Nomenclature

Graphes non orientés

- Un graphe non orienté $G = (V,E)$ est formé
  - D'un ensemble V dont les éléments sont appelés sommets
  - D'un ensemble E dont les éléments sont appelés arêtes
  - D'une fonction d'incidence qui associe à chaque arête e une paire $\{ u(e), v(e) \}$ de sommets appelés extrémités de l'arête e

- Une arête e relie ses deux extrémités u et v et est incidente avec elles
- Deux sommets u et v sont dits adjacents s'il existe une arête e qui les relie. Ils sont incidents à l'arête e

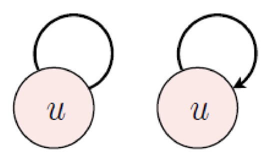
Graphes orientés

- Un graphe orienté $G = (V,E)$ est formé
  - D'un ensemble V dont les éléments sont appelés sommets
  - D'un ensemble E dont les éléments sont appelés arcs
  - D'une fonction d'incidence qui associe à chaque arête e une paire $(u(e), v(e))$ de sommets appelés extrémités de l'arête e

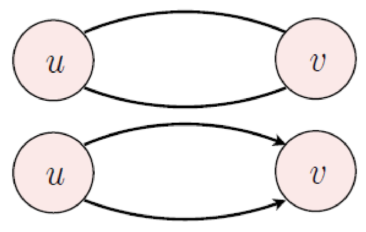
- On appelle u l'extrémité initiale et v l'extrémité finale de l'arc e
- On remplaçant tous les arcs de G par des arêtes de même extrémités, on obtient son graphe sous-jacent (non orienté)

Boucles, arêtes et arcs multiples

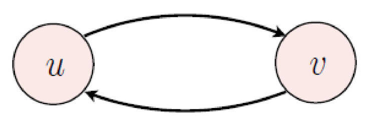
- Une boucle est une arête ou un arc dans les deux extrémités sont le même sommet



- Deux (arêtes / arcs) partageant les mêmes (extrémités / extrémités initiale et finale) sont dites multiples.



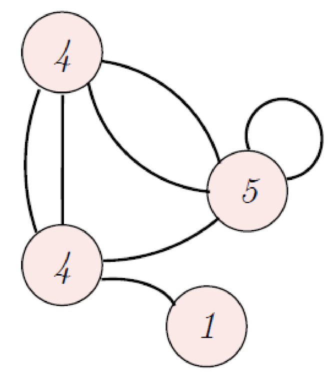
- Deux arcs ayant les mêmes extrémités mais inversées sont de sens opposés, pas des arcs multiples



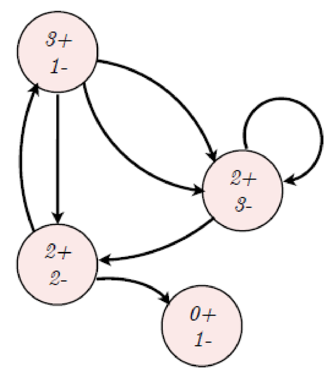
- Un graphe simple n'a ni boucle ni arête/arc multiple. Un multigraphe en a.

Degrés, demi-degrés

- Le degré $deg(v)$ d'un sommet v est le nombre d'arêtes ou d'arcs incidents à v, les boucles étant comptées à double.
  - Notons que l'on a $\sum_v deg(v) = 2 \cdot |E|$
  - Un sommet de degré 1 est dit pendant



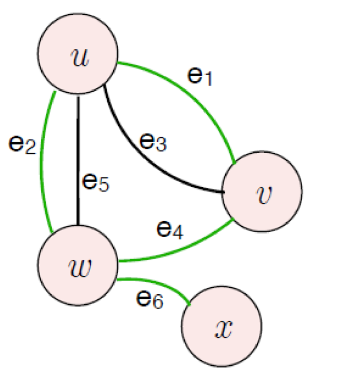
- Pour un graphe orienté, on peut préciser
  - Le demi-degré sortant $deg_+(v)$ est le nombre d'arcs dont v est l'extrémité initiale
  - Le demi-degré entrant $deg_-(v)$ est le nombre d'arcs dont v est l'extrémité finale
  - On a les relations suivantes : $deg(v) = deg_+(v) + deg_-(v)$
    $\sum_v deg_+(v) = \sum_v deg_-(v) = |E|$



