F[0][j]=0;*

*//Tao bang PA*

*for (int i=1; i<=n;i++)*

*for(int j=1;j<=m;j++)*

*if (i>j) F[i][j]=F[j][j];*

*else F[i][j]=F[i-1][j]+F[i][j-i];*

*cout<<F[n][m]<<'\n';*

*}*

***Có thể cải tiến sử dụng mảng một chiều để tiết kiệm bộ nhớ***

**6/ Ví dụ 6:**

Cho số tự nhiên N. Tính số cách phân tích N thành tổng các số tự nhiên nhỏ hơn bằng nó. Các cách phân tích là hoán vị của nhâu chỉ tính 1.

|  |  |  |
| --- | --- | --- |
| Ví dụ N=5; có 7 cách | Ví dụ N=6; có 11 cách | Ví dụ N=7; có 15 cách |
| 1. 5=5 2. 5=4+1 3. 5=3+2 4. 5=3+1+1 5. 5=2+2+1 6. 5=2+1+1+1 7. 5=1+1+1+1+1 | 1. 6=6 2. 6=5+1 3. 6=4+2 4. 6=4+1+1 5. 6=3+3 6. 6=3+2+1 7. 6=3+1+1 8. 6=2+2+2 9. 6=2+2+1+1 10. 6=2+1+1+1+1 11. 6=1+1+1+1+1+1 | 1. 7=7 2. 7=6+1 3. 7=5+2 4. 7=5+1+1 5. 7=4+3 6. 7=4+2+1 7. 7=4+1+1+1 8. 7=3+3+1 9. 7=3+2+2 10. 7=3+2+1+1 11. 7=3+1+1+1+1 12. 7=2+2+2+1 13. 7=2+2+1+1+1 14. 7=2+1+1+1+1+1 15. 7=1+1+1+1+1+1+1 |

***Tìm công thức truy hồi:***

Bài toán này phụ thuộc 2 tham số là số cần phần tích n và các số k tham gia vào tổng để cộng lại bằng n. Phương án sử dụng mảng 2 chiều.

Gọi F[i,j] là số cách phân tích số j thành tổng các số ≤ i. Khi đó có 2 cách phân tích số j thành tổng các số nguyên dương ≤ i:

**TH1:** Tổng không chứa số i trong phép phân tích, khi đó số cách phân tích này chính là số cách phân tích số j thành tổng các số nguyên dương <i, tức số cách phân tích bằng số j thành tổng các số số nguyên dương ≤ i-1 **và F[i,j]=F[i-1,j].**

**TH2:** Có chứa ít nhất một số i trong phép phân tích. Khi đó nếu trong các cách phân tích ta bỏ đi số i thì sẽ được các cách phân tích số j-i thành tổng các số nguyên dương ≤ i. (đều này chỉ đúng khi không lặp lại các hoán vị của một cách). Có nghĩa là về mặt số lượng, số các cách phân loại này bằng F[i,j-i].

Trong trường hợp này i>j thì rõ ràng chỉ có các cách phân tích TH1, còn trong trường hợp i≤j thì sẽ có cả các cách phân tích TH1, và TH2. Vì thế:

F[i,j] = F[i-1,j]; nếu i>j

Và F[i,j] = F[i-1,j]+F[i,j-i]; nếu i≤j.

Ta xây dựng công thức F[i,j] từ F[i-1,j] và F[i,j-i]; kết quả cuối cùng là F[n,m]

**Cài đặt:**

void PA(){

//khoi tao

memset(F,0,sizeof(F));

for (int i=1; i<=n;i++) F[i][0]=1;

//Tao bang PA

for (int i=1; i<=n;i++)

for(int j=1;j<=n;j++)

if (i>j) F[i][j]=F[i-1][j];

else F[i][j]=F[i-1][j]+F[i][j-i];

cout<<F[n][m];

}

**Cải tiến mảng một chiều:**

Tại mỗi bước ta chỉ cần lưu lại một dòng của bảng F (F là mảng 2 chiều), bằng mảng một chiều, sau đó dùng mảng đó tính lại chính nó để sau khi tính, mảng một chièu sẽ lưu giá trị của bảng F ở dòng kế tiếp. đoạn chương trình cài đặt dưới F được thay bằng S.

int S[1000];

memset(S,0,sizeof(S));

S[0]=1;

for (int i=1; i<=n;i++)

for(int j=i;j<=n;j++)

S[j]=S[j]+S[j-i];

cout<<S[n]<<'\n';

**7/ Ví dụ 7: Dãy con có tổng bằng M**

Cho N  số nguyên dương tạo thành dãy A={A1, A2, ..., AN}. Tìm ra một dãy con của dãy A (không nhất thiết là các phần tử liên tiếp trong dãy) có tổng bằng M cho trước.

