

CALCULE DU NOMBRE DE POINTS
D'ARTICULATION DANS UN GRAPHE NON
ORIENTÉ

RAPPORT DE TP2

November 25, 2018

MAHAMDI Mohammed
BOUKABENE Randa

Table des Matières

1	Introduction	3
1.1	Approche de solution	3
1.1.1	La solution triviale	3
1.1.2	La solution optimale choisi	4
2	Implémentation de la solution	5
2.1	Processus	6
2.2	Calcul de la complexité	9
2.3	Les outils utilisés	9
3	Conclusion	10

Table des Figures

2.1	Saisie de le nombre du neuds	6
2.2	Saisie des données concernant le graphe	7
2.3	voir le nombre de points d'articulation	8
2.4	Les points d'articulation sont coloré en rouge	8

Chapter 1

Introduction

Dans un graphe non orienté, un sommet s'appelle un point d'articulation s'il est supprimé et que tous les arcs qui lui sont associés , le nombre de composantes connexes va augmenter.

Les points d'articulation représentent les vulnérabilités d'un réseau connecté . Ils sont utiles pour concevoir des réseaux fiables.

1.1 Approche de solution

1.1.1 La solution triviale

Soit G un graphe à n sommets et m arêtes. Un algorithme trivial de complexité d'ordre $O(nm)$ est le suivant :

$a =$ nombre de composantes connexes de G (déterminé à l'aide de DFS)

```
Pour chaque sommet  $v$  de  $V$  ayant des aretes incidentes
    retirer  $v$  de  $V$ 
     $b =$  nombre de composantes connexes de  $G$  une fois  $v$  elimine
    si  $b > a$ 
         $v$  est un PA de  $G$  # Afficher  $v$  ou incrementer un compteur
    remettre en place  $v$ 
```

mais on veut pas une complixité de $O(nm)$.

1.1.2 La solution optimale choisi

L'idée est d'utiliser DFS (Depth First Search). Dans le parcours DFS, nous suivons les sommets sous forme d'arbre. un sommet u est le parent d'un autre sommet v , si v est découvert par u (évidemment, v est un adjacent de u dans le graphe). un sommet u est un point d'articulation si l'une des deux conditions suivantes est vraie:

1. u est la racine de DFS et il a au moins deux fils.
2. u n'est pas la racine de l'arbre DFS et u a un fils v tel qu'aucun sommet du sous-arbre qui a v comme racine avec v n'a pas d'arc de retour de l'un des parents de u .

Chapter 2

Implémentation de la solution

On a créer une classe Graph qui contient les attributs d'un graph (nombre du noeuds,le variable temps qu'on aura besoin au future et le nombre de points d'articulation) avec les deux fonctions suivants :

```
find_Articulation_Points(self):
    DFS(self , node , visited , articulation_points , parent , low , discovery_time):
```

la première fonction **find_Articulation_Points()** sert à initialiser les paramètres , et qui appelle récursivement la deusième fonction **DFS(self, node, visited, articulation_points, parent, low, discovery_time)**.

Dans la premiere fonction on initialise le nombre de points d'articulation et le temp de découvert à zéro, et on parcours l'ensemble du noeuds , si la noeud N n'est pas visité on exucute la fonction DFS(..,N) (N est un paramètre)

pour la deusième fonction est la fonction DFS mais elle est modifiée,pour garder la trace des parents ,et savoirs'il existe un arc de retour (Back edge) ou non ,pour vérifie les deux conditions cités précédament.

