Projet de synthèse

Algorithmes pour calculer l'enveloppe convexe en 2D

Giorgio Lucarelli

giorgio.lucarelli@univ-lorraine.fr

janvier 2022

UNIVERSITÉ DE LORRAINE | UFR MATHÉMATIQUES INFORMATIQUE MÉCANIQUE ET AUTOMATIQUE

Objectif du cours

- Utiliser les connaissances acquises pendant le L2 en
  - Maths Discrètes
  - Programmation C
  - Algorithmique et Structures des Données
  - Programmation Java
  - Interfaces graphiques

Organisation du cours

- travailler en **binômes**
- **les membres de chaque binôme appartiennent au même groupe TP**
- **déclaration des binômes :** par émail (giorgio.lucarelli@univ-lorraine.fr) jusqu'à dimanche 30 janvier
- sujet sur arche à partir de mardi

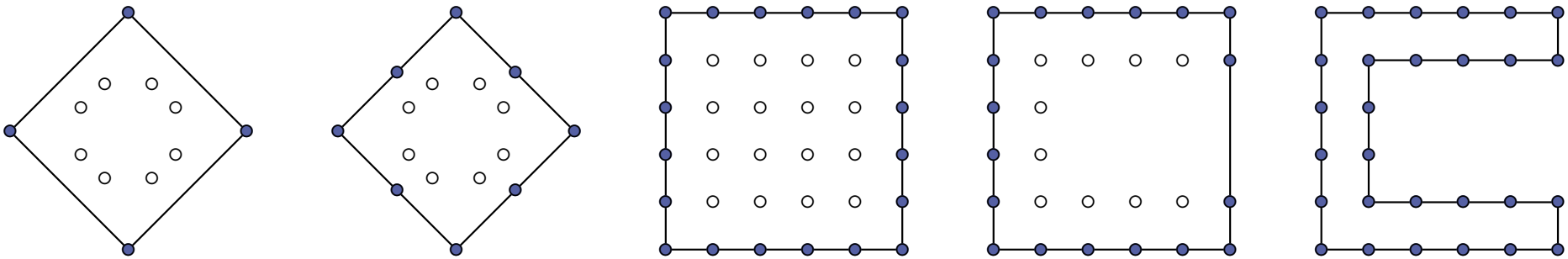
Emploi du temps

- 4h CM : aujourd'hui
  - présentation du projet et des outils
- 18h TP
  - suivi du projet
  - intervenants : Imad Assayakh (TP3, TP4), Giorgio Lucarelli (TP1, TP2, TP5)
- 8h TD : soutenances
  - chaque binôme présentera son projet pendant 30 minutes
  - vous devrez être présents que pendant votre présentation

Sujet : développer et simuler des différents algorithmes pour calculer l'enveloppe convexe d'un ensemble de points en 2D

Entrée : un ensemble $\mathcal{P} = \{P_1, P_2, \ldots, P_n\}$ de n points en 2D

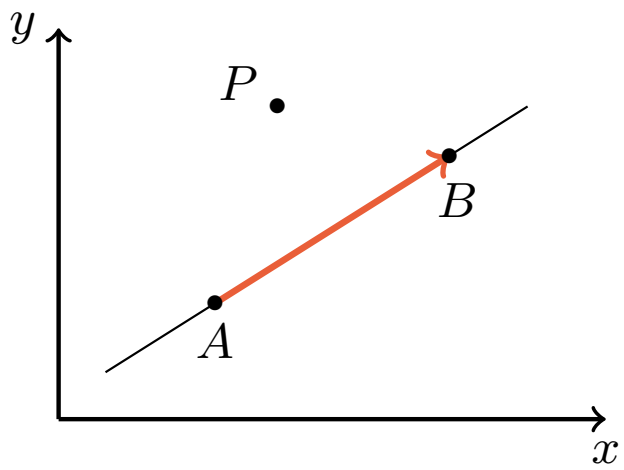
Sortie : une liste $\mathcal{L}$ des points de $\mathcal{P}$ dans le sens horaire,
tel que le polygone qui correspond aux points de $\mathcal{L}$ soit convexe
et inclue tous les points de $\mathcal{P}$.



non-convexe !

Un peu de géométrie

- Position d'un point par rapport à une droite
  - $\overrightarrow{AB}$: une droite orientée
  - P : un point



L'orientation de la droite est très importante dans les calculs !

