UNIVERSITÉ LAVAL

TP2 - Question 4

réalisé par Claire Bouttes - 536793110

IFT-3001

Table des matières

Introduction **2**

1 Présentation et explication **2**

2 Analyse **3**

2.1 OptimiserDegats 3

2.2 RecupererMonstres 4

2.3 programmationDynamique 5

Conclusion **5**

6 décembre 2020

Introduction

Dans un jeu de cartes simulant une bataille entre des monstre stirés d'un univers fantastique, sur chaque carte figure un monstre i capable d'infliger d_i points de dégât à son adversaire. Pour jouer cette carte, il faut utiliser r_i unités de magie rouge et b_i unités de magie bleue.

C'est le dernier tour du jeu et la stratégie gagnante consiste à jouer les cartes qui maximisent le nombre total de points de dégât infligés à son adversaire avec pour ce tour R unités de magie rouge et B unités de magie bleue.

Ce document explique et analyse un algorithme de programmation dynamique prenant en entrée un ensemble de n tuples $M = \langle d_1, b_1, r_1\rangle, ..., \langle d_n, b_n, r_n\rangle$, une quantité de magie rouge $R \in N$ et une quantité de magie bleue $B \in N$.

Cet algorithme retourne un sous-ensemble $S \subseteq M$ qui maximise la somme des points de dégâts tout en ayant une consommation totale de magie rouge et de magie bleue inférieure à R et à B.

1 Présentation et explication

Considérons une instance constituée :

— des i premiers monstres $0 \leq i \leq n$ constitué d'une magie rouge $r_1 \cdots r_i$, d'une magie bleue $b_1 \cdots b_i$ et de dégâts $d_1 \cdots d_i$

— d'une quantité de magie Rouge $0 \leq j \leq R$

— et d'une quantité de magie Bleue $0 \leq k \leq B$.

Soit $D[i, j, k]$ = la valeur de la solution optimale de cette instance Donc $D[i, j, k]$ = la valeur du sous-ensemble de valeur maximale des i premiers monstres dont le cout totale en magie bleue et rouge n'excède pas respectivement j et k.

Nous avons la condition initiale suivante :

$$D[0, j, k] = 0$$

Ainsi que la récurrence :

$$D[i,j,k] = \begin{cases} max\big(D[i-1,j,k], d_i + D[i-1,j-r_i,k-b_i\big) & sij >= r_i \wedge k >= b_i \\ D[i-1,j,k] & sinon \end{cases} \quad (1)$$

2 Analyse

2.1 OptimiserDegats

Algorithm 1: OptimiserDegats(M,R,B)

for $j \leftarrow 0 \cdots R$ **do**
  for $k \leftarrow 0 \cdots B$ **do**
    $D[0,j,k] = 0$
  end
end
for $i \leftarrow 1 \cdots M$ **do**
  for $j \leftarrow 1 \cdots R$ **do**
    for $k \leftarrow 1 \cdots B$ **do**
      if $r_i \leq j \wedge b_i \leq k$ **then**
        $D[i,j,k] = max\big(D[i-1,j,k], d_i + D[i-1,j-r_i,k-b_i]\big)$
      end
      else
        $D[i,j,k] = D[i-1,j,k]$
      end
    end
  end
end
return D

Nous choisirons comme opérations de base les assignations à D. Le temps d'exécution dépend uniquement de la taille des instances n, R et B.

$$\begin{aligned} C_{OptimiserDegats}(n,R,B) &= \sum_{j=0}^{R}\sum_{k=0}^{B} 1 + \sum_{i=1}^{n}\sum_{j=1}^{R}\sum_{k=1}^{B} 1 \\ &= (R+1)(B+1) + nRB \end{aligned} \tag{2}$$

2.2 RecupererMonstres

Algorithm 2: RecupererMonstres(D,M,R,B)

$l \leftarrow 0$;
$j \leftarrow R$;
$k \leftarrow B$;
for $i \leftarrow M \cdots 1$ **do**
  if $D[i,j,k] > D[i-1,j,k]$ **then**
    $l = l + 1; L[l] = D[i,k,k]$;
    $j = j - r_i$;
    $k = k - b_i$;
  end
end
return L

Nous choisirons comme opération de base la condition $D[i,j,k] > D[i-1,j,k]$. Le temps d'exécution dépend uniquement de la taille de l'instance.

$$C_{RecupererMonstres}(n,R,B) = \sum_{i=1}^{n} 1 \tag{3}$$

$$n$$

2.3 programmationDynamique

Algorithm 3: programmationDynamique

return $RecupererMonstres(OptimiserDegats(M, R, B), M, R, B)$

Nous avons alors :

$$\begin{aligned} C(n, R, B) &= C_{OptimiserDegats}(n, R, B) + C_{RecupererMonstres}(n, R, B) \\ &= RB + R + B + 1 + nRB + n \end{aligned} \tag{4}$$

D'après la règle du maximum, nous sommes alors en $\theta(nRB)$ pour n > 0, B > 0 et R > 0.

Conclusion

En conclusion, l'algorithme designé a un nombre d'exécution qui se situe en $\theta(nRB)$ ce qui me semble être correct.