

Introduction

- approche algorithmique \rightarrow opération élémentaire
 \hookrightarrow ne dépend pas de l'environnement.

- Temps d'exécution \rightarrow # opération élémentaire
 \hookrightarrow taille de l'instance : ordre de croissance

• $\mathcal{O}(g(n))$

$$f(n) = 17n^2 + 8n \leq 25n^2$$

$$\left\{ \begin{array}{l} f(n) \in \mathcal{O}(n^2) \\ 17n^2 + 8n \leq cn \quad n_0 \neq \end{array} \right.$$

• $\mathcal{S}\mathcal{Q}(g(n))$

$$f(n) = n^2 + 4n^2 + 3 \geq n^2 \rightarrow f(n) \in \mathcal{S}\mathcal{Q}(n^2)$$

• $\Theta(g(n))$

$$f(n) = n^2 + 4n^2 + 3 \leq 8n^2 \rightarrow f(n) \in \mathcal{O}(n^2)$$

$$f(n) = n^2 + 4n^2 + 3 \geq n^2 \rightarrow f(n) \in \mathcal{S}\mathcal{Q}(n^2)$$

$$\hookrightarrow f(n) \in \Theta(n^2) \quad (\text{Sandwich})$$

Si: $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = \begin{cases} c & f(n) \in \Theta(g(n)) \\ 0 & f(n) \in \mathcal{O}(g(n)) \\ \infty & f(n) \in \mathcal{S}\mathcal{Q}(g(n)) \end{cases}$

$$\mathcal{O}(n) \subset \mathcal{O}(n^2)$$

$$f(n) = \log_a n, g(n) = \log_b n \rightarrow \lim_{n \rightarrow \infty} \frac{\log_a n}{\log_b n} = \frac{\ln(n)/\ln(a)}{\ln(n)/\ln(b)} \cdot \frac{\ln(b)}{\ln(a)} = \log_a b$$

$$x = \log_a n \rightarrow a^x = n \quad x \ln a = \ln n \rightarrow x = \frac{\ln n}{\ln a}$$

* Opération Béromètre

ordre de croissance de n , qui s'execute plus souvent.

$$T(n) = \sum_{i=1}^{\log_2 n} \sum_{j=1}^n 1 = n \log_2 n$$

$\left. \begin{array}{l} 2^0 \\ 2^1 \\ 2^2 \\ 2^3 \\ \vdots \\ 2^K = n \end{array} \right\} K = \log_2 n$

* Pire Cas et Meilleur Cas

$$T_W(n) \leftarrow T_B(n) \quad \text{s: dépend pas de } n \rightarrow T_B = T_W = T_{avg}$$

$$T_B(n) \leq T(s) \leq T_W(n) \quad \text{exemple de trie}$$

$$\left. \begin{array}{ll} \text{meilleur:} & T(s) \in \mathcal{O}(1) \\ \text{Pire:} & T(s) \in \mathcal{O}(n) \end{array} \right. \quad 1 \leq T(s) \leq n$$

* Algo de trie

$$T(n) = T(n/2) + 1$$

$$T(1) = 1$$

$$\begin{aligned} T(n) &= T(n/2) + 1 & \dots &= T(1) + k \\ &= T(n/4) + 2 & \vdots &= T\left(\frac{n}{2^k}\right) + k \\ &= T(n/8) + 3 & n/2^k = 1 & \rightarrow n = 2^k \end{aligned}$$

$$K = \log_2 n \quad T(n) = \log_2 n + 1$$

fonction harmonique:

$$T(n) = 1 + \log_2(n)$$

$$T(n) \in \Theta(\log_2 n) \text{ pour } n = 2^K \quad f(n) = \log_2 n$$

$$f(n) < \begin{array}{l} \text{non décroissante} \\ \log_2(2n) \in \Theta(\log_2(n)) \end{array} \quad \checkmark$$

