Quick Algorithmique et Modélisation

Quelques éléments de correction

Jean-Marc.Vincent@imag.fr

UGA UFR IM²AG Université Grenoble Alpes

Mars 2022

Attention : ceci est un corrigé, il y a bien entendu d'autres solutions, d'autres approches tout aussi valides. La rédaction a été détaillée pour aider à la compréhension, un tel niveau de détail n'était pas attendu sur la copie.

Corrigé Q1 2022

Bogosort, un algorithme de tri randomisé (1)

Le tableau T de taille n (indicé de 0 à $n-1$) contient des éléments distincts comparables deux à deux. On souhaite effectuer une permutation aléatoire des éléments de ce tableau. Pour cela on dispose d'un générateur aléatoire `Aléa` (a,b) qui génère uniformément un entier compris entre a et b inclus, les appels successifs à `Aléa` seront supposés indépendants.
On considère l'algorithme suivant opérant sur le tableau T.

```
Permuter (T)
  for i = n − 1 to 1
    j = Aléa ( 0 , i )
    Échanger (T,i,j)        // échange les valeurs T[i] et T[j]
```

Montrer que `Permuter` (T) génère bien une permutation du tableau T.

- argument sur la fonction d'échange
- invariant : *Les valeurs contenues dans le tableau* $T[0, n-1]$ *sont identiques aux valeurs du tableau en début d'exécution*
- terminaison : la valeur de la variable i est un variant d'itération
- L'algorithme se termine, l'invariant est vérifié en début d'itération (trivial), il est vrai au sortir de l'itération, et prouve la propriété demandée en fin d'exécution de l'algorithme.

Bogosort, un algorithme de tri randomisé (2)

Combien existe-t-il de permutations différentes du tableau T ?

Il y a $n!$ permutations différentes, en effet, c'est le nombre de bijections de $\{1, \cdots, n\}$ sur $\{1, \cdots, n\}$. Il y a n valeurs possible pour l'mage de 1, puis $n-1$ pour l'image de 2, etc, soit $n.(n-1)\cdots 1 = n!$

(Bonus) Réécrire la fonction `Permuter` sous forme récursive

Il faut rajouter un paramètre pour indiquer la partie du tableau à permuter (ici `Permuter` $(T$,k$k)$ effectue une permutation aléatoire uniforme des valeurs du sous-tableau de l'indice 0 à l'indice $k-1$

```
Permuter (T,k)
  if k ⩾ 1
    j = Aléa ( 0 , k − 1 )
    Échanger (T,k − 1,j)
    Permuter (T,k-1)
```

Bogosort, un algorithme de tri randomisé (3)

```
Permuter (T)
  for i = n − 1 to 1
    j = Aléa ( 0 , i )
    Échanger (T,i,j)        // échange les valeurs T[i] et T[j]
```

Prouver que l'algorithme `Permuter` génère une permutation selon une loi uniforme sur l'ensemble de toutes les permutations possibles. C'est à dire : étant donnée une permutation spécifique de T, la probabilité que `Permuter` (T) génère cette permutation spécifique est égale à 1/(le nombre total de permutations possibles).

Notations : $T = [e_0, e_1, \cdots, e_{n-1}]$, et soit une permutation arbitraire $T' = [e_{i_0}, e_{i_1}, \cdots, e_{i_{n-1}}]$. Il faut montrer que la probabilité d'observer T' à la fin de l'exécution de l'algorithme est $\frac{1}{n!}$.
Remarquons dans l'itération les éléments de l'indice i à $n-1$ sont inchangés (invariant !!)
Le dernier élément est affecté par uniquement la premi ère itération, d'où

$$\mathbb{P}(T'[n-1] = e_{i_{n-1}}) = \frac{1}{n}$$

car `Aléa` suit une loi uniforme sur $\{0, \cdots, n-1\}$

puis

$$\mathbb{P}(T'[n-2] = e_{i_{n-2}}, T'[n-1] = e_{i_{n-1}}) =$$

$$\mathbb{P}(T'[n-2] = e_{i_{n-2}}, \Big| T'[n-1] = e_{i_{n-1}})\mathbb{P}(T'[n-1] = e_{i_{n-1}}) = \frac{1}{n-1}\frac{1}{n}$$

et ainsi de suite

$$\mathbb{P}(T' = [e_{i_0}, e_{i_1}, \cdots, e_{i_{n-1}}]) = \frac{1}{1}\frac{1}{2}\cdots\frac{1}{n-1}\frac{1}{n} = \frac{1}{n!}$$

