Quick Algorithmique et Modélisation Q1

Quelques éléments de correction

Jean-Marc.Vincent@imag.fr

Université Joseph Fourier
GRENOBLE
UFR IM²AG

18 mars 2015

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.

Exercice : Random-SAT

On se donne une expression booléenne sous forme normale conjonctive $\Phi(x_1, \cdots, x_n)$ portant sur n variables notées $\{x_1, x_2, \cdots x_n\}$, ayant m clauses, chaque clause ayant k littéraux. C'est une instance du problème k-SAT.

$$\Phi(x_1, \cdots, x_n) = \bigwedge_{i=1}^{m} (x_{i_1} \vee x_{i_2} \cdots \vee x_{i_k});$$

avec x_{i_j} étant l'une des variables x_l ou sa négation. On suppose, sans perte de généralité qu'une clause ne contient pas deux fois la même variable.
Pour résoudre ce problème, on génère aléatoirement et uniformément les valeurs des n variables et on évalue les m clauses. On suppose que la génération des valeurs des variables x_i se fait de manière indépendantes et uniforme : probabilité $\frac{1}{2}$ d'avoir la valeur 1 (VRAI) et $\frac{1}{2}$ d'avoir la valeur 0 (FAUX).
Soit Z_i la variable aléatoire modélisant la valeur de la clause i et soit N la variable aléatoire modélisant le nombre de clauses vérifiées, on a :

$$N = Z_1 + Z_2 + \cdots + Z_m.$$

Université Joseph Fourier UFR IM²AG

Question 1

Calculer la probabilité que la clause i ne soit pas vérifiée ($\mathbb{P}(Z_i = 0)$), en déduire la probabilité que la clause i soit vérifiée ($\mathbb{P}(Z_i = 1)$).

Pour qu'une clause (c'est une disjonction $\vee$) constituée de k littéraux soit fausse il faut que chacun des termes de la clause soit faux. Comme la génération d'une variable x_i suit une loi uniforme, la probabilité que la variable x_j soit vraie est $\frac{1}{2}$ et la probabilité qu'elle soit fausse $\frac{1}{2}$ De même pour la négation de cette variable. Comme les variables dans une clause sont distinctes et que les variables sont indépendantes

$$\mathbb{P}(Z_i = 0) = \underbrace{\frac{1}{2} \times \frac{1}{2} \times \cdots \times \frac{1}{2}}_{k \text{ fois}} = \frac{1}{2^k}.$$

On en déduit immédiatement que

$$\mathbb{P}(Z_i = 1) = 1 - \frac{1}{2^k}.$$

On retrouve bien l'intuition, si k est grand la probabilité que la formule soit vraie augmente et se rapproche de 1.

Question 2

Calculer $\overline{m}$ le nombre moyen de clauses vérifiées.

On calcule d'abord l'espérance de Z_i, qui est $1 \times \mathbb{P}(Z_i = 1) + 0 \times \mathbb{P}(Z_i = 0) = 1 - \frac{1}{2^k}$. Puis en utilisant la linéarité de l'espérance (la moyenne d'une somme est égale à la somme des moyennes) on obtient .

$$\overline{m} = \mathbb{E}N = \mathbb{E}(Z_1 + Z_2 + \cdots + Z_m) = \mathbb{E}Z_1 + \mathbb{E}Z_2 + \cdots + \mathbb{E}Z_m = m\left(1 - \frac{1}{2^k}\right).$$

Penser à bien justifier l'argument.

Question 3

En déduire que si $\overline{m} > m - 1$ alors la formule Φ est satisfiable.
Indication : remarquer que $\overline{m} \leqslant m$

Comme la formule est constituée de m clauses, au plus m clauses peuvent être vérifiées. Si la moyenne est strictement plus grande que $m - 1$ cela veut dire que la probabilité que le nombre de clauses vérifiées soit supérieur strict à $m - 1$ est strictement positive. Or la seule valeur possible au-delà de $m - 1$ est m, donc la probabilité que les m clauses soit vraies est strictement positive. Donc il existe des instances telles que la formule soit satisfiable.
Donc Φ est satisfiable si $\overline{m} > m - 1$, c'est à dire

$$m\left(1 - \frac{1}{2^k}\right) > m - 1,$$

soit encore

$$1 > \frac{m}{2^k}.$$

C'est le cœur de l'exercice, on pourra utiliser ce résultat par la suite. Sous cette condition, le nombre de clauses est plus petit que le nombre de configurations des k valeurs de la clause. Au passage on remarque que le nombre de variables est plus grand que k car les littéraux d'une clause sont concernent des variables différentes.

