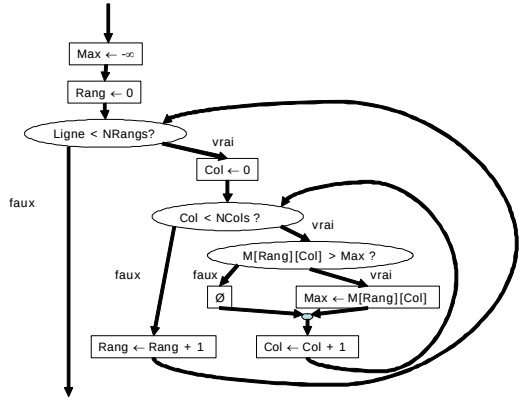
ITI1520 Section 9 Solutionnaire aux exercices

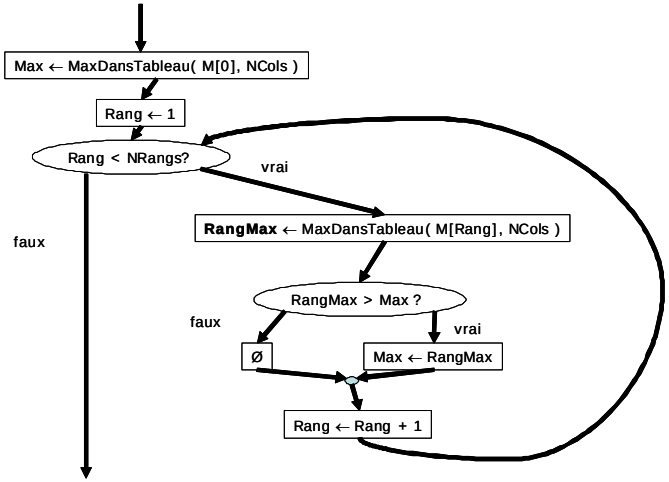
- Exercice 9-1: La matrice M est un tableau de 4 tableaux, chacun ayant 6 éléments. Ainsi:
  M[1][2] contient **75**
  M[2][5] contient **88**
  M[4][1] contient **une erreur d'exécution!**
  M[3] contient une **réfèrence à { 58, 72, 66, 57, 74, 74 }**

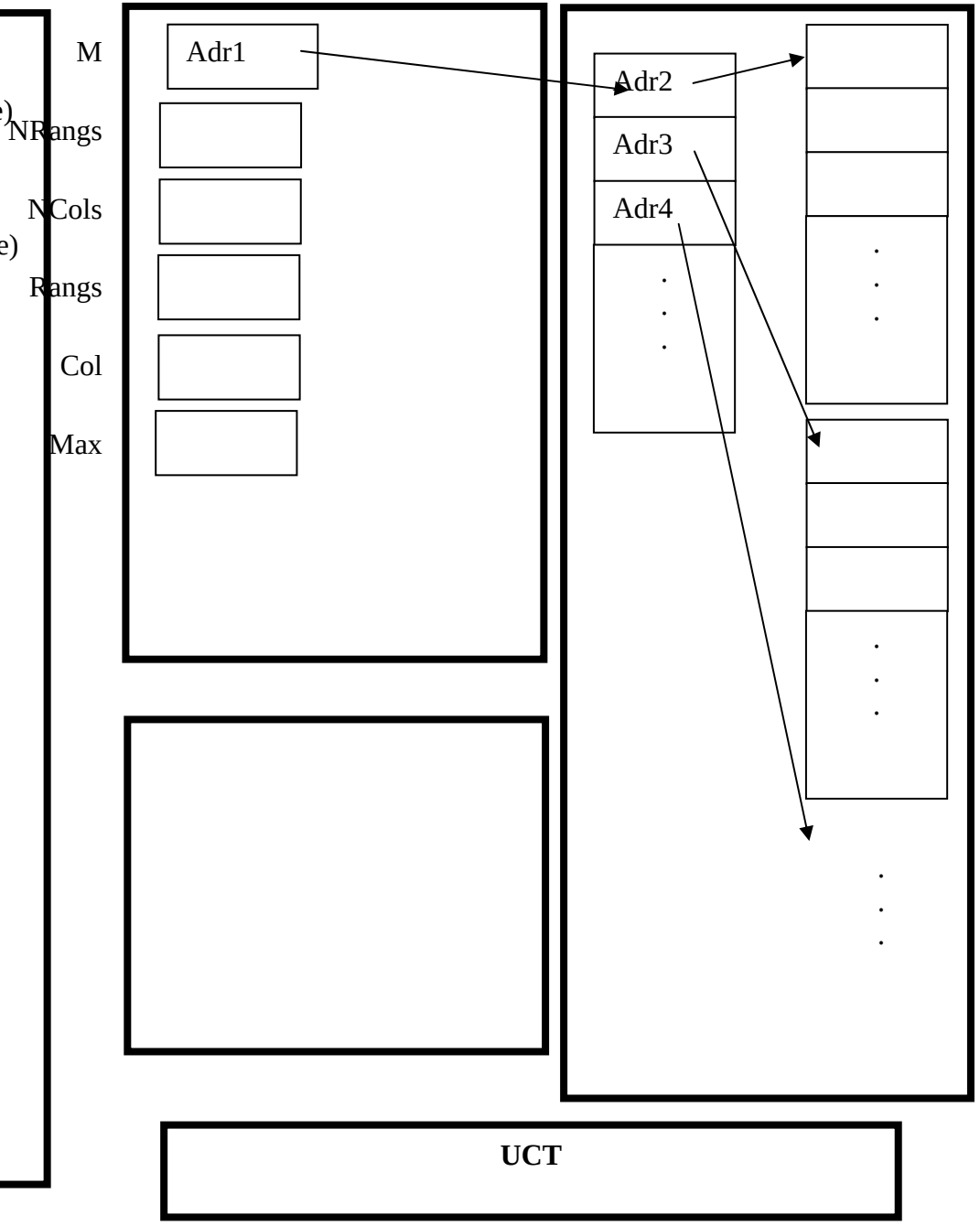
Mémoire de programme **Exercice 9-2 – Valeur maximale dans une matrice** **Mémoire de travail** **Mémoire globale**

