TD5 : Algorithmes probabilistes

1. Classes de complexité

1.1. Montrer que $\mathbf{P} \subset \mathbf{ZPP}$.

Un algorithme déterministe est un algorithme probabiliste exact qui n'utilise aucun choix probabiliste. De plus, vu comme algo probabiliste, l'espérance de son temps d'exécution coïncide avec le temps d'exécution, et est donc borné par un polynôme.

1.2. Montrer que $\mathbf{RP} \subset \mathbf{NP}$.

Pour transformer $\mathcal{A}$ algorithme probabiliste en $\mathcal{B}$ algorithme non-déterministe, il suffit de remplacer tous les choix probabilistes par des choix non-déterministes. Alors, si $\mathcal{A}$ est un algo **RP**, $\mathcal{B}$ est un algo **NP**. En effet, si $x \in L$, la probabilité que $\mathcal{A}$ réponde oui est au moins $\frac{1}{2}$. Il existe donc une exécution acceptante de $\mathcal{B}$ sur x.

1.3. Montrer que $\mathbf{RP} \cap \mathbf{co\text{-}RP} \subseteq \mathbf{ZPP}$.

Soit L un problème de décision dans $\mathbf{RP} \cap \mathbf{co\text{-}RP}$. Il existe donc un polynôme p et deux algorithmes $\mathcal{A}$ et $\mathcal{B}$ dont le temps d'exécution est borné par $p(n)$ et tels que

- $\mathcal{A}$ donne la bonne réponse si $x \notin L$, et renvoie « $\in L$ »avec probabilité au moins $\frac{1}{2}$ si $x \in L$;
- $\mathcal{B}$ donne la bonne réponse si $x \in L$, et renvoie « $\notin L$ »avec probabilité au moins $\frac{1}{2}$ si $x \notin L$.

Définissons l'algorithme suivant :

```
ZPP-algo(x)
    while true do
        if A retourne « ∈ L »sur x then return « ∈ L »
        if B retourne « ∉ L »sur x then return « ∉ L »
    endWhile
```

Par définition, l'algorithme `ZPP-algo` renvoie toujours une réponse correcte. Calculons l'espérance du temps d'exécution de `ZPP-algo` sur l'entrée x, notée $T(x)$. Sans perte de généralité, on suppose $x \in L$, l'autre cas étant symétrique.

On peut facilement montrer par récurrence que la probabilité que `ZPP-algo` ne termine pas en k tours de boucle est bornée par $\frac{1}{2^k}$. La probabilité d'effectuer au moins k itérations est donc bornée par $\frac{1}{2^{k-1}}$. Le temps d'exécution de k tours de boucle successifs, est lui borné par $2kp(|x|)$. On en déduit

$$\begin{aligned}\mathbb{E}(T(x)) &= \sum_{k>0} \mathbb{P}\left(\texttt{ZPP-algo} \text{ prend } k \text{ itérations }\right) \times \text{ Temps } (k \text{ itérations}) \\ &\leq \sum_{k>0} \frac{1}{2^{k-1}} \cdot 2kp(|x|) \\ &= 4p(|x|) \sum_{k>0} \frac{k}{2^k} \ .\end{aligned}$$

La somme converge, donc on peut conclure directement. Cependant, à titre d'exercice, on peut calculer cette somme en utilisant des séries génératrices. Posons $G(z) = \sum_{k\geq 0} \frac{z^k}{2^k}$. Alors $G'(1) = \sum_{k>0} \frac{k}{2^k}$, est un majorant de $\mathbb{E}(T(x))$. Or $G(z) = \frac{1}{1-\frac{z}{2}} = \frac{2}{2-z}$, d'où $G'(z) = \frac{2}{(2-z)^2}$, et $G'(1) = 2$. On en déduit que

$$\mathbb{E}(T(x)) \leq 8p(|x|) \ .$$

Ainsi, `ZPP-algo` est un algorithme probabiliste exact, de temps moyen d'exécution borné par un polynôme. Le problème L initial appartient donc à la classe **ZPP**.

Pour l'inclusion inverse, on rappelle l'inégalité de Markov : pour X une variable aléatoire réelle positive et $t \in \mathbb{R}_+^*$

$$\mathbb{P}(X \geq t) \leq \frac{\mathbb{E}[X]}{t} \ .$$

Montrer que **ZPP** $\subseteq$ **RP** $\cap$ **co-RP**.

