GROSPELLIER Antoine

927 : Exemples de preuves d'algorithmes : correction et terminaison

Ref : Cormen
+ Ex (Algo para de A. Legrand et Y. Robert)

# I Exemples fondamentaux

## ① L'exponentiation rapide :

* Récursif : exp-rec (a,n) ::  disjonction sur n :

- si n = 0 alors renvoyer 1
- si n est pair alors [ soit b = exp-rec(a, $\frac{n}{2}$) ; renvoyer a*a ]
- si n est impair alors [ soit b = exp-rec(a, $\frac{n-1}{2}$) ; renvoyer a * (b*b) ]

- Terminaison : récurrence sur n.

Plus généralement, si f : Arg → Sortie ; on exhibe φ : Arg → (E,<) où < est un ordre strict bien fondé. Mq ∀x ∈ Arg, f(x) fait appel à f(y) où φ(x) > φ(y).

- Correction : Par récurrence forte sur n : exp-rec(a,n) = $a^n$.

* itératif :

exp-it ($a_0$, $n_0$) ::  Soit a ← $a_0$ et n ← $n_0$ et res ← 1

```
tant que n ≠ 0 faire   // inv : res * a^n = a0^n0 //
    si n est pair alors n ← n/2
             sinon [ n ← (n-1)/2
                     res ← res * x
    x ← x * x
Renvoyer res
```

- Terminaison : Par variant de boucle (même principe que pour un algo récursif).
- Correction : Par invariant de boucle ($res * a^n = a_0^{n_0}$)
  - → vrai avant d'entrer dans la boucle
  - → Si c'est vrai avant une itération de boucle alors c'est vrai à la fin.
  - → Si la boucle termine (ce qui est le cas), n = 0 ⇒ res = $a_0^{n_0}$.

Rq : la complexité est O(log(n)).

## ② La multiplication de matrices

Mult-Mat (A,B) ::  Soit n = taille(A) et C une nouvelle matrice

```
Pour i = 0 à (n-1) faire   // inv-1 //
    Pour j = 0 à (n-1) faire   // inv-2 //
        Pour k = 0 à (n-1) faire   // inv-3 //
            C[i][j] ← C[i][j] + A[i][k] * B[k][j].
```

- Correction : inv1 ≡ ∀i' < i, ∀j' : C[i'][j'] = (AB)[i'][j']

inv2 ≡ (inv1) ∧ (∀j' < j : C[i][j'] = (AB)[i][j'])

inv3 ≡ (inv2) ∧ (C[i][j] = $\sum_{k'=0}^{k}$ A[i][k'] * B[k'][j])

## ③ Fonction d'Ackermann

$$ack(m,n) = \begin{cases} n+1 & \text{si } m = 0 \\ Ack(m-1,1) & \text{si } m>0 \text{ et } n=0 \\ Ack(m-1, Ack(m,n-1)) & \text{sinon} \end{cases}$$

- Terminaison : φ ≡ id et < ≡ ordre lexicographique

# II Diviser pour régner / Programmation dynamique

## ① Concepts généraux :

En général, D&R = rec et Prog Dyn = itératif.

- Terminaison : on fait appel à des sous problèmes
- Correction : Mq correction pour ss-pb ⇒ correction pour pb (propriété de sous-structure optimale ...) + cas de base.

## ② Problème du plus court chemin entre toute paire de sommets

Entrée : G = (S,A) et w : A → ℕ (matrice d'adjacence).

Sortie : D tel que D[i][j] soit le poids min. d'un chemin entre i et j.

Pour i, j et k ; soit P(i,j,k) la propriété $D^{(k)}$[i][j] est le poids min d'un chemin entre i et j passant seulement par {1, ..., k}.

②

Floyd - Warshall $(W)$ : Soit $n$ : taille$(W)$ et $D^{(0)} = W$

Pour $k = 1$ à $n$ faire // inv 1 : $\forall i', \forall j' : P(i', j', k-1)$ //
  Pour $i = 1$ à $n$ faire // inv 2 $\equiv$ inv 1 $\wedge \forall i' < i, \forall j' : P(i', j', k)$ //
    Pour $j = 1$ à $n$ faire // inv 3 $\equiv$ inv 2 $\wedge \forall j' < j, P(i, j', k)$ //
      $D^{(k)}[i-1][j-1] \leftarrow \min(D^{(k-1)}[i-1][j-1], D^{(k-1)}[i-1][k-1] + D^{(k-1)}[k-1][j-1])$

Retourner $D^{(n)}$.

Rq : Cplx : $O(s^3)$.

③ Utilisation d'outils mathématiques pour la correction :

→ Algorithme de Strassen.
→ Algorithme FFT.

