INF3105 – Structures de données et algorithmes
Examen final – Été 2014

Éric Beaudry
Département d'informatique
Université du Québec à Montréal

Lundi 28 juillet 2014 – 17h30 à 20h30 (3 heures) – Local PK-R610

Instructions

- Aucune documentation n'est permise excepté l'aide-mémoire C++ (une feuille recto verso).
- Les appareils électroniques, incluant les téléphones et les calculatrices, sont strictement interdits.
- Répondez directement sur le questionnaire à l'intérieur des endroits appropriés.
- Pour les questions demandant l'écriture de code :
  - le fonctionnement correct, la robustesse, la clarté, l'efficacité (temps et mémoire) et la simplicité du code sont des critères de correction à considérer ;
  - vous pouvez scinder votre solution en plusieurs fonctions à condition de donner le code pour chacune d'elles ;
  - vous pouvez supposer l'existence de fonctions et de structures de données raisonnables ;
- Aucune question ne sera répondue durant l'examen. Si vous croyez qu'une erreur ou qu'une ambigüité s'est glissée dans le questionnaire, indiquez clairement la supposition que vous avez retenue pour répondre à la question.
- L'examen dure 3 heures et contient 5 questions.
- Ne détachez pas les feuilles du questionnaire, à l'exception des annexes à la fin.
- Le côté verso peut être utilisé comme brouillon. Des feuilles additionnelles peuvent être demandées.

