ALGORITHMIQUE, chap. 3

Structures de données de base

En informatique, il existe plusieurs manières de représenter la notion mathématique d'ensemble. Il n'existe pas une représentation qui soit meilleure que les autres dans l'absolu : pour un problème donné, la meilleure représentation sera celle qui permettra de concevoir le meilleur algorithme.

Chaque élément de ces ensembles pourra comporter plusieurs champs qui peuvent être examinés dès lors que l'on possède un pointeur sur cet élément. Certains ensembles dynamiques supposent que l'un des champs de l'objet contient une **clé** servant d'identifiant.

Ces ensembles supportent potentiellement tout une série d'opérations :

- Recherche(S, k) : étant donné un ensemble S et une clé k, le résultat de cette requête est un pointeur sur un élément de S de clé k, s'il en existe un, et la valeur Nil sinon.
- Insertion(S, x) : ajoute à l'ensemble S l'élément pointé par x.
- Suppression(S, x) : supprime de l'ensemble S son élément pointé par x.

Si on a un ordre total, d'autres opérations sont possibles :

- Minimum(S) : renvoie l'élément de S de clé minimale.
- Maximum(S) : renvoie l'élément de S de clé maximale.
- Successeur(S, x) : renvoie, si celui-ci existe, l'élément de S immédiatement plus grand que l'élément de S pointé par x, et Nil dans le cas contraire.
- Prédécesseur(S, x) : renvoie, si celui-ci existe, l'élément de S immédiatement plus petit que l'élément de S pointé par x, et Nil dans le cas contraire.

Piles.

Définition :

Une pile est une structure de données mettant en oeuvre le principe " dernier entré, premier sorti " (*LIFO : Last*-*In*, *First*-*Out* en anglais).

L'élément ôté de l'ensemble par l'opération Suppression est spécifié à l'avance.

L'opération Insertion dans une pile est communément appelée Empiler, et l'opération Suppression, Dépiler.

Il est facile d'implémenter une pile au moyen d'un tableau.

La seule difficulté dans cette implémentation est la gestion des débordements de pile qui interviennent quand on tente d'effecteur l'opération Dépiler sur une pile vide et l'opération Empiler sur un tableau codant la pile qui est déjà plein.

Ce dernier problème n'apparaît pas lorsque l'on implémente les piles au moyen d'une structure de données dont la taille n'est pas fixée *a priori* (comme une liste chaînée).

Exercice :

Coder une procédure Pile-vide

Coder une procédure Empiler

Coder une procédure Dépiler

Files.

Définition :

Une file est une structure de données mettant en oeuvre le principe " premier entré, premier sorti " (*FIFO : First-In*, *First-Out* en anglais).

L'élément ôté de l'ensemble par l'opération Suppression est spécifié à l'avance : l'élément supprimé est celui qui est resté le plus longtemps dans la file.

Une file se comporte exactement comme une file d'attente de la vie courante.

On peut implémenter les files au moyen de tableaux. Par exemple pour les files à $n-1$ éléments, au moyen d'un tableau à n éléments et de deux attributs :

- tête(F) qui indexe (ou pointe) vers la tête de la file;
- queue(F) qui indexe le prochain emplacement où sera inséré un élément nouveau.

Quand tête(F) = queue(F), la liste est vide.

La seule difficulté dans cette implémentation est la gestion des débordements de file qui interviennent quand on tente d'effectuer l'opération Suppression sur une pile vide et l'opération Insertion sur un tableau codant la file qui est déjà plein.

Ce dernier problème n'apparaît pas lorsque l'on implémente les files au moyen d'une structure de donnée dont la taille n'est pas fixée *a priori* (comme une liste doublement chaînée).

Exercice :

Coder une procédure File-vide

Coder une procédure Insertion

Coder une procédure Suppression

Listes chainées.

Définition :

Une liste chaînée est une structure de données dans laquelle les objets sont arrangés linéairement, l’ordre linéaire étant déterminé par des pointeurs sur les éléments.

Chaque élément de la liste, outre le champ clé, contient un champ successeur qui est pointeur sur l’élément suivant dans la liste chaînée. Si le champ successeur d’un élément vaut Nil, cet élément n’a pas de successeur et est donc le dernier élément ou la queue de la liste. Le premier élément de la liste est appelé la tête de la liste. Une liste L est manipulée via un pointeur vers son premier élément, que l’on notera Tête(L). Si Tête(L) vaut Nil, la liste est vide.

Une liste chaînée peut prendre plusieurs formes :

- **Liste doublement chaînée** : en plus du champ successeur, chaque élément contient un champ prédécesseur qui est un pointeur sur l'élément précédant dans la liste. Si le champ prédécesseur d'un élément vaut Nil, cet élément n'a pas de prédécesseur et est donc le premir élément ou la **tête** de la liste.
- **Triée** ou **non triée** : suivant que l'ordre linéaire des éléments dans la liste correspond ou non à l'ordre linéaire des clés de ces éléments.
- **Circulaire** : si le champ précécesseur de la tête de la liste pointe sur la queue, et si le champ successeur de la queue pointe sur la tête. La liste est alors vue comme un anneau.

Exercices :

• Coder une procédure Recherche(L,k) pour le cas simple et double chaînage.

• Coder une procédure Insertion(x,L) pour le cas simple et double chaînage.
x est inséré en tête de la liste L.

• Coder une procédure Suppression(L,x) pour le cas simple et double chaînage.
On suppose que l'on a un pointeur sur x.
Attention, le cas du simple chaînage est plus difficile !

Exercices :

Implémenter une file à l’aide de deux piles.

