# 2I006

# Algorithmique appliquée et structures de données

2014-2015

**Fascicule de TD**
**Séance 7 à 11**


| | |
|---|---|
| Gr2 Mercredi 16h00-19h45 : | Nawal Benabbou et Hilaire Chevreau |
| Gr3 Jeudi 10h45-12h30 / 14h00-15h45 : | Grégoire Cotté et Thibaut Lust |
| Gr2 Vendredi 14h00-17h45 : | Bruno Escoffier |

# TD7 : Arbres binaires équilibrés

## Exercice 1 – Arbres binaires de recherche équilibrés

Un arbre binaire de recherche est dit parfaitement équilibré si pour tout noeud de l'arbre, le nombre de noeuds dans le sous-arbre gauche et le nombre de noeuds dans le sous-arbre droit diffèrent au plus de 1.
Un arbre binaire de recherche est dit équilibré si pour tout noeud de l'arbre, la hauteur du sous-arbre gauche et la hauteur du sous-arbre droit diffèrent au plus de 1.
On considère, dans la suite de l'énoncé, des arbres binaires de recherche équilibrés contenant des entiers.

**Q 1.1** Décrire l'arbre binaire de recherche équilibré obtenu si on ajoute les nombres entiers de 1 à 10 dans un arbre binaire équilibré vide.

**Q 1.2** Écrire une fonction `AB_hauteur` qui étant donné un arbre binaire, retourne sa hauteur.

**Q 1.3** Écrire une fonction `AB_rotationDroite` qui permet de réaliser une rotation droite d'un arbre binaire. La figure 1 présente une telle rotation.

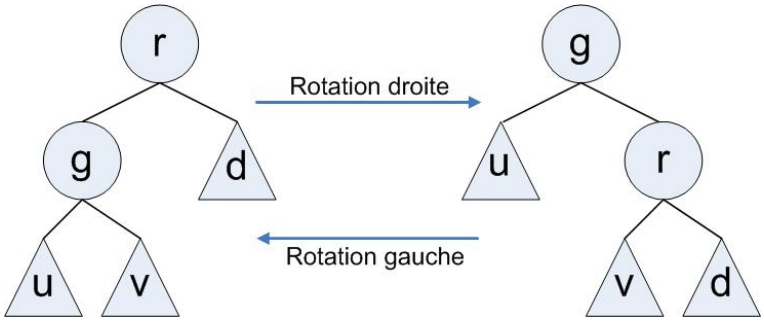



FIGURE 1 – Rotation droite et rotation gauche dans un arbre binaire

**Q 1.4** Écrire une fonction `AB_rotationGauche` qui permet de réaliser une rotation gauche d'un arbre binaire.

**Q 1.5** Écrire une fonction `ABRE_insererElt` qui étant donnés un ABR équilibré $ab$ et une étiquette $e$, retourne un arbre équilibré obtenu en ajoutant $e$ à $ab$. Pour ce faire, on insére tout d'abord l'étiquette $e$ en utilisant la fonction d'insertion dans un ABR définie dans l'exercice 2. Soit $A$, l'arborescence ainsi obtenue. Soit $G$ le sous-arbre gauche de $A$ et $D$ le sous-arbre droit de $A$. On réorganise alors $A$ de la manière suivante :

- Si $|h(G) - h(D)| < 2$, aucune réorganisation est nécessaire.
- Si $h(G) - h(D) = 2$, alors soient $g$ et $d$ les sous-arborescences gauche et droite de $G$. Si $h(g) < h(d)$ alors on effectue une rotation gauche de $G$. Dans tous les cas, on effectue une rotation droite de $A$.

— Si $h(G) - h(D) = -2$ alors soient $g$ et $d$ les sous-arborescences gauche et droite de $D$. Si $h(d) < h(g)$ alors on effectue une rotation droite de $D$. Dans tous les cas, on effectue une rotation gauche de $A$.

