Desafios de Programação
Problema da Mochila

Wladimir Araújo Tavares [1]

[1]Universidade Federal do Ceará - Campus de Quixadá

8 de maio de 2018

Problema da Mochila

1. Inicialmente, temos n objetos tal que cada objeto i possui um peso p_i e um valor v_i.
2. Sua mochila possui uma capacidade máxima W
3. Seu objetivo é escolher um subconjunto de objetos cujo peso total não ultrapasse a capacidade da mochila W obtendo o valor máximo.

Problema da Mochila

- $M_{i,j}$ valor máximo obtido considerando os objetos $[1..i]$ com a capacidade da mochila j.
- Para definir a estrutura recursiva do nosso subproblema temos que considerar dois casos:
  - Se o objeto i está na solução. $M[i-1][j-p[i]] + v[i]$
  - Se o objeto i não está na solução. $M[i-1][j]$

$$M[i][j] = \begin{cases} max(M[i-1][j], M[i-1][j-p[i]] + v[i]) & j \geq p[i] \\ M[i-1][j] & \text{caso contrário} \end{cases}$$

- Caso Base: $M[i][0] = 0$ e $M[0][j] = 0$

Top-down

```
int main(){
    int n, W;
    cin >> W >> n;
    for(int i = 1; i <= n; i++){
            cin >> peso[i] >> valor[i];
    }
    for(int i = 1; i <= n; i++)
            for(int j = 1; j <= W; j++)
                    dp[i][j] = -1;
    cout << knapsack(n, W) << endl;
}
```

Top-down

```
#include <bits/stdc++.h>

using namespace std;

#define MAXN 2010
#define MAXS 2010

int n, valor[MAXN], peso[MAXN], dp[MAXN][MAXS];

int knapsack(int i, int W){
        if(dp[i][W]>=0) return dp[i][W];
        if(i == 0 || W==0) return 0;
        int nao_coloca = knapsack(i-1, W);
        dp[i][W] = nao_coloca;
        if(peso[i]<=W){
                int coloca = knapsack(i-1, W- peso[i]) + valor[i]
                dp[i][W] = max( nao_coloca, coloca);
        }
        return dp[i][W];
}
```

Bottom-up

```
  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("%d\n\n",m[N][W]);
}
```

Problema do troco

- Dado um conjunto de moedas com valores $v_1, v_2, \ldots, v_n$, queremos saber se um valor M pode ser obtido.
- Por exemplo, se as moedas disponíveis são 2,5,7, o valor 19 pode ser obtido com duas moedas de 5, uma moeda de 2 e uma moeda de 7.
- Estrutura recursiva

$$DP(x) = DP(x-2)||DP(x-5)||DP(x-7) \quad (1)$$

- Casos Bases: DP(0) = true, DP(x) = false, x $<$ 0.

Top-down

```
vector <int> dp;

bool solve(int x, vector<int> &c){

   if(x==0) return true;

   if(x<0) return false;

   if(dp[x]>=0) return dp[x];

   for(int i=0;i<c.size();i++)
      if(solve(x-c[i], c))
               return dp[x-c[i]]= true;
   return dp[x]=false;
}
```

Top-down

```
int main(){
  int n, M;

  vector <int> c;

  cin >> n >> M;

  c.resize(n);

  dp.assign(M+1, -1);

  for(int i = 0; i < n; i++)
    cin >> c[i];

  cout << (solve(M, c) ? "S" : "N") ;

}
```

Bottom-up

```
vector <int> v;
vector <bool> val;
cin >> n >> m;
v.resize(n);

for(int i = 0; i < n; i++){
  cin >> v[i];
}

val.assign(m+1, false);
val[0] = true;
for(int i = 0; i < n; i++){
  for(int j = v[i]; j <= m; j++){
    if( val[j - v[i]] ) val[j] = true;
  }
}

cout << (val[m] ? "S" : "N") << endl;
```

Problema do troco

- Dado um conjunto de moedas com valores $v_1, v_2, \ldots, v_n$, queremos saber se um valor M pode ser obtido utilizando menos que 10 moedas.
- Por exemplo, se as moedas disponíveis são 2,5,7, o valor 19 pode ser obtido com duas moedas de 5, uma moeda de 2 e uma moeda de 7 utilizamos] apenas 4 moedas.
- Estrutura recursiva DP(x) = número de moedas utilizadas para obter o troco x.

$$DP(x) = min(DP(x-2), DP(x-5), DP(x-7)) + 1 \qquad (2)$$

- Casos Bases: DP(0) = 0, DP(x < 0) = 11.

Bottom-up

```
val.assign(m+1, 11);
val[0] = 0;

for(int i = 0; i < n; i++){
    for(int j = v[i]; j <= m; j++){
      if( val[j] > val[j - v[i]] + 1 ){
        val[j] = val[j-v[i]] + 1;

      }
    }
}
cout << (val[m] < 10 ? "S" : "N") << endl;
```

k-Sum

1. Dados n inteiros positivos distintos, inteiro k (k <= n) e um número alvo.
2. Encontre os k números tal que a soma deles é igual a um certo valor S. Calcule quantas maneiras existem?
3. Por exemplo, Dado [1,2,3,4], k = 2, S = 5.
4. Temos duas maneiras: 1+4, 2+3

kSum

- Subproblemas
  $F[i, j, k]$ = maneiras podemos obter a soma k utilizando j números considerando $v[1 \ldots i]$
- Estrutura Recursiva

$$F[i, j, k] = F[i-1][j][k] + \sum_{k - v[i] > 0 \text{ and } j > 0} F[i-1][j-1][k-v[i]]$$

- Caso Base:

$$F[0][j][k] = \begin{cases} 1, & k = 0 \\ 0, & k \neq 0 \end{cases}$$

Bottom-up

```
vector <int> v;
cin >> n >> l >> s;
v.resize(n+1);
for(int i = 1; i <=n; i++) cin >> v[i];
for(int j = 0; j <= l; j++){
  for(int k = 0; k <= s; k++){
    F[0][j][k] = k == 0 ? 1 : 0;
  }
}
for(int i = 1; i <= n; i++){
  for(int j = 0; j <= l; j++){
    for(int k = 0; k <= s; k++){
      F[i][j][k] = F[i-1][j][k];
      if( j > 0 && k - v[i] >= 0){
        F[i][j][k] += F[i-1][j-1][k-v[i]];
      }

    }
  }
}
cout << F[n][l][s];
```

Problema da Mochila 3D

1. Um mergulhador usa um cilindro com dois recipientes: um com oxigênio e outro com nitrogênio. Dependendo do tempo que ele quer ficar debaixo d'água e da profundidade do mergulho, o mergulhador precisa de uma quantidades de oxigênio e nitrogênio. O mergulhador tem ao seu dispor um certo número de cilindros.
2. Cada cilindro pode ser descrito pelo seu peso e um volume de oxigênio e nitrogênio.
3. Dada a quantidade específica de oxigênio e nitrogênio que o mergulhador precisa, encontre o peso total mínimo de cilindros que ele deve levar para completar a tarefa?
4. `http://www.spoj.com/problems/SCUBADIV/`

Exercícios

1. Isso aqui é que não dá! `http://www.codcad.com/problem/137`
2. Em Busca do Corpo Perfeito `http://www.codcad.com/problem/67`
3. SCUBADIV `http://www.spoj.com/problems/SCUBADIV/`
4. Caça ao tesourohttp://br.spoj.com/problems/TESOURO/