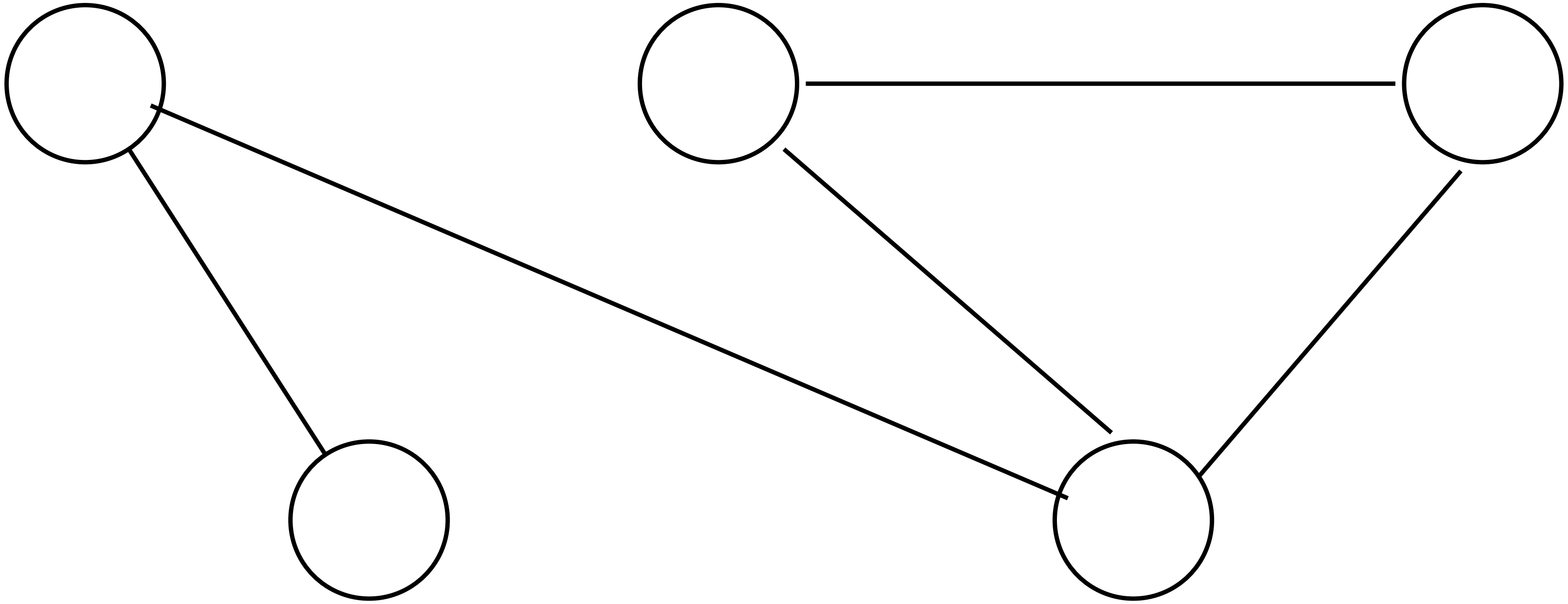
Les graphes

Graphe non orienté

$G = (V,E)$ est un graphe non orienté si :

- V est un ensemble de sommets
- E est un ensemble d'arêtes

Exemple

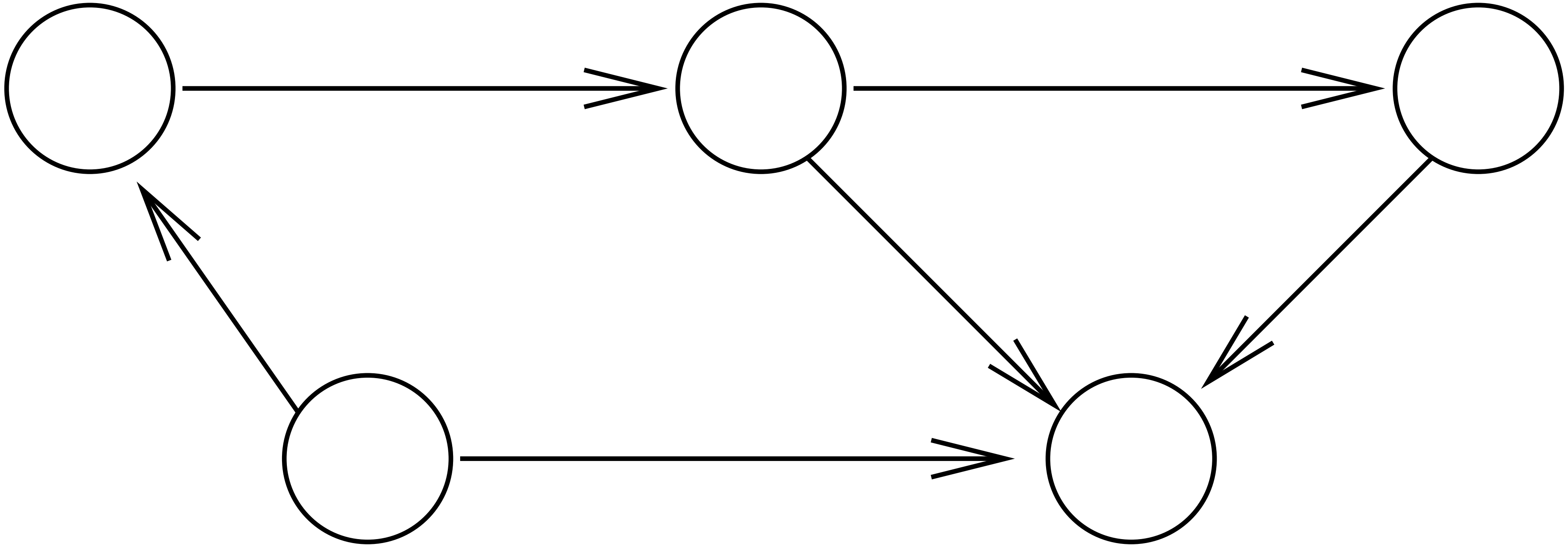


Graphe orienté

$G = (S,A)$ est un graphe orienté si :