Identification

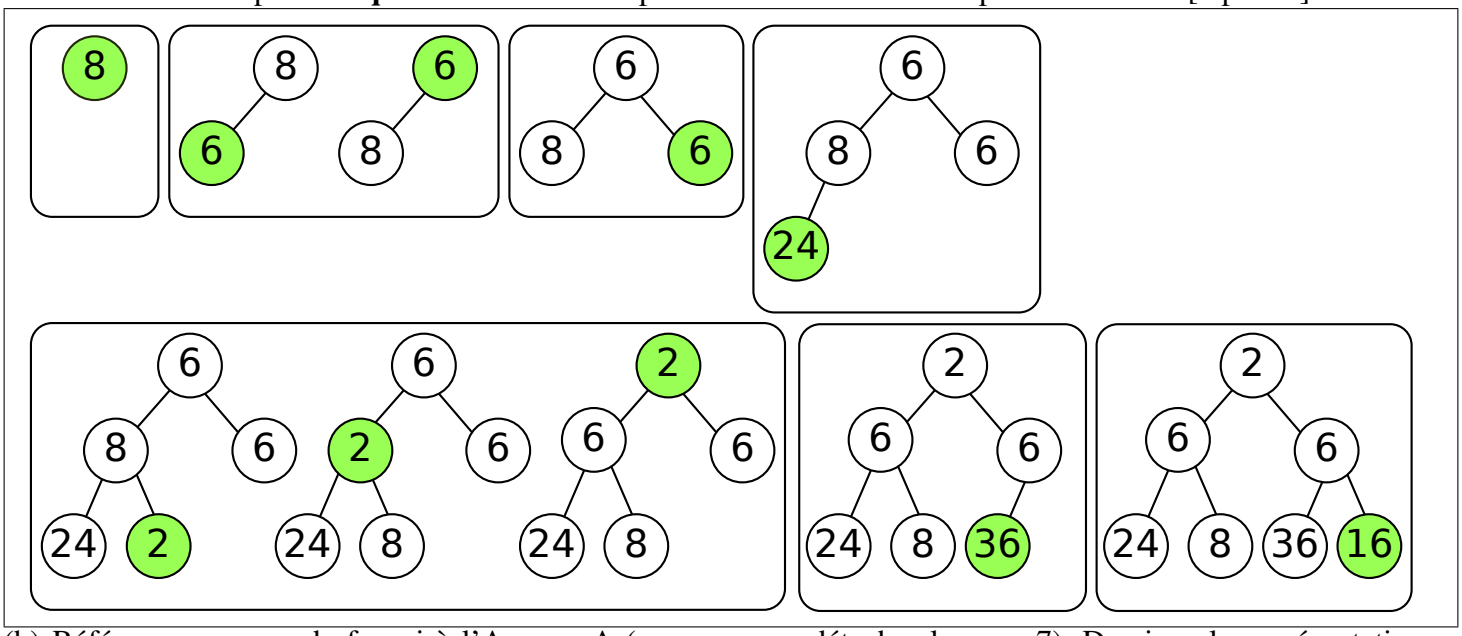
Nom : Solutionnaire

Résultat

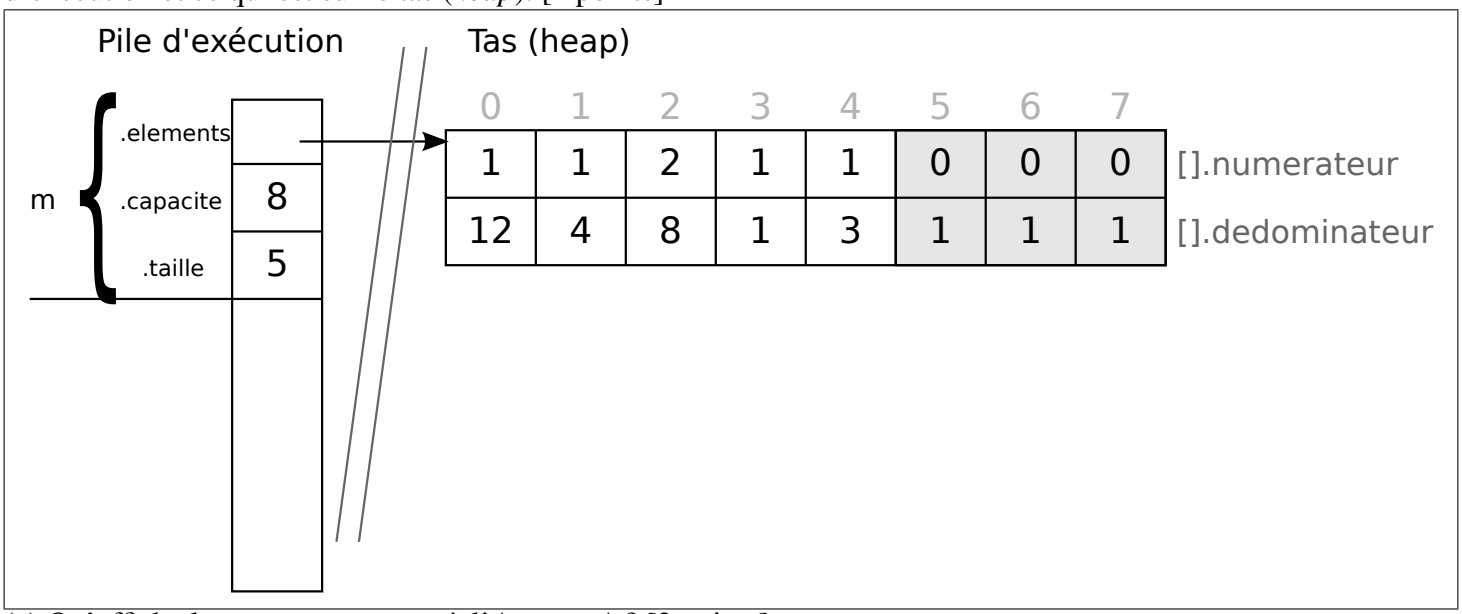
Q1		/ 6
Q2		/ 6
Q3		/ 5
Q4		/ 5
Q5		/ 3
Total		/ 25

1 Monceaux (*Heaps*) et connaissances techniques de C++ [6 points]

(a) Insérez les entiers 8, 6, 6, 24, 2, 36 et 16 dans un monceau initialement vide. Dessinez sous forme d'arbre l'état du monceau après **chaque** insertion. Les étapes intermédiaires ne sont pas demandées. [2 points]