Étape :

$$\left. \begin{array}{l} \text{horning} \\ \#1) \text{ Vérifier si } f(n) \text{ est non décroissante } f(n) \leq f(n+1) \\ \#2) \quad " \quad \text{ si } f(2n) \in \Theta(f(n)) \end{array} \right\}$$

$$\#3) \quad " \quad \text{ si } T(n) \text{ est non décroissante + } T(n) \in \Theta(f(n)) \\ \text{pour } n = b^k \quad b \geq 2$$

$$\text{Alors } T(n) \in \Theta(f(n))$$

On gère, vérifier $f(n)$ est harmonique et que $T(n)$ est non décroissante et que $T(n) \in \Theta(f(n))$ pour $n = b^k \quad b \geq 2$

$$\text{Alors } T(n) \in \Theta(f(n)) \quad \forall n \geq 2 \quad \checkmark$$

* Récursion : si faux condition d'arrêt.

$$n! = n(n-1)! = n(n-1)(n-2)! = \dots = n(n-1) \dots 1$$

} Acc⁺ $\rightarrow (1-0) = 0$
ou $n=1$

* Structures de données

- Vecteur
- liste
- files
- Piles

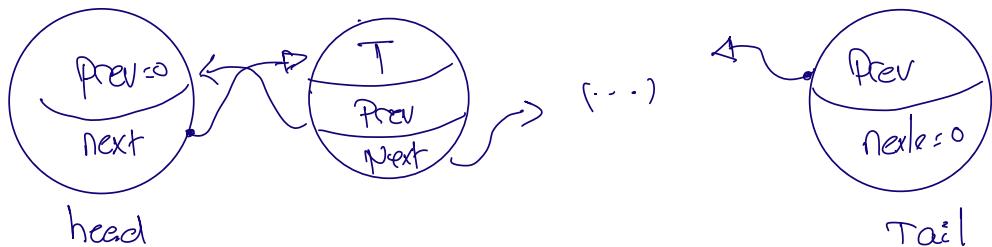
* Vecteur

- Accès au élément en $O(1)$
- supprimer ou insertion $O(n)$ } $k^{\text{ième}}$ élément
- élément occuper dans la mémoire. Tableau dynamique \neq tableau fixe
- Pour insérer, il faut décaler de la mémoire en $O_w(n)$ }
- On peut modifier la taille } $O_B(1)$

* liste

- supprimer un élément à la position courante
- inserer un élément avant la position courante
- accéder à l'élément suivant (`++`)
- " .. précédent (doublement)"

On peut par accéder au $k^{\text{ième}}$ élément en $O(1)$



On utilise la liste pour implementer la file et le piles

{ File : first in first out \rightarrow
Pile : first in last out $\xrightarrow{\text{last}}$ }

vector : vecteur

list : liste

forward list : liste simplement chaînée

queue : file

stack : pile

dequeue : file dans 2 direction

* Graphes

- Connexe pour graphe non-orienté : Si il existe un chemin entre n'importe quel sommet
- un graphe non-orienté connexe à au moins $(n-1)$ arêtes
- " " de n sommet avec plus de $(n-1)$ arête possède au moins un cycle.
- Un graphe orienté est faiblement connexe s'il est connexe en remplaçant les arcs par des arêtes
- Fortement connexe si il existe un chemin entre n'importe quel sommet

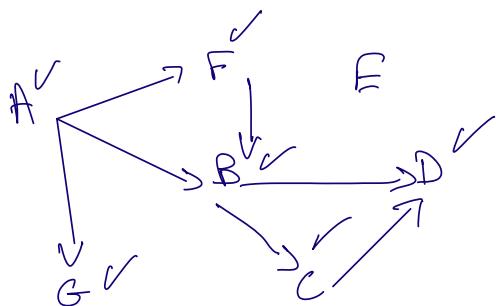
* Matrice adjacente : Pour $O(1)$ pour savoir si l'arc (i,j) existe
 $O(n)$ pour obtenir le voisin du sommet
 $O(n^2)$ pour implémenter

* liste d'adjacence : $\Theta(n+m)$ espace en mémoire
 $\Theta(V(n))$ tout les nœud adjacent vs $\Theta(n)$ matrice
 désavantage : accès en $\Theta(V(n))$ pour arc (i_j) v.s $\Theta(1)$

* Parcours en profondeur :