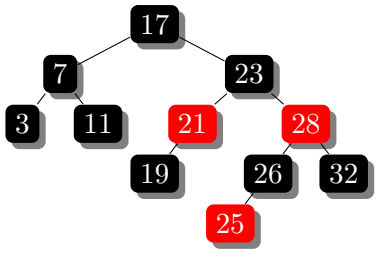
## Exercice 2 – Arbre Rouge-Noir

Les Arbres Binaires de Recherche (ABR) dans leur version équilibrée (AVL) ont des propriétés très intéressantes mais gardent certains désavantages : par exemple, la suppression d'un élément peut demander un nombre important de rotations. D'autres structures de données existent donc, qui essayent de pallier à ces inconvénients. Par exemple, les arbres rouge-noir. Un arbre rouge-noir (ARN) est un arbre dont les nœuds sont colorés en noir ou rouge et vérifie :

1. l'arbre est un arbre binaire de recherche ;
2. la racine d'un ARN est noire ;
3. les enfants d'un nœud rouge (s'ils existent) sont noirs ;
4. le nombre de nœuds noirs sur tous les chemins de la racine à une feuille est constant.

Dans cet exercice, on appellera simplement "chemin" dans un arbre, un chemin de la racine à une feuille de l'arbre.

La figure ci-dessous représente un ARN (les sommets rouges sont les sommets 21, 25 et 28).




**Q 2.1** Montrer que dans un ARN, on ne peut avoir deux nœuds rouges successifs le long d'un chemin. En quoi cette propriété aide-t-elle à interdire des ARN trop déséquilibrés ? Plus précisément, il s'agit de prouver que, dans un ARN, la longueur du plus grand chemin ne peut pas être plus de 2 fois plus grands que celui du plus petit chemin.

**Q 2.2** Modifier la structure du type `btree`, vue en cours, afin d'obtenir un type `ARNtree` représentant un nœud d'ARN.

**Q 2.3** Écrire une fonction C (`int nbr_rouges(ARNtree* t)`) de calcul du nombre de nœuds rouges dans un ARN.

**Q 2.4** Écrire une ou des fonctions C permettant de vérifier qu'un `ARNtree` vérifie bien les propriétés d'un arbre rouge-noir. Attention à la condition (4) ! Ne pas oublier la condition (1) !

**Q 2.5** Proposer un algorithme d'insertion dans un ARN. Insérer 18 puis 24 dans l'ARN de la figure. Quelle est la complexité, voyez-vous une modification nécessaire dans la structure de donnée de ARNtree ?

# TD8 : Graphes

## Exercice 1 – Structure de graphes

On considère une carte routière où l'on connaît les coordonnées GPS $x$ et $y$ de chaque ville. La carte est très grande et contient un grand nombre de villes. On veut pouvoir retrouver rapidement différentes informations sur cette carte déterminer si deux villes sont reliées par une route et la distance entre elles.

**Q 1.1** Proposer une SDA permettant de stocker les villes et l'existence d'une route entre deux villes. Quelle implémentation proposez-vous ?

**Q 1.2** Proposer une structure C pour cette SDA et une fonction de création de la structure en l'initialisant à $n$ villes.

**Q 1.3** Créer des fonction de mise à jour des coordonnées. Utilisez-là dans un main pour entrer quelques villes.

**Q 1.4** Proposer une fonction d'ajout d'une route entre deux villes . Utilisez-là dans votre main.

**Q 1.5** Comment peux-t-on dessiner cette structure (voir brochure sur le format gnuplot ci-joint).

## Exercice 2 – Coloration

On considère le problème suivant intervenant dans la gestion des stations de radios. Un organisme de gestion des ondes radios désire affecter des fréquences aux différentes antennes qui couvrent une vaste zone géographique. En effet, lorsque deux antennes couvrent une même zone et émettent sur la même fréquence, cela crée des interférences qui empêchent une bonne réception. On considère un ensemble $S$ d'antennes radios qui émettent chacune une unique station de radio. On connait également l'ensemble $E$ des paires d'antennes qui s'interfèrent.

**Q 2.1** Proposez une modélisation de ce problème en utilisant un graphe non orienté où les arêtes indiquent un conflit.

**Q 2.2** L'objectif est d'attribuer des fréquences. Indiquer comment utiliser des "couleurs" permet de traiter le problème.

**Q 2.3** Comment trivialement peut-on alors résoudre le problème en utilisant $|S|$ couleurs ?

**Q 2.4** On désire positionner un nombre réduit de fréquences. Proposer un algorithme "glouton" pour attribuer des fréquences (on appelle algorithme glouton un algorithme qui prend des décisions les unes après les autres sans remettre en cause les décisions prises auparavant).