DONNÉES: M (référence à une matrice)
NRangs (nombres de rangées dans matrice)
NCols (nombres de colonnes dans matrice)
INTERMÉDIAIRES:
Rang (index de rangée courante)
Col (index de colonne courante)
RÉSULTAT: Max (la valeur maximale dans la matrice)
EN-TÊTE: Max ← TrouveMaxMatrice(M, NRangs, NCols)
MODULE:



Alternative:

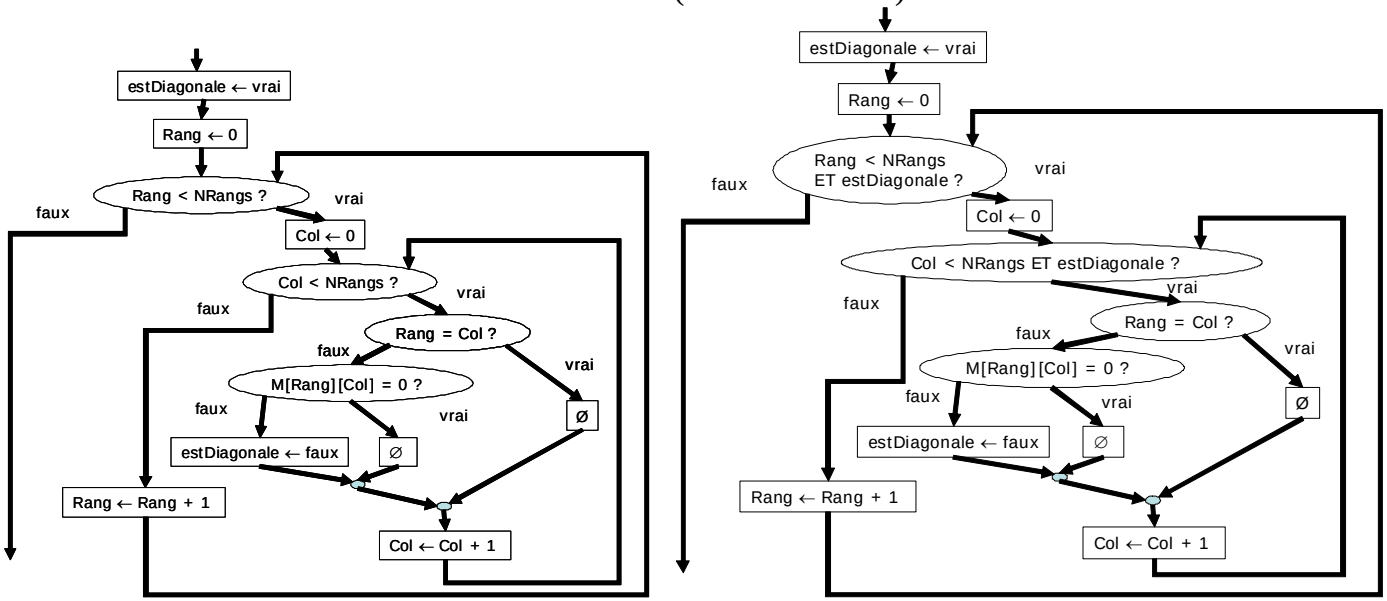




Mémoire de programme **Exercice 9-3 – Algorithme VérifDiag** **Mém de travail** **Mém globale**

