Programação Dinâmica

Soma de Subconjunto[1]

Problema : Dado um conjunto de n números a_i que a soma total é igual a M, e para algum K≤M, se existe um subconjunto dos números tais que a soma desse subconjunto dá exatamente K?

Solução 1:

Vamos usar uma tabela unidimensional m[0..M], m[b] indica se b pode ser alcançado.

```
for(i=0;i<=M;i++) m[i] = 0;

m[0]=1;

for(i=0;i<n;i++)

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

   m[j] |= m[j-a[i]]
```

Observações: A idéia original é usar uma matriz bidimensional, onde cada coluna depende apenas da coluna anterior. Mas utilizando um truque de programação (sobreposição) podemos utilizar apenas um vetor unidimensional. Mas precisamos escrever o loop j na ordem reversa para evitar confusões.

Mais do mesmo

Existem várias variações do problema da soma de subconjuntos

Doces para duas crianças[1]: Os valores dos a[i]'s representam os valores dos doces. Nós queremos dividir igualmente os doces entre as duas crianças ou da maneira mais justa. Agora o problema não é encontrar um subconjunto com soma fixa K. Nós queremos encontrar um certo K mais próximo de M/2.

http://br.spoj.pl/problems/TESOURO/

http://www.spoj.pl/problems/FCANDY/

Soma de subconjunto com múltiplos suprimentos[1]: Cada a pode ser usado quantas vezes você precise na soma para alcançar o valor K? Talvez você demore ou não para ver a solução. A solução é apenas inverter a direção do loop j na solução j.

Troco (coin change)[1]: Agora a_i's representam moedas, você quer passar um troco de exatamente K. Talvez existam várias maneiras de você fazer isso, então você quer minimizar (ou maximiza) o número de moedas que você precisa usar. A estrutura da solução não precisa ser alterada, nós precisamos apenas mudar o significado do vetor m. Agora m[b] não será 0 ou 1, será exatamente o número mínimo de moedas que você precisa para alcançar b.

https://br.spoj.pl/problems/MOEDAS/

http://www.spoj.pl/problems/NOCHANGE/

Doces para três crianças[1]: Nós queremos dividir os doces mais equilibradamente o possível entre as três crianças. A questão é o que representa "mais equilibradamente o possível". A solução pode ser feita de diferentes maneiras, mas a estrutura da solução são quase as mesmas. Use um vetor bidimensional m [][], m[b][c] indica se podemos ou não dividir os doces de maneira que b doces vão para a primeira criança, c doces para a segunda criança e o restante vai para a terceira criança.

Contando o troco (couting change)[2]: Os a_i's continuam representando moedas, agora você quer saber quantas maneiras você pode passar o troco K. O número de maneiras de trocar uma quantidade A usando N tipos de moedas é igual a:

1. O número de maneiras de trocar a quantidade A usando todas as moedas anteriores, +

2. O número de maneiras de trocar a quantidade A-D usando todos os N tipos de moedas, onde D é o valor da n enésima moeda.

A árvore de recursão do processo vai reduzir gradualmente o valor de A, então podemos usar estas regras, para determinar o número de maneiras de trocar um certo valor

1. Se A == 0, então vamos contar 1 maneiras.

2. Se A <=0, então vamos contar 0 maneiras

3. Se N tipos de moedas == 0, então vamos contar 0 maneiras.

```
int coin[] = {1,5,10,25,50};

int count[MAX_MONEY+1];

memset(count,0,sizeof(count));

count[0] = 1;

for (int i=0;i<COIN_TYPES;++i)

      for (int j=coin[i];j<=MAX_MONEY;++j)

           count[j] += count[j-coin[i]];
```

Outras variações

http://br.spoj.pl/problems/SEQUENCI/

Maior Subseqüência Comum (Longest Commom Subsequence)(LCS)[3]

Problema 2: Dado duas seqüências s1[0..M-1] e s2[0..N-1] qual é a maior subseqüência comum delas?

Uma subseqüência de uma determinada seqüência é apenas a seqüência dada com zero ou mais elementos omitidos. Formalmente, dada uma seqüência X = < x1; … ; xm >, outra seqüência Z = < z1; … ; zk > _e uma subseqüência de X se existe uma seqüência estritamente crescente < i1; … ; ik > de índices de X tais que para todo j= 1,2,...,k temos que $x_{ij} = z_j$. Por exemplo, Z=< B;C;D;B > é uma subseqüência de X=< A;B;C;B;D; A;B > com seqüência de índice correspondente < 2; 3; 5; 7 >. Dada duas seqüência X e y, dizemos que uma seqüência Z é uma subseqüência comum de X e Y se Z _e subseqüência de X e Y ao mesmo tempo.