(b) Référez-vous au code fourni à l'Annexe A (vous pouvez détacher la page 7). Dessinez la représentation en mémoire de l'objet `m` de la fonction `main` rendue à la ligne 29. Indiquez clairement ce qui est sur la pile d'exécution et ce qui est sur le tas (*heap*). [2 points]



(c) Qu'affiche le programme `main` à l'Annexe A ? [2 points]

```
1/12 1/4 2/8 1/3 1/1
```

2 Table de hachage (*Hashtable*) [6 points]

(a) Complétez le tableau suivant en indiquant la complexité temporelle des opérations dans une table de hachage contenant n entrées (paires clé-valeur). Supposez une gestion de collisions utilisant une liste chaînée comme structure externe. Utilisez la notation grand O. [1 point]

Opération	Cas moyen	Pire Cas
Insertion d'une nouvelle entrée clé-valeur	$O(1)$	$O(n)$
Retourner la valeur associée à une clé	$O(1)$	$O(n)$

Pour les sous-questions suivantes, considérez l'annexe B (vous pouvez détacher la page 8).

(b) Que **pourrait** afficher l'exécution `./q2 < q2.txt` ? Plusieurs réponses possibles. [1 point]

```
jaune 3
vert 7
bleu 12
rouge 3
```

L'ordre d'affichage n'est pas garanti. Ça dépend entre autres de l'implémentation de la fonction de hachage. Notez qu'un `unordered_map` gère les collisions. Ainsi, même si deux chaînes ont la même clé réduite, elles sont comparées et considérées comme des entrées distinctes.

(c) Supposez que le fichier d'entrée contienne n lignes. Chaque ligne contient une chaîne de caractères différente et un entier différent. Quelle serait la complexité temporelle en notation grand O ? [1 point]

Cas moyen : $O(n)$. L'expression «`um[s]`» (ligne 13) coûte $O(1)$ à chaque chaîne lue. Les mots étant tous différents, chaque monceau (`priority_queue`) sera initialement vide et on y ajoutera qu'un seul entier. Ainsi, chaque ajout (`push`, ligne 13) coûte $O(1)$. Multiplié par n itérations, cela fait $O(n)$.
Pire cas : $O(n^2)$. Dans une table de hachage, le pire cas survient lorsque toutes les entrées ont des clés différentes, et qu'elles tombent dans le même casiers (clés $\rightarrow$ fonction de hachage $\rightarrow$ fonction de réduction (ex. : modulo taille du tableau) $\rightarrow$ même valeur). Dans ce cas, les opérations de recherche, d'insertion et d'enlèvement dégénèrent en $O(n)$. Ainsi, l'insertion de n entrées dégénère en $O(n^2)$.

(d) Supposez que le fichier d'entrée contienne n lignes. Toutes les lignes contiennent la même chaîne de caractères et un entier différent. Quelle serait la complexité temporelle en notation grand O ? [1 point]

Cas moyen : $O(n \log n)$ **Pire cas :** $O(n \log n)$
Dans le pire cas et le cas moyen, puisque toutes les lignes ont la même clé, le dictionnaire `um` n'aura qu'une seule entrée. Ainsi, l'expression «`um[s]`» (ligne 13) coûte toujours $O(1)$. Contrairement à la sous-question précédente, il n'y a donc aucun enjeu lié à la table de hachage.
Ensuite, on peut observer que les entiers seront tous insérés dans le même monceau. L'insertion dans un monceau coûte $O(\log n)$. L'insertion de n nombres dans un monceau coûte $O(n \log n)$.
Enfin, il est à noter que le pire cas n'est pas pire que le cas moyen.

