Épreuve d'informatique de l'X - 2014 - PT/PSI

Tournois et Pronostics

Partie I. Tournois

Question 1 La condition (3) s'interprète comme « *les $n-1$ derniers éléments de la représentation d'un tournoi sont des matchs réels entre deux joueurs (ou vainqueurs) différents* ».

La condition (4) s'interprète comme « *les $n-1$ derniers éléments de la représentation d'un tournoi sont des matchs non triviaux entre deux joueurs ou vainqueurs de matchs précédents. Les joueurs du match ne peuvent être les vainqueurs d'un match qui n'a pas encore eu lieu.* ».

La condition (5) s'interprète comme « *les n joueurs et les vainqueurs des $n-2$ premiers matchs réels participent à un match et ne peuvent participer à un autre match que s'ils sont vainqueurs.* ».

Question 2 On parcourt les éléments d'indice $k' > k$ dans le tableau et on compte le nombre de fois k appartient à l'ensemble $\{T[k'][1], T[k'][2]\}$. Si ce nombre est égal à 1, on retourne `True` et `False` sinon (ligne 6).

```
1 def EstVraiCondition5(T,k):
2     compteur = 0
3     for i in range(k+1,2*n):
4         if T[i][1] == k or T[i][2] == k:
5             compteur += 1
6     return(compteur==1)
```

Dans la boucle des lignes 3 à 5, on effectue au plus $2n-1-k$ additions et $4n-2-2k$ comparaisons, la complexité est en $O(2n-k)$.

(PSI*)

Question 3 On vérifie que les cinq conditions sont bien vérifiées. On remarque que si la condition (2) est vérifiée, il n'y a pas besoin de vérifier la condition (1) pour $1 \leqslant k \leqslant n$.

```
1  def EstUnTournoi(T):
2      for k in range(1,n+1):    # condition 2
3          if T[k][1] != k or T[k][2] != k:
4              return(False)
5      for k in range(n+1,2*n): # condition 1 et 3
6          if T[k][1] >= T[k][2]:
7              return(False)
8      for k in range(n+1,2*n): # condition 4
9          if T[k][2] >= k:
10             return(False)
11     for k in range(1,2*n-1): # condition 5
12         if not(EstVraiCondition5(t,k)):
13             return(False)
14     return(True)
```

On effectue au plus $2n$ comparaisons dans la première boucle (ligne 2 à 4) et $2(n-1)$ comparaisons dans les deuxième et troisième boucles (lignes 5 à 10), ce qui nous donne une complexité linéaire pour les lignes 2 à 10. Le coût de `EstVraiCondition5` étant en $O(2n-k)$, le coût de la boucle des lignes 11 à 13 est en $O(\sum_{k=1}^{2n-2} 2n-k) = O\left(\frac{(2n-2)(2n-1)}{2}\right) = O(n^2)$. On en déduit que :

la complexité globale de la fonction `EstUnTournoi` est quadratique (en $O(n^2)$).

Question 4 On vérifie que pour tous les couples (i,j) tel que $i \neq j$, $O[i][j]$ et $O[j][i]$ sont distincts et que les $O[i][i]$ sont tous égaux à `True`. Si ce n'est pas le cas, on arrête la double boucle et on retourne `False`.

```
1 def EstUnOracle(O):
2     for i in range(1,n+1):
3         if O[i][i]== False
4             return(False)
5         for j in range(i+1,n+1):
6             if O[i][j] == O[j][i]:
7                 return(False)
8     return(True)
```

On effectue au plus $\sum_{i=1}^{n} n-i = \frac{n(n-1)}{2}$ comparaisons.

La complexité de la fonction `EstUnOracle` est quadratique (en $O(n^2)$).

Question 5 Voici une première solution itérative qui remplit le tableau des résultats de chaque match. On commence par donner les résultats des matchs triviaux (boucle des lignes 3 à 4), puis on remplit le reste du tableau des résultats (boucle des lignes 5 à 11) à l'aide des résultats des matchs précédents et des prédictions de l'oracle.

```
1  def Vainqueur(T,O):
2      tableau_resultat = np.zeros(2*n, dtype=int)
3      for k in range(n+1):
4          tableau_resultat[k] = k
5      for k in range(n+1,2*n):
6          joueur1 = tableau_resultat[T[k][1]]
7          joueur2 = tableau_resultat[T[k][2]]
8          if O[joueur1][joueur2]:
9              tableau_resultat[k] = joueur1
10         else:
11             tableau_resultat[k] = joueur2
12     return(tableau_resultat[2*n-1])
```

Voici une solution (plus élégante ?) utilisant une fonction auxiliaire récursive et n'utilisant pas de tableau qui calcule le résultat d'un match en fonction des résultats des matchs précédents. La fonction auxiliaire « `VainqueurMatch(k)` » retourne le vainqueur du match k. Si $k \leqslant n$, k est le vainqueur (match trivial), sinon, on détermine les participants au match réel k (lignes 6 et 7), puis le vainqueur du match k (lignes 8 à 11).

```
1  def Vainqueur(T,O):
2      def VainqueurMatch(k):
3          if k <= n:
4              return(k)
5          else :
6              joueur1 = VainqueurMatch(T[k][1])
7              joueur2 = VainqueurMatch(T[k][2])
8              if O[joueur1][joueur2]:
9                  return(joueur1)
10             else:
11                 return(joueur2)
12     return(VainqueurMatch(2*n-1))
```

Dans les deux cas, le nombre d'opérations est majoré, à une constante multiplicative près, par $2n-1$. Pour chaque match, on effectue un nombre limité d'opérations ne dépendant pas du match étudié.

la complexité en en $O(n)$.

