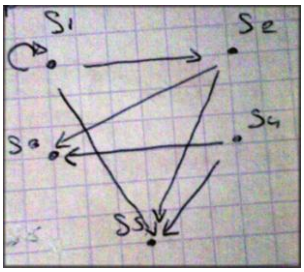
Tri topologique d'un graphe orienté

I. Graphe orienté (complet)

a.



b. G n'est pas complet, exemple : S_1 n'est pas lié à S_4

c. Définition d'un graph par ensemble de sommet et ensemble d'arrête
$\mathcal{G} :< S, A >$
$A = ? quand\ \mathcal{G}\ est\ complet, pour\ S$
En général $A \subseteq S * S$
Un graph complet équivaut à : $A = S * S$

d. $|S| = \mathcal{O}(n) \rightarrow |S * S| = \mathcal{O}(n^2)$

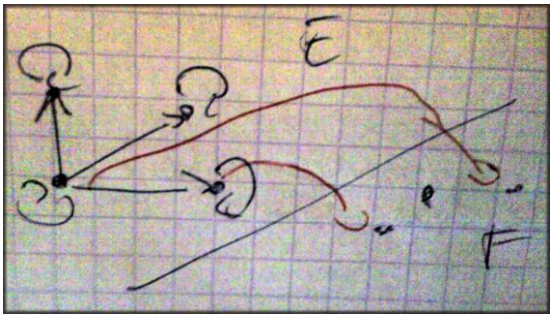
e. On ne précise pas si c'est un graph complet
Donc si $\mathcal{G}'$ est complet, $\mathcal{G} \leq \mathcal{G}'$
$A_{\mathcal{G}} \leq A_{\mathcal{G}'}$, $|A_{\mathcal{G}}| \leq c|A_{\mathcal{G}'}|$ avec c = 1 si complet
$|A_{\mathcal{G}}| \in \mathcal{O}(n^2)$

II. Cycle d'un graph

a. G1 à 0 cycle et une boucle
G2 à 2 cycle et une boucle : $S_1 \rightarrow S_3 \rightarrow S_4 \rightarrow S_2 \rightarrow S_1$ et $S_1 \rightarrow S_4 \rightarrow S_2 \rightarrow S_1$

b. $\mathcal{G} \equiv < S, A >$
$S = E \cup F$ avec $E \cap F = \emptyset$ et $|E| = |E| si\ |S| = 2k, k \in \mathbb{N}$
$|E| = |F[+1$ si $|S| = 2k + 1, k \in \mathbb{N}$ et il n'y a pas de cycle

Exemple pour avoir une intuition :



$|A_E| \in \mathcal{O}(n)$
$|A_F| \in \mathcal{O}(n)$
$|A| \leq |E * F| \in \mathcal{O}(n^2)$

III. Tri topologique

a. $\{S_5, S_2, S_1, S_3, S_4\}$ (par ordre topologique)

On crée donc une liste L que l'on remplie et tri petit a petit

$L = \emptyset$

$L = \{S_1\}$

$L = \{S_2, S_1\}$

$L = \{S_2, S_1, S_3\}$

$L = \{S_2, S_1, S_3, S_4\}$

$L = \{S_5, S_2, S_1, S_3, S_4\}$

b. Si on a un cycle, on a toujours un précédent, or un tri topologique s'effectue dans l'ordre de celui qui a aucun précédent à celui qui en a le plus

Il y a donc contradiction

Supposons avoir un cycle d'ordre k, $S_{i1}, S_{i2}, \dots, S_{ik}$

Alors on doit avoir $S_{i1} \leq S_{i2} \leq \cdots \leq S_{ik}$ avec $\leq$ signifiant precede

Mais aussi $S_{ik} \leq S_{i1}$ puisque c'est un cycle, donc impossibilité

d. De 1 à $n \rightarrow \mathcal{O}(n)$

e. a le nombre d'arrête = cardinalité de A

Pour tout S_j de Lsucc, on a au pire a execution

Comme $\mathcal{G}$ est acyclique, la liste des successeurs vaut n taille

Il y a au moins n arrette, donc $(n + a)$

Examen 2

1. Exo

a. $\lim_{n\to\infty} \frac{h(n)}{g(n)} = 0$

$\exists\, n_0\ tq\ \forall\, n \geq n_0, |\frac{h(n)}{g(n)} - 0| < \varepsilon$

$-\varepsilon < |\frac{h(n)}{g(n)}| < \varepsilon$

Comme $\forall n\ \in \mathbb{N}\ h(n)\ \in \mathbb{N}\ et\ g(n) \in \mathbb{N} \rightarrow 0 \leq h(n) \leq\ \varepsilon g(n)$

