Université Clermont Auvergne - IUT Clermont Ferrand
Département Informatique
première année

# SAÉ S2.02 : Exploration algorithmique d'un problème

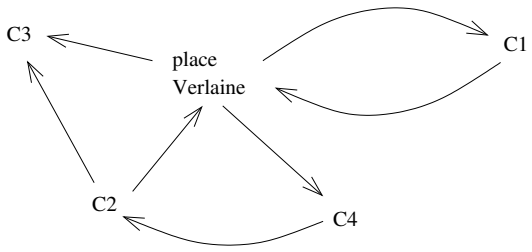
**Compétence 2 :** Optimiser des applications

**Apprentissages critiques :**
- AC1 : Analyser un problème avec méthode (découpage en éléments algorithmiques simples, structures de données)
- AC2 : Comparer des algorithmes pour des problèmes classiques (tris simples, recherche...)
- AC 3 : Formaliser et mettre en œuvre des outils mathématiques pour l'informatique

**Projet en 2 parties indépendantes, à traiter toutes les deux.**

# 1 Partie 1 : Un réseau routier urbain

On considère un réseau routier urbain modélisant les rues (selon leur sens de circulation) d'une ville. Ce réseau représente différents points caractéristiques de la ville (tels que des carrefours ou extrémité d'impasse), et les rues en tenant compte de leur sens de circulation entre ces points caractéristiques. Il se peut qu'il y ait une possibilité d'aller d'un carrefour A à un carrefour B directement par une rue, mais pas forcément de B à A si cette rue est en sens unique de A vers B. Voir la figure ci-dessous pour un exemple.




## 1.1 But

On souhaite modéliser un réseau routier urbain, pouvoir faire des opérations simples sur ce réseau (par exemple, ajout d'un point caractéristique) et pouvoir résoudre un problème d'accessibilité à savoir pouvoir dire si une personne partant en voiture d'un point caractéristique A donné peut se rendre au point caractéristique B donné que ce soit en prenant une rue seulement ou une succession de rues (en respectant le sens de circulation, bien sûr).

## 1.2 Travail à faire :

1. Sachant que les opérations les plus courantes réalisées sur ce réseau routier sont :
   - Ajout/suppression d'un point caractéristique ;
   - Ajout/suppression d'une liaison (rue) ;
   - Parcours des points caractéristiques accessibles directement depuis un point A en suivant une rue ;

— Parcours des points caractéristiques accessibles depuis un point A (en suivant une ou une succession de rues) ;

indiquer la ou les structures de données nécessaires pour stocker/représenter un tel réseau (code ou pseudo code). Vous détaillerez comment représenter un point caractéristique, une rue, un réseau : est-ce par un type objet ou une structure composée de tels ou tels champs/attributs, un simple entier ou une chaîne de caractères, un tableau de ..., une liste de ..., ou une autre structure plus complexe. Illustrer les explications avec des schémas.

2. Justifier votre choix (pourquoi telles structures plutôt que telles autres) en termes de place mémoire, temps d'exécution ou autre.
3. Indiquer en pseudo code l'algorithme permettant à partir d'une paire de points A et B donnée de savoir si B est accessible directement depuis A en suivant une rue.
4. Indiquer en pseudo code l'algorithme permettant à partir d'une paire de points A et B donnée de savoir si on peut se rendre depuis A vers B que ce soit en utilisant une rue seulement ou une succession de rues (selon le sens de circulation).
5. Donner pour chacun de ces algorithmes leur complexité (leur nombre d'opérations en fonction de leur nombre de points caractéristiques et/ou de liaisons). Il ne suffit pas de réciter une complexité, mais d'expliquer à partir de l'algorithme le coût de chaque instruction pour en déduire un coût total.

### 1.3 Consignes pour cette partie :

Si vous choisissez de donner vos algorithmes en pseudo-code, soyez précis. Des exemples de pseudo-code sont donnés en Annexe.

Si vous choisissez de donner vos algorithmes sous forme de code, ne donnez pas l'intégralité de votre programme, mais seulement les quelques lignes décrivant votre algorithme. Les seuls langages qui seront acceptés sont C et C++.

## 2 Partie 2 : Comparaison d'algorithmes

Voici trois algorithmes écrits en C.

```
void algo1(int tab[], int nb){
    int echange = 1;
    int i;
    int deb = 0, fin = nb-1;

    while(echange>0){
        echange=0;
        for(i=deb; i < fin; i++){
            if(tab[i]> tab[i+1]){
                echanger(tab, i,i+1);
                echange++;
            }
        }
        fin--;
        for(i=fin; i > deb; i--){
            if(tab[i]< tab[i-1]){
                echanger(tab, i,i-1);
                echange++;
            }
        }
        deb++;
    }
}
```