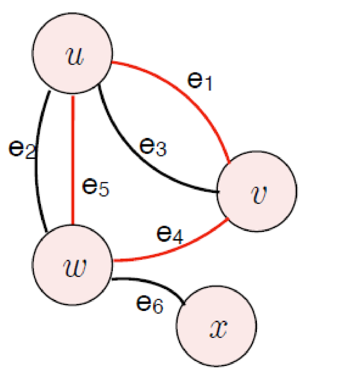
Chaîne et cycle

- Dans un graphe non-orienté, une chaine est une suite alternée de sommets et d'arêtes
  - Commençant et terminant par un sommet
  - Où chaque sommet est incident aux arêtes qui l'entourent et vice-versa
- La longueur d'une chaine est son nombre d'arêtes (répétitions comprises)
- Un cycle est un chaîne fermée de longueur non nulle commençant et se terminant au même sommet.
- Une chaine / un cycle est élémentaire si aucun sommet n'y est répété
- Une chaine / un cycle est simple si aucune arête n'y est répétée

Chaine x-e_6-w-e_4-v-e_1-u-e_2-w



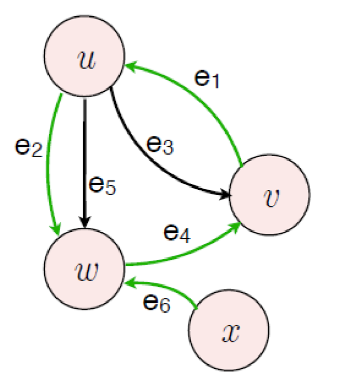
Cycle w-e_4-v-e_1-u-e_5-w



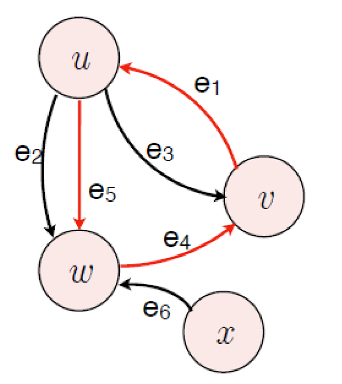
Chemin et circuit

- Dans un graphe orienté, un chemin est une suite alternée de sommets et d'arcs
  - Commençant et terminant par un sommet
  - Où chaque arc est précédé de son extrémité initial, suivi de son extrémité finale
- Un circuit est un chemin fermé de longueur non nulle commençant et se terminant au même sommet.

Chemin x-e_6-w-e_4-v-e_1-u-e_2-w



Circuit w-e_4-v-e_1-u-e_5-w

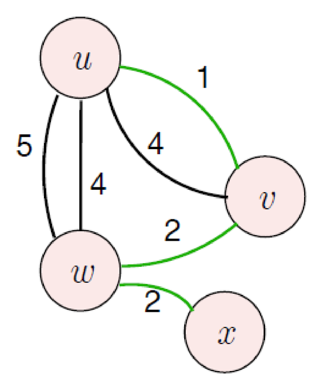


Arbre et forêt

- Un graphe est appelé acyclique s'il n'a pas de cycle simple
- On parle de forêt s'il est en plusieurs morceaux disjoints, d'arbre s'il est d'un seul tenant
- Pour les graphes orientés, on parle d'arborescence

Graphe pondéré

- Dans de nombreux problèmes, il est pertinent d'associer un poids $c(e)$ à chaque arête / arc e du graphe.
- On s'intéressera à des problèmes tels que
  - Trouver la chaîne / le chemin de poids minimum entre 2 sommets
  - Déterminer s'il existe des cycles / circuits de poids négatif
  - Trouver l'arbre couvrant de poids minimum

