LINFO 1121
DATA STRUCTURES AND ALGORITHMS

UCLouvain

Union-Find, Heap, Huffman

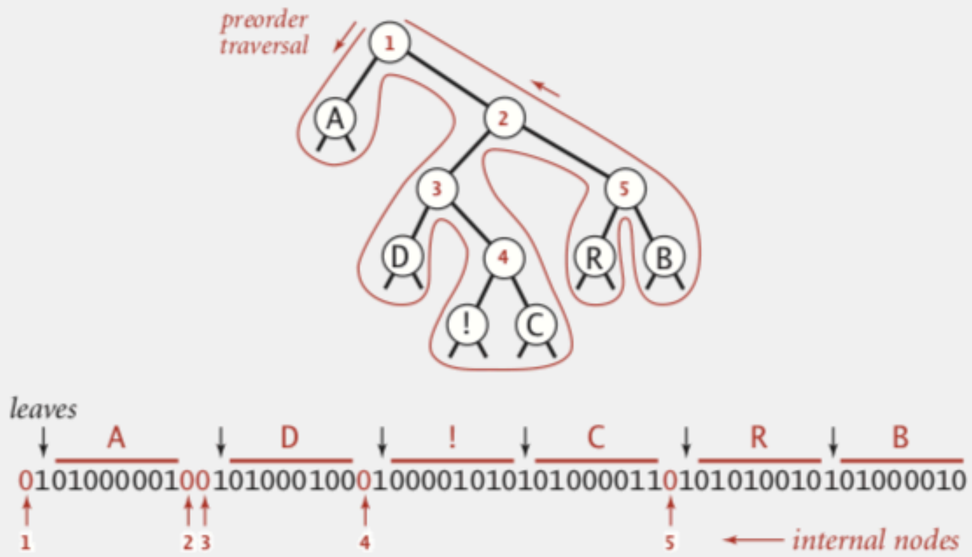
Pierre Schaus

http://algs4.cs.princeton.edu

5.5 HUFFMAN CODING DEMO

Huffman: Entête

- Nécessite un fichier d'entête qui contient un arbre sérialisé (parcourt préfixe) avec l'information nécessaire au codage.



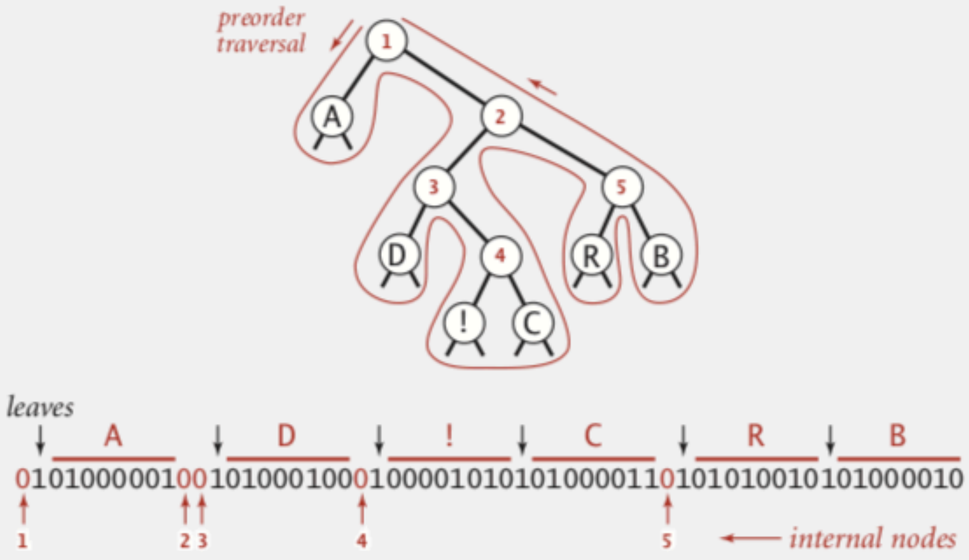
Using preorder traversal to encode a trie as a bitstream

```
private static void writeTrie(Node x)
{
   if (x.isLeaf())
   {
      BinaryStdOut.write(true);
      BinaryStdOut.write(x.ch, 8);
      return;
   }
   BinaryStdOut.write(false);
   writeTrie(x.left);
   writeTrie(x.right);
}
```

5.2.1 Huffman

- Pensez-vous qu'il serait plus ou moins intéressant d'un point de vue mémoire de stoker pour chaque symbole, son codage binaire plutôt que l'arbre sérialisé ?

- *Non: exemple*



Using preorder traversal to encode a trie as a bitstream

```
private static void writeTrie(Node x)
{
   if (x.isLeaf())
   {
      BinaryStdOut.write(true);
      BinaryStdOut.write(x.ch, 8);
      return;
   }
   BinaryStdOut.write(false);
   writeTrie(x.left);
   writeTrie(x.right);
}
```

5.2.2 Huffman

Peut-on gagner encore en taux de compression si l'on réapplique l'algorithme de compression de Huffman sur un fichier déjà comprimé une première fois ? Que se passe-t-il dans ce cas ? Cela ouvre-t-il la porte vers un algorithme de compression récursif et optimal ?

Pourquoi Huffman compresse-t-il bien ?

5.2.2 Huffman

Peut-on gagner encore en taux de compression si l'on réapplique l'algorithme de compression de Huffman sur un fichier déjà comprimé une première fois ? Que se passe-t-il dans ce cas ? Cela ouvre-t-il la porte vers un algorithme de compression récursif et optimal ?

Pourquoi Huffman compresse-t-il bien ?

- *Lorsque le nombre de symboles à compresser est moindre que l'ensemble des caractères ascii (8 bits, 256 symbols) et/ou lorsqu'il y a des grosses différences dans le nombre d'occurrences des symboles (typiquement le cas pour les langues naturelles).*
- *Après une première compression,*
  - *on perd généralement ces deux propriétés car ça n'est plus un texte en langue naturelle.*
  - *il faut aussi ajouter le fait qu'on va vouloir compresser l'entête qui ne va pas être bien compressé.*
- *En conclusion: aucune garantie que cela va être utile et les compteurs devraient être beaucoup plus uniforme et de plus on ajoute une entête.*

5.2.3 Huffman

- Quel est, approximativement, le taux de compression obtenu si l'on applique l'algorithme de compression de Huffman sur un un fichier comportant une seule chaîne composée du caractère "a" répété un million (+- 2^20) de fois, suivi du caractère *b* présent une seule fois ?

5.2.3 Huffman

- Quel est, approximativement, le taux de compression obtenu si l'on applique l'algorithme de compression de Huffman sur un un fichier comportant une seule chaîne composée du caractère "a" répété un million (+- 2^20) de fois, suivi du caractère *b* présent une seule fois ?

Hypotheses: input = ASCI 8 bit (256 symboles). On néglige l'encodage de l'entête car le fichier est très grand.