- S est un ensemble de sommets
- A est un ensemble d'arcs

Exemple



Les boucles

- Dans un graphe orienté les boucles existent
- Dans un graphe non orienté, elles n'existent pas

Vocabulaire

- Si (u,v) est un arc d'un graphe orienté $G = (S,A)$, on dit que (u,v) part de u et arrive en v
- Si (u,v) est une arête d'un graphe non orienté $G = (V,E)$, on dit que (u,v) est incidente à u et v.

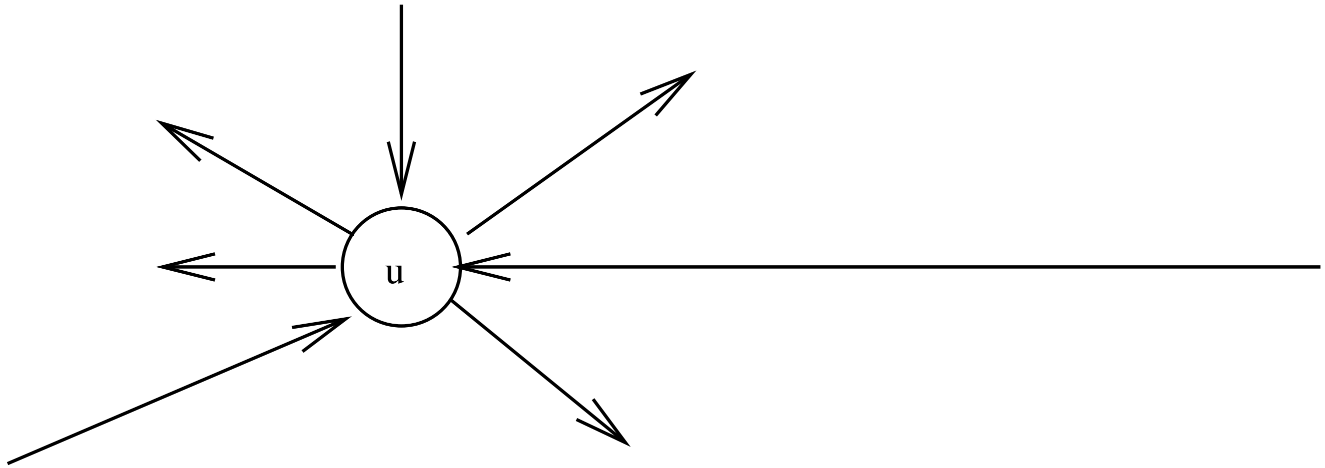
Degré

- Dans un graphe non orienté, le degré d'un sommet est le nombre d'arêtes qui lui sont incidentes.

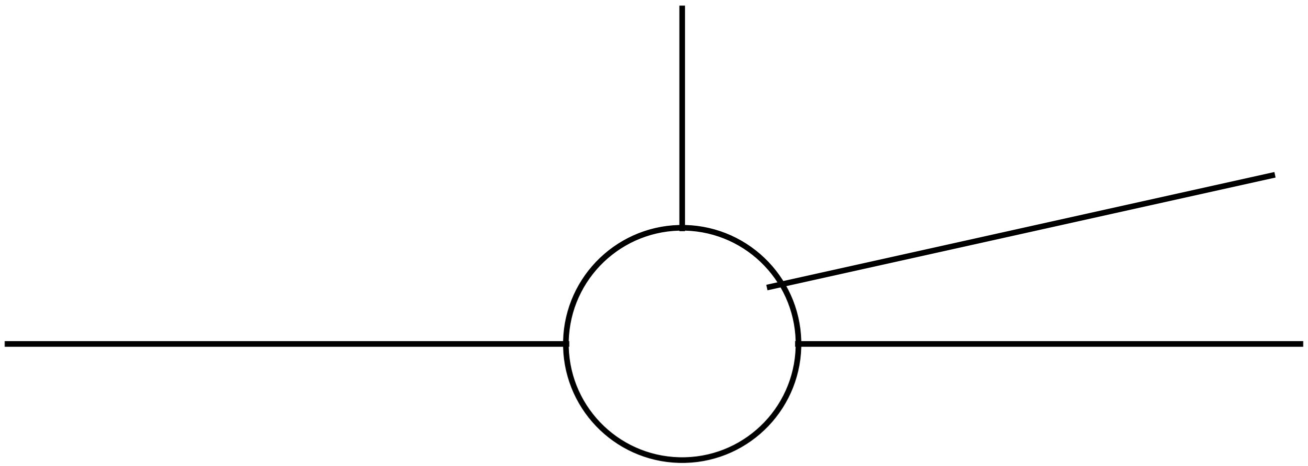
- Si un sommet est de degré 0, il est dit isolé.

- Dans un graphe orienté, le degré sortant d'un sommet est le nombre d'arcs qui en partent, le degré rentrant est le nombre d'arcs qui y arrivent et le degré est la somme du degré entrant et du degré sortant.

