Corrigé de l'examen TPRO // 2024 - 2025

Exercice 1

1. La fonction X déplace le plus grand élément du sous tableau A[i..j] à la position j et rend dans A[j] le plus grand élément.

2. Preuve

Pour i = j :

A[i] = A[j] = Le plus grand. CQFD

Pour i > j :

On suppose l'appel interne vrai, c'est à dire le sous-tableau A[i..j-1] ré arrangé et A[j-1] est le plus grand. L'analyse de l'instruction qui suit l'appel nous permet d'affirmer:

- Si ce dernier élément est plus grand que A[j], une permutation est entreprise et donc le tableau A[i..j] est ré arrangé et A[j] devient son plus grand élément. CQFD
- S'il n'est pas plus grand, A[j] reste le plus grand élément. CQFD

3. Complexité

Soit n le nombre d'éléments entre i et j. n= j - i + 1

L'équation de récurrence est

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

Il s'agit d'une équation non homogène

$$T_n - t_{n-1} = b = 1^n b$$

Ayant comme équation caractéristique

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

La solution est donc

$$C_1 . 1^2 + C_2 .n .1^n$$

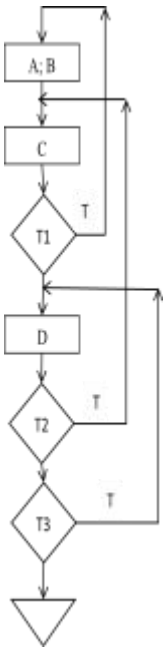
C'est O(n)

Exercice 2

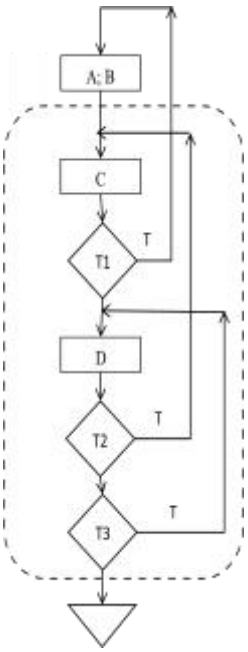
Nous donnerons deux solutions.

1. Transformation de Bohm & Jaccopini

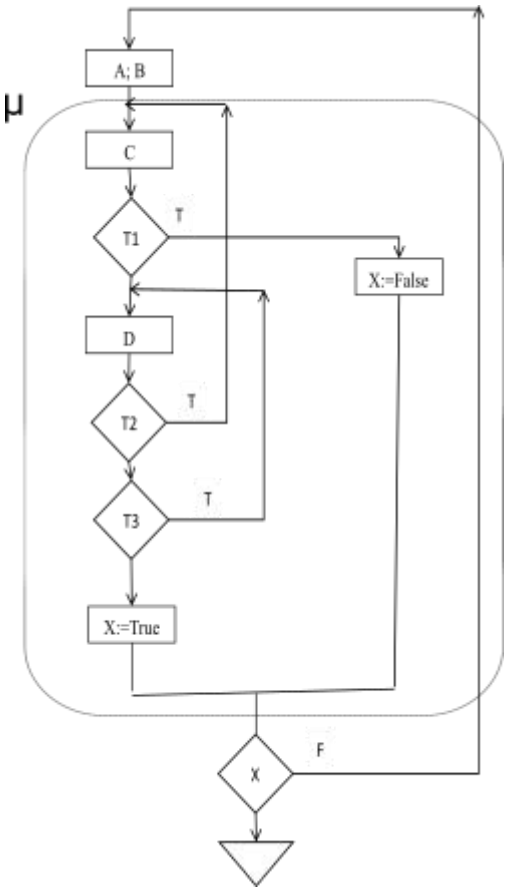
L'organigramme correspondant au B-algorithme est le suivant :



Il est de la catégorie 1.



