Introduction à l'algorithmique **Hiver 2018**

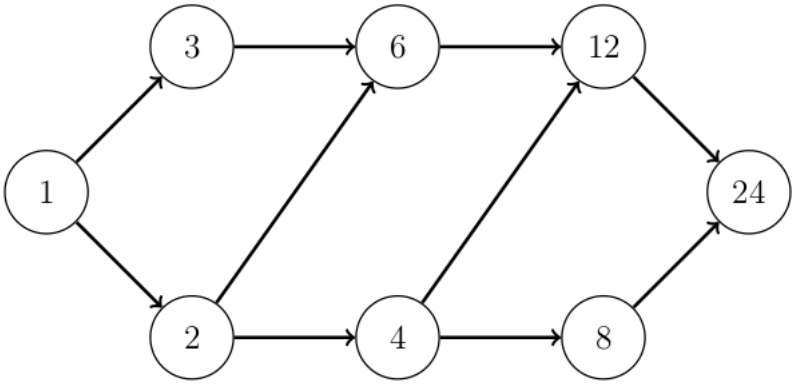
IFT2125-6001 TA: Stéphanie Larocque

Démonstration 9

À partir des corrigés de Maelle Zimmermann

1

Question: Faire le tri topologique des sommets du graphe orienté et acyclique suivant.



Solution: Voir fichier jupyter notebook/python pour le code

Il suffit d'effectuer un parcours en profondeur du graphe et de numéroter chaque sommet à la toute fin de son exploration, puis d'inverser l'ordre obtenu.

L'ordre `postnum` obtenu après le parcours en profondeur est: 24, 8, 12, 4, 6, 2, 3, 1.

Inverser cet ordre trie les sommets topologiquement: 1, 3, 2, 6, 4, 12, 8, 24.

2

Question: Calculer le nombre espéré de lancés de deux dés avant d'obtenir une paire de 1. Idem pour obtenir deux chiffres qui somment à 7.

Solution: Ce problème revient à calculer l'espérance d'une loi géométrique à paramètre p. En effet la loi géométrique considère le problème de répéter une épreuve dont la probabilité de succès est p, et la variable aléatoire géométrique dénote le rang du premier succès.

Soit p la probabilité d'obtenir l'évènement recherché lors d'un lancer. Le nombre espéré de lancers n est donné par:

$$\begin{aligned} n &= 1 \cdot p + 2(1-p)p + 3(1-p)^2 p + ... \\ &= \sum_{k=1}^{\infty} k(1-p)^{k-1} p. \end{aligned}$$

Or nous avons pour $x \in [0, 1)$:

$$\begin{aligned} f(x) &= \sum_{k=0}^{\infty} x^k = \frac{1}{1-x} \qquad \text{(série géométrique)} \\ f'(x) &= \sum_{k=1}^{\infty} k x^{k-1} = \frac{1}{(1-x)^2}. \end{aligned}$$

Comme $n = p \sum_{k=1}^{\infty} k(1-p)^{k-1}$, nous obtenons

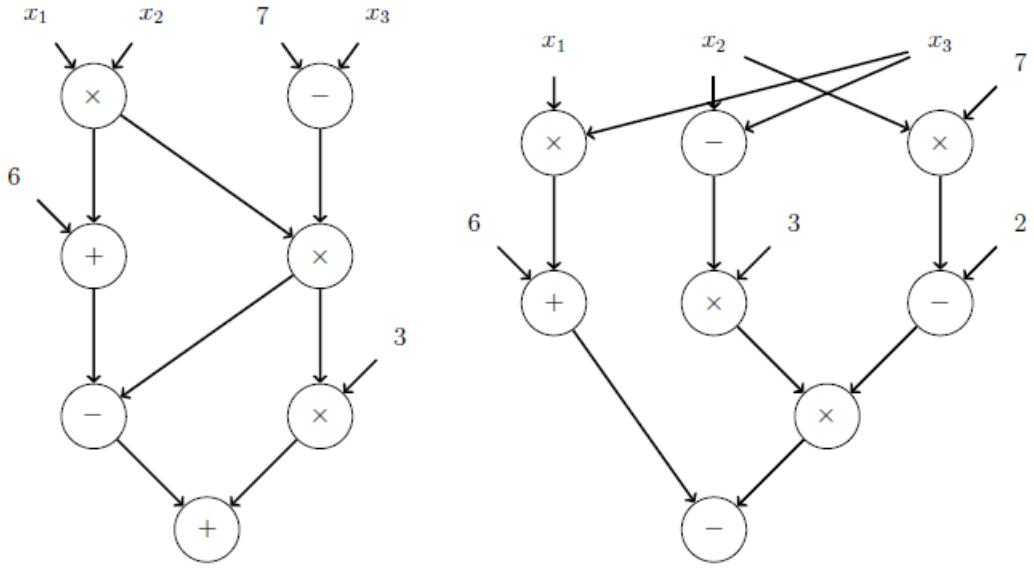
$$n = \frac{p}{(1-(1-p))^2} = \frac{1}{p}.$$

