Algorithmique Probabiliste

Philippe Duchon

LaBRI - ENSEIRB-Matméca - Université de Bordeaux

2014-15

Le problème de Sélection

- **Entrée :** une liste $L = (x_1 \ldots, x_n)$ de nombres, un entier $k \leq n$
- **Problème :** trouver le k-ème plus petit des x_i ($k = 1$: le minimum ; $k = n$: le maximum ; $k = \lfloor n/2 \rfloor$: le médian)

Le problème de Sélection

- **Entrée :** une liste $L = (x_1 \ldots, x_n)$ de nombres, un entier $k \leq n$
- **Problème :** trouver le k-ème plus petit des x_i ($k = 1$: le minimum ; $k = n$: le maximum ; $k = \lfloor n/2 \rfloor$: le médian)
- Ce n'est pas un problème très dur : on peut toujours trier L en temps $O(n \log n)$, puis retourner le k-ème élément de la liste triée

Le problème de Sélection

- **Entrée :** une liste $L = (x_1 \ldots, x_n)$ de nombres, un entier $k \leq n$
- **Problème :** trouver le k-ème plus petit des x_i ($k = 1$: le minimum ; $k = n$: le maximum ; $k = \lfloor n/2 \rfloor$: le médian)
- Ce n'est pas un problème très dur : on peut toujours trier L en temps $O(n \log n)$, puis retourner le k-ème élément de la liste triée
- Inversement, pour n'importe quel k, même si on nous "donne gratuitement" la réponse il faut au moins $n - 1$ comparaisons pour la vérifier, donc $\Omega(n)$

Solutions déterministes

- Il existe des algorithmes déterministes, en $O(n)$ comparaisons

Solutions déterministes

- Il existe des algorithmes déterministes, en $O(n)$ comparaisons
- Les meilleurs connus (assez complexes) atteignent $2.95n + o(n)$

Solutions déterministes

- Il existe des algorithmes déterministes, en $O(n)$ comparaisons
- Les meilleurs connus (assez complexes) atteignent $2.95n + o(n)$
- Il existe un **théorème de borne inférieure (Dor et Zwick, 1996) :** il existe un certain $\epsilon > 0$ tel qu'aucun algorithme déterministe ne trouve (dans le cas le pire !) le médian d'une liste en moins de $(2 + \epsilon)n$ comparaisons.

Solutions déterministes

- Il existe des algorithmes déterministes, en $O(n)$ comparaisons
- Les meilleurs connus (assez complexes) atteignent $2.95n + o(n)$
- Il existe un **théorème de borne inférieure (Dor et Zwick, 1996) :** il existe un certain $\epsilon > 0$ tel qu'aucun algorithme déterministe ne trouve (dans le cas le pire !) le médian d'une liste en moins de $(2 + \epsilon)n$ comparaisons.
- C'est un cas rare où on est capable d'exhiber un algorithme probabiliste pour lequel on prouve que **tout** algorithme déterministe fait **strictement moins bien** (**LazySelect** : $2n + o(n)$ en moyenne et avec probabilité proche de 1)

Une solution déterministe linéaire

- Algorithme diviser-pour-régner pour la sélection :
  - On découpe la liste en $n/7$ sous-listes de taille 7

Une solution déterministe linéaire

- Algorithme diviser-pour-régner pour la sélection :
  - On découpe la liste en $n/7$ sous-listes de taille 7
  - On trie chaque sous-liste, p. ex. en 13 comparaisons (pas super facile, mais faisable ; 12 impossible car $7! > 2^{12}$)

Une solution déterministe linéaire

- Algorithme diviser-pour-régner pour la sélection :
  - On découpe la liste en $n/7$ sous-listes de taille 7
  - On trie chaque sous-liste, p. ex. en 13 comparaisons (pas super facile, mais faisable ; 12 impossible car $7! > 2^{12}$)
  - Récursivement, on trouve le médian M des $n/7$ médians de sous-listes

Une solution déterministe linéaire

- Algorithme diviser-pour-régner pour la sélection :
  - On découpe la liste en $n/7$ sous-listes de taille 7
  - On trie chaque sous-liste, p. ex. en 13 comparaisons (pas super facile, mais faisable ; 12 impossible car $7! > 2^{12}$)
  - Récursivement, on trouve le médian M des $n/7$ médians de sous-listes
  - On a déjà $2n/7$ éléments connus pour être plus petits que M, et $2n/7$ connus pour être plus grands que M ; en $3n/7$ comparaisons supplémentaires, on trouve le rang de M et on partitionne la liste de départ en l'ensemble des plus grands et des plus petits que M