(d) Améliorez la fonction `main` en modifiant les lignes 7, 13 et 15-22. [2 points]

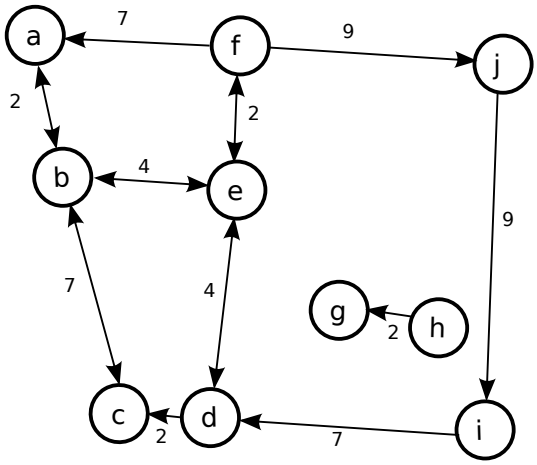
Dans le programme à l'annexe B, le monceau n'est aucunement utile, car au final, c'est la somme des entiers qui est affichée. Pour améliorer le programme, il suffit de remplacer le `priority_queue` par un entier.

```
7. unordered_map<string, int> um;
13. um[s] += i;
15. unordered_map<string, int>::const_iterator i = um.begin();
16. while(i != um.end()) {
17.   cout << i->first << " " << i->second << endl;
18.   ++i;
19.}
```

On aurait pu envisager remplacer le `unordered_map` par un `map` (arbre rouge-noir). Bien que le pire cas en (c) aurait réduit à $O(n \log n)$, le cas moyen aurait augmenté à $O(n \log n)$.

3 Graphes / Algorithmes de base [5 points]

Considérez le graphe ci-dessous. En (a) et (b), les arêtes sortantes d'un sommet doivent être parcourues en ordre alphabétique de leur sommet d'arrivée.



(a) Écrivez l'ordre de visite des sommets d'un parcours en **profondeur** à partir du sommet **j**. [1 point]

$\langle j, i, d, c, b, a, e, f \rangle$

(b) Écrivez l'ordre de visite des sommets d'un parcours en **largeur** à partir du sommet **j**. [1 point]

$\langle j, i, d, c, e, b, f, a \rangle$

(c) Énumérez les composantes **fortement connexes** du graphe. Indiquez uniquement les sommets. [1 point]

$\{\{a, b, c, d, e, f, j, i\}, \{g\}, \{h\}\}$

(d) Simulez l'algorithme de Dijkstra pour calculer le plus court chemin de d à a. Il y a plusieurs façons de présenter votre réponse. L'important est de démontrer votre compréhension de l'algorithme. Les éléments clés à présenter sont l'ordre de visite des sommets et les valeurs *Dist* et *Parent*. [2 points]

	choix	a	b	c	d	e	f	g	h	i	j	File prioritaire
#0	(init)	$+\infty$	$+\infty$	$+\infty$	$0/-$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	d
#1	d	$+\infty$	$+\infty$	$2/d$	$0/-$	$4/d$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	c,e
#2	c	$+\infty$	$9/c$	$2/d$	$0/-$	$4/d$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	e,b
#3	e	$+\infty$	$8/e$	$2/d$	$0/-$	$4/d$	$6/e$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	f,b
#4	f	$13/f$	$8/e$	$2/d$	$0/-$	$4/d$	$6/e$	$+\infty$	$15/f$	$+\infty$	$+\infty$	b,a,j
#5	b	$10/b$	$8/e$	$2/d$	$0/-$	$4/d$	$6/e$	$+\infty$	$15/f$	$+\infty$	$+\infty$	a,j
#6	**a**	$10/b$	$8/e$	$2/d$	$0/-$	$4/d$	$6/e$	$+\infty$	$15/f$	$+\infty$	$+\infty$	a,j

