PROG
TP#6 : Permutations aléatoires

2022-W42

Le but de cet exercice est d'implémenter deux algorithmes permettant de générer une permutation aléatoire (proche de l') uniforme des entiers de 1 à N (les deux algorithmes se généralisant tous deux facilement à n'importe quel ensemble de taille N).

Q.1 : On utilise d'abord l'algorithme suivant : 1) on initialise un tableau avec N paires (r_i, i) où i prend les N valeurs dans $[\![1, N]\!]$ et où les r_i sont des nombres tirés uniformément dans un ensemble de taille N' [1] ; 2) on trie la liste suivant les valeurs des r_i ; 3) on retourne les valeurs i dans l'ordre trié [2]. On peut montrer que quand $N' \approx N^2$, la distribution des permutations générées par cet algorithme est proche de l'uniforme (pour une certaine définition de « proche »).

Implémentez l'algorithme ci-dessus pour $N < 2^{64}$ dans une fonction :

```
void shuffle1(size_t nelem, uint64_t p[nelem]);
```

où `p` est alloué en dehors de la fonction, et devra contenir à l'issue de son exécution une permutation des entiers entre 1 et `nelem`. Cette fonction devra utiliser (de façon interne) la structure :

```
struct uints
{
        uint64_t key;
        uint64_t dat;
};
```

pour représenter les paires d'éléments et la fonction `qsort` de la bibliothèque standard pour effectuer le tri.

N'oubliez pas de tester votre fonction (par exemple sur de petits cas), et de vérifier l'absence de bugs facilement détectables ainsi que de fuites mémoire.

Q.2 :

1. Mesurez le temps d'exécution (par exemple en utilisant `time` en ligne de commande, ou la fonction `gettimeofday`) pour des permutations de 1 000 000, 10 000 0000, 20 000 000, 40 000 000, 80 000 000, 160 000 000 éléments.
2. Refaites les mêmes tests avec une autre implémentation du tri (si disponible sur votre machine), par exemple `heapsort`.

Pour chacun des points ci-dessus, quels commentaires pouvez-vous faire ?

1. Par exemple pour $N = 4$, $N' = 16$, un tableau possible serait $[(7, 1); (3, 2); (13, 3); (11, 4)]$.
2. Pour le tableau de l'exemple précédent, on obtiendrait donc $[2, 1, 4, 3]$.

On s'intéresse maintenant à l'algorithme suivant, donné en pseudo-code :

```
1       /****************************************
2       input: T = [0,...,N-1]
3       ****************************************/
4       for (i = 0; i < N - 1)
5       {
6               s = uniform_random(i, N-1);
7               swap(T[i], T[s]);
8       }
```

Q.3 : Implémentez l'algorithme ci-dessus dans une fonction `shuffle2` similaire à `shuffle1`.

Mesurez le temps d'exécution comme dans **Q.2** et comparez les résultats. Quelles raisons peuvent expliquer les éventuelles différences ?

Q.4⋆ : Montrez (par exemple par induction) que le tirage effectué par ce second algorithme est uniforme (à condition que le tirage d'aléa à la ligne 6 soit lui-même uniforme).