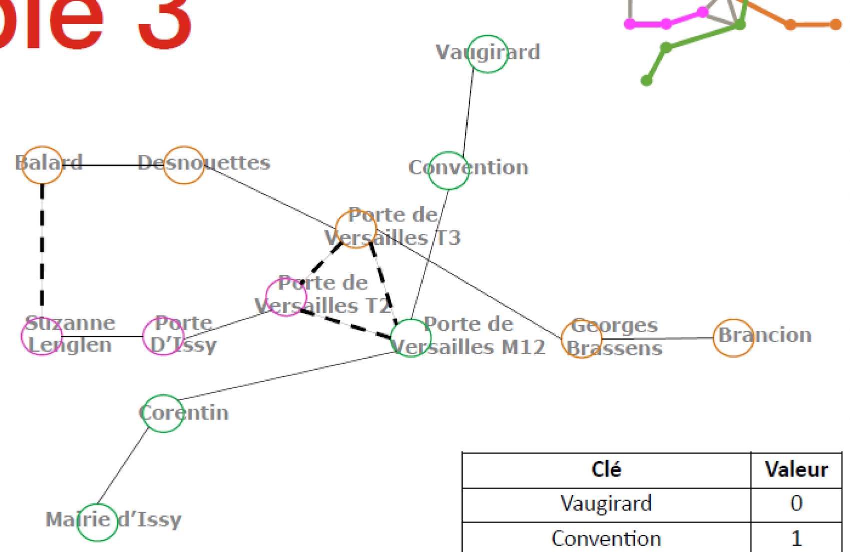


En vert, la chaîne de poids minimum reliant u et x.

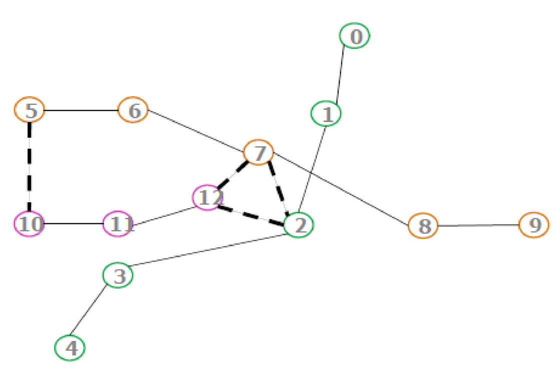
C'est aussi l'arbre couvrant de poids minimum.

Exemple 3

- Pour le métro, on peut par exemple assigner un indice à chaque station de métro, et les retrouver via une table de symboles



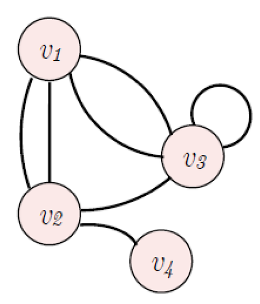

```
std::vector<string> noms;
noms[0] = "Vaugirard";
noms[1] = "Convention";
...

std::map<string, int> indices;
for(size_t i = 0; i<noms.size(); ++i) {
    indices[noms[i]] = i;
}
```



Clé	Valeur
Vaugirard	0
Convention	1
Porte de Versailles M12	2
Corentin Celton	3
Mairie d'Issy	4
Balard	5
Desnouettes	6
Porte de Versailles T3	7
Georges Brassens	8
Brancion	9
Suzanne Lenglen	10
Porte d'Issy	11
Porte de Versailles T2	12

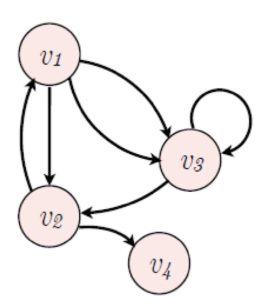
Matrice d'adjacence

- Pour un graphe $G = (V, E)$ à $n = |V|$ sommets, la matrice d'adjacence est une matrice $A : n \times n$ éléments
- Pour un graphe non-orienté, a_{ij} stocke le nombre d'arêtes $\{v_i, v_j\}$.
  - La matrice est symétrique.
  - Pour un graphe simple, une matrice de booléens suffit



$$A = \begin{bmatrix} 0 & 2 & 2 & 0 \\ 2 & 0 & 1 & 1 \\ 2 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

- Pour un graphe orienté, a_{ij} stocke le nombre d'arc (v_i, v_j), i.e. d'extrémité initiale v_i et finale v_j
  - Tous les sommets atteignables depuis v_i sont notés dans la ligne i
  - Elle n'est pas forcément symétrique



$$A = \begin{bmatrix} 0 & 1 & 2 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

```
Précondition: les sommets sont
 marqués comme non visités
                DFS
fonction profondeur (sommet v,
      Fn pre, (opt.) Fn post)

  pre(v) // en pré-ordre

  marquer v visité
  pour tout w adjacent à v
    si w n'est pas visité
      profondeur(w, pre, post)

  post(v) // en post-ordre
```

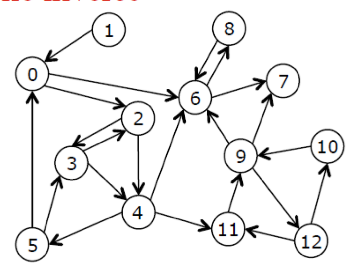
```
fonction largeur (sommet v,
               BFS    Fn action)

  Initialiser une file Q
  Q.push(v)
  marquer v

  tant que Q n'est pas vide
    v ← Q.pop()
    action(v)
    pour tout w adjacent à v
      si w n'est pas marqué
        Q.push(w)
        marquer w
```

