L2 GÉNIE INFORMATIQUE • SEMESTRE 3

THÉORIE DES GRAPHES ET OPTIMISATION DES PROCESSUS

Séance 2 : Parcours en Largeur (BFS)

Ing. GBAGUIDI AISSE Alexandre

ESGC VERECHAGUINE A.K. • Cotonou, Bénin

Plan de la séance

📚 Rappel : Files

FIFO, opérations

5 min

🔍 Algorithme BFS

Couche par couche

40 min

🎯 Applications

Distance, chemin

30 min

⏱ Complexité

O(V+E) analyse

15 min

✏️ TD (1h)

3 exercices

Papier

💻 TPE (1h20)

Implémenter BFS

Code C++

Objectifs de la séance

1. **Comprendre le principe du BFS**
   Explorer un graphe couche par couche
   Utiliser une file (FIFO) pour gérer l'ordre de visite
2. **Implémenter l'algorithme BFS**
   Structures : visited[], distance[], parent[]
   Pseudo-code détaillé et exemple pas à pas
3. **Calculer les distances minimales**
   Distance = nombre minimum d'arêtes
   Reconstruction du chemin le plus court
4. **Détecter la connexité**
   Tester si un graphe est connexe
   Compter les composantes connexes
5. **Analyser la complexité**
   Complexité temporelle : $O(V + E)$
   Complexité spatiale : $O(V)$

Pourquoi étudier les parcours ?

🌐 Exploration systématique

- Visiter tous les sommets
- Sans oublier aucun
- Sans visiter deux fois

🔧 Base des algorithmes

- Plus courts chemins
- Connexité, composantes
- Détection de cycles

💼 Applications réelles

- GPS : routes et distances
- Réseaux sociaux : suggestions
- Web : crawlers et indexation

🚀 Fondamental en graphes

- BFS et DFS = fondations
- Tous les autres algorithmes en dépendent
- Comprendre pour maîtriser

PARTIE 2

Rappel : Structure de File

"First In, First Out"

Structure de File (Queue)

FIFO = First In, First Out

Le premier élément ajouté est le premier à sortir
Comme une file d'attente à la banque !

Opérations principales

• enqueue(x) : Ajouter x à la fin
• dequeue() : Retirer et retourner le premier élément
• is_empty() : Tester si la file est vide
• front() : Consulter le premier élément (sans retirer)

Exemple :

```
File vide : []
enqueue(5) → [5]
enqueue(10) → [5, 10]
enqueue(3) → [5, 10, 3]
dequeue() → retourne 5, file = [10, 3]
dequeue() → retourne 10, file = [3]
```

Ordre de sortie : A, B, C

File en C++ - std::queue

```
#include <queue>
#include <iostream>
using namespace std;

int main() {
    queue<int> Q;

    // Ajouter des éléments
    Q.push(5);    // enqueue
    Q.push(10);
    Q.push(3);

    // Consulter le front
    cout << "Front: " << Q.front() << endl;   // 5

    // Retirer des éléments
    while (!Q.empty()) {
        int x = Q.front();
        Q.pop();  // dequeue
        cout << x << " ";
    }
    // Affiche : 5 10 3

    return 0;
}
```

✓ Complexité

push() : O(1) pop() : O(1) front() : O(1)
empty() : O(1)

Toutes les opérations en temps constant !

PARTIE 3

Algorithme BFS

Breadth-First Search

"Parcours en largeur d'abord"

Pourquoi avons-nous besoin du BFS ?

📱 Problème 1 - Réseau social

Question : Quel est le degré de séparation entre vous et Bill Gates ?

Degré de séparation = nombre minimum de "liens d'amitié"
Exemple : Vous → Ami A → Ami B → Bill Gates → Degré = 3

Comment calculer cela algorithmiquement ?

🚗 Problème 2 - Transport Gozem (Cotonou)

Question : Nombre minimal de transferts de Étoile Rouge à Ganhi ?

Réseau de stations :
Étoile Rouge → Station A → Station B → Ganhi → 3 transferts

💡 BFS résout exactement ce problème !