Exemple



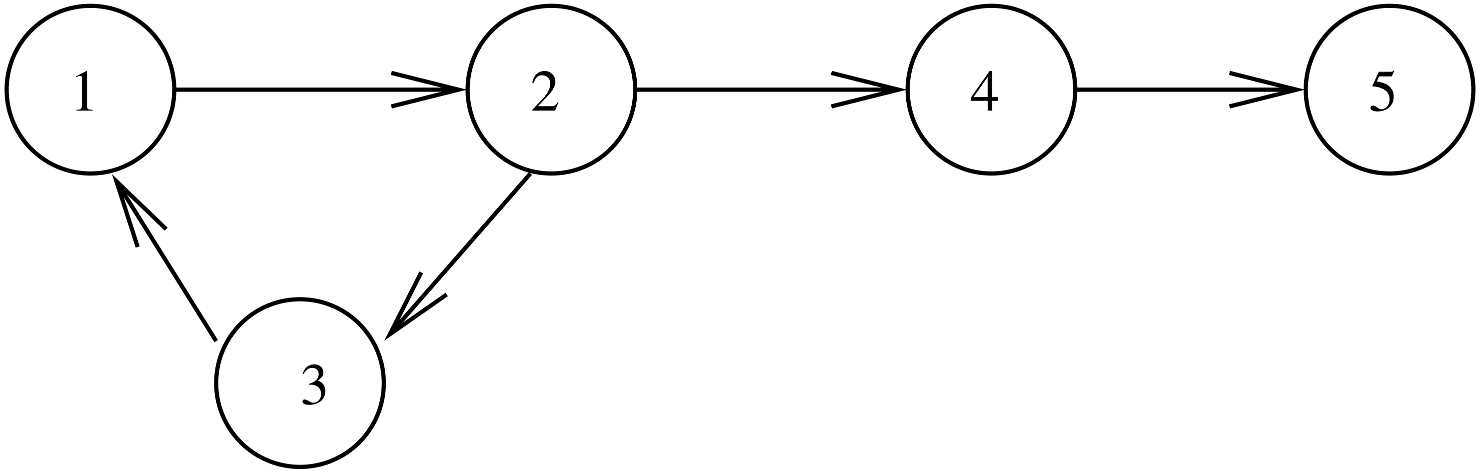
Exemple



Notion de chemin Dans un graphe orienté $G = (S,A)$, un chemin de longueur k d'un sommet u à un sommet v est une suite de sommets $(u_0,u_1,\ldots,u_k)$ telle que $u = u_0$, $v = u_k$ et $(u_{i-1},u_i) \in A$ pour tout i dans $\{1,\ldots,k\}$.

Un chemin est dit élémentaire si ses sommets sont distincts deux à deux.

Exemple



Un chemin $(u_0,u_1,\ldots,u_k)$ forme un **circuit** si $u_0 = u_k$.

Chemins (suite) On définit dans les graphes non orientés la notion correspondante de chaîne.

Une chaîne $(u_0,u_1,\ldots,u_k)$ forme un cycle si $k \geq 3$ et si $u_0 = u_k$.

Un graphe sans cycle est dit acyclique.

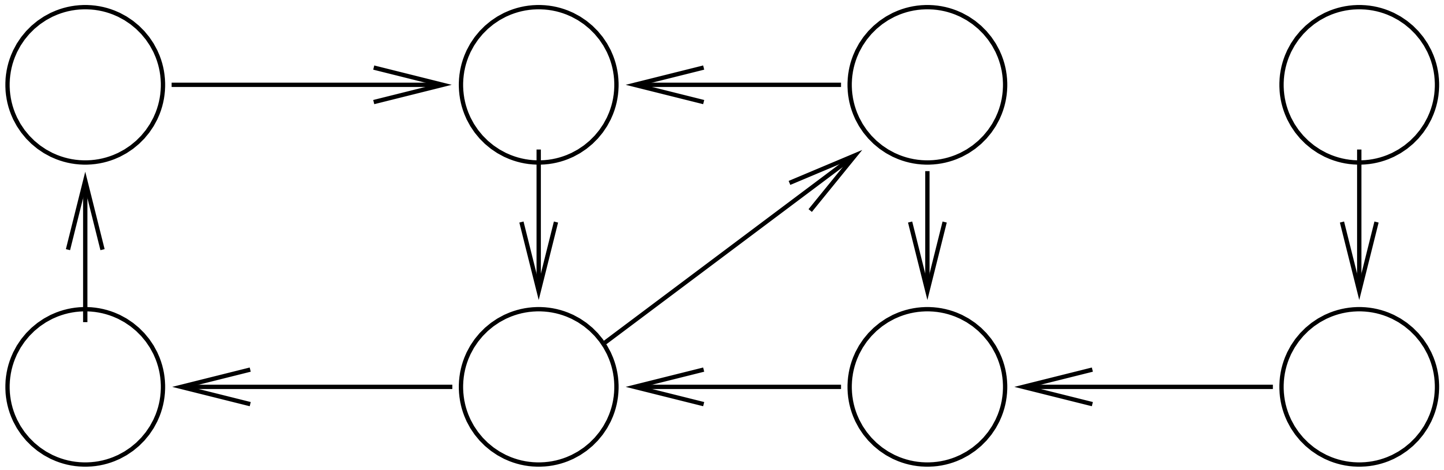
Connexité

Un graphe non orienté est connexe si chaque paire de sommets est reliée par une chaîne.

