Corrigé Examen TPRO 2023 -2024

Exercice 1

Equation de recurrence:

$T(1) = a$

$T(n) = T(n-1) + bn$

Dilatation

$T(n) = T(n-1)+ bn$

$= T(n-2) + b(n-1) + bn$

$= T(n-2) + 2bn - b$

$= T(n-3) + b(n-2) + 2bn - 1$

$= T(n-3) + 3bn - 3b$

$= T(n-4) + b(n-3) + 3bn - 3b$

$= T(n-4) + 4bn - 6b$

...

$= T(n-i) + ibn - (i(i-1)/2)\, b$

Pour i = n-1, T(n-i) est T(1) = a

Donc

$T(n) = a + n(n-1)b - ((n-1)(n-2))/2\, b$

$T(n)= a + b/2 (2n(n-1) + (n-1)(n-2))$

$T(n= a + b/2 (n^2 + n - 2)$

C 'est $O(n^2)$

Equation non homogène

$T_n - T_{n-1} = bn = 1^n\, bn$

Equation caractéristique

$(x-1) (x-1)^2 = 0$

Solution

$T(n) = c1\ 1^n + c2\ n\ 1^n + c3\ n^2\ 1^n$

Pour n=1,

$T(1) = c1 + c2 + c3 = a$

On peut prendre:

c1=0, c2=0 et c3 = a

$T(n) = a\ n^2$

C'est $O(n^2)$

Devinette

Nous supposons que T(n-1) est $O((n-1)^2)$

c'est à dire

Il existe c et n0 positifs tels que pour tout n > n0

$T(n-1) \leq c(n-1)^2$ (1)

Et nous devons montrer que

$T(n) = O(n^2)$

Comme on a

$T(n) = T(n-1) + bn$

La relation (1) devient

$T(n) \leq c\ (n-1)^2 + bn$

$T(n) \leq c\ (n^2 - 2n +1) + bn$

$T(n) \leq c\ n^2 - n(2c - b) + c$

Pour c = b/2, on a

$T(n) \leq (b/2)\ n^2 + b/2$

$T(n) \leq (b/2) (n^2 + 1)$

On a donc bien trouvé un rang (on peut prendre n0) et une constante (on peut prendre b/2) tel que

$T(n) \leq (b/2) (n^2 + 1)$.

T(n) est donc $O(n^2+1)$ qui est aussi $O(n^2)$

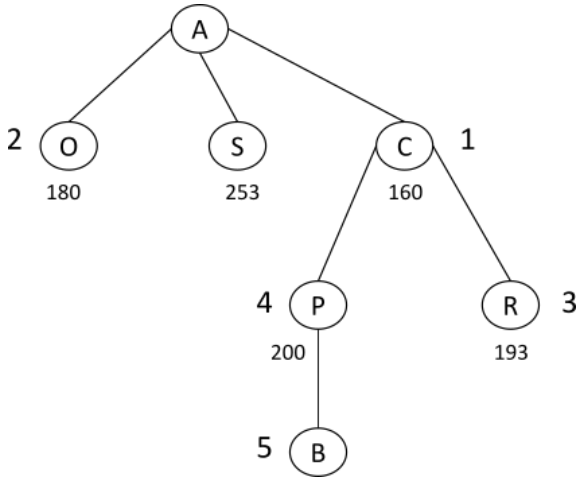
Exercice 2

Pour BFS (Best First Search),

- ne tenir compte que de l'estimation de la distance restante
- ne pas considérer des nœuds déjà générés.

