L3 Info - Algorithmique

UFR IM²AG
UNIVERSITÉ Grenoble Alpes

Travaux dirigés 1, séance 1

Analyse d'algorithmes, calculs de coûts

Exercice 1. Itérations emboîtées (30 min)

Compter le nombre d'additions exécutées par chacun des algorithmes suivants. On ne compte pas les incrémentations des indices de boucles.

1.
```
(1) pour i = 1 à n faire
(2)     pour j = 1 à n faire
(3)         x := x+a
```

2.
```
(1) pour i = 1 à n faire
(2)     pour j = 1 à i faire
(3)         x := x+a
```

3.
```
(1) pour i de 5 à n-5 faire
(2)     pour j de i-5 à i+5 faire
(3)         x := x+a
```

4.
```
(1) pour i = 1 à n faire
(2)     pour j = 1 à i faire
(3)         pour k = 1 à j faire
(4)             x := x+a
```

5. Plus généralement, que pouvez-vous dire de la complexité d'un algorithme en observant le nombre de boucles emboîtées ?

Pour tous ces algorithmes la ligne x := x + a effectue exactement une addition (et c'est la seule).

1. La boucle (2) s'exécute n fois et elle effectue donc n additions.
   La boucle (1) s'exécute n fois et elle effectue donc $n \times n = n^2$ additions.
2. La boucle (2) s'exécute i fois et elle effectue donc i additions.
   La boucle (1) s'exécute n fois, pour i allant de 1 à n. Elle effectue donc $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$ additions.
3. Il y a ici un "piège" car le nombre d'itérations de la boucle interne ne dépend pas de i.
   La boucle (2) s'exécute $(i+5)-(i-5)+1 = 11$ fois et elle effectue donc 11 additions.
   La boucle (1) s'exécute $(n-5)-4 = n-9$ fois et elle effectue donc $11(n-9) = 11n-99$ additions.
4. La boucle (3) s'exécute j fois et elle effectue donc j additions.
   La boucle (2) s'exécute i fois, pour j allant de 1 à i. Elle effectue donc $\sum_{j=1}^{i} j = \frac{i(i+1)}{2} = \frac{i^2}{2} + \frac{i}{2}$ additions.
   La boucle (1) s'exécute n fois, pour i allant de 1 à n. Elle effectue donc $\sum_{i=1}^{n} \frac{i^2}{2}+\frac{i}{2} = \frac{1}{2}\sum_{i=1}^{n} i^2+\frac{1}{2}\sum_{i=1}^{n} i = \frac{1}{2}\frac{n(n+1)}{2} + \frac{1}{2}\frac{n(n+1)(2n+1)}{6}$ additions.
   Notons que cette dernière expression est en $\mathcal{O}(n^3)$.
5. La complexité sera en général un polynôme dont le degré est égal au nombre maximal de boucles emboîtées, **à condition que l'intervalle de variation de chaque indice soit proportionnel à celui d'un des indices supérieurs**.

Exercice 2. Recherche séquentielle (30 min)

On étudie un algorithme de recherche séquentielle dans une table. On se place dans le cas où il n'y a pas d'hypothèse sur le fait que la table est ordonnée ni sur la présence de l'élément cherché dans la table.

Il s'agit d'évaluer le nombre de comparaisons effectuées lors d'une recherche d'un élément x dans une table T de taille n.

— Spécifier et écrire proprement un tel algorithme.

RECH_SEQ(e, T, n)
Données : l'élément e à rechercher, un tableau T de taille n
Résultat : le premier indice i tel que $T[i] = e$, ou -1 s'il est absent
$i := 0$;
while $i < n$ *et puis* $T[i] \neq e$
 ⌊ $i := i + 1$
if $i < n$
 ⌊ **return** i
else
 ⌊ **return** -1

— Déterminer les cas favorables et défavorables et les nombres de comparaisons correspondants.
Les pires cas sont faciles à exhiber : ce sont ceux où la boucle effectue un maximum d'itérations.
Cela se produit si e est absent de T ou en dernière position, d'où $C^{max}_{RECH\_SEQ}(n) = n$.
Le cas favorable est celui où $T[1] = e$, et alors $C^{min}_{RECH\_SEQ}(n) = 1$.

Exercice 3. Valeurs numériques et ordres de grandeurs (30 min)

On suppose qu'on travaille sur une machine capable d'effectuer environ un milliard d'opérations par seconde.