La compression est donc de 1/8 (il n'a besoin que de 0 et 1 dans sa code-table)

5.2.3 Huffman

- Le taux de compression obtenu varie-t-il avec la longueur du fichier (par exemple, si le caractère *a* est répété deux millions de fois) ?
  - *?*
- A votre avis, quel est le nombre minimal de bits nécessaires pour représenter sous forme comprimée ce fichier ?
  - *?*
- Peut-on adapter la technique de compression par un codage de Huffman en mesurant la fréquence d'autre chose que les caractères présents ?
  - *?*
- Peut-on utiliser une autre technique de compression qui serait plus efficace dans ce cas particulier ?
  - *?*

5.2.3 Huffman

- Le taux de compression obtenu varie-t-il avec la longueur du fichier (par exemple, si le caractère *a* est répété deux millions de fois) ?
  - *Non, ça reste 1/8*
- A votre avis, quel est le nombre minimal de bits nécessaires pour représenter sous forme comprimée ce fichier ?
  - *Si on sait que la forme du fichier est $[a*b]$, on n'a besoin que d'encoder un nombre = le nombre d'occurence de a.*
- Peut-on adapter la technique de compression par un codage de Huffman en mesurant la fréquence d'autre chose que les caractères présents ?
  - *Oui on peut compter n'importe quoi qui utilise un nombre constant de nombre de bits, par exemple des couleurs dans une image.*
- Peut-on utiliser une autre technique de compression qui serait plus efficace dans ce cas particulier ?
  - *Oui on peut par exemple avoir un encodage du type: "character" suivit du nombre d'occurence consécutives. Par exemple "aaabbbbbbbccaaa" devient "a3b7c2a3".*

Reprenons notre fichier

- aaaaaaa (x2^10)

- Si on applique Huffman 2x quel est le taux de compression ?

5.2.4

Imaginez une implémentation d'une file de priorité par une heap à l'aide d'une structure chaînée pour représenter l'arbre binaire essentiellement complet correspondant au tas.

- Combien de liens sont nécessaires dans chaque noeud ?
  - *?*
- Écrivez le code des méthodes insert, delMax. Complexité.
  - *?*
- Est-il utile de donner la taille max N dans le constructeur ? Comment faites-vous pour ajouter un nouveau noeuds dans la heap ou retirer le prochain noeud ? Est-ce que cela peut être fait au départ de la taille courante de la heap ?
  - *?*

5.2.4

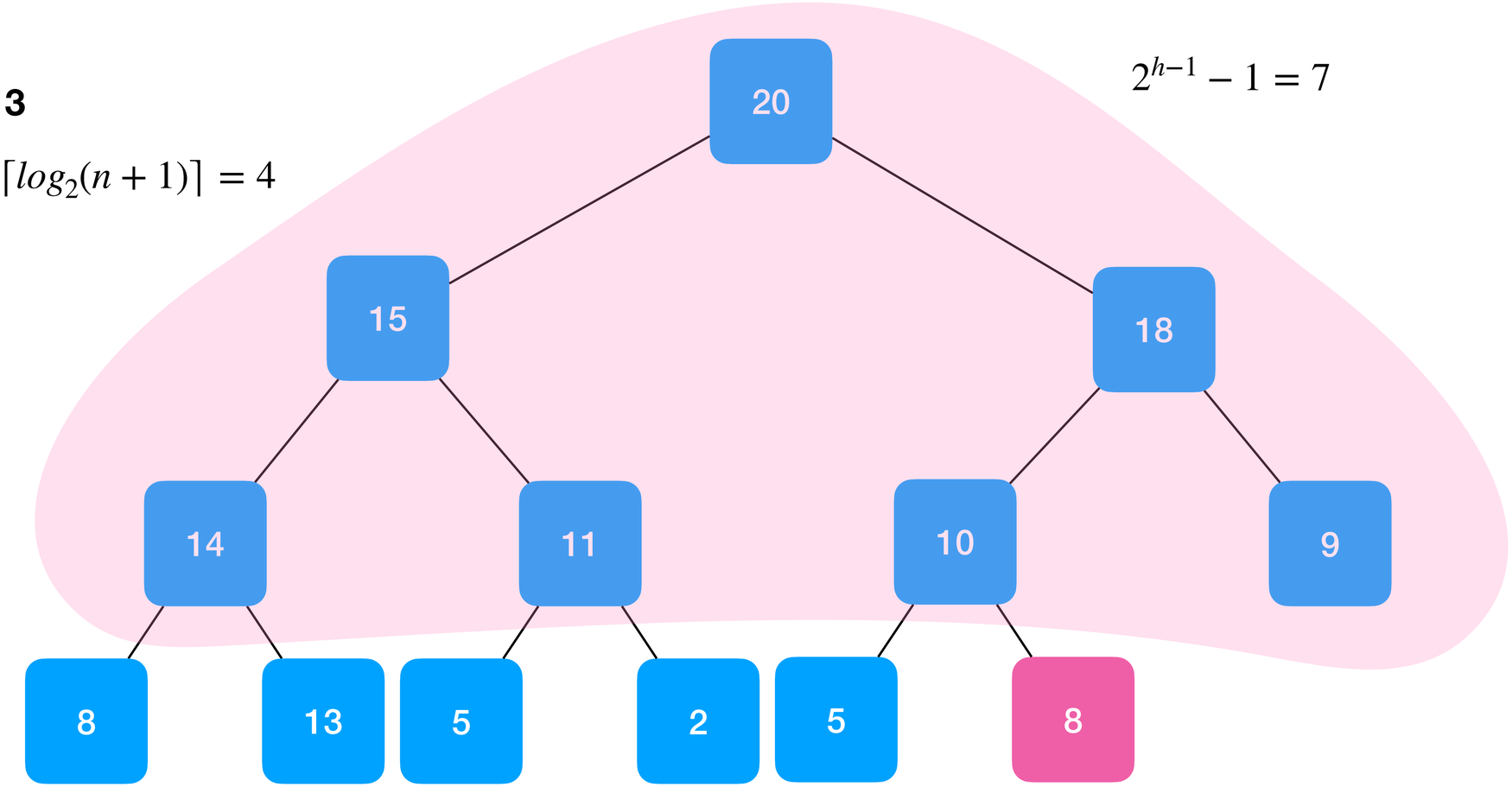
Imaginez une implémentation d'une file de priorité par une heap à l'aide d'une structure chaînée pour représenter l'arbre binaire essentiellement complet correspondant au tas.

- Combien de liens sont nécessaires dans chaque noeud ?
  - *Pour faire un swim il faut avoir accès aux parents et aux enfants (pour faire un exchange). Il faut donc trois liens.*
- Écrivez le code des méthodes insert, delMax. Complexité.
  - *Laissé en exercice (bonne préparation pour l'examen)*
- Est-il utile de donner la taille max N dans le constructeur ? Comment faites-vous pour ajouter un nouveau noeuds dans la heap ou retirer le prochain noeud ? Est-ce que cela peut être fait au départ de la taille courante de la heap ?