Comme la probabilité d'obtenir une paire de 1 est donnée par $p = 1/36$, le nombre espéré de lancers est 36. Similairement, la probabilité d'obtenir une paire de chiffres qui somment à 7 est $1/6$, donc le nombre espéré de lancers est 6.

3

Question: Donner un algorithme de Monte Carlo qui permet de déterminer si deux polynômes p et q sur $\mathbb{Z}$, présentés par des circuits arithmétiques, sont égaux. Nous pouvons utiliser sans preuve le fait que le degré d'un polynôme est borné par la hauteur du circuit.

Solution: Considérons l'exemple suivant:



Les circuits C_p et C_q calculent respectivement les polynômes suivants:

$$p(x_1, x_2, x_3) = ((6 + (x_1x_2)) - (x_1x_2)(7 - x_3)) + 3(x_1x_2)(7 - x_3)$$
$$q(x_1, x_2, x_3) = (6 + (x_1x_3)) - 3(x_2 - x_3)(7x_2 - 2)$$

Afin de déterminer si $p = q$, nous posons $r = p - q$ et nous testons si r est nul. Autrement dit, nous testons si le circuit $C_r = C_p - C_q$ calcule le polynôme 0. Pour ce faire, nous choisissons une affectation aléatoire de $x_1, x_2, ..., x_m$, c'est-à-dire un ensemble de nombres $a_1, a_2, ..., a_m \in \mathbb{Z}$ et nous évaluons $r(a_1, a_2, ..., a_m)$. Si $r(a_1, a_2, ..., a_m) \neq 0$, alors il est impossible que r soit le polynôme 0. En revanche si $r(a_1, a_2, ..., a_m) = 0$, il est tout de même possible que r ne soit pas le polynôme 0, si on a choisi une racine de r.

Dans le cas où $m = 1$, c'est à dire que le polynôme est univarié, il suffit de choisir a_1 dans un ensemble de taille $k \cdot \deg(r)$ pour avoir une probabilité d'erreur au plus $1/k$. En effet, r possède au plus $\deg(r)$ racines.

Dans le cas multivarié, le nombre de racines de r n'est plus limité par $\deg(r)$. En fait,

r peut posséder une infinité de racines. En effet, le polynôme $p(x_1, x_2, x_3) = x_1$ possède une infinité de racines de la forme $(0, x_2, x_3)$. Nous verrons grâce à un lemme que la stratégie du cas univarié se généralise tout de même aux polynômes multivariés.

Soient h_p et h_q la hauteur de C_p et C_q respectivement. Nous donnons l'algorithme suivant:

1. Poser $S = \{1, 2, ..., 4 \cdot \max(2^{h_p}, 2^{h_q})\}$
2. Choisir aléatoirement et indépendamment $a_1, a_2, ..., a_m \in \text{uniforme}(S)$
3. Poser $r = p - q$
4. Si $r(a_1, a_2, ..., a_m) = 0$ alors retourner oui, sinon retourner non.

Lorsque l'algorithme répond non, nous avons la certitude que p et q ne sont pas égaux. Lorsque l'algorithme répond oui, il est possible que p et q ne soient pas égaux et que l'algorithme se trompe. Pour évaluer la probabilité que l'algorithme se trompe dans le cas multivarié, nous avons besoin du lemme suivant:

Lemme de Schwartz-Zippel: soit $p \in \mathbb{Q}[x_1, x_2, ..., x_m]$ un polynôme non nul. Soient $S \subset \mathbb{Q}$ un sous-ensemble fini et $a_1, a_2, ..., a_m$ choisis dans S de façon uniforme et indépendante. Alors,

$$\Pr[p(a_1, a_2, ..., a_m) = 0] \leq \frac{\deg(p)}{|S|}. \tag{1}$$

Preuve: Le lemme est démontré par induction sur le nombre de variable.