- $D = (X_B - X_A) * (Y_P - Y_A) - (Y_B - Y_A) * (X_P - X_A)$
  - Si $D > 0$, alors P est *à gauche* de $\overrightarrow{AB}$
  - Si $D < 0$, alors P est *à droite* de $\overrightarrow{AB}$
  - Si $D = 0$, alors A, B et P sont *colinéaires*

Algorithme glouton, mais pas rapide

Idée ?

- Pour chaque couple de points ordonné (A, B), examiner la position des autres points par rapport à $\overrightarrow{AB}$.
- Si ils sont tous à droite, alors $\overrightarrow{AB}$ fait partie du polygone convexe.

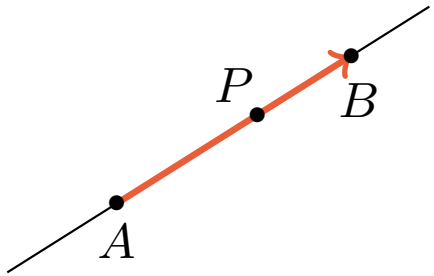
Pourquoi à droite ?

- L'ordre A, B respecte le sens horaire.

Cas special ?

- Si il existe un point P qui est
  1. colinéaire avec A et B, et
  2. dans le segment entre A et B

  alors $\overrightarrow{AB}$ ne fait pas partie du polygone convexe (dans ce cas, $\overrightarrow{AP}$ et $\overrightarrow{PB}$ appartiennent éventuellement au polygone convexe)



Algorithme glouton, mais pas rapide

Algorithme SlowConvexHull

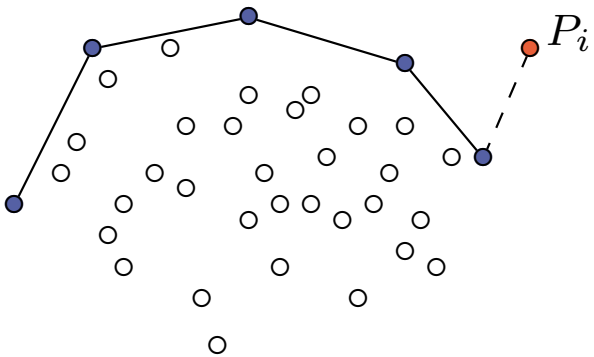
1: $E \leftarrow \emptyset$ // *l'ensemble d'arêtes orientées du polygone* convexe
2: **pour** chaque couple (A, B) de points ordonné **faire**
3: $ok \leftarrow true$
4: **pour** chaque point $P \neq A, B$ **faire**
5: **si** P est à gauche de $\overrightarrow{AB}$ ou fait partie du segment $\overrightarrow{AB}$ **alors**
6: $ok \leftarrow false$
7: **si** $valid$ **alors**
8: $E = E \cup \{\overrightarrow{AB}\}$
9: En utilisant E, créer la liste $\mathcal{H}$ de points de l'enveloppe convexe dans le sens horaire

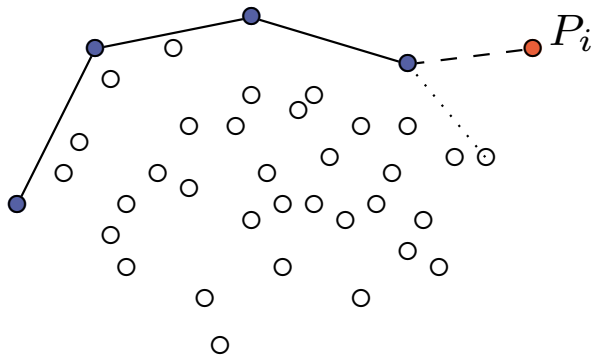
Complexité ?

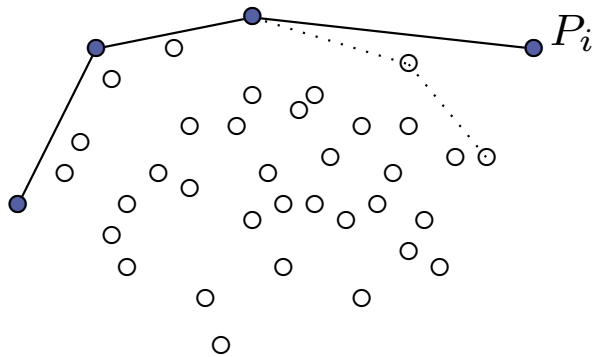
Est-ce qu'on peut faire mieux ?