DONNÉES: M (référence à une matrice)
NRangs (nombre de rangées dans matrice)
NCols (nombre de colonnes dans matrice)
INTERMÉDIAIRES:
Rangs (index de la rangée courante)
Col (index la colonne courante)
RÉSULTAT: estDiagonale (Booléen: vrai si matrice est diagonale)
EN-TÊTE: estDiagonale ← VérifDiag(M, NRangs, NCols)
MODULE:

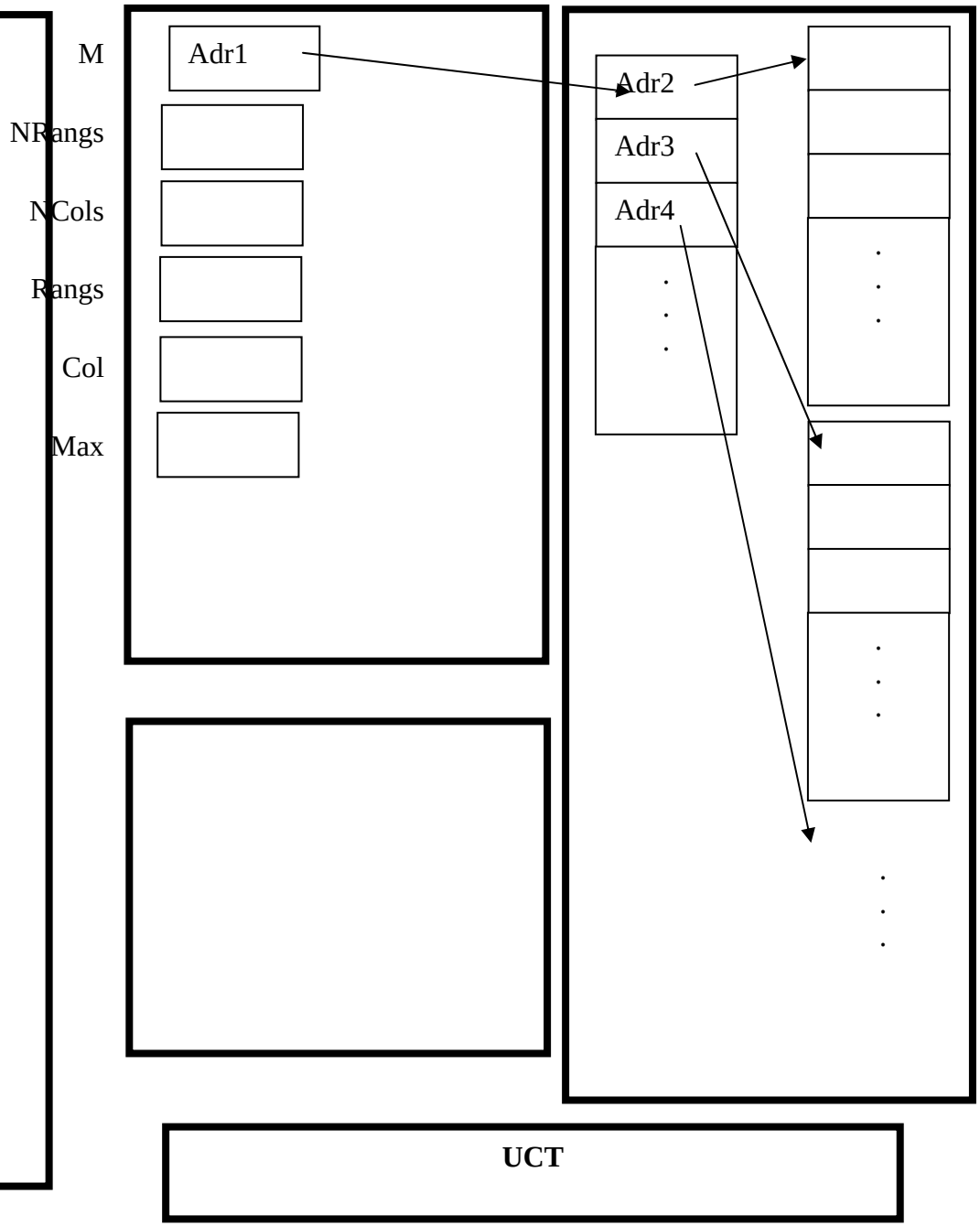
(Version efficace)



