INF421-a
Bases de la programmation et de l'algorithmique

Equilibrage d'arbres binaires et Graphes
(Bloc 9 / 9)

Philippe Baptiste

CNRS LIX, École Polytechnique

21 octobre 2005

Aujourd'hui

Équilibrage d'arbres (Red \ Black)

Graphes

Fermeture transitive

Parcours

Plus court chemin

De la hauteur d'un arbre binaire de recherche

Nous avons vu (cours 7) :

- La hauteur moyenne d'un arbre binaire de recherche est $O(n \log n)$
- Dans le pire des cas $h = n$

Question fondamentale : Comment garantir une hauteur en $O(n \log n)$ (pire cas) ? $\rightarrow$ Équilibrage

- AVL
- Red \ Black trees

Équilibrage : Red \ Black Tree

Attention : Dans la suite, les valeurs sont portées par les sommets "internes", (*i.e.*, pas de valeur dans les feuilles) $\rightarrow$ hauteur d'un tel arbre = hauteur sans les feuilles

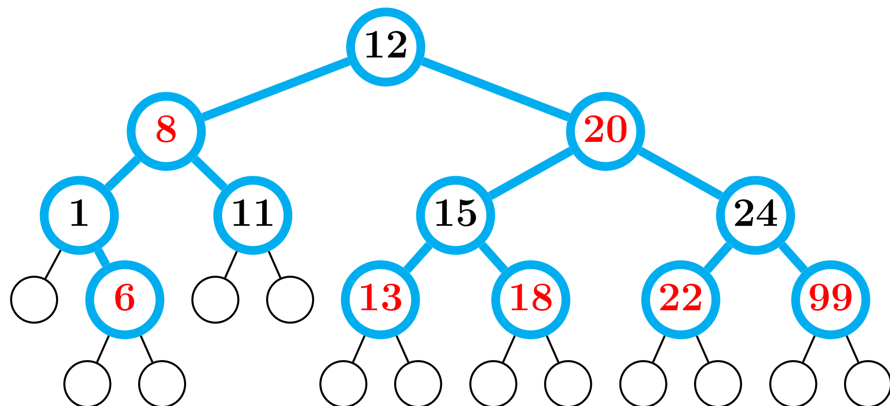
Un arbre rouge et noir est un arbre binaire de recherche dans lequel chaque nœud est de couleur rouge ou noire et

1. les feuilles sont NOIRES,
2. les fils d'un nœud ROUGE sont NOIRS,
3. tous les chemins de la racine à une feuille ont le même nombre de NOIRS.

(2) $\rightarrow$ les rouges ne sont "pas trop" nombreux

(3) $\rightarrow$ l'arbre est "assez" équilibré (*i.e.*, sans les rouges, on a un arbre binaire parfaitement équilibré)

Équilibrage : Red \ Black Tree



Équilibrage : Red \ Black Tree

La "hauteur noire" d'un nœud i notée $\eta(i)$ est le nombre de nœuds noirs internes dans un chemin du nœud à une de ses feuilles

- La def est valide grâce à (3) : "le nombre de nœuds noirs le long d'un chemin de la racine à une feuille est constant"

le nombre de nœuds internes du sous-arbre enraciné en i est au moins égal à $2^{\eta(i)} - 1$

- Par induction sur la hauteur noire
- Si $\eta(i) = 0$ c'est une feuille et le sous arbre enraciné en i contient 0 nœud internes.
- Si $\eta(i) > 0$ alors pour un fils j, $\eta(j) = \eta(i)$ si i est rouge et $\eta(j) = \eta(i) - 1$ si i est noir.
- le sous arbre enraciné en i contient au moins $2 \times (2^{\eta(i)-1} - 1) + 1 = 2^{\eta(i)} - 1$ nœuds

Équilibrage : Red \ Black Tree, Borner la hauteur

- h la hauteur d'un arbre rouge-noir
- Plus de la moitié nœuds vers une feuille sont noirs
- La hauteur noire est au moins $\frac{h}{2}$
- $h \leq 2 \times \eta(root) \leq 2 \times \log_2(n+1)$

Équilibrage : Red \ Black Tree, Insérer

- On insère un élément dans l'arbre binaire de recherche
- Le nouveau nœud est ROUGE
- Attention : la propriété (2) peut être violée (si son père est aussi rouge)
- On va procéder à des rotations pour rétablir la situation

Soit x l'élément inséré et p son père
Dans la suite, on suppose que x et p sont rouges.

Équilibrage : Red \ Black Tree, Insérer (Cas 1)

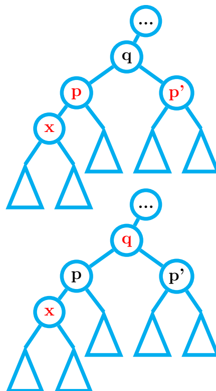
Premier cas : Le père p de l'élément inséré x est la racine

- Il suffit de mettre du NOIR sur la racine !

Équilibrage : Red \ Black Tree, Insérer (Cas 2)

Deuxième cas : le frère p' de p est rouge

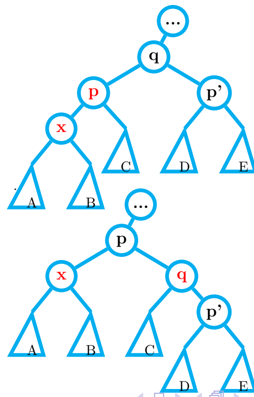
- Les nœuds p et p' deviennent noirs et leur père q devient rouge
- (3) est vérifiée mais
- (2) peut être violée
- On a donc reporté le problème (deux nœuds rouges consécutifs) vers la racine



Équilibrage : Red \ Black Tree, Insérer (Cas 3)

Troisième cas : le frère p' de p est noir

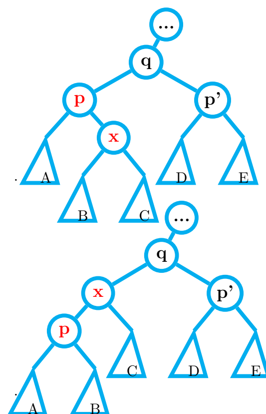
- Sans perte de généralité, p = fils gauche de son père q
- Si x est le fils gauche de p
  - Rotation droite entre p et q
  - p en NOIR et q en rouge



Équilibrage : Red \ Black Tree, Insérer (Cas 3)

Troisième cas : le frère p' de p est noir

- Si x est le fils droit de p
  - Rotation gauche entre x et p (p est alors le fils gauche de x)
  - Retour au cas précédent !
  - Rotation droite entre x et q
  - x noir et q rouge



Aujourd'hui

- Équilibrage d'arbres (Red \ Black)
- Graphes
- Fermeture transitive
- Parcours
- Plus court chemin

Graphes (rappel)

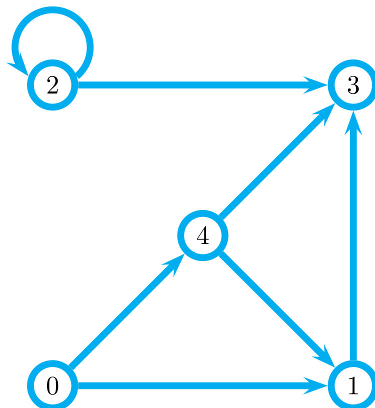
1. Un graphe orienté (digraph) $G = (S, A)$ est un couple formé
   - d'un ensemble de nœuds (ou sommets) S ($|S| = n$)
   - et d'un ensemble $A \subseteq S \times S$ d'arcs
2. Un graphe non orienté $G = (S, A)$ est un couple formé
   - d'un ensemble de nœuds S
   - et d'un ensemble de paires A de sommets appelées arêtes
3. Un graphe peut être valué (sur les arcs/arêtes/sommets)

Comment représenter un graphe ?

- Matrice d'adjacence M $(n \times n)$
  - Sommet numérotés 1, ..., n
  - $M_{ij} = 1$ ssi il existe un arc (arête) de i à j
  - Taille mémoire : $O(n^2)$
- Liste de successeurs
  - On associe à chaque sommet une liste de successeurs
  - Taille mémoire : $O(n^2)$
- Comment représenter de gros graphes peu denses ?

Comment représenter un graphe ?