Il est transformé comme suit :



La transformation donne

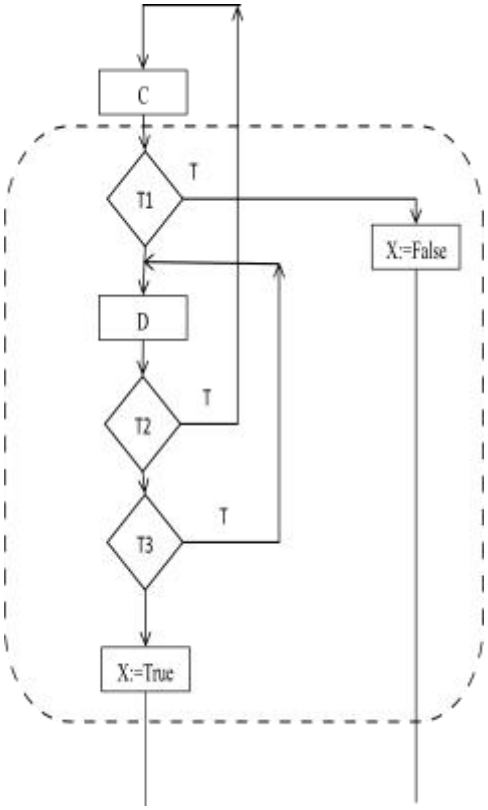
Repeat

A;B

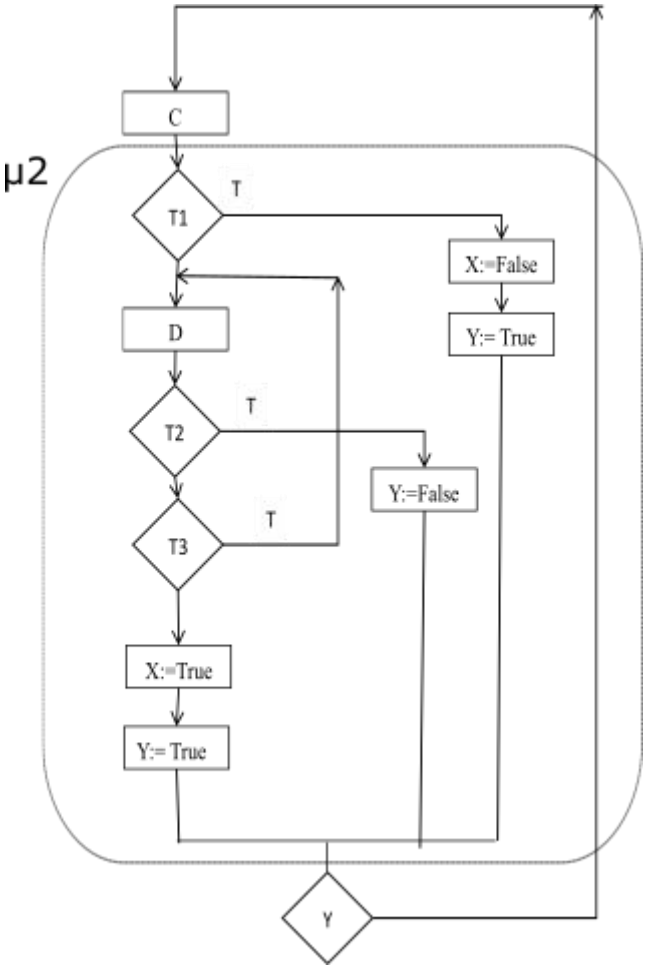
μ

Until X

Transformation de μ
Il est de la catégorie 1

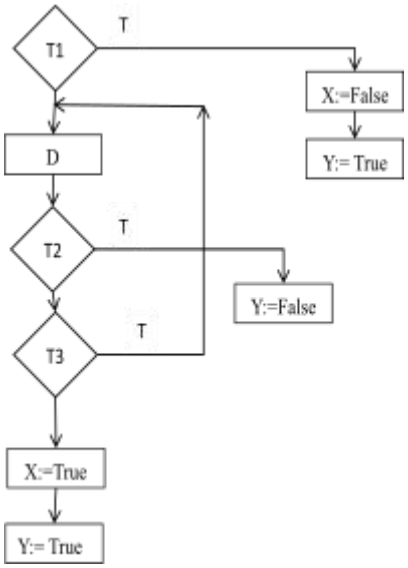


Il est transformé comme suit :