UCT

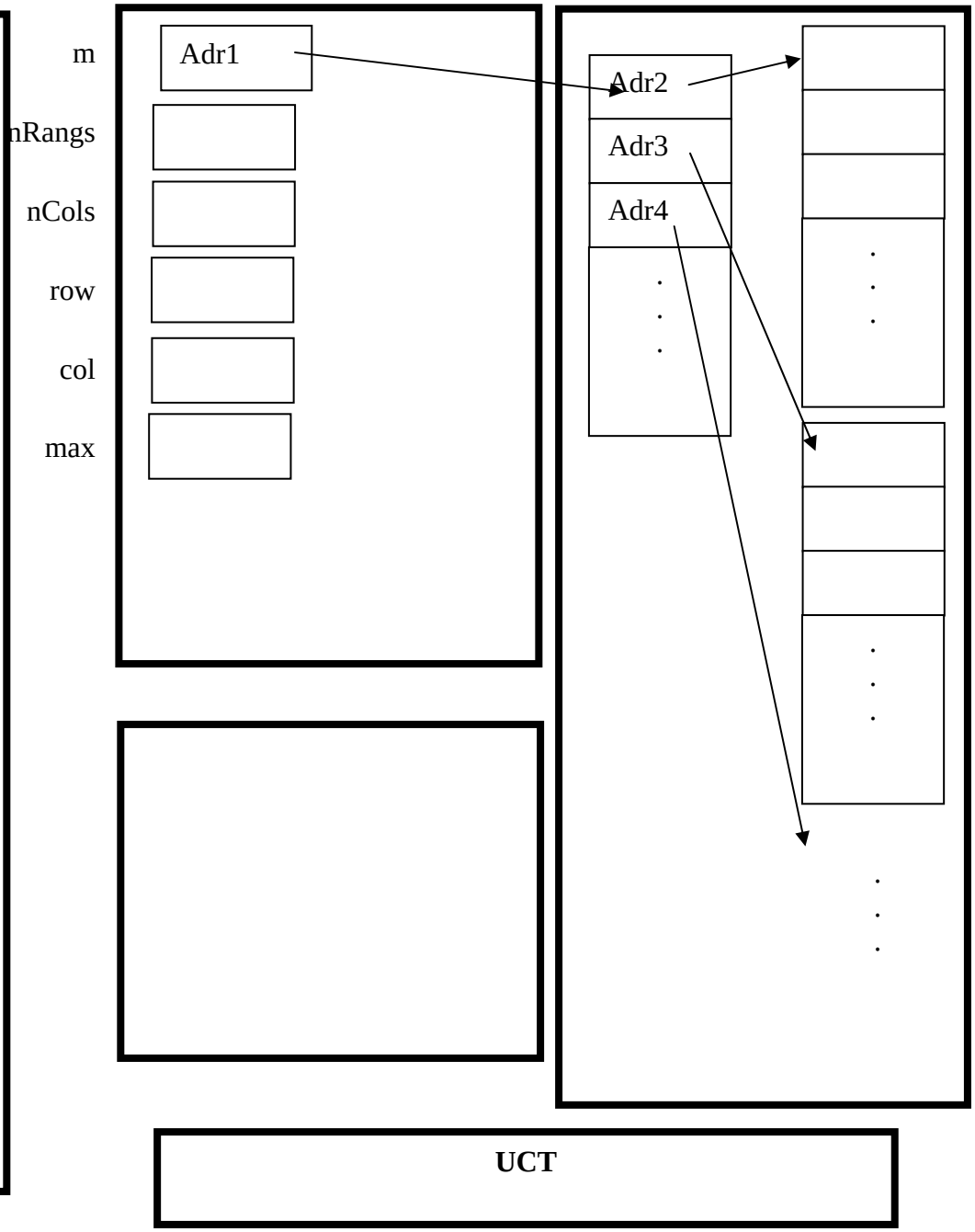
Exercice 9-4 – Valeur maximale en Java


```
// Note:  Integer.MIN_VALUE est la valeur
// entière la plus négative permise pour un
// int Java, et peut être utilisé pour –∞.

public static int maxMatrice (int[][] m,
                     int nRangs, int nCols)
{
   int max = Integer.MIN_VALUE;
     // RÉSULTAT. m[0][0] pourrait être une
    // autre option… si la matrice M n'est
   //  pas vide!
   int rang;   // INTERMÉDIAIRE
   int col;    // INTERMÉDIAIRE
   for ( rang = 0; rang < nRangs;
         rang = rang + 1 )
   {
      for ( col = 0; col < nCols;
            col = col + 1 )
      {
         if ( m[rang][col] > max )
         {
            max = m[rang][col];
         }
         else
         {
            /* ne rien faire */ ;
         }
      }
   }
   return max;
}
```