L'arbre suivant est obtenu:

- les valeurs sous les nœuds désignent les estimations des distances restantes
- les valeurs à coté des nœuds indiquent l'ordre des nœuds visités

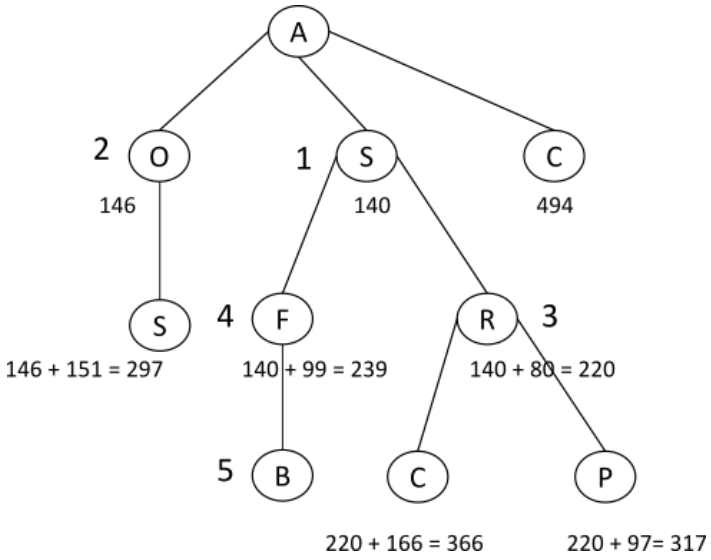


Nœuds visités	Nœuds ouverts
A	C(160), O(180), S(253)
C	O(180), R(193), P(200), S(253)
O	R(193), P(200), S(253)
R	P(200), S(253)
P	B(0), S(253)
B	

Le chemin déterminé par BFS est donc A - C - P - B

Pour BB (Branch and Bound),
- ne tenir compte que des cumuls des distances réelles
- on considère les nœuds déjà générés.

L'arbre suivant est obtenu:



Nœuds visités	Branches ouvertes
A	A-S(140), A-O(146), A-C(494)
S	A-O(146), A-S-R(220), A-S-F(239), A-C(494)
O	A-S-R(220), A-S-F(239), A-O-S(297), A-C(494)
R	A-S-F(239), A-O-S(297), A-S-R-P (317), A-S-R-C (366), A-C(494)
F	A-S-F-B (239), A-O-S(297), A-S-R-P (317), A-S-R-C (366), A-C(494)
B	

Le chemin déterminé par BFS est donc A - S - F - B

Exercice 3

Il s'agit de transformer la formule SAT qui suit en une formule 3-SAT équivalente

F = x1 (x1+x2') (x1'+x2+ x3 + x4' + x5)

x1 est transformé en (x1 + a + b) (x1 + a + b') (x1 + a' + b) (x1 + a' + b')
(x1+x2') est transformé en (x1+x2' + c) (x1+x2' + c')
(x1'+x2+ x3 + x4' + x5) est transformé en (x1'+x2+d)(d'+x3+e)(e'+x4'+x5)

F exprimé en 3-SAT est donc
F= (x1 + a + b) (x1 + a + b') (x1 + a' + b) (x1 + a' + b') (x1+x2' + c) (x1+x2' + c') (x1'+x2+d)(d'+x3+e)(e'+x4'+x5)

Exercice 4

1. Rappelons que A est NP-complet si
   - A ∈ NP,
   - ∃ B ∈ NP-complet : B $\leq_P$ A

CLIQUE est bien dans NP, car il existe un vérificateur en temps polynomial ($O(k^2)$) pour tester si un sous ensemble constitue une clique.
Comme CLIQUE est dans NP et SAT $\leq_P$ CLIQUE, alors CLIQUE est NP-complet.

2. Il n'existe pas un vérificateur en temps polynomial pour tester si un sous ensemble constitue une couverture minimale (VC). Le seul vérificateur qui existe est exponentiel car il faudrait considérer tous les sous ensembles formés d'un seul element, puis tous les sous ensembles formés de deux éléments, etc. Le vérificateur s'arrête dés qu'il trouve un sous ensemble W dans lequel pour chaque arête de V au moins une extrémité appartient à W.

3. Nous utilisons le théorème "Si B est NP-complet et si B $\leq_P$ A, alors A est NP-difficile"
Comme CLIQUE est NP-complet et CLIQUE $\leq_P$ VC, alors VC est NP-difficile.