0	1	0	0	1
0	0	0	1	0
0	0	1	1	0
0	0	0	0	0
0	1	0	1	0



Changer de représentation (graphe orienté)

```
class graphe {
    boolean[][] m;
    int n;
    graphe (int n) {
        this.n = n;
        m = new boolean[n][n]; initialisée à false
    }
    void ajouterArc(int x, int y) {
        m[x][y] = true;
    }
    public void showMat() {
        for (int i = 0; i < n; i++) {
            System.out.print("succ du sommet "+ i + " : ");
            for (int j = 0; j < n; j++)
                if (m[i][j] == true) System.out.print(j + " ");
            System.out.println();
        }
}
```

Changer de représentation (graphe orienté)

Passons aux listes de successeurs

```
liste[] calculSucc() {
    liste[] succ = new liste[n];
    for (int x = 0; x < n; x++)
        for (int y = 0; y < n; y++)
            if (m[x][y] == true)
                succ[x] = new liste(y, succ[x]);
    return succ;
}
```

Aujourd'hui

Équilibrage d'arbres (Red \ Black)

Graphes

Fermeture transitive

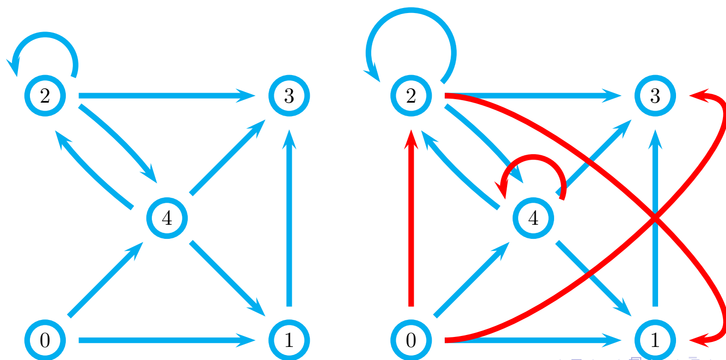
Parcours

Plus court chemin

Calculer la fermeture transitive d'un graphe orienté

- Etant donné un graphe orienté $G = (V, E)$,
- la fermeture transitive $G' = (V, E')$ de G est le plus petit graphe contenant G tel que

$$\forall a, b, c \in V, (a, b) \in E', (b, c) \in E' \Rightarrow (a, c) \in E'$$



Calculer la fermeture transitive d'un graphe orienté

```
void fermetureTransitiveNaiveEtLaide() {
  boolean rienDeNeuf = false;
  while (! rienDeNeuf) {
    rienDeNeuf = true;
    for (int i = 0; i < n; i++)
      for (liste l1 = succ[i]; l1 ≠ null; l1 = l1.suivant) {
        int j = l1.contenu;
        for (liste l2 = succ[j]; l2 ≠ null; l2 = l2.suivant) {
          int k = l2.contenu;
          boolean dejaSuccesseurs = false;
          for (liste l3 = succ[i]; l3 ≠ null; l3 = l3.suivant)
            if (l3.contenu == k)
              dejaSuccesseurs = true;
          if (! dejaSuccesseurs) {
            rienDeNeuf = false;
            succ[i] = new liste(k, succ[i]);
          }}}}}
```

Calculer la fermeture transitive d'un graphe orienté

Soit M la matrice d'adjacence d'un graphe G orienté

- Le nombre de chemins de longueur k de i à j est exactement $(M^k)_{i,j}$
  - Vrai pour $k = 1$
  - Le nombre de chemins de longueur k de i à j est exactement

$$\sum_{(i,u)\in E} \textit{nombre de chemins de longueur } k-1 \textit{ de } u \rightarrow j$$

soit donc

$$\sum_{(i,u)\in E} (M^{k-1})_{u,j} = \sum_{u} M_{i,u} \times (M^{k-1})_{u,j} = (M^k)_{i,j}$$

Calculer la fermeture transitive d'un graphe orienté

- Le nombre de chemins de longueur k de i à j est exactement $(M^k)_{i,j}$
- La matrice $\sum_{u=1}^{k}(M^u)$ représente donc le nombre de chemins distincts de longueur $\leq k$ d'un sommet à un autre
- Rappel : un chemin élémentaire est un chemin dans lequel on ne rencontre pas deux fois le même sommet
- La longueur d'un chemin élémentaire est donc $O(n)$
- $\sum_{u=1}^{n}(M^u)$ représente donc le nombre de chemins distincts de i à j dans G

Calculer la fermeture transitive d'un graphe orienté

Pour calculer $\sum_{u=1}^{n}(M^u)$,

- $T = I$ une matrice $n \times n$
- Pour $k = 1$ à $n - 1$ faire
- $T = I + MT$
- puis $T = MT$

Complexité associée : $O(n^4)$

Calculer la fermeture transitive d'un graphe orienté

Une méthode inspirée de la technique de Floyd

- Méthode ascendante
- Déterminer pour k croissant de 1 à n et pour tous les couples de sommets i, j, si oui ou non, il existe un chemin de i à j dont les sommets intermédiaires sont dans $\{1, ..., k\}$
- De tels chemins sont des k-chemins
- Soit alors $d[i, j, k]$ un Booléen qui correspoind à l'existence d'un k chemin entre j et k
- Rq : On cheche les valeurs $d[i, j, n]$
- Rq : $d[i, j, 0]$ vaut `true` ssi $M_{ij} = 1$
- Récurrence :
  $d[i, j, k] = d[i, j, k-1] \vee (d[i, k, k-1] \wedge d[k, j, k-1])$

Calculer la fermeture transitive d'un graphe orienté

Implémentons l'algorithme de Floyd

```
void fermetureTransitive () {
  for (int k = 0; k < n; ++k)
    for (int i = 0; i < n; ++i)
      for (int j = 0; j < n; ++j)
          m[i][j] = m[i][j] || (m[i][k] && m[k][j]);
}
```

- Complexité $O(n^3)$

Aujourd'hui

Équilibrage d'arbres (Red \ Black)

Graphes

Fermeture transitive

Parcours

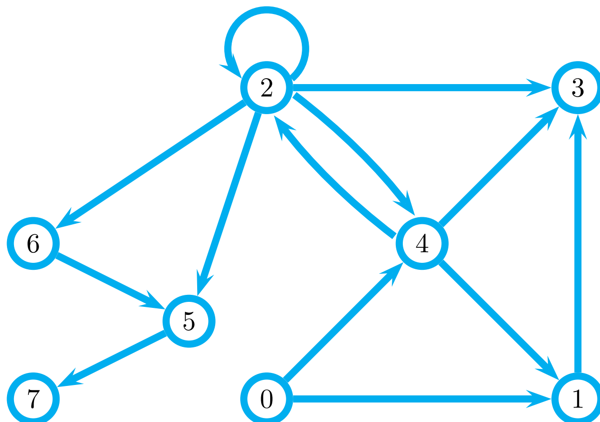
Plus court chemin

Parcourir un graphe

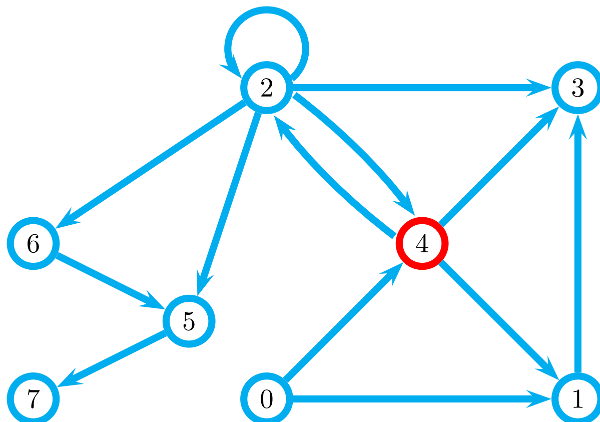
- Etant donné un graphe (sous la forme de listes d'adjacence), et un sommet de départ s,
- parcourir tous les sommets atteignables depuis s et imprimer les sommets dans l'ordre du parcours
- On va procéder par "inondations successives" à partir de s

```
succ du sommet 0 : 4 1
succ du sommet 1 : 3
succ du sommet 2 : 6 5 4 3 2
succ du sommet 3 : null
succ du sommet 4 : 3 2 1
succ du sommet 5 : 7
succ du sommet 6 : 5
succ du sommet 7 : null
```