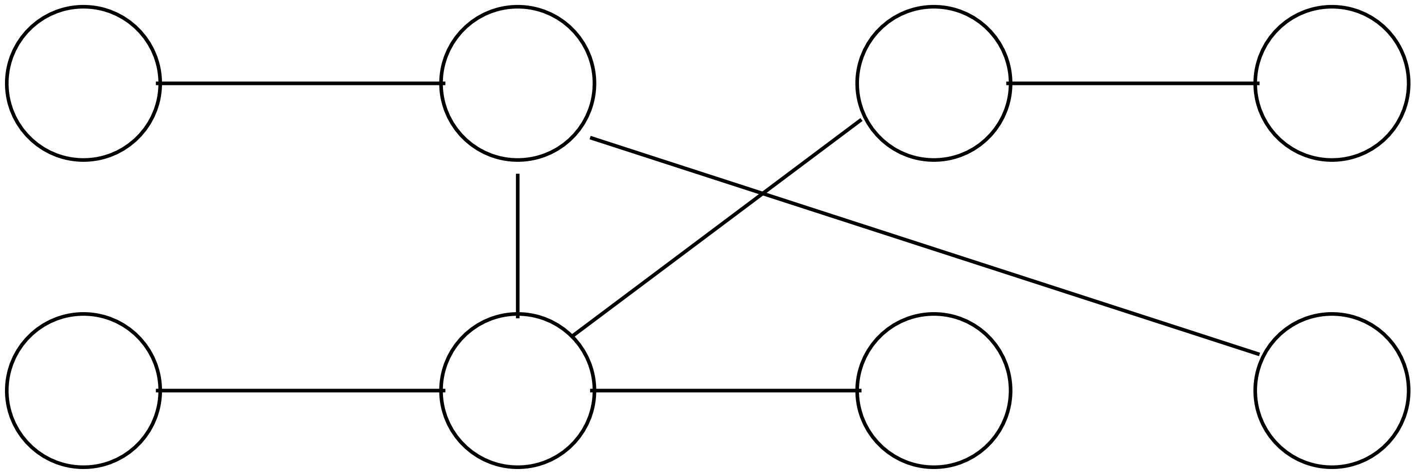
Les composantes connexes sont les classes d'équivalence de sommets induites par la relation d'accessibilité.

Un graphe orienté est fortement connexe si chaque sommet est accessible à partir de n'importe quel autre.

Exemple



Exemple



Représentation des graphes

Motivation

Pour pouvoir manipuler de manière efficace un graphe, il faut le représenter par une structure particulière.

Deux grands formalismes s’oppposent et se complètent dans ce but.

On va ainsi voir les notions de matrices et de listes d’adjacences.

Liste d'adjacence

- Représentation compacte pour des graphes creux (graphes avec $|E| << |V|^2$)

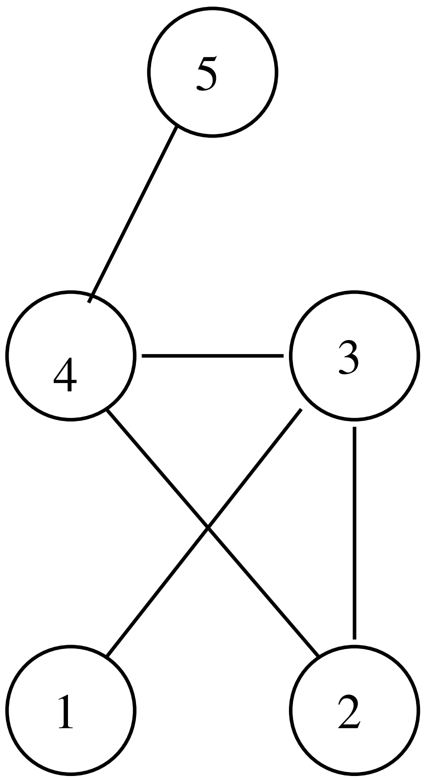
Matrice d'adjacence

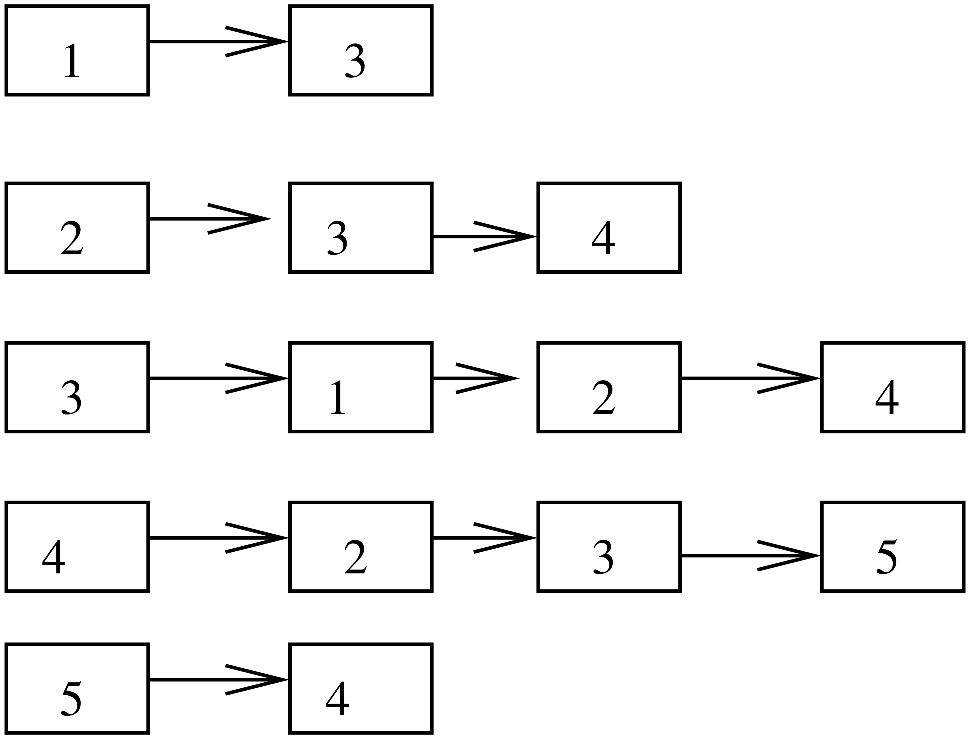
- Représentation compacte pour des graphes dense (graphes avec $|E| \approx |V|^2$)

