

Structures de données et algorithmes – TP7

Iulia-Cristina Stanica

The bottom of the slide features a decorative graphic consisting of several overlapping, wavy, horizontal bands. The colors transition from a light gray at the top to a dark gray, and finally to a solid black band at the very bottom. The waves create a sense of movement and depth.

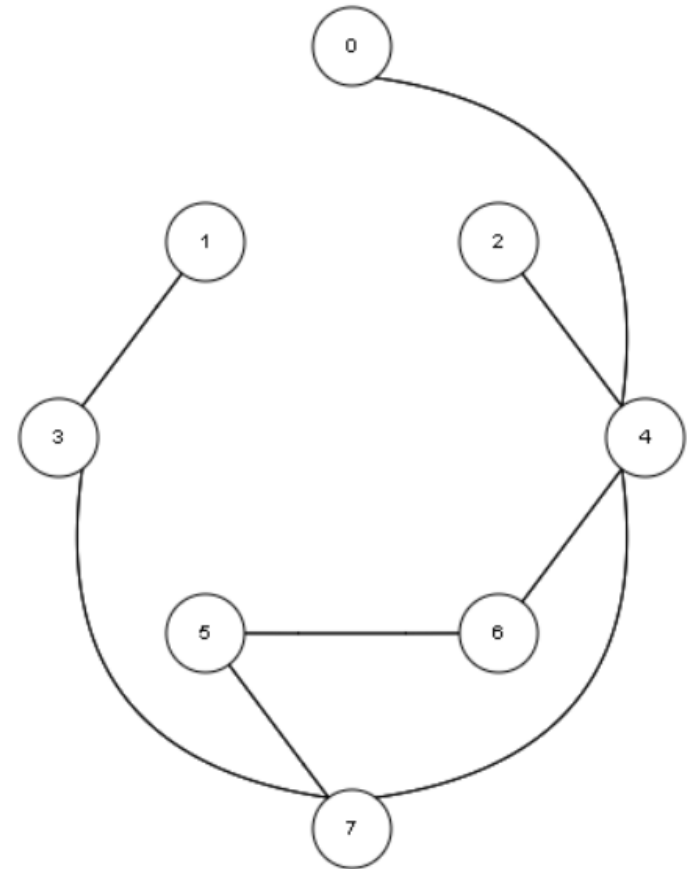
Objectifs pour aujourd'hui

Graphe non-orienté

- ▶ Algorithmes de parcours (BFS et DFS)
- ▶ Graphe biparti

Ex. 1

- Pour le graphe de l'image, compléter les tâches suivantes dans `adjacencymatrix.cpp`:
 - Corriger le constructeur en fonction du nombre de nœuds
 - Compléter les arêtes
 - Appliquer DFS et BFS à partir du nœud 4



Graphe biparti

- Un graphe biparti est un graphe dont les sommets peuvent être divisés en 2 groupes disjoints (les arêtes connectent seulement 2 sommets appartenant au groupes différents)

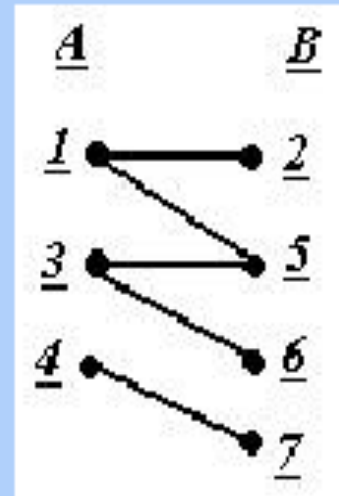
$G=(X,U)$

$X=\{1,2,3,4,5,6,7\}$

$U=\{[1,2];[1,5];[3,5];[3,6];[4,7]\}$


$A=\{1,3,4\}$

$B=\{2,5,6,7\}$









Graphe biparti – application vie réelle 😊



Dating and Friendship Gender Friendship Advice Friendship Dating Advice

Personal Question 

I like girl A. We are very good friends. A's best friend B likes me. Also, A likes a guy C who is my best bro and C likes another girl D who is in a relationship with another guy. What should we do?


 Answer  Follow · 143  Request  8  

90 Answers

**Priyanka Dhaka**, studied at Indian Institute of Technology, Delhi 
Updated Jul 28, 2014

Originally Answered: I like girl A. We are very good friends, but her best friend B likes me. Also, A likes guy C, who is my best bro. C likes another girl D, who is in a relationship with another guy. What should we do?







