Programação Dinâmica

Problema da Mochila

William D. Costa - RA 89239

Conceito do Problema

O problema da mochila (Knapsack Problem) é uma metáfora utilizada para ilustrar uma grande variedade de problemas de otimização.
A formulação do problema é simples, dado uma mochila com capacidade X, e uma série de objetos com peso L e valor P, qual a melhor combinação possível de L tal que a soma de P seja a maior enquanto a soma L não ultrapasse X.

Conceito da Solução

Sendo este um problema clássico de otimização combinatória e programação dinâmica, iremos utilizar esta abordagem para encontrar a solução ótima considerando como entrada os parâmetros do problema original:

```
backpack(capacity, loads, values)
```

onde *capacity* é a capacidade máxima da mochila,
loads é o array com os valores dos pesos e *values* o array correspondente dos valores.

A modelagem inicial do programa gera uma matriz de tamanho A x B onde:
A é a quantidade de itens a serem analisados;
B é o numero de unidades da capacidade da mochila.
Esta matriz é a chave da abordagem de programação dinâmica para este problema, pois funciona como uma memória que armazena as soluções encontradas conforme o algoritmo avança.

> A fim de facilitar a manipulação de matrizes foi adotada a biblioteca Numpy, que torna transparente uma série de operações relacionadas a estas estruturas. https://www.numpy.org/

a partir daí a "mochila" é preenchida item a item, testando cada objeto a fim de encontrar uma combinação com maior valor que o anteriormente encontrado.

Resultados

Dada a entrada:

```
values = [100, 60, 70, 15, 15]
loads = [8, 3, 6, 4, 2]
capacity = 10
```

podemos facilmente observar que o valor máximo possível de "carregar" na mochila é 130 (itens de peso 6 e 4), e o que obtemos na saída do programa é:

```
[[  0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0 100 100 100]
 [  0   0   0  60  60  60  60  60 100 100 100]
 [  0   0   0  60  60  60  70  70 100 130 130]
 [  0   0   0  60  60  60  70  75 100 130 130]
 [  0   0  15  60  60  75  75  75 100 130 130]]
('The biggest value is: ', 130)
```

Vamos explorar uma outra instância:

```
values = [1, 6, 18, 22, 28, 40, 60]
loads = [1, 2, 5, 6, 7, 9, 11]
capacity = 23
```

onde a solução ótima, está na seleção dos itens de peso 1, 2, 9 e 11. Esta combinação resulta em um valor de 107, que é exatamente o que podemos ver na saída do programa

```
[[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1]
 [  0   1   6   7   7   7   7   7   7   7   7   7   7   7   7   7   7   7   7   7   7   7   7   7]
 [  0   1   6   7   7  18  19  24  25  25  25  25  25  25  25  25  25  25  25  25  25  25  25  25]
 [  0   1   6   7   7  18  22  24  28  29  29  40  41  46  47  47  47  47  47  47  47  47  47  47]
 [  0   1   6   7   7  18  22  28  29  34  35  40  46  50  52  56  57  57  68  69  74  75  75  75]
 [  0   1   6   7   7  18  22  28  29  40  41  46  47  50  58  62  68  69  74  75  80  86  90  92]
 [  0   1   6   7   7  18  22  28  29  40  41  60  61  66  67  67  78  82  88  89 100 101 106 107]]
('The biggest value is: ', 107)
```

Selecionada uma entrada aleatória de tamanho maior podemos observar o comportamento do programa em termos de desempenho

```
values = [1, 6, 18, 22, 28, 40, 60, 18, 65, 52, 41, 37, 19, 14, 35, 70, 71, 59, 62, 45]
loads = [1, 2, 5, 6, 7, 9, 11, 3, 10, 13, 15, 16, 3, 8, 12, 14, 17, 3, 5, 9]
capacity = 35
```