avec :

```
void echanger(int tab[], int i, int j){
    int tmp;
    tmp = tab[i];
    tab[i] = tab[j];
    tab[j] = tmp;
}

void algo2(int tab[], int nb){
    int* cpt;
    int min = tab[0], max = tab[0];
    int i,j;

    for(i=1; i<nb; i++){
        if(min > tab[i]) min = tab[i];
        if(max < tab[i]) max = tab[i];
    }

    cpt = (int*)calloc((max-min+1),sizeof(int));
    if(cpt==NULL){
        printf("pb aloc mémoire\n");
        exit(1);
    }
    for(i = 0; i<nb; i++){
        cpt[tab[i]-min]++;
    }

    j=0;
    for(i=0; i< max-min+1 ; i++){
        while(cpt[i]!=0){
            tab[j] = i+min;
            j++;
            cpt[i]--;
        }
    }
    free(cpt);
}

void algo3(int tab[], int n){
  int i, pos, j, tmp;
  for(i = 1; i< n; i++){
    pos = recherchePos(tab,i, tab[i]);
    if(pos !=i){
      tmp = tab[i];
      for(j = i; j> pos; j--){
        tab[j] = tab[j-1];
      }
      tab[pos] = tmp;
    }
  }
}
```

avec :

```
int recherchePos(int tab[], int n, int val){
    int i;

    for(i = 0; i<n; i++){
        if(val < tab[i]){
            return i;
        }
    }
    return i;
}
```

### 2.1 Travail à faire :

1. Dérouler les algos sur différents tableaux d'entiers et expliquer les différentes étapes en détaillant par des schémas. Les tableaux utilisés seront ici de taille raisonnable (6 à 8 éléments suffisent) et différents dans le sens : déjà triés à l'endroit, triés à l'envers ou mélangés. [3 pts]
2. Pour chaque algorithme, calculer sa complexité théorique (son nombre d'opérations en fonction du nombre d'éléments du tableau pris en paramètre) dans le pire cas en détaillant toutes les étapes de votre analyse. Quel algorithme a la meilleure complexité théorique ? [3 pts]
3. Après avoir testé ces algos sur des tableaux de tailles variées et pouvant être très grandes (remplis de valeurs aléatoires par exemple), établir des courbes expérimentales d'efficacité pour les différents algorithmes, en fonction de la taille des tableaux pris en paramètre. Vous pourrez considérer différentes grandeurs d'intérêt (temps de calcul, utilisation de la mémoire, etc.) et différents outils mathématiques d'analyse (fonctions, statistiques descriptives, etc.). Expliquer brièvement votre protocole, et commenter les résultats. (Remarque : pour mesurer un temps d'exécution d'une fonction on peut utiliser les fonctions clock() ou time() de time.h) [3 pts]
4. En conclusion, pour chacun des trois algorithmes : résumer son fonctionnement en une ou deux phrases, puis lister ses éventuels avantages et/ou inconvénients. [1 pt]

# 3 Organisation :

Vous travaillerez par groupes d'au plus trois étudiants. Les groupes ne sont pas libres mais formés par l'équipe enseignante.

# 4 Livrables :

À l'issue de cette SAÉ, le livrable que vous devez fournir est un rapport au format PDF contenant les réponses de chaque partie de ce sujet.

# 5 Barème indicatif :

— Partie 1 (10 pts) : 2 pts pour chacune des 5 questions
— Partie 2 (10 pts) : détaillé dans le sujet.

# 6 Modalités de rendu :

Vous rendrez un document pdf intitulé NOM1-NOM2-NOM3.pdf sur le moodle, avant le vendredi 24 mars 2023, 23h59.

# 7 Annexe : Exemples de pseudo-codes

Exemple 1 :

```
Fibonacci(n):
    v1 = 0, v2 = 1
    si n = 0:
        retourner v1
    si n = 1:
        retourner v2
```

```
        pour i de 2 à  n:
            tmp = v2
            v2 = v1+v2
            v1 = tmp

        retourner v2
```

Exemple 2 :

```
    Max(tab ){
        res:= tab[0]

        for i = 1 … taille(tab) {
            if tab[i] > res {
                res:= tab[i]
            }
        }
        return res
    }
```

Exemple 3 :

```
    Mineurs(Liste<Personne> l ):
        Liste<Personne> res = {}

        pour tout p dans  l:
            si p.age <  18:
                ajouter p dans res

        retourner res
```

Exemple 4 :

```
    testPourUnDictionnaire() :
        Map<string, string> dictionnaire={}

        m["toto"] = "personnage sympatique"
        m["azalée]= "petite espèce de rhododendron"

        Paire<string,string> p = {"zèbre", "mammifère équidé à robe rayée" }
        m.inserer(p)

        m["toto"] = "personnage enfantin finalement"

        afficher("il y a " + m.taille() + "éléments dans le dico, qui sont : ")
        pour chaque Paire<string,string> p de m faire
            afficher(p.premier + " a pour définition : " + p.deuxieme)
        fin pour
```