La probabilité de générer une permutation arbitraire est égale à $\frac{1}{n!}$, (la même pour n'importe quelle permutation), la génération est donc uniforme sur l'ensemble des permutations possibles.
De plus, si on considère une séquence d'appels à `Permuter`, les permutations générées seront indépendantes car font des appels différents à la fonction `Aléa` (qui est un générateur supposé correct).

Bogosort, un algorithme de tri randomisé (4)

On envisage de trier le tableau T de taille n en le mélangeant jusqu'à ce qu'il soit trié avec la fonction `Bogosort` (.).

```
Bogosort (T)
  repeat
    | Permuter (T)
  until Est_trié (T)
```

Au minimum, combien de fois la fonction `Bogosort` (T) appelle-t-elle la fonction `Permuter` (T) ? Au maximum, combien de fois la fonction `Bogosort` (T) appelle-t-elle la fonction `Permuter` (T) ?

Au minimum 1 fois, il suffit que la première permutation range les éléments comme il faut. L'algorithme peut effectuer un nombre arbitraire d'appels à `Permuter`, donc potentiellement un nombre infini d'appels.

En moyenne, combien de fois la fonction `Bogosort` (T) appelle-t-elle la fonction `Permuter`? Quelle est la probabilité que la fonction `Bogosort` (T) appelle exactement k fois la fonction `Permuter` (T) ?

On a un algorithme basé sur le rejet, la probabilité d'acceptation est $p_a = \frac{1}{n!}$. Donc le nombre moyen d'itérations sera $\frac{1}{p_a} = n!$ et la loi du nombre d'itérations est une loi géométrique de paramètre p_a.

Corrigé Q1 2022

Avant La Reine des neiges et Le Roi lion

On dispose d'un tableau des films de Walt Disney dans l'ordre chronologique. Ce tableau contient des triplés (Rang,Titre,Année). On souhaite réorganiser le tableau en plaçant les films par rang.
Par exemple, pour les 13 premiers films, on a le tableau sur la gauche que l'on veut transformer en le tableau de droite.

Rang	Titre du film	Année
1	Blanche-Neige et les Sept Nains	1937
13	Dumbo	1941
3	Bambi	1942
4	Cendrillon	1950
12	Alice au pays des merveilles	1951
8	Peter Pan	1953
7	La Belle et le Clochard	1955
9	La Belle au bois dormant	1959
5	Les 101 Dalmatiens	1961
11	Merlin l'Enchanteur	1963
2	Le Livre de la jungle	1967
6	Les Aristochats	1970
10	Robin des Bois	1973

Rang	Titre du film	Année
1	Blanche-Neige et les Sept Nains	1937
2	Le Livre de la jungle	1967
3	Bambi	1942
4	Cendrillon	1950
5	Les 101 Dalmatiens	1961
6	Les Aristochats	1970
7	La Belle et le Clochard	1955
8	Peter Pan	1953
9	La Belle au bois dormant	1959
10	Robin des Bois	1973
11	Merlin l'Enchanteur	1963
12	Alice au pays des merveilles	1951
13	Dumbo	1941

Le tableau T est indicé de 1 à n ; on accède au rang du film de la ligne i par $T[i].rang$. Dans l'exemple ci-dessus, $T[13].rang$ vaut 10.

Avant La Reine des neiges et Le Roi lion(2)

Écrire un algorithme efficace (en $\mathcal{O}(n)$) qui retourne un nouveau tableau contenant les éléments de T triés par rang.

```
ranger_externe (T, T')
  for i = 1 to n
    T'[T[i].rang] = T[i]
```

Le coût de cet algorithme est clairement en (n) l'espace mémoire utilisé est le double de l'espace mémoire initial.