5.2.4 Solution1

n=13

$h = \lceil log_2(n+1) \rceil = 4$



$2^{h-1} - 1 = 7$

$index = n - (2^{h-1} - 1) - 1 = 5$

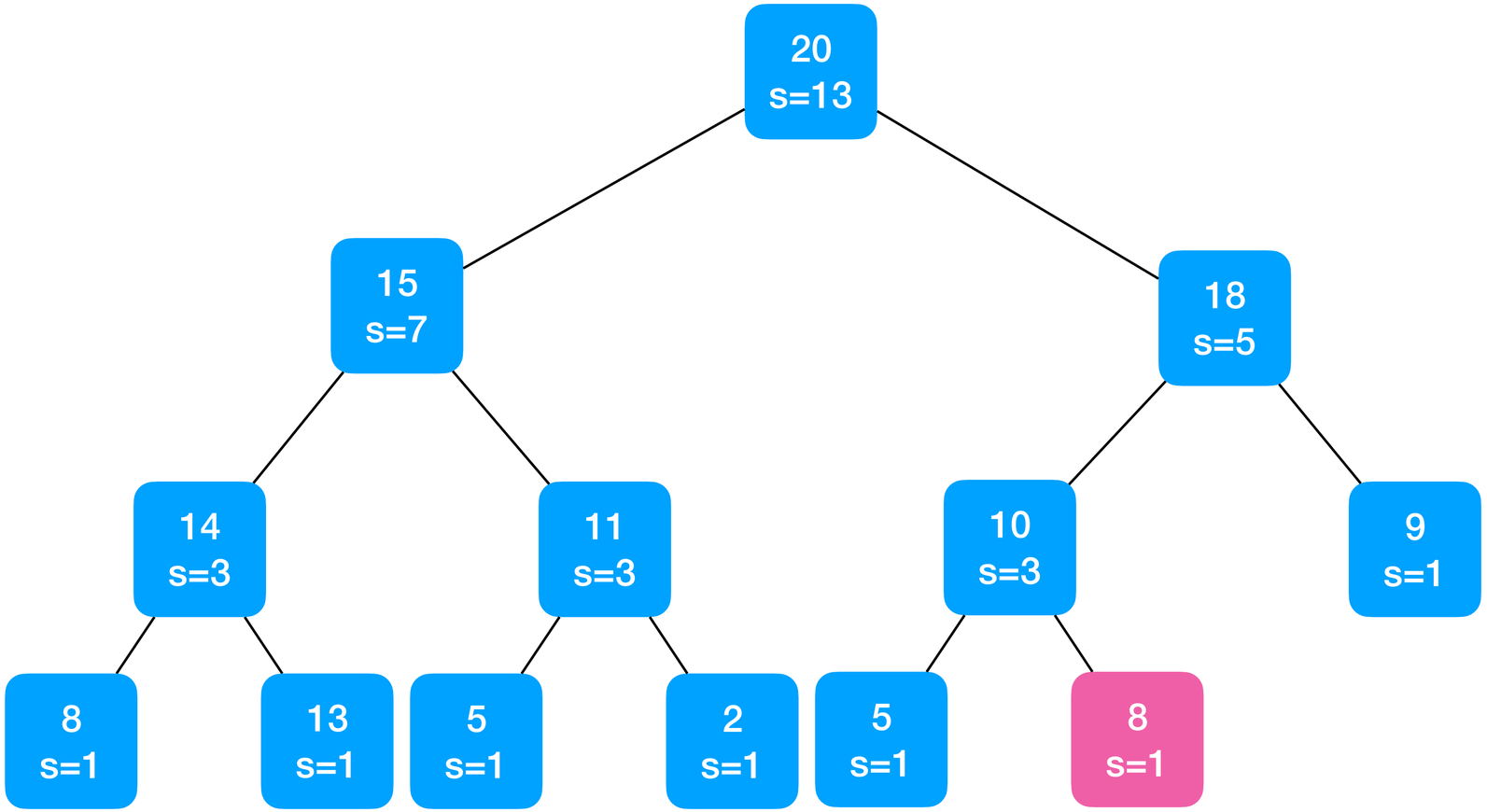
5=101=right->left->right

```java
private Key removeLastNodeInLastLayer() {
    // height of the heap = ceil (log_2 (n+1))
    int h = (int) Math.ceil((Math.log(size()+1) / Math.log(2)));
    // position of the last node on the last layer
    int index = size() - ((1 << (h-1)) - 1) - 1;
    Node current = root;
    current.size--;
    for (int i = h-2; i >= 0; i--) {
        // if i_th bit is 0, follow left otherwise follow right
        if ((1 << i & index) == 0) {
            current = current.left;
        } else {
            current = current.right;
        }
        current.size--;
    }
    Key k = current.value;
    if (current.parent.left == current) {
        current.parent.left = null;
    } else {
        current.parent.right = null;
    }
    return k;
}
```

5.2.4 Solution2

- Maintenir dans chaque noeud la taille du sous arbre.

- Pour trouver le dernier noeud = suivre le descendant avec l'arbre incomplet, si les deux sont complets avec le même nombre de noeuds aller à droite, sinon aller à gauche



```
private Key removeLastNodeInLastLayer() {
    assert (root!= null && root.left != null);
    Node current = root;
    if (size() == 1) {
        root = null;
        return current.value;
    }
    current.size--;
    boolean right = true;
    while (current.left != null) {
        boolean leftComplete = isPowerOfTwo(current.left.size+1);
        if (!leftComplete || current.right == null) { // left incomplete or right = null
            current = current.left;
        } else { // left complete & current.right != null
            boolean rightComplete = isPowerOfTwo(current.right.size+1);
            if (!rightComplete) {
                current = current.right;
            } else {
                // left and right are complete
                if (current.left.size > current.right.size) {
                    current = current.left;
                } else {
                    current = current.right;
                }
            }
        }
        current.size--;
    }
    if (current.parent.left == current) {
        current.parent.left = null;
    } else {
        current.parent.right = null;
    }

    return current.value;
}
```

5.2.5 Min-Max Heap

Proposez une structure de données qui supporterait les opérations suivantes en temps logarithmique:

- Insertion
- supprimer le maximum,
- supprimer le minimum;
- et les opérations suivantes en temps constant: trouver le maximum et le minimum.

Min-Max Heap

- Les niveaux pairs (**min**) sont: 0 (racine), 2, 4, etc.

- Les niveaux impairs (**max**) sont 1, 3, 5, etc.

Pour n'importe quel élément x dans la min-max heap on a la propriété suivante:

- Si x est à un niveau min, tous les descendants de x sont supérieurs à x.

- Si x est à un niveau max, tous les descendants de x sont inférieurs à x.

Questions:

- D'après cette propriété déterminez quel est le plus petit élément de la heap ?

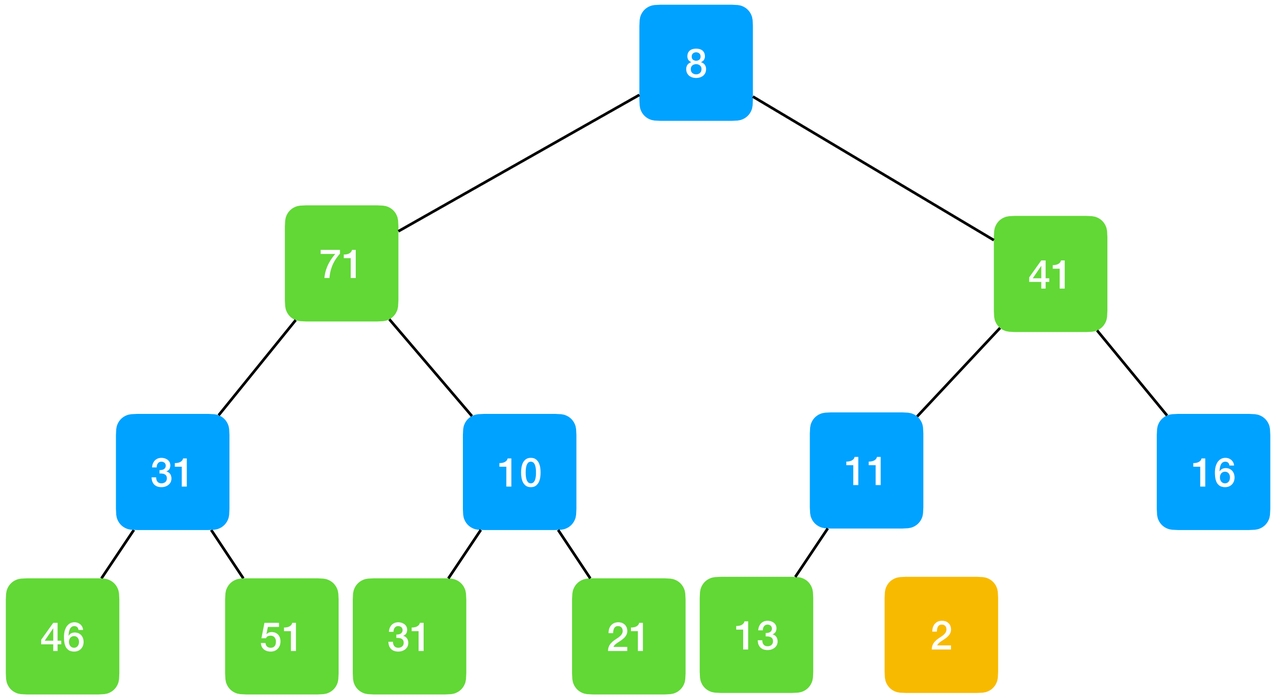
- Quel est le plus grand élément de la heap ?

