

Définition: Une chaîne dans un graphe

$G = (V, E)$  est une suite  $(v_0, e_1, v_1, \dots, e_k, v_k)$  qui :

- $v_i \in V$
- $e_i \in E$
- $e_{i+1} = v_i v_{i+1}$

$k$  est la longueur de la chaîne

Une chaîne est élémentaire si les sommets sont 2 à deux distincts

Un chaîne élémentaire avec  $n$  sommets est notée  $P_{n-1}$

$v_0$  et  $v_k$  sont les extrémités

Définition: Un graphe acyclique est une forêt

Définition: Un graphe  $G$  est connexe si il existe une chaîne entre  $u$  et  $v$   $\forall u, v \in V(G)$

Exemple:



graphe connexe



graphe non connexe

Définition: Soit  $G$  un graphe,  
 Un arbre couvrant de  $G$   
 est un sous-graphe couvrant  
 de  $G$  qui est acyclique (donc  
 un arbre)

**Proposition**

Un graphe est connexe  $\iff$  il contient un arbre couvrant. (et un seul)

**Démonstration**

- Soit  $G$  un graphe connexe.
- Retirons de  $G$ , tant qu'il est possible, une arête qui ne coupe pas le graphe (le graphe reste connexe).
- On obtient un sous-graphe partiel  $T$  qui est connexe par la condition sur les arêtes, et il n'a pas de cycles puisque s'il y aurait un cycle, on pourrait enlever une arête du cycle sans couper le graphe.
- $T$  est donc un arbre.
- Le converse est évident.

↳ Soit  $G$  un graphe géant. Est-il connexe ?

⇒ ALGORITHME DE MARQUAGE

**Entrées** : graphe  $G = (V, E)$  et sommet  $s \in V$

**début**

  marquer( $s$ ) ;

**tant que**  $\exists uv \in E$  où  $u$  est marqué et  $v$  non marqué **faire**

    marquer( $v$ )

Mais il y a mieux ...

## Parcours en largeur (BFS<sup>1</sup>)

**Entrées :** graphe  $G = (V, E)$  et sommet  $s \in V$

**début**

```
    créer file( $Q$ ) ;  
    marquer( $s$ ) ;  
    enfiler( $Q, s$ ) ;  
tant que  $Q \neq \emptyset$  faire  
         $u \leftarrow$  défiler( $Q$ ) ;  
        pour tous les  $uv \in E$  faire  
            si  $v$  non marqué alors  
                marquer( $v$ ) ;  
                enfiler( $Q, v$ )
```

1. Breadth-first search

Il existe une version avec distance.

## Parcours en largeur (version avec distances)

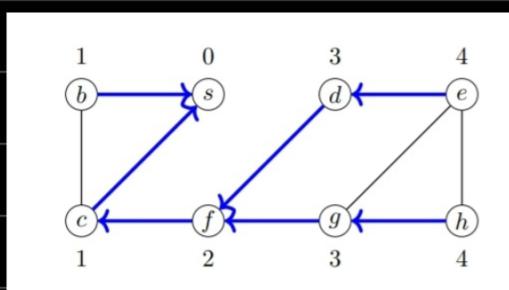
**Entrées :** graphe  $G = (V, E)$  et sommet  $s \in V$

**début**

```
    pour tous les  $u \in V \setminus \{s\}$  faire  
         $d(u) \leftarrow \infty$   
        dist( $s$ )  $\leftarrow 0$  ;  
         $Q \leftarrow [s]$  ;  
tant que  $Q \neq \emptyset$  faire  
         $u \leftarrow$  défiler( $Q$ ) ;  
        pour tous les  $uv \in E$  faire  
            si  $d(v) = \infty$  alors  
                 $d(v) = d(u) + 1$  ;  
                enfiler( $Q, v$ ) ;  
                parent( $v$ )  $\leftarrow u$ 
```

(Dans le poly,  veut dire  
le parent de  $c$  est  $d$ )

A la fin de cet  
algo, grâce aux  
parents, on est sur  
d'obtenir un plus



court chemin en suivant les flèches.

## Complexité de BFS:

- Si  $G$  est représenté par une liste d'adjacence.
- BFS considère chaque arête 2 fois.
- Chaque sommet est enfilé / défilé 1 fois.
- Donc  $\text{comp}(\text{BFS})$  est  $\mathcal{O}(m + n)$   
où  $m = |E|$  et  $n = |V|$

Quelle est sa complexité sur une matrice d'adjacence ?

### Correction du BFS

#### Lemme 1

Après terminaison du BFS,  $d(v) \geq \text{dist}(s, v)$  pour tout sommet  $v \in V$ .

#### Lemme 2

Si au cours du BFS  $Q = [v_1, v_2, \dots, v_r]$ , alors  $d(v_r) \leq d(v_1) + 1$  et  $d(v_i) \leq d(v_{i+1})$  pour  $i = 1, 2, \dots, r - 1$ .

#### Corollaire 1

Si le sommet  $v_i$  est enfilé dans  $Q$  avant  $v_j$ , alors  $d(v_i) \leq d(v_j)$ .

(Pour la démonstration voir poly)

#### Théorème

BFS découvre tous les sommets accessibles à partir de  $s$ , et après terminaison,  $d(v) = \text{dist}(s, v)$  pour tout  $v \in V$ .

(Idem)

Implémentation en Python:

```
def bfs (s, Adj):
```

```
level = {s: 0}
parent = {s: None}
i = 1
frontier = [s]
while frontier:
    next = []
    for u in frontier:
        for v in Adj[u]:
            if v not in level:
                level[v] = i
                parent[v] = u
                next.append(v)
    frontier = next
    i += 1
```