Algorithme incrémental / itératif

- trier les points dans l'ordre croissant d'abscisses
- considérer les points un par un selon cet ordre
- à l'itération i, on crée une solution valide pour les i premiers points







- Remarques :
  - les points avec la plus petite et la plus grande abscisse appartiennent toujours dans l'enveloppe convexe
  - appliquer la même procédure pour trouver l'enveloppe supérieure et ensuite l'enveloppe inférieure

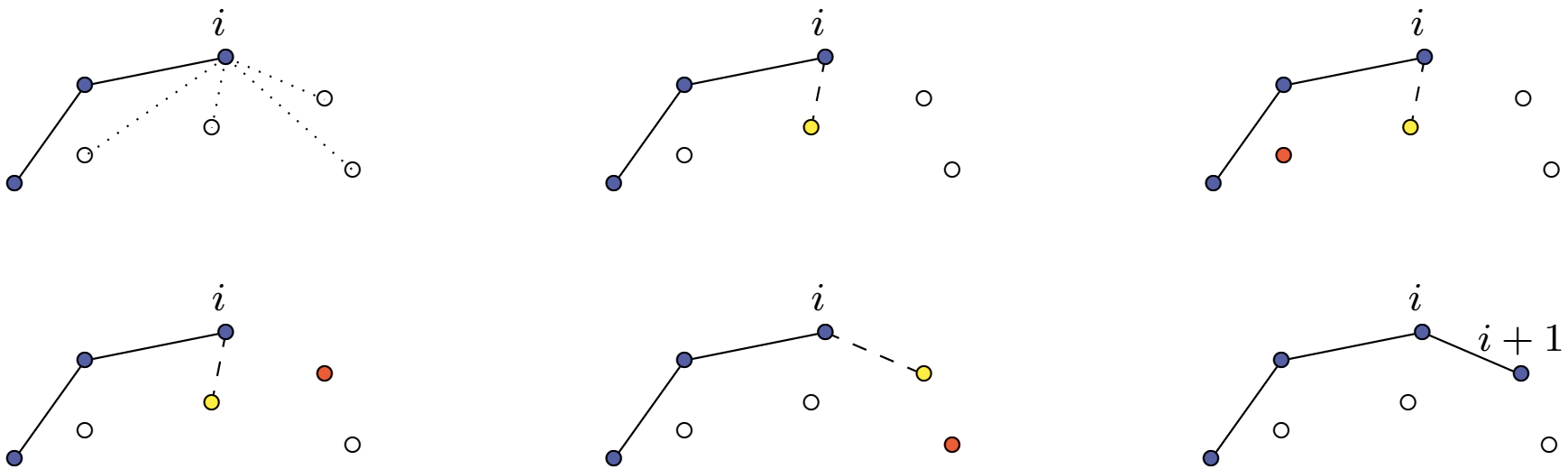
Algorithme ConvexHull

1: Trier les points par ordre croissant des abscisses. Dans le cas d'égalité, trier par ordre croissant des ordonnées. Soit $P_1, P_2, \ldots, P_N$.
2: $\mathcal{H}_{sup} \leftarrow \{P_1, P_2\}$ // *Liste de points de l'enveloppe supérieure*
3: **pour** $i \leftarrow 3$ à N **faire**
4: Insérer P_i à la fin de la liste $\mathcal{H}_{sup}$.
5: **tant que** $|\mathcal{H}_{sup}| > 2$ **et** P_s est à gauche de $\overrightarrow{P_q P_r}$ (P_q, P_r, P_s sont les 3 derniers points de $\mathcal{H}_{sup}$ dans cet ordre) **faire**
6: Supprimer P_r de $\mathcal{H}_{sup}$.
7: Répéter la même procédure (lignes 2–6) dans l'ordre inverse afin de créer la liste de points de l'enveloppe inférieure $\mathcal{H}_{inf}$ ($\mathcal{H}_{inf}$ est initialisée avec $\{P_N, P_{N-1}\}$ et dans la boucle **pour** i va de $N-2$ à 1).
8: Supprimer le dernier point de $\mathcal{H}_{sup}$ et le dernier point de $\mathcal{H}_{inf}$.
9: Concaténer $\mathcal{H}_{sup}$ et $\mathcal{H}_{inf}$ afin de créer la solution finale $\mathcal{H}$.