- 1) tout mettre à faux
- 2) empiler et marquer $V(A)$
- 3) Tant que la pile n'est pas vide:
 dépiler, pour chaque nœud adj
 le marquer et empiler

exemple:



Pile = A

$A \checkmark$

A : FBG

$Q = \{ F \ B \ G \}$

$Q = \{ B \ G \}$

$Q = \{ G \}$

$Q = \{ D \ C \ A \}$

$Q = \{ C \ A \}$

$Q = \{ A \}$

$Q = \{ \}$ pile

sinon on utilise explore:

explore(v)

$V = \{v\}$

{point de adj a v}

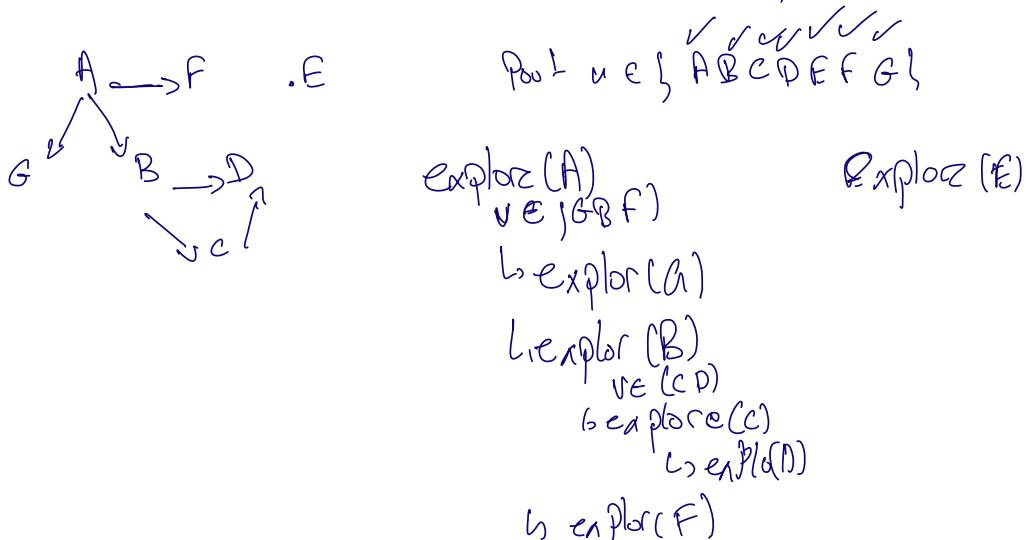
si u faux explore(v)

Parcour Profondeur ($G(S,A)$)

Point Aude \subseteq : visité $f(v) = \text{faux}$

.. u .. faux

Si $\text{visit}(v) = \text{faux} \rightarrow \text{explor}(v)$ ✓



Profondeur $\in \Theta(n+m)$

* explore trouve la composante d'un graphe orienté

$\text{explor}(v)$:

debut [v] = clock

clock += 1

$\text{visit}(v) = \text{visi}$

Pour u adj. à v faire:

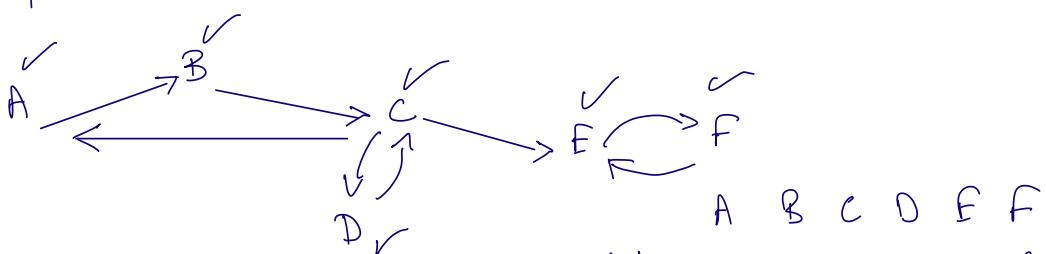
si $\text{visit}(u) < \text{clock} \rightarrow \text{explor}(u)$

$\text{fin}(v) = \text{clock}$

clock += 1

pile. empile(v)

Exemple:



$\text{explor}(A)$

↳ $\text{explor}(B)$

↳ $\text{explor}(C)$

↳ $\text{explor}(D)$

↳ $\text{explor}(E)$

↳ $\text{explor}(F)$

fin: 12 11 10 9 8

pile: {D F E C B A}

ordre topologique graphe acyclique, obtenu grâce à
Recherche en profondeur.