Algorithme de Kosaraju-Sharir

- Calculer le post ordre inverse du parcours en profondeur pour le graphe inverse de G

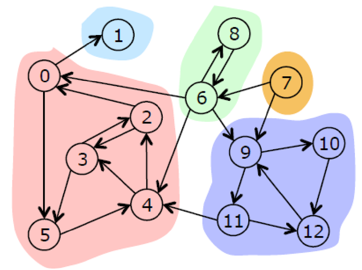
Pre Ordre:	0 2 3 4 5 11 9 6 7 8 12 10 1
Post Ordre:	5 7 8 6 10 12 9 11 4 3 2 0 1
Post Inverse:	1 0 2 3 4 11 9 12 10 6 8 7 5

Graphe inverse



- Calculer les composantes connexes par parcours en profondeur DFS sur G dans cet ordre

DFS(1) -> { 1 }
DFS(0) -> { 0, 5, 4, 2, 3 }
DFS(11) -> { 11, 12, 9, 10}
DFS(6) -> { 6, 8 }
DFS(7) -> { 7 }



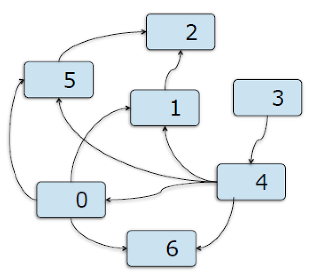
1. On inverse le graphe (sens des flèches inversé)
2. On fait le post-ordre sur celui-ci
3. On inverse le post-ordre
4. CC(x) = groupes des composantes fortement connexes dans l'ordre de ce dernier parcours, chaque groupe dans le pré-ordre du graphe normal

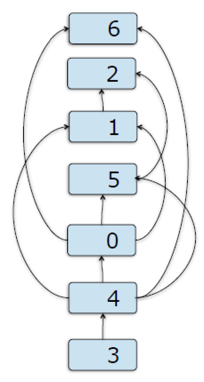
Tri topologique = Inverse du post-ordre

- Tri topologique - redessiner un graphe orienté acyclique (Directed Acyclic Graph ou DAG) pour que tous ses arcs pointent dans la même direction
- Ce n'est évidemment pas possible pour un graphe comprenant un circuit

Listes d'adjacence

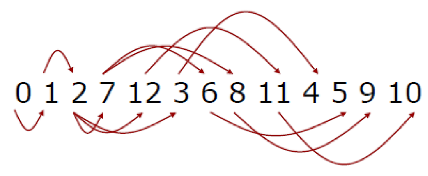
0 → 1,5,6
1 → 2
2 → -
3 → 4
4 → 0,1,5,6
5 → 2
6 → -

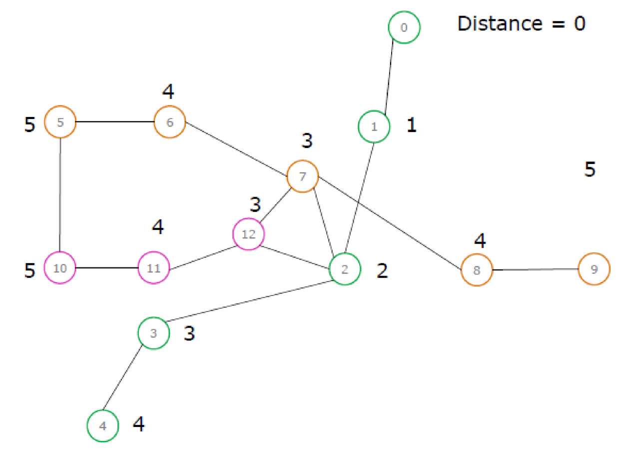




Distances et plus courtes chaines

- BFS parcourt les sommets par ordre de distance croissante au sommet de départ
- A tout moment, la file ne contient que des sommets à distance k ou $k+1$
- Le parcours en largeur calcule les chaines les plus courtes (nombre d'arêtes) à partir du sommet de départ