## Exercice 3 – Degrés des sommets

On consdière un même graphe orienté contenant $n$ sommets et $m$ arcs. On désire comparer les deux implémentations classique de ce graphe : l'une sous forme matricielle, l'autre utilisant des listes d'ad-

jacence.

**Q 3.1** Donner les schémas correspondant à ces deux implémentations.

**Q 3.2** On appelle *degré sortant* (resp. *degré entrant*) le nombre d'arcs sortant (res. entrant) d'un sommet. Donner les algorithmes permettant de calculer les degrés entrants et sortant d'un sommet donné.

**Q 3.3** Indiquer la complexité de ces deux fonctions.

# Visualisation de graphes en format postscript

**Description générale**

Le langage ou format postcript permettant de décrire une page composé de dessins et de textes. Il repose sur des formules vectorielles pour la plupart de ses éléments, mais peut aussi comporter des description matricielle de certains composants (images etc). Ce format est un standard, la plupart des imprimantes lasers haut de gamme peuvent traiter directement le format PostScript. Le développement du PostScript est arrêté par Adobe depuis 2007, afin que le PDF puisse prendre la relève. Néanmoins il s'agit d'un format très léger et simple qui permet de dessiner rapidement de très grands dessins composés d'une grande quantité d'éléments simples. C'est pourquoi il est parfaitement adapté pour représenter des graphes.

On peut visualiser un dessin codé par un langage postscript en utilisant le logiciel ghostview (gv). On peut aussi noter que la plupart des outils apte à lire les formats pdf peuvent la plupart du temps interpréter le format postscritp, mais il le font plus lentement que ghostview qui est dédié à cet effet. Il est également possible de transformer un dessin en langage postscript en un fichier pdf par exemple (ou autre format). Sous linux, le logiciel ps2pdf permet de réaliser cette transformation. Le dessin résultat peut alors être visualisé, imprimé ou transformé dans un format facile à diffuser (pdf, jpg),...

**Langage poscript**

Le langage postscript est constitué d'un fichier texte qui peut être interprété directement par une imprimante.

Afin de ne pas dépendre des formats de page (A4, american letter,...), une page postscript est défini par une unité de mesure appelée points. Le format A4 correspond à 595 points sur et 842 points. Afin de dessiner un graphe dont les coordonnées des points ont un écart entre $dimX$ et $dimy$, il suffit alors de faire une homothétie pour se ramener à ces coordonnées A4.

**Visualiser des graphes**

Dans ce document, nous nous intéressons uniquement à la représentation d'un graphe en langage postscript dans le cas où le graphe est décrit:
- par un ensemble de sommets repérés par des coordonnées planaires
- par une liste d'arêtes données par des couples de points.

Il s'agit donc ici d'un graphe non-orienté (vous pourrez néanmoins déduire facilement de ce document des façons de représenter les arcs d'un graphe orienté).

Noter que dans le cas où les coordonnées des sommets ne sont pas connus, il est préférable d'utiliser des outils de représentations de graphe capable d'attribuer automatiquement des coordonnées bien choisies aux sommets de manière à représenter le graphe le plus agréablement possible. Le logiciel Graphviz est par exemple très adapté pour cela (en revanche, il ne permet pas de représenter des graphes de dimensions importantes).

**Commandes utiles pour visualiser des graphes**

On va donc ici représenter les sommets d'un graphe par des cercles et les arêtes par des lignes droites:
- Pour dessiner un cercle de 2,5 points de rayon et de centre de coordonnées $x, y$, il suffit d'utiliser les trois lignes suivantes

```
x y 2.5 0 360 arc
fill
stroke
```

- Pour dessiner une droite entre les points $x1, y1$ et $x2, y2$, il suffit d'utiliser les trois lignes suivantes

```
x1 y1 moveto
x2 y2 lineto
stroke
```

- Par défaut, le trait de dessin est en noir, si l'on désire dessiner d'une autre couleur, il suffit de faire précéder la commande précédente par

```
r g b setrgbcolor
```

où *rgb* est le codage en pourcentage de la quantité de rouge, vert et bleu de la couleur demandée: par exemple `1 0 0` correspond au rouge et `0 1 0` correspond au vert.