- Si v appartient à une composante fortement connexe peut explorer voir uniquement le noeud de cette composante
- Car acyclique, le premier noeud n'est pas fini.
- Dernier noeud empilé dans parcours en profondeur \rightarrow Sources
- Le dernier noeud empilé \rightarrow Fin grande

$$C \xrightarrow{ } D \rightarrow \max \{ \text{fin}(v) \mid v \in C \} > \max \{ \text{fin}(v) \mid v \in D \}$$

Preuve: si on commence à C et termine à D $\text{fin}(v)$ grand
 .. \hookrightarrow $\rightarrow D$ on reste à D $\text{fin}(v)$ petit

Graphe inverse : le dernier empilé est un puits

idée est de trouver composante fortement connexe.

V^* est le dernier noeud de G^I : puits

$\text{explorer}(V^*)$ nous donne la composante fortement connexe de ce puits

$\text{explorer}(V, G)$

$$\text{visit}(v) = \text{visit}(z) = \text{visit}(u(v))$$

Pour tout noeud adjacent à v

si $\text{fin}(u) < \text{fin}(v)$

G^P

Q : Parcours en Profondeur de G^P

C : en mélange composante

Tant que Q n'est pas vide

V. = depiler(Q)

if NM

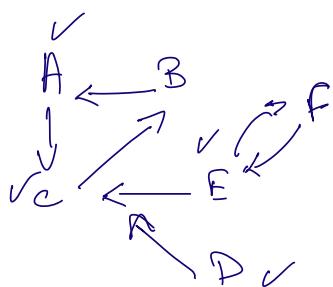
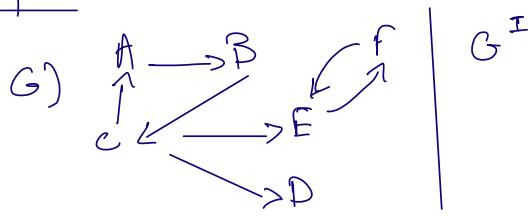
$Z = \{t\}$

explor(v, a)

$C = C \cup \{t\}$

Retour C

exemple:



A B C D E F

debut: 1 3 2 7 9 0 explor(A)

fin: 6 4 ← 8 12 11 ↳ explor(C)

Pile: B C A D F E ↳ explor(B)

$Q = \{B, C, \checkmark A, \checkmark D, \checkmark F, \checkmark E\}$

explor(D)

explor(E)

explor(E)

$Z = \{E\}$

↳ explor(F)

$Z = \{E, F\}$

$C = \{EFF\}$

explor(D)

$Z = \{D\}$

$C = \{EFF, FD\}$

explor(D)

$Z = \{A\}$

↳ explor(B)

↳ explor(F)

$Z = \{B, A\} \cup \{C\}$

$$C = \{ \{E\} \cup \{D\} \cup \{ABC\} \}$$

$\Sigma = \{A, B, C\}$

KOSA($\alpha, \beta \in \Sigma^*(n+m)$)

* Parcours en largeur

Le noeud le plus proche, on utilise une file

$$\begin{cases} V(v) \text{ faux} \\ \text{Prédecesseur}(v) = \emptyset \end{cases}$$

$$V(v) = V_{\text{init}}$$

Q. enfile(v)

Tant que Q n'est pas vide
 $v^* = \text{defiler}(Q)$

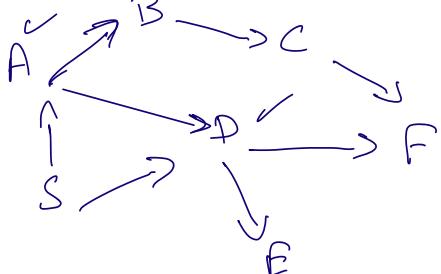
Pour $\forall u \text{ adj } v^*$ $V(u) = \text{falso}$

$$V(u) = V_{\text{ini}}$$

$$P(u) = u^*$$

Q. defile(v)

Exemple:



