Colle 3-Algorithmes probabilistes et d'approximation

21 janvier

1 Algorithmes probabilistes

On donne ici quelques éléments de syntaxe C et Ocaml au sujet de la génération (pseudo-)aléatoire de nombres dans ces deux langages. Dans les deux cas, il faut commencer par initialiser le générateur de nombres pseudo-aléatoires (PRNG : pseudorandom number generator) à l'aide d'une graine. Il y a deux façons de faire cette initialisation, à choisir selon le comportement souhaité :

- Initialiser le générateur avec une graine connue fixe. Dans ce cas, à chaque exécution de votre code, les choix aléatoires effectués seront toujours les mêmes et votre algorithme probabiliste se comportera comme un algorithme déterministe. Ce comportement est utile lors des phases de debug.
- Initialiser le générateur avec une graine qui changera à chaque exécution. Dans ces conditions, votre algorithme probabiliste fera des choix différents à chaque exécution (c'est le comportement souhaité une fois le debug effectué). En Ocaml, l'appel à `Random.self_init ()` permet d'obtenir ce comportement. En C, on fournit généralement au générateur pseudo aléatoire la date courante pour l'obtenir ; cette dernière se récupère à l'aide de `time(NULL)` et nécessite d'importer `time.h`.

Récapitulatif des commandes qui peuvent vous être utiles :

Action	Ocaml	C
Initialiser le générateur avec une graine fixe s	`Random.init s`	`srand(s)`
Initialiser le générateur avec une graine changeante	`Random.self_init ()`	`srand(time(NULL))`
Tirer un entier entre 0 inclus et n exclus	`Random.int n`	`rand() % n`
Tirer un booléen	`Random.bool ()`	`rand() % 2 == 0`
Tirer un flottant entre 0 et 1 inclus	`Random.float 1.`	`(float)rand() / RAND_MAX`

Pour retrouver les commandes adaptées en C, il suffit de se souvenir du fonctionnement de la fonction `rand` : cette fonction ne prend pas d'entrée et génère un entier aléatoirement entre 0 et une valeur maximale fixée dont le nom est `RAND_MAX` et dont la valeur dépend de la machine.

2 N-reines : backtracking et Las Vegas

On s'intéresse au problème des N reines. Il s'agit de placer sur un échiquier de taille $N \times N$, N reines sans qu'elles puissent se prendre l'une l'autre. Cette partie du TP sera à traiter en C. Une solution du problème sera représentée par un tableau de taille N où la case d'indice k contiendra le numéro de la colonne où se trouve la reine de la ligne k s'il en contient une (en effet une ligne contient au plus une reine).

On rappelle que la reine se déplace en ligne, en colonne ou en diagonale.

On va utiliser tout d'abord un algorithme de backtracking que l'on transformera en un algorithme de type Las Vegas. Le principe est le suivant : on essaie de positionner les reines ligne par ligne en vérifiant qu'une nouvelle reine positionnée ne peut pas prendre une reine préalablement positionnée.

1 Ecrire une fonction `bool check(int n, int sol[], int k)` qui teste si la position de la reine dans la ligne `k` est compatible avec celles des reines en lignes `0` à `k-1`. On suppose que les `k` premières lignes sont convenablement remplies et on vérifie si la keme prolonge la solution partielle ou non. La solution en cours de construction est stockée dans le tableau `sol`.

2 Ecrire une fonction `bool solve(int n, int sol[], int k)` qui teste si une solution partielle avec une grille remplie jusqu'à la ligne `k-1`, stockée dans `sol`, est prolongeable. La fonction sera récursive et complètera la solution au passage.

3 On veut tirer une colonne possible de manière uniforme dans l'ensemble des colonnes acceptables pour la ligne `k`.

On utilise la procédure suivante :

```
t=0;
col_choisie=0;
pour chaque colonne v :
   sol[k]=v;
   si (check(n,sol,k)) :
        t++;
       col_choisie = v avec proba 1/t
si (t==0) false
sinon sol[k]=col_choisie;
```

Justifier que cette procédure va choisir une colonne compatible de manière uniforme.

4 Ecrire une fonction `bool solve_prob(int n, int sol[], int k)` qui est une adaptation de `solve` où au lieu de prendre toutes les colonnes de manière systématique, on en choisit une uniformément au hasard.

5 Est ce que ce programme va toujours renvoyer une solution quand elle existe? Ecrire une fonction de type Las Vegas qui cherche une solution au problème des N reines à partir de la fonction de la question précédente.