DP (cont.): Knapsack

Introducción

- Se tiene una mochila que puede almacenar elementos, con un peso total máximo de K. Dados N elementos, donde cada uno tiene un peso W y un valor C. ¿Cuál es la cantidad óptima de elementos que puede cargar en la mochila, de forma tal de maximizar el valor total de los elementos?

Ejemplo

- Ejemplo:
  - $K = 9$, $N = 3$, $E[(W, C)] = \{(4, 3), (4, 3), (5, 4)\}$
  - Solución: elementos 0 y 2 (ó 1 y 2), con peso 9 y valor 7

¿Se puede resolver con DP?

- Tenemos que pensar si cumple con las propiedades necesarias:
  - Optimal Substructure
  - Overlapping Subproblems

Optimal substructure

- Tenemos que encontrar la función recursiva que resuelva el problema
- Podríamos pensarlo con una función knapsack(índice, capacidad actual)

$$k(i, w) = \begin{cases} 0 & k == n \lor w == 0 \\ k(i+1, w) & w_i > w \\ \max(k(i+1, w), k(i+1, w-w_i) + v[i]) & w_i <= w \end{cases}$$

Overlapping subproblems

- Pensar en el caso en que tenemos elementos de peso {2, 3, 2, 3, 6, 4}

Solución Top-Down

```
int n, k, elem[MAXN][2], DP[MAXN][MAXK]; // DP inicializado en -1
int dp (int idx, int w) {
    if (DP[idx][w] != -1) return DP[idx][w];
    if (idx == n || w == 0) return 0;
    int ans = dp(idx+1, w);
    if (elem[idx][0] <= w)
        ans = max(ans, dp(idx+1, w-elem[idx][0]) + elem[idx][1]);
    return DP[idx][w] = ans;
}
```

Solución Bottom-Up

```
int n, k, elem[MAXN][2], DP[MAXN+1][MAXK];
int dp () { // Consideramos n = 0 como sin elementos
    for (int i=0; i<=n; i++) DP[i][0] = 0;
    for (int j=0; j<=k; j++) DP[0][j] = 0;
    for (int i=1; i<=n; i++) for (int j=1; j<=k; j++) {
        if (elem[i-1][0] > j) DP[i][j] = DP[i-1][j];
        else DP[i][j] = max(DP[i-1][j], DP[i-1][j-elem[i-1][0]] + elem[i-1][1]);
    }
    return DP[n][k];
}
```