Soit donc $L \in$ **ZPP**, et $\mathcal{A}$ un algorithme probabiliste exact pour L de temps moyen d'exécution T borné par $p(n)$. Définissons l'algorithme suivant :

```
RP-algo(x)
    Exécuter A sur x pendant 2p(|x|) étapes
    if A s'arrête et renvoie R, then return R
    else return « ∉ L »
```

Par définition, le temps d'exécution de l'algorithme probabiliste `RP-algo` est toujours borné par un polynôme (ici $2p$). Montrons que `RP-algo` satisfait bien les propriétés d'un algorithme **RP**.
Supposons $x \notin L$. Alors `RP-algo` retourne nécessairement « $\notin L$ ».
Supposons maintenant $x \in L$. On va utiliser l'inégalité de Markov : pour X une variable aléatoire réelle positive et $t \in \mathbb{R}_+^*$

$$\mathbb{P}(X \geq t) \leq \frac{\mathbb{E}[X]}{t} \ .$$

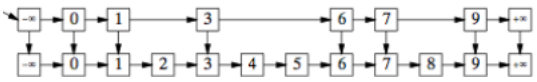
Appliquons l'inégalité de Markov à la variable aléatoire $T(x)$ et à $t = 2p(|x|)$:

$$\mathbb{P}\left(T(x) \geq 2p(|x|)\right) \leq \frac{\mathbb{E}(T(x))}{2p(|x|)} \ .$$

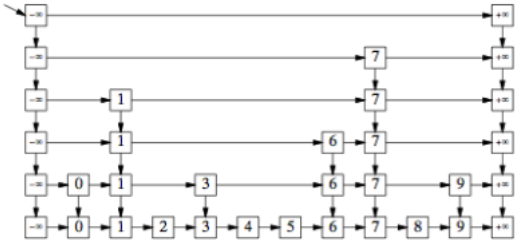
Or, par hypothèse sur l'algorithme $\mathcal{A}$, $\mathbb{E}(T(x)) \leq p(|x|)$. D'où, $\mathbb{P}\left(T(x) \geq 2p(|x|)\right) \leq \frac{1}{2}$, et donc $\mathbb{P}\left(T(x) < 2p(|x|)\right) \geq \frac{1}{2}$. Ainsi, si $x \in L$, l'algorithme `RP-algo` retourne sur l'entrée x « $\in L$ » avec probabilité au moins $\frac{1}{2}$. On a donc prouvé que $L \in$ **RP**.
De façon analogue, on peut montrer que $L \in$ **co-RP**.

2. Skip lists

Une liste à sauts (*skip list*) SL est une liste chaînée triée comportant quelques raccourcis. Pour accélérer la recherche, on crée une deuxième liste triée construite en dupliquant chaque élément de la liste de départ avec probabilité 1/2. On rajoute un pointeur de chaque doublon vers l'original.



On utilise cette idée récursivement, ce qui donne par exemple la structure suivante.



En moyenne, il y a $n/2$ éléments dans la liste du haut. Le nombre de comparaisons en bas dépend du nombre d'éléments consécutifs n'ayant pas été dupliquées dans la liste du haut. La probabilité qu'il y ait k éléments non dupliqués à la suite est de $1/2^k$, donc le nombre de comparaisons en bas est borné par $1 + \Sigma 1/2^k$.

2.1. Écrire l'algorithme de recherche d'un élément.

On dispose de deux fonctions *right* et *down* :

$v \leftarrow L$
while $L \neq Null$ and $cle(v) \neq x$ **do**
  if $cle(right(v)) > x$ **then**
    $v \leftarrow down(v)$
  else
    $v \leftarrow right(v)$
  end if
end while
Return v

Supposons que les clés soient les entiers de 1 à n. Soit $L(x)$ le nombre de niveaux qui contiennent la clé x (sans compter la liste en bas).

2.2. Soit x une clé dans une skip list SL. Calculer l'espérance $\mathbb{E}_x[L(x)]$ du nombre de niveaux pour x.

$L(x) + 1$ suit une loi géométrique de paramètre $1/2$ (on ne compte pas le niveau du bas), donc $\mathbb{E}[L(x)] = \frac{1}{\frac{1}{2}} - 1 = 1$.

Pour estimer le pire cas moyen, on a besoin d'une borne sur $L = max_x L(x)$.

2.3. Montrer que la hauteur de la skip list est $O(log(n))$ avec probabilité supérieure à $1 - 1/n^{c-1}$ pour tout $c > 1$. Indication : calculer $\mathbb{P}(L \geq c \log(n))$.

La hauteur $> l$ ssi il existe une clé dupliquée au moins l fois, ce qui arrive avec une probabilité de $1/2^l$. La probabilité pour n clés est donc $\leq n/2^l$.
Si on pose $l = c \log(n)$, on a $\mathbb{P}(L \geq c \log(n)) \leq \frac{n}{2^{c \log n}} = 1/n^{c-1}$

2.4. Montrer la borne suivante sur l'espérance de la taille de la skip list : $\mathbb{E}[L] \leq \log(n)+1$. En déduire l'espérance de la complexité dans le pire des cas de recherche d'un élément.

$\mathbb{E}[L] = \Sigma_{l \geq 0} l \mathbb{P}(L = l) = \Sigma_{l \geq 1} \mathbb{P}(L \geq l)$
$\mathbb{E}[L] = \Sigma_1^{\log n} \mathbb{P}(L \geq l) + \Sigma_{l > \log n} \mathbb{P}(L \geq l)$
$\mathbb{E}[L] \leq \log n + \Sigma_{l > \log n} n/2^l = \log n + \Sigma_{l > \log n} 1/2^{l - \log n} = \log n + \Sigma_{l > 0} 1/2^l$
$\mathbb{E}[L] \leq \log n + 1$
Le coût de recherche va être en $O(\log n)$.