Partie II. Vainqueurs Potentiels

Question 6 Si on considère un oracle tel que, par exemple, le premier joueur gagne tous ses matchs contre les autres adversaires alors le premier joueur est le seul vainqueur possible.

Si on considère un oracle tel que le premier joueur gagne tous ses matchs sauf contre le deuxième joueur et tel que le deuxième joueur perd tous ses matchs sauf contre le premier alors :

- le premier joueur ne sera pas le vainqueur s'il rencontre le deuxième joueur lors de son premier match
- le premier joueur sera le vainqueur si le deuxième joueur rencontre tout autre joueur que le premier lors de son premier match

Question 7 On considère le vainqueur du tournoi qui vérifie, $T[n+1][1] = 1$, $T[n+1][2] = 2$ et pour tout $n+2 \leqslant k \leqslant 2n-1$, $T[k][1] = k-n$ et $T[k][2] = k-1$. Le premier match non trivial oppose les premier et deuxième joueurs et ensuite de suite, le k-ème match opposant le $(k+1)$-ème joueur avec le vainqueur du match précédent([1]).

```
1  def UnVainqueur(O):
2      Vainqueur = 1
3      for k in range(2,n+1):
4          if O[k][Vainqueur]:
5              Vainqueur = k
6      return(Vainqueur)
```

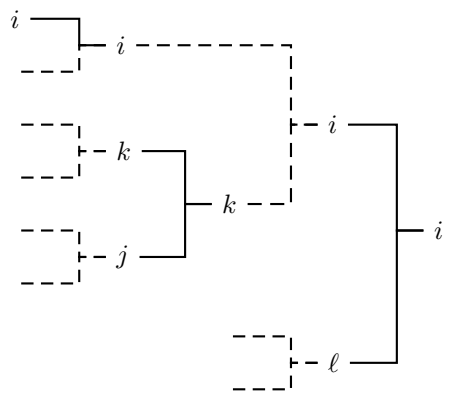
Dans la boucle des lignes 3 à 5, on effectue $(n-1)$ comparaisons.

La complexité de la fonction `UnVainqueur` est en $O(n)$.

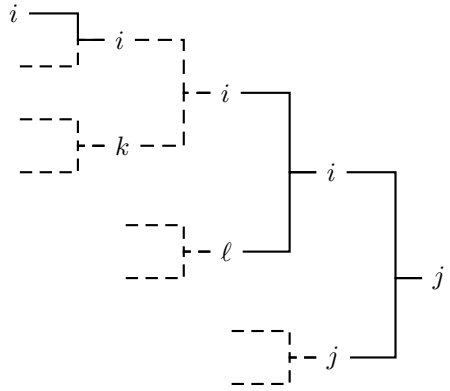
Question 8 Si i est un vainqueur potentiel de O, et que O prédit que j remporte le match entre i et j, c'est qu'il existe un tournoi pour lequel i est vainqueur et donc dans lequel i ne rencontre par j. On considère la série de matchs qui mène jusqu'à la dernière victoire de j. Cette série de matchs en assimilable à un tournois t_1. Dans le tournoi initial, le joueur j perd contre un certain joueur k avec $k \neq i$. on remplace le match faisant se rencontrer i et k par le match faisant se rencontrer i et j après qu'il ait rencontré le gagnant de son dernier match.

1. On aurait pu créer le tableau T du match correcpondant et faire appel à la fontion `Vainqueur` de la question 5.

On part du tournois suivant qui voit la victoire du joueur i et l'échec du joueur j (les lignes pleines représentent un match et les lignes en pointillés peuvent représenter une succession de matchs) :



On transforme le tournois précédent en le tournoi ci-dessous : on supprime le match perdu par j et on ajoute un match faisant s'affronter le vainqueur du dernier match remporté par i (donc i) et le vainqueur du dernier math remporté par j (donc j)



Le tournoi ainsi construit permet au joueur j d'être le gagnant du tournoi.

Question 9 Considérons un tournoi T et l'ensemble

$$A = \{i \in [\![1, 2n-1]\!] \mid \text{ le match } i \text{ est gagné par un élément de } X\}.$$

L'ensemble A est une partie non vide majorée de $\mathbb{N}$ donc admet un plus grand élément i_0. Supposons que $i_0 \neq 2n+1$. Par définition, le match i_0 est gagné par un élément de X. D'après les propriétés de T, il existe un unique entier $k > i_0$ tel que $i_0 \in \{T[k][1], T[k][2]\}$. Soit j_0 tel que $\{i_0, j_0\} = \{T[k][1], T[k][2]\}$.