- Dessinez une min-max heap qui contient les éléments suivants: 10,8,71,31,41,46,51,31,21,11,16,13.

- Décrivez l'opération d'insertion dans une min-max heap? Donnez le pseudo-code.

5.2.5. Min-Max Heap

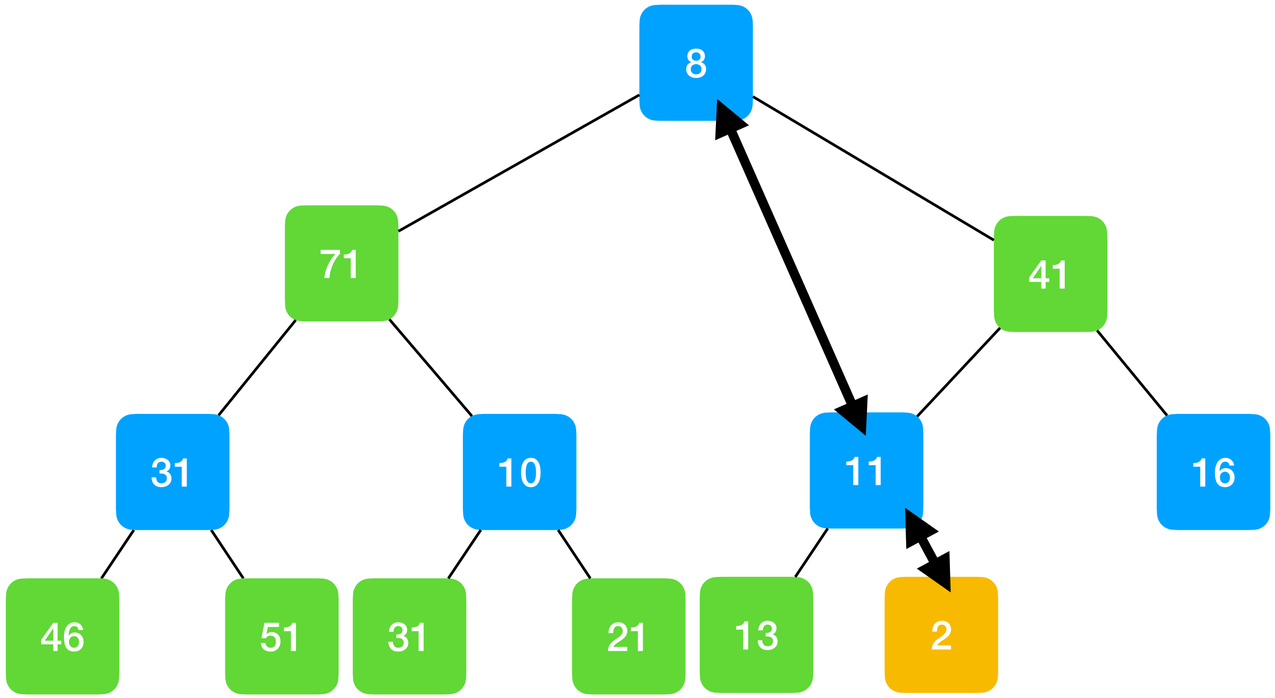
- D'après cette propriété déterminez quel est le plus petit élément de la heap ?
  - *?*
- Quel est le plus grand élément de la heap ?
  - *?*
- Dessinez une min-max heap qui contient les éléments suivants: 10,8,71,31,41,46,51,31,21,11,16,13.



- Décrivez l'opération d'insertion dans une min-max heap? Donnez le pseudo-code.
- Attention, il faut aller voir aussi au niveau i-2 pour voir s'il ne faut pas swapper.

5.2.5. Min-Max Heap

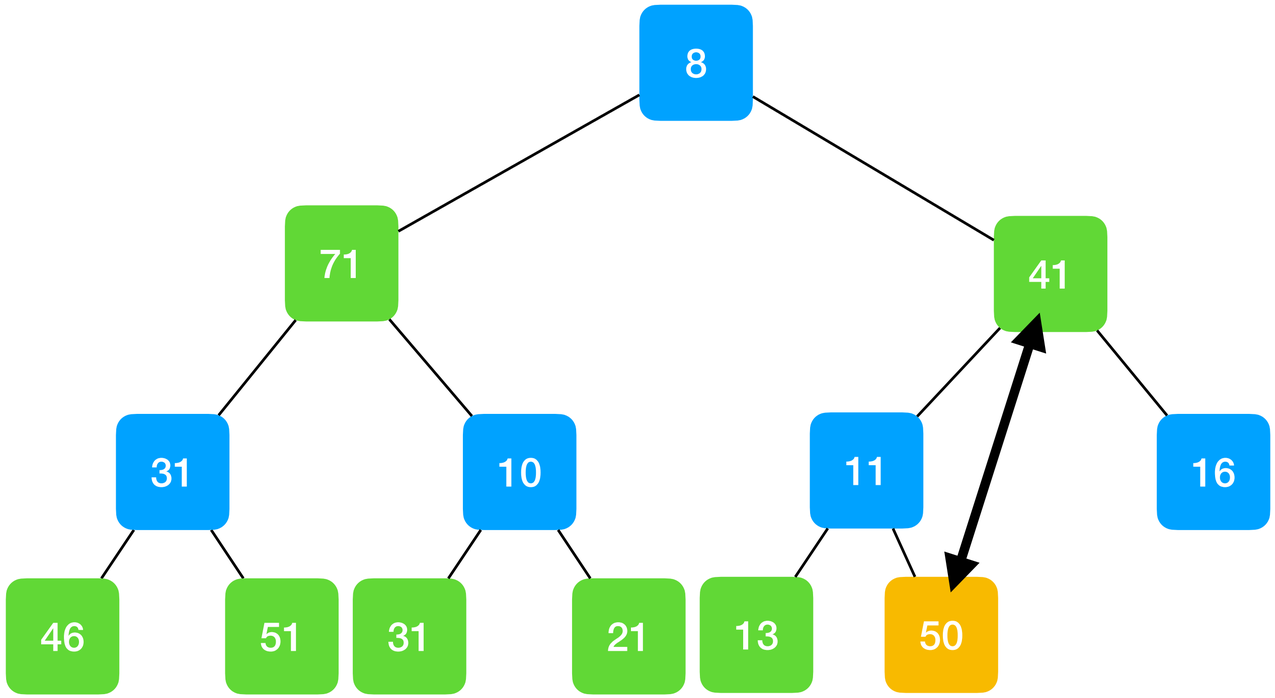
- D'après cette propriété déterminez quel est le plus petit élément de la heap ?
  - *L'élément racine (du niveau 0, min)*
- Quel est le plus grand élément de la heap ?
  - *Le max entre les deux éléments du niveau 1*
- Dessinez une min-max heap qui contient les éléments suivants: 10,8,71,31,41,46,51,31,21,11,16,13.