como resultado obtemos:

```
[[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1
    1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1]
 [  0   1   6   7   7   7   7   7   7   7   7   7   7   7   7   7   7   7
    7   7   7   7   7   7   7   7   7   7   7   7   7   7   7   7   7   7]
 [  0   1   6   7   7  18  19  24  25  25  25  25  25  25  25  25  25  25
   25  25  25  25  25  25  25  25  25  25  25  25  25  25  25  25  25  25]
 [  0   1   6   7   7  18  22  24  28  29  29  40  41  46  47  47  47  47
   47  47  47  47  47  47  47  47  47  47  47  47  47  47  47  47  47  47]
 [  0   1   6   7   7  18  22  28  29  34  35  40  46  50  52  56  57  57
   68  69  74  75  75  75  75  75  75  75  75  75  75  75  75  75  75  75]
 [  0   1   6   7   7  18  22  28  29  40  41  46  47  50  58  62  68  69
   74  75  80  86  90  92  96  97  97 108 109 114 115 115 115 115 115 115]
 [  0   1   6   7   7  18  22  28  29  40  41  60  61  66  67  67  78  82
   88  89 100 101 106 107 110 118 122 128 129 134 135 140 146 150 152 156]
 [  0   1   6  18  19  24  25  28  36  40  46  60  61  66  78  79  84  85
   88  96 100 106 107 118 119 124 125 128 136 140 146 147 152 153 158 164]
 [  0   1   6  18  19  24  25  28  36  40  65  66  71  83  84  89  90  93
  101 105 111 125 126 131 143 144 149 150 153 161 165 171 172 183 184 189]
 [  0   1   6  18  19  24  25  28  36  40  65  66  71  83  84  89  90  93
  101 105 111 125 126 131 143 144 149 150 153 161 165 171 172 183 184 189]
 [  0   1   6  18  19  24  25  28  36  40  65  66  71  83  84  89  90  93
  101 105 111 125 126 131 143 144 149 150 153 161 165 171 172 183 184 189]
 [  0   1   6  18  19  24  25  28  36  40  65  66  71  83  84  89  90  93
  101 105 111 125 126 131 143 144 149 150 153 161 165 171 172 183 184 189]
 [  0   1   6  19  20  25  37  38  43  44  65  66  71  84  85  90 102 103
  108 109 112 125 126 131 144 145 150 162 163 168 169 172 180 184 190 191]
 [  0   1   6  19  20  25  37  38  43  44  65  66  71  84  85  90 102 103
  108 109 112 125 126 131 144 145 150 162 163 168 169 172 180 184 190 191]
 [  0   1   6  19  20  25  37  38  43  44  65  66  71  84  85  90 102 103
  108 109 112 125 126 131 144 145 150 162 163 168 169 172 180 184 190 191]
 [  0   1   6  19  20  25  37  38  43  44  65  66  71  84  85  90 102 103
  108 109 112 125 126 131 144 145 150 162 163 168 172 173 180 184 190 195]
 [  0   1   6  19  20  25  37  38  43  44  65  66  71  84  85  90 102 103
  108 109 112 125 126 131 144 145 150 162 163 168 172 173 180 184 190 195]
 [  0   1   6  59  60  65  78  79  84  96  97 102 103 124 125 130 143 144
  149 161 162 167 168 171 184 185 190 203 204 209 221 222 227 231 232 239]
 [  0   1   6  59  60  65  78  79 121 122 127 140 141 146 158 159 164 165
  186 187 192 205 206 211 223 224 229 230 233 246 247 252 265 266 271 283]
 [  0   1   6  59  60  65  78  79 121 122 127 140 141 146 158 159 164 166
  186 187 192 205 206 211 223 224 229 231 233 246 250 252 265 268 271 283]]
('The biggest value is: ', 283)
```

Apêndice

```python
# https://pt.stackoverflow.com/questions/205528/como-criar-uma-matriz-em-python
# https://stackoverflow.com/questions/17870612/printing-a-two-dimensional-array-in-pythc
# https://www.youtube.com/watch?v=pEH5uuC4nlw
#  pip install numpy
import numpy as np


def backpack(capacity, loads, values):
    # obtem a quantidade de itens
    numItens = len(loads)
    # matriz de tamanho numItens X capacity que funciona como memoria para armazenar as
    memory = np.zeros((numItens + 1, capacity +1), dtype=np.int)
    # for i, j para percorrer a matriz memoria
    for i in range(numItens + 1):
        for j in range(capacity + 1):
            # testa se e a primeira linha ou primeira coluna da matriz,
            # condicao de referencia
            if i == 0 or j == 0:
                memory[i][j] = 0
            else:
                # se o item i "couber" em uma mochila de capacidade j
                if loads[i -1] <= j:
                    # a memoria armazena o maior valor entre a soma do valor do item j c
                    memory[i][j] = max(values[i-1] + memory[i-1][j-loads[i-1]], memory[i
                else:
                    # caso o valor i seja menor que o peso do item j (nao "cabe") a memc
                    memory[i][j] = memory[i-1][j]
    # apenas para visualizarmos a matriz memoria
    print(np.matrix(memory))
    return memory[numItens][capacity]


# values = [1, 6, 18, 22, 28, 40, 60]
# loads = [1, 2, 5, 6, 7, 9, 11]
# capacity = 23

values = [1, 6, 18, 22, 28, 40, 60, 18, 65, 52, 41, 37, 19, 14, 35, 70, 71, 59, 62, 45]
loads = [1, 2, 5, 6, 7, 9, 11, 3, 10, 13, 15, 16, 3, 8, 12, 14, 17, 3, 5, 9]
capacity = 35

# values = [100, 60, 70, 15, 15]
# loads = [8, 3, 6, 4, 2]
# capacity = 10
result = backpack(capacity, loads, values)

print('The biggest value is: ', result)
```