Une solution déterministe linéaire

- Algorithme diviser-pour-régner pour la sélection :
  - On découpe la liste en $n/7$ sous-listes de taille 7
  - On trie chaque sous-liste, p. ex. en 13 comparaisons (pas super facile, mais faisable ; 12 impossible car $7! > 2^{12}$)
  - Récursivement, on trouve le médian M des $n/7$ médians de sous-listes
  - On a déjà $2n/7$ éléments connus pour être plus petits que M, et $2n/7$ connus pour être plus grands que M ; en $3n/7$ comparaisons supplémentaires, on trouve le rang de M et on partitionne la liste de départ en l'ensemble des plus grands et des plus petits que M
  - Récursivement, on se ramène à une sélection dans un ensemble de taille au plus $5n/7$

Une solution déterministe linéaire

- Algorithme diviser-pour-régner pour la sélection :
  - On découpe la liste en $n/7$ sous-listes de taille 7
  - On trie chaque sous-liste, p. ex. en 13 comparaisons (pas super facile, mais faisable ; 12 impossible car $7! > 2^{12}$)
  - Récursivement, on trouve le médian M des $n/7$ médians de sous-listes
  - On a déjà $2n/7$ éléments connus pour être plus petits que M, et $2n/7$ connus pour être plus grands que M ; en $3n/7$ comparaisons supplémentaires, on trouve le rang de M et on partitionne la liste de départ en l'ensemble des plus grands et des plus petits que M
  - Récursivement, on se ramène à une sélection dans un ensemble de taille au plus $5n/7$
- Récurrence sur la complexité dans le cas le pire :

$$C_n \leq \frac{13n}{7} + \frac{3n}{7} + C_{n/7} + C_{5n/7}$$

Une solution déterministe linéaire

- Algorithme diviser-pour-régner pour la sélection :
  - On découpe la liste en $n/7$ sous-listes de taille 7
  - On trie chaque sous-liste, p. ex. en 13 comparaisons (pas super facile, mais faisable ; 12 impossible car $7! > 2^{12}$)
  - Récursivement, on trouve le médian M des $n/7$ médians de sous-listes
  - On a déjà $2n/7$ éléments connus pour être plus petits que M, et $2n/7$ connus pour être plus grands que M ; en $3n/7$ comparaisons supplémentaires, on trouve le rang de M et on partitionne la liste de départ en l'ensemble des plus grands et des plus petits que M
  - Récursivement, on se ramène à une sélection dans un ensemble de taille au plus $5n/7$
- Récurrence sur la complexité dans le cas le pire :

$$C_n \leq \frac{13n}{7} + \frac{3n}{7} + C_{n/7} + C_{5n/7}$$

- On vérifie que la solution est $O(n)$ (ici, la constante est mauvaise !)

Solutions probabilistes de faible complexité

- **RandQuickSelect** : une solution inspirée et adaptée de **RandQuickSort** ; $(2 + 2\ln(2))n + o(n)$ en moyenne, pour $k = n/2$

Solutions probabilistes de faible complexité

- **RandQuickSelect** : une solution inspirée et adaptée de **RandQuickSort** ; $(2 + 2\ln(2))n + o(n)$ en moyenne, pour $k = n/2$
- **LazySelect** : algorithme basé sur le principe de l'échantillonnage ; $2n + o(n)$ en moyenne et avec probabilité asymptotique 1

```
RandQuickSelect(L,k,a,b):
  i=Uniforme(a,b)
  j=Partitionne(L,a,b,i)
  Si j=k:
    Retourner L[i]
  Si j<k:
    Retourner RandQuickSelect(L,k,a,j-1)
  Si j>k:
    Retourner RandQuickSelect(L,k,j+1,b)

Partitionne(L,d,f,p):
  [Partitionne les elements de L d'indice ]
  [entre d et f par rapport au pivot initialement ]
  [en position p]
```

Analyse de la complexité moyenne de **RandQuickSelect**

- $X_{i,j} = 1$ si la i-ème et la j-ème clés (dans l'ordre trié) sont comparées, 0 sinon