Distance minimale en nombre d'arêtes

BFS : Explorer couche par couche

Principe

BFS explore le graphe niveau par niveau :
1. Visiter tous les voisins de la source (niveau 1)
2. Puis tous les voisins des voisins (niveau 2)
3. Et ainsi de suite...

Comme des vagues concentriques dans l'eau !

🌊 Métaphore

Imaginez une goutte d'eau tombant dans un lac :

- Les vagues se propagent dans toutes les directions
- Chaque vague atteint d'abord les points proches
- Puis progressivement les points plus éloignés

C'est exactement ainsi que BFS explore !

Niveaux d'exploration

Niveau 0 : S

Niveau 1 : A B C

Niveau 2 : D E F G

Niveau 3 : H I

BFS : Quelles structures de données ?

1

File Q *Type : queue<int>*

Rôle : Gérer l'ordre de visite (FIFO)

Contient les sommets en attente de traitement

2

Tableau visited[] *Type : vector<bool>*

Rôle : Marquer les sommets déjà visités

Valeurs : true/false pour chaque sommet

3

Tableau distance[] *Type : vector<int>*

Rôle : Stocker la distance depuis la source

Valeurs : 0 pour source, puis 1, 2, 3...

4

Tableau parent[] *Type : vector<int>*

Rôle : Reconstruire le chemin le plus court

parent[v] = u si on est arrivé à v depuis u

BFS : Algorithme de base

```
BFS(Graph G, int source):
    // Initialisation
    Queue<int> Q
    for each vertex v in G:
        visited[v] = false

    // Démarrer depuis la source
    visited[source] = true
    Q.enqueue(source)

    // Boucle principale
    while not Q.is_empty():
        u = Q.dequeue()
        print(u)   // Traiter le sommet u

        for each neighbor v of u:
            if not visited[v]:
                visited[v] = true
                Q.enqueue(v)
```

✓ Idée clé

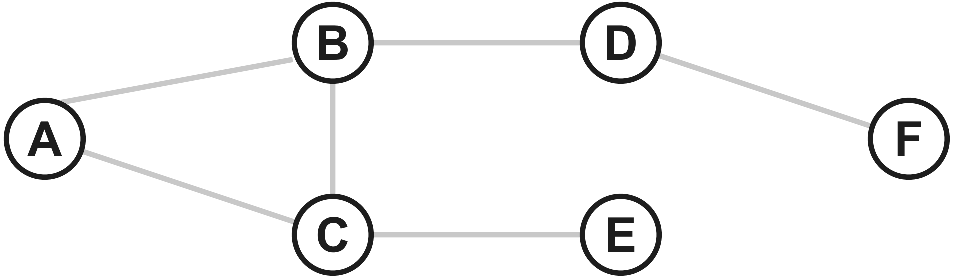
- On visite chaque sommet exactement 1 fois
- On enfile les voisins non visités
- La file garantit l'ordre niveau par niveau

BFS : Version complète avec distances

```
BFS(Graph G, int source):
    // Initialisation de tous les tableaux
    for each vertex v in G:
        visited[v] = false
        distance[v] = ∞
        parent[v] = -1

    // Démarrer depuis la source
    visited[source] = true
    distance[source] = 0
    parent[source] = -1

    Queue<int> Q
    Q.enqueue(source)

    // Boucle principale
    while not Q.is_empty():
        u = Q.dequeue()

        for each neighbor v of u:
            if not visited[v]:
                visited[v] = true
                distance[v] = distance[u] + 1
                parent[v] = u
                Q.enqueue(v)
```

✓ distance[v] : Compte le nombre d'arêtes depuis source • ✓ parent[v] : Permet de reconstruire le chemin • ✓ ∞ signifie "non atteignable depuis source"