μ =
Repeat
C
μ2
Until Y

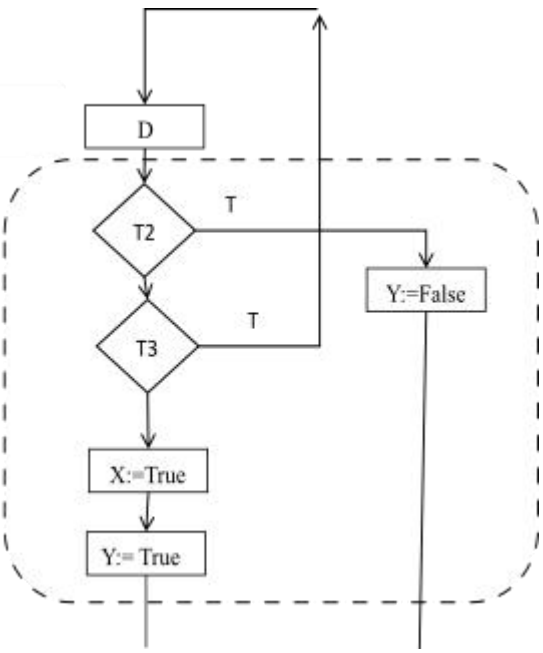
μ2 est l'organigramme suivant:



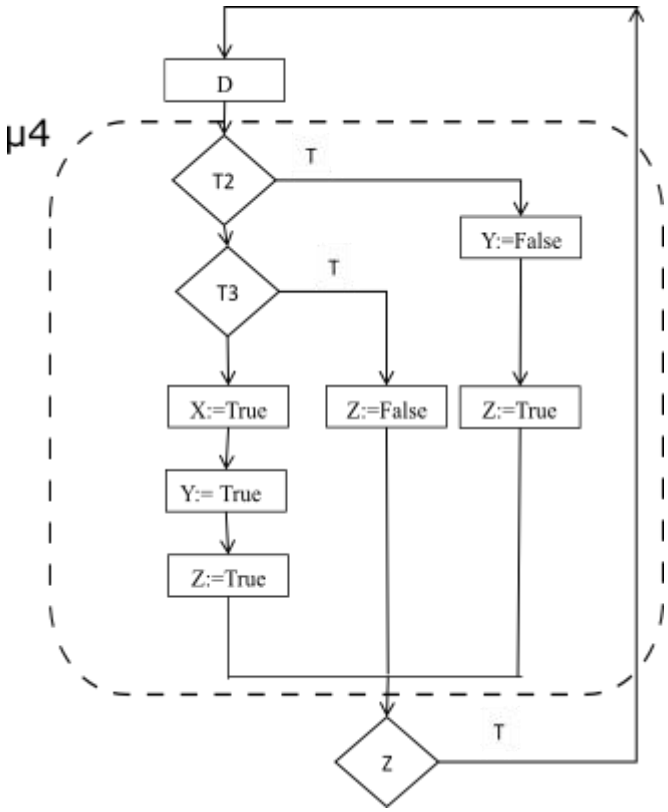
qui peut s'exprimer comme suit:

If T1
X:= False;
Y:= True;
Else
μ3
Endif

Avec μ3 l'organigramme suivant qui est de la catégorie 1.

