Programação Dinâmica 2D
Professor Wladimir

Pedido de Desculpas

Cuca quer escrever um pedido de desculpas no cartão da floricultura, usando frases coletadas da internet. O objetivo é escolher um subconjunto dessas frases que caibam no cartão (limitado por número de caracteres) e que maximize o número de vezes que a palavra ”desculpe”apareça.

Entrada

- C (capacidade do cartão em caracteres)
- F (número de frases coletadas)
- w[] : vetor do número de caracteres da frase.
- v[] : vetor do número de vezes que a palavra “desculpe” aparece nessa frase.

Solução

A solução apresentada corresponde à implementação clássica do algoritmo de **programação dinâmica para o problema da mochila 0/1**, que resolve o problema por meio de **etapas (ou estágios)**, onde cada estágio depende diretamente do anterior.

Matriz de Estados

Seja $m[i][j]$ a matriz onde:

- i representa o número de frases consideradas até o momento.
- j representa a quantidade de caracteres disponíveis no cartão.
- $m[i][j]$ armazena o número máximo de ocorrências da palavra `''desculpe''` que podem ser obtidas utilizando até as i primeiras frases, sem exceder j caracteres.

Etapa 1: Inicialização

```
for(i = 0; i <= N; i++) m[i][0] = 0;
for(j = 0; j <= W; j++) m[0][j] = 0;
```

- incluímos um *objeto virtual* de índice 0 (isto é, uma frase fictícia com peso e valor nulos: $w[0] = 0$, $v[0] = 0$) para facilitar a indexação das frases no algoritmo.
- Se o cartão possui zero capacidade ($j = 0$), nenhuma frase pode ser utilizada: $m[i][0] = 0$.
- Se nenhuma frase foi considerada ($i = 0$), não é possível escrever nada: $m[0][j] = 0$.

Essa etapa define a **base da recorrência** da programação dinâmica.

Etapa 2: Construção da Solução por Estágios

```
for(i = 1; i <= N; i++) {
    for(j = 1; j <= W; j++) {
        if (j < w[i])
            m[i][j] = m[i-1][j];
        else
            m[i][j] = max(m[i-1][j], m[i-1][j - w[i]] + v[i]);
    }
}
```

- Para cada frase i e para cada capacidade j:
  - Se a frase i **não cabe** no cartão ($j < w[i]$), não podemos usá-la:

$$m[i][j] = m[i-1][j]$$

  - Se a frase i **cabe** no cartão ($j \geq w[i]$), temos duas opções:
    1. Não usar a frase:

$$m[i][j] = m[i-1][j]$$

    2. Usar a frase:

$$m[i][j] = m[i-1][j-w[i]] + v[i]$$

  - Tomamos o máximo entre as duas opções:

$$m[i][j] = \max(m[i-1][j], m[i-1][j-w[i]] + v[i])$$

Código

```
//SPOJ submission 3499824 (C) plaintext list. Status: AC, problem DESCULPA, contest
↪  SPOJBR. By wladimir (Wladimir Araujo Tavares[UFC]), 2010-04-13 05:37:38.
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int N,W;
int m[51][1001];
int w[51];
int v[51];
int i,j,teste=1;
int main(){
    while( scanf("%d %d",&W,&N) ){
            if( N==W && W==0) break;
            for(i=1;i<=N;i++){
              scanf("%d %d",&w[i],&v[i]);
            }
            for(i=0;i<=N;i++) m[i][0] = 0;
            for(j=0;j<=W;j++) m[0][j] = 0;
            for(i=1;i<=N;i++){
             for(j=1;j<=W;j++){
                 if( j < w[i] )
                   m[i][j] = m[i-1][j];
                 else
                   m[i][j] = m[i-1][j]  > m[i-1][j-w[i]] + v[i] ? m[i-1][j] :
                   ↪  m[i-1][j-w[i]] + v[i];
              }
            }
            printf("Teste %d\n",teste++);
            printf("%d\n\n",m[N][W]);
    }
    return 0;
}
```

Referência:

- Pedido de Desculpas https://judge.beecrowd.com/pt/problems/view/2299

Subset Sum

Os primos João e José encontraram um mapa de um tesouro escondido e, ao seguirem caminhos diferentes, encontraram jóias antes de chegarem juntos a uma arca com mais objetos de valor. Eles decidiram dividir o tesouro da seguinte forma:

Cada primo fica com os itens que encontrou antes da arca;

O conteúdo da arca deve ser dividido de modo que o valor total do tesouro (itens pessoais + itens da arca) de cada primo seja o mesmo.