Exemple BFS : Graphe de départ

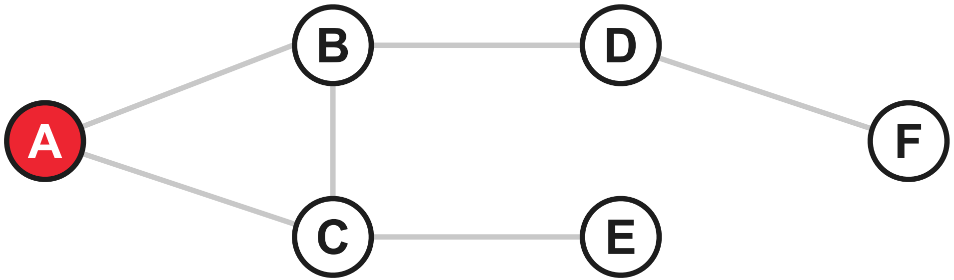


Source : A

Sommets : {A, B, C, D, E, F}
Arêtes : (A,B), (A,C), (B,C), (B,D), (C,E), (D,F)

Question : Quel sera l'ordre de visite avec BFS depuis A ?

BFS : Étape 0 - Initialisation



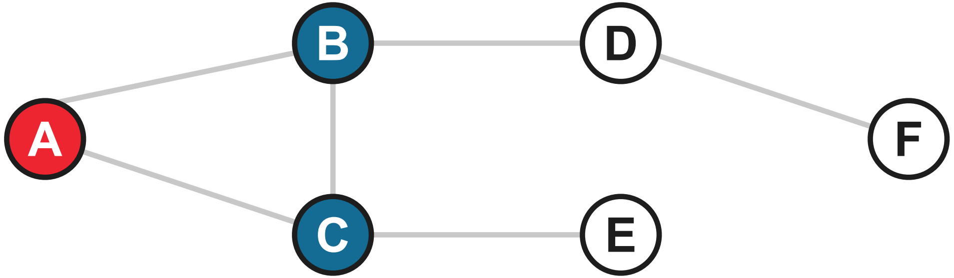
État après étape 0

Sommet	visited	distance	parent
A	T	0	-1
B	F	∞	-1
C	F	∞	-1
D	F	∞	-1
E	F	∞	-1
F	F	∞	-1

`File Q : [A]`

BFS : Étape 1 - Traiter A

Action : Défile A, enfile ses voisins B et C



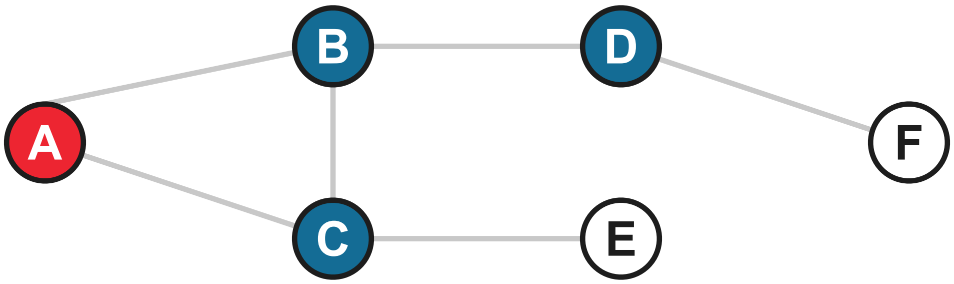
État après étape 1

Sommet	visited	distance	parent
A	T	0	-1
B	T	1	A
C	T	1	A
D	F	∞	-1
E	F	∞	-1
F	F	∞	-1

`File Q : [B, C]`

BFS : Étape 2 - Traiter B

Action : Défile B, enfile D (C déjà visité)



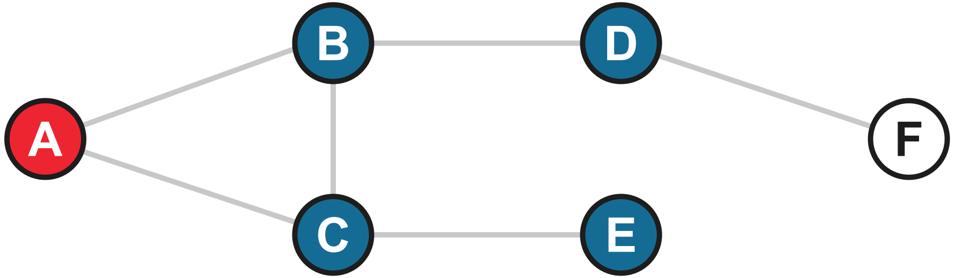
État après étape 2

Sommet	visited	distance	parent
A	T	0	-1
B	T	1	A
C	T	1	A
D	T	2	B
E	F	∞	-1
F	F	∞	-1

