Labyrinthes (X 2020 MP). Corrigé

1)

```
def creer_File(n) :  return [[0]*n,0,0]
```

2)

```
def ajoute_debut(F,e) :
     n = len(F)
     if F[1] > 0 :
         F[1] = F[1] - 1 ; F[2] = F[2] + 1 ; F[0][F[1]] = e
     else F[1] = n-1 :  F[2] += 1 ; F[0][F[1]] = e
```

3)

```
def retire_debut(F) :
     n = len(F) ; e = F[0][F[1]]
     if F[1] < n-1 :
         F[1] = F[1] + 1 ; F[2] = F[2] - 1
     else :
         F[1] = 0 ; F[2] = F[2] - 1
     return e
```

4) Le code (non demandé est le suivant) :

```
def plus_court_chemin(G,src,dst) :
     # étape 1
     n = len(G) ; distance = [-1]*n ; F = creer_file(n)
     ajouter_debut(F,src) ; distance[src] = 0
     # étape 2
     while F :
     v = retirer_debut(F)
         if v == dst :  return distance[dst]
         for w  in G[v] :
              if distance[w] == -1 :
                   ajouter_fin(F,w) ;  distance[w] = distance[v]+1
```

On note $d(v)$ la distance de v, c'est-à-dire la longueur minimale d'un chemin de la source à v.

On montre que l'algorithme admet les invariants suivants :

(a) Les éléments v_1, v_2, ... , v_p de la file d'attente vérifient : $d(v_1) \leq ... \leq d(v_p) \leq d(v_1) + 1$

(b) Tous les sommets de distance $d(v_1)$ ou moins ont déjà été insérés (et éventuellement retirés)

(c) Toutes les valeurs de `distance` différentes de -1 sont les distances des sommets correspondants.

Cette propriété est vraie lors de l'initialisation (étape 1) car la file ne contient que `src` qui est à distance 0 et c'est le seul sommet de distance 0.

On suppose que cette propriété est vraie avant un passage ; on retire le sommet v_1 et on ajoute les voisins de v_1 non encore insérés w_1, ... , w_q (ceux dont la valeur dans distance vaut -1).

Montrons que les propriétés (a), (b) et (c) sont conservées :

(a) On accède à ces sommets depuis v_1 donc leur distance est au plus $d(v_1)+1$, comme les sommets à distance $d(v_1)$ ont été déjà ajoutés, leur distance est exactement $d(v_1)+1$.
On a donc $d(v_1) \leq ... \leq d(v_p) \leq d(v_1)+1 = d(t_1) = d(t_2) = ... = d(t_q)$.

(b) Le nouveau sommet en début de file est v_2
Si on a $d(v_2) = d(v_1)$ alors tous les sommets de distance $d(v_2)$ au plus ont été insérés par hypothèse.
Si on a $d(v_2) = d(v_1)+1$, alors tous les sommets de distance $d(v_1)$ ont étés traités (et retirés de la file) donc tous les sommets de distance $d(v_2) = d(v_1)+1$ ont été insérés, car ils sont nécessairement voisins d'au moins un somme de distance $d(v_1)$.

(c) Comme voisins de v_1, la distance de chaque t_j est bien $\leq d(v_1)+1$.
Par (b) de l'hypothèse de récurrence, la distance de chaque t_j est $> d(v_1)$.
Donc la distance de chaque t_j vaut effectivement $d(v_1)+1$.
Ainsi la propriétés restent vraies.

Comme le graphe est connexe, le sommet `dst` est atteint (en un nombre d'étapes égal à sa distance) donc la valeur de `distance[dst]` renvoyée est bien la valeur minimale de la longueur d'un chemin de `src` à `dst`.

5)

```
01.  def minimum_monstre(graphe,monstre,src,dst) :
02.       n = len(G) ; distance = [-1]*n ; F = creer_file(n)
03.       ajouter_debut(F,src)
04.      if monstre[src] :  distance[src] = (0,0)
05.      else :  distance[src] = (1,0)
06.      while F :
07.            v = retirer_debut(F)
08.            if v == dst :  return distance[dst]
09.            (m,l) = distance[v]
10.            for w  in G[v] :
11.                 if distance[w] == -1 :
12.                     if monstre[w] :
13.                         ajouter_fin(F,w) ; distance[w] = (m+1,l+1)
14.                     else :
15.                          ajouter_debut(F,w) ; distance[w] = (m,l+1)
16.      return None
```

6) L'algorithme parcourt tous les sommets de la composante connexe.
Dans le cas où il n'existe aucune chemin reliant la source et la destination, la file se retrouve vide à un certain moment, donc la fonction renvoie `None`.