Question 4

Sous la condition ci-dessus $\overline{m} > m - 1$, proposer un algorithme randomisé calculant une solution à $\Phi(x_1, \cdots, x_n) = 1$.

On a un algorithme simple, on génère des valeurs aléatoires indépendantes uniformes pour les x_i, comme la probabilité d'avoir une solution à la formule Φ est strictement positive, on finira par obtenir la solution.

SAT-ALEATOIRE(Φ)

Data: Une formule Φ à satisfaire ayant n variables, m clauses et des littéraux de taille k et vérifiant $m < 2^k$

Output: Une solution $x = (x_1, \cdots, x_n)$ satisfaisant $\Phi(x)$

repeat

  for $i = 1$ **to** n **do**

    `// génère une configuration du vecteur de variables` X

    $X[i]$ = (`Random ()` $< \frac{1}{2}$) `// Random génère un nombre réel aléatoire de loi uniforme sur` $[0, 1[$

until $\Phi(x)$

`Retourne` X

Question 5

(plus difficile) Soit p la probabilité de générer une solution. Montrer que

$$p.m + (1-p)(m-1) \geqslant \overline{m}.$$

En déduire un minorant de p. Donner un majorant de la moyenne du nombre de tirages nécessaires pour trouver une solution.

On sait déjà que p est positif. La moyenne $\overline{m}$ est donc plus petite que $pm + (1-p)(m-1)$. C'est à dire que toutes les clauses sont satisfaites avec la probabilité p sinon $m-1$ clauses sont satisfaites, avec la probabilité $1-p$.
On en déduit que

$$pm + (1-p)(m-1) > m\left(1 - \frac{1}{2^k}\right);$$

soit encore, en développant les 2 termes et en simplifiant :

$$p > 1 - \frac{m}{2^k}.$$

Dans un tirage à pile ou face (biaisé avec p la probabilité d'avoir pile) le nombre moyen de jets de pièce jusqu'à obtenir un pile est $\frac{1}{p}$, donc on est dans la même situation ici et le nombre moyen d'itérations pour obtenir une solution est

$$\frac{1}{1 - \frac{m}{2^k}}.$$

Université Joseph Fourier UFR IM²AG

Question 6

Que pensez-vous de cette approche ?

- demande la condition $m < 2^k$, ce qui n'est pas forcément le cas.
- simple à mettre en œuvre
- complexité ne dépendant pas du nombre de variables n (que du nombre de littéraux et de clauses)
- le problème k-SAT étant $\mathcal{NP} - complet$ la recherche de solution se fera en général par énumération des cas, ce qui sera dépendant de l'ordre dans lequel on évalue les variables

C'est un algorithme ultra-simple, mais dans un contexte restreint.

Exercice SUM

Soit $\mathcal{E}$ un ensemble de n éléments. À chaque élément i de $\mathcal{E}$ on associe une valeur entière v_i. Pour éviter les cas particuliers on supposera les v_i distincts.
On recherche dans un premier temps (Questions 1,2,3) tous les couples (i, j) avec $i \neq j$ tels que

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

ou *pas de couple* s'il n'en existe pas.
Dans un deuxième temps, on recherche maintenant tous les sous-ensembles de $\mathcal{E}$ tels que la somme des valeurs des éléments du sous-ensemble soit égale à S, sans contrainte sur la taille des sous-ensembles. Un peu d'étude bibliographique indique que ce problème est dans $\mathcal{NP}$. (Questions 4, 5, 6)

Question 1

Écrire un algorithme naïf qui résout le problème en $\mathcal{O}(n^2)$ opérations.

Pour la force brute, on examine toutes les associations possibles de i et j, on peut prendre évidemment $i < j$ car la formule est symétrique en i et j. L'algorithme est de coût au pire $\frac{n(n-1)}{2}$ soit du $\mathcal{O}(n^2)$.

BRUTE-FORCE(T,n,S)

Data: Un tableau T de n entiers, et S un entier

Output: Un couple (i, j) distincts tel que $T[i] + T[j] = S$ ou *pas de couple* s'il n'en existe pas

```
1 for i = 1 to n − 1 do
2     for j = i + 1 to n do
3         if T[i] + T[j] = S
4             Retourne (i, j)
5 Retourne pas de couple
```

C'est au programme de L1 (ou de seconde).

