

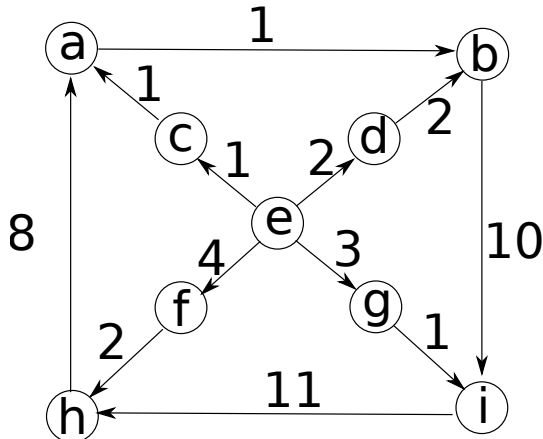
## 1 Monceaux (*heap*)

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 A

Le graphe A 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 énumérées dans l'ordre  $\langle c, d, f, g \rangle$ .
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 `Graphe20::rechercheProfondeur` et `Graphe20::rechercheLargeur` fournies au verso.

### 2.1 Rappels des algorithmes

---

#### Algorithme 1 Recherche en profondeur

---

```

1. RECHERCHEPROFONDEUR( $G = (V, E)$ ,  $v \in V$ )
2.    $v.\text{visité} \leftarrow \text{vrai}$ 
3.   pour toute arête  $e \in v.\text{aretesSortantes}()$ 
4.      $w \leftarrow e.\text{arrivee}$ 
5.     si  $\neg w.\text{visité}$ 
6.       RechercheProfondeur( $G, w$ )

```

---



---

#### Algorithme 2 Recherche en largeur

---

```

1. RECHERCHELARGEUR( $G = (V, E)$ ,  $v \in V$ )
2.    $file \leftarrow \text{CRÉERFILE}$ 
3.    $s.\text{visité} \leftarrow \text{vrai}$ 
4.    $file.\text{ENFILER}(v)$ 
5.   tant que  $\neg file.\text{vide}()$ 
6.      $s \leftarrow file.\text{défiler}()$ 
7.     pour tout arête  $a = (s, s') \in E$ 
8.       si  $\neg s'.\text{visité}$ 
9.          $s'.\text{visité} \leftarrow \text{vrai}$ 
10.         $file.\text{ENFILER}(s')$ 

```

---

## 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.

```

1 // Représentation de graphe pour groupe 20
2 class Graphe20 {
3     map<string, set<string> > sommets;
4     //map : un dictionnaire basé sur un
5     //set : un arbre binaire de recherche
6     //rouge-noir
7     public:
8     //...
9 };

```

```

1 void Graphe20::ajouterArete(const string& a, const string& b){
2     sommets[a].insert(b); // ajoute une arête du sommet a à b
3 }
4 void Graphe20::parcoursRechercheProfondeur(const string& s) const{
5     vector<string> visites;
6     //vector: un tableau dynamique linéaire, très similaire à notre classe Tableau<T>
7     parcoursRechercheProfondeur2(s, visites);
8 }
9 void Graphe20::parcoursRechercheProfondeur2(const string& s, vector<string>& visites) const{
10     if(visites.find(s)==visites.end()) // on suppose l'existence de la fonction vector::find
11         return;
12     visites.push_back(s); // push_back ajoute à la fin
13     cout << s << '\t';
14     set<string>& voisins = sommets[s]; // sommets[s] ou sommets.at(s)
15     for(set<string>::const_iterator i=voisins.begin();i!=voisins.end();++i)
16         parcoursRechercheProfondeur2(*i, visites);
17 }
18 void Graphe20::parcoursRechercheLargueur(const string& s) const{
19     set<string> visites;
20     queue<string> file;
21     visites.insert(s);
22     cout << s << '\t';
23     file.push(s);
24     while(!file.empty()){
25         set<string>& voisins = sommets[file.front()]; // sommets[..] ou sommets.at(..)
26         file.pop();
27         for(set<string>::const_iterator i=voisins.begin();i!=voisins.end();++i){
28             if(visites.find(*i)!=visites.end()){
29                 visites.insert(*i);
30                 cout << *i << '\t';
31                 file.push(*i);
32             }
33         }
34     }
35 }

```