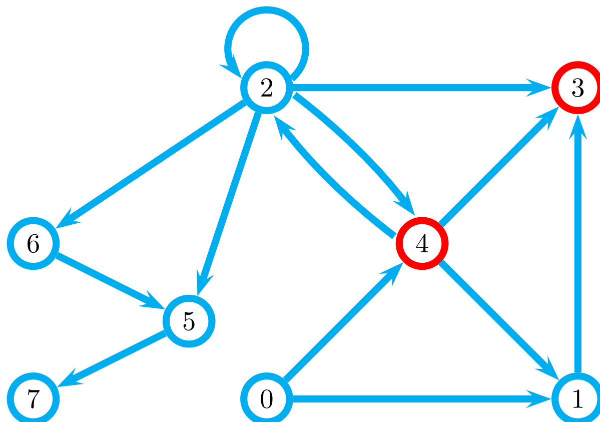
Parcourir un graphe



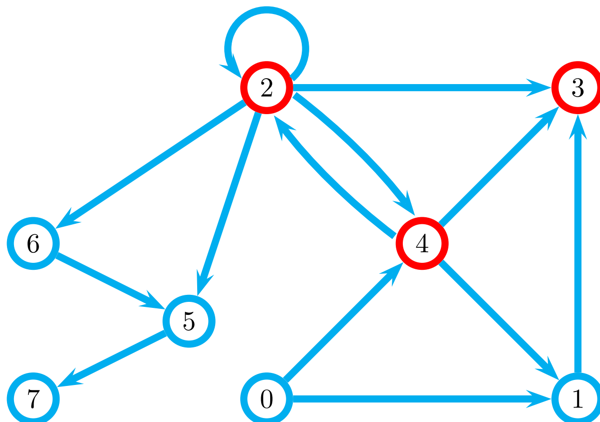
Parcourir un graphe



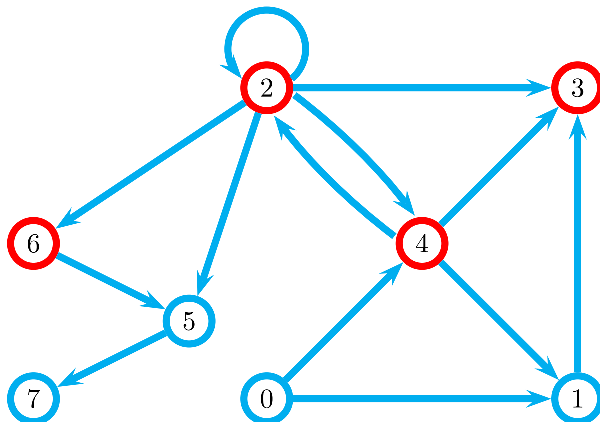
Parcourir un graphe



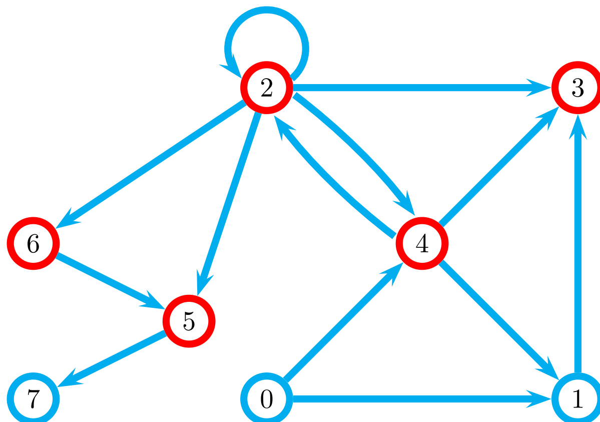
Parcourir un graphe



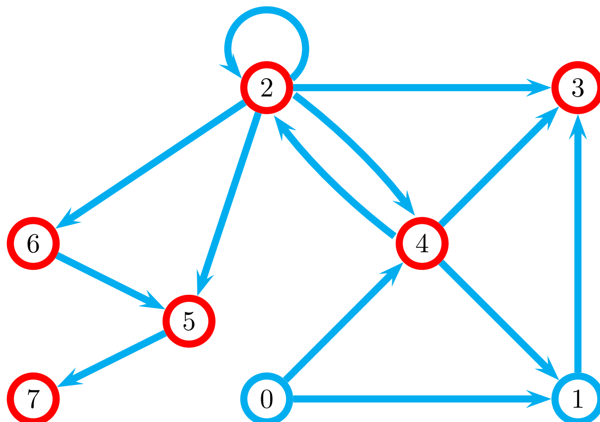
Parcourir un graphe



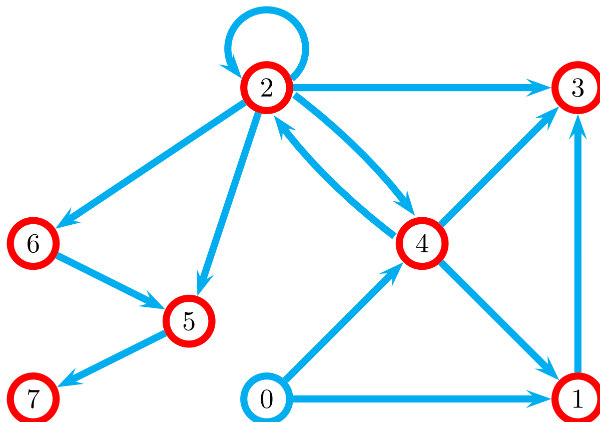
Parcourir un graphe



Parcourir un graphe



Parcourir un graphe



Parcourir un graphe

- Deux couleurs :
  - rouge (le sommet a déjà été visité)
  - noir (le sommet n'a pas été visité)
- Au début, tous les sommets sont noirs
- Codage récursif. Visiter un sommet c'est :
  - Le passer en rouge
  - Visiter ses voisins qui sont encore noirs
- très proche d'un parcours d'arbre préfixe.

Ce qu'on a vu = Depth First Search

Parcourir un graphe

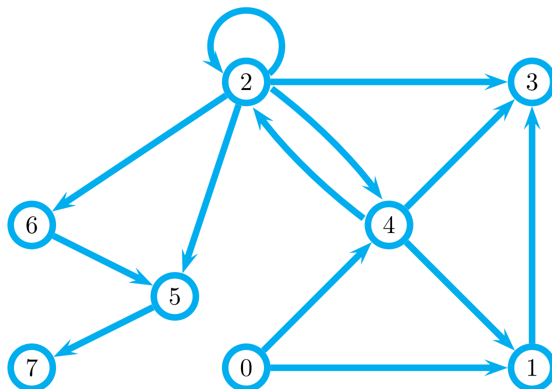
```
boolean[] vu;
void initialisation() {
    vu = new boolean[n];
    for (int i = 0; i < n; ++i)
        vu[i] = false; }
void dfs (int x) {
    vu[x] = true;
    for (liste ls = succ[x]; ls ≠ null; ls = ls.suivant) {
        int y = ls.contenu;
        if (! vu[y])
            dfs(y); }}
// plus loin dans le main
g.initialisation();
g.dfs(4);
```

Parcourir un graphe

- Un DFS lancé à partir d'un sommet x visite tous les sommets y tels qu'il existe un chemin de x à y
  - tous les sommets du graphe ne sont donc pas obligatoirement visités
- La complexité d'un DFS est $O(n+m)$
  - On ne visite un sommet qu'une fois
  - et on emprunte donc les arcs adjacents d'un sommet qu'une fois

