

Paradigmas de Resolução de Problemas

Programação Dinâmica: Mochila Binária

Prof. Edson Alves - UnB/FGA

2020

1. Definição
2. Solução do problema da mochila binária
3. Aplicações

Definição

Problema da Mochila Binária

Considere um conjunto $C = \{c_1, c_2, \dots, c_N\}$, onde $c_i = (w_i, v_i)$, e seja M um inteiro positivo.

O problema da mochila binária consiste em determinar um subconjunto $S \subset \{1, 2, \dots, N\}$ de índices de C tal que

$$W = \sum_{j \in S} w_j \leq M$$

e que a soma

$$V = \sum_{j \in S} v_j$$

seja máxima.

Características do problema da mochila binária

- Os elementos do conjunto C são denominados objetos ou items
- M é o capacidade da mochila
- w_i é o peso/tamanho do objeto, o qual determina se o objeto pode ou não ser transportado na mochila, de acordo com a capacidade ainda disponível na mesma
- v_i é o valor/ganho do objeto, e a soma dos valores dos objetos selecionados deve ser maximizada
- O termo “mochila binária” remete ao fato de que, para cada um dos objetos, há duas opções: escolhê-lo ou não

Solução do problema da mochila binária

Solução do problema da mochila binária

- Como $|C| = N$, há 2^N subconjuntos de índices a serem avaliados
- Como cada subconjunto pode ser avaliado em $O(N)$, uma solução de busca completa tem complexidade $O(N2^N)$
- É possível, entretando, reduzir esta complexidade por meio de um algoritmo de programação dinâmica
- Seja $v(i, m)$ a soma máxima dos valores que pode ser obtida a partir dos primeiros i elementos de C e uma mochila com capacidade m
- São dois casos-base: o primeiro deles acontece quando não resta mais nenhum elemento a ser considerado
- Neste casos, temos $v(0, m) = 0$

Solução do problema da mochila binária

- O segundo caso-base acontece quando não há mais espaço disponível na mochila: $v(i, 0) = 0$
- Também são duas as transições possíveis:
 1. ignorar o i -ésimo elemento e considerar apenas os $i - 1$ primeiros; ou
 2. caso possar ser transportado, pegar o i -ésimo elemento e colocá-lo na mochila
- A primeira transição não modifica o estado da mochila e nem o total dos valores transportados
- Caso a mochila não consiga transportar o i -ésimo elemento, esta será a única transição possível
- Assim,

$$v(i, m) = v(i - 1, m), \quad \text{se } w_i > m$$

Solução do problema da mochila binária

- A segunda transição só é possível se $w_i \leq m$
- Caso esta condição seja atendida, a capacidade da mochila é reduzida em w_i unidades, e o total dos valores transportados é acrescido em v_i
- Assim, deve-se optar pela transição que produz o maior valor:

$$v(i, m) = \max\{ v(i-1, m), v(i-1, m-w_i) + v_i \}, \quad \text{se } w_i \leq m$$

- A solução do problema será dada por $v(N, M)$
- O número de estados distintos é $O(NM)$ e cada transição é feita em $O(1)$
- Portanto a solução de programação dinâmica para o problema do troco tem complexidade $O(NM)$

Implementação *top-down* da mochila binária

```
1 #include <bits/stdc++.h>
2
3 using namespace std;
4 using ii = pair<int, int>;
5 using ll = long long;
6
7 const int MAXN { 2010 }, MAXM { 2010 };
8
9 ll st[MAXN][MAXM];
10
11 ll dp(int i, int m, int M, const vector<ii>& cs)
12 {
13     if (i < 0)
14         return 0;
15
16     if (st[i][m] != -1)
17         return st[i][m];
18
19     auto res = dp(i - 1, m, M, cs);
20     auto [w, v] = cs[i];
21
```

Implementação *top-down* da mochila binária

```
22     if (w <= m)
23         res = max(res, dp(i - 1, m - w, M, cs) + v);
24
25     st[i][m] = res;
26     return res;
27 }
28
29 ll knapsack(int M, const vector<ii>& cs)
30 {
31     memset(st, -1, sizeof st);
32
33     return dp((int) cs.size() - 1, M, M, cs);
34 }
35
```