Por exemplo, se X=< A;B;C;B;D; A;B > E Y=< B;D;C; A;B;A >, a seqüência < B;C;A > é uma subseqüência comum das seqüência X e Y e < B;C;B;A > _e a subseqüência comum mais longa.

No problema da subseqüência comum mais longa, temos duas seqüências X = < x1; … ; xm > e Y = < y1; … ; yn >, e desejamos encontrar uma subseqüência comum de comprimento máximo(longest comum subsequence(LCS) de X e Y.

Etapa 1: Caracterizar um subseqüência comum mais longa
Para encontrar uma LCS de X = < x1; … ; xm > e Y = < y1; … ; yn>
temos que analisar dois subproblemas:

1. se xm = yn, devemos encontrar uma LCS de X_{m-1} e Y_{n-1}. A anexação de xm = yn a essa LCS produz uma LCS de X e Y.
2. se xm != yn, então temos que resolver dois subproblemas: encontrar uma LCS de X_{m-1} e Y e encontrar uma LCS de X e Y_{n-1}. A maior entre as duas LCS é uma LCS de X e Y.

Podemos ver que muitos subproblemas compartilham subproblemas. A sobreposição de problemas pode ser vista de imediato.

Etapa 2: Definição recursiva
Vamos definir c[i,j] como o comprimento de uma LCS das seqüência Xi e Yi. Se i=0 ou j=0, uma das seqüências tem o comprimento 0; assim, a LCS tem comprimento 0. A subestrutura ótima do problema da LCS fornece a fórmula recursiva

$$c[i,j] = \begin{cases} 0 & \text{se i=0 ou j=0} \\ c[i-1,j-1]+1 & \text{se i,j} > 0 \text{ e} x_i = y_j \\ max(c[i,j-1], c[i-1,j]) & \text{se i,j} > 0 \text{ e} x_i \neq y_j \end{cases}$$

```
int max(int a,int b){
     return a>b?a:b;
}

m=strlen(X);
n=strlen(Y);

for (i=1;i<=m;i++) c[i][0]=0;
for (j=0;j<=n;j++) c[0][j]=0;

for (i=1;i<= m;i++)
for (j=1;j<= n;j++)
     if (X[i-1]==Y[j-1]) c[i][j]=c[i-1][j-1]+1;
     else c[i][j] = max (c[i-1][j], c[i][j-1]);
```

	x_i	A	B	C	B	D	A	B
y_i	0	0	0	0	0	0	0	0
B	0							
D	0							
C	0							
A	0							
B	0							
A	0							

	x_i	A	B	C	B	D	A	B
y_i	0	0	0	0	0	0	0	0
B	0	0	1	1	1	1	1	1
D	0							
C	0							
A	0							
B	0							
A	0							

i=1

	x_i	A	B	C	B	D	A	B
y_i	0	0	0	0	0	0	0	0
B	0	0	1	1	1	1	1	1
D	0	0	1	1	1	2	2	2
C	0	0	1	2	2	2	2	2
A	0							
B	0							
A	0							

i=2

	x_i	A	B	C	B	D	A	B
y_i	0	0	0	0	0	0	0	0
B	0	0	1	1	1	1	1	1
D	0	0	1	1	1	2	2	2
C	0	0	1	2	2	2	2	2
A	0	1	1	2	2	2	3	3
B	0							
A	0							

i=3

	x_i	A	B	C	B	D	A	B
y_i	0	0	0	0	0	0	0	0
B	0	0	1	1	1	1	1	1
D	0	0	1	1	1	2	2	2
C	0	0	1	2	2	2	2	2
A	0	1	1	2	2	2	3	3
B	0	1	2	2	3	3	3	4
A	0							

i=4

	x_i	A	B	C	B	D	A	B
y_i	0	0	0	0	0	0	0	0
B	0	0	1	1	1	1	1	1
D	0	0	1	1	1	2	2	2
C	0	0	1	2	2	2	2	2
A	0	1	1	2	2	2	3	3
B	0	1	2	2	3	3	3	4
A	0	1	2	2	3	3	4	4

i=5

```
Print_LCS(b, X, i, j)
1: if i = 0 ou j = 0 then
2:   pare
3: if b[i, j] = " ↖ " then
4:   Print_LCS(b, X, i − 1, j − 1)
5:   escreva x_i
6: else
7:   if b[i, j] = " ↑ " then
8:     Print_LCS(b, X, i − 1, j)
9:   else
10:    Print_LCS(b, X, i, j − 1)

Chamada Inicial: Print_LCS(b, X, m, n)
```