Exercice 5

Par Automate:

```
1 : a
3 : Si t1 allera 2
    b
    Allera 1
2 : Si t2 Allera 3
    c
    Si t3 Allera 4
    Allera 3
4 : Stop
```

On réécrit l'algorithme comme suit en attribuant une étiquette ($) au test t3

```
1 : a
3 : Si t1 allera 2
    b
    Allera 1
2 : Si t2 Allera 3
    c
$: Si t3 Allera 4
    Allera 3
4 : Stop
```

Nous pouvons alors dresser l'automate suivant :

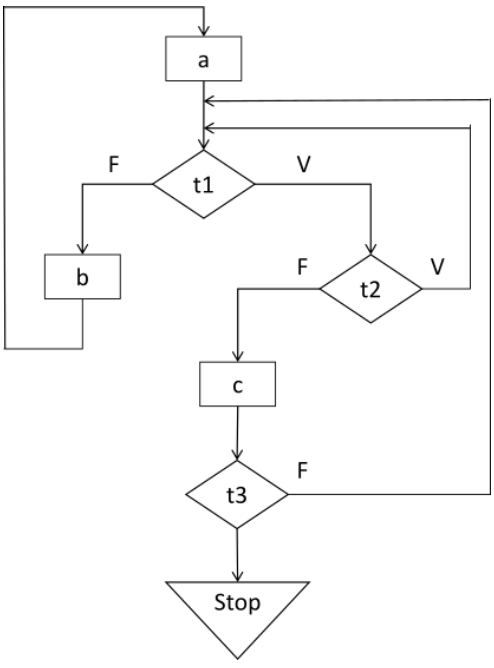
Étiquettes (tests)	Vrai	Faux
1	a ; Allera 3	
3 (t1)	Allera 2	b ; Allera 1
2 (t2)	Allera 3	c ; Allera $
$ (t3)	Allera 4	Allera 3
4	Stop	

Le D-Algorithme suivant est obtenu à partir de l'automate:

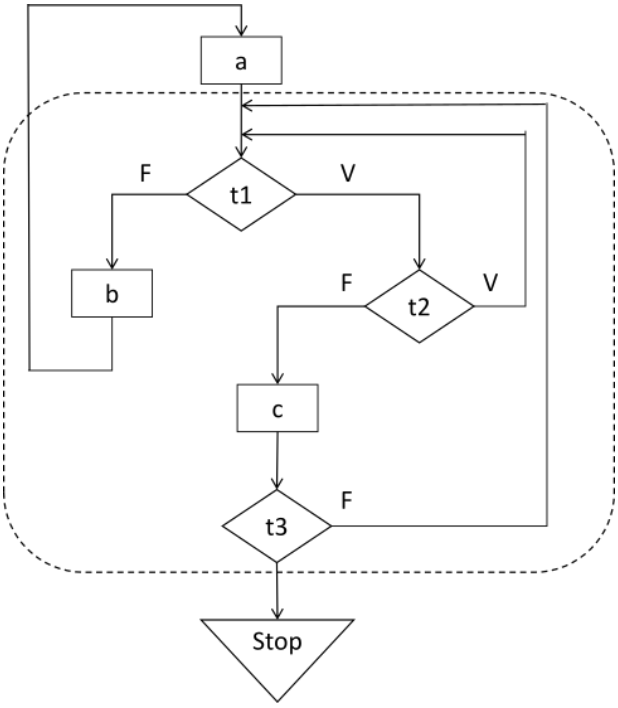
```
Etat := 1
Tq Etat <> 4
        Si Etat = 1
                a; Etat := 3
        Sinon
                Si Etat = 3
                        Si t1 : Etat := 2 Sinon b; Etat := 1 Fsi
                Sinon
                        Si Etat=2
                                Si t2 : Etat := 3 Sinon c ; Etat := $  Fsi
                        Sinon
                                Si Etat = $ :
                                        Si t3 : Etat := 4 Sinon Etat := 3 Fsi
                                Fsi
                        Fsi
                Fsi
        Fsi
Ftq
```