Question 2

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

Pour avoir un algorithme en $\mathcal{O}(n \log n)$, on peut regarder ce qu'apporterait le fait que T soit trié. Pour i fixé, si T est trié la recherche d'un élément j tel que $T[i] + T[j] = S$, c'est à dire $T[j] = S - T[i]$ revient à faire une recherche dichotomique sur $T[i+1, n]$. Le coût de l'algorithme revient donc à trier les éléments, ce qui se fait en $\mathcal{O}(n \log n)$ opérations, puis pour tout i la recherche dichotomique se fait en $\mathcal{O}(\log n)$ opérations donc finalement la recherche se fait en $\mathcal{O}(n \log n)$ opérations et l'algorithme est bien en $\mathcal{O}(n \log n)$ opérations.

Ici il fallait un peu d'imagination, penser au tri. Ensuite utiliser la recherche dichotomique. Chacune des 2 opérations a le même coût.

Question 3

En utilisant une structure de donnée adaptée, écrire un algorithme qui résout le problème en $\mathcal{O}(n)$ opérations.

Là il faut un peu d'intuition, en cours, on a vu que la recherche pouvait se faire en $\mathcal{O}(1)$ avec une structure de donnée adaptée, dans notre cas une table de hachage. On suppose donc que l'on dispose d'une fonction de hachage h dans une table H.

```
FINESSE(T,n,S)
    Data: Un tableau T de n entiers, et S un entier
    Output: Un couple (i, j) distincts tel que T[i] + T[j] = S ou pas de couple s'il
            n'en existe pas
6   for i = 1 to n do
7       insere(i,T[i],H)                    // insère i à l'adresse h(T[i])
8   for i = 1 to n do
        // On cherche pour tout i si S − T[i] est dans la table
9       if conflit h(S − T[i])
10          if il existe un élément j différent de i haché vers h(S − T[i]) et T[i] +
            T[j] = S
11              Affiche (i, j)
12  Retourne pas de couple
```

Question 3 (suite)

Si la table de hachage H est suffisamment grande (plus grande que n et que la fonction de hachage a de bonnes propriétés d'uniformité le nombre de tests effectués à chaque itération de la deuxième boucle est en $\mathcal{O}(1)$. D'où un algorithme en $\mathcal{O}(n)$!

Dans cet exercice il fallait avoir compris le principe des tables de hachage et comment l'utiliser. Par contre on hache ici sur des valeurs, et on utilise la propriété de la somme.

Question 4

Cas où toutes les valeurs $\{v_i\}$ sont positives. En vous inspirant de l'algorithme d'énumération des parties proposer un algorithme qui visite tous les sous-ensembles de somme S. On précisera avec soin les conditions sous lesquelles on effectuera les appels récursifs.

procédure `énumérer_visiter_parties_S` (X, Y)
Data: X, Y ensembles d'éléments
Output: Une et une seule visite de chaque partie $p \cup Y$ avec $p \in \mathcal{P}(X)$

if X **est vide**
  if Condition A `Visiter` (Y)
else
  `Choisir` (X, x) `// extrait` x `de` X
  if Condition B_1 `énumérer_visiter_parties_S` (X, Y)
  if Condition B_2 `énumérer_visiter_parties_S` $(X, Y \cup \{x\})$

L'appel initial est `énumérer_visiter_parties_S`$(\mathcal{E}, \emptyset)$.

Sans les conditions A, B_1 et B_2 cet algorithme visite toutes les parties de $X \cup Y$ contenant Y (c'est du cours)

Université Joseph Fourier UFR IM²AG

Question 4 (solution brutale)

Une première méthode consiste à énumérer toutes les parties et lorsque l'on s'arrête on visite la partie si la somme est S. C'est à dire que la condition A ligne 1 serait que la somme des éléments de Y est égale à S et les conditions B_1 et B_2 seraient toujours vraies.
Cette solution est correcte car l'algorithme énumérer-visiter est correct et que seule les parties de somme S sont visitées par la condition A.