- Décrivez l'opération d'insertion dans une min-max heap? Donnez le pseudo-code.
- Attention, il faut aller voir aussi au niveau i-2 pour voir s'il ne faut pas swapper.

5.2.5. Min-Max Heap

- D'après cette propriété déterminez quel est le plus petit élément de la heap ?
  - *L'élément racine (du niveau 0, min)*
- Quel est le plus grand élément de la heap ?
  - *Le max entre les deux éléments du niveau 1*
- Dessinez une min-max heap qui contient les éléments suivants: 10,8,71,31,41,46,51,31,21,11,16,13.



- Décrivez l'opération d'insertion dans une min-max heap? Donnez le pseudo-code.
- Attention, il faut aller voir aussi au niveau i-2 pour voir s'il ne faut pas swapper.

Want to see the code ?
Exam January 2019

Min/Max = Easy

```
public class MinMaxHeap<Key extends Comparable<Key>> {

    public Key[] pq; // contains the elements starting at position 1, don't rename it or change the visibility
    private int N = 0; // number of elements in the heap
    int height = 0; // should help you to know if you are at a level min or max

    public MinMaxHeap(int maxN) {
        pq = (Key[]) new Comparable[maxN + 1];
    }
    public boolean isEmpty() {
        return N == 0;
    }
    public int size() {
        return N;
    }
    public Key min() {
        return pq[1];
    }
    public Key max() {
        if (N == 1) return min();
        else if (N == 2) return pq[2];
        else if (less(2, 3)) return pq[3];
        else return pq[2];
    }
```

O(1)

O(1)

.

And the insert ?

```
public class MinMaxHeap<Key extends Comparable<Key>> {

   // you should not add any additional instance variables (but you can if you want)
   public Key[] pq; // constains the elements starting at position 1, don't rename it or change the visibility
   private int N = 0; // number of elements in the heap
   int height = 0; // should help you to know if you are at a level min or max
   public MinMaxHeap(int maxN) {
      pq = (Key[]) new Comparable[maxN + 1];
   }
   public void insert(Key v) {
      pq[++N] = v;
      if (N >= (1 << height)) height++;
      swim(N);
   }
   private void swim(int k) {
      if (k > 1) {
         boolean minLayer = height % 2 == 1;
         if ((minLayer && less(k / 2, k)) || (!minLayer && less(k, k / 2))) {
            exch(k / 2, k);
            k = k / 2;
            minLayer = !minLayer;
         }
         while (k > 3 && ((minLayer && less(k, k / 4)) || (!minLayer && less(k / 4, k)))) {
            exch(k / 4, k);
            k = k / 4;
         }
      }
   }
   public boolean less(int i, int j) {
      return pq[i].compareTo(pq[j]) < 0;
   }
   private void exch(int i, int j) {
      Key e = pq[i];
      pq[i] = pq[j];
      pq[j] = e;
   }
}
```

Last layer full, increase height

First, exchange with direct parent if necessary

Then do standard swim with layers -2

5.2.6 Mediane

- Imaginez une structure de données qui supporte

  1. l'**insertion** en temps logarithmique
  2. l'opération **trouver la médiane** en temps constant
  3. **supprimer la médiane** en temps logarithmique.

Solution:

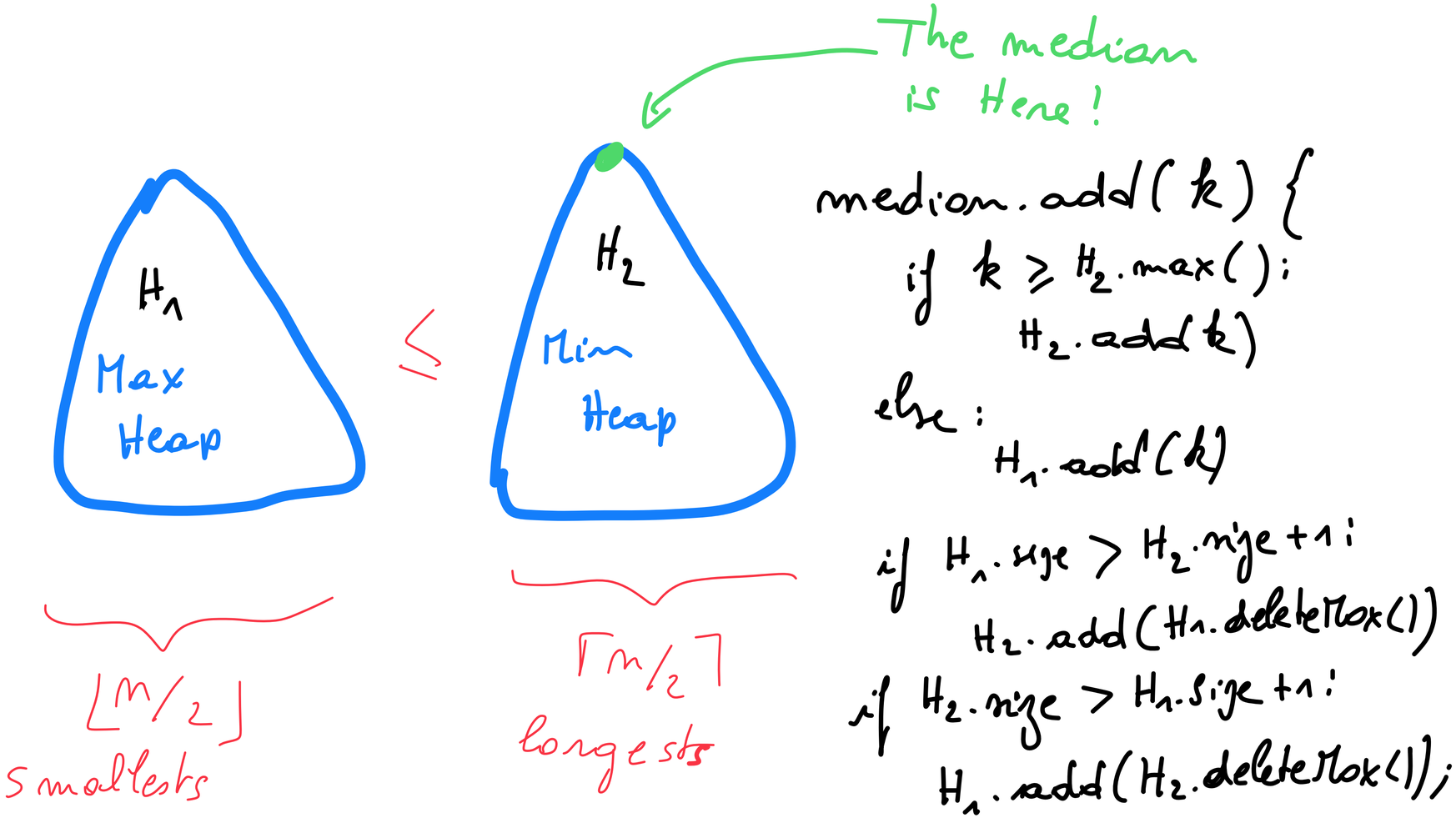
- *?*

5.2.6 Mediane

- Imaginez une structure de données qui supporte
  1. l'**insertion** en temps logarithmique
  2. l'opération **trouver la médiane** en temps constant
  3. **supprimer la médiane** en temps logarithmique.