Mémoire de programme **Exercice 9-4 – Lecture d'une matrice** **Mémoire de travail** **Mémoire globale**

```
public static int[][] litMatrice(
                    int nRangs, int nCols)
{
  int rang; // INT: index pour les rangées
  int col;  // INT: index pour les colonnes
  int[][] m ; // RÉSULTAT: Matrice lue.

  m = new int[nRangs][nCols]; // IMPORTANT!
  for ( rang = 0 ; rang < nRangs ;
        rang = ligne+1 )
  {
     System.out.println(
       "Entrez les valeurs pour la rangée"
                                + ligne );
     for ( col=0; col < nCols ;
           col = col+1)
     {
        m[rang][col] = ITI1520.readInt() ;
     }
  }
  return m;
}
```



Mémoire de programme **Exercice 9-5 – Trouver le vol direct le moins cher** **Mémoire de travail** **Mémoire globale**

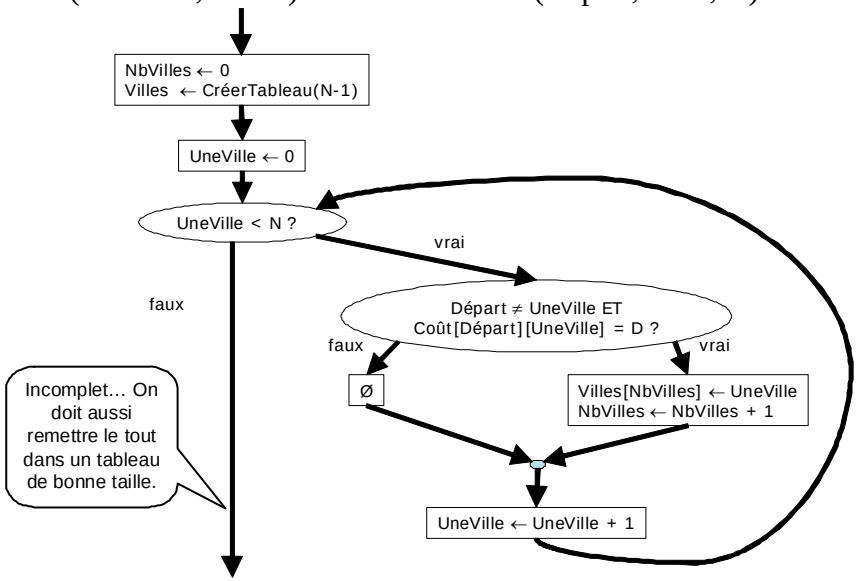
DONNÉES:
- Départ (le numéro de la ville où vous êtes)
- Coût (référence à la matrice de coûts)
- D (votre budget)
- N (le nombre total de villes)