Saída

	x_i	A	B	C	B	D	A	B
y_i	0	0	0	0	0	0	0	0
B	0	0	1	1	1	1	1	1
D	0	0	1	1	1	2	2	2
C	0	0	1	2	2	2	2	2
A	0	1	1	2	2	2	3	3
B	0	1	2	2	3	3	3	4
A	0	1	2	2	3	3	4	4

Saída A

	x_i	A	B	C	B	D	A	B
y_i	0	0	0	0	0	0	0	0
B	0	0	1	1	1	1	1	1
D	0	0	1	1	1	2	2	2
C	0	0	1	2	2	2	2	2
A	0	1	1	2	2	2	3	3
B	0	1	2	2	3	3	3	4
A	0	1	2	2	3	3	4	4

Saída AB

	x_i	A	B	C	B	D	A	B
y_i	0	0	0	0	0	0	0	0
B	0	0	1	1	1	1	1	1
D	0	0	1	1	1	2	2	2
C	0	0	1	2	2	2	2	2
A	0	1	1	2	2	2	3	3
B	0	1	2	2	3	3	3	4
A	0	1	2	2	3	3	4	4

Problema relacionado: https://br.spoj.pl/problems/PARQUE/

Alfândega

Nome do arquivo fonte: `alfa.c`, `alfa.cpp`, *ou* `alfa.pas`

Matheus, Bruno e Ricardo são responsáveis pelo setor de pesquisa da Indústria de Obras Intermináveis (IOI) e viajam juntos constantemente para outros países a fim de pesquisar diferentes métodos, equipamentos e matérias primas para suas obras faraônicas. Além disso, eles atuam no mercado de importação informal de produtos eletrônicos, trazendo em suas viagens equipamentos que seus amigos e colegas de trabalho pedem.

Antes de viajar eles elaboram uma lista de N produtos que devem ser comprados, cada um deles com um preço P_i, em Dinheiro Estrangeiro (DE$). Ao chegarem no Brasil eles devem respeitar a cota de importação de Q Dinheiros Estrangeiros por pessoa. Cada um que exceder a cota é obrigado a pagar uma taxa de importação de A porcento sobre o valor que exceder Q. Tal taxa deve ser paga em Dinheiro Estrangeiro.

Como os três sempre viajam juntos, notaram que se distribuirem os produtos de maneira adequada podem reduzir a quantidade de imposto total que devem pagar. Determinar tal combinação é uma tarefa muito complicada para eles e, por conta disso, pediram a sua ajuda.

Tarefa

Escreva um programa que, dados os valores dos aparelhos comprados, a franquia individual de importação, e a alíquota do imposto de importação, determina qual é o imposto mínimo total que Matheus, Bruno e Ricardo devem pagar.

Entrada

A entrada contém um único conjunto de testes, que deve ser lido do *dispositivo de entrada padrão* (normalmente o teclado). A primeira linha contém um inteiro N $(1 \leq N \leq 100)$, que representa a quantidade de produtos comprados no exterior.

A linha seguinte contém dois inteiros, Q e A, $(1 \leq Q \leq 500, 1 \leq A \leq 200)$, que representam a cota de importação, em Dinheiro Estrangeiro, e a alíquota de importação, em forma de porcentagem.

As N linhas seguintes contêm cada uma um inteiro P_i $(1 \leq P_i \leq Q)$, que representa o preço do i-ésimo produto em DE$.

Saída

Seu programa deve imprimir, na *saída padrão*, uma única linha, contendo o menor valor possível do imposto a ser pago por Matheus, Bruno e Ricardo, em DE$, com duas casas decimais.

Entrada	**Entrada**	**Entrada**
6 9 20 9 6 3 3 3 3	7 50 30 20 20 20 20 20 20	4 10 1 10 9 8 7
Saída	**Saída**	**Saída**
0.00	3.00	0.05

Referências:

[1] The *Hitchhiker's Guide* to the *Programming* Contests(Guia dos Mochileiros para Competições de Programação), Entre em pânico.

[2] **Arefin,Ahmed Shamsul, Art of Programming Contest (Special edition for UVa Online Judge users), 2 edição, Gyankosh Prokashoni, Bangladesh**

[3] **Cormen**, **Thomas H**.; et al. Introduction to Algorithms. MIT Press, 2001