Solution:

- *Il faut utiliser deux heap, chacune contenant la moitié des éléments.*
- *La première heap est une max-heap et contient les n/2 plus petits éléments.*
- *La deuxième heap est une min-heap et contient les n/2 plus grands éléments.*
- *Assez facile de maintenir cette propriété lors de l'insertion d'un élément et le retrait de la médiane.*




```
median.add(k) {
    if k ≥ H2.max():
        H2.add(k)
    else:
        H1.add(k)
    if H1.size > H2.size+1:
        H2.add(H1.deleteMax())
    if H2.size > H1.size+1:
        H1.add(H2.deleteMax());
}
```

LINFO 1121
DATA STRUCTURES AND ALGORITHMS

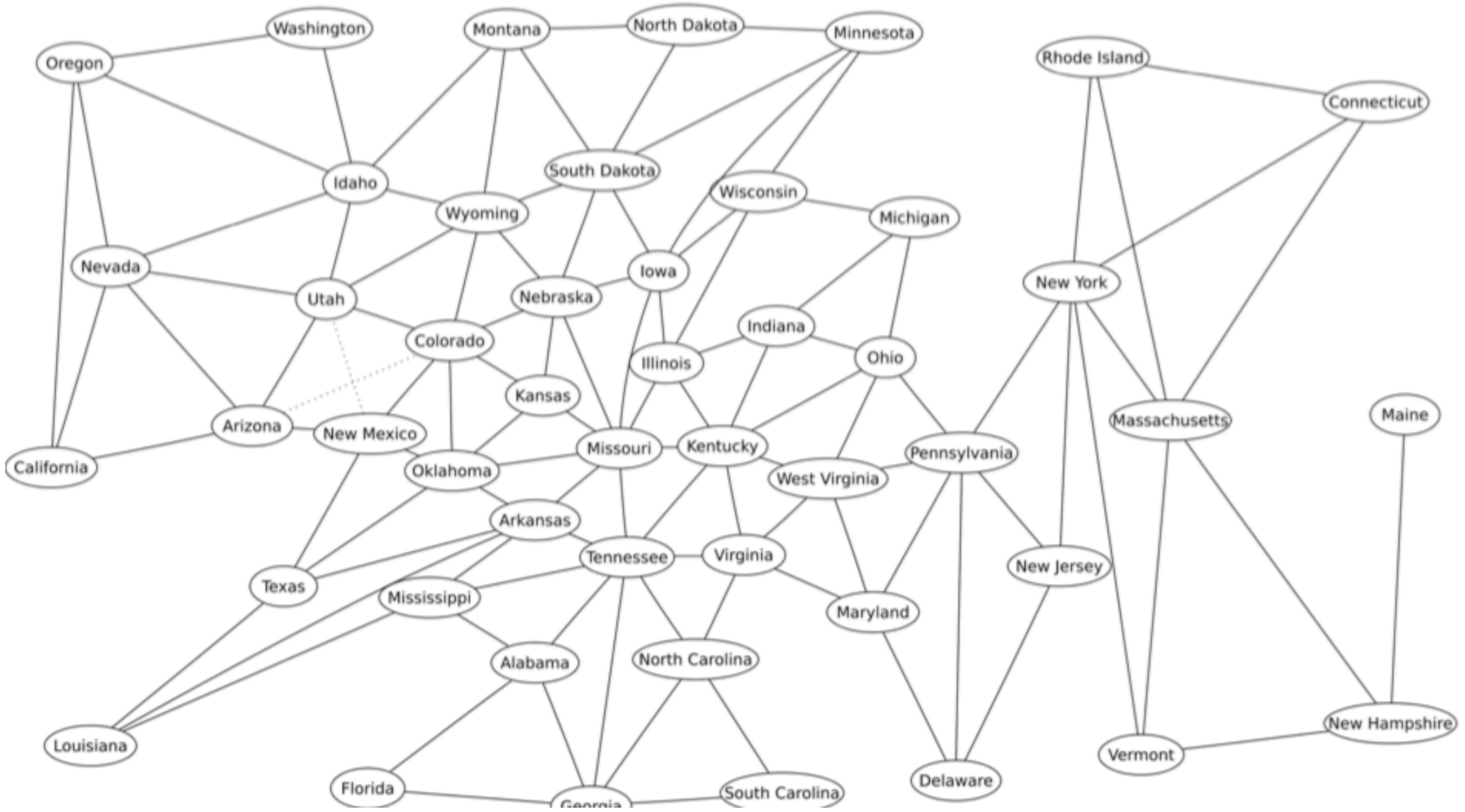
UCLouvain

Les Graphes

Pierre Schaus

Rappel: les tables de symboles

- Type abstrait de données permettant:
  - De representer des réseaux



Pourquoi étudier les graphes

- Des milliers d'applications

problem	description
s→t path	*Is there a path from s to t ?*
shortest s→t path	*What is the shortest path from s to t ?*
directed cycle	*Is there a directed cycle in the graph ?*
topological sort	*Can the digraph be drawn so that all edges point upwards?*
strong connectivity	*Is there a directed path between all pairs of vertices ?*
transitive closure	*For which vertices v and w is there a directed path from v to w ?*
PageRank	*What is the importance of a web page ?*

- Des centaines d'algorithmes (nous n'étudierons que les plus connus)

- C'est un branche complète de l'informatique (il y a un cours dédié à ce sujet LINMA1691)

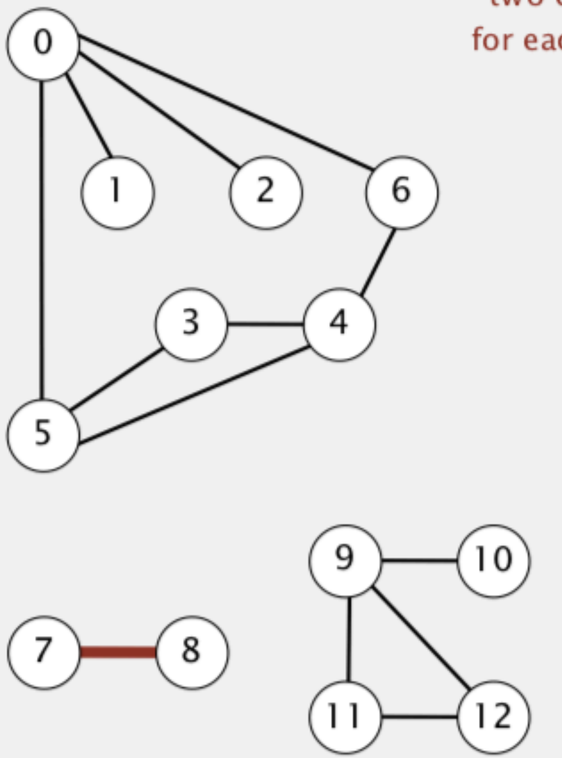
Graph API

public class Graph		
	Graph(int V)	*create an empty graph with V vertices*
	Graph(In in)	*create a graph from input stream*
void	addEdge(int v, int w)	*add an edge v-w*
Iterable<Integer>	adj(int v)	*vertices adjacent to v*
int	V()	*number of vertices*
int	E()	*number of edges*