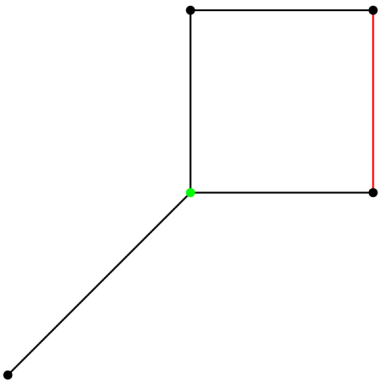
**Exemple de fichier**

Il est à noter que pour un tel dessin, il n'est même pas nécessaire de mettre une entête à un fichier texte correspondant à un fichier postscript.

Afin d'automatiser le dessin d'un graphe à partir d'un programme possédant une structure de données contenant ce graphe, il suffit donc de créer un fichier texte contenant les commandes précédentes les unes à la suite des autres.

Voici un exemple de fichier postscript et le graphe correspondant:

```
100 200 moveto
200 300 lineto
stroke
200 300 moveto
200 400 lineto
stroke
200 400 moveto
300 400 lineto
stroke
1 0 0 setrgbcolor
300 400 moveto
300 300 lineto
stroke
0 0 0 setrgbcolor
300 300 moveto
200 300 lineto
stroke
100 200 2.5 0 360 arc
fill
0 1 0 setrgbcolor
200 300 2.5 0 360 arc
fill
0 0 0 setrgbcolor
200 400 2.5 0 360 arc
fill
300 400 2.5 0 360 arc
fill
300 300 2.5 0 360 arc
fill
```



# TD9 : Parcours et plus cours chemins

## Exercice 1 – Parcours et détection de circuits

On considère ici un graphe non-orienté et son implémentation par listes d'adjacence vue dans le TD précédent. On suppose dans tout l'exercice que le graphe est connexe, c'est-à-dire que, pour tout couple de sommets, il existe une chaîne les reliant.

**Partie 1 :** *Parcours récursif en profondeur*

**Q 1.1** On suppose tout d'abord que ce graphe est sans circuit. Proposer un algorithme récursif explorant tous les sommets du graphe à partir d'un sommet donné.

**Q 1.2** Donner des exemples de graphes sans circuit (dits graphes acycliques) et mettez en œuvre votre code sur ces exemples. Quels sont ceux qui correspondent à des arbres ? Comment se nomme alors le parcours en profondeur dans les arbres.

**Q 1.3** On suppose que le graphe contien à présent un circuit. Que se passe-t-il pour votre code sur ce graphe ? Proposer une correction du problème.

**Partie 2 :** *Détection de cycle*

**Q 1.4** Comment transformer le code de l'exercice précédent pour détecter s'il existe un cycle dans le graphe ? Donnez une fonction permettant de tester s'il existe un circuit dans un graphe.

**Q 1.5** On désire à présent récupérer en retour un cycle s'il en existe un. Pour cela, nous allons conserver l'arborescence (non connexe parfois) correspondant au parcours en profondeur effectué. Comment stocker efficacement cette arborescence ?

**Q 1.6** Comment retrouver le cycle dans la structure de la question précédente ?

**Q 1.7** Donnez une fonction qui retourne un circuit dans un graphe, s'il en existe un, sous forme d'une liste chaînée.

**Partie 3 :** *Parcours itératif en profondeur et en largeur*

On suppose ici que l'on dispose d'une bibliothèque permettant de gérer une file et une pile d'entier.

**Q 1.8** Proposer une implémentation itérative du parcours en profondeur.

**Q 1.9** Modifier ce code pour effectuer un parcours en largeur.

**Q 1.10** On désire calculer la taille minimale en nombre d'arcs reliant un sommet à tous les sommets du graphe. Utiliser un parcours en largeur pour ce calcul.

## Exercice 2 – Autour de la preuve de l'algorithme de Dijkstra

Soit un graphe non orienté $G = (V, E)$, avec $V$ l'ensemble des sommets, et $E$ l'ensemble des arêtes, et une fonction de pondération $w$ sur les arêtes $e \in E : w : E \to R^+$.