Par organigramme

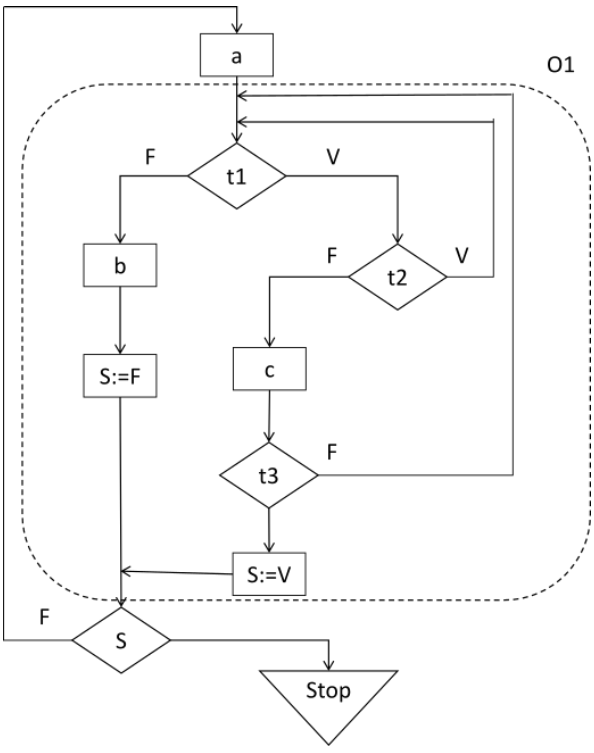
L'organigramme associé au B-algorithme est le suivant:



L'organigramme commence par une action, il est donc du type 1 comme montré dans la figure suivante:



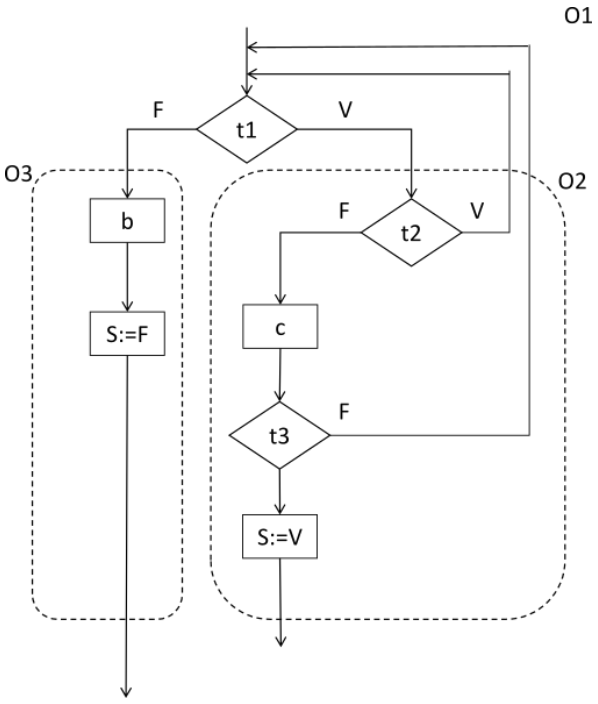
Il est transformé comme suit :