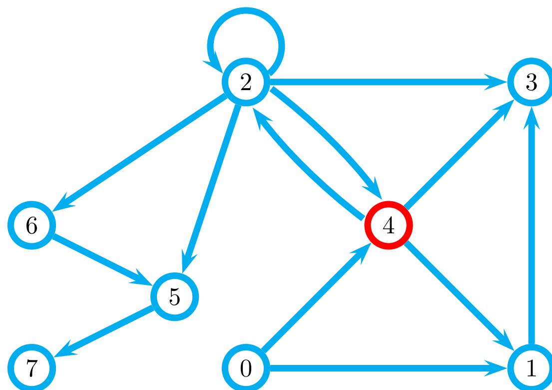
Parcourir un graphe

En rouge : les arcs empruntés par DFS



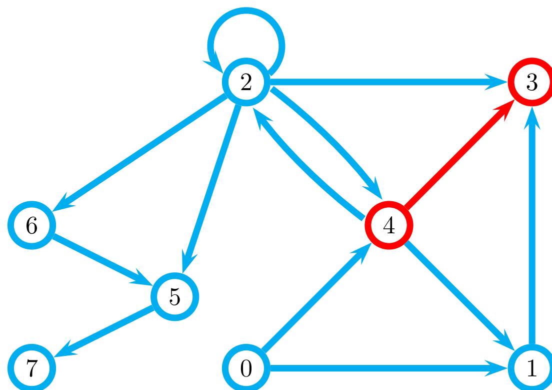
Parcourir un graphe

En rouge : les arcs empruntés par DFS



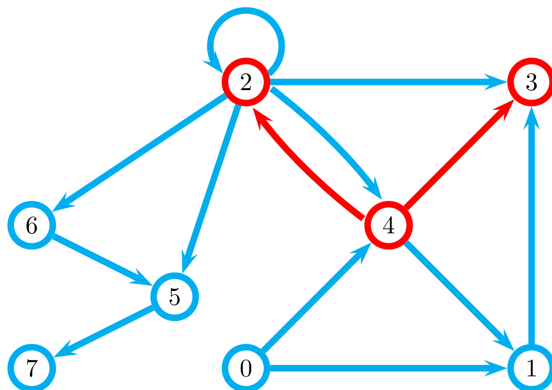
Parcourir un graphe

En rouge : les arcs empruntés par DFS



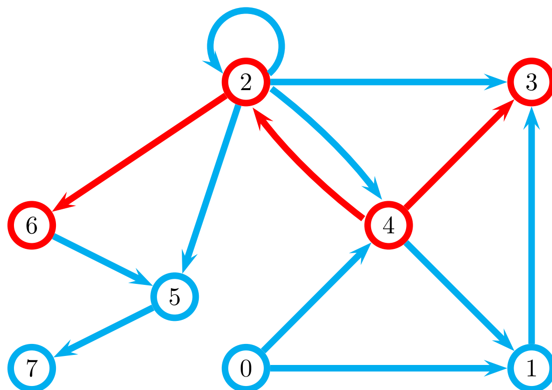
Parcourir un graphe

En rouge : les arcs empruntés par DFS



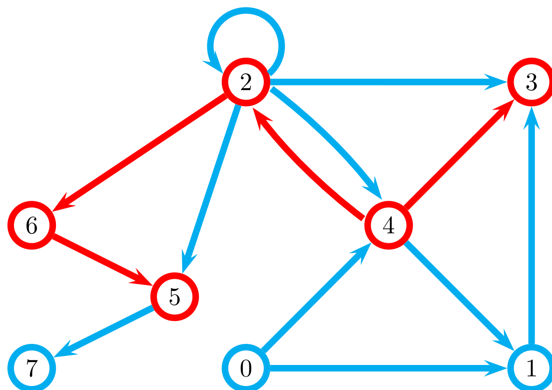
Parcourir un graphe

En rouge : les arcs empruntés par DFS



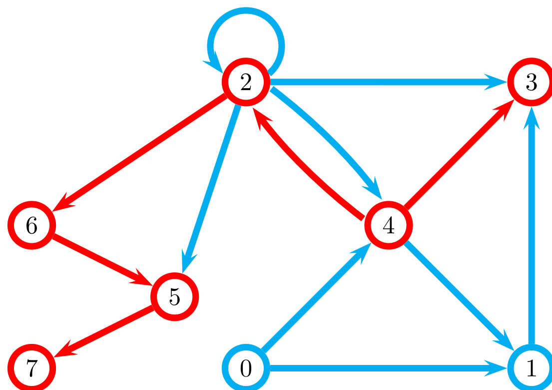
Parcourir un graphe

En rouge : les arcs empruntés par DFS



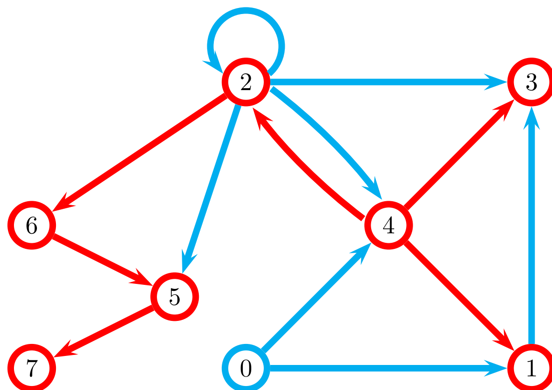
Parcourir un graphe

En rouge : les arcs empruntés par DFS



Parcourir un graphe

En rouge : les arcs empruntés par DFS



Parcourir un graphe

- Le sous-graphe rouge est un arbre
- Pourquoi ?
  - Peut-on avoir plusieurs arcs rouges issus d'un même sommet rouge ?
  - Peut-on avoir plusieurs arcs rouges qui pointent sur un même sommet rouge ?

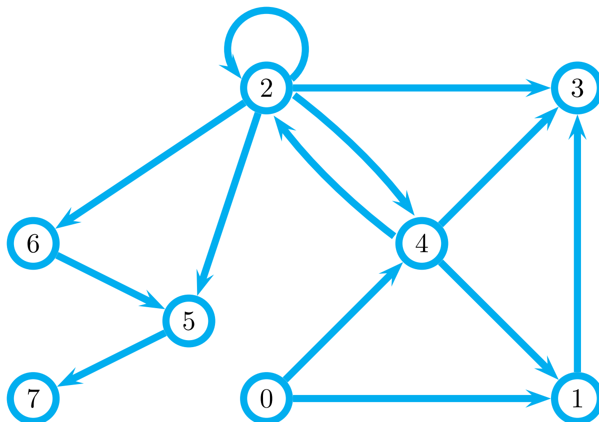
Parcourir un graphe pour (re)numéroter les sommets

```
int[] num; int numCourant;
void initialisation() {
    num = new int[n];
    numCourant = 0;
    for (int x = 0; x < n; ++x)
        num[x] = -1; // convention -1 = pas vu
}
void dfs (int x) {
    num[x] = numCourant; // numérotation préfixe
    numCourant++;
    for (liste ls = succ[x]; ls ≠ null; ls = ls.suivant) {
        int y = ls.contenu;
        if (num[y] == -1)
            dfs(y);
    }
}
```

Parcourir un graphe avec une pile

```
void dfs(int a) {
    Pile pile = new Pile ();
    pile.ajouter (a);
    while (!pile.estVide ()) {
        a = pile.valeur();
        pile.supprimer();
        if (! vu[a]) {
            for (liste fl = succ[a]; fl ≠ null; fl = fl.suivant)
                pile.ajouter(fl.contenu);
            vu[a] = true;
        }
    }
}
```

Parcourir un graphe avec une pile

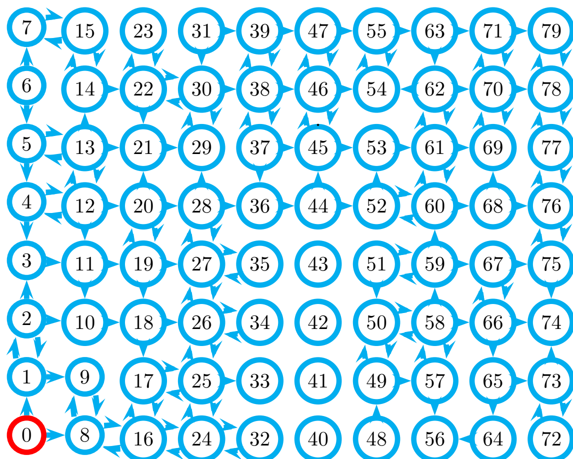



```
Empile 4
Depile 4
Empile 3
Empile 2
Empile 1
Depile 1
Depile 2
Empile 6
Empile 5
Depile 5
Empile 7
Depile 7
Depile 6
Depile 3
```

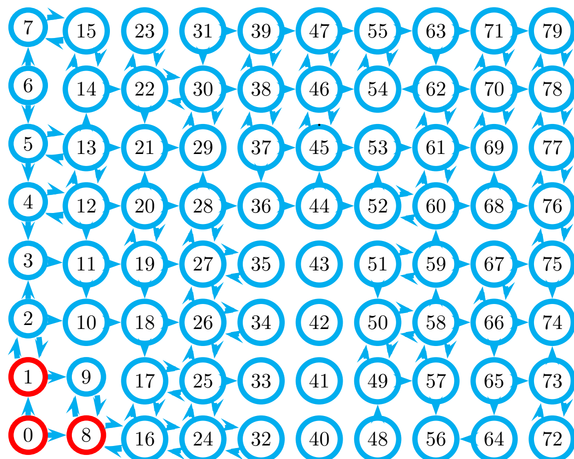
Parcourir un graphe en largeur d'abord

```
void bfs(int a) {
    File file = new File ();
    file.ajouter (a);
    while (!file.estVide ()) {
        a = file.valeur();
        file.supprimer();
        if (! vu[a]) {
            for (liste fl = succ[a]; fl ≠ null; fl = fl.suivant)
                file.ajouter(fl.contenu);
            vu[a] = true;
        }
    }
}
```

Parcourir un graphe en largeur d'abord

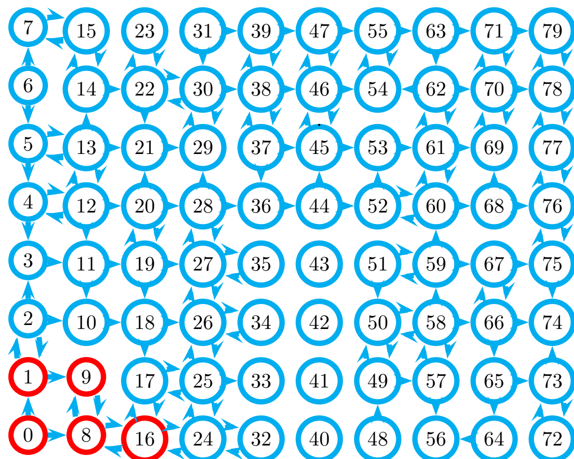


Parcourir un graphe en largeur d'abord

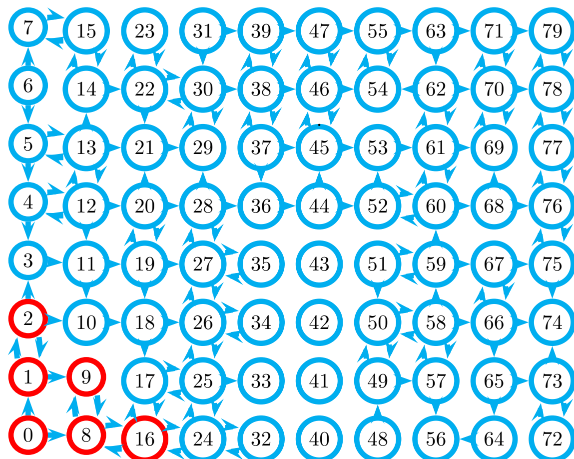


.