Analyse de la complexité moyenne de **RandQuickSelect**

- $X_{i,j} = 1$ si la i-ème et la j-ème clés (dans l'ordre trié) sont comparées, 0 sinon
- $C = \sum_{i<j} X_{i,j}$

Analyse de la complexité moyenne de **RandQuickSelect**

- $X_{i,j} = 1$ si la i-ème et la j-ème clés (dans l'ordre trié) sont comparées, 0 sinon
- $C = \sum_{i<j} X_{i,j}$
- $\mathbf{E}(C) = \sum_{i<j} p_{i,j}$, où $p_{i,j}$ désigne la probabilité de comparer les i-ème et j-ème clés

Analyse de la complexité moyenne de **RandQuickSelect**

- $X_{i,j} = 1$ si la i-ème et la j-ème clés (dans l'ordre trié) sont comparées, 0 sinon
- $C = \sum_{i<j} X_{i,j}$
- $\mathbf{E}(C) = \sum_{i<j} p_{i,j}$, où $p_{i,j}$ désigne la probabilité de comparer les i-ème et j-ème clés
- $p_{i,j} = \ldots$

Analyse de la complexité moyenne de **RandQuickSelect**

- $X_{i,j} = 1$ si la i-ème et la j-ème clés (dans l'ordre trié) sont comparées, 0 sinon
- $C = \sum_{i<j} X_{i,j}$
- $\mathbf{E}(C) = \sum_{i<j} p_{i,j}$, où $p_{i,j}$ désigne la probabilité de comparer les i-ème et j-ème clés
- $p_{i,j} = \ldots$
  - On compare les deux clés si l'une des deux est prise comme pivot, dans un tableau qui contient encore la k-ème clé

Analyse de la complexité moyenne de **RandQuickSelect**

- $X_{i,j} = 1$ si la i-ème et la j-ème clés (dans l'ordre trié) sont comparées, 0 sinon
- $C = \sum_{i<j} X_{i,j}$
- $\mathbf{E}(C) = \sum_{i<j} p_{i,j}$, où $p_{i,j}$ désigne la probabilité de comparer les i-ème et j-ème clés
- $p_{i,j} = \ldots$
  - On compare les deux clés si l'une des deux est prise comme pivot, dans un tableau qui contient encore la k-ème clé
  - Si on prend un pivot entre i et j, "c'est mort"

Analyse de la complexité moyenne de **RandQuickSelect**

- $X_{i,j} = 1$ si la i-ème et la j-ème clés (dans l'ordre trié) sont comparées, 0 sinon
- $C = \sum_{i<j} X_{i,j}$
- $\mathbf{E}(C) = \sum_{i<j} p_{i,j}$, où $p_{i,j}$ désigne la probabilité de comparer les i-ème et j-ème clés
- $p_{i,j} = \ldots$
  - On compare les deux clés si l'une des deux est prise comme pivot, dans un tableau qui contient encore la k-ème clé
  - Si on prend un pivot entre i et j, "c'est mort"
  - **(différence avec RandQuickSort)** Si on prend un pivot qui sépare les deux clés i et j de la k-ème, c'est mort aussi ($i < j < p < k$ par exemple)

Analyse de la complexité moyenne de **RandQuickSelect**

- $X_{i,j} = 1$ si la i-ème et la j-ème clés (dans l'ordre trié) sont comparées, 0 sinon
- $C = \sum_{i<j} X_{i,j}$
- $\mathbf{E}(C) = \sum_{i<j} p_{i,j}$, où $p_{i,j}$ désigne la probabilité de comparer les i-ème et j-ème clés
- $p_{i,j} = \ldots$
  - On compare les deux clés si l'une des deux est prise comme pivot, dans un tableau qui contient encore la k-ème clé
  - Si on prend un pivot entre i et j, "c'est mort"
  - **(différence avec RandQuickSort)** Si on prend un pivot qui sépare les deux clés i et j de la k-ème, c'est mort aussi ($i < j < p < k$ par exemple)
  - Au total, les deux clés sont comparées si on prend l'une des deux comme premier pivot dans l'intervalle $[\min(k, i), \max(k, j)]$