- Si le vainqueur du match j_0 est un élément de X alors le vainqueur du match k est aussi un élément de X et donc $k \in A$ ce qui contredit la maximalité de i_0,
- Si le vainqueur du match j_0 n'est pas un élément de X alors il va perdre contre le vainqueur du match i_0 qui est dans X et donc $k \in A$ ce qui contredit la maximalité de i_0,

L'hypothèse $i_0 < 2n-1$ est donc fausse, on a donc $i_0 = 2n-1$, c'est-à-dire que le tournoi T est remporté par un élément de X.

Tous les vainqueurs potentiels de O sont dans X

Question 10 On commence par créer un vecteur rempli de `False` (ligne 2). On crée un pile contenant le numéro d'un vainqueur potentiel (ligne 3). Tant que la pile n'est pas vide, on dépile un élément x. Si cet élément n'a pas été déclaré comme un vainqueur potentiel alors on le rajoute aux vainqueurs potentiels (ligne 7), puis on ajoute à la pile tous les joueurs plus forts que le joueur x et qui n'ont pas été encore déclarés comme des vainqueurs potentiels (boucle des lignes 8 à 10).

```
1  def vainqueursPotentiel(O):
2      v = np.zeros(n+1, dtype=bool)
3      pile = [UnVainqueur(O)]
4      while pile != [] :
5          x = pile.pop()
6          if not(v[x]):
7              v[x] = True
8              for k in range(1,n+1):
9                  if not(v[k]) and O[k][x]:
10                     pile.append(k)
11     return(v)
```

La complexité est en $O(n^2)$, on va parcourir au plus n fois une ligne du tableau de l'oracle.

Partie III. Pronostics

Question 11 La fonction `PremierJoueurPossible` retourne un tableau de booléens représentant les joueurs susceptibles d'avoir gagné le match $T[k][1]$. La fonction `VainqueursPossibles` est récursive et modifie le tableau de booléens L représentant les joueurs susceptibles de participer au combat k. Si $1 \leqslant k \leqslant n$, seul le joueur k convient. Si $k > n$, on recherche les premier et deuxième joueurs possibles avec les appels récursifs des lignes 7 et 8.

Dans la fonction `PremierJoueurPossible`, on a :

- Si $k \leqslant n$, seul le joueur k convient (lignes 13 et 14).
- Si $k > n$, le premier joueur possible est l'un des vainqueurs du match $T[k][1]$.

La commande `np.zeros(n+1, dtype=bool)` retourne un tableau de longueur $(n+1)$ (indexé de 0 à n) et rempli de `false`.

```
1  def VainqueursPossibles(T,k,L):
2      if k <= n:
3          L[k] = True
4      else:
5          match1 = T[k][1]
6          match2 = T[k][2]
7          VainqueursPossibles(T,match1,L)
8          VainqueursPossibles(T,match2,L)
9
10 def PremierJoueurPossible(T,k):
11     L = np.zeros(n+1, dtype=bool)
12     if k <= n:
13         L[k] = True
14         return(L)
15     else:
16         joueur = T[k][1]
17         VainqueursPossibles(T,joueur,L)
18         return(L)
```

Question 12

On commence par créer un tableau temporaire rempli de zéros (ligne 2). Lignes 2 et 3, on récupère la liste des premiers joueurs possibles et la liste des seconds joueurs possibles sous forme de tableaux de booléens. Dans la boucle des lignes 5 à 19, on calcule la chance que le joueur i a de remporter le match k en appliquant les formules données par l'énoncé suivant que i est un premier joueur possible du match k (lignes 5 à 11) ou un second joueur possible du match k (ligne 12 à 17) ou ne peut pas participer au match k (ligne 19). On met à jour le tableau V dans la boucle des lignes (20 à 21).

```
1  def MiseAJour(T, P, k, V ):
2      Vtemp = np.zeros(n+1)
3      Premier = PremierJoueurPossible(T,k)
4      Second = SecondJoueurPossible(T,k)
5      for i in range(1,n+1):
6          if Premier[i] :
7              S = 0
8              for j in range(1,n+1):
9                  if Second[j]:
10                     S += V[j]*P[i][j]
11             Vtemp[i] = V[i]*S
12         elif Second[i]:
13             S = 0
14             for j in range(1,n+1):
15                 if Premier[j]:
16                     S += V[j]*P[i][j]
17             Vtemp[i] = V[i]*S
18         else:
19             Vtemp[i] = V[i]
20     for i in range(1,n+1):
21         V[i] = Vtemp[i]
```

Question 13

Pour la fonction `ChancesDeVictoire`, il suffit de partir d'un vecteur rempli de 1 puis d'appliquer la fonction `MiseAJour` pour tous les matchs k pour k compris entre $n+1$ et $2n-1$.

```
1  def ChancesDeVictoire(T, P):
2      V = np.ones(n+1)
3      for k in range(n+1,2*n):
4          MiseAJour(T, P, k, V)
5      return(V)
```