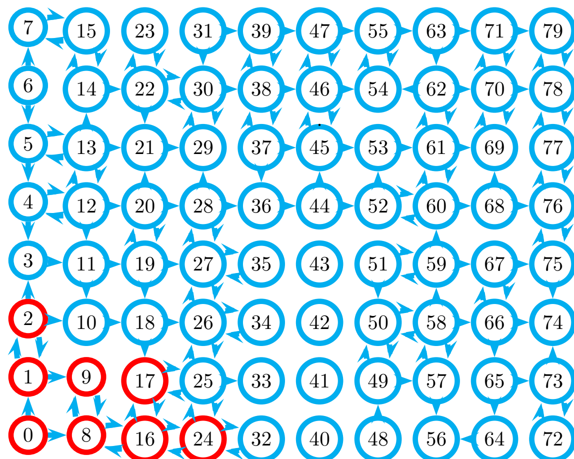
Parcourir un graphe en largeur d'abord



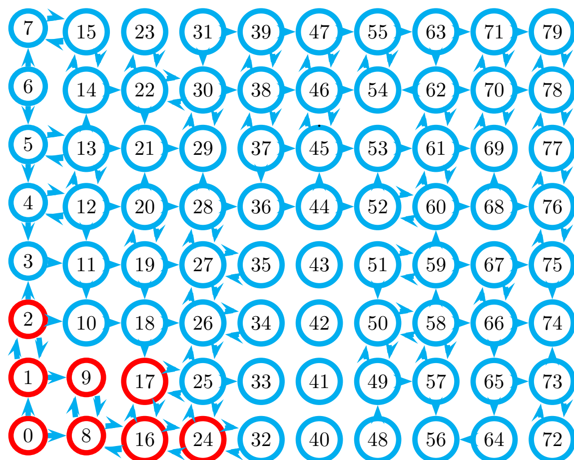
Parcourir un graphe en largeur d'abord



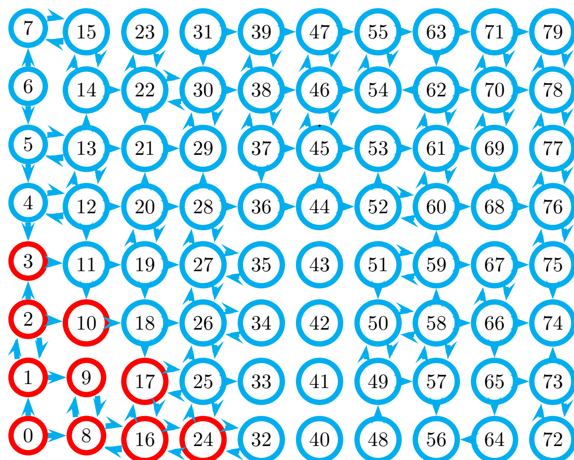
Parcourir un graphe en largeur d'abord



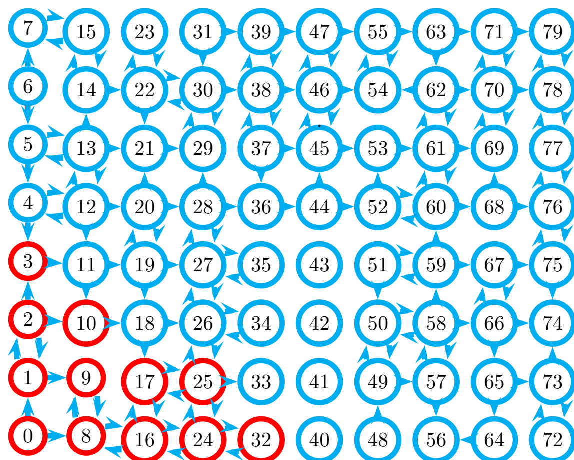
Parcourir un graphe en largeur d'abord



Parcourir un graphe en largeur d'abord

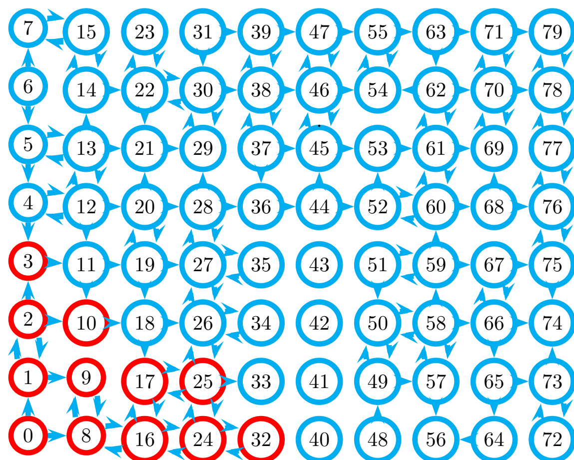


Parcourir un graphe en largeur d'abord

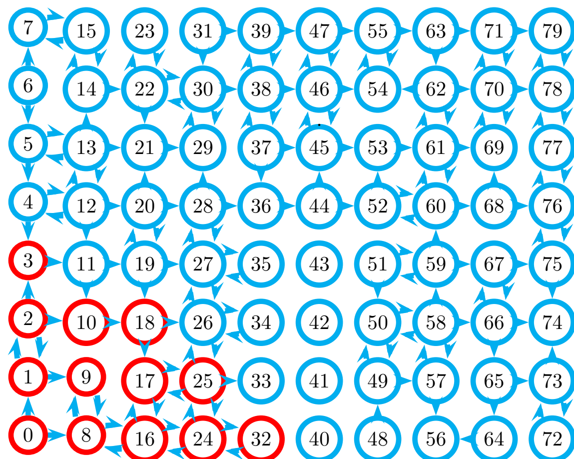


.

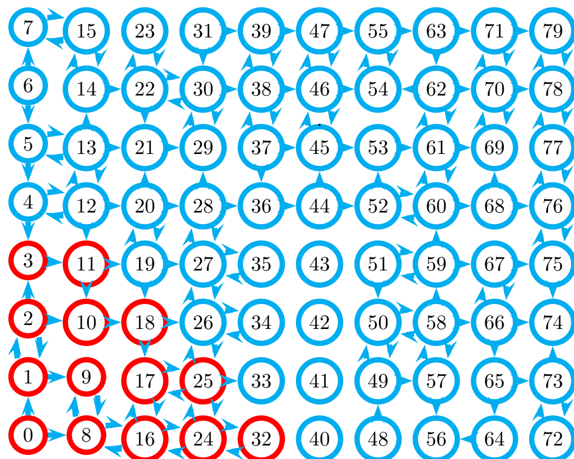
Parcourir un graphe en largeur d'abord



Parcourir un graphe en largeur d'abord

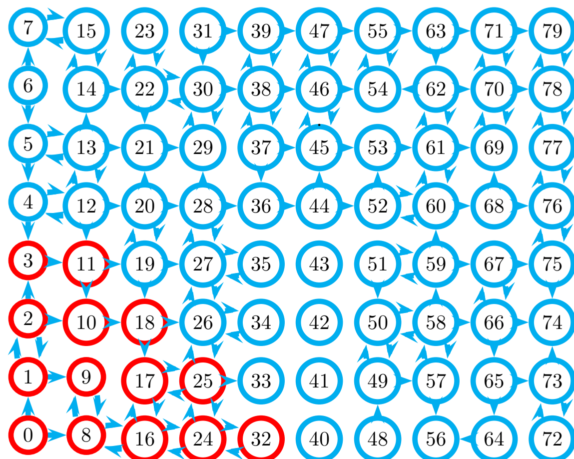


Parcourir un graphe en largeur d'abord

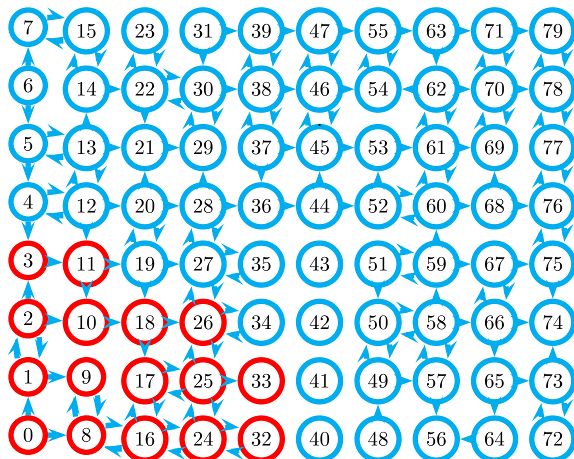


.