Complexité ?

Est-ce qu'on peut faire mieux si la solution est petite ?

- Rappel : le point d'abscisse minimale appartient à notre solution
- Supposons qu'on a les i premiers points de l'enveloppe convexe finale dans le sens horaire en commençant par le point d'abscisse minimale
- Le point à ajouter dans la solution à l'itération $i+1$ est celui que se trouve le plus à gauche possible par rapport au point i



- Attention aux points colinéaires dans "le plus à gauche possible" !

Algorithme RapidConvexHull

1: Trouver le point avec l'abscisse minimale, soit $P_{\min}$.
2: $\mathcal{H} \leftarrow \{P_{\min}\}$ // *Liste de points de l'enveloppe convexe*
3: **faire**
4: Insérer le premier point (candidat) de l'entrée à la fin de la liste $\mathcal{H}$.
5: **pour** tout point P de l'entrée sauf A et B, où A et B sont respectivement le avant dernier et le dernier point de $\mathcal{H}$ **faire**
6: **si** P est à gauche de $\overrightarrow{AB}$ ou fait partie du segment $\overrightarrow{AB}$ **alors**
7: Supprimer le dernier point de $\mathcal{H}$ (c'est-à-dire B).
8: Insérer P à la fin de $\mathcal{H}$.
9: **tant que** le premier et le dernier points de $\mathcal{H}$ ne sont pas identiques
10: Supprimer le dernier point de $\mathcal{H}$.

Complexité ?

Algorithmes de tri

- on a besoin de trier les points de l'entrée dans l'algorithme ConvexHull
- on s'intéresse sur la complexité en temps mais aussi sur la taille de la mémoire utilisée

Tri par sélection (tri par échanges)

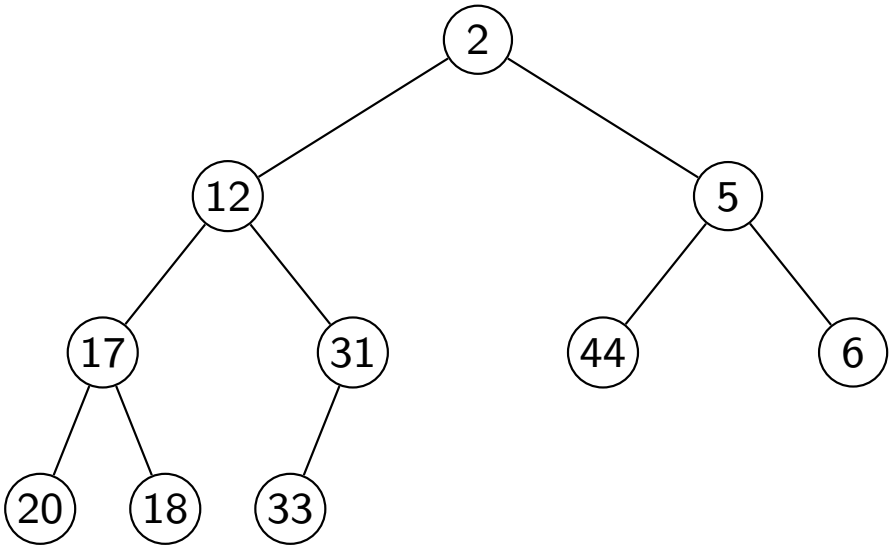
1: T : tableau à trier, N : nombre d'éléments
2: **pour** i de 0 à $N-2$ **faire**
3: Chercher la position $PosMin$ du min entre i et $N-1$.
4: Échanger $T[i]$ et $T[PosMin]$

- algorithme *en place*
- **complexité ?**

Tri par tas

Arbre Binaire Complet

- représentation avec des pointeurs
- représentation avec les indices d'un tableau



0	1	2	3	4	5	6	7	8	9
2	12	5	17	31	44	6	20	18	33

Pour le nœud à la position i

- position de son père : $\frac{i-1}{2}$
- position du fils gauche : $2i + 1$
- position du fils droit : $2i + 2$

Tas =

- Arbre Binaire Complet (avec pointeurs ou tableau)
- Chaque nœud a plus grande priorité que ses fils

Tri par tas

Tri par tas avec arbre

1: T : tableau à trier, N : nombre d'éléments
2: **pour** i de 0 à $N-1$ **faire**
3: Insérer $T[i]$ dans le tas $\mathcal{A}$.
4: **pour** i de 0 à $N-1$ **faire**
5: $T[i] \leftarrow$ Extraire l'élément du $\mathcal{A}$ avec la plus grande priorité.

