Práctica 2. Programación dinámica

Alejandro Guitarte Fernández
alejandro.guifer@alum.uca.es
Teléfono: 601048359
NIF: 32092444S

24 de noviembre de 2022

1. Formalice a continuación y describa la función que asigna un determinado valor a cada uno de los tipos de defensas.

$$f(dmg, salud, APS, rango) = \frac{dmg}{APS} + 0.6 * salud + 0.1 * rango$$

   Notas: dmg es el daño de la defensa y APS los ataques por segundo de la misma.

   Como se puede ver, la función valora, sobre todo, la relación daño/APS, es decir, el DPS (damage per second) de la defensa, y en menor medida, la salud y el rango.

2. Describa la estructura o estructuras necesarias para representar la tabla de subproblemas resueltos.

   En este caso, se ha usado una matriz estática normal de C. Se define el ancho de la matriz como el número de ases disponibles menos los que cuesta incluir el centro de extracción de minerales, que evidentemente ha de ser incluido. Por otro lado, la altura de la matriz es el número de defensas disponibles menos una (el centro de extracción de minerales).

3. En base a los dos ejercicios anteriores, diseñe un algoritmo que determine el máximo beneficio posible a obtener dada una combinación de defensas y *ases* disponibles. Muestre a continuación el código relevante.

```
void matriz(float** mat, unsigned int c, const std::list<Defense*>& defenses){
    int n = defenses.size();
    auto i = defenses.begin();

    for(int j = 0; j <= c; j++){
        if(j < (*i)->cost)
            mat[0][j] = 0;
        else
            mat[0][j] = value(*i);
    }

    for(++i; i != defenses.end(); i++){
        for(int j = 0; j <= c; j++){
            if(j < (*i)->cost)
                mat[indice(i, defenses)][j] = mat[indice(i, defenses)-1][j];
            else
                mat[indice(i, defenses)][j] =
                std::max(mat[indice(i, defenses)-1][j], mat[indice(i, defenses)-1][j-(*i)->
                    cost] + value(*i));
        }
    }
}
```

4. Diseñe un algoritmo que recupere la combinación óptima de defensas a partir del contenido de la tabla de subproblemas resueltos. Muestre a continuación el código relevante.

```
void recuperar(float** mat, std::list<int>& resultado, int n, int c, const std::list<Defense
    *>& defenses){
    int i = n-1, j = c;

    while(i != 0){
        if(mat[i][j] != mat[i-1][j]){
            resultado.push_front(defensa(i, defenses)->id);
            j -= defensa(i,defenses)->cost;
        }
        i--;
    }

    if(j >= (*defenses.begin())->cost)
        resultado.push_front((*defenses.begin())->id);
}
```

Todo el material incluido en esta memoria y en los ficheros asociados es de mi autoría o ha sido facilitado por los profesores de la asignatura. Haciendo entrega de este documento confirmo que he leído la normativa de la asignatura, incluido el punto que respecta al uso de material no original.