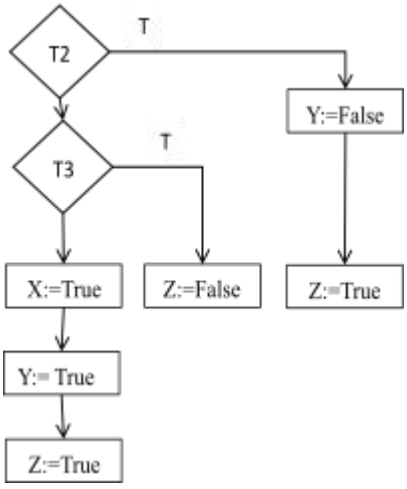


µ3 se transforme comme suit:



µ3 =
Repeat
D
µ4
Until Z

µ4 étant l'organigramme suivant:



La transformation se termine comme µ4 correspondant à un D-algorithme.

```
μ4 =
If T2
        Y:= False;
        Z := True
Else
        If T3
                Z:= False
        Else
                X:= True;
                Y:= True;
                Z := True
        Endif
Endif
```

Le D-algorithme final est donc :

```
Repeat
        A;B
        /* μ */
        Repeat
                C
                /* μ2  */
                If T1
                        X:= False;
                        Y:= True;
                Else
                        /* μ3 */
                        Repeat
                                D
                                /* μ4 */
                                If T2
                                        Y:= False;
                                        Z := True
                                Else
                                        If T3
                                                Z:= False
                                        Else
                                                X:= True;
                                                Y:= True;
                                                Z := True
                                        Endif
                                Endif
                        Until Z
                Endif
        Until Y
Until X
```

2. **Transformation par automate :**
Le B-algorithme peut être ré écrit comme suit:

```
1 : A ;  B
2: C
$1:  If T1 Goto 1
3: D
$2: If T2 Goto 2
$3: If T3 Goto 3
    Stop
```

L'automate correspondant est le suivant :

States	Vrai (t1, t2, t3)	Faux
1	A; B ; → 2	
2	C ; → $1	
$1	→ 1	→ 3
3	D; → $2	
$2	→ 2	→ $3
$3	→ 3	Stop

Le D-Algorithme correspondant est :

```
State := 1
While State <> Stop
        Case State of
                1 : A; B; State := 2
                2 : C; State := $1
                $1 : If t1 State := 1 Else State := 3 Endif
                3 : D; State := $2
                $2 : If t2 State := 2 Else State := $3 Endif
                $3 : If t3 State := 3 Else State := Stop
        EndCase
EndWhile
```

Exercice 3

1. Classe du problème SSP : NP

On peut vérifier une solution en temps polynomiale. La vérification consiste à calculer la somme des éléments d'un ensemble.

2. Solution Brute force

A désigne l'ensemble des valeurs, N le nombre de valeurs et K une somme donnée.
La fonction SSP rend la valeur Vrai si un sous ensemble de somme K existe dans l'ensemble représenté par le tableau A, Faux sinon.

