DS d'informatique N°2

Tournois et pronostics X–PSI/PT–2014 — Corrigé

Partie I : Tournois

Question 1

Les conditions (3), (4) et (5) peuvent s'exprimer de la manière suivante :

— **(3)** : Lors d'un vrai match, un joueur ne peut pas jouer contre lui-même.

— **(4)** : Les deux participants du match k sont issus d'un match qui a eu lieu avant le match k.

— **(5)** : Le gagnant de n'importe quel match qui n'est pas la finale, participe à un seul match ultérieur.

Question 2

```
def EstVraieCondition5(T,k):
  res = False
  # On cherche k dans les couples de joueurs des matchs suivants
  for kp in range(k+1, 2*n-1):
    if T[kp][0] == k or T[kp][1] == k:
      if res == False:
        # On n'a pas encore trouvé k
        res = True
      else:
        # On l'avait déjà trouvé, c'est donc le deuxième
        return False
  return res
```

La boucle principale s'exécute dans le pire des cas $2n-k-2$ fois, la complexité de cette fonction est donc $O(2n-k)$.

Question 3

```
def EstUnTournoi(T):
  for k in range(2*n-1):
    # Condition 1
    if T[k][1]<T[k][0]:
      return False
    # Condition 2
    if k<n and (T[k][0]!=k or T[k][1]!=k):
      return False
    # Condition 3
    if k>=n and T[k][0] == T[k][1]:
      return False
    # Condition 4
    if k>=n:
      if T[k][0] < 0 or T[k][0] >= k:
        return False
      if T[k][1] < 0 or T[k][1] >= k:
        return False
    # Condition 5
    if not EstVraieCondition5(T,k):
      return False
  return True
```

La boucle principale s'exécute $2n-1$ fois, et la complexité de la fonction `EstVraieCondition5` est $O(2n-k)$. Donc la complexité de cette fonction est $O(n^2)$

Question 4

```
def EstUnOracle(O):
  for i in range(n):
    if O[i][i] != True:      # Diagonale
      return False
    for j in range(i):
      if O[i][j] == O[j][i]:
        return False
  return True
```

Cette fonction est de complexité quadratique $O(n^2)$.

Question 5

```
def Vainqueur(T,O):
  v = []    # Liste des vainqueurs de chaque match
  for i in range(n):
    v.append(i)   # Matchs triviaux du début
  for k in range(n, 2*n-1):
    j1 = v[T[k][0]]    # Les joueurs du match sont les vainqueurs
    j2 = v[T[k][1]]    # des matchs précédents
    if O[j1][j2] :
      v.append(j1)   # Le premier joueur gagne le match
    else
      v.append(j2)   # Le second joueur gagne
  return v[-1] #Le vainqueur du tournoi est le vainqueur du dernier match
```

Partie II. Vainqueurs Potentiels

Question 6

Un oracle qui n'a qu'un vainqueur potentiel est un oracle pour lequel l'un des joueurs gagne tous ses matchs contre tous les autres joueurs. Ce joueur ne peut pas être battu donc aucun autre joueur ne peut gagner le tournoi.

Par exemple l'oracle suivant n'a comme vainqueur potentiel que le joueur 0.

	0	1	2
0	T	T	T
1	F	T	T
2	F	F	T

Tout oracle qui n'a pas cette caractéristique possède plusieurs vainqueurs potentiels. Par exemple avec l'oracle suivant :

	0	1	2
0	T	T	F
1	F	T	T
2	T	F	T

Le joueur 0 gagne si le joueur 1 commence par battre le joueur 2, puis est battu par le joueur 0. Mais si le premier match est entre le joueur 0 et le joueur 2, c'est le joueur 1 qui l'emporte en finale contre le joueur 2.

Question 7 Pour trouver un vainqueur potentiel, on va trouver le vainqueur d'un tournoi assez simple : le joueur 0 rencontre le joueur 1 puis le gagnant rencontre le joueur 2, etc.

```
def UnVainqueur(O):
  v = 0  # vainqueur courant
  for i in range(1,n):
    if O[i][v] :   # si le joueur i gagne contre le vainqueur courant
      v = i   # i est le nouveau vainqueur courant
  return v
```

Question 8

```
def vainqueursPotentiels(O):
  joueurs = []              # Tous les joueurs
  v = UnVainqueur(O)
  vainqueurs = [False]*n
  vainqueurs[v] = True
  pile = [v]   # Les vainqueurs potentiels à traiter
  while len(pile)>0:        # Tant qu'il y a des vainqueurs à traiter
    v = pile.pop()          # on traite le premier vainqueur de la pile
    for i in range(n):      # On cherche des joueurs qui gagne un match
      if (not vainqueurs[i]) and O[i][v]:
        pile.append(i) # On les ajoute à la pile
        vainqueurs[i] = True
  return vainqueurs
```

Dans le meilleur des cas, tous les joueurs gagnent un match contre le premier vainqueur potentiel et la complexité de cette fonction est $O(n)$. Dans le pire des cas, il n'y a a chaque fois qu'un joueur qui gagne un match contre le vainqueur potentiel en cours de traitement et la complexité de la fonction devient $O(n^2)$

Partie III. Pronostics

Question 9

```
def PremierJoueurPossible(T,k):
    pile = [T[k][0]]
    v = [False]*n
    while(len(pile)>0):
        m = pile.pop()
        if m<n:                 # C'est un match trivial, on ajoute le joueur à la liste
            v[m] = True
        else:                   # Sinon on empile les deux matchs qui amènent à ce match
            pile.append(T[m][0])
            pile.append(T[m][1])
    return v
```

Question 10

L'écriture de la fonction découle directement de l'énoncé de la question.

```
def MiseAJour(T,P,k,V):
    for i in range(n):
        if PremierJoueurPossible(T,k):
            S = 0
            for j in range(n):
                if SecondJoueurPossible(T,k):
                    S += V[j]*P[i][j]
            V[i] = V[i]*S
        if SecondJoueurPossible(T,k):
            S = 0
            for j in range(n):
                if PremierJoueurPossible(T,k):
                    S += V[j]*P[i][j]
            V[i] = V[i]*S
```

Question 11

Il suffit d'initialiser les probabilités à 1 et de faire la mise à jour après tous les matchs.

```
def ChancesDeVictoire(T,P):
  V = [1]*n
  for k in range(n,2*n-1):
      MiseAJour(T,P,k,V)
  return V
```

* *
*