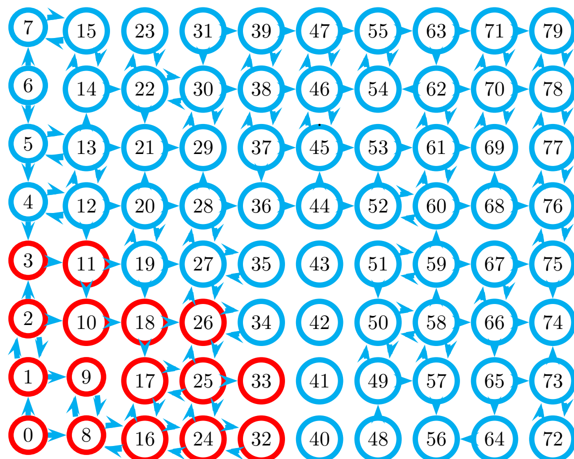
Parcourir un graphe en largeur d'abord



Parcourir un graphe en largeur d'abord

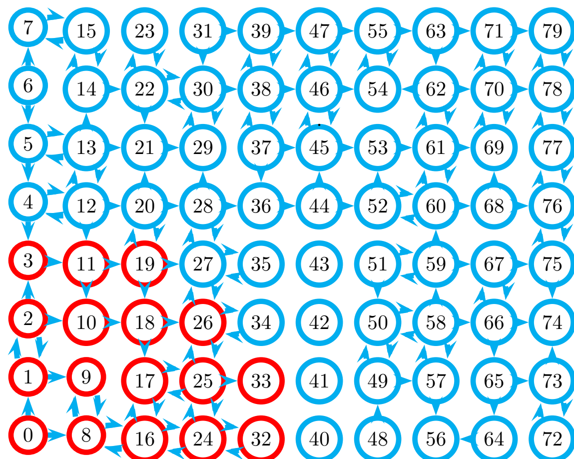


Parcourir un graphe en largeur d'abord

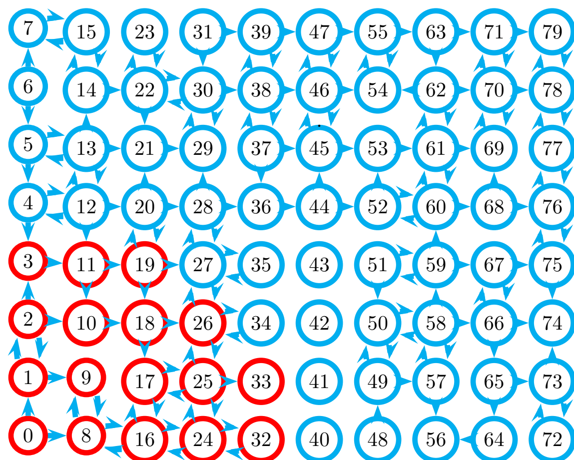


.

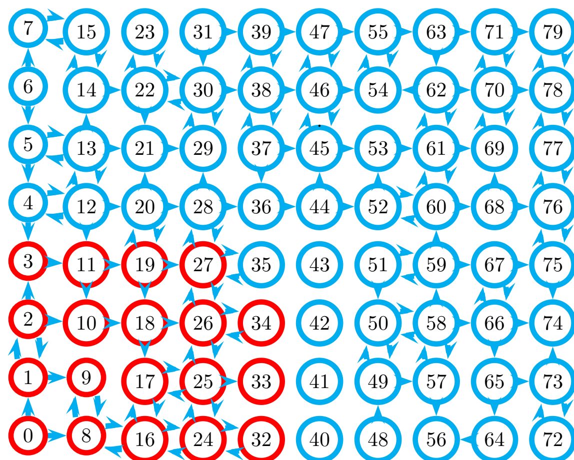
Parcourir un graphe en largeur d'abord



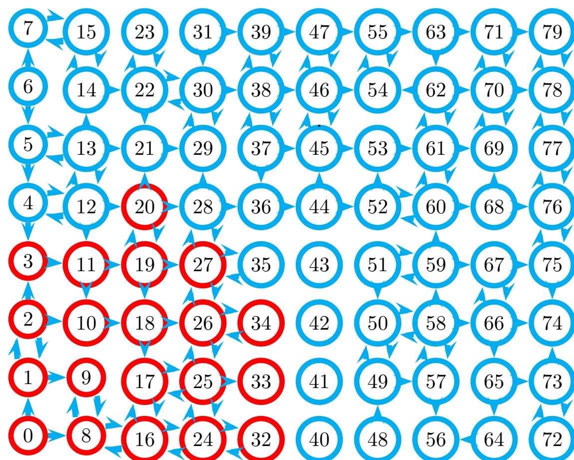
Parcourir un graphe en largeur d'abord



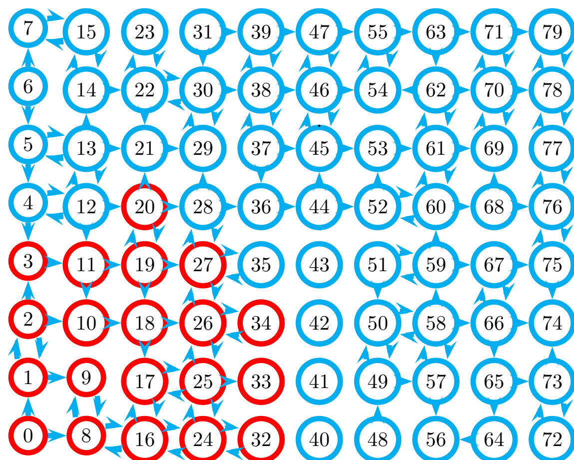
Parcourir un graphe en largeur d'abord



Parcourir un graphe en largeur d'abord

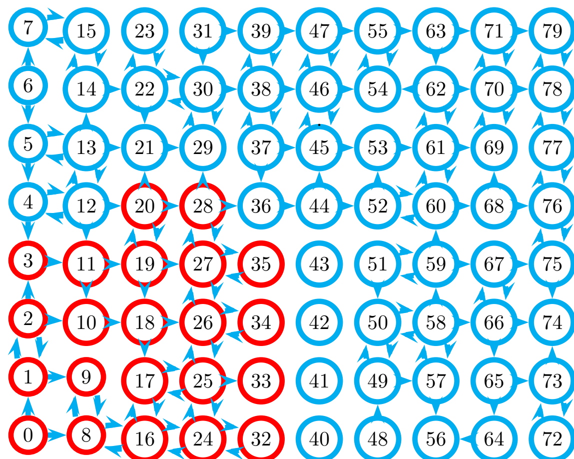


Parcourir un graphe en largeur d'abord



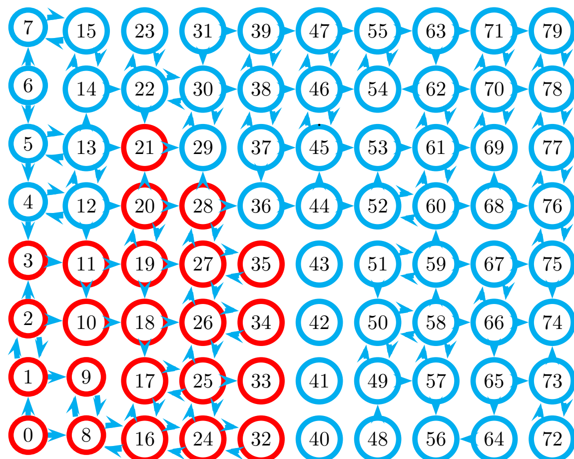
.

Parcourir un graphe en largeur d'abord



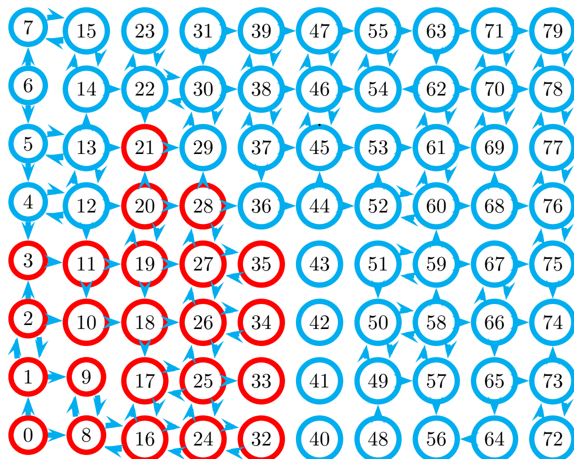
.

Parcourir un graphe en largeur d'abord



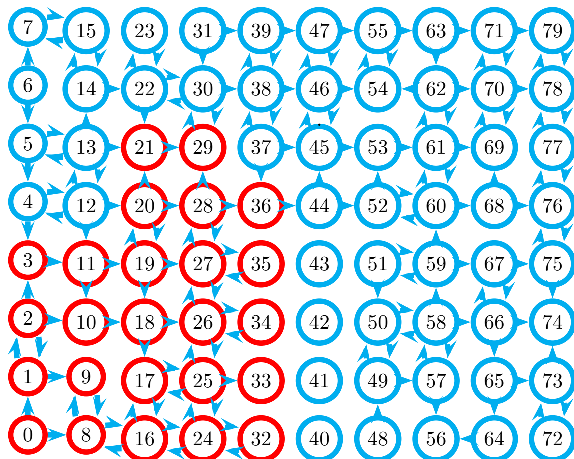
.

Parcourir un graphe en largeur d'abord



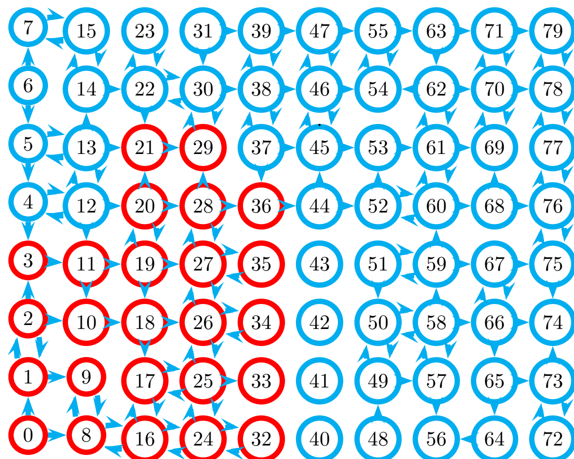
.