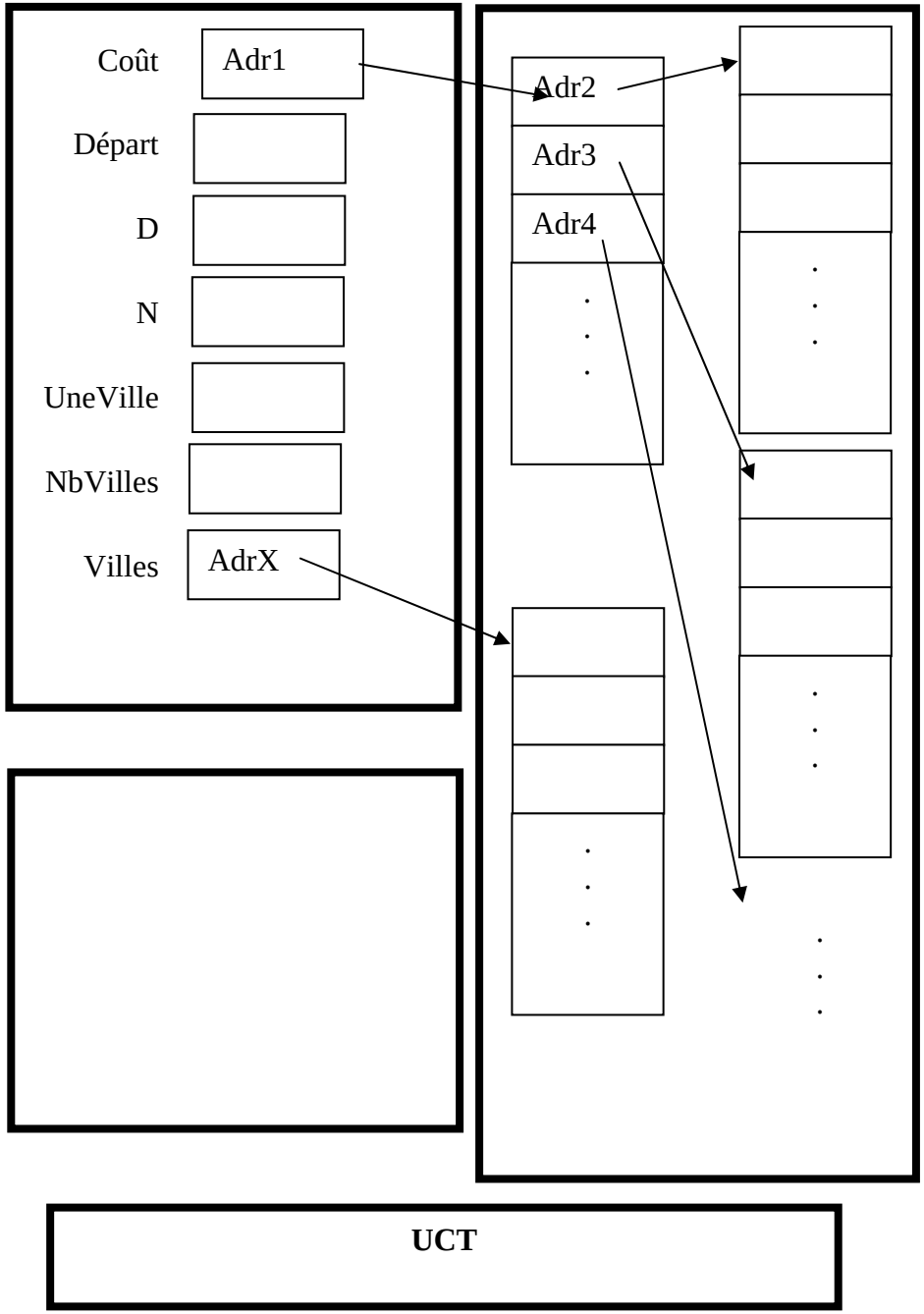
RÉSULTATS:
- Villes (référence à un tableau de villes visitables)

INTERMEDIATE:
- UneVille (la destination présentement considérée)
- NbVilles (le nombre de villes visitables)