Implem 1: Matrice d'incidence

Maintain a two-dimensional V-by-V boolean array;

for each edge v–w in graph: `adj[v][w] = adj[w][v] = true.`



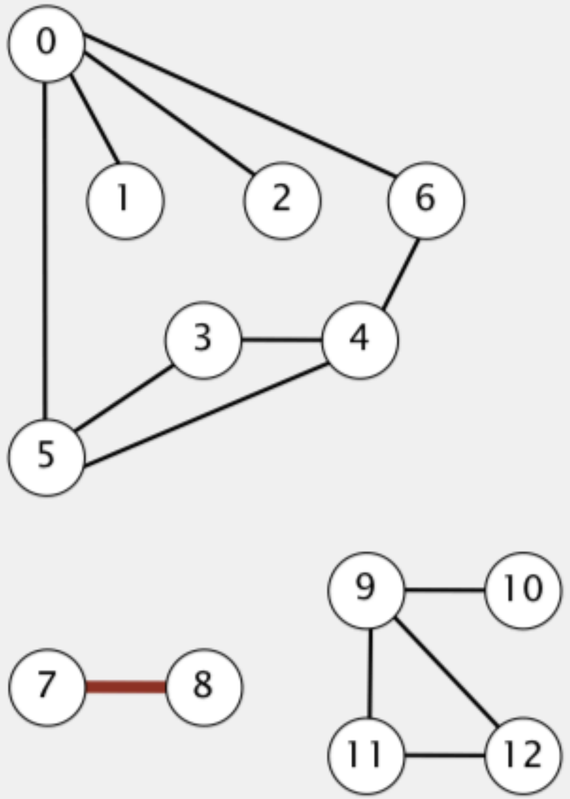
two entries for each edge

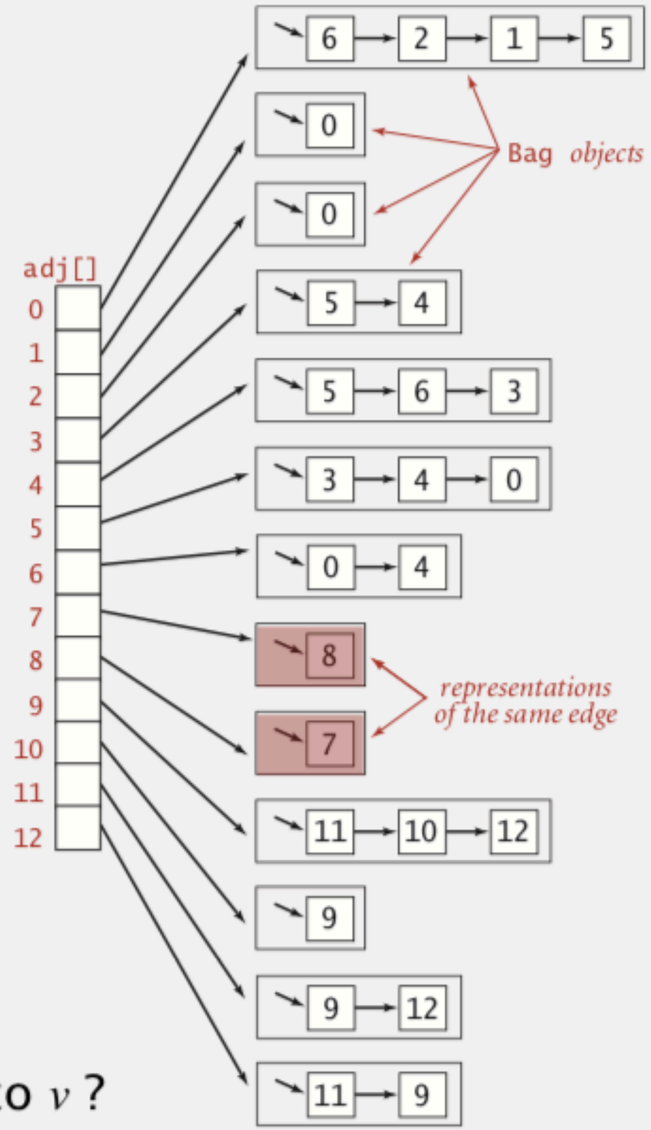
	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0	1	1	0	0	1	1	0	0	0	0	0	0
1	1	0	0	0	0	0	0	0	0	0	0	0	0
2	1	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	1	1	0	0	0	0	0	0	0
4	0	0	0	1	0	1	1	0	0	0	0	0	0
5	1	0	0	1	1	0	0	0	0	0	0	0	0
6	1	0	0	0	1	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	1	0	0	0	0
8	0	0	0	0	0	0	0	1	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0	1	1	1
10	0	0	0	0	0	0	0	0	0	1	0	0	0
11	0	0	0	0	0	0	0	0	0	1	0	0	1
12	0	0	0	0	0	0	0	0	0	1	0	1	0