Analyse de la complexité moyenne de **RandQuickSelect**

- $X_{i,j} = 1$ si la i-ème et la j-ème clés (dans l'ordre trié) sont comparées, 0 sinon
- $C = \sum_{i<j} X_{i,j}$
- $\mathbf{E}(C) = \sum_{i<j} p_{i,j}$, où $p_{i,j}$ désigne la probabilité de comparer les i-ème et j-ème clés
- $p_{i,j} = \ldots$
  - On compare les deux clés si l'une des deux est prise comme pivot, dans un tableau qui contient encore la k-ème clé
  - Si on prend un pivot entre i et j, "c'est mort"
  - **(différence avec RandQuickSort)** Si on prend un pivot qui sépare les deux clés i et j de la k-ème, c'est mort aussi ($i < j < p < k$ par exemple)
  - Au total, les deux clés sont comparées si on prend l'une des deux comme premier pivot dans l'intervalle $[\min(k,i), \max(k,j)]$
  - $p_{i,j} = \frac{2}{1+\max(k,j)-min(k,i)}$

Analyse de la complexité moyenne de **RandQuickSelect**

- $X_{i,j} = 1$ si la i-ème et la j-ème clés (dans l'ordre trié) sont comparées, 0 sinon
- $C = \sum_{i<j} X_{i,j}$
- $\mathbf{E}(C) = \sum_{i<j} p_{i,j}$, où $p_{i,j}$ désigne la probabilité de comparer les i-ème et j-ème clés
- $p_{i,j} = \ldots$
  - On compare les deux clés si l'une des deux est prise comme pivot, dans un tableau qui contient encore la k-ème clé
  - Si on prend un pivot entre i et j, "c'est mort"
  - **(différence avec RandQuickSort)** Si on prend un pivot qui sépare les deux clés i et j de la k-ème, c'est mort aussi ($i < j < p < k$ par exemple)
  - Au total, les deux clés sont comparées si on prend l'une des deux comme premier pivot dans l'intervalle $[\min(k,i), \max(k,j)]$
  - $p_{i,j} = \frac{2}{1+\max(k,j)-min(k,i)}$
  - est-ce que la différence est assez importante pour passer de $n\log(n)$ à $O(n)$?

LazySelect : idée de l'échantillonnage

- Calculer le rang d'un élément dans un ensemble de taille m, ça coûte $m - 1$ comparaisons

LazySelect : idée de l'échantillonnage

- Calculer le rang d'un élément dans un ensemble de taille m, ça coûte $m - 1$ comparaisons
- Si un ensemble est "petit" (par exemple, n^α avec $\alpha < 1$), on peut le trier en temps $o(n)$

LazySelect : idée de l'échantillonnage

- Calculer le rang d'un élément dans un ensemble de taille m, ça coûte $m - 1$ comparaisons
- Si un ensemble est "petit" (par exemple, n^α avec $\alpha < 1$), on peut le trier en temps $o(n)$
- Si on est capable d'identifier un "petit" ensemble qui contient "presque sûrement" l'élément qu'on cherche, on a le droit de trier intégralement cet ensemble

LazySelect : idée de l'échantillonnage

- Calculer le rang d'un élément dans un ensemble de taille m, ça coûte $m - 1$ comparaisons
- Si un ensemble est "petit" (par exemple, n^α avec $\alpha < 1$), on peut le trier en temps $o(n)$
- Si on est capable d'identifier un "petit" ensemble qui contient "presque sûrement" l'élément qu'on cherche, on a le droit de trier intégralement cet ensemble
- Si on choisit des éléments aléatoires de la liste, pour peu qu'on en prenne "assez", leurs rangs devraient être répartis assez uniformément

L'algorithme

- $m = n^{3/4}$, $\ell = k/n^{1/4}$, $x = \ell - \sqrt{n}$, $y = \ell + \sqrt{n}$
- Tirer m éléments aléatoires uniformes de L pour former S
- Trier S
- (Si $x > 0$) Prendre pour a, le x-ème élément de S
- (Si $y \leq m$) Prendre pour b, le y-ème élément de S
- Comparer a et b à chaque élément de L et calculer
  - le rang r de a dans L
  - l'ensemble T des éléments de L compris entre a et b
- Si $|T| > 4m$, ou si $r > k$, ou si $r + |T| < k$, recommencer (l'algorithme échoue)
- Sinon, trier T et en retourner le $k - r$-ème élément de T

Analyse de LazySelect

- Si l'algorithme n'échoue pas, il retourne bien le k-ème plus petit élément du tableau