Cas de base (m=1): dans ce cas, p est univarié et possède au plus $\deg(p)$ racines, donc on obtient que

$$\Pr[p(a_1) = 0] \leq \frac{\deg(p)}{|S|}.$$

Etape d'induction: nous pouvons mettre chaque puissance de x_1 en évidence dans p et obtenir la réécriture suivante.

$$p(x_1, x_2, ...x_m) = \sum_{i=0}^{d} x_1^i p_i(x_2, ...x_m)$$

Puisque p est non nul, il existe au moins un i tel que p_i est non nul. Soit i le plus grand tel indice, alors par hypothèse d'induction:

$$\Pr[p_i(a_2, ...a_m) = 0] \leq \frac{\deg(p_i)}{|S|} \leq \frac{\deg(p) - i}{|S|} \tag{2}$$

Si $p_i(a_2, ...a_m) \neq 0$, alors $p(x_1, a_2, ..., a_m)$ est un polynôme à une variable (x_1) de degré i, car i est le plus grand degré de x_1 non nul dans p. Ainsi par hypothèse d'induction:

$$\Pr[p(a_1, a_2, ..., a_m) = 0 \mid p_i(a_2, ...a_m) \neq 0] \leq \frac{i}{S} \tag{3}$$

Si on appelle A l'évènement $p(a_1, a_2, ..., a_m) = 0$ et B l'évènement $p_i(a_2, ...a_m) = 0$, alors on a

$$\begin{aligned}
\Pr[A] &= \Pr[A|B]\Pr[B] + \Pr[A|B^c]\Pr[B^c] \\
&\leq \Pr[B] + \Pr[A|B^c] \\
&\leq \frac{\deg(p) - i}{|S|} + \frac{i}{|S|} \quad \text{par (2) et (3)} \\
&= \frac{\deg(p)}{|S|}.
\end{aligned}$$

Ce qui conclut la preuve. □

Considérons maintenant le cas où l'algorithme ci-haut réponde oui. Soit w la probabilité que l'algorithme se trompe, autrement dit que $r(a_1, a_2, ...a_m) = 0$ sachant que r est non nul. Nous avons:

$$\begin{aligned}
w &\leq \frac{\deg(r)}{|S|} && \text{par (1)} \\
&\leq \frac{\max(\deg(p), \deg(q))}{|S|} && \text{car } \deg(r) \leq \max(\deg(p), \deg(q)) \\
&= \frac{\max(\deg(p), \deg(q))}{4 \cdot \max(2^{h_p}, 2^{h_q})} && \text{par déf. de } S \\
&\leq \frac{\max(\deg(p), \deg(q))}{4 \cdot \max(\deg(p), \deg(q))} && \text{car } \deg(p) \leq 2^{h_p} \text{ et } \deg(q) \leq 2^{h_q} \\
&= \frac{1}{4}
\end{aligned}$$

Ainsi l'algorithme se trompe avec probabilité au plus $\frac{1}{4}$ lorsqu'il répond oui.

Analyse du temps d'exécution: Nous analysons le temps d'exécution de l'algorithme selon la taille de l'entrée, i.e. selon le degré de p et q et selon le nombre de variables m. Si on peut tirer à pile ou face en $O(1)$, il est possible de sélectionner un nombre a_i dans S par recherche binaire en temps $\log_2 |S| = \log_2(4 \cdot \max(2^{h_p}, 2^{h_q})) = 2 + \max(h_p, h_q)$ tirs à pile ou face. Le temps d'exécution des algorithme de sélection aléatoire est donc dans $O(m \cdot \max(h_p, h_q))$. Evaluer le circuit r se fait en temps polynômial. Le temps d'exécution de l'algorithme est donc polynomial dans la taille de l'entrée.

4

Question: Donner un algorithme qui calcule la distance d'édition d, aussi appelée distance de Levenshtein, entre deux mots. Plus précisément, soient u et v deux mots, alors $d(u, v)$ est le nombre minimal de lettres qu'il faut supprimer, insérer ou substituer pour passer de u à v.