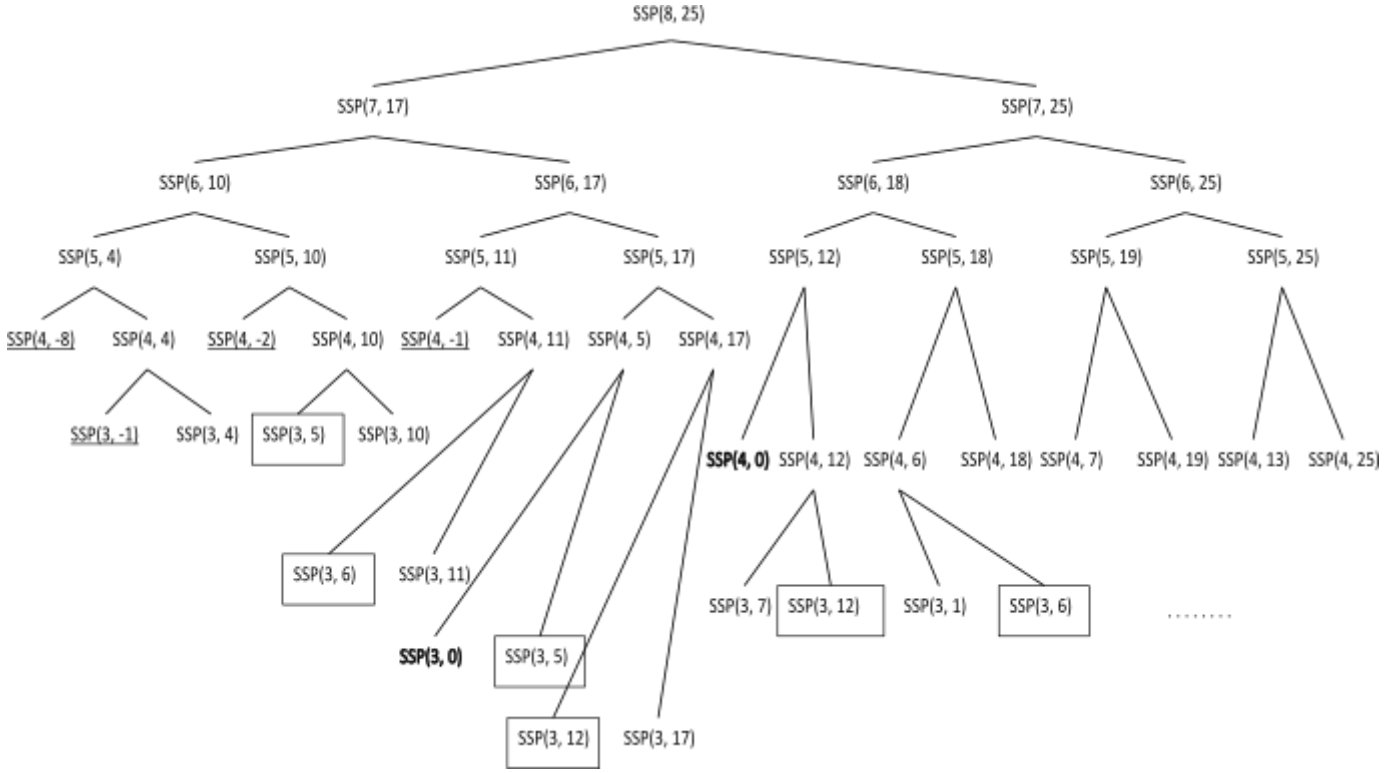
```
FONCTION SSP ( A , N , K ) : BOOLEEN
SOIT
```

```
    A : VECTEUR ( 8 ) ;
    N , K : ENTIERS ;
    Inclure , Exclure : BOOLEENS ;

  DEBUT
{ Retourne Vrai  si  la somme devient égale à 0 (Sous ensemble trouvé) }
    SI ( K = 0 )
       SSP:= VRAI
    SINON
    { Il ne reste plus d'élément ou la somme devient négative }
       SI ( N < 1 ) OU ( K < 0 )
          SSP:= FAUX
       SINON
       { Cas 1. Inclure l'élément courant A[n] dans le sous ensemble et appel récursif pour
   les n-1 éléments restant avec le total restant k-A[n]}
            Inclure := SSP( A , N - 1 , K - ELEMENT ( A [ N ] ) ) ;
            { Cas 2.Exclure l'élément courant A[n]et appel récursif pour les n-1 éléments restants }

            Exclure:= SSP( A , N - 1 , K ) ;
            { Retourne Vrai si on peut obtenir le sous ensemble en incluant ou en excluant l'élément courant}
            SSP:= Inclure OU Exclure ;

       FSI
    FSI
  FIN
```

2. Programmation dynamique

Oui, on peut appliquer la programmation dynamique du fait qu'il y a des sous problèmes identiques qui se répètent.
Dans la figure ci-après, nous avons dressé l'arbre des appels engendrés par le programme SSP pour K = 25 et A=[3 , 1 , 4 , 5 , 12 , 6 , 7 , 8].

En gras : solutions
Souligné : les cas d'échec
Encadré : les cas de répétitions



Arbre des appels

3. **Solution Programmation Dynamique**

Les instructions en Gras désignent ce qui a été rajouté à la fonction précédente.