2.1 Processus

1. On lit le nombre du neuds :

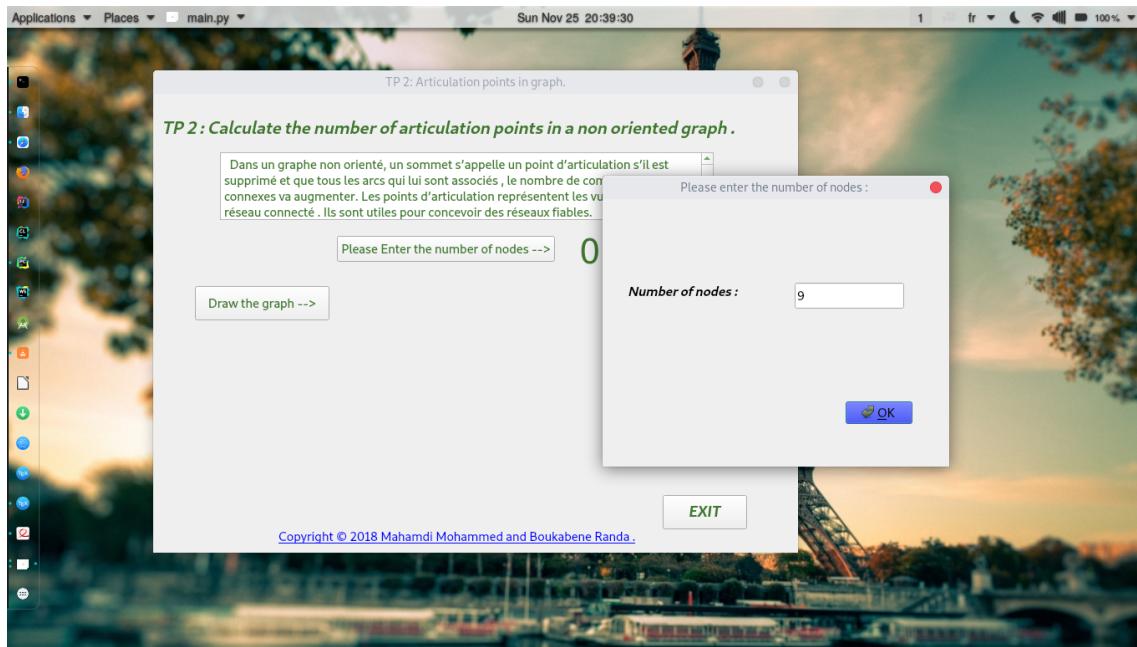
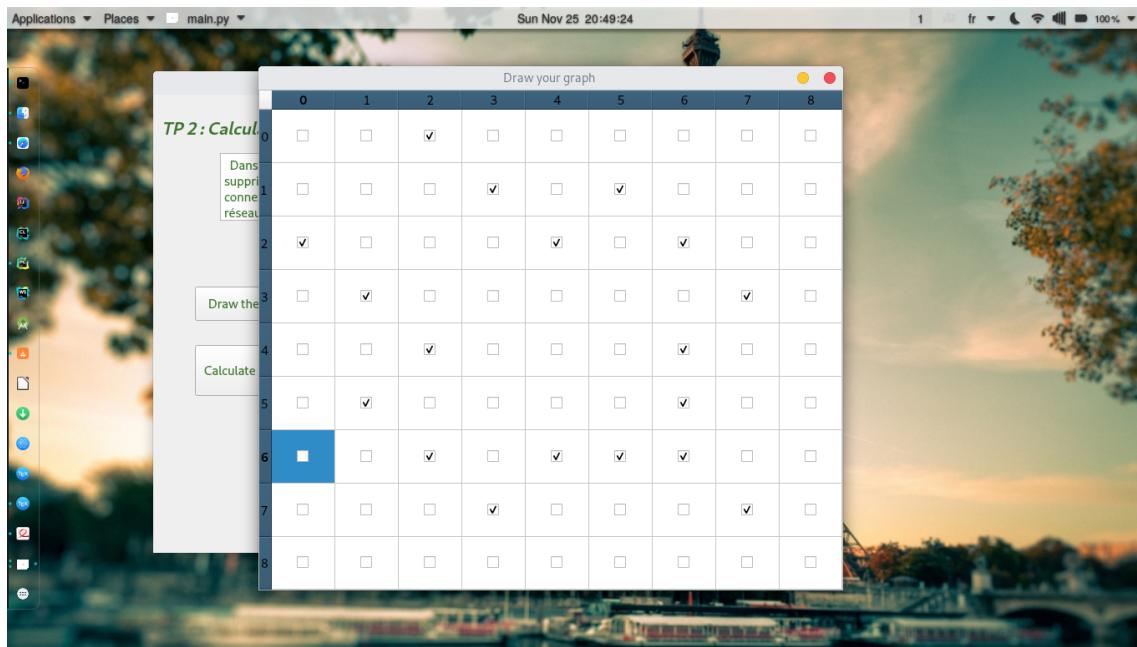
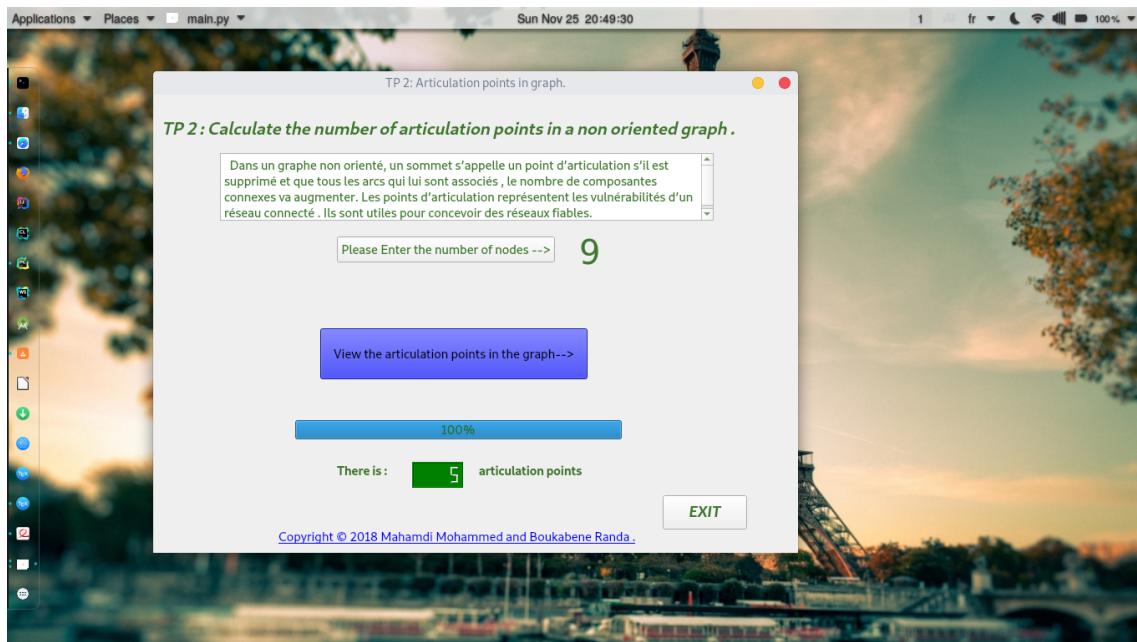