Input

* Dòng đầu tiên ghi hai số nguyên dương N và M (0<N≤200) và N (0<M≤40000)
* Các dòng tiếp theo lần lượt ghi N số hạng của dãy A là các số A1, A2, ..., AN (0<Ai≤200)

Output

* Nếu bài toán vô nghiệm thì in ra “NO”
* Nếu bài toán có nghiệm thì in ra “YES”

|  |  |
| --- | --- |
| **Input** | **Output** |
| 5 14  2 4 1 11 8 | YES |

Đặt F[i,j]=1 nếu *có thể tạo ra tổng j từ một dãy con của dãy gồm các phần tử A1,A2,..Ai*. Ngược lại thì F[i,j]=0. Nếu F[n,m]=1 thì đáp án của bài toán trên là “có”.

Ta có thể tính F[i,j] theo công thức:

Khi xét đến phần tử thứ i:

+ Nếu j<a[i] thì chấp nhận cách tính tổng j bằng tổng của i-1 số trước đó

**F[i,j]=F[i-1,j];**

+ Nếu j>=a[i]:

* + Nếu trước đó đã có cách tính j bằng tổng trong các số a1,a2,…,ai-1thì không xét nữa mà chấp nhận cách tính đó: **nếu F[i–1,j]==1 thì F[i,j]=1;**
  + Nếu chưa có cách tính j bằng tổng của i-1 số trước đó (F[i-1,j]==0) thì xét số j-a[i] đã tính được chưa, nếu số j-a[i] đã tính được, **tức F[i-1,j-a[i]]==1 thì F[i,j]=1;**

**F[i,0]=1; hiển nhiên //vì 0 bằng tổng của 0 phần tử.**

**F[i,j]=F[i-1,j]=1;**

**Nếu (j>=a[i] and F[i][j]==0 and F[i-1][j-a[i]]==1) F[i][j]=1,luu[j]=j-a[i];**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| i=0 | J=0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
| 1 | 1 |  | 1 |  |  |  |  |  |  |  |  |  |  |  |  |
| 2 | 1 |  | 1 |  | 1 |  | 1 |  |  |  |  |  |  |  |  |
| 3 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |  |  |  |  |  |  |  |
| 4 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |  |  |  | 1 | 1 | 1 | 1 |
| 5 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Luu[] | 0 | 0 | 0 | 2 | 0 | 4 | 2 | 6 | 0 | 1 | 2 | 0 | 1 | 2 | 3 |

**Cài đặt**

Nếu áp dụng luôn công thức trên thì ta cần dùng bảng phương án hai chiều. Với mỗi i=1..n, duyệt j=m..1, gán F[i,j]=1 nếu F[i-1,j]==1; . Nếu (F[i][j]==0 and F[i-1][j-a[i]]==1) thì gán F[i,j]=1

Nếu muốn truy vết giá trị trị thì khi gán F[i,j]=1 ta dùng thêm màng luu[j]=j-a[i];

*int luu[1000];*

*memset(F,0,sizeof(F));*

*for (int i=0; i<=n;i++) F[i][0]=1;*

*//Tao bang PA*

*for (int i=1; i<=n;i++)*

*for(int j=m;j>=1;j--){*

*F[i][j]= F[i-1][j];*

*if (j>=a[i] and F[i][j]==0 and F[i-1][j-a[i]]==1) {*

*F[i][j]=1;luu[j]=j-a[i];}*

*}*

**Truy vết xuất kết quả từ mảng lưu (lưu ý có nhiều dãy con, QHD chỉ đưa ra 1 dãy)**

if (F[n*][m]!=1) cout<<"No"<<endl;*

*else {*

*cout<<"YES"<<endl;*

*int j= m;*

*while (j>0){*

*cout<<j-luu[j]<<" ";*

*j=luu[j];*

*}*

*}*

**Cải tiến mảng 2 chiều thành mảng một chiều:**

Ta có thể nhận xét rằng để tính dòng thứ i, ta chỉ cần dòng i–1. Bảng phương án khi đó chỉ cần 1 mảng 1 chiều S[0..M] và được tính như sau:

memset(luu,0,sizeof(luu));

memset(S,0,sizeof(S));

S[0]=1;

for (int i=1; i<=n;i++) {

for(int j=m;j>=a[i];j--)

if (S[j]==0 and S[j-a[i]]==1) { S[j]=1;luu[j]=j-a[i];}

Dễ thấy chi phí không gian của cách cài đặt trên là O(m), chi phí thời gian là O(nm), với m là tổng của n số. Hãy tự kiểm tra xem tại sao vòng for thứ 2 lại là for downto chứ không phải là for to.