Recuperação dos elementos selecionados

- Uma variante comum do problema é exibir a lista dos elementos selecionados que maximizaram a soma dos valores, respeitando a capacidade da mochila
- Para recuperar os elementos escolhidos é precisa uma tabela adicional p , com as mesmas dimensões da tabela de memorização
- Se $p(i, m) = 1$, a transição escolhida por $v(i, m)$ foi a segunda, isto é, o elemento c_i foi escolhido
- Caso contrário, $p(i, m) = 0$
- Com esta informação basta recuperar os itens, usando os valores de $p(i, m)$ para rastrear os estados que levaram à solução ótima

Implementação *bottom-up* da mochila binária

```
1 #include <bits/stdc++.h>
2
3 using namespace std;
4 using ii = pair<int, int>;
5 using ll = long long;
6
7 const int MAXN { 2010 }, MAXM { 2010 };
8
9 ll st[MAXN][MAXM];
10 char ps[MAXN][MAXM];
11
12 pair<ll, vector<int>> knapsack(int M, const vector<ii>& cs)
13 {
14     int N = (int) cs.size() - 1; // Os elementos começam em 1
15
16     // Casos-base
17     for (int i = 0; i <= N; ++i)
18         st[i][0] = 0;
19
20     for (int m = 0; m <= M; ++m)
21         st[0][m] = 0;
```

Implementação *bottom-up* da mochila binária

```
22
23 // Transições
24 for (int i = 1; i <= N; ++i)
25 {
26     for (int m = 1; m <= M; ++m)
27     {
28         st[i][m] = st[i - 1][m];
29         ps[i][m] = 0;
30
31         auto [w, v] = cs[i];
32
33         if (w <= m and st[i - 1][m - w] + v > st[i][m])
34         {
35             st[i][m] = st[i - 1][m - w] + v;
36             ps[i][m] = 1;
37         }
38     }
39 }
40
```

Implementação *bottom-up* da mochila binária

```
41 // Recuperação dos elementos
42 int m = M;
43 vector<int> is;
44
45 for (int i = N; i >= 0; --i)
46 {
47     if (ps[i][m])
48     {
49         is.push_back(i);
50         m -= cs[i].first;
51     }
52 }
53
54 reverse(is.begin(), is.end());
55
56 return { st[N][M], is };
57 }
```

Aplicações

Subset sum

- Um problema comum que pode ser resolvido pela mochila binária é o *subset sum*
- Dados N inteiros positivos $\{x_1, x_2, \dots, x_N\}$ e um inteiro S , o problema consiste em determinar se é possível escolher k destes inteiros, com $1 \leq k \leq N$, de modo que a soma dos elementos escolhidos seja igual a S
- Veja que este problema pode ser reduzido a uma mochila binária
- A soma S é a capacidade da mochila
- Os pesos dos elementos c_i são dados pelos inteiros x_i
- Os valores podem ser desprezados
- Ao invés de escolher a transição de maior soma, deve-se considerar o “ou” lógico entre ambas
- O caso base $v(0, 0)$ será verdadeiro, e todos os demais casos-base serão falsos

Implementação do *subset sum*

```
1 #include <bits/stdc++.h>
2
3 using namespace std;
4
5 const int MAXN { 2010 }, MAXM { 2010 };
6
7 int st[MAXN][MAXM];
8
9 int dp(int i, int m, int M, const vector<int>& xs)
10 {
11     if (i < 0)
12         return m == 0 ? 1 : 0;
13
14     if (st[i][m] != -1)
15         return st[i][m];
16
17     auto res = dp(i - 1, m, M, xs);
18
19     if (xs[i] <= m)
20         res |= dp(i - 1, m - xs[i], M, xs);
21 }
```

Implementação do *subset sum*

```
22     st[i][m] = res;
23     return res;
24 }
25
26 bool subset_sum(int S, const vector<int>& xs)
27 {
28     memset(st, -1, sizeof st);
29
30     return dp((int) xs.size() - 1, S, S, xs);
31 }
32
```

1. **HALIM**, Steve; **HALIM**, Felix. *Competitive Programming 3*, Lulu, 2013.
2. **LAARKSONEN**, Antti. *Competitive Programmer's Handbook*, 2017.