Parcourir un graphe en largeur d'abord

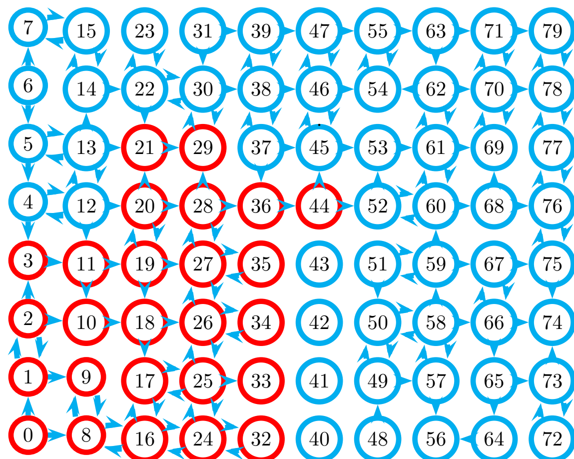


.

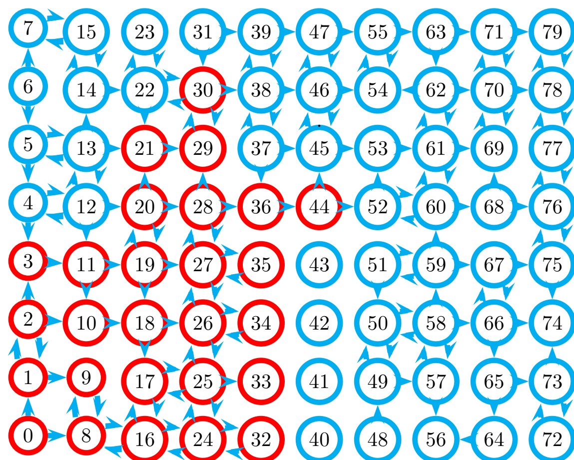
Parcourir un graphe en largeur d'abord



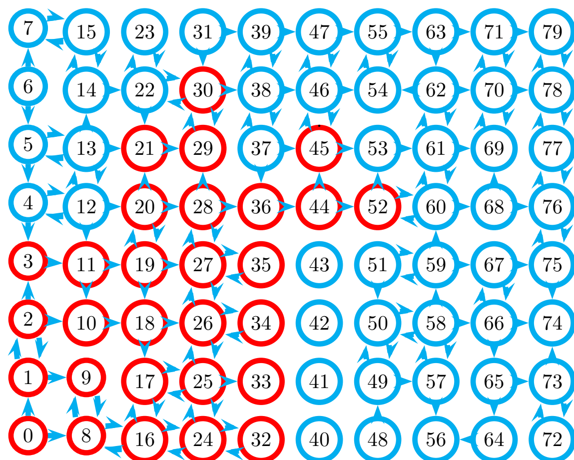
Parcourir un graphe en largeur d'abord



Parcourir un graphe en largeur d'abord

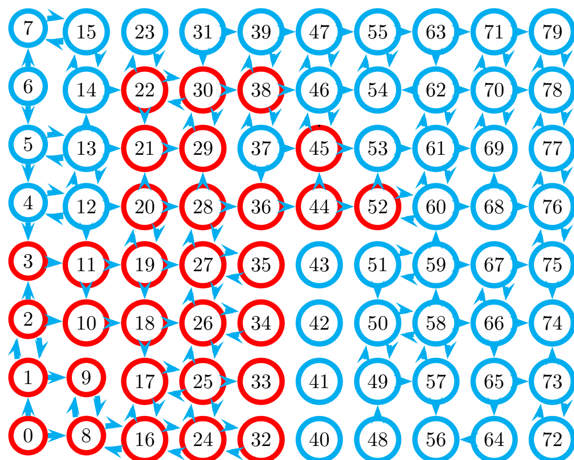


Parcourir un graphe en largeur d'abord



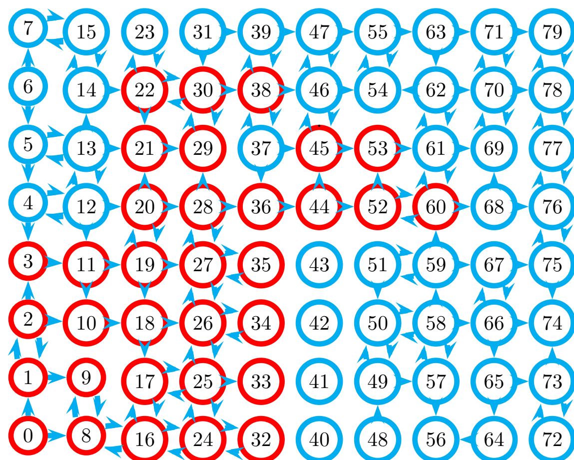
.

Parcourir un graphe en largeur d'abord



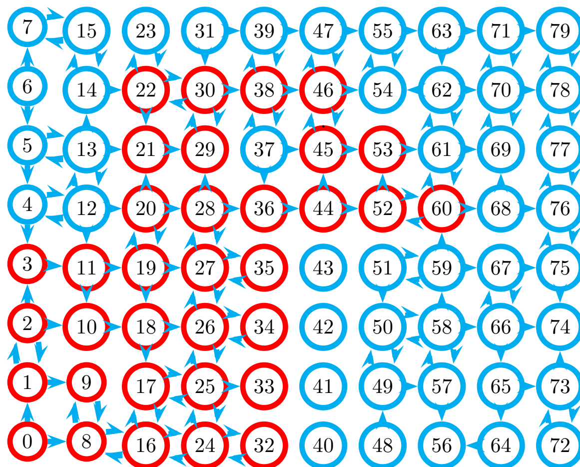
.

Parcourir un graphe en largeur d'abord



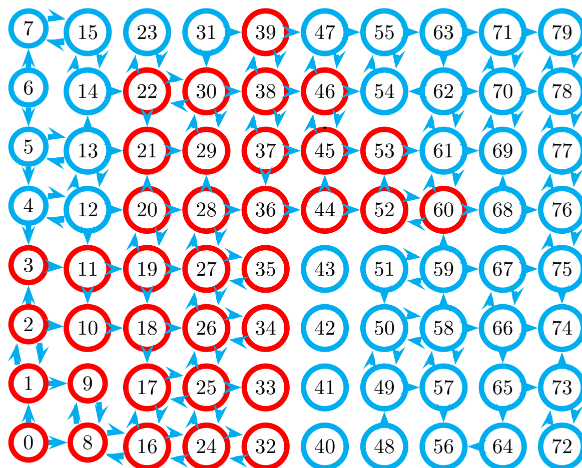
.

Parcourir un graphe en largeur d'abord



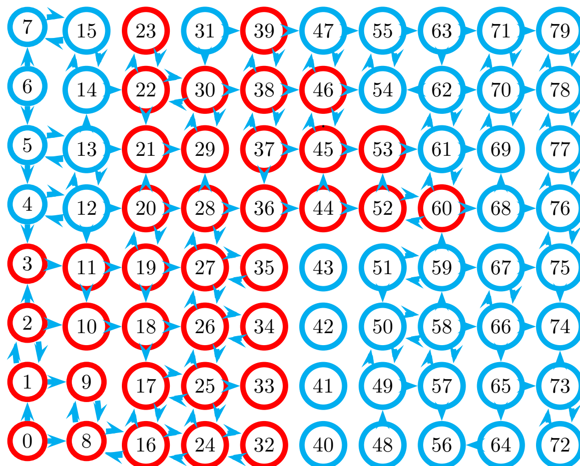
.

Parcourir un graphe en largeur d'abord



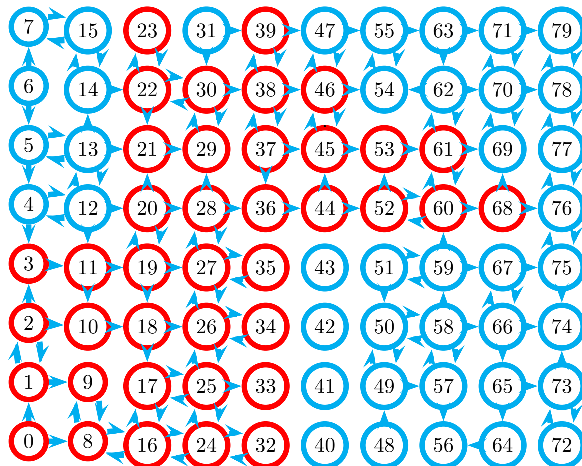
.

Parcourir un graphe en largeur d'abord



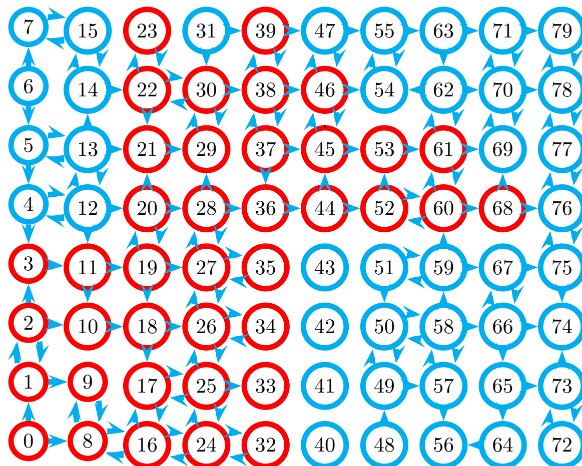
.