Avant La Reine des neiges et Le Roi lion(3)

Écrire un algorithme qui opère directement sur le tableau (en place). Prouver cet algorithme. Donner sa complexité en temps et en mémoire.

```
ranger_interne (T)
  for i = 1 to n
  |  while T[i].rang ≠ i
  |  |  Échanger (T,i,T[i].rang)
  |
```

Correction de l'algorithme

- Invariant de la boucle externe : pour tout k, avec $1 \leqslant k < i$ les $T[k].rang = k$ (les éléments de 1 à $i-1$ sont bien placés
- On peut remarquer que tout élément bien placé ne sera jamais déplacé
- Chaque itération de la boucle interne déplace un élément à sa place, et ne touche pas les éléments bien placés
- Le nombre d'itérations interne est fini car le nombre d'éléments restant à placer décroit strictement et est minoré par 0
- À la fin de l'itération externe l'élément de rang i est bien placé (condition de sortie du while) et preuve de l'invariant
- Le nombre d'itérations externe est égal à n (fini) donc l'algorithme se termine et l'invariant de la boucle externe indique que chaque élément est bien placé.

Avant La Reine des neiges et Le Roi lion(4)

Écrire un algorithme qui opère directement sur le tableau (en place). Prouver cet algorithme. Donner sa complexité en temps et en mémoire.

```
ranger_interne (T)
  for i = 1 to n
  |  while T[i].rang ≠ i
  |  |  Échanger (T,i,T[i].rang)
  |
```

Complexité de l'algorithme

- un élément ne peut être Échanger que 2 fois, une fois pour le placer à une position i, puis à l'itération interne suivante le mettre en place. UNe fois en place il ne peut plus être déplacé
- le nombre d'échanges est donc en (n)
- Le nombre de passage dans la boucle externe est également en (n), cet algorithme est donc linéaire en la taille des entrées
- l'espace mémoire utilisé est de la taille de la donnée en entrée, avec un petit surcoût (en (1) pour la fonction Échanger

Corrigé Q1 2022

SUM, conception d'algorithme

Soit $\mathcal{E}$ un ensemble de n éléments. À chaque élément i de $\mathcal{E}$, on associe une valeur v_i.
Pour éviter les cas particuliers, on supposera les v_i distincts.
On cherche tous les couples (i,j) avec $i \neq j$ tels que :

$$v_i + v_j = S \qquad \text{pour un S donné.;}$$

Ce problème a été vu en ALGO5 ; où un algorithme naïf en $\mathcal{O}(n^2)$ testait tous les couples (i,j) possibles. Une version améliorée basée sur un pré-calcul (tri du tableau) permettait d'améliorer la complexité en $\mathcal{O}(n \log(n))$.

Proposer un algorithme qui résout le problème en $\mathcal{O}(n)$ opérations.