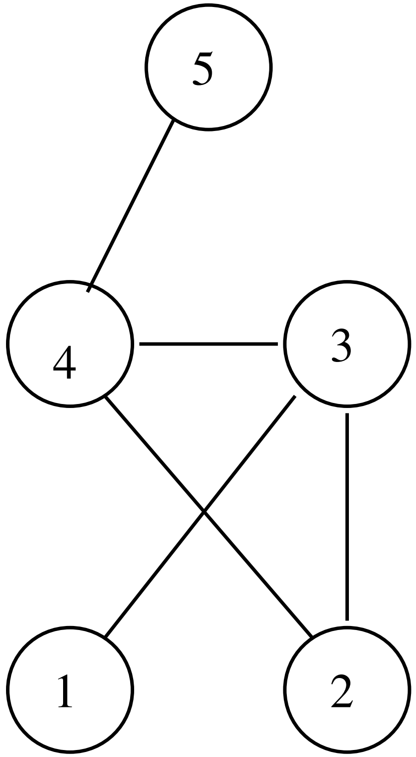
Orienté vs non orienté

Le principe est le même

Cas non orienté







1	0	0	1	0	0
2	0	0	1	0	0
3	1	1	0	1	0
4	0	0	1	0	1
5	0	0	0	1	0
	1	2	3	4	5

Exercice

Etant donné la représentation par une liste d’adjacence d’un graphe, combien de temps faut-il pour trouver le degré sortant/entrant de chaque sommet ?

Exercice

Donner la représentation en liste d’adjacence d’un arbre binaire complet.

Donner également la représentation du même graphe en tant que matrice d’adjacence.

Exercice

G^2 est le graphe orienté construit à partir du graphe orienté G sur les mêmes sommets, mais dont les arcs sont différents :

- Il existe un arc entre deux sommets u et v si il existe un chemin entre u et v dans G.

Donner un algorithme pour calculer G^2 à partir de G selon que l'on ait l'une ou l'autre des représentations.

Les complexités sont-elles semblables ?

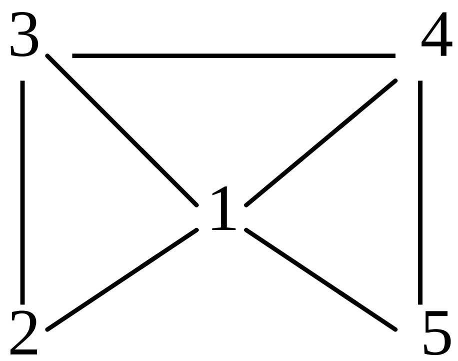
Exercice

Montrer que le degré de tout sommet s de G est inférieur à n, et que G ne peut contenir à la fois un sommet de degré 0 et un sommet de degré $n - 1$; en déduire que G admet au moins deux sommets de même degré.

Exercice

Soit G un graphe non orienté. On se propose de colorier G, les “couleurs” étant des entiers positifs, au moyen de l’algorithme suivant (dit **glouton**) : on colorie successivement tous les sommets en utilisant chaque fois la plus petite couleur disponible (non utilisée pour colorier un sommet voisin).

Appliquer cet algorithme au graphe



en supposant que les sommets sont examinés dans l’ordre $1,2,3,4,5$, puis en supposant que les sommets sont examinés dans l’ordre $1,2,3,5,4$. En déduire que l’algorithme glouton n’est pas optimal (en ce sens qu’il peut utiliser plus de couleurs que nécessaire).

Hachage

- Les ensembles sont des structures essentielles en informatique.
- Nous avons vu au cours précédent comment représenter des ensembles de manière simple et non satisfaisante.
- Nous voudrions pouvoir représenter des ensembles de valeurs provenant d'ensembles très grand.

Il y a trois opérations importantes sur les ensembles :

- AJOUTER un élément à un ensemble
- SUPPRIMER un élément d'un ensemble
- RECHERCHER un élément dans un ensemble

Définition :

Un ensemble dynamique muni des trois opérations ci-dessus est un dictionnaire.

exemples :

- catalogue dans une bibliothèque

  éléments : des livres

  clé : nom de l'auteur

  opérations : ajout, suppression et recherche

- tableau de symboles pour un compilateur

  éléments : identificateurs et informations pertinentes (type...)
  clé : nom de l'identificateur
  opérations : ajout et recherche

On supposera dans la suite que les clés appartiennent à $U \subseteq \mathbb{N}$.

Lorsque ce ne sera pas le cas, il faudra utiliser un codage des clés.

exemple :
Comment coder l'identificateur "pt" ?

On aura alors un ensemble $E \subseteq U$ à ranger, la taille de E est inconnue, mais peut être nettement plus petite que celle de U

Tableaux d’adressage direct.

Principe

Pour représenter un ensemble dynamique, on utilise un tableau $T[0..n-1]$, où l'on range la clé x à l'adresse x.
Si l'ensemble ne contient pas la clé x, alors $T[x] = NIL$.

Exercice :

- Donner l'implémentation des trois opérations.
- Donner la représentation de l'ensemble $\{2,3,5\}$ lorsque $U = [0..7]$.

