Projet Algorithmique

Nathan Buskulic/Hakim ???

1 Partie Théorique

1.1 Enumeration des solutions

Question 1

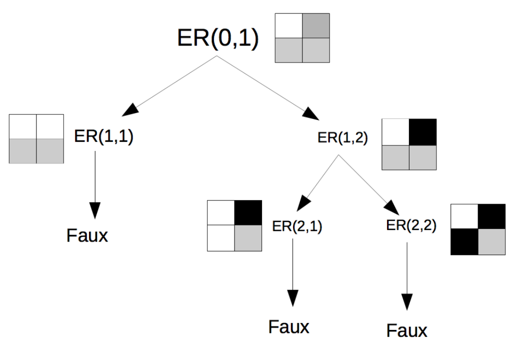

Figure 1: Arbre d'exploration ER(0,2)

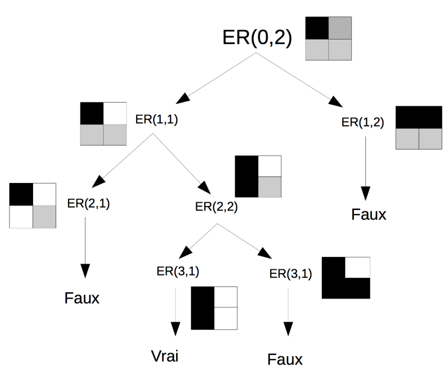

Figure 2: Arbre d'exploration ER(0,2)

A chaque noeud, on explore d'abord la branche à gauche puis celle de droite. Ainsi on explore d'abord ER(1,1) avant ER(1,2).

Question 2
Puisque chaque case peut être soit blanche, soit noire, et qu'on peut parcourir l'arbre de toutes les combinaisons possibles issues des p cases libres, on a une complexité en 2^p

1.2 Problème Décision : cas particulier du vecteur

1.2.1 Cas du vecteur libre

Question 3
Si on connaît tous les *T(j,l)*, on connaît particulièrement les *T(j,l)* avec *l=k* (si *l=k*, on respecte la séquence L puisque l'on prend en considération tous les blocs de cette séquence).
Il suffit donc de trouver un *T(j,l)* vrai avec *l=k* pour savoir si on peut colorier V en respectant L.

Question 4
l=0 signifie que l'on doit avoir au plus 0 cases noires sur le vecteur. Puisque nous sommes dans le cas d'un vecteur libre, aucune case n'est coloriés. On a donc un vecteur de taille 1 à m libre. Il nous suffit de colorier toutes les cases d'un tel vecteur en blanc pour satisfaire *T(j,0)*, quelque soit $j \in \{0...m\text{-}1\}$

Question 5
a) On observe que $V[0...L_1-1]$ contient L_1 cases ce qui suffit pour respecter la séquence L_1 (en effet, il n'y a pas à avoir de case blanches servant d'interstices entre les blocs puisque l'on a qu'un bloc à respecter ici). Il faut donc juste colorier en noir toutes les cases du vecteur $V[0...L_1-1]$ pour répondre au problème. $T(L_1\text{-}1,1)$ est donc vrai.

b)Si *l=1*, on a besoin de L_1 cases pour avoir *T(j,l)* = vrai. Or on a $j < L_1 - 1$ ce qui donne que *V[0...j]* a au maximum L_1-1 cases. On a donc $T(L_1\text{-}1,1)$ avec $j < L_1 - 1$ et *l=1* faux.

Plus généralement, on peut observer que pour avoir *T(j,l)* vrai, il faut que j ait autant de blocs noirs que ce que la séquence L impose en plus des blocs blancs qu'il y aura entre chaque série de blocs noirs. Il faut donc que j soit égal au minimum à :

$$\sum_{i=1}^{l}(L_i + 1) - 1$$

On observe un -1 pour représenter le fait que la dernière série de blocs noirs n'a pas besoin d'être suivi d'une case blanche.
On peut donc représenter la formule sous cette forme :

$$\sum_{i=1}^{l-1}(L_i) + L_l + l - 1$$

Donc même si éventuellement $\sum_{i=1}^{l-1}(L_i)$ peut être nul, si on a $l \geq 2$, on a besoin de $j \geq L_l + l - 2$ avec $l - 2 \geq 0$, or on a $j < L_l - 1$. Donc *T(j,l)* avec $l \geq 2$ et $j < L_l - 1$ est faux.
On a donc prouvé que *T(j,l)* avec $j < L_l - 1$ et $l = 1$ ou $l \geq 2$ était faux. C'est donc faux pour $l \geq 1$.

c) En utilisant le même raisonnement qu'à la question précédente, on observe que si $l \geq 2$, il nous faut un nombres de cases égal à :

$$\sum_{i=1}^{l-1}(L_i) + L_l + l - 1$$

On a donc besoin au minimum de $j \geq L_l + l - 2$ avec $l - 2 \geq 0$. On obtient donc qu'il nous faut $j \geq L_l$ or on a $j \leq L_l - 1$. Dans ces conditions, *T(j,l)* est toujours faux.

Question 6
Si V[j] = 1, *T(j,l)* est vrai pour :

$$j \geq \sum_{i=1}^{l-1}(L_i) + L_l + l - 2$$

Si V[j] = 1, *T(j,l)* est vrai pour :

$$j \geq \sum_{i=1}^{l-1}(L_i) + L_l + l - 1$$

Dans tout les autres cas de figure, *T(j,l)* est faux.

Question 7
Cas de base :

$$T(j, 0) = vrai$$

$$T(L_1 - 1, 1) = vrai$$

Soit *T(j,l)* = vrai alors :

$$T(j + 1, l) = vrai$$

$$T(j + L_{l+1} + 1, l + 1) = vrai$$

1.2.2 Cas général d'un vecteur non libre

Question 8
Cette ligne sert à ne pas recalculer la case du tableau si elle a déjà une valeur (et donc qu'elle a déjà été calculé), c'est le principe de la programmation dynamique.

Question 9
Voila le nouvel algorithme :

```
Algorithme TestVecteur Rec(V : vecteur, j : entier, l : entier, T T : tableau) : booléen
  c1: booléen -- Cas où la case j est blanche
  c2: booléen -- Cas où la case j est noire

  Si (l = 0) Alors Retourner TestSiAucun(V, 0, j, 2)
  Si (l = 1) et (j = Ll − 1) Alors Retourner TestSiAucun(V, 0, j, 1)
  Si (j ≤ Ll - 1) Alors Retourner faux
  Si (V [j] = 2) Alors
    c1 ←faux;
  Sinon
    Si (T T[j-1][l] != ‘‘non visité’’) Alors Retourner T T[j-1][l]
    Sinon
      c1 ← TestVecteur Rec(V, j − 1, l, T T)
    FinSi
  FinSi
  Si (non TestSiAucun(V, j − (Ll − 1), j, 1)) Alors
    c2 ← faux
  Sinon
    Si (V [j − Ll]=2) Alors
      c2 ← faux
    Sinon
      Si (T T[j-Ll-1][l-1] != ‘‘non visité’’) Alors Retourner T T[j-Ll-1][l-1]
      Sinon
        c2 ← TestVecteur Rec(V, j − Ll − 1, l − 1, T T)
      FinSi
    FinSi
  FinSi
  T T[j][l] ← c1 ou c2
  Return T T[j][l]
```

On observe que la fonction *TestSiAucun* a une compléxité en $O(m)$ et on va faire au maximum m appel récursifs sur *TestVecteurRec*. On aura donc une complexité total en $O(m^2)$