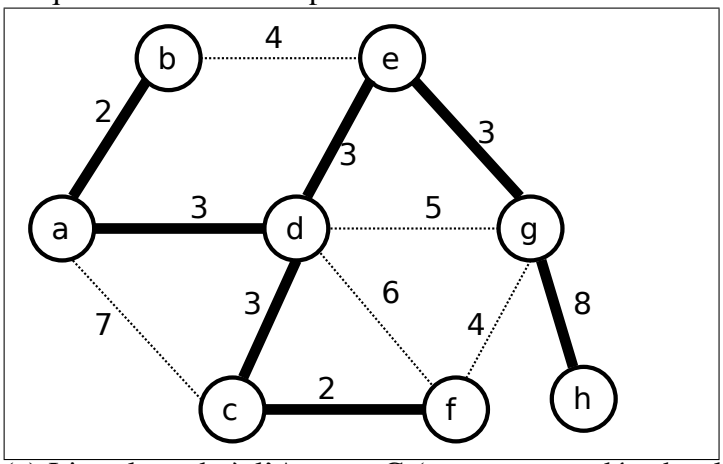
Chemin : $\langle (d,e), (e,b), (b,a) \rangle$
Coût : 10

4 Graphes / Arbre de recouvrement à coût minimal (ARM) [5 points]

(a) Donnez un exemple d'application où il est utile de calculer un ARM. Expliquez brièvement. [1 point]

Réseau de télécommunication.
Minimiser le coût pour relier n sites.
Sommets : sites à relier.
Arêtes : liens potentiels ;
Objectifs : minimiser le coût pour relier n sites.

(b) Calculez l'arbre de recouvrement minimal du graphe suivant en utilisant l'algorithme de Prim-Jarnik. Indiquez clairement l'ordre dans lequel les sommets et arêtes ont été sélectionnés. La réponse n'est pas forcément unique. Maximum de 1 point si vous faites la trace de Kruskal plutôt que Prim-Jarnik. [2 points]



Choix d'un sommet arbitaire : a.

1. (a,b)
2. (a,d)
3. (d,c)
4. (c,f)
5. (d,e)
6. (e,g)
7. (g,h)

(c) Lisez le code à l'Annexe C (vous pouvez détacher la page 9). La fonction `Carte::calculerCoutARM` est incomplète. Telle que présentée, cette fonction ne fonctionne pas et retourne toujours zéro (0). Donnez les modifications nécessaires pour rendre cette fonction fonctionnelle et correcte. Indiquez clairement les endroits concernés à l'aide de numéros de ligne. [2 points]

On **remplace** la ligne 22 par :

```
q.push(A(sommets.begin()->first)); // d vaudra 0.0
```

5 Résolution d'un problème [3 points]

Postes Canada vous mandate pour écrire un programme qui détermine l'emplacement optimal où installer une boîte postale sur une carte. La boîte postale doit être installée sur un nœud de la carte. L'emplacement retourné doit minimiser la distance moyenne de marche vers tous les nœuds de la carte. Ne vous souciez pas des sens uniques. À titre d'exemple, dans le graphe à la Question 4(b), l'emplacement optimal est d. Vous devez utiliser la représentation de la classe `Carte` à l'Annexe C. Continuez au verso si vous manquez d'espace.

```
1  string Carte::question5()const{
2      string meilleur;
3      double dm=numeric_limits<double>::infinity();
4      for(map<string, Sommet> i=sommets.begin();i!=sommets.end();i++){
5          double d = calculerDistMoyenne(i->first);
6          if(d<dm){
7              dm=d;
8              meilleur=i->first;
9          }
10     }
11     return meilleur;
12 }
13 double Carte::calculerDistMoyenne(const string& origine){
14     // solution non publique
15
16
17
18
19
20
21
22 }
```