Complexité

$\Theta(1)$ par opération en pire cas.

Problème

$\mid T \mid = \mid U \mid$ et U peut être un ensemble énorme : il y a par exemple environ 10^9 identificateurs en Fortran.

Ce n'est donc pas une solution satisfaisante la plupart du temps !

Tableaux de hachage avec chaînage.

On utilise toujours un tableau $T[0,..,n-1]$.

On range la clé x à l'adresse $h(x)$, où

$$h : U \longrightarrow \{0,1,2,...,n-1\}$$

est une fonction de hachage.

On appelle $h(x)$ la valeur de hachage de la clé x.

Avantage

T peut être de taille beaucoup plus petite que l'ensemble U.

problème

Deux clés peuvent être en collision (=avoir la même valeur de hachage).

Exercice

Donner un exemple de collision.

Une méthode de résolution des collisions est le chaînage.

L'idée est de mettre toutes les clés avec la même valeur de hachage dans une liste chainée.

$\forall j, 0 \leq j \leq n-1$, $T[j]$ contient un pointeur vers la liste contenant les clés dont la valeur de hachage est j.

Les opérations sont les suivantes :

HACHAGE-CHAINE-RECHERCHER (T,x)

chercher x dans la liste $T[h(x)]$

HACHAGE-CHAINE-AJOUTER (T,x)

ajouter x à la fin de la liste $T[h(x)]$

HACHAGE-CHAINE-SUPPRIMER (T,x)

supprimer x de la liste $T[h(x)]$

Complexité du hachage avec chaînage

On suppose que le temps de calcul de la valeur de hachage est $\mathcal{O}(1)$.

Soit $T[0,\ldots,m-1]$ un tableau de hachage de taille m.

On cherche à ranger dans T un ensemble E à n clés.

Définition

Le taux de remplissage du tableau est $\alpha = \dfrac{n}{m}$

Complexité en pire cas de la recherche

C'est $\Theta(n)$ car au pire toutes les clés de E ont la même valeur de hachage.

Complexité en moyenne de la recherche

On va faire des hypothèses sur les éléments de l'ensemble et leurs hachages.

1. Les éléments de E sont choisis uniforméments dans U.

2.

$$\forall j, \ \mid h^{-1}(j) \mid = \frac{\mid U \mid}{m}$$

C'est à dire que si y est un éléments aléatoire de U, alors $h(y)$ est un éléments aléatoire de $[0,\ldots,m-1]$

Théorème :

Avec les hypothèses précédentes, la complexité en moyenne de la recherche pour le hachage avec chaînage est :

$$\Theta(1+\alpha)$$

corollaire :

Si α est une constante, alors la complexité est $\Theta(1)$

Exemples de fonctions :

$h(x) = x \mod n$ **pour un** n **bien choisi.**

Tableau de hachage direct.

Toutes les clés sont rangés dans le tableau de hachage lui même. Et donc chaque case de T contient une clé ou NIL. Et le taux de remplissage est forcément plus petit que 1.

Les collisions sont résolues par calcul à l'intérieur du tableau.

Pour ajouter une clé nouvelle au tableau, on essaie plusieurs cases jusqu'à en trouver une vide.

La fonction de hachage devient alors une fonction à 2 arguments :

$$h : U \times \{0,\ldots,m-1\} \longrightarrow \{0,\ldots,m-1\}$$

et on essaie successivement les cases $T[h(x,0)]$, $T[h(x,1)]$...

Attention ! pour que cela fonctionne, il faut que la suite $h(x,0),h(x,1),\ldots,h(x,m-1)$ soit une permutation de $0,\ldots,m-1$.

On notera cependant que cette structure ne supporte pas l'opération de suppression.

Exercice

Ecrire les algorithmes d’ajout et de recherche.

Présentation de techniques pour construire les essais successifs

Hachage linéaire :

$$h(x,i) = f(x) + i \mod n$$

Hachage quadratique :

$$h(x,i) = f(x) + i^2 \mod n$$

Double hachage :

$$h(x,i) = (f(x) + i \times g(x)) \mod n$$

On remplit par hachage un tableau de 20 cases numérotées de 0 à 19, en insérant successivement 20 clés dans l'ordre suivant:

$$8\ 14\ 15\ 11\ 12\ 18\ 10\ 9\ 0\ 4\ 13\ 1\ 6\ 2\ 5\ 17\ 3\ 19\ 7\ 16.$$

On considère les deux fonctions de hachage suivantes:

x	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$h(x)$	8	15	4	11	2	5	9	17	3	2	3	18	17	8	12	19
$k(x)$	7	13	3	9	11	3	13	3	11	13	1	9	3	7	19	3

1. Représenter le résultat successivement pour:
   (a) un hachage avec chaînage utilisant la fonction h.
   (b) un hachage linéaire utilisant la fonction h.
   (c) un double hachage utilisant les fonctions h et k.

1. On veut insérer n clés tirées uniformément dans un tableau de m cases. On suppose que la fonction de hachage est uniforme. Calculer la probabilité que les n valeurs de hachage soient toutes distinctes.
2. Pour $m = 365$, calculer le plus petit n tel que cette probabilité soit $< 1/2$.

Polynômes

Problème de l'évaluation des puissances

Input : x,n avec $x \in \mathbb{R}$ et $n \in \mathbb{N}$

Output : x^n

L'opération élémentaire est la multiplication, et l'on veut minimiser ce nombre d'opérations. On va étudier un problème équivalent.

Exemple

Combien de multiplications au minimum pour x^5 ?