You can apply Bipartite Matching algorithm of Graph Theory. All you have to do is:

1. Put all girls as vertices (a's) in Part A, and boys as vertices (b's) in Part B.
2. Draw an edge between vertex a to vertex b if a likes b.
3. It will be a bipartite graph (there are no edges within vertices of Part A, and same for Part B) unless your friends are not straight.
4. Then find maximum matching for this bipartite graph. (refer this link [Maximum Bipartite Matching - GeeksforGeeks](https://www.geeksforgeeks.org/maximum-bipartite-matching/) )
5. You will get maximum matching as output and the matched edges will be the pairs who should be together.
6. You can apply weighted bipartite matching algorithm if you know how much they love/like each other. In that case, assign weights according to their amount of love. and repeat step 4 and 5.

Thanks for asking this question. Finally I have found a real life problem which can be solved by what Prof Panda has been teaching us for 2-3 years.

P.S. I can provide you code for both weighted and unweighted maximum bipartite matching which I made for my project.

203.1K views · View 13,701 upvotes · View 1 share

 13.7K   1  110  

<https://www.geeksforgeeks.org/maximum-bipartite-matching/>

Ex. 2

- ▶ Vérifier si un graphe est biparti et, si oui, afficher les éléments de ses deux ensembles A et B.
- ▶ **Hint:** Cherchez un algorithme qui est basé sur BFS. Utiliser le « nodeInfo » pour assigner des étiquettes pour les ensembles A et B. Utilisez le fichier `adjacencymatrix.cpp` de la plateforme pour tester (et comprendre) premièrement les algorithmes BFS et DFS. Ajouter les lignes qui manquent.
- ▶ Vérifiez votre code pour les graphes suivants:
 - $G1 = (\{0,1,2,3,4,5,6,7,8\}, \{ (0,1), (0,2), (3,4), (4,5), (6,4), (1,3), (4,7), (6,8), (3,2), (7,8) \})$
 - $G2 = (\{0,1,2,3,4,5,6,7,8\}, \{ (0,1), (0,2), (3,4), (4,5), (6,4), (1,3), (4,7), (6,8), (3,2), (7,8), (3,6) \})$

Ex. 3 (réseau social)

- ▶ Prenons un graphe non orienté, qui représente un réseau social. Chaque sommet représente un utilisateur.
- ▶ A est ami avec B s'il existe une arête entre A et B (dans ce cas-là on dit que le degré d'amitié est 1). Les amis de mes amis ont le degré d'amitié 2.
- ▶ Tenant compte d'un utilisateur, affichez tous ses amis ayant le degré $\leq N$ (N est donné). Utiliser pour le test le graphe de l'exercice précédant.

Exemple:

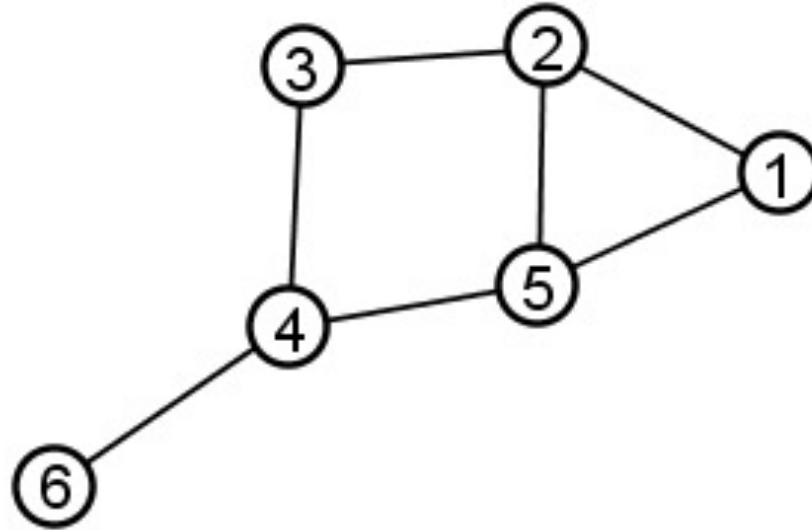
- ▶ Soit le graphe avec les arêtes:
 $\{(0,1);(2,3);(0,2);(0,6);(4,5);(7,5);(6,4)\}$
- ▶ Si l'utilisateur est dans le sommet 0 et le degré maximum d'amitié est 2 ($N=2$), on affiche:
 - Les amis avec degré 1 sont: 1, 2, 6
 - Les amis avec degré 2 sont: 3, 4

HINT: L'un des deux algorithmes (BFS ou DFS) doit être appliqué. Lequel?

Vous pouvez utiliser les étiquettes des sommets ou des arêtes pour stocker des informations nécessaires.