Tri par tas avec tableau

1: T : tableau à trier, N : nombre d'éléments
2: Réorganiser en place le tableau T pour satisfaire les propriétés d'un tas
3: **pour** i de $N-1$ à 0 **faire**
4: $T[i] \leftarrow$ Extraire l'élément du $\mathcal{A}$ avec la plus grande priorité.

Complexité ?

Structures de données

- **Représentation de l'entrée** (ensemble de points)
  - tableau de taille fixée
- **Représentation d'une solution** (enveloppe convexe, sens horaire)
  - liste doublement chaînée avec pointeur vers le dernier élément
- **Algorithmes de tri**
  - tri par sélection : tableau d'entrée (algorithme en place)
  - tri par tas avec arbre : Arbre Binaire Complet (structure avec pointeurs) + primitives du tas avec cette structure
  - tri par tas avec tableau : tableau d'entrée (algorithme en place) + primitives du tas avec le tableau

Objectif final du projet

- Évaluer et comparer la **performance** (en temps d'exécution)
  - de différentes algorithmes pour calculer l'enveloppe convexe
  - de différentes algorithmes de tri
  - de l'implémentation d'un tas avec tableau ou avec pointeurs

Méthodologie

- ✓ Analyse de complexité théorique
- ✓ Expériences pour valider l'analyse théorique
  - des données vont être fournies début avril

Programmation en C

- **squelette fourni**
  - organisation du code
  - définition des `struct` utilisées
  - prototypes des fonctions
  - prototypes des structures des données
- **à coder**
  - les algorithmes de tri
  - les algorithmes pour calculer l'enveloppe convexe
  - les structures de données utilisées

Attention ! Vous n'avez pas le droit de modifier les prototypes des fonctions et des structures fournis.

Compilation & Exécution

- `Makefile` fourni (à modifier si besoin)

- `make` : compilation
- `make run` : compilation & exécution
- `make clean` : suppression de fichiers intermédiaires
- `make clean run` : nettoyage, compilation & exécution
- `make memorycheck` : utilisation du `valgrind` pour examiner l'état de la mémoire à la fin de l'exécution de l'algorithme

Remarques générales sur le code C

- Pourquoi utiliser `static` pour certains fonctions/procédures ?
  - sous-programmes privés dans le fichier qui sont définies
  - on ne peut pas les appeler dehors de ce fichier
  - pas de prototype dans le fichier `.h`
  - objectif : protéger nos structures de données
- Paramètres de fonctions avec `const`
  - objectif : garantir que l'argument ne sera pas modifié par la fonction
  - exemple : une procédure qui affiche un point ne doit pas le modifier
- Le type de données `void*`
  - objectif : définir de données génériques
  - le type est défini dans l'utilisation
  - exemple : une liste chaînée peut contenir d'entiers ou de points ...

Utilisation du type void*

```
// créer un entier
int n = 3;

// créer une variable générique
// il s'agit d'un pointeur !
void* v;

// affecter l'entier n à la variable générique
v = &n;

// récupérer le contenu de v
int* contenu = (int*) v;
```

- attention à la récupération du contenu
  - on doit faire une **conversion de type**

Présentation du canevas...