Par exemple, on peut passer de *table* à *stage* en 3 opérations:

$$\begin{aligned} table &\rightarrow tale && \text{(supprimer } b\text{)} \\ &\rightarrow stale && \text{(insérer } s\text{)} \\ &\rightarrow stage && \text{(substituer } l \text{ pour } g\text{)} \end{aligned}$$

Solution: Nous construisons un tableau D de taille $|u| + 1 \times |v| + 1$ tel que $D[i][j]$ est la distance entre les sous-mots $u[1...i]$ et $v[1...j]$.

Notons d'abord que $D[i][0] = i$ pour tout $0 \leq i \leq |u|$, et $D[0][j] = j$ pour tout $0 \leq j \leq |v|$. En effet, pour un mot vide ϵ, on a $d(x, \epsilon) = d(\epsilon, x) = |x|$ car l'édition minimale consiste dans le premier cas à effacer toutes les lettres de x ($|x|$ suppressions) et dans le deuxième cas à ajouter toutes les lettres de x au mot vide ($|x|$ insertions). Plus généralement, pour passer d'un mot $u[1...i]$ à un mot $v[1...j]$, il y a trois possibilités:

- Supprimer $u[i]$ puis passer de $u[1...i-1]$ à $v[1...j]$
- Passer de $u[1...i]$ à $v[1...j-1]$ puis insérer $v[j]$
- Passer de $u[1...i-1]$ à $v[1...j-1]$ puis substituer $u[i]$ pour $v[j]$ (si ces lettres sont différentes)

Cela donne la règle:

$$D[i][j] = \min(\underbrace{D[i-1][j]+1}_{\text{suppression}}, \underbrace{D[i][j-1]+1}_{\text{insertion}}, \underbrace{D[i-1][j-1]+b_{i,j}}_{\text{substitution}})$$

où $b_{i,j} = 0$ si $u[i] = v[j]$ et 1 sinon.

Voici une implémentation de l'algorithme, qui prend un temps dans $\Theta(|u||v|)$:

```
def d_tab(u, v):
    D = [[0]*(len(v) + 1) for _ in range(len(u) + 1)]

    for i in range(len(u)+1):
        D[i][0] = i

    for j in range(len(v)+1):
        D[0][j] = j

    for i in range(1, len(u) + 1):
        for j in range(1, len(v) + 1):
            b = 1 if u[i] != v[j] else 0
            D[i][j] = min(D[i-1][j] + 1, D[i][j-1] + 1, D[i-1][j-1] + b)
    return D

def d(u, v):
    return d_tab(u, v)[-1][-1]
```

5

Question: Soient $u = AGGAGGA$ et $v = AAGCTAAG$. Identifier tous les alignements entre u et v de coût $d(u, v)$, c'est-à-dire tous les alignements optimaux.

Solution: Le tableau des distances est:

	ϵ	A	A	G	C	T	A	A	G
ϵ	0	1	2	3	4	5	6	7	8
A	1	0	1	2	3	4	5	6	7
G	2	1	1	1	2	3	4	5	6
G	3	2	3	1	2	3	4	5	5
A	4	3	2	2	2	3	3	4	5
G	5	4	3	2	3	3	4	4	4
G	6	5	4	3	3	4	4	5	4
A	7	6	5	4	4	4	4	4	5

Ainsi la distance de Levenshtein entre les deux mots est de 5. pour trouver quel alignement et donc quelle séquence d'opérations donne ce coût, il faut remonter le tableau depuis la case `T[-1][-1]` en suivant les règles de l'algorithme.

	ε	A	A	G	C	T	A	A	G
ε	0	1	2	3	4	5	6	7	8
A	1	0	1	2	3	4	5	6	7
G	2	1	1	1	2	3	4	5	6
G	3	2	2	1	2	3	4	5	5
A	4	3	2	2	2	3	3	4	5
G	5	4	3	2	3	3	4	4	4
G	6	5	4	3	3	4	4	5	4
A	7	6	5	4	4	4	4	4	5

Le chemin du haut correspond à l'alignement de séquence suivant:

A A G C T A A G -
- A G - G A G G A

ce qui correspond à faire deux insertions (A et C), deux substitutions (G pour T et G pour A) et une suppression (A). En tout nous obtenons 6 chemins, et les 5 autres correspondent aux alignements de séquences suivants:

A A G C T A A G
A G G A G G A -

A A G C T A A G -
A G G - - A G G A

A A G C T A A G -
A - G - G A G G A

A A G C T A A G -
- A G G - A G G A

A A G C T A A G -
A - G G - A G G A