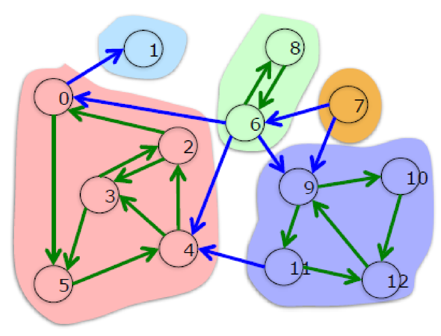
Chaine la plus courte entre v et w

- Recherche de la chaîne la plus courte entre v et w
- En remontant de parent en parent
- Jusqu'à l'origine du parcours en largeur, que l'on reconnait via `P(w)==w`

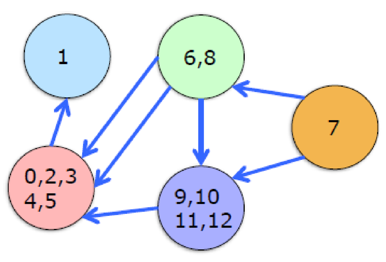
```
fonction chaine (Parents P, sommet w)

  Initialiser chaine vide

  si P(w) == -1
    retourner chaine vide

  sinon,
    tant que P(w) != w
      ajouter w à la chaine
      w ← P(w)

    retourner chaine
```


Comment calculer les CFC ?

- On distingue deux types d'arcs.
  - ceux qui restent dans la même CFC. Ils forment des cycles au sein de chaque CFC.
  - ceux qui mènent d'une CFC à une autre.

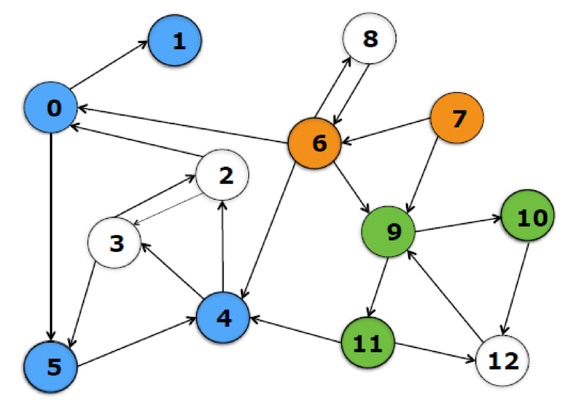


- En regroupant tous les sommets d'une même CFC et en ne gardant donc que les arcs bleus, on obtient un graphe acyclique



Parents avec sources multiples

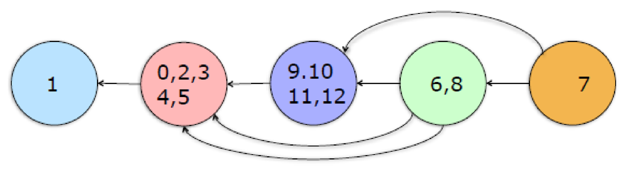
- Dans le graphe ci-dessous …
  - Quel est le sommet le plus proche de 2 parmi 0,7 et 9 ?
  - Quel est le chemin le plus court de ce sommet au sommet 2 ?



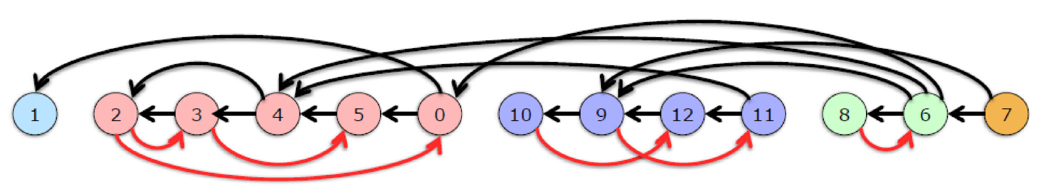
```
fonction parents_en_largeur (Sommets S)

  Initialiser tableau Parents à -1
  Initialiser file Q vide

  pour tout sommet v de S
    Q.push(v)
    Parents(v) ← v
  tant que Q n'est pas vide
    v ← Q.pop()
    pour tout w adjacent à v
      si Parents(w) == -1
        Q.push(w)
        Parents(w) ← v

  retourner Parents
```


Comment calculer les CFC ? (2)

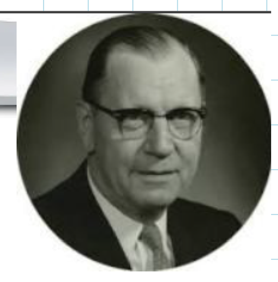
- Ce graphe étant un DAG, on peut le trier topologiquement (de droite à gauche)