## III Algorithmes gloutons

### ① Le choix d'activité

Entrée : Une liste d'activités avec leurs heures de débuts et de fins : $(d_i)_{i \in [\![1,n]\!]}$ et $(f_i)_{i \in [\![1,n]\!]}$.

Sortie : Maximiser la taille de $S \subseteq [\![1,n]\!]$ telle que $\forall i \neq j \in S : [d_i, f_i[ \cap [d_j, f_j[ = \emptyset$.

Algo : Soit $a_1 \dots a_n$ les activités triées dans l'ordre croissant des fins. Soit $A = \{a_1\}$ et $k = 1$

Pour $m = 2$ à $n$ faire // inv //
  Si $d_m \geq f_k$ alors $\begin{cases} A = A \cup \{a_m\} \\ k = m \end{cases}$
Retourner $A$.

inv : $a_k$ est le dernier élément ajouté à $A$ et $A$ est maximal pour $a_1 \dots a_{m-1}$

### ② Les Matroïdes

Def 1 : Un matroïde pondéré est $M = (E, I, w)$ où :
- $E$ est un ensemble fini.
- $\emptyset \neq I \subseteq \mathcal{P}(E)$ s'appelle l'ensemble des sous-ensembles indépendants et vérifie : * hérédité : $B \in I \wedge A \subseteq B \Rightarrow A \in I$
  * Propriété d'échange : $A, B \in I \wedge |A| < |B| \Rightarrow \exists x \in B \setminus A, A \cup \{x\} \in I$
- $w : E \to \mathbb{R}_+^*$ et on l'étend aux ensembles de $I$.

Def 2 : Un sous ensemble optimal est $F \in I$ qui maximise $w(F)$.

Glouton$(E, I, w)$ : Soit $A = \emptyset$, trier $E$ et soit $n = |E|$.
Pour $i = 0$ à $n-1$ faire // il existe $B_i \in I$ optimal tq $A_i \subseteq B_i$ et $(B_i \setminus A_i) \subseteq E[i \dots n-1]$ //
  Si $\{E[i]\} \cup A \in I$ alors $A \leftarrow A \cup \{x\}$
Retourner $A$

Théo 3 : Glouton termine et renvoie un sous-ensemble optimal.

} Dev 1

Ex 4 : Soit $G = (S, A)$. Le matroïde graphique est $M_G = (E_G, I_G)$ où : $E_G = A$ et $I_G = \{F \subseteq A \mid F \text{ est acyclique}\}$.
⇝ Calcul d'un arbre couvrant minimum par l'algorithme de Kruskal

Ex 5 : $(E, I_k)$ où $I_k = \{A \subseteq E \mid \mathrm{card}(A) \leq k\}$.

## IV Algorithmes parallèles

On se place dans le cadre PRAM, CREW avec une infinité de processeurs
- Correction : vérifier que l'on est dans le cadre CREW et que les calculs parallèles sont indépendants pour se ramener à une analyse d'algorithme non-parallèle.

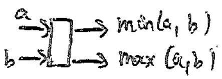
### ① Multiplication de Matrices

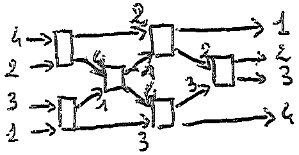
→ Algo naïf : ... faire les boucles sur $i$ et $j$ en parallèle dans l'algo de I

→ D&R : $\begin{pmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{pmatrix} \begin{pmatrix} B_{1,1} & B_{1,2} \\ B_{2,1} & B_{2,2} \end{pmatrix} = \begin{pmatrix} A_{1,1}B_{1,1} & A_{1,1}B_{1,2} \\ A_{2,1}B_{1,1} & A_{2,1}B_{1,2} \end{pmatrix} \begin{pmatrix} A_{1,2}B_{2,1} & A_{1,2}B_{2,2} \\ A_{2,2}B_{2,1} & A_{2,2}B_{2,2} \end{pmatrix}$

② Tri par transposition pair / impair

Brique de base :




## V Problème du flot maximum

Def 6 : • Un réseau de flot est un graphe $G = (S, A)$ + $s, t \in S$ + $c : A \to \mathbb{R}_+^*$ sans arcs inverses

• Un flot de $G$ est $f : A \to \mathbb{R}$ tq :

- $\forall (u,v) \in A,\ 0 \leq f(u,v) \leq c(u,v)$
- $\forall u \in S \setminus \{s,t\}$ , $\sum_{v \in S} f(u,v) = \sum_{v \in S} f(v,u)$

• La valeur du flot est $|f| = \sum_{v \in S} \big( f(s,v) - f(v,s) \big)$.

• Le problème du flot maximum est de trouver un flot qui max $|f|$

Rq 7 : On peut simuler un réseau à plusieurs source, puits et avec des arcs inverses.