Annexe A pour la Question 1

Cette page peut être détachée.

```
1  /* monceau.h  |  classe Monceau telle que presentee dans le cours */
2  template <class T> class Monceau {
3    public:
4      void inserer(const T&);
5      T enleverMinimum();
6      bool estVide() const;
7    private:
8      Tableau<T> valeurs; // Rappel Tableau : pointeur, capacite, taille
9      // ...
10 };
11 // ...
```

```
1  /* question1b.cpp */
2  #include <monceau.h>
3  #include <iostream>
4  using namespace std;
5  class Fraction{
6    public:
7      Fraction(int n=0, int d=1);
8      bool operator<(const Fraction& f) const;
9    private:
10     int numerateur, denominateur;
11   friend ostream& operator << (ostream& os, const Fraction& f);
12 };
13 Fraction::Fraction(int n=0, int d=1)
14  : numerateur(n), denominateur(d)
15 {} // <<== notez que le corps du constructeur est vide
16 bool Fraction::operator<(const Fraction& f)const{
17   return numerateur * f.denominateur < f.numerateur * denominateur;
18 }
19 ostream& operator << (ostream& os, const Fraction& f){
20   cout << f.numerateur << "/" << f.denominateur; return os;
21 }
22 int main(){
23   Monceau<Fraction> m;
24   m.inserer(Fraction(1,3));
25   m.inserer(Fraction(1,4));
26   m.inserer(Fraction(2,8));
27   m.inserer(Fraction(1));
28   m.inserer(Fraction(1,12));
29   /* Ici pour Question 1(b) */
30   while(!m.estVide())
31     cout << m.enleverMinimum() << '\t';
32   cout << endl;
33   return 0;
34 }
```

Annexe B pour la Question 2

Cette page peut être détachée.

```cpp
/* q2.cpp */
#include <iostream>
#include <queue> // priority_queue implemente un monceau (heap)
#include <unordered_map> // table de hachage (dictionnaire)
using namespace std;
int main(){
    unordered_map<string, priority_queue<int> > um;
    while(cin && !cin.eof()){
       string s;
       int i;
       cin >> s >> i >> ws;  // lit une chaine s et un entier i
       // >> std::ws force lecture espace blanc pour detecter fin fichier
       um[s].push(i);
    }
    unordered_map<string, priority_queue<int> >::iterator iter = um.begin();
    while(iter!=um.end()){
        priority_queue<int>& pq = iter->second;
        int s=0;
        while(!pq.empty()){s+= pq.top(); pq.pop();}
        cout << iter->first << '\t' << s << endl;
        ++iter;
    }
    return 0;
}
```

Un fichier d'entrée contient n lignes. Chaque ligne contient une chaîne de caractères s et un entier i.
Exemple de fichier d'entrée `q2.txt` :

```
rouge 2
bleu 5
vert 6
bleu 7
jaune 3
rouge 1
vert 1
```

Exécution pour la sous-question (b) `q2.txt` :

```
g++ -o q2 q2.cpp
./q2 < q2.txt
```

Exemples d'entrée pour les sous-questions (c) et (d) :

a 1	a 1
c 3	a 3
b 2	a 2
z 9	a 9
...	...

Annexe C pour les questions 4 et 5

Cette page peut être détachée.

```
1  class Carte{
2    public:
3      double calculerCoutARM() const; // retourne le cout d'un ARM minimal
4      string question5() const;
5      // ...
6    private:
7      struct Sommet{
8        map<string, double> voisins; // noeud voisin -> distance
9      };
10     map<string, Sommet> sommets;
11 };
12 struct A{
13     A(const string& s_="", double d_=0) : s(s_),d(d_){}
14     string s;
15     double d; // poids
16     bool operator<(const A& a){return a.d<d;} // <<== Il n'y a pas d'erreur ici
17 }
18 // Cette fonction est incomplete et basee sur l'algorithme Prim-Jarnik
19 double Carte::calculerCoutARM() const{
20     priority_queue<A> q;
21     set<string> r;
22     r.insert(sommets.begin()->first); //->first retourne cle
23     double coutarm=0;
24     while(!q.empty()){
25        A a = q.top(); // .top() retourne une reference sur le plus prioritaire
26        q.pop(); // .pop() ==>enlever;  .push() ==>inserer
27        if(r.find(a.s)!=r.end()) continue; //si r contient a.s alors continuer ligne24
28        r.insert(a.s);
29        coutarm+=a.d;
30        map<string,double>::const_iterator iter=sommets[a.s].voisins.begin();
31        while(iter!=sommets[a.s].voisins.end()){
32          q.push(A(iter->first, iter->second)); // //->second retourne valeur
33          ++iter;
34        }
35     }
36     return coutarm;
37 }
```