$h(n) + g(n) \leq\ \varepsilon g(n) + f(n) \leq 2g(n)$

$h(n) \in \mathcal{O}(g(n))$

b. 1) $c * g(n) \in \mathcal{O}(n)$

2) $g(n) \in \mathcal{O}(g(n))$

1)$\exists\, n_0\ tq\ cg(n) \leq c'g(n) =\ g(n) \leq \frac{\frac{c''}{c'}}{c'''}\big(c * g(n)\big) \rightarrow g(n) \in \mathcal{O}(c * g(n))$

2)$\exists\, n_0\ tq\ \forall\, n > n_0, g(n) \leq c'g(n) \leq c' * c''g(n) \rightarrow cg(n) \in \mathcal{O}(g(n))$

Donc on a prouvé 1) et 2), alors si $c > 0, \mathcal{O}\big(c * g(n)\big) =\ \mathcal{O}(g(n))$

c. $P(n) =\ a_1x^n\ +\ a_2x^{n-1} + \cdots + a_nx + a_{n+1}$

$\mathcal{O}\big(P(n)\big) =\ \mathcal{O}(\max\{a_1x^n\ +\ a_2x^{n-1} + \cdots + a_nx + a_{n+1}\})$

$=\ \mathcal{O}(a + x^n) =\ \mathcal{O}(x^n)$

2. Exo
   a. Cet algorithme vérifie s'il existe 2 cases avec la même valeur
   b. La complexité est en $\mathcal{O}(n^2)$:
      $(n-1)+(n-2)+\cdots+1 \rightarrow \sum_{i=1}^{n-1}(n-1)$
      $= \sum_{i=1}^{n-1} n - \sum_{i=1}^{n-1} i = n*(n-1) - \frac{(n-1)n}{2} = \frac{n(n-1)}{2}$
      Une configuration qui réalise le pire des cas est si toutes cases sont différentes
   c. Il est donc de classe P (car resolvable, en un temps concevable)
   d. Pour avoir un algorithme de meilleure complexité, on trie le tableau de valeur auparavant : $\mathcal{O}(n \log n)$

3. Problèmes des n reines
   a. La fonction booléen ligneLibre est en $\mathcal{O}(n)$

   b. Booléen colLibre(entier i, entier j)

   {

   Entier k ;

   Pour k de 1 à n

   Si(E[k][j] == k et k != i)

   Return faux

   Return vrai

   }

   c. Complexité de diagLibre : on prend le max entre les 4 parties qui contrôle un quart de diagonale chacune, ou on calcule $\mathcal{O}(n+4)$, soit $\mathcal{O}(n)$
   d. configReinesValides est de complexité $\mathcal{O}(n^3)$ car il y a 2 boucles imbriquées, et la complexité de la vérification des diagonales libres, donc $\mathcal{O}(n) * \mathcal{O}(n) * \mathcal{O}(n) = \mathcal{O}(n^3)$
   e. Les problèmes des n reines est dans la classe NP :
      Il faut un algorithme qui vérifie qu'une solution existe ou non à un temps polynomial.
      Or ici on a $\mathcal{O}(n^3)$ qui renvoie à une solution si c'est juste
      Donc problème ∈ NP

      Autre façon : choix d'un algorithme dans NP répondant à la demande
      Pour i de 1 à n
        Pour j de 1 à n
          E[i][j] = choix(0,R) → cout unitaire qui trouve toujours la bonne solution

      Donc temps en $\mathcal{O}(n^2)$, ∈ NP

4. Exo bonus
   1) $f(n) \notin \mathcal{O}(g(n))$
      $\forall\, c > 0, \forall n_0\ \exists n > n_0\ tf\ f(n) \geq c * g(n)$
   2) $g(n) \notin \mathcal{O}(f(n))$
      $\forall\, c > 0, \forall n_0\ \exists n > n_0\ tq\ g(n) \geq c * f(n)$

      $f(n) = \alpha\ g(n) \rightarrow$ il y a donc contradiction, cette possibilité n'existe pas