Question 4 (solution plus fine)

On peut bien sur améliorer cette première solution en modifiant les conditions de branchement B_1 et B_2.
Comme on suppose toutes les valeurs des éléments positives, on va dans un premier temps définir une fonction `somme` qui prend en argument un ensemble et retourne la valeur de la somme des valeurs de ses éléments.
L'invariant que l'on va chercher à propager lors des appels récursifs est

$$\texttt{somme}(Y) \leqslant S \leqslant \texttt{somme}(X) + \texttt{somme}(Y).$$

On peut ainsi exprimer la condition B_1 par $\texttt{somme}(X) + \texttt{somme}(Y) > S$, c'est à dire que si cette condition n'est pas vérifiée alors on ne pourra pas trouver de partie de somme S, ce n'est donc pas la peine de parcourir cette branche.
Pour la condition B_2, on va ajouter x à Y, c'est la première inégalité qui doit être préservée, c'est à dire

$$\texttt{somme}(Y) + \texttt{valeur}(x) \leqslant S.$$

Ce n'est pas la peine d'explorer la branche lorsque la somme des éléments de $Y \cup x$ est strictement supérieure à S.

Université Joseph Fourier
UFR IM²AG

Question 5

Écrire la preuve de votre algorithme.

On s'inspire de la preuve faite pour les parties de cardinal k.
On prouve que `énumérer_visiter_parties_S` (X, Y) visite toutes les parties de $X \cup Y$, contenant Y et de somme S (propriété $\mathcal{P}$) avec comme pré-condition de l'appel

$$\texttt{somme}(Y) \leqslant S \leqslant \texttt{somme}(X) + \texttt{somme}(Y) \text{ et } X \cap Y = \emptyset.$$

- **Correction partielle**
  - Pour le cas de base, lorsque X est vide la pré-condition indique que $\texttt{somme}(Y) = S$ et donc que Y est visité et c'est la seule partie de $X \cup Y$ contenant Y et de somme S.
  - Supposons que les appels dans le corps de la procédure vérifient $\mathcal{P}$ (raisonnement par induction).
    Si on n'est pas dans le cas de base, X est non vide. Il existe donc un élément x dans X que l'on choisit.
    Cet élément étant fixé, l'ensemble des parties de $X \cup Y$, contenant Y et de somme S se décompose en une union disjointe de $\mathcal{E}_{x-}$ ensemble des parties de $X \cup Y$, contenant Y, de somme S et ne contenant pas x et de l'ensemble $\mathcal{E}_{x+}$ des parties de $X \cup Y$, contenant Y, de somme S et contenant x.
    Or si $\texttt{somme}(X) - \textit{valeur}(x) + \texttt{somme}(Y) < S$ (x a été extrait de X), ce n'est pas la peine de rechercher une telle partie dans $\mathcal{E}_{x-}$, donc l'instruction de la ligne 2, en utilisant l'hypothèse d'induction $\mathcal{P}$, visite les toutes parties de $\mathcal{E}_{x-}$ de somme S...
    De même, si $\texttt{somme}(X) + \texttt{somme}(Y) < S$, ce n'est pas la peine de rechercher une telle partie dans $\mathcal{E}_{x+}$, donc l'instruction de la ligne 3, en utilisant l'hypothèse d'induction $\mathcal{P}$, visite les toutes parties de $\mathcal{E}_{x+}$ de somme S...
    Le corps de la procédure visite une et une seule fois (union disjointe de toutes les parties de $\mathcal{E}_{x-}$ et $\mathcal{E}_{x+}$) toutes les parties ...

  Ainsi, si l'appel se termine, `énumérer_visiter_parties_S`(X, Y) aura visité toutes les parties vérifiant $\mathcal{P}$

Université Joseph Fourier UFR IM²AG

Question 5 (suite)

- **Terminaison** Pour la terminaison, on utilise le variant $|X|$, nombre entier qui diminue strictement à chaque appel récursif. Donc l'arbre des appels est fini et l'appel se termine.

Enfin, la procédure récursive étant correcte et se terminant, `énumérer_visiter_parties_S`$(\mathcal{E}, \emptyset)$ visite toutes les parties de $\mathcal{E}$ de somme S. Dans ce cas, on utilise le fait que les valeurs sont positives pour couper lorsque l'on dépasse S, ce qui ne sera pas possible avec des valeurs positives et négatives.

On pouvait écrire plus rapidement la preuve, mais il faut mettre en avant l'invariant, le cas de base, l'induction, le fait que l'invariant est vérifié avant l'appel de la procédure.

Question 6

Cas où les valeurs v_i peuvent être positives ou négatives, adapter l'algorithme précédent en changeant les conditions d'appels récursifs.

les conditions de coupe sont plus difficiles à établir...
essayez de trouver des conditions

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...

Université Joseph Fourier UFR IM²AG