Calculer (**sans calculatrice**) le temps nécessaire approximatif pour exécuter des programmes dont les coûts sont donnés ci-dessous, avec des données de différentes tailles en entrée :

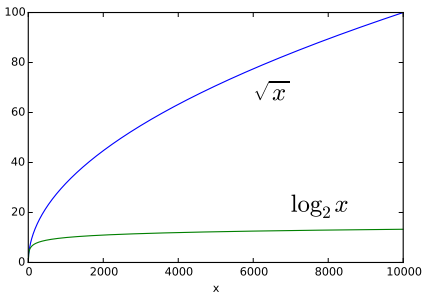
↓ Coût de l'algorithme \ Taille des données →	1 000	1 000 000	1 000 000 000
n	10^{-6} s	10^{-3} s	1 s
$n \log_2 n$	10^{-5} s	$1/50$ s	30 s
$n + 1\,000\,000$	10^{-3} s	2×10^{-3} s	1 s
$\frac{n^2}{1\,000} + 1\,000n$	10^{-3} s	2 s	11 jours et demi

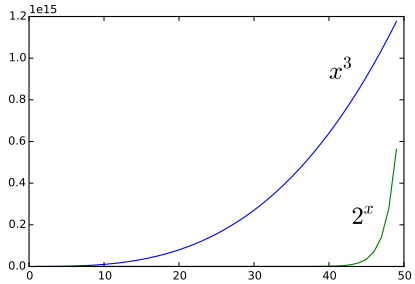
1. Lesquels de ces algorithmes sont utilisables :
   — à chaque chargement d'une page web ?
   n et $n + 1\,000\,000$, à la limite la version $n \log_2 n$ si pas trop de données à traiter
   — à chaque démarrage d'une machine ?
   Tous sauf $n^2/1000$
   — pour produire les plans d'une usine ?
   Tous (on ne construit pas une usine tous les jours !), à condition que $n = 10^9$ soit considéré comme un volume de données raisonnable pour ce problème.

   Quelle(s) conclusion(s) plus générale(s) en tirez-vous sur les ordres de grandeurs respectifs de ces coûts ?
   — n et $n + 1\,000\,000$ sont équivalents.
   — $n \log_2$ est presque du même ordre sauf pour de très grandes données.
   — n^2 est nettement au-dessus, même avec une constante faible.
2. Tracer sur une même figure les allures des courbes des fonctions $\log_2(n)$ et $\sqrt{n}$ sur une échelle assez grande ($n = 10\,000$ par exemple).

   Même consigne pour $10^{10} \times n^3$ et 2^n, avec n compris entre 0 et 50.

   Il faut voir ici que les allures de courbes apprises au lycée ne sont plus vraiment valables pour n grand, et en particulier que $\log_2$ et 2^n ne présentent pas de branche pseudo-parabolique comme on a tendance parfois à le croire.





Exercice 4

On considère l'algorithme suivant :

```
(1)  a := T[n]
(2)  pour i = 1 à n-1 faire
(3)        si T[i] > a faire
(4)           traiter T[i]
(5)           a := T[i]
```

1. Que calcule-t-il ? Expliciter les données, le résultat.
   Il calcule le maximum d'un tableau. Les données sont T et n, le résultat est a.
2. Expliciter le modèle de coût. Quel est son coût minimal, son coût maximal ?
   Considérons que l'appel à traiter est l'opération coûteuse et négligeons le reste.
   Selon le résultat du test, le coût de l'instruction conditionnelle (3) est compris entre 0 et 1.

Par conséquent, le coût de la boucle (2) est compris entre 0 et $n-1$ et on peut exhiber des cas favorable et défavorable pour lesquels ces bornes sont atteintes.
Donc $C^{min}(n) = 0$ et $C^{max}(n) = n-1$.

3. Peut-on espérer écrire un algorithme plus efficace pour résoudre le même problème ? Supposons qu'un algorithme A effectue strictement moins de comparaisons, donc n-2 au maximum. Alors quelles que soient les comparaisons effectuées, il existe deux sous-ensembles T_1 et T_2 de T tels qu'aucun élément de T_1 ne soit comparé avec un élément de T_2.
Si A choisit un élément de T_1 comme maximum, on peut construire une donnée telle que A choisisse le même élément mais que le maximum réel soit dans T_2.
Un tel algorithme A ne peut donc pas être correct.

À préparer pour la prochaine fois

On rappelle ci-dessous le principe de deux algorithmes de tri d'un tableau indexé de 1 à n.

1. Lire et comprendre les descriptions de ces algorithmes.
2. Illustrer leur fonctionnement par des schémas.
3. Rédiger proprement ces deux tris en langage algorithmique.

A. Tri par insertion : une itération ($i = 2$ à n) à chaque pas de laquelle on insère à sa place l'élément d'indice i dans la séquence triée formée des $i-1$ premiers éléments.
Initialement : l'élément 1 forme une séquence triée.
Finalement : les n éléments sont triés.
On effectue l'insertion par une recherche séquentielle de l'emplacement k de l'élément i, et un décalage vers la droite des éléments de k à $i-1$. L'algorithme classique effectue ces deux opérations ensemble, c'est-à-dire décale l'élément i vers la gauche (par un échange) jusqu'à ce qu'il atteigne sa "bonne" place.

B. Tri par sélection (du minimum) : une itération ($i = 1$ à $n-1$) à chaque pas de laquelle on trouve et met à sa place i l'élément correspondant.
Finalement : les $n-1$ premiers éléments sont à leur place, donc le dernier est bien placé aussi.
On effectue la sélection par un parcours des éléments de i à n en sélectionnant le minimum, et en mémorisant son indice k, puis en effectuant l'échange entre les éléments i et k.