Analyse de LazySelect

- Si l'algorithme n'échoue pas, il retourne bien le k-ème plus petit élément du tableau
- Chaque tentative a un coût $2n + O(n^{3/4} \log(n))$, et une probabilité d'échec $p(n)$

Analyse de LazySelect

- Si l'algorithme n'échoue pas, il retourne bien le k-ème plus petit élément du tableau
- Chaque tentative a un coût $2n + O(n^{3/4} \log(n))$, et une probabilité d'échec $p(n)$
- Le nombre moyen de tentatives jusqu'au premier succès est donc géométrique, de paramètre $1 - p(n)$, donc a pour espérance $1/(1 - p(n))$

Analyse de LazySelect

- Si l'algorithme n'échoue pas, il retourne bien le k-ème plus petit élément du tableau
- Chaque tentative a un coût $2n + O(n^{3/4} \log(n))$, et une probabilité d'échec $p(n)$
- Le nombre moyen de tentatives jusqu'au premier succès est donc géométrique, de paramètre $1 - p(n)$, donc a pour espérance $1/(1 - p(n))$
- Donc le coût moyen de l'algorithme est $(2n + O(n^{3/4} \log(n)))/(1 - p(n))$

Analyse de LazySelect

- Si l'algorithme n'échoue pas, il retourne bien le k-ème plus petit élément du tableau
- Chaque tentative a un coût $2n + O(n^{3/4} \log(n))$, et une probabilité d'échec $p(n)$
- Le nombre moyen de tentatives jusqu'au premier succès est donc géométrique, de paramètre $1 - p(n)$, donc a pour espérance $1/(1 - p(n))$
- Donc le coût moyen de l'algorithme est $(2n + O(n^{3/4} \log(n)))/(1 - p(n))$
- Pour peu que $p(n)$ tende vers 0, on a $1/(1 - p(n)) = 1 + p(n) + o(p(n))$ et la complexité moyenne est bien $2n + o(n)$ comme annoncé.

Cas d'échec de LazySelect

Pour que l'algorithme échoue (et recommence), il faut au moins qu'un des quatre événements suivants se produise :

- A : le rang r de a dans L, est supérieur à k ; autrement dit, S contient **moins de** x éléments qui sont parmi les k plus petits de L.

Cas d'échec de LazySelect

Pour que l'algorithme échoue (et recommence), il faut au moins qu'un des quatre événements suivants se produise :

- A : le rang r de a dans L, est supérieur à k ; autrement dit, S contient **moins de** x éléments qui sont parmi les k plus petits de L.
- B : le rang s de b dans L, est inférieur à k ; autrement dit, S contient **moins de** y éléments qui sont parmi les k plus petits de L

Cas d'échec de LazySelect

Pour que l'algorithme échoue (et recommence), il faut au moins qu'un des quatre événements suivants se produise :

- A : le rang r de a dans L, est supérieur à k ; autrement dit, S contient **moins de** x éléments qui sont parmi les k plus petits de L.
- B : le rang s de b dans L, est inférieur à k ; autrement dit, S contient **moins de** y éléments qui sont parmi les k plus petits de L
- C : le rang r de a dans L, est inférieur à $k - 2m^{3/4}$; autrement dit, S contient **plus de** x éléments qui sont parmi les $k - 2m^{3/4}$ plus petits de L

Cas d'échec de LazySelect

Pour que l'algorithme échoue (et recommence), il faut au moins qu'un des quatre événements suivants se produise :

- A : le rang r de a dans L, est supérieur à k ; autrement dit, S contient **moins de** x éléments qui sont parmi les k plus petits de L.
- B : le rang s de b dans L, est inférieur à k ; autrement dit, S contient **moins de** y éléments qui sont parmi les k plus petits de L
- C : le rang r de a dans L, est inférieur à $k - 2m^{3/4}$; autrement dit, S contient **plus de** x éléments qui sont parmi les $k - 2m^{3/4}$ plus petits de L
- D : le rang s de b dans L, est supérieur à $k + 2m^{3/4}$; autrement dit, S contient **moins de** y éléments qui sont parmi les $k + 2m^{3/4}$ plus petits de L.