Objetivo do Problema

Os valores X e Y, correspondentes aos valores dos itens encontrados individualmente por João e José;

Uma lista de N valores dos objetos encontrados na arca.

Determine se é possível dividir os objetos da arca entre os dois de forma que a soma total de cada um fique igual.

Saída

- Imprima “S” se a divisão for possível;
- Imprima “N” caso contrário.

Solução

Sejam:

- X: valor total dos objetos encontrados por João antes da arca.
- Y: valor total dos objetos encontrados por José antes da arca.
- $v_1, v_2, \ldots, v_N$: valores dos N objetos dentro da arca.
- $S = \sum_{i=1}^{N} v_i$: soma total dos valores dos objetos da arca.

Objetivo

Nosso objetivo é determinar se existe um subconjunto dos valores da arca cuja soma seja:

$$Z = \frac{X + Y + S}{2} - X$$

Ou seja, o quanto João deve obter da arca para que ambos terminem com o mesmo valor final.

Definição da Programação Dinâmica

Seja $\text{dp}[i][j]$ uma matriz booleana tal que:

$$\text{dp}[i][j] = \begin{cases} \text{true} & \text{se é possível obter soma } j \text{ usando os } i \text{ primeiros objetos da arca;} \\ \text{false} & \text{caso contrário.} \end{cases}$$

Inicialização

$$\text{dp}[0][0] = \text{true}$$

Com 0 objetos, só é possível obter soma 0.

Para $j > 0$:

$$\text{dp}[0][j] = \text{false}$$

Relação de Recorrência

Para cada $i = 1$ até N, e para cada soma $j = 0$ até Z, temos:

$$\text{dp}[i][j] = \text{dp}[i-1][j] \vee \begin{cases} \text{dp}[i-1][j-v_i], & \text{se } j \geq v_i \\ \text{false}, & \text{caso contrário} \end{cases}$$

Essa recorrência representa duas possibilidades:

- Não usamos o i-ésimo objeto: herdamos o valor da linha anterior para mesma soma j.
- Usamos o i-ésimo objeto: verificamos se era possível fazer $j - v_i$ com os $i - 1$ primeiros objetos.

Reduzindo a memória

A versão otimizada da solução utiliza um vetor unidimensional para simular a recorrência do problema da soma de subconjuntos. O objetivo continua sendo verificar se é possível dividir o tesouro de forma justa entre os primos, considerando os valores que cada um já encontrou antes de chegar à arca.

1. Inicializar um vetor booleano `soma` com:

$$\texttt{soma}[0] \leftarrow \text{true}, \quad \texttt{soma}[j] \leftarrow \text{false para } j > 0$$

2. Para cada objeto v_i da arca, atualizar o vetor de somas possíveis:
   - Percorrer de trás para frente (de T até 0):

$$\text{Se } \texttt{soma}[j] = \text{true}, \text{ então } \texttt{soma}[j + v_i] \leftarrow \text{true}$$

3. Ao final, verificar se é possível formar Z:

$$\text{Se } \texttt{soma}[Z] = \text{true} \Rightarrow \text{resposta é 'S'; caso contrário, 'N'}$$

Vantagens da Otimização: Ao usar um vetor unidimensional, economizamos memória. A complexidade de tempo continua sendo $O(N \cdot T)$, mas a complexidade de espaço reduz de $O(N \cdot T)$ para $O(T)$, onde T é a soma total dos valores.

Observação: A atualização do vetor `soma` deve ser feita de trás para frente para evitar a contagem múltipla de um mesmo item no mesmo estágio da simulação.

Código

```c
#include <stdio.h>
#include <string.h>
#define MAX 10000
int v[101];
char soma[10001];

int main(){
    int x,y,n;
    int total;
    bool podeser;
    int teste=1;
    int i,j;

    while(1){
        scanf("%d %d %d",&x,&y,&n);

        if(x+y+n==0) break;

        total = x+y;
        for(i=1;i<=n;i++){
            scanf("%d",&v[i]);
            total += v[i];
        }

        podeser = true;

        if(total%2==0){

            memset(soma,0,sizeof(soma));
            soma[0]=1;
            for(j=1;j<=n;j++){
                for(i=total;i>=0;i--){
                    if(soma[i]==1){
                        soma[i+v[j]]=1;
                    }
                }
            }
            int metade = total/2;
            if(soma[ metade-x  ] ==0 )
              podeser = false;
        }else{
            podeser= false;
        }

        printf("Teste %d\n",teste++);
        if(podeser) printf("S\n\n");
        else printf("N\n\n");
    }

}
```

Referência

- Caça ao tesouro https://judge.beecrowd.com/pt/problems/view/2228