`File Q : [C, D]`

BFS : Étape 3 - Traiter C

Action : Défile C, enfile E (A et B déjà visités)



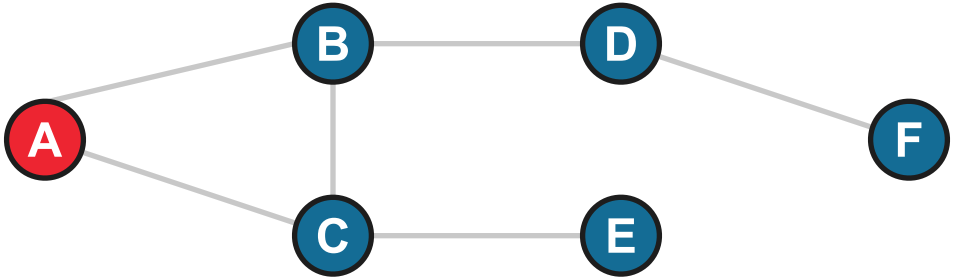
État après étape 3

Sommet	visited	distance	parent
A	T	0	-1
B	T	1	A
C	T	1	A
D	T	2	B
E	T	2	C
F	F	∞	-1

`File Q : [D, E]`

BFS : Étapes 4-5 - Traiter D puis E

Étape 4 : Défile D, enfile F
Étape 5 : Défile E, pas de nouveau voisin



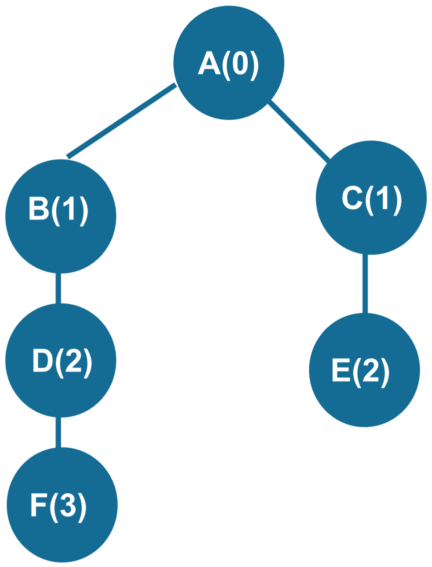
État après étape 4-5

Sommet	visited	distance	parent
A	T	0	-1
B	T	1	A
C	T	1	A
D	T	2	B
E	T	2	C
F	T	3	D

`File Q : [F]`

BFS : Arbre résultant

Arbre BFS :



Résumé

Ordre de visite :
A → B → C → D → E → F

Distances depuis A :
- dist(A) = 0
- dist(B) = dist(C) = 1
- dist(D) = dist(E) = 2
- dist(F) = 3

✓ Observation

- BFS visite d'abord tous les sommets à distance 1
- Puis tous les sommets à distance 2
- Et ainsi de suite : exploration par niveaux !

PARTIE 4

Applications du BFS

"Que peut-on faire avec BFS ?"

Application 1 : Distance minimale

Définition

Distance entre deux sommets = Nombre MINIMUM d'arêtes dans un chemin les reliant
Note : Cette distance n'est valable que pour graphes NON PONDÉRÉS (toutes arêtes = poids 1)

✓ Propriété de BFS

- BFS calcule la distance minimale depuis source vers TOUS les sommets
- En un seul parcours du graphe
- Résultat stocké dans distance[]

📱 Exemple concret - Réseau social

Vous lancez BFS depuis votre profil :
- distance[Alice] = 1 → Alice est un ami direct
- distance[Bob] = 2 → Bob est ami d'un de vos amis
- distance[Charlie] = 3 → Charlie est à "3 degrés de séparation"

C'est exactement ce que fait LinkedIn pour "People You May Know" !

Application 2 : Reconstruction du chemin

❓ Problème

BFS calcule distance[dest] depuis source
Mais comment obtenir le CHEMIN lui-même ?

