Informatique - MPI

Question 1. On trouve 11 fleurs.
Commentaire – On attend une réponse orale ; si elle est correcte, le jury ne demande même pas de justification.

Question 2. La récolte en une case dépend récursivement de celle de la case du haut et de celle de la case de gauche. La formule générique est donc à adapter lorsqu'on se trouve sur le bord gauche ou le bord haut du champ.

```
int recolte(int champ[m][n], int i, int j){
  /* Cas de base */
  if ( (i == 0) && (j == 0) )
   return champ[0][0];
  if (i == 0)
    return champ[0][j] + recolte(champ,0, j - 1);
  if (j == 0)
    return champ[i][0] + recolte(champ,i - 1, 0);
  /* Cas´general */
  return champ[i][j] + max(recolte(champ,i - 1, j), recolte(champ,i, j - 1));
}
```

Commentaire – Il est attendu des candidats qu'ils soient attentifs aux cas de base. Ils peuvent factoriser la présentation de leur code et l'établissement de la relation de récurrence.

Question 3. On trouve 6 appels à `recolte`.
Commentaire – Plusieurs méthodes sont acceptées pour répondre à cette question ; le candidat peut par exemple compter le nombre d'appels en modifiant la fonction précédente ou à la main en déployant l'arbre d'appels.

Question 4. On calcule d'abord les valeurs des cases sur les bords haut et gauche puis on propage soit en remplissant les lignes de gauche à droite, soit en remplissant les colonnes de haut en bas.
Commentaire – Une réponse orale claire peut suffire. Le candidat peut aussi s'aider d'un schéma indiquant les dépendances entre les différents termes à calculer.

Question 5. Il s'agit d'une traduction des questions 2 et 4. Le code ci-dessous répond aussi à la question 7.

```
int recolte_iterative(int champ[m][n], int i, int j,int fleurs[m][n]){
  int x, y;
  fleurs[0][0] = champ[0][0];
  /* Bord haut */
  for (x = 1; x <= i; x++) {
    fleurs[x][0] = champ[x][0] + fleurs[x - 1][0];
  }
  /* Bord gauche */
  for (y = 1; y <= j; y++) {
    fleurs[0][y] = champ[0][y] + fleurs[0][y - 1];
  }
  /* Autres cases */
  for (y = 1; y <= j; y++) {
    for (x = 1; x <= i; x++) {
      fleurs[x][y] = champ[x][y] + max(fleurs[x - 1][y], fleurs[x][y - 1]);
    }
  }
  deplacements(fleurs,i, j);
  return fleurs[i][j];
}
```

Question 6. On commence par distinguer les cas limites. Puis on appelle récursivement `deplacements` en observant quelle est la case voisine qui avait permi d'obtenir le plus de fleurs.

```
void deplacements(int fleurs[m][n], int i, int j){
  if (i == 0 && j == 0) {
       printf("Case A, ");
       return;
  }
```

```
6   if (i == 0) {
7         deplacements(fleurs,0, j - 1);
8         printf("Aller`a droite, ");
9         return;
10  }
11  if (j == 0) {
12        deplacements(fleurs,i - 1, 0);
13        printf("Descendre, ");
14        return;
15  }
16  if (fleurs[i - 1][j] > fleurs[i][j - 1]) {
17     deplacements(fleurs,i - 1, j);
18     printf("Descendre, ");
19  }
20  else {
21     deplacements(fleurs,i, j - 1);
22     printf("Aller`a droite, ");
23  }
24 }
```

Commentaire – On attend bien l'affichage des déplacements, et pas uniquement leur calcul. La mise en forme de l'affichage (avec sauts de lignes, tabulations, ...) est en revanche libre.

Question 7. Voir le code proposé à la question 5.
Commentaire – Le jury est attentif à l'endroit où l'appel à cette fonction est effectué.