EN-TÊTE

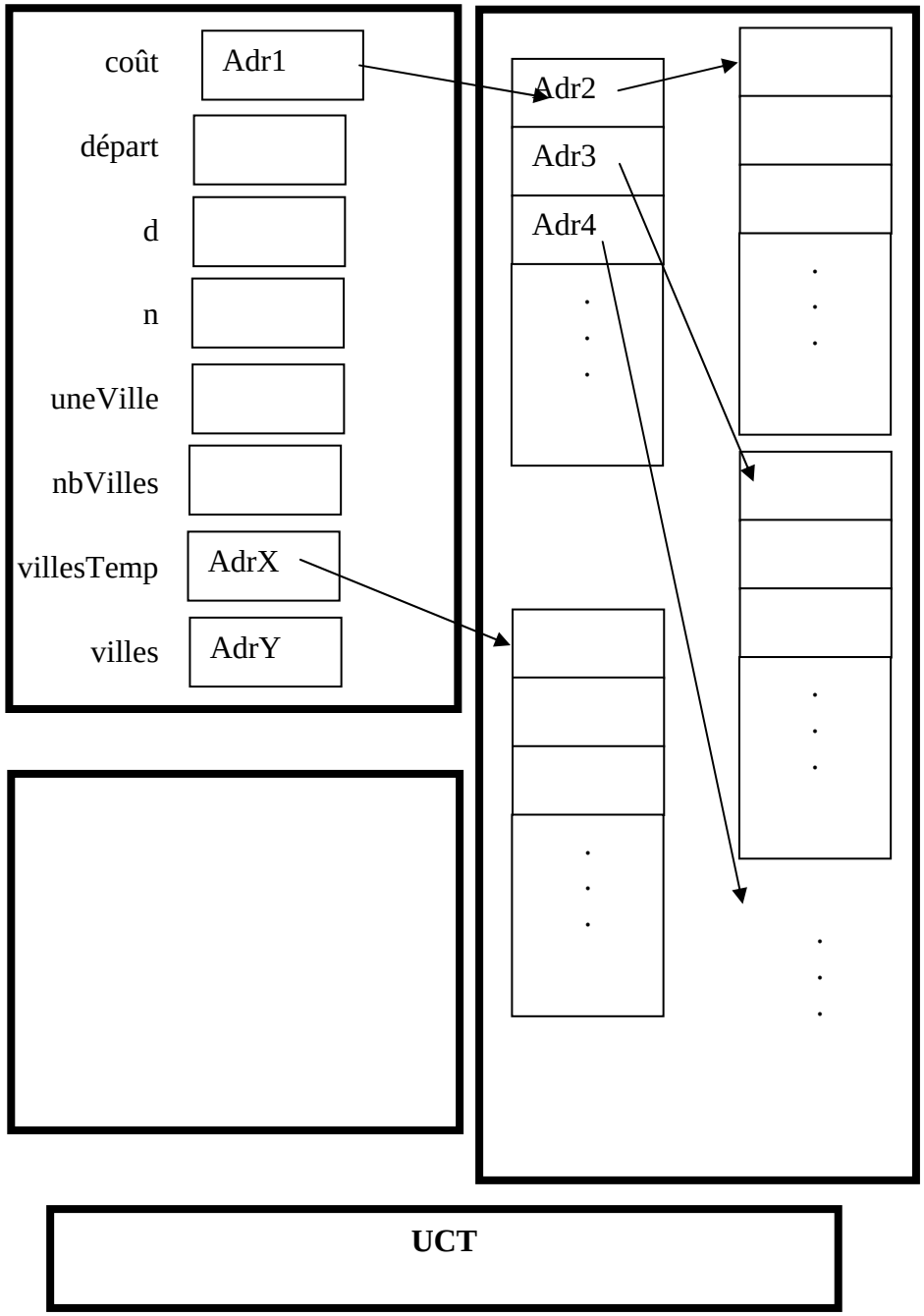
(NbVilles, Villes) ← VolsPasChers (Départ, Coût, D)





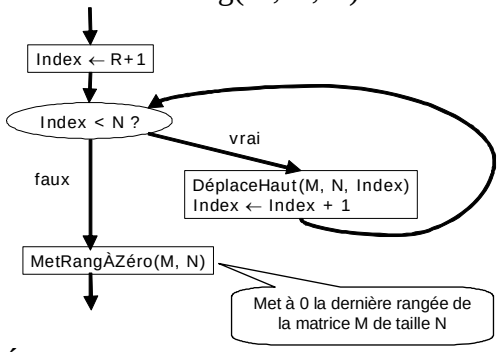
Mémoire de programme **Exercice 9-5 (suite) – Traduction en Java** **Mémoire de travail** **Mémoire globale**

```
public static int[] volsPasCher ( int départ,
                  int[][] coût, int d, int n )
{
   int uneVille; //INT, index de la ville con.
   int nbVilles; //INT, # de villes visitables
   int[] villes; //RÉS, tableau de villes visit

   // MODULE DE L'ALGORITHME
   nbVilles = 0 ;
   villes = new int[n-1];
   for (uneVille = 0; uneVille < n ;
        uneVille = uneVille+1 )
   {
      if ( ( départ != uneVille ) &&
           ( coût[départ][uneVille] <= d ) )
      {
         villes[nbVilles] = uneVille;
         nbVilles = nbVilles + 1;
      }
      else
      {
         /* ne rien faire */ ;
      }
   }
   // Retourner un tableau de bonne longueur.
   // (villes pourrait être trop grand).
   int [] villesTemp = new int[nbVilles];
   for ( uneVille = 0; uneVille < nbVilles;
         uneVille = uneVille + 1 )
   {
      villesTemp[uneVille] = villes[uneVille];
   }
   villes = villesTemp;
   // RÉSULTAT: Maintenant, le résultat peut
   // être retourné…
   return villes;
}
```