✓ Solution

Utiliser le tableau parent[]

Algorithme de reconstruction

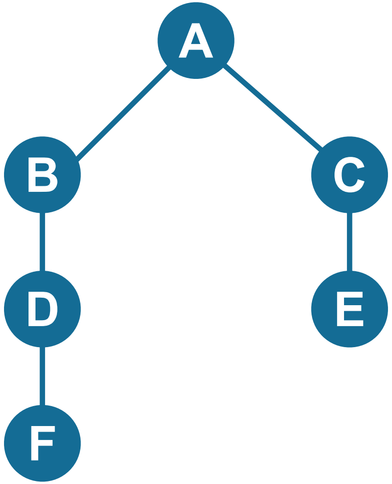
```
reconstruct_path(source, dest):
    if distance[dest] == ∞:
        return "Pas de chemin"

    path = []
    current = dest

    while current != -1:
        path.prepend(current)   // Ajouter au début
        current = parent[current]

    return path
```

Exemple : Chemin de A à F

parent[F] = D, parent[D] = B, parent[B] = A, parent[A] = -1
Reconstruction : F ← D ← B ← A
Chemin : A → B → D → F

Exemple : Reconstruire tous les chemins depuis A

Arbre parent[]



Chemins depuis A

A → A :	`[A]`	*(distance 0)*
A → B :	`[A, B]`	*(distance 1)*
A → C :	`[A, C]`	*(distance 1)*
A → D :	`[A, B, D]`	*(distance 2)*
A → E :	`[A, C, E]`	*(distance 2)*
A → F :	`[A, B, D, F]`	*(distance 3)*

Code C++

```cpp
vector<int> shortest_path(int source, int dest) {
    vector<int> path;
    int current = dest;
    while (current != -1) {
        path.insert(path.begin(), current);
        current = parent[current];
    }
    return path;
}
```

Application 3 : Tester si un graphe est connexe

Définition

Un graphe non-orienté est CONNEXE si il existe un chemin entre toute paire de sommets

✓ Test avec BFS

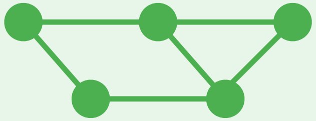
Algorithme :

1. Lancer BFS depuis un sommet quelconque (ex: sommet 0)
2. Compter combien de sommets ont été visités
3. Si tous les V sommets sont visités → CONNEXE
   Sinon → NON CONNEXE

```
is_connected(Graph G):
    BFS(G, 0)  // Partir du sommet 0

    visited_count = 0
    for each vertex v:
        if visited[v]:
            visited_count++

    return (visited_count == V)
```

Exemples

✓ CONNEXE



Tous les sommets atteignables

✗ NON CONNEXE

2 composantes séparées

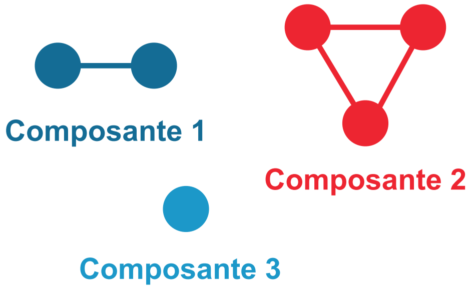
Complexité : O(V + E) - un seul BFS !

Application 4 : Compter les composantes connexes

Pour graphes NON connexes

Composante connexe = sous-graphe connexe maximal

Exemple : 3 composantes connexes



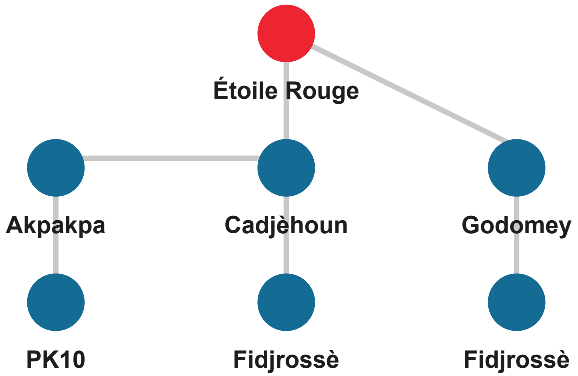

```
count_components(Graph G):
    // Réinitialiser visited[] à false
    for each vertex v:
        visited[v] = false

    count = 0

    // Lancer BFS depuis chaque sommet non visité
    for each vertex v:
        if not visited[v]:
            BFS(G, v)   // Visite une composante complète
            count++

    return count
```

Complexité : O(V + E) - chaque sommet visité exactement 1 fois

Application : Réseau de transport Gozem

🚗 Gozem = Application de transport à Cotonou
Stations de mototaxi réparties à travers la ville

Réseau de stations



❓ Problème

Nombre minimal de transferts de Étoile Rouge à Palais ?

