Aluno: Lenington do C. Rios, 13211187

1 Memorização

Algorithm 1: ksMem(W,P,c)

input : Weight W, profit P and capacity c
output: The maximum capacity of knapsack

```
1 HashTable H ← ∅;
2 i ← |W|;
3 return memo(W, P, c, i, H);
```

Algorithm 2: memo(W,P,c,i,H)

input : Weight W, profit P, capacity c, position i, HashTable H
output: The maximum knapsack recursive memorization

```
1  if i = 0 or c = 0 then
2  |  return 0;
3  end
4  if (c, i) ∈ H then
5  |  return H[(c, i)];
6  else if W[i − 1] > c then
7  |  H[(c, i)] ← memo(W, P, c, i − 1, H);
8  |  return H[(c, i)];
9  else
10 |  return max(P[i − 1] + memo(W, P, c − W[i − 1], i −
   |    1, H), memo(W, P, c, i − 1, H));
11 return memo(W, P, c, i, H);
```

2 Tabulação

Algorithm 3: ksTab(W,P,c)

input : Weight W, profit P and capacity c
output: The last element in table

```
1 T ← [n...m];
2 i ← |W|;
3 return tab(W, P, c, i, T);
```

Algorithm 4: tab(W,P,c,p,T)

input : Weight W, profit P, capacity c, position i, table T
output: The last element in table

```
1  for n in range(i + 1) do
2  |   for m in range(c + 1) do
3  |   |   if n = 0 or m = 0 then
4  |   |   |   T[n][m] ← 0;
5  |   |   else if W[n − 1] <= m then
6  |   |   |   T[n][m] ← max(P[n−1]+T[n−1][m−W[n−1]], T[n−1][m]);
7  |   |   else
8  |   |   |   T[n][m] ← T[n − 1][m]);
9  |   end
10 end
11 return T[n][m];
```