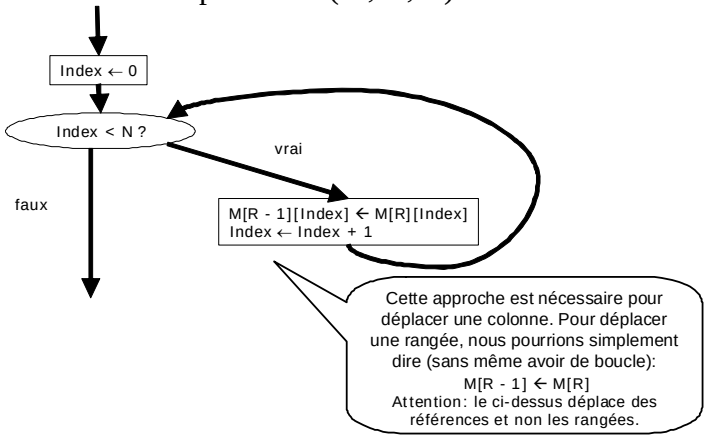
Mémoire de programme **Exercice 9-6–** Algorithme EffaceRang **Mém de travail** **Mém globale**

DONNÉES**:** M *(référence à une matrice carrée)*
N *(nombre de rangées et de colonnes dans matrice)*
R *(numéro de la rangée à effacer)*
RÉSULTAT**:** Aucun
MODIFIÉE**:** M *(rangée R est effacée, les autres rangées sont déplacées, et la dernière rangées est mise à 0)*
INTERMÉDIAIRE**:** Index *(indice de la rangée à déplacer)*
EN-TÊTE : EffaceRang(M, N, R)




```
public static void effaceRang(int [][]m,
                              int n, int r)
{
  int index; // INTERMÉDIAIRE
  for (index = r + 1; index < n;
       index = index + 1)
  {
     déplaceHaut(m, n, index);
  }
  metRangÀZéro(m, n);
}
```

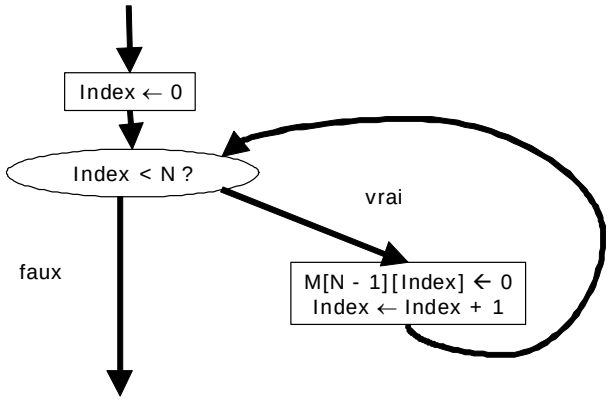
DONNÉES**:** M *(une matrice carrée)*
N *(taille de M)*
R *(numéro de la rangée à déplacer)*
RÉSULTAT**: (**Aucun)
MODIFIÉE**:** M *(rangée R copiée à rangée R-1)*
INTERMÉDIAIRE**:** Index *(indice de la colonne)*
EN-TÊTE: DéplaceHaut(M, N, R)




```
private static void déplaceHaut(int [][] m,
                                int n, int r)
{
  int index; // INTERMÉDIAIRE
  for (index = 0; index < n;
       index = index + 1)
  {
     m[r - 1][index] = m[r][index];
  }
}
```

UCT

DONNÉES: M *(une matrice carrée)*
N *(taille de M)*
RÉSULTAT: Aucun
MODIFIÉE: M *(dernière rangée mise à 0)*
INTERMÉDIAIRE: Index *(index d'une colonne)*
EN-TÊTE: MetRangÀZéro(M, N)




```
private static void metRangÀZéro(int [ ][ ]m,
                                 int n)
{
  int index; // INTERMÉDIAIRE
  for (index = 0; index < n;
       index = index + 1)
  {
      m[n - 1][index] = 0;
  }
}
```

UCT