Borne sur la probabilité d'échec

- Chacun des 4 événements A, B, C et D s'exprime sous la forme : une certaine variable binomiale, de paramètres $n^{3/4}$ et p (variable), est plus grande ou plus petite que son espérance d'au moins $n^{1/2}$;

Borne sur la probabilité d'échec

- Chacun des 4 événements A, B, C et D s'exprime sous la forme : une certaine variable binomiale, de paramètres $n^{3/4}$ et p (variable), est plus grande ou plus petite que son espérance d'au moins $n^{1/2}$;
- or une telle variable binomiale, a une variance qui est inférieure à $n^{3/4}/4$, donc un écart-type σ qui est au plus $n^{3/8}/2$;

Borne sur la probabilité d'échec

- Chacun des 4 événements A, B, C et D s'exprime sous la forme : une certaine variable binomiale, de paramètres $n^{3/4}$ et p (variable), est plus grande ou plus petite que son espérance d'au moins $n^{1/2}$;
- or une telle variable binomiale, a une variance qui est inférieure à $n^{3/4}/4$, donc un écart-type σ qui est au plus $n^{3/8}/2$;
- donc, un écart à la moyenne de $n^{1/2}$, représente au moins $2n^{1/8}$ fois l'écart-type ;

Borne sur la probabilité d'échec

- Chacun des 4 événements A, B, C et D s'exprime sous la forme : une certaine variable binomiale, de paramètres $n^{3/4}$ et p (variable), est plus grande ou plus petite que son espérance d'au moins $n^{1/2}$;
- or une telle variable binomiale, a une variance qui est inférieure à $n^{3/4}/4$, donc un écart-type σ qui est au plus $n^{3/8}/2$;
- donc, un écart à la moyenne de $n^{1/2}$, représente au moins $2n^{1/8}$ fois l'écart-type ;
- donc, par l'inégalité de Tchebycheff, chacun des 4 événements a une probabilité inférieure à $1/4n^{1/4}$;

Borne sur la probabilité d'échec

- Chacun des 4 événements A, B, C et D s'exprime sous la forme : une certaine variable binomiale, de paramètres $n^{3/4}$ et p (variable), est plus grande ou plus petite que son espérance d'au moins $n^{1/2}$;
- or une telle variable binomiale, a une variance qui est inférieure à $n^{3/4}/4$, donc un écart-type σ qui est au plus $n^{3/8}/2$;
- donc, un écart à la moyenne de $n^{1/2}$, représente au moins $2n^{1/8}$ fois l'écart-type ;
- donc, par l'inégalité de Tchebycheff, chacun des 4 événements a une probabilité inférieure à $1/4n^{1/4}$;
- donc, la probabilité d'échec $p(n)$ est au plus de $n^{-1/4}$, qui tend bien vers 0.

Comparaison des deux algorithmes

- RandQuickSelect : en moyenne, $n(2 + 2\log(k/n))$ comparaisons, $3.4n$ pour $k = n/2$

Comparaison des deux algorithmes

- RandQuickSelect : en moyenne, $n(2 + 2\log(k/n))$ comparaisons, $3.4n$ pour $k = n/2$
- LazySelect : en moyenne, $2n + o(n)$ comparaisons

Comparaison des deux algorithmes

- RandQuickSelect : en moyenne, $n(2 + 2\log(k/n))$ comparaisons, $3.4n$ pour $k = n/2$
- LazySelect : en moyenne, $2n + o(n)$ comparaisons
- **En pratique,** le $o(n)$ de LazySelect cache des termes en $n^{3/4}\log(n)$, qui ne sont réellement négligeables devant n que pour de très grands n

Évaluation d'un arbre ET/OU

(Un bel exemple où un algorithme probabiliste simple fait asymptotiquement mieux que tous les algorithmes déterministes)

- **Donnée :** un arbre binaire complet, de hauteur $h = 2k$ (donc $n = 2^{2k}$ feuilles)

Évaluation d'un arbre ET/OU

(Un bel exemple où un algorithme probabiliste simple fait asymptotiquement mieux que tous les algorithmes déterministes)

- **Donnée :** un arbre binaire complet, de hauteur $h = 2k$ (donc $n = 2^{2k}$ feuilles)
- chaque feuille est valuée : 0 ou 1

Évaluation d'un arbre ET/OU

(Un bel exemple où un algorithme probabiliste simple fait asymptotiquement mieux que tous les algorithmes déterministes)

