Compte Rendu TME1

Wenzhuo ZHAO, Zhaojie LU, Chengyu YANG, Zhen HOU

Février 2021

1 Liste d'adjacence

Pour réaliser l'algorithme BFS, nous implémentons la liste d'adjacence pour présenter le graphe. Nous avons un tableau des têtes de listes qui stocke tous les noeuds, nommé **data** dans le figure ci-dessous. Pour les arrêts de chaque noeud, nous avons une liste **first** de noeuds **adjVex** pour présenter que le noeud **data** et le noeud **adjVex** constituent un arrêt.

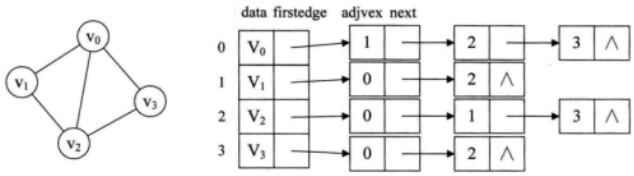

Figure 1: Liste d'adjacence

```
//the node on edge connected to the start node of array
struct EdgeNode{
    long adjVex;
};

//the start node of array
struct VertexNode{
    long data;
    std::list<EdgeNode> first;
};
```

2 BFS

Le but de réaliser l'algorithme BFS est de trouver le noeud le plus éloigné **v** du noeud **u** dans le graphe et aussi de trouver la distance **dist(u,v)** entre eux. Nous définissons deux variables **furthestNode** et **maxDistance** pour stocker

ces résultats. En cours de l'algorithme BFS, nous mettons à jours ces deux variables. A la fin de l'algorithme, la fonction renvoie un tableau contenant la distance entre **u** et tous les autres noeuds dans ce graphe.

2.1 Pseudo Code

Algorithm 1: BFS

```
Data: Adjaence array list
Data: Node start
Result: Map<Node, Integer> distance
Queue<Node> queue;
Map<node, integer> distance;
queue.push(start);
distance.insert(< start, 0 >);
while queue.isNotEmpty() do
    Node node = queue.pop();
    Integer newDist = distance[node] + 1;
    for EdgeNode e in list[node] do
        /* The node is not visited                          */
        if distance notContains e then
            /* Update the furthest node and the maximum
               distance                                     */
            if maxDistance < newDist then
                maxDistance = newDist;
                furthestNode = e.adjVex;
            distance.insert(< EdgeNode.adjVex, newDist >);
            queue.push(EdgeNode.adjVex);
return distance
```

Notre implémentation de cet algorithme en C++ produit les résultats sur de différents graphes dans un temps:

- moins de 1 secondes dans un graphe [1] de 334,863 noeuds et de 925,872 arrêts
- 30 secondes dans un graphe [2] de 3,997,962 noeuds et de 34,681,189 arrêts
- 1 minute 34 secondes dans un graphe [3] de 3,072,441 noeuds et de 117,185,083 arrêts

3 Lower Bound et Upper Bound du diamètre de graphe

Pour calculer le Lower Bound du diamètre, nous prenons un noeud aléatoire **N1** dans un graphe, puis calculons **BFS(N1)** à partir de ce noeud et obtiendrons

le noeud le plus éloigné **N2** de **N1**. Nous refaisons ce processus à partir de **N2**, et obtiendrons le noeud **N3** et la distance maximum **maxDistance**. Cette distance maximum est le Lower Bound que nous voudrions obtenir.
Pour calculer le Upper Bound, il faut générer l'arbre couvrant à partir du noeud **N2** et obtiendrons un chemin entre **N2** et **N3**. Nous prenons un noeud **N4** au milieu de ce chemin. Faisons **BFS(N4)** à partir de ce noeud, nous allons obtenir une distance **maxDistance** et on peut calculer le Upper Bound en multipliant cette distance par 2.

3.1 Exemple

Nous prenons un graphe suivant:

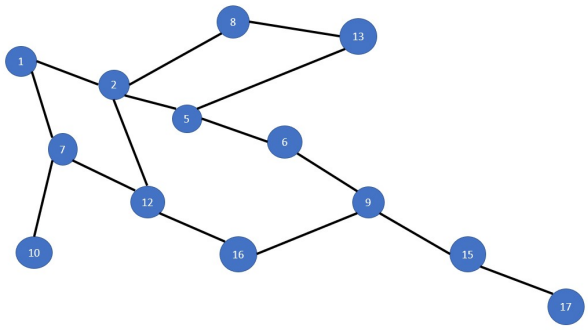

Figure 2: Liste d'adjacence

Prenons un noeud aléatoire, soit noeud numéroté 1, faisons un BFS à partir de ce noeud et nous obtiendrons un chemin de noeud 1 jusqu'au noeud 17 le plus éloigné du noeud 1. Évidemment, le noeud 6 tourné en vert est le noeud au milieu de ce chemin et aussi au milieu du graphe.