Q. How long to iterate over vertices adjacent to v ?

Implem 2: Liste d'incidences

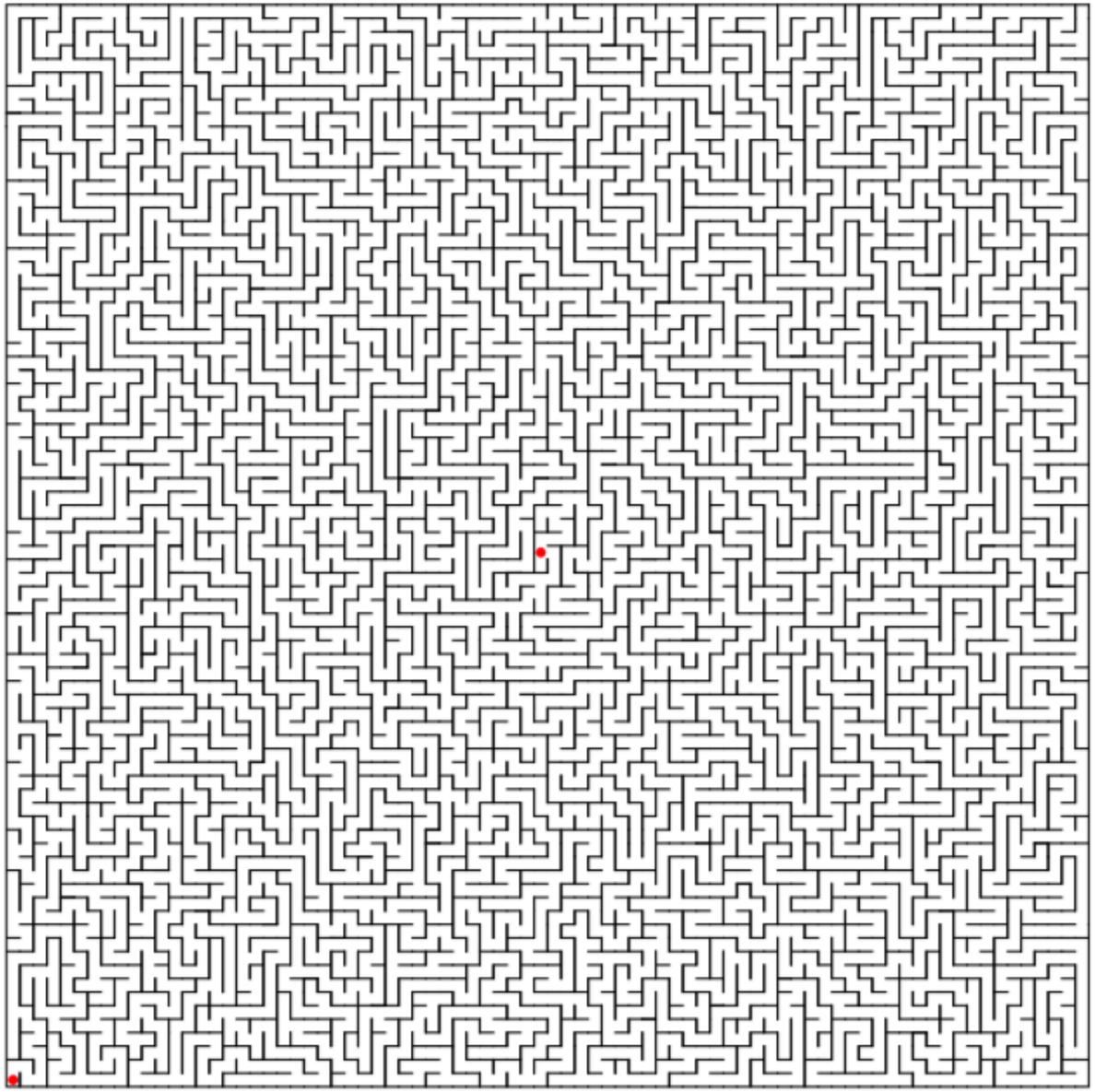
Maintain vertex-indexed array of lists.



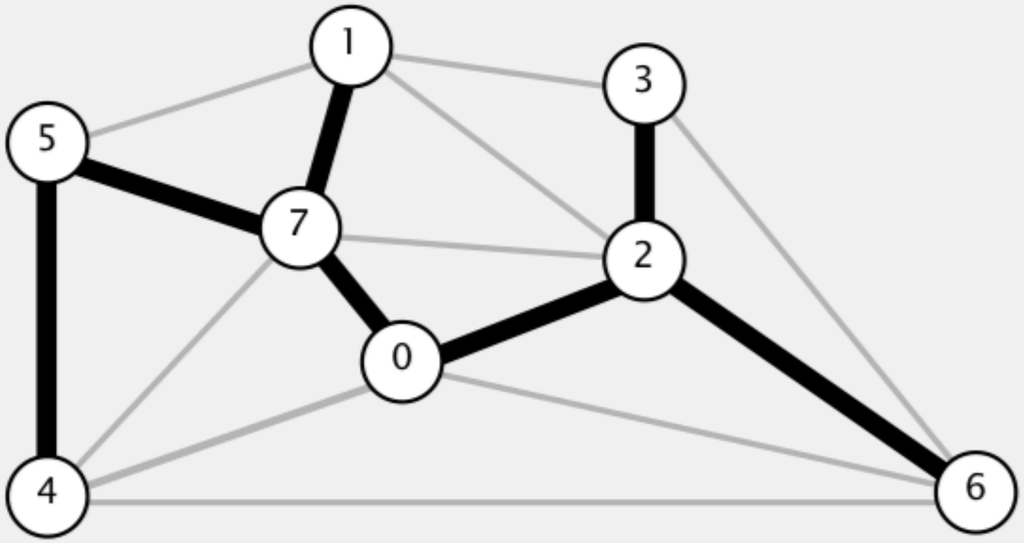


Q. How long to iterate over vertices adjacent to v ?

BFS : Application



Spanning Tree



a minimum spanning tree

0-7	0.16
2-3	0.17
1-7	0.19
0-2	0.26
5-7	0.28
1-3	0.29
1-5	0.32
2-7	0.34
4-5	0.35
1-2	0.36
4-7	0.37
0-4	0.38
6-2	0.40
3-6	0.52
6-0	0.58
6-4	0.93

DiGraph API (graph Dirigé)

public class Digraph		
	Digraph(int V)	*create an empty digraph with V vertices*
	Digraph(In in)	*create a digraph from input stream*
void	addEdge(int v, int w)	*add a directed edge v→w*
Iterable<Integer>	adj(int v)	*vertices pointing from v*
int	V()	*number of vertices*
int	E()	*number of edges*
Digraph	reverse()	*reverse of this digraph*
String	toString()	*string representation*

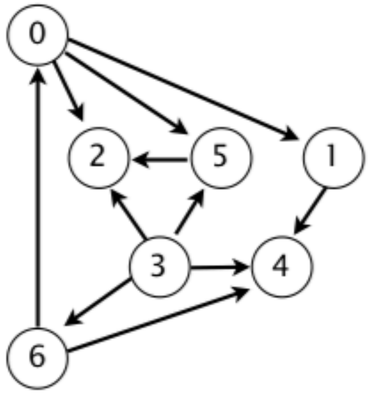
Topological Sort

DAG. Directed acyclic graph.

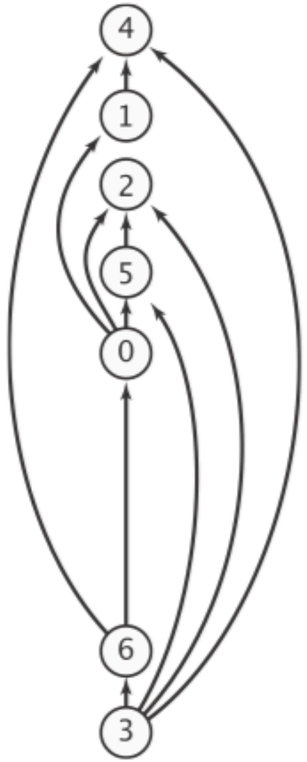
Topological sort. Redraw DAG so all edges point upwards.

0→5	0→2
0→1	3→6
3→5	3→4
5→2	6→4
6→0	3→2
1→4	

directed edges



DAG



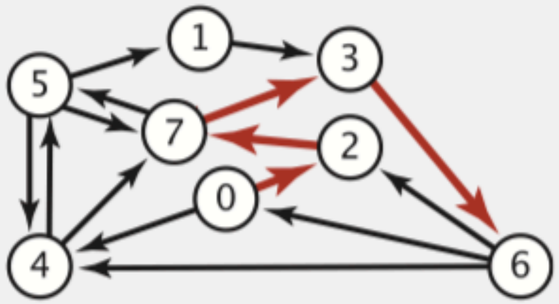
topological order

Solution. DFS. What else?

Exemple Plus court chemin

edge-weighted digraph

```
4->5  0.35
5->4  0.35
4->7  0.37
5->7  0.28
7->5  0.28
5->1  0.32
0->4  0.38
0->2  0.26
7->3  0.39
1->3  0.29
2->7  0.34
6->2  0.40
3->6  0.52
6->0  0.58
6->4  0.93
```



shortest path from 0 to 6

```
0->2  0.26
2->7  0.34
7->3  0.39
3->6  0.52
```