✓ Solution avec BFS

Source = Étoile Rouge
BFS calcule :

- distance[Palais] = 2
- Chemin : Étoile Rouge → Godomey → Fidjrossè
- 2 transferts minimum

✓ BFS trouve le chemin avec le MOINS de transferts
✓ Utile pour optimiser les trajets • Gozem utilise ce type d'algorithme !

PARTIE 5

Analyse de Complexité

"Pourquoi BFS est-il efficace ?"

Complexité temporelle de BFS

Analyse détaillée

1. Initialisation : O(V)

```
for each vertex v:
    visited[v] = false   // O(V)
```

2. Boucle principale

```
while not Q.is_empty():
    u = Q.dequeue()          // Fait V fois
total
    for each neighbor v of u:   // Parcourt
toutes les arêtes
        ...
```

Analyse

- Chaque sommet est enfilé et défilé exactement 1 fois : O(V)
- Chaque arête est explorée 1 ou 2 fois (dépend si orienté) : O(E)

Total : O(V + E)

✓ Optimalité

O(V + E) est OPTIMAL pour parcourir un graphe car il faut au minimum :

• Visiter chaque sommet : V opérations • Examiner chaque arête : E opérations

Complexité spatiale de BFS

Structures utilisées

1. **File Q** → O(V)
   Au pire, tous les sommets d'un niveau sont enfilés
   Niveau le plus large possible ≤ V

2. **Tableau visited[]** → O(V)
   Un booléen par sommet

3. **Tableau distance[]** → O(V)
   Un entier par sommet

4. **Tableau parent[]** → O(V)
   Un entier par sommet

Total : O(V + V + V + V) = O(V)

Impact de la représentation sur BFS

Tableau comparatif

Opération	Liste d'adjacence	Matrice d'adjacence
Itérer sur voisins de u	`O(deg(u))`	`O(V)`
BFS complet	`O(V + E)`	`O(V²)`

Explication détaillée

✓ Avec LISTE d'adjacence

for each neighbor v of u:
    ...
Itère seulement sur les vrais voisins → O(deg(u))
Sur tout le graphe : O(V + E) ✓

✗ Avec MATRICE d'adjacence

for v = 0 to V-1:
    if adj[u][v] == 1: ...
Doit scanner toute la ligne → O(V)
Sur tout le graphe : $O(V^2)$ ✗

✓ Liste d'adjacence est MEILLEURE pour BFS
✓ Surtout pour graphes creux ($E << V^2$) • Matrice gaspille du temps à scanner des 0

Récapitulatif de la séance

📚 File FIFO

Structure de données essentielle
Opérations en O(1)
Gère l'ordre de visite

🔍 Principe du BFS

Explorer niveau par niveau
File Q + tableaux visited[], distance[], parent[]
Algorithme simple et systématique

📏 Distance minimale

BFS calcule distance = nombre d'arêtes
Valable pour graphes NON pondérés
Un seul parcours suffit

🛤️ Reconstruction chemins

Tableau parent[] stocke l'arbre BFS
Remonter depuis destination jusqu'à source
Obtenir le chemin le plus court

🔗 Connexité

Tester si graphe connexe
Compter composantes connexes
Tester graphe biparti

⚡ Complexité optimale

Temps : O(V + E) - optimal !
Espace : O(V)
Linéaire pour graphes creux

BFS : Quand l'utiliser ?