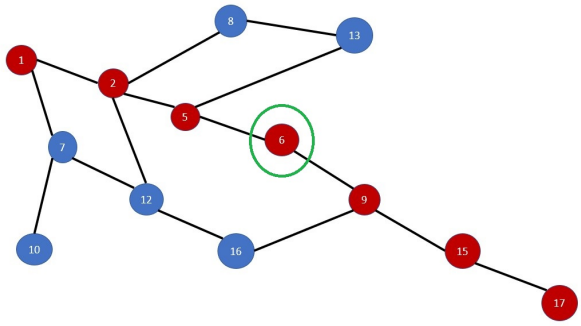

Figure 3: Lower Bound

Faisons un BFS à partir de ce noeud 6, et nous obtenons le noeud 10 le plus éloigné.

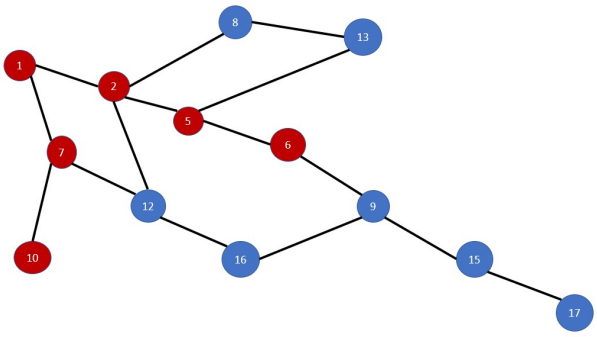

Figure 4: Upper bound

La distance entre le noeud 6 et le noeud 10 est 5, et nous pouvons en déduire que l'upper bound du diamètre est 5 * 2 = 10.

3.2 Performance

Cette suite de processus s'execute dans un temps et obtient des résultats:

- 3 secondes en graphe Amazon product co-purchasing network [1] et lower bound = 47, upper bound = 52
- 90 secondes en graphe LiveJournal social network and ground-truth communities [2] et lower bound = 21, upper bound = 26

- 5 minutes en graphe Orkut social network and ground-truth communities [3] et lower bound = 9, upper bound = 12

4 Liste de triangles

Pour créer une liste de triangles , nous prenons un graphe G et cet algorithme suivant.

- 1. Pour tout noeud u dans le graphe G , nous calculons la liste de ses voisins $tsl[u]$.
- 2. Pour tout arête (u,v) dans le graphe G , nous calculons l'intersection W de $tsl[u]$ et $tsl[v]$.
- 3. Pour chaque noeud w dans l'intersection W , nous ajoutons la triangle {u , v , w} dans la liste de triangles.

4.1 Exemple

Nous prenons un graphe suivant:

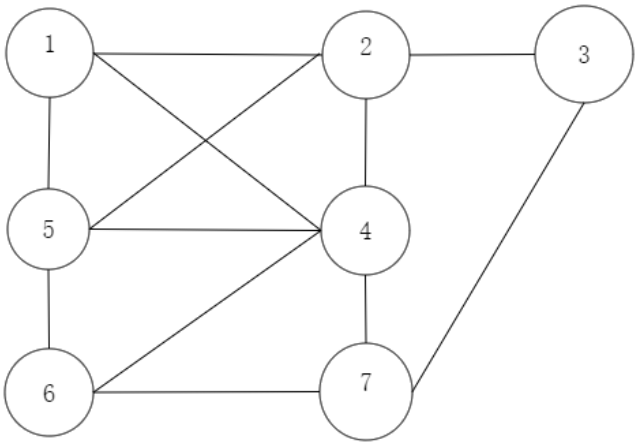

Figure 5: Exemple

Chargeons une "liste d'adjacence" partielle sans répétition d'arêtes sur deux bouts d'une arête : tsl[1] = {2, 4, 5}, tsl[2] = {3, 4, 5}, tsl[3] = {7}, tsl[4] = {5, 6, 7}, tsl[5] = {6}, tsl[6] = {7} et tsl[7] = {∅}.

Calculons l'intersection de voisins entre chaque paire de noeuds. W(1,2) = {4, 5}, W(1,4) = {5}, W(1,5) = {∅}, W(2,4) = {5}, W(2,5) = {∅}, W(4,5) = {6}, W(4,6) = {7}, W(4,7) = {∅}, W(5,6) = {∅} et W(6,7) = {∅}.

À la fin, la liste de triangles est [{1,2,4},{1,2,5} ,{1,4,5},{2,4,5},{4,5,6},{4,6,7}] qui vérifie bien ce que l'on visualise dans le figure 5.

4.2 Performance

Après l'exécution, nous avons obtenu les résultats ci-dessous :

- en graphe Amazon product co-purchasing network [1] le nombre de triangle est 667129 et le temps d'exécution est 5 secondes.
- en graphe LiveJournal social network and ground-truth communities [2] le nombre de triangle est 177820130 et le temps d'exécution est une heure et 33 secondes .

Appendix A Source Code

Le source code de ce TME est disponible sur ce répertoire de GitHub [4].

References

[1] *Amazon product co-purchasing network and ground-truth communities.* `http://snap.stanford.edu/data/com-Amazon.html`.

[2] *LiveJournal social network and ground-truth communities.* `http://snap.stanford.edu/data/com-LiveJournal.html`.

[3] *Orkut social network and ground-truth communities.* `http://snap.stanford.edu/data/com-Orkut.html`.

[4] W.Zhao. *CPA Graph.* `https://github.com/valeeraZ/Sorbonne_CPA_Graph/tree/master/TME1`.