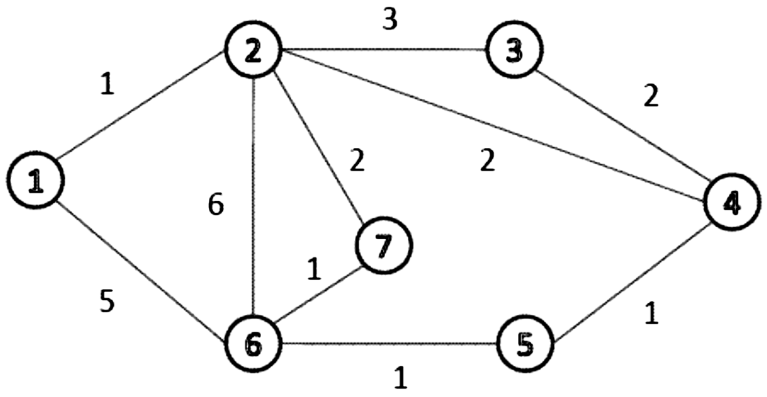



FIGURE 1 –

L'algorithme de Dijkstra peut être écrit en pseudo-langage sous la forme condensée suivante :

**entrée** : graphe $G = (V, E)$, sommet d'origine $r$
**sortie** : le tableau $pred()$ indiquant le sommet prédécesseur

*Initialisation*
  $\lambda(v) = \infty \ \forall v \in V$
  $pred(v) = -1 \ \forall v \in V$
  $marque(v) = False \ \forall v \in V$
  $Bordure$ comme un tas vide
Ajouter $r$ à distance 0 dans $Bordure$
marque[v]=True
tant que $Bordure \neq \emptyset$
  $s =$ supprimerMin($Bordure$)
  $Marquer[s] = True$  MettreAJourBordure($Bordure, s$)
fin tant que

Soit $\mu(r, v)$ le chemin défini entre le sommet origine $r$ et un sommet $v$ quelconque du graphe, obtenu après application de l'algorithme de Dijkstra, grâce à l'information contenue dans le tableau $pred()$. Le coût associé à ce chemin est donné par $\lambda(v)$. Le tableau $marque()$ indique si le sommet $v$ a déjà été traité complètement ou non par l'algorithme, autrement dit, s'il a été rentré puis sorti de la bordure B.

**Q 2.1** Indiquer dans le cadre du calcul des plus courts chemins ce que représente $\lambda(v)$ en fonction des poids $w(e)$.

**Q 2.2** Redonner la définition de la bordure d'un ensemble $S \subset V$ de sommets du graphe G.

**Q 2.3** La fonction **mettreAJourBordure** met à jour la bordure B en prenant en compte tous les voisins v de s, donc liés à s par un arête $e = (s, v) \in E$. En considérant les deux cas $v \in B$ et $v \neq B$,

donner le principe des mises à jour de B et des valeurs $\lambda(v)$, $pred(v)$ (si nécessaire).

**Q 2.4** La fonction **supprimerElementBordure** retourne parmi tous les sommets de la bordure un certain sommet s, en appliquant un critère de choix particulier sur $\lambda(v)$ pour tout sommet $v \in B$. Préciser ce critère de choix. En démontrer la validité.

**Q 2.5** Appliquer l'algorithme de Dijkstra au graphe de la figure 2 pour la détermination des plus courts chemins issus du sommet 1 vers tous les autres sommets de $V$.

**Q 2.6** Donner le code permettant de retrouver le chemin de $r$ à un sommet $i$ en utilisant le tableau $pred$.

## Exercice 3 – Implémentation de l'algorithme de Dijkstra

En utilisant la structure de Graphe par liste d'adjacence précédemment et en utilisant la structure de tas vue à la dernière séance de TME, proposez une implémentation de l'algorithme de Dijkstra.

# TD10 : Flot de valeur maximale

## Exercice 1 – Réseau commercial

Une société de vente par correspondance gère un réseau commercial et logistique composé de :
- trois centres de prise de commandes $C_i$ $(i = 1, 2, 3)$ ;
- deux centres de préparation de commandes $P_j$ $(j = 1, 2)$ ;
- deux centres de distribution $D_k$ $(k = 1, 2)$.