A	B	C	D	E	F	S
PO	S	A	B	S	D	D

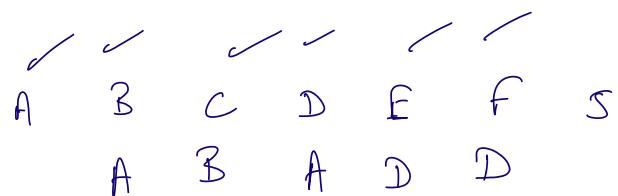
Defile = S

A D

D B

D E F B C

les algorithmes



file: A B P C E F

B P C E F

Bellman-Ford: Pour $i = 1$ à n

() Pour $v \in V$ faire $d(v) = \infty$

() $v = s$ faire $d(v) = 0$

après vérifier si pour tous les couples :

$s: d(v) > d(u) + w(v, u)$ faire

* Si pas de cycle négatif. :

1) $C: L \supset \text{corr} \rightarrow d(v_k) = D(s, v_k)$

2) $C \subset \text{pas de cyc} \rightarrow$ donc $C: \text{contient au plus}$

3) à chaque itération au moins une flèche (≤ 1)

$c \rightarrow c \rightarrow \dots$

u) d'après le résultat \dots

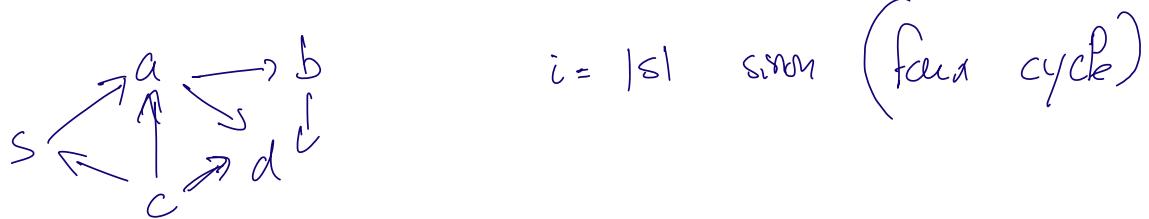
$$\text{on } d(v) = d(p(v)) + w(v, p(v))$$

$$d(v) \leq d(u) + w(v, u) \quad \text{dès réécrire une flèche}$$

si $\forall u \text{ done } d(v) \leq d(u) + w_{uv} \quad (v_0 = v_0)$

$$d(v_0) + d(v_1) + \dots + d(v_k) \leq d(v_0) + c(v_0, v_0) + \dots + d(v_k) + c(\dots)$$

$0 \rightarrow \omega$



$i = |S| \text{ sinon } (\text{faux cycle})$

* si un cycle (i.e. loop) fait au moins 1 pas pour
 $|S| < |S|$ donc faux

* si pas de cycle $|S| = |V \setminus R| + \text{tree}$

G^T un seul relachement

si cycle Tri Topo Retour faux

Pas cycle:

C: $\angle \rightarrow$ cors

donc bondu
relachement

$$W = \begin{pmatrix} 0 & 1 & \infty & 8 \\ \infty & 0 & 4 & \infty \\ \infty & 7 & 0 & 9 \\ 0 & 2 & \infty & 0 \end{pmatrix} \quad P_0 = \begin{pmatrix} -1 & - & & 1 \\ - & 2 & - & - \\ -3 & - & 3 & - \\ 4 & 4 & - & - \end{pmatrix}$$

$$D_k[i,j] = \min(D_{k-1}[i,j], D_{k-1}[i,k] + D_{k-1}[k,j])$$

$$D_1 = \begin{pmatrix} 0 & 1 & \infty & 8 \\ \infty & 0 & 4 & \infty \\ \infty & 7 & 0 & 9 \\ 0 & 1 & \infty & 0 \end{pmatrix} \quad P_1 = \begin{pmatrix} - & 1 & - & 1 \\ - & - & 2 & - \\ - & 3 & - & 3 \\ 4 & 1 & 2 & - \end{pmatrix}$$

$$P_K[i,j] = P_{k-1}[K;j]$$