Figure 2.1: Saisie de le nombre du neuds

2. On lit le graphe sous forme d'une matrice comme le montre la figure suivante :

**Figure 2.2:** Saisie des données concernant le graphe

3. Calcule du nombre de points d'articulation:

**Figure 2.3:** voir le nombre de points d'articulation

4. Voir les points d'articulation dans le graphe (pour la vérification) :

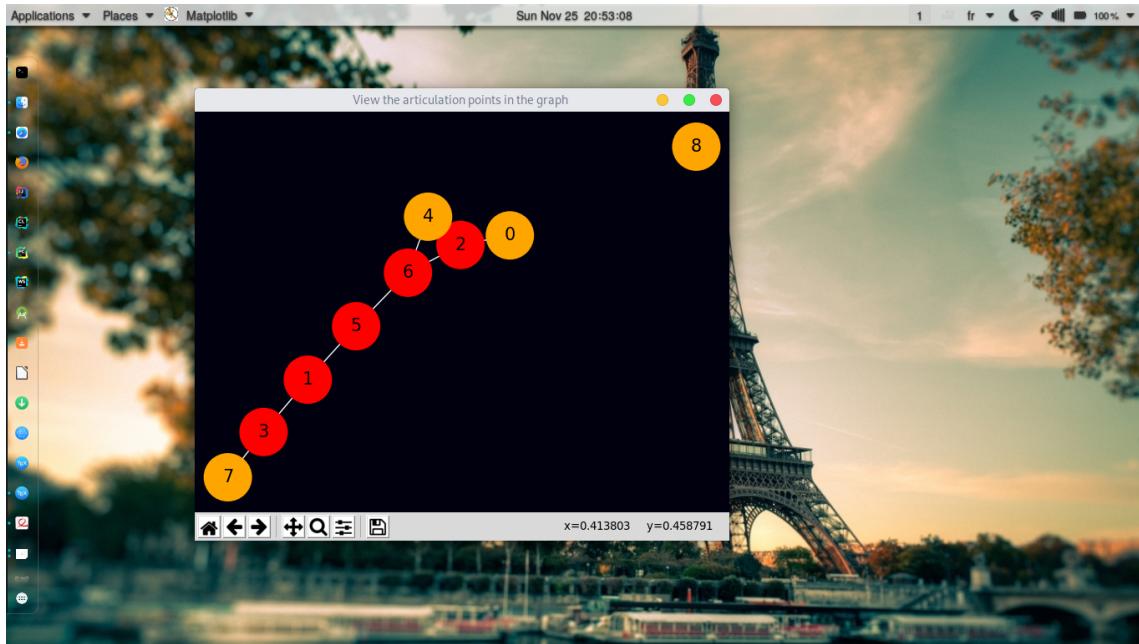


Figure 2.4: Les points d'articulation sont coloré en rouge

2.2 Calcul de la complexité

La complexité de l'algorithme : $O(n+m)$.

preuve : On a implémenter le graphe à l'aide d'une liste d'adjacents , tout les nœuds sont visité une seule fois et chaque arc apparaîtra deux fois. Une fois dans la liste de d'adjacence de chaque extrémité . Donc, la complexité finale sera : $O (n) + O (2m) = O (n+m)$.

2.3 Les outils utilisés

1. **Python3** : langage de programmation.
2. **PyQt 5** : L'interface graphique.
3. **networkx et matplotlib.pyplot** : Afficher le graphe et visualiser les points d'articulation avec les options(zoomer,enregistrer l'image,modifier les couleurs de le graphe , de les arcs ...).
4. **Latex** : pour générer ce rapport .

Chapter 3

Conclusion

La programmation dynamique sert à résoudre de très grand nombre de problèmes, on remarque cela dans ce TP.

La complexité de la programmation dynamique est polynomial tout en gardant la simplicité de la récursivité avec une complexité base.

La programmation dynamique est un cas de l'intelligence artificielle