Parcourir un graphe en largeur d'abord



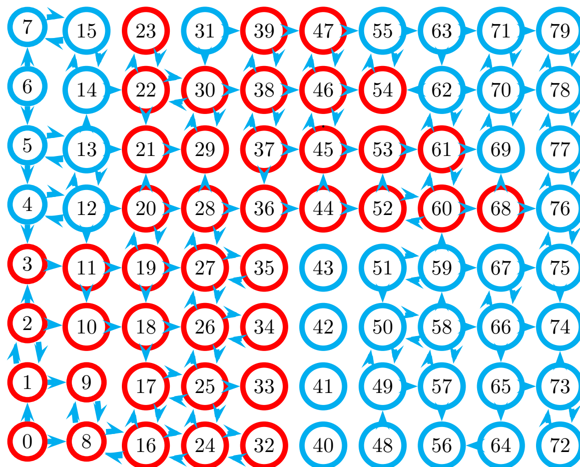
.

Parcourir un graphe en largeur d'abord



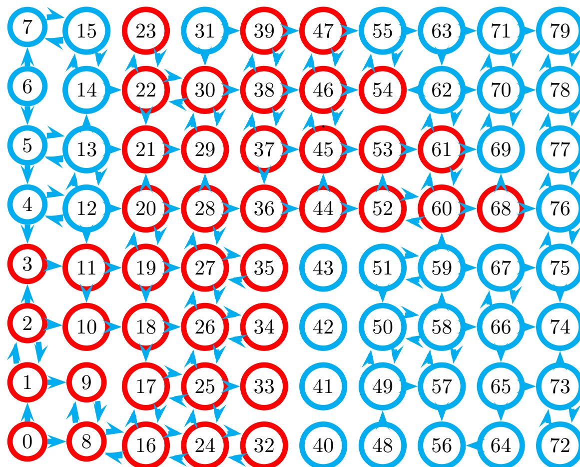
.

Parcourir un graphe en largeur d'abord



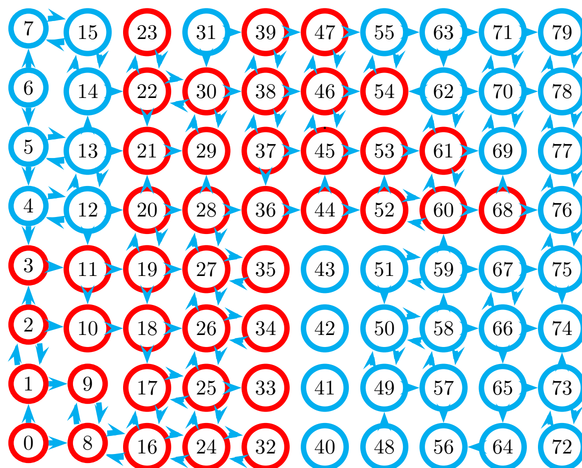
.

Parcourir un graphe en largeur d'abord



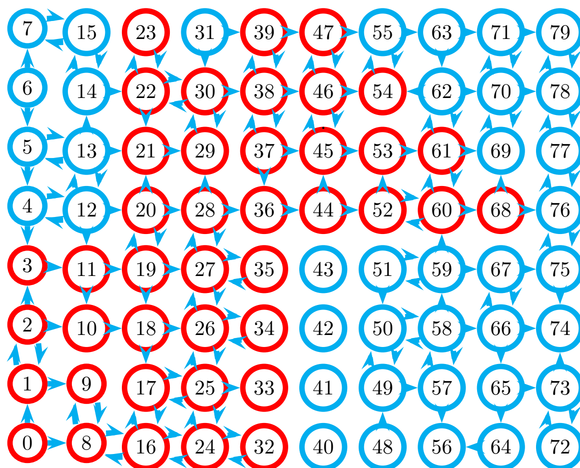
.

Parcourir un graphe en largeur d'abord



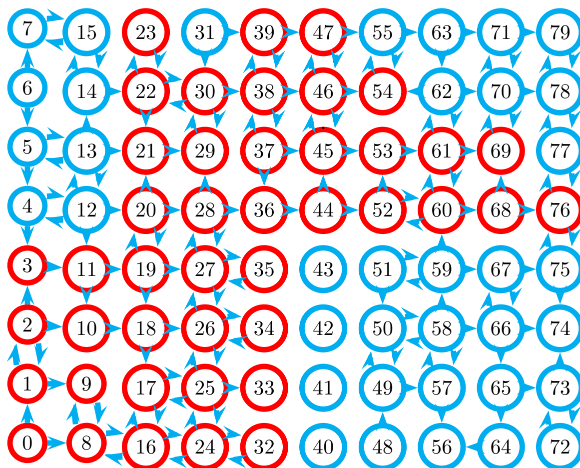
.

Parcourir un graphe en largeur d'abord



.

Parcourir un graphe en largeur d'abord



.

Aujourd'hui

- Équilibrage d'arbres (Red \ Black)
- Graphes
- Fermeture transitive
- Parcours
- Plus court chemin

BFS et plus court chemin

Problématique : Etant donné un sommet s, trouver le plus court chemin de s à tout autre sommet x dans le graphe orienté $G = (V, E)$

- Ici, longueur d'un chemin = nombre d'arcs
  - le problème valué est plus complexe (des détails en IF 431)
- Soit $\delta(u)$ la distance de s à u.
- $\delta(s) = 0$
- $\delta(u) = \min_{v:(v,u)\in E} (\delta(v) + 1)$
- C'est "presque" une formule de récurrence
  - Hélas, avec un cycle ...

BFS et plus court chemin

- Pour ne pas "cycler"
- visiter le graphe en calculant les distances des voisins immédiats de s
- puis les distances de s aux voisins de ses voisins
- *etc.*

C'est un BFS

BFS et plus court chemin

distance[] (la distance d'un sommet au sommet "origine")

```
void bfs(int s) {
    for (int i = 0; i < n; i++) distance[i] = Integer.MAX_VALUE;
    distance[s] = 0;
    File file = new File ();
    file.ajouter(s);
    vu[s] = true; distance[s] = 0;
    while (!file.estVide()) {
        s = file.valeur(); file.supprimer();
        for (liste fl = succ[s]; fl ≠ null; fl = fl.suivant) {
            int b = fl.contenu;
            if (! vu[b]) {
                file.ajouter(b); vu[b] = true;
                distance[b] = distance[s] + 1; }}}}
```

A la fin de l'algorithme quels sont les *i* tq. distance[i] = Integer.MAX_VALUE ?

BFS et plus court chemin

BFS calcule en $O(|E|)$ les plus courts chemins de s à tous les sommets de G

- Soit $\delta(i)$ la distance de s à i dans G (il existe un chemin de s à i)
- Soit $d(x)$ la valeur de `distance[x]` à la fin de BFS
- Montrons que $\forall i$ pour lequel il existe un chemin de s à i, $d(i) = \delta(i)$
- Rq. BFS est une procédure de marquage et donc tous les sommets atteignables à partir de s sont marqués.
- $d(i)$ est bien la longueur d'un chemin de s à i dans G (chemin dans l'arbre rouge)
- Et donc $d(i) \geq \delta(i)$

BFS et plus court chemin

- Si un sommet u est ajouté à la file F avant v, alors $d(u) =$ `distance[u]` est $\leq$ à $d(v) =$ `distance[v]`
  - Par induction : (regarder les pères dans l'arbre rouge de u et de v)
- Soit y tq $d(y) > \delta(y)$ avec $\delta(y)$ minimal
- Lors du BFS, y a été visité en passant par x avec $(x, y) \in E$
- et donc, $d(y) = d(x) + 1 = \delta(x) + 1$
- $d(y) > \delta(y)$ donc, $\exists z$ tq. $\delta(y) = \delta(z) + 1$ et $d(x) > d(z)$.
- Comme $\delta(y)$ est minimal, $\delta(x) = d(x)$ et $\delta(z) = d(z)$
- z a donc été ajouté à F avant x.
- y a donc été marqué depuis z. Absurde !

BFS et plus court chemin : Le TD

Plan du métro de Tokyo

Bureau des Transports, Gouvernement de la Ville de Tokyo

Tokyo Metro

フランス語

Un bilan

- Java
- Structures dynamiques
- Listes, Arbres
- Applications
- Quelques points sur les graphes
- Quelques exemples
  - Analyse expérimentale
  - Analyse en moyenne

Un bilan

- Vous avez vu beaucoup de choses et vous avez LARGEMENT le niveau pour suivre INF 431 (Morain + Steyaert)
- L'informatique est une science ($\neq$ hackers) ; comme les autres, elle utilise des outils variés (maths, des math apps, des stats, etc...)
- C'est vrai qu'on travaille dans les cours d'info mais ...