Problème de la chaîne additive

Input : $n \in \mathbb{N}$

Output : une chaîne additive pour n

Définition

Une chaîne additive pour n est une suite d'entiers $(a_0, a_1, \ldots, a_r)$ telle que :

1. $a_0 = 1$

2. $a_r = n$

3. $\forall i (1 \leq i \leq r)\ \exists k, j (0 \leq k \leq j \leq i)$ tel que $a_i = a_j + a_k$

Equivalence ?

Oui car $x^p.x^q = x^(p + q)$

La longueur de la chaîne additive la plus courte est donc égale au nombre de multiplications pour la puissance.

Exemple

1,2,3,5,7,10,13,20,40,60,70,75 est une chaîne additive de longueur 11 pour 75.

Notation :

$L(n)$ est la longueur de la plus courte chaîne additive pour n.

Proposition 1 :

$$L(n) \geq \lceil \log_2 n \rceil$$

Preuve :

$\forall i$ on a $a_i \leq 2^i$

Proposition 2 :

$$L(2^p) = p$$

Preuve :

$(1,2,4,\ldots,2^{(p-1)},2^p)$ est une chaîne additive pour 2^p.

Proposition 3 :

$$L(n) \leq \lfloor \log_2 n \rfloor + d(n) - 1$$

avec $d(n)$ qui est le nombre de '1' dans l'écriture binaire de n.

Preuve :

Soit $n = 2p + \varepsilon$ avec $\varepsilon = 0$ *ou* 1.
Soit A une chaîne additive calculant p.

- Si $\varepsilon = 0$ alors $A \cup \{p + p\}$ calcule n.

- Si $\varepsilon = 1$ alors $A \cup \{p + p, 2p + 1\}$ calcule n.

et donc : $L(n) \leq L(p) + 1 + \varepsilon \leq \log_2 p + d(p) - 1 + 1 + \varepsilon$.
D'où le résultat.

L'algorithme binaire pour le problème :

1. Ecrire n en binaire : $b_0 b_1 b_2 \ldots b_k$

2. $A := (1)$

3. Pour i allant de 1 à k :

   si $b_i = 0$ alors $A := A \cup (2 \times last(A))$

   sinon $A := A \cup (2 \times last(A), 2 \times last(A) + 1)$

Exercice :

Appliquer l'algorithme à 75.

Appliquer l'algorithme à 15, est-il optimal ?

Karatsuba

Motivation

• Les techniques de multiplications rapides sont très utilisées pour les calculs informatiques.

• Actuellement les meilleures techniques sont des techniques de transformation de domaines.

• Mais ces dernières sont difficiles à implémenter et donc on utilise encore d'autres techniques.

• La plus connue est la méthode de Karatsuba (1962)

• La multiplication classique de deux nombres de taille n se fait en $\mathcal{O}(n^2)$

L'algorithme

On prend deux nombres U et V en base X, alors :

$$U(X) = a_0 + a_1X + a_2X^2 + ... + a_{n-1}X^{n-1}$$

$$V(X) = b_0 + b_1X + b_2X^2 + ... + b_{n-1}X^{n-1}$$

Et donc :

$$U \times V = a_0b_0 + (a_0b_1 + a_1b_0)X + ...$$

Comment faire plus simple ?

Supposons :

$$U(X) = a_0 + a_1 X$$
$$V(X) = b_0 + b_1 X$$

Alors :

$$U \times V = a_0 b_0 + ((a_0 + a_1)(b_0 + b_1) - a_0 b_0 - a_1 b_1) X + a_1 b_1 X^2$$

On fait alors 3 multiplications au lieu de 4 ! (mais plus d'additions).

On peut toujours généraliser :

$$U(X) = a_0 + a_1 X + a_2 X^2 + ... + a_{n-1} X^{n-1}$$

peut s'écrire :

$$U(X) = (a_0 + a_1 X + a_2 X^2 + ... a_{n/2} X^{n/2}) + X^{n/2}(a_{n/2+1} X + ... + a_{n-1} X^{n/2-1})$$

Et on utilise donc une approche "diviser pour régner"...

Complexité

$$T(0) = 1$$

$$T(m) = 3T(m/2) + 5m$$

La solution de cette équation est $\mathcal{O}(m^{\log_2 3}) = \mathcal{O}(m^{1,59})$

Révisions

Exercice 1

Démontrer que, pour tous réels a et b, on a : $(x+b)^a = \Theta(x^a)$.

Exercice 2

Soit T un tableau de $2n$ entiers distincts. Donner un algorithme en pseudo-langage qui permette de calculer à la fois le maximum et le minimum du tableau avec au plus $3n$ comparaisons. On pourra parcourir le tableau dans l'ordre en regardant les éléments par paires (on regardera ensemble $T[1]$ et $T[2]$, puis ensemble $T[3]$ et $T[4]$, etc.), et en maintenant le maximum et le minimum courant à chaque étape.

Exercice 3

On appelle un élément x d'un tableau de taille n majoritaire si x apparaît strictement plus que $n/2$ fois dans le tableau. Donner un algorithme "diviser pour régner" qui calcule l'élément majoritaire s'il existe en supposant **qu'on ne peut que** tester l'égalité entre deux éléments. Analyser la complexité de l'algorithme.

Donner un algorithme linéaire pour le même problème. Idée: On suppose qu'il y a un élément majoritaire. Si on enlevait une paire d'éléments différents du tableau et on répétait cela jusqu'à ce qu'il ne restait que des éléments d'une même valeur, cette valeur serait majoritaire. Comment utiliser cette idée quand on parcourt le tableau du début vers la fin ?

Exercice 4

Calculer le nombre maximal de nœuds d'un arbre k-aire de profondeur p.