Cette société désire évaluer la capacité mensuelle du réseau (nombre maximum de commandes pouvant être prises, préparées et livrées en un mois). Le réseau possède les caractéristiques suivantes (en milliers de commandes par mois) :
- les capacités de prise de commandes de $C_1$, $C_2$ et $C_3$ sont respectivement de 30, 30 et 10 ;
- les capacités de préparation de commandes de $P_1$ et $P_2$ sont respectivement de 10 et 60 ;
- les capacités de distribution de $D_1$ et $D_2$ sont respectivement de 30 et 50 ;
- chaque centre de prise de commandes peut alimenter les 2 centres de préparation, mais les capacités des liaisons informatiques limitent à un maximum de 20000 commandes par mois le flux entre un centre $C_i$ et un centre $P_j$ ;
- $P_1$ alimente uniquement $D_1$ ; $P_2$ alimente uniquement $D_2$ ;
- $D_2$ a la possibilité de transférer une partie de son activité sur $D_1$ ; ce transfert ne peut dépasser 20000 commandes par mois, il ne réduit pas la capacité de distribution de $D_2$ ;

**Q 1.1** Construire un réseau de transport modélisant le contexte.

**Q 1.2** Définir le nombre maximum de commandes que la société peut traiter mensuellement. Traduire le flot obtenu en termes d'activité pour chaque centre.

**Q 1.3** Identifier une coupe de capacité minimale.

## Exercice 2 – Allocation de tâches

On s'intéresse à la modélisation d'un problème d'allocation de tâches correspondant aux procédures d'un programme à deux processeurs en minimisant le coût total de communication entre processeurs et le coût d'exécution de ces procédures.
Le coût d'exécution de la procédure $i$ sur le processeur 1 (respectivement 2) est défini par $\alpha_i$ (respectivement $\beta_i$) . Les coûts de communication sont donnés par $c_{ij}$ si $i$ et $j$ sont affectées à des processeurs différents. Nous supposons de plus que 4 procédures doivent être affectées. Les coûts sont définis dans les tableaux ci-dessous :

| $i$ | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| $\alpha_i$ | 6 | 5 | 10 | 4 |
| $\beta_i$ | 4 | 10 | 3 | 8 |

**Q 2.1** Formuler ce problème sous-forme de problème de coupe minimum dans un graphe non-orienté que vous définirez.

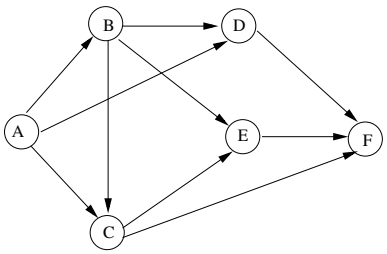
| $c_{ij}$ | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | 0 | 5 | 0 | 0 |
| 2 | 5 | 0 | 6 | 2 |
| 3 | 0 | 6 | 0 | 1 |
| 4 | 0 | 2 | 1 | 0 |

**Q 2.2** Transformer le problème de coupe-minimum en un problème de flot maximal dans un graphe orienté.

**Q 2.3** Trouver l'affectation optimale.

## Exercice 3 – Plan d'évacuation et flot avec contraintes sur les sommets

**Q 3.1** On condidère le graphe $G$ ci-dessous. On dit que deux chemins reliant les sommets $A$ à $F$ sont sommets-disjoints s'ils n'ont aucun sommet commun en dehors de leurs extrémités. Donner le nombre maximal de chemins sommets-disjoints entre $A$ et $F$ que l'on peut dessiner en même temps sur le graphe $G$. Noter qu'un sommet différent de A et F appartient à au plus un des chemins retenus.




**Q 3.2** On considère un graphe non-orienté $G = (S, A)$ ayant $n$ sommets et deux sous-ensembles $X$ et $Y$ de $S$ (non nécessairement disjoints) ; les éléments de $X$ sont des salles d'n bâtiment et les éléments de $Y$ sont des issues de ce bâtiment. Un plan d'évacuation est un ensemble de chemins dans le graphe tel que deux d'entre eux n'ont pas de sommet commun et ou chacun d'eux mène d'une salle à une issue. Afin de créer un nombre important de chemins qui seraient chacun courtsn on désire déterminer un plan d'évacuation optimal c'est-à-dire contenant le maximum de chemins. Proposer un algorithme pour déterminer un plan d'évacuation optimale.