Def 8 : • Pour $f$ un flot, le réseau résiduel est défini par :

$$c_f(u,v) = \begin{cases} c(u,v) - f(u,v) & \text{si } (u,v) \in A \\ f(u,v) & \text{si } (v,u) \in A \\ 0 & \text{sinon} \end{cases}$$

$G_f = (S, A_f)$ où $A_f = \{(u,v) \in S \times S \mid c_f(u,v) > 0\}$.

• Soit $f'$ un flot dans un réseau résiduel. On définit $(f \uparrow f')$ par :

$$(f \uparrow f')(u,v) = \begin{cases} f(u,v) + f'(u,v) - f'(v,u) & \text{si } (u,v) \in A \\ 0 & \text{sinon} \end{cases}$$

Prop 9 : $(f \uparrow f')$ est un flot de $G$ tq $|f \uparrow f'| = |f| + |f'|$

Coro 10 : On peut construire un flot par augmentations successives

Def 11 : Soit $p$ un chemin améliorant dans $G_f$ (chemin de $s$ à $t$)

On note $c_f(p) = \min \{ c_f(u,v) \mid (u,v) \in p \}$.

alors $f_p(u,v) = \begin{cases} c_f(p) & \text{si } (u,v) \in p \\ 0 & \text{sinon} \end{cases}$ est un flot de $G_p$.

Algo de Ford-Fulkerson : $\begin{cases} f = 0 ; \\ \text{tant qu'il existe un chemin améliorant on augmente} \\ \text{le flot.} \end{cases}$

Def 12 : • Une coupe $G$ est $(E,T)$ où $s \in E$, $t \in T$ et [illegible] $S = E \sqcup T$

• Le flot net à travers $(E,T)$ est $f(E,T) = \sum_{u \in E} \sum_{v \in T} \big[ f(u,v) - f(v,u) \big]$

• La capacité de la coupe $(E,T)$ est $c(E,T) = \sum_{u \in E} \sum_{v \in T} c(u,v)$.

Prop 13 : $f(E,T) = |f|$.

Théo 14 : Les propositions suivantes sont équivalentes :

(i) $f$ est un flot max de $G$

(ii) $G_f$ ne contient aucun chemin améliorant

(iii) $|f| = c(E,T)$ pour une certaine coupe $(E,T)$ de $G$.

Prop 15 : • L'algo de Ford-Fulkerson renvoie un flot max s'il s'arrête.

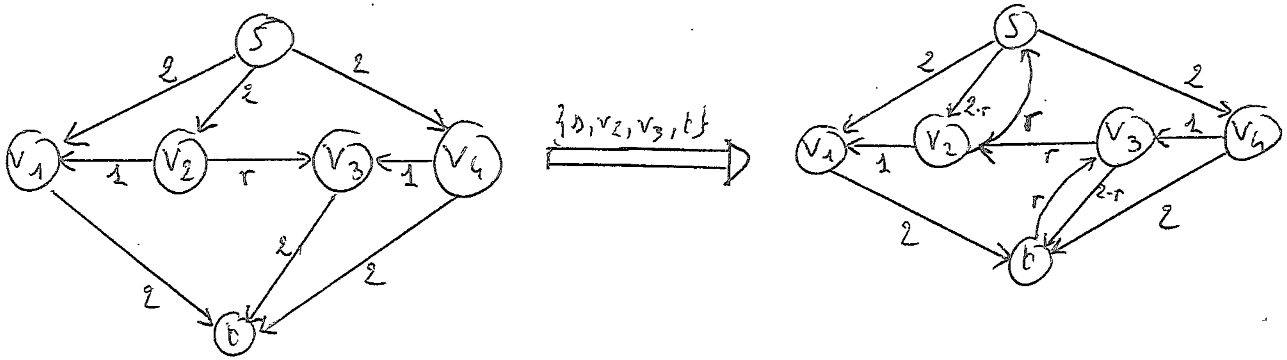
• Il s'arrête pour des capacités entières (ou rationnelles)

Ex 16 : Instance où l'algo de Ford-Fulkerson ne termine pas (annexe)

Algo 17 d'Edmonds-Karp : C'est l'algo de Ford-Fulkerson où le chemin améliorant est choisit par une recherche en largeur.

Prop 18 : L'algorithme d'Edmonds-Karp termine. ] Dev 2.

Ex16: $r = \frac{\sqrt{5}-1}{2}$ ‖ $r^2 = 1 - r$ ‖ $p_1 = \{s, v_4; v_3; v_2; v_1; t\}$ ‖ $p_2 = \{s; v_2; v_3; v_4; t\}$ ‖ $p_3 = \{s; v_1; v_2; v_3; t\}$




puis $p_1, p_2, p_1, p_3$

$p_1, p_2, p_1, p_3$

⋮