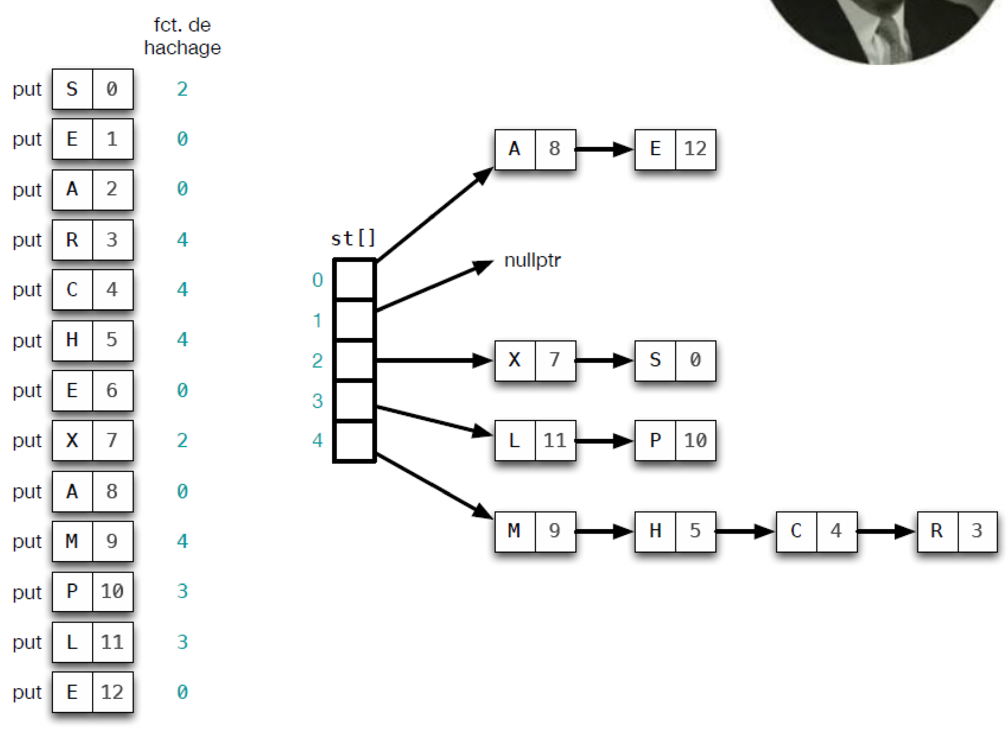
- En dégroupant les sommets, on obtiendrait un graphe presque trié, dont les seuls arcs mal orientés sont à l'intérieur des CFC



Résolution des collisions par chaînage

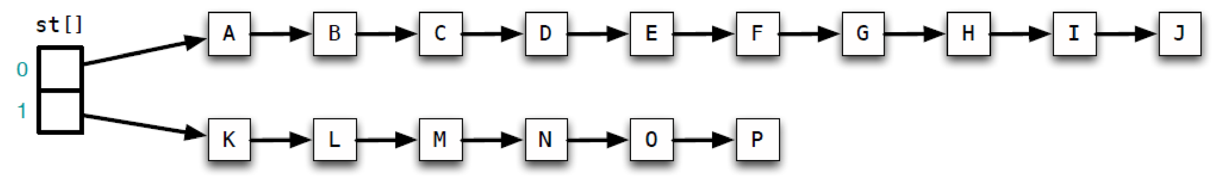


- Inventé par H.P. Luhn, IBM 1953
- Utilise un tableau de M listes simplement chaînées pour stocker N paires clé/valeur, avec $M < N$
- Hachage : transforme la clé k en un entier $0 \leq h(k) < M$
- Insertion : Insère l'élément au début de la liste d'index $h(k)$, s'il n'y est pas déjà présent.
- Recherche: Parcourt la liste d'indice $h(k)$ uniquement
- $\alpha = N/M$ est le taux d'occupation. Ici $\alpha = 2$

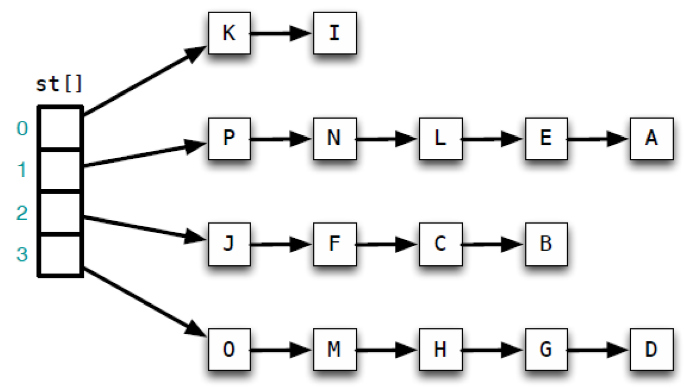


Redimensionner la table de hachage

Avant redimensionnement



Après redimensionnement



- Pour garder une longueur moyenne de liste N/M relativement constante, on va
  - doubler M quand $\dfrac{N}{M} \geq 8$
  - diviser M par deux quand $\dfrac{N}{M} \leq 2$