✓ UTILISER BFS pour

- Graphes NON pondérés
- Toutes les arêtes ont le même "coût"
- Distance = nombre d'arêtes

- Plus court chemin (en nombre d'arêtes)
- Garantit de trouver le chemin minimal

- Explorer niveau par niveau
- Applications nécessitant cette structure

- Connexité et composantes
- Test de connexité, décompte composantes

- Graphes bipartis
- Coloration alternée, test bipartition

✗ NE PAS utiliser BFS pour

- Graphes pondérés
- BFS ne tient pas compte des poids
- → Utiliser Dijkstra (Séance 4)

- Détection de cycles
- BFS peut, mais DFS est plus naturel
- → DFS identifie les back edges

- Tri topologique
- Nécessite DFS (Séance 3)
- → Ordre des temps de fin

💡 Note importante

BFS garantit le plus court chemin UNIQUEMENT pour graphes non pondérés
Pour graphes pondérés, utiliser Dijkstra (poids positifs) ou Bellman-Ford (poids négatifs)

BFS vs DFS : Première comparaison

Tableau comparatif

Critère	BFS	DFS
Structure	File (FIFO)	Pile (LIFO) ou récursion
Exploration	Niveau par niveau	Profondeur d'abord
Plus court chemin	✓ Oui (non pondéré)	✗ Non
Détection cycle	Possible	✓ Plus naturel
Ordre de visite	Largeur	Profondeur
Mémoire	O(V)	O(V)
Temps	O(V + E)	O(V + E)

DFS sera étudié en détail à la Séance 3 ! • BFS et DFS sont complémentaires

PARTIE 7

Travaux Dirigés

1 heure - 3 exercices

TD : 3 exercices progressifs

1

🟢 FACILE

Dérouler BFS à la main

• Comprendre le fonctionnement pas à pas
• Graphe donné avec 7 sommets
• Tableau à remplir pour chaque étape
• File, visited[], distance[], parent[]

⏱ *20 min*

2

🟡 MOYEN

Distance et reconstruction de chemin

• Appliquer BFS pour trouver chemins
• Calculer distance entre deux sommets
• Reconstruire le chemin complet
• Vérifier l'optimalité

⏱ *20 min*

3

🔴 DIFFICILE

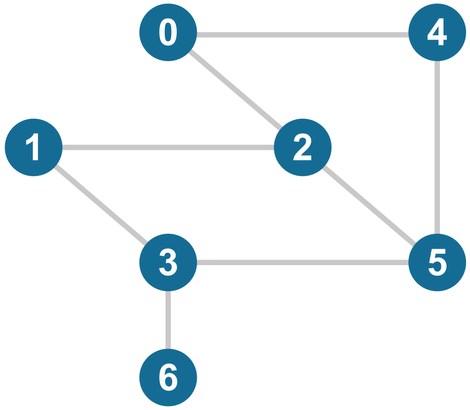
Composantes connexes

• Analyser la structure du graphe
• Graphe non connexe donné
• Identifier toutes les composantes
• Compter le nombre de composantes

⏱ *20 min*

TD - Exercice 1 : Dérouler BFS pas à pas

Graphe donné



Source : Sommet 0

Consigne

Remplissez le tableau suivant à chaque étape du BFS :

Colonnes :
- Étape
- File Q
- Sommet traité
- Voisins ajoutés
- visited[]
- distance[]

Questions supplémentaires

1. Quel est l'ordre complet de visite ?
2. Quelle est la distance de 0 à 6 ?
3. Quel est le chemin le plus court de 0 à 4 ?

TD - Exercices 2 et 3

🟡 Exercice 2

Distance et reconstruction

```
Graphe :
    A ─── B ─── C
     \   / \   /
      \ /   \ /
       D ─── E

Source : A
Destination : E
```

🔴 Exercice 3

Composantes connexes

Graphe non connexe :

[A—B] [C—D—E] [F—G] [H]

Questions :
1. Combien de composantes connexes ?
2. Lister les sommets de chaque composante
3. Quel(s) BFS faut-il lancer pour tout visiter ?

📝 **Travaillez sur papier, montrez vos calculs**
⏱ **20 minutes par exercice • ✓ Correction collective après chaque exercice**

PARTIE 8

Travail Personnel Encadré

1h20 - Implémenter BFS en C++