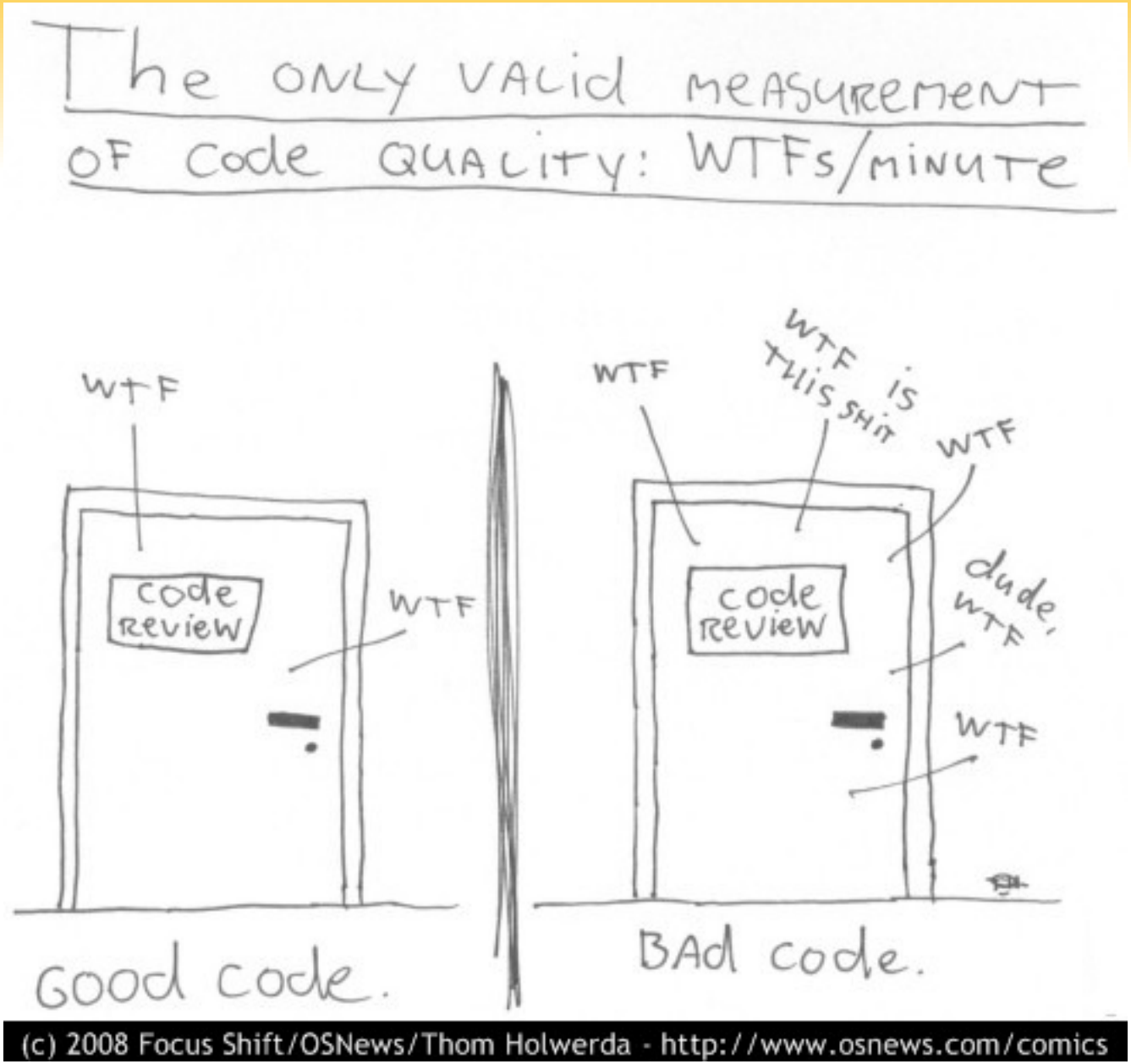
- Penser table de hachage avec une fonction h
- Première passe remplissage avec i à l'adresse $h(v_i)$
- Deuxième passe test si $S - v_j$ est dans la table, si c'est le cas i et j sont candidats
- Vérifier que $i \neq j$

Corrigé Q1 2022

Conseils pour un examen

- au préalable : quels sont les grands chapitres du cours au programme de l'examen ? (identifier les gros blocs, les exercices types, les "grands résultats")
- lire le sujet en entier (pour voir où l'on va)
- rédiger les réponses, ainsi s'il y a une erreur sur l'algorithme vous aurez expliqué votre méthode
- rédiger les preuves correctement hypothèses/déroulement logique des arguments/conclusion
- un algorithme sans explication c'est comme une Ferrari sans roue
- soigner la présentation et la rédaction

Bref, entrainez-vous...


The ONLY VALID MEASUREMENT OF CODE QUALITY: WTFs/MINUTE
WTF
code review
WTF
Good code.
WTF
WTF IS THIS SHIT
WTF
code review
dude, WTF
WTF
BAd code.
(c) 2008 Focus Shift/OSNews/Thom Holwerda - http://www.osnews.com/comics