- **Donnée :** un arbre binaire complet, de hauteur $h = 2k$ (donc $n = 2^{2k}$ feuilles)
- chaque feuille est valuée : 0 ou 1
- les noeuds internes sont étiquetés $\textsc{Ou}$ (la racine, et les noeuds à profondeur paire) ou $\textsc{Et}$ (les noeuds à profondeur impaire)

Évaluation d'un arbre ET/OU

(Un bel exemple où un algorithme probabiliste simple fait asymptotiquement mieux que tous les algorithmes déterministes)

- **Donnée :** un arbre binaire complet, de hauteur $h = 2k$ (donc $n = 2^{2k}$ feuilles)
- chaque feuille est valuée : 0 ou 1
- les noeuds internes sont étiquetés $\textsc{Ou}$ (la racine, et les noeuds à profondeur paire) ou $\textsc{Et}$ (les noeuds à profondeur impaire)
- le but est de déterminer la valeur de la racine

Évaluation d'un arbre ET/OU

(Un bel exemple où un algorithme probabiliste simple fait asymptotiquement mieux que tous les algorithmes déterministes)

- **Donnée :** un arbre binaire complet, de hauteur $h = 2k$ (donc $n = 2^{2k}$ feuilles)
- chaque feuille est valuée : 0 ou 1
- les noeuds internes sont étiquetés $\textsc{Ou}$ (la racine, et les noeuds à profondeur paire) ou $\textsc{Et}$ (les noeuds à profondeur impaire)
- le but est de déterminer la valeur de la racine
- (Idée : l'arbre représente un jeu où deux joueurs choisissent alternativement un fils du noeud courant, jusqu'à atteindre une feuille ; le premier joueur "gagne" si on termine sur une feuille 1 ; la question est de savoir si le premier joueur a une stratégie gagnante)

Complexité des algorithmes déterministes

Proposition

Pour tout algorithme déterministe de calcul de la valeur d'un arbre Et/Ou, et pour toute hauteur, il existe un arbre de cette hauteur pour lequel l'algorithme examine la valeur de chacune des n feuilles.

Complexité des algorithmes déterministes

Proposition

Pour tout algorithme déterministe de calcul de la valeur d'un arbre Et/Ou, et pour toute hauteur, il existe un arbre de cette hauteur pour lequel l'algorithme examine la valeur de chacune des n feuilles.

Idée de preuve : chaque fois que la valeur d'un noeud est examinée ou calculée, si le noeud frère n'a pas été évalué ou examiné, cela ne suffit pas à déterminer la valeur du père.

Un algorithme probabiliste simplissime

- **Récursivement :** pour évaluer un noeud (autre qu'une feuille), on choisit aléatoirement un de ses fils, qu'on évalue ; et on évalue l'autre fils seulement si la valeur du premier ne permet pas de conclure.

Un algorithme probabiliste simplissime

- **Récursivement :** pour évaluer un noeud (autre qu'une feuille), on choisit aléatoirement un de ses fils, qu'on évalue ; et on évalue l'autre fils seulement si la valeur du premier ne permet pas de conclure.
- L'analyse, telle quelle, demande de distinguer selon l'étiquette $\textsc{Et}/\textsc{Ou}$ du noeud, et les valeurs des fils ; il est plus simple de se ramener à un arbre $\textsc{NonEt}$

Un algorithme probabiliste simplissime

- **Récursivement :** pour évaluer un noeud (autre qu'une feuille), on choisit aléatoirement un de ses fils, qu'on évalue ; et on évalue l'autre fils seulement si la valeur du premier ne permet pas de conclure.
- L'analyse, telle quelle, demande de distinguer selon l'étiquette ET/OU du noeud, et les valeurs des fils ; il est plus simple de se ramener à un arbre NONET
- Si, conservant les valeurs des feuilles, on remplace toutes les étiquettes des noeuds internes par l'opération NONET, on change les valeurs de tous les noeuds à profondeur impaire, et on conserve celles des noeuds à profondeur paire (l'arbre est de hauteur paire).

Analyse de la complexité moyenne

- On définit Z_h comme le maximum, sur tous les arbres NonEt de hauteur h dont la racine vaut 0, du nombre moyen de feuilles examinées.

Analyse de la complexité moyenne

- On définit Z_h comme le maximum, sur tous les arbres NonEt de hauteur h dont la racine vaut 0, du nombre moyen de feuilles examinées.
- De même, U_h désigne le maximum, sur tous les arbres NonEt de hauteur h et dont la racine vaut 1, du nombre moyen de feuilles examinées.