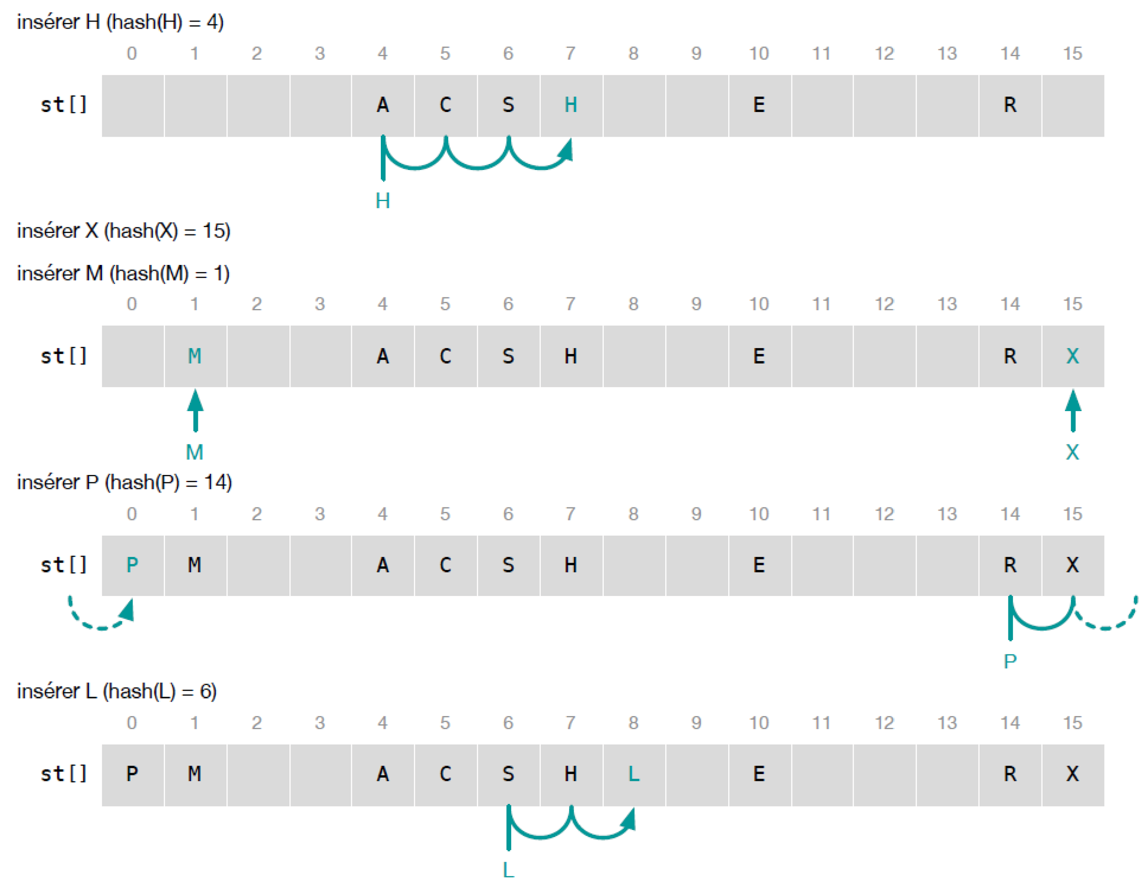
Division réelle

- Attention, il faut re-hacher toutes les clés quand on change M

Dans l'ordre dans lequel ils sont rangés et on hash d'abord avant d'insérer le nouvel élément.