**Q 3.3** On considère le problème de flot avec contraintes sur les sommets définit comme un problème de flot maximal où l'on ajoute des contraintes supplémentaires définies par un ensemble de nombres $s_i$, $i = 1, \cdots, n$ tel que la somme des flots entrants en $x_i$ doit être inférieure à $s_i$. Proposer un algorithme pour déterminer une solution à ce problème.

# TD11: Généricité et manipulation des SDA

## Exercice 1 – Polymorphisme

Une variable de type `void *` permet de stocker une adresse mémoire non typée, c'est-à-dire dont on ne connaît pas le type stocké à cette adresse. Attention, une donnée stockée à une adresse non typée n'est donc pas lisible car on ne connaît pas le nombre d'octets utilisés, ni le codage binaire de ces octets.
La fonction `memcpy(a,b,s)` de la bibliothèque `string.h` permet de copier $s$ octets commençant à l'adresse $b$ dans l'emplacement de $s$ octets commençant à l'emplacement $a$.

**Q 1.1** Proposer une fonction `swap` qui permet d'échanger le contenu de deux variables de type quelconque. Donner une fonction `main` testant cette fonction.

## Exercice 2 – Manipulation de bibliothèque générique de SDA

Le langage C ne possède pas directement de bibliothèque de manipulation de SDA. Il existe par contre des bibliothèques génériques disponibles comme la bibliothèqe gstl. Elles sont pas d'accès facile car le langage C est difficilement générique...
On va plutôt s'intéresser dans cet exercice à la bibliothèque standard du C++ : la STL au travers d'un exemple simple. Le C++ est une évolution du C permettant la programmation générique et la programmation objet. Toutes les commandes du C sont réutilisées et les bibliothèques du C sont incluses dans le C++. Pour compiler les lignes suivantes, il suffit d'utiliser un compilateur C++ comme g++ à la place de votre compilateur gcc.

**Q 2.1** Redonnez la définition d'une implémentation générique d'une SDA.

**Q 2.2** La bibliothèque STL du C++ est générique. Le code suivant permet de définir une liste doublement chaînés d'entiers et d'ajouter en queue un entier à cette liste. Définissez de la même façon le pour définir une liste d'élément `struct` contenant un char et un flottant.

```
1  #include<list>
2
3  using namespace std;
4
5  int main(){
6
7    list<int> L;
8
9    L.push_back(3);
10
11 }
```

**Q 2.3** Les lignes suivantes permettent de parcourir et d'afficher les éléments de la liste $L$ d'entiers définie à la question précédente. Donner un code permettant de parcourir et d'afficher votre liste d'éléments struct

```cpp
#include<list>
#include<stdio.h>

using namespace std;

int main(){

  list<int> L;
  list<int>::const_iterator it;
  L.push_back(4);
  L.push_back(-2);
  L.push_back(10);
  for (it=L.begin();it!=L.end();it++)
    printf("%d ",(*it));
  printf("\n");
}
```

**Q 2.4** Le type générique list de la stl est en fait une classe d'objets. Cet objet contient une fonction (stockée comme un champ d'un struct) appelée méthode. Une de ses méthodes permet de trier l'objet list. On l'utilise ainsi :

```cpp
L.sort();
```

A votre avis, quelle est sa complexité ? Pouvez-vous l'utilisez pour votre liste d'éléments ?

## Exercice 3 – En route vers l'objet : Tableau hétérogène

Nous désirons manipuler le contenu d'une médiathèque qui contienne des livres et des disques. Pour cela, nous allons utiliser un seul tableau qui contienne des éléments des deux sortes.

**Q 3.1** Créer deux types `struct` correspondant à un type livre (titre, auteur, nombre de pages) et à un type CD (titre, interprete, auteur, compositeur, duree en minutes (flottants)).

**Q 3.2** Proposer un type permettant de stocker à la fois un élément livre et un élément CD. Créer un tableau dont chaque case contient un élément de ce type. Donner un programme permettant de remplir un tel tableau.

**Q 3.3** Une fois le tableau rempli, comment pouvoir retrouver si une case contient un élement livre ou un CD ? Appliquer cette idée à votre implémentation

**Q 3.4** Donner une fonction permettant d'afficher le contenu hétérogène du tableau.