Afin d'éviter les appels identiques, on sauvegarde dans une liste les résultats des appels récursifs.

Un élément de la liste est donc un triplet (N, K, Résultat) avec Résultat = Vrai ou Faux.

Les modules suivants sont utilisés :

- Ajouter_liste : Ajoute un nouvel triplet à une liste
- Recherche_liste : recherche un triplet dans une liste

```
FONCTION SSP_DP ( A , N , K ) : BOOLEEN
  SOIT
     A : VECTEUR ( 8 ) ;
     N , K : ENTIERS ;
     Inclure , Exclure : BOOLEENS ;

  DEBUT
 { Retourne Vrai  si  la somme devient égale à 0 (Sous ensemble trouvé) }
     SI ( K = 0 )
        SSP_DP:= VRAI
     SINON
    { Il ne reste plus d'élément ou bien la somme devient négative }
        SI ( N < 1 ) OU ( K < 0 )
           SSP_DP:= FAUX
        SINON
            APPEL Recherche_liste ( L , N , K , Trouv , P ) ;
            SI  NON Trouv
                  { Cas 1. Inclure l'élément courant A[n] dans le sous ensemble et appel récursif
                    pour les n-1 éléments restant avec le total restant k-A[n]}
```

```
            Inclure := SSP_DP( A , N - 1 , K - ELEMENT ( A [ N ] ) ) ;
            { Cas 2. Exclure l'élément courant A[n]et appel récursif pour les n-1 éléments restants }
            Exclure:= SSP_DP( A , N - 1 , K ) ;
            { Retourne Vrai si on peut obtenir le sous ensemble en incluant ou en excluant l'élément courant}
            SSP_DP:= Inclure OU Exclure ;
            APPEL Ajouter_liste ( L , N , K , Inclure OU Exclure )
        SINON
            SSP_DP := STRUCT ( VALEUR ( P ) , 3 )     // Résultat sauvegardé
        FSI
      FSI
    FSI
  FIN
```

4. Classe du problème KNAPSACK: non dans NP. Il est dans EXP.
On ne peut vérifier une solution en temps polynomiale.

5. Reduction
On démarre de SSP

-E = {a1, a2, ..., an]

-Une somme S.

Pour chaque élément de E, on crée l'élément i dans le problème Knapsack ayant comme poids Pi = ai et comme valeur Vi = ai
On prendra comme capacité C = S.

Le problème du Knapsack revient a trouver un sous ensemble d'éléments tel que :

- Total des poids , $\sum Pi \leq C = S$
- Total des valeurs . $\sum Vi = S$

Ceci peut être interprétée comme suit :
- Si le problème Knapsack a une solution avec un poids total et une valeur totale les deux égaux à S, alors le sous ensemble correspondant représente une solution au problème SSP.
- Si aucune solution existe, le problème SSP n'a pas de sous ensemble dans la somme est égale à S.

Exercice 4
$(\lambda x.\lambda y.\lambda z.(xz)(yz))\ f\ g\ x \rightarrow (\lambda y.\lambda z.(fz)(yz))\ g\ x \rightarrow (\lambda z.(fz)(gz))\ x \rightarrow (f\ x)(g\ x)$
$(\lambda p.\lambda q.p\ q\ F)\ T\ F \rightarrow (\lambda q.T\ q\ F)\ F \rightarrow T\ F\ F \rightarrow (\lambda x.\lambda y.x)\ F\ F \rightarrow F$

Exercice 5
Soit G le graphe orienté avec les arêtes suivantes : (a, b), (b, c), (c, d), (a, e), (e, d)

Le programme PROLOG est le suivant:

```
edge(a, b).
edge(b, c).
edge(c, d).
edge(a, e).
edge(e, d).
path(X, Y) :- edge(X, Y).
path(X, Y) :- edge(X, Z), path(Z, Y).
```