Licence 2 — Université de Nantes
Algorithmique et Structures de données 1 — 2016/2017

Feuille de travaux pratiques n° 2
Chronométrage et complexité (temporelle) pratique

2 heures

L'objectif de ce TP est de mettre en œuvre la démarche d'évaluation des performances de l'implémentation en programmes C++ d'un algorithme. Cela passe par le monitoring du processeur et de la mémoire vive de la machine exécutant le programme. La mesure de la quantité de mémoire vive (RAM) utilisée par un programme est possible mais requiert des fonctionnalités systèmes bas niveau et peu portables ; nous nous limiterons donc à des mesures temporelles dans ce TP.

Le programme `tp2.cpp` disponible sur Madoc illustre les notions présentées ci-dessous. Après l'avoir récupéré, engendrez sa documentation avec doxygen afin de la consulter parallèlement à la lecture de ce sujet.

1 Chronométrage

C++ n'offre pas de mécanisme permettant de mesurer les performances d'un programme à la façon dont vous le feriez pour un algorithme, en comptant le nombre d'opérations réalisées (ou d'unités de temps t). Pour y parvenir, il faudrait modifier le programme pour réaliser manuellement ce calcul, ce qui est fastidieux et source d'erreur.

Au lieu de cela, nous nous contenterons en TP de chronométrer les programmes. Le temps d'exécution d'un programme dépend bien sûr de la machine, du système d'exploitation, du langage de programmation, de la charge de la machine au moment de l'exécution, etc. Il doit toutefois rester proportionnel au coût de l'algorithme en unité de temps et refléter ainsi la complexité théorique de l'algorithme qu'il implémente.

Pour chronométrer un programme, on peut utiliser la fonction `clock` disponible dans la bibliothèque C `time.h` (`#include` <ctime> en préambule de votre programme). Cette fonction sans paramètre retourne un entier qui représente le nombre de *ticks* d'horloge processeur passés à exécuter votre programme, excluant donc les temps d'interaction utilisateur, d'accès disque, ou d'autres opérations liées au système. Pour connaître le temps correspondant, il faut diviser ce nombre par la constante `CLOCKS_PER_SEC` (également définie dans la bibliothèque `time.h`) qui représente, pour votre machine et votre système, le nombre de ticks d'horloge par seconde.

Afin de rendre son utilisation plus simple, nous définissons les *macros* C++ suivantes (cf. `tp2.cpp:25-41`) :

```
#define START chrono=clock()
```

```
#define STOP chrono=clock()−chrono;
#define TEMPS double(chrono)/CLOCKS_PER_SEC
```

Ces macros nécessitent la définition d'une variable globale `chrono` (cf. `tp2.cpp:23`) de type `clock_t` (également défini dans la bibliothèque `time.h`). Cette variable n'est jamais utilisée directement.

Pour chronométrer un programme (ou une partie de celui-ci), il suffit alors d'utiliser la macro `START` au début pour lancer le chronomètre ; de l'arrêter à la fin avec la macro `STOP` ; et de consulter le temps chronométré (en secondes) avec la macro `TEMPS` (cf. `tp2.cpp:188-191`).

Attention : la fonction `clock` a une précision qui est limitée par la représentation des entiers en C++ : un temps inférieur à quelques millisecondes ou supérieur à une heure ne pourra être chronométré correctement au moyen de cette fonction.

2 Performance pratique VS complexité théorique

Afin de vérifier en pratique que les performances d'un programme correspondent à la complexité théorique de l'algorithme qu'il implémente, il faut réaliser un *programme de test de performance*. Pour cela, il est préférable de toujours faire du programme à analyser une routine paramétrée par les données attendues, le programme principal étant consacré au test de performance qui comporte trois éléments :

- *Performance en meilleur cas :* la routine à analyser est appelée sur une donnée vérifiant la condition de forme des données en meilleur cas et les performances sont chronométrées (cf. `tp2.cpp:185-191`). Elles doivent être comparées à la complexité en meilleur cas de l'algorithme.
- *Performance en pire cas :* la routine à analyser est appelée sur une donnée vérifiant la condition de forme des données en pire cas et les performances sont chronométrées (cf. `tp2.cpp:194-200`). Elles doivent être comparées à la complexité en pire cas de l'algorithme.
- *Performances en moyenne :* la routine à analyser est appelée sur une donnée de forme aléatoire et les performances sont chronométrées (cf. `tp2.cpp:203-225`). Comme les performances peuvent varier selon la donnée testée dès lors que le meilleur et pire cas de l'algorithme sont distincts, ce test doit être répété afin qu'une analyse statistique valable du programme puisse être établie. Les indicateurs statistiques pertinents sont la performance maximum (max), minimum (min) et moyenne (moy) sur toutes les répétitions du test. Ces répétitions et le calcul des indicateurs sont automatisés dans une boucle qui répète le test sur une donnée aléatoire à chaque fois (cf. `tp2.cpp:210-220`). Plus les répétitions sont nombreuses, plus les indicateurs sont fiables. Cependant, le temps pris par cette analyse statistique peut devenir rédhibitoire si elles sont trop nombreuses. Les résultats obtenus doivent être comparés à la complexité théorique en meilleur et en pire cas de l'algorithme. Il est en particulier intéressant de noter si le meilleur/pire cas théorique est effectivement atteint lors des répétitions (indicateurs min et max) et si la performance moyenne moy se situe plus proche du meilleur ou du pire cas théorique.

Lorsque la complexité théorique dépend de la taille de la donnée, l'analyse peut être répétée pour des tailles croissantes afin de vérifier que les performances pratiques suivent les fonctions de complexité établies pour l'algorithme. Il est alors commode de reporter les indicateurs statistiques mesurés dans un tableur afin d'en tirer un graphique dont l'abscisse représente la taille de la donnée et l'ordonnée le temps, éventuellement en échelle(s) logarithmique(s) si les variations

sont importantes. La superposition sur un tel graphique de la courbe de complexité temporelle théorique permet ainsi la vérification des résultats pratiques (et théoriques).

Exercice 2.1

1. Compilez et exécutez plusieurs fois le programme fourni. Observez que les résultats fluctuent.
2. Expliquez et tentez de corriger ce problème.
3. Exécutez le programme pour des tailles de données croissantes et relevez les résultats dans un tableur.
4. Comparez-les à la complexité théorique de l'algorithme de recherche.
5. Enfin, écrivez un programme de test de performances pour d'autres algorithmes vus en cours et/ou TD.