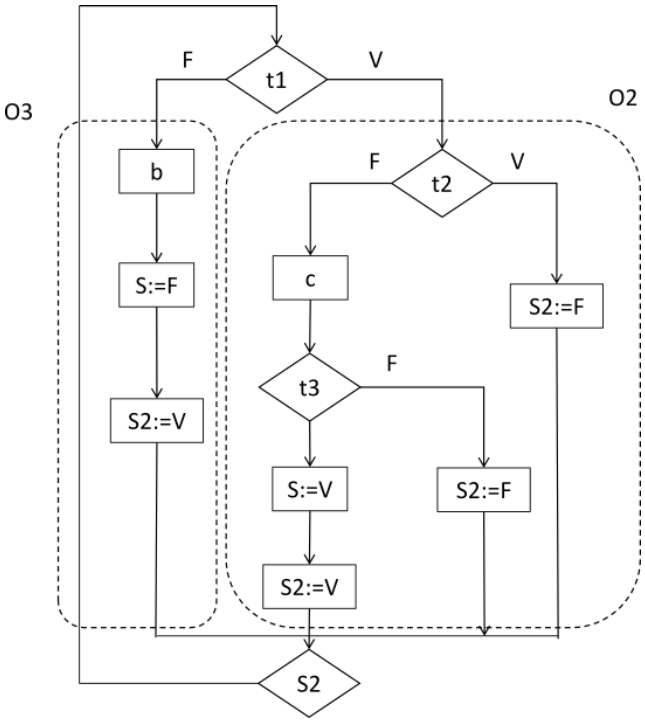
Le code associé est

Répéter a; O1 Jusqu'à S

Nous devons maintenant transformer O1 qui est le suivant :



Il s'agit du type 2. Il est transformé comme suit :



Le code correspondant à O1 est donc le suivant :

Répéter Si t1 O2 Sinon O3 Fsi Jusqu'à S2

Avec O2 :

Si t2 : S2 := Faux
Sinon
c
Si t3 S:= Vrai; S2:= Vrai Sinon S2 := faux Fsi
Fsi

Et O3 : *b; S:= Faux ; S2 := Vrai*

Code final :

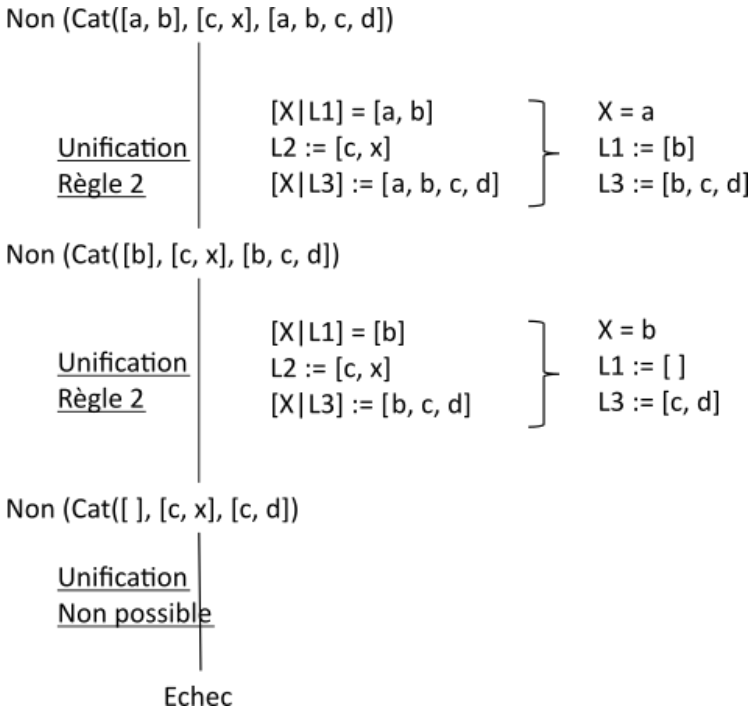
```
Répéter
    a
    Répéter
        Si t1
            Si t2
                S2 := Faux
            Sinon
                c
                Si t3
                    S:= Vrai ; S2:= Vrai
                Sinon
                    S2 := faux
                Fsi
            Fsi
        Sinon
            b ;  S:= Faux ; S2 := Vrai
        Fsi
    Jusqu'à S2
Jusqu'à S
```

Exercice 6

Soit le programme Prolog suivant:

```
Cat([ ], L, L).                          // Règle 1
Cat([X|L1], L2, [X|L3] ):-Cat(L1, L2, L3)   // Règle 2
```

Fonctionnement de Prolog pour la question Cat ([a, b], [c, x], [a, b, c, d])



Fonctionnement de Prolog pour la question Cat (L1, [c, d], [a, b, c, d])