Support théorique

Exemple graphe



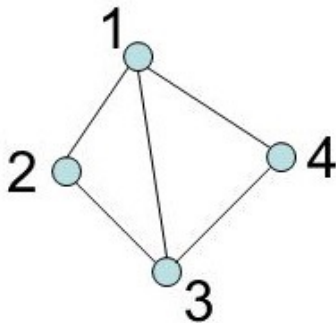
► Les sommets: $\{1,2,3,4,5,6\}$

Les arêtes:

$$A=\{(1,5),(1,2),(2,5),(2,3),(3,4),(4,5),(4,6)\}$$

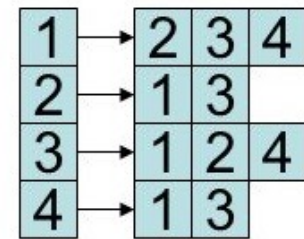
Représentation

- ▶ 1. Matrice d'adjacence
- ▶ 2. Liste des voisins (d'adjacence)



	1	2	3	4
1	0	1	1	1
2	1	0	1	0
3	1	1	0	1
4	1	0	1	0

Adjacency matrix
Graph Algorithms



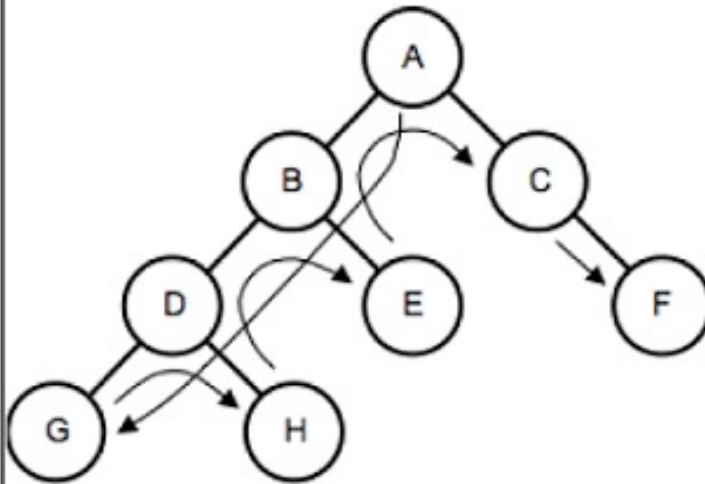
Adjacency list

Algorithmes de parcours

- ▶ 1. Algorithme de parcours en profondeur
Depth-First Search (DFS)
- ▶ 2. Algorithme de parcours en largeur
Breadth-First Search (BFS)

Example

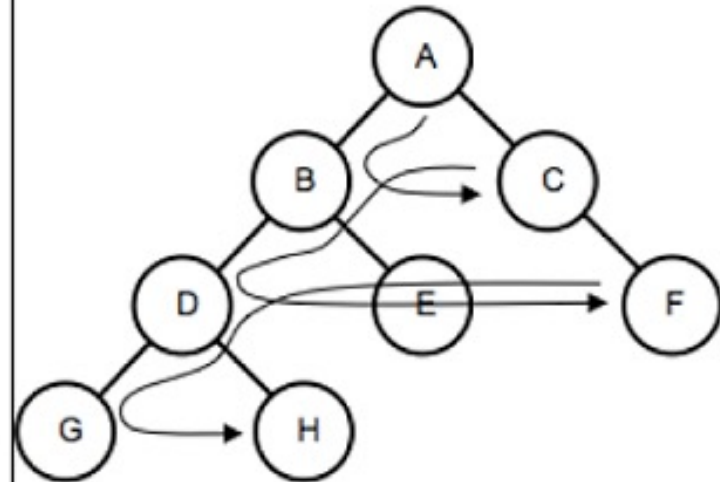
Depth First Search



1. Start at node A. Visit the children before the siblings.
2. Visit the children from A, which are B, D, and G.
3. Visit the other child of D, which is H, the sibling of G.
4. Visit the other child of B, which is E, the sibling of D.
5. Visit the other children from A, which are C and F.

Result: A, B, D, G, H, E, C, F

Breadth First Search



1. Start at node A.
2. Visit the children of node A (siblings B and C).
3. Visit the children of B (siblings D and E).
4. Visit the children of C (node F).
5. Visit the children of D (siblings G and H).

Result: A, B, C, D, E, F, G, H

Voyons le code...

1. Implémentation DFS

```
DFS (vertex u) {  mark u as visited
  for each vertex v directly reachable from u
    if v is unvisited
      DFS (v)
}
```

- ▶ Initialement tous les sommets sont marqués comme *unvisited*.
- ▶ « Aller assez loin que possible »

2. Implémentation BFS

```
enqueue S to Q and mark S as visited
while Q not empty
  dequeue the first vertex x from Q
  print x
  for each vertex y directly reachable from x
    if y is unvisited
      enqueue y to Q
      mark y as visited
```

- ▶ Initialement tous les sommets sont “unvisited” et la file d’attente Q est vide.
- ▶ « Trouver tous les chemins possibles »