Analyse de la complexité moyenne

- On définit Z_h comme le maximum, sur tous les arbres NonEt de hauteur h dont la racine vaut 0, du nombre moyen de feuilles examinées.
- De même, U_h désigne le maximum, sur tous les arbres NonEt de hauteur h et dont la racine vaut 1, du nombre moyen de feuilles examinées.
- Clairement $U_0 = Z_0 = 1$, $Z_1 = 2$, $U_1 = 3/2$

Analyse de la complexité moyenne

- On définit Z_h comme le maximum, sur tous les arbres NonEt de hauteur h dont la racine vaut 0, du nombre moyen de feuilles examinées.
- De même, U_h désigne le maximum, sur tous les arbres NonEt de hauteur h et dont la racine vaut 1, du nombre moyen de feuilles examinées.
- Clairement $U_0 = Z_0 = 1$, $Z_1 = 2$, $U_1 = 3/2$
- On montre facilement les récurrences :

Analyse de la complexité moyenne

- On définit Z_h comme le maximum, sur tous les arbres NonEt de hauteur h dont la racine vaut 0, du nombre moyen de feuilles examinées.
- De même, U_h désigne le maximum, sur tous les arbres NonEt de hauteur h et dont la racine vaut 1, du nombre moyen de feuilles examinées.
- Clairement $U_0 = Z_0 = 1$, $Z_1 = 2$, $U_1 = 3/2$
- On montre facilement les récurrences :
  - $Z_h = 2U_{h-1}$

Analyse de la complexité moyenne

- On définit Z_h comme le maximum, sur tous les arbres NonEt de hauteur h dont la racine vaut 0, du nombre moyen de feuilles examinées.
- De même, U_h désigne le maximum, sur tous les arbres NonEt de hauteur h et dont la racine vaut 1, du nombre moyen de feuilles examinées.
- Clairement $U_0 = Z_0 = 1$, $Z_1 = 2$, $U_1 = 3/2$
- On montre facilement les récurrences :
  - $Z_h = 2U_{h-1}$
  - $U_h = \max(Z_{h-1}, Z_{h-1} + \frac{1}{2}U_{h-1})$

Analyse de la complexité moyenne

- On définit Z_h comme le maximum, sur tous les arbres NonEt de hauteur h dont la racine vaut 0, du nombre moyen de feuilles examinées.
- De même, U_h désigne le maximum, sur tous les arbres NonEt de hauteur h et dont la racine vaut 1, du nombre moyen de feuilles examinées.
- Clairement $U_0 = Z_0 = 1$, $Z_1 = 2$, $U_1 = 3/2$
- On montre facilement les récurrences :
  - $Z_h = 2U_{h-1}$
  - $U_h = \max(Z_{h-1}, Z_{h-1} + \frac{1}{2}U_{h-1})$
- Donc $U_h = \frac{1}{2}U_{h-1} + 2U_{h-2}$

Analyse de la complexité moyenne

- On définit Z_h comme le maximum, sur tous les arbres NonEt de hauteur h dont la racine vaut 0, du nombre moyen de feuilles examinées.
- De même, U_h désigne le maximum, sur tous les arbres NonEt de hauteur h et dont la racine vaut 1, du nombre moyen de feuilles examinées.
- Clairement $U_0 = Z_0 = 1$, $Z_1 = 2$, $U_1 = 3/2$
- On montre facilement les récurrences :
  - $Z_h = 2U_{h-1}$
  - $U_h = \max(Z_{h-1}, Z_{h-1} + \frac{1}{2}U_{h-1})$
- Donc $U_h = \frac{1}{2}U_{h-1} + 2U_{h-2}$
- L'équation caractéristique $X^2 - X/2 - 2$ a deux racines $x = \frac{1 \pm \sqrt{33}}{4}$; $U_h = \Theta(1.686^h)$, idem pour Z_h.

Récapitulons. . .

- Tout algorithme déterministe peut avoir, dans le pire des cas, à examiner $n = 2^{2k}$ feuilles

Récapitulons. . .

- Tout algorithme déterministe peut avoir, dans le pire des cas, à examiner $n = 2^{2k}$ feuilles
- Pour tout arbre, l'algorithme probabiliste examinera en moyenne, au plus $1.686^{2k} \simeq n^{0.754}$ feuilles