/**** INF3105 / Quiz 2 du 30 novembre 2016 / Annexe pour le Groupe 30 ****/

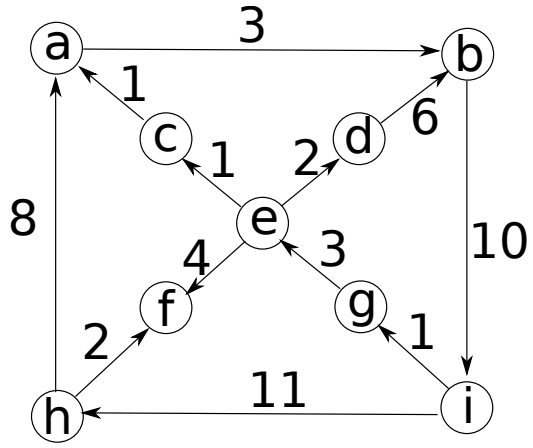
1 Monceaux (*heap*)

Considérez la structure monceau (tas ou *heap*) telle que présentée en classe. Celle-ci garde l'élément le plus petit (et non le plus grand) à la racine.

Pour les questions sur la complexité : à moins d'avis contraire (vide), supposez que le monceau contient n éléments.

2 Graphes | Simulation d'algorithmes sur le graphe B

Le graphe B est le suivant.



Pour les questions sur la simulation des algorithmes de recherche en profondeur et en largeur, supposez :

1. Les arêtes sortantes sont énumérées en ordre alphabétique. Par exemple, les arêtes sortantes du sommet e sont énumérérés dans l'ordre $\langle c, d, f \rangle$.

2. Un sommet est réputé être visité au moment où il est marqué visité. Cela correspond à la ligne 2 de l'Algorithme 1 et aux lignes 3 et 9 de l'Algorithme 2. Cela correspond aussi au moment d'affichage (`cout << ...`) dans le code source des fonctions `Graphe30::rechercheProfondeur` et `Graphe30::rechercheLargeur` fournies au verso.

2.1 Rappels des algorithmes

Algorithme 1 Recherche en profondeur

```
1.  RECHERCHEPROFONDEUR(G = (V,E), v ∈ V)
2.      v.visité ← vrai
3.      pour toute arête e ∈ v.aretesSortantes()
4.          w ← e.arrivee
5.          si ¬w.visité
6.              RechercheProfondeur(G, w)
```

Algorithme 2 Recherche en largeur

```
1.  RECHERCHELARGEUR(G = (V,E), v ∈ V)
2.      file ← CRÉERFILE
3.      s.visité ← vrai
4.      file.ENFILER(v)
5.      tant que ¬file.vide()
6.          s ← file.defiler()
7.          pour tout arête a = (s,s') ∈ E
8.              si ¬s'.visité
9.                  s'.visité ← vrai
10.                 file.ENFILER(s')
```

3 Graphes | Analyse de la complexité du code au verso

Supposez : n = nombre de sommets ; m = nombre d'arêtes ; et $n < m < n^2$. Notez qu'il y a au plus une arête sortante partant d'un sommet x vers un sommet y. Soyez conscients que la représentation des classes et des fonctions au verso ne sont pas forcément optimales. Ainsi, leur complexité peut être supérieure au résultat de l'analyse présentée en classe et dans les notes de cours.

```cpp
// Représentation de graphe pour groupe 30
class Graphe30 {
    map<string, map<string, int> >
    sommets;
    //map : un dictionnaire basé sur un
    arbre binaire de recherche rouge-noir
  public:
    //...
};
```

```cpp
void Graphe30::ajouterArete(const string& a, const string& b, int poids){
    sommets[a][b] = poids; // ajoute une arête du sommet a à b avec un coût de poids
}
void Graphe30::parcoursRechercheProfondeur(const string& s) const{
    set<string> visites;
    parcoursRechercheProfondeur2(s, visites);
}
void Graphe30::parcoursRechercheProfondeur2(const string& s, set<string>& visites) const{
    if(visites.find(s)==visites.end())
        return;
    visites.insert(s);
    cout << s << '\t';
    map<string, int>& voisins = sommets[s]; // sommets[s] ou sommets.at(s)
    for(map<string,int>::const_iterator i=voisins.begin();i!=voisins.end();++i)
        parcoursRechercheProfondeur2(i->first, visites);
}
void Graphe30::parcoursRechercheLargueur(const string& s) const{
    vector<string> visites;
    queue<string> file;
    visites.push_back(s);
    cout << s << '\t';
    file.push_back(s);
    while(!file.empty()){
        map<string, int>& voisins = sommets[file.front()]; // sommets[..] ou sommets.at(..)
        file.pop();
        for(map<string,int>::const_iterator i=voisins.begin();i!=voisins.end();++i){
            if(visites.find(i->first)!=visites.end()){  // on suppose que vector::find existe
                cout << i->first << '\t';
                file.push(i->first);
                visites.push_back(i->first);
            }
        }
    }
}
```