**Truy vết xuất kết quả từ mảng lưu, hoàn toàn giống mảng 2 chiều**

***Bài này có thể mở rộng để tìm độ dài dãy con có tổng bằng M, dài nhất hoặc ngắn nhất***

Cho N  số nguyên dương tạo thành dãy A={A1, A2, ..., AN}. Tìm ra một dãy con của dãy A (không nhất thiết là các phần tử liên tiếp trong dãy) có tổng bằng M cho trước có độ dài lớn nhất hoặc nhỏ nhất

Input

* Dòng đầu tiên ghi hai số nguyên dương N và M (0<N≤200) và N (0<M≤40000)
* Các dòng tiếp theo lần lượt ghi N số hạng của dãy A là các số A1, A2, ..., AN (0<Ai≤200)

Output

* Nếu bài toán vô nghiệm thì in ra “NO”
* Nếu bài toán có nghiệm thì in ra “YES” và dòng thứ 2 ghi độ dài

|  |  |
| --- | --- |
| **Input** | **Output** |
| 5 15  2 4 1 11 8 | YES  4 //dài nhất |

Làm tương tự, nhưng cần sắp xếp mảng A tăng dần để tìm độ dài lớn nhất hoặc sắp xếp giảm dần để tìm độ lớn nhỏ nhất.

Trong test trên độ dài ngắn nhất là 2 (4+11=15) và độ dài lớn nhất là 4 (2+4+1+8=15)

Đoạn đầu chương trình hoàn toán tương tự bài trên, chỉ thêm biến đếm độ dài d, phần truy vết:

if (S[m]!=1) cout<<"No"<<endl;

else {

int kq[n+1];

cout<<"YES"<<endl;

int j= m,d=0;

while (j>0){

kq[++d]=j-luu[j];//cout<<j-luu[j]<<" ";

j=luu[j];

}

cout<<"Do dai = "<<d<<endl;

for (int k=d; k>=1;k--) cout<<kq[k]<<" ";

}

}

**8/ Ví dụ 8: Dãy con có tổng chia hết cho K**

Gọi F[i,j] là độ dài lớn nhất của dãy con có tổng là j và chọn các phần tử từ a1,a2,…,ai.

**Giới hạn:** 1 ≤ N ≤ 1000; 1 ≤ K ≤ 100; |ai|≤ 109

10 3

3 2 5 7 9 6 12 7 11 15

Dãy dài nhất có 9 phần tử là 3, 5, 7, 9, 6, 12, 7, 11, 153,5,7,9,6,12,7,11,15

Không chọn tổng là j vì: 1/ j mod k=j cũng bằng (j+k)mod j=j. 2/ Nếu để tổng là j thì khi xét đến F[i,j-a[i]], j-a[i] sẽ nhỏ hơn 0.

Khi không chọn phần tử nào thì tổng các phần tử bằng 0; ***F[0][0]=0;***

Khi không chọn phần tử nào mà có tổng bằng j, thì kết quả là vô nghiệm, nên cho giá trị bằng là vô cùng, vì tìm Max nên giá trị ứng với âm vô cùng ***F[0][j]= - vô cùng;***

Khi xét đến phần tử thứ i, thì ta lấy phần dư của a[i] mod k để xét thay cho a[i] (nếu để a[i] khi chọn giá trị trước sẽ ra giá trị âm).

**TH1:** Nếu không chọn phần tử a[i] thì độ dài lớn nhất khi xét đến phần tử ai sẽ bằng độ dài dãy con lớn nhất khi xét đến pần tử ai-1.

**F[i,j]=F[i-1, j]**

**TH2:** Nếu chọn phần tử a[i] thì độ dài lớn nhất sẽ bằng độ dài lớn nhất của phần thừa

***j+k-a[i]*** hay ***(j-k+a[i]) mod k*** sau đó cộng cho 1 (tăng độ dài thêm 1, và vì j<k cần chia dư giá trị này cho k)

**F[i,j]=F[i-1, (j+k-a[i]) mod k]+1**

Để đạt được độ dài của dãy con lớn nhất thì lấy giá trị lớn nhất trong 2 trường hợp trên.

**F[i,j]=max(F[i,j]=F[i-1, j] , F[i-1, (j+k-a[i]) mod k]+1)**

Kết quả cuối cùng là F[n,k]

For i=1..n

For j=1..K

F[i,j]= Max(F[i-,j] , F[i-1, (j+k -a[i]) mod k ] +1 )

cout<<F[n,k];

**=====================================================**

[Quy hoạch động](https://vnspoj.github.io/category/dp) như sau:

Trước hết nhận xét ta có thể mod các số của dãy đi k mà không ảnh hưởng đến bản chất bài toán. Gọi F[i][j] là **độ dài dài nhất** của dãy con xét đến vị trí thứ i sao cho tổng của nó mod k bằng j. Kết quả bài toán là F[n][0].

Tính F[i][j] : Có hai khả năng cho vị trí thứ i : Thứ nhất là không chọn, thì nó là F[i-1][j], nếu chọn thì nó sẽ bằng F[i-1][t]+1 với t là module sao cho khi module của a[i] với t cho k được j. Dễ tính t = (j-a[i]+k)%k.

Do đó F[i][j] = max(F[i-1][j], F[i-1][(j-a[i]+k)%k]+1).

**9/ Ví dụ 9: Xâu con đối xứng dài nhất**