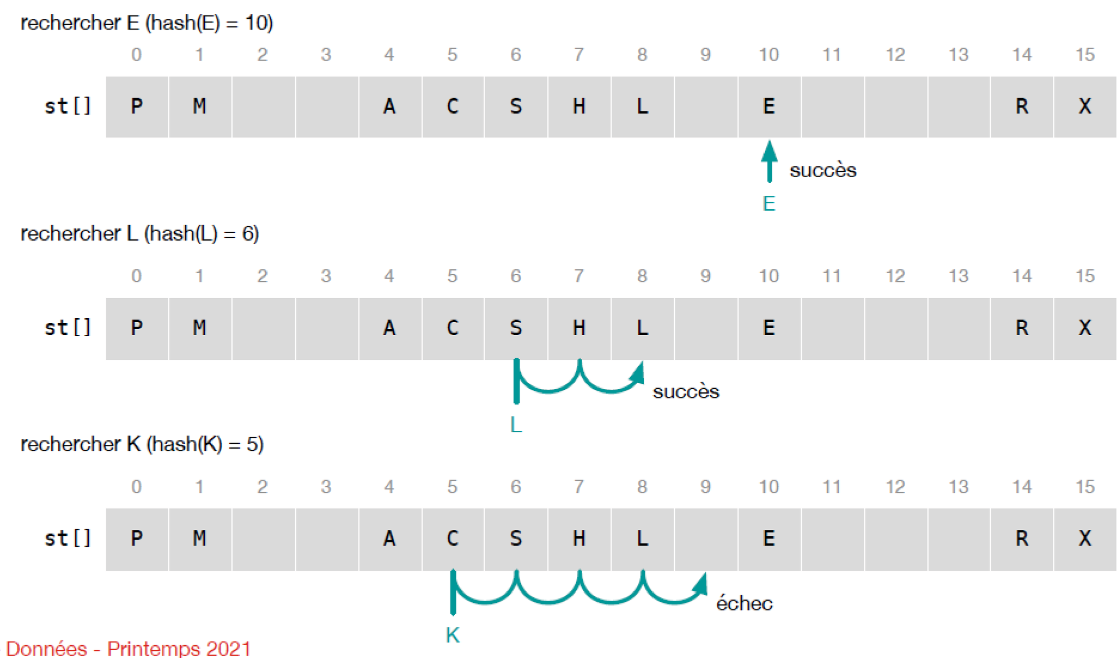
Sondage linéaire

Insérer une clé k : si l'emplacement à l'indice $h(k)$ est occupé, essayer $(h(k)+1) \mod M$, $(h(k)+2) \mod M$, … jusqu'à trouver un emplacement vide



Sondage linéaire

- Chercher la clé k : si l'emplacement à l'indice $h(k)$ est occupé mais ne correspond pas à k, essayer $(h(k)+1) \mod M$, $(h(k)+2) \mod M$, … jusqu'à trouver soit k soit un emplacement vide



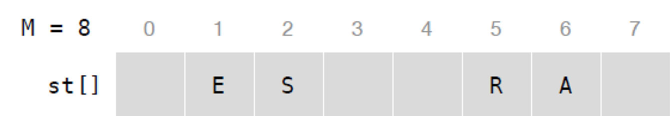
Redimensionner une table de hachage avec sondage linéaire

Pour garder le taux d'occupation $N/M \leq 1/2$

- Doubler la taille du tableau quand $N/M \geq 1/2$
- Réduire le tableau à la moitié quand $N/M \leq 1/8$
- On doit re-hacher toutes les clés quand on change la taille

S'il y a redimensionnement, comme pour le chaînage, on rehash d'abord les éléments déjà présents avant d'insérer le nouveau

Avant redimensionnement



Après redimensionnement

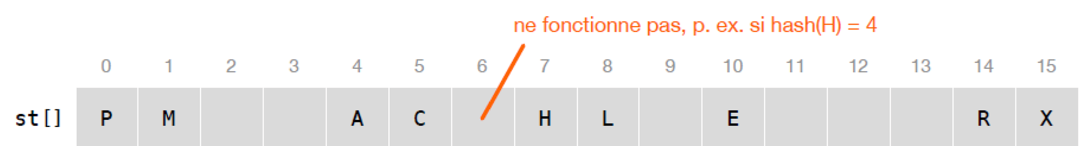
Supprimer un élément

- On ne peut pas simplement enlever un élément du tableau
- Il faut ré-insérer tous les éléments qui suivent jusqu'au premier emplacement vide

Avant suppression de S

Donc si l'on supprime un élément, tous ceux qui le suivaient doivent être rehashés, jusqu'à rencontrer un trou

Après suppression de S ?



Pour les 2 types de résolution, VERIFIER LA TAILLE A CHAQUE FOIS que l'on insère/supprime !