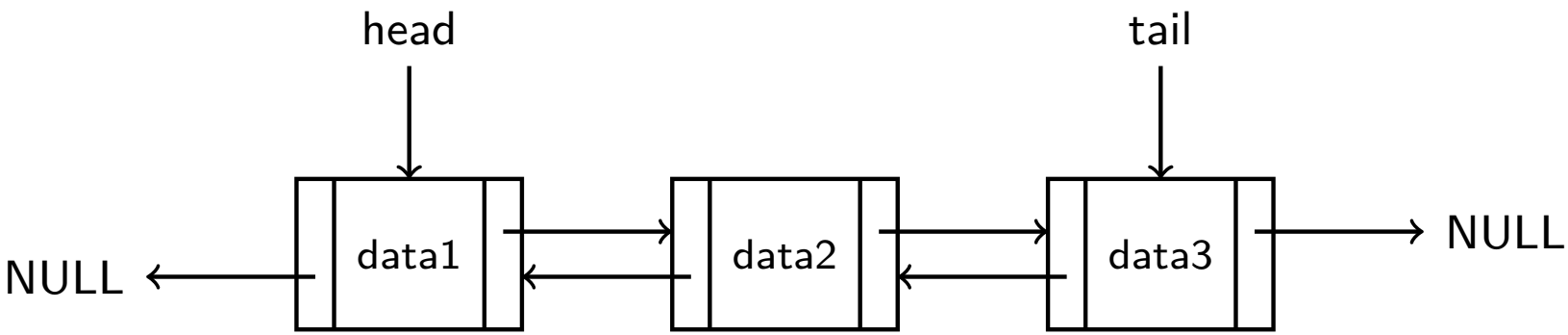
- `util.h` et `util.c` (code fourni)

Présentation du canevas...

- `util.h` et `util.c` (code fourni)
- `list.h` et `list.c`

Listes doublement chaînées

- structure itérative

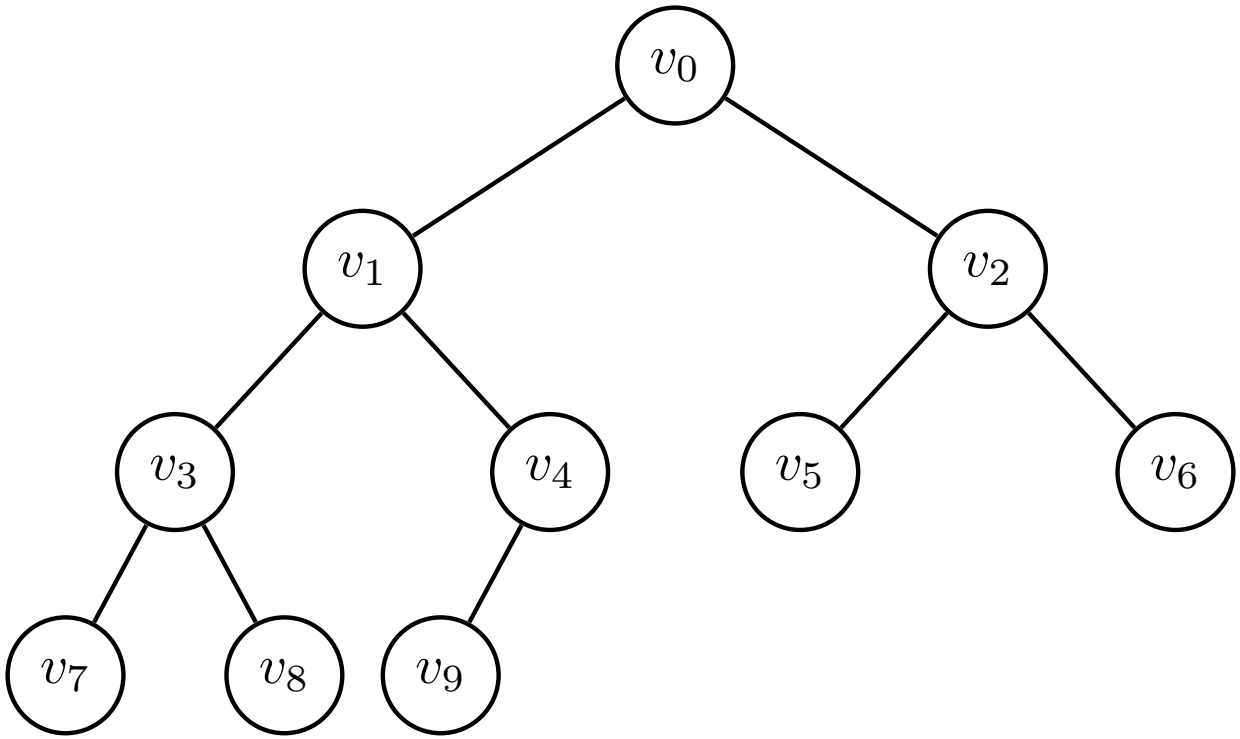


Présentation du canevas...

- `util.h` et `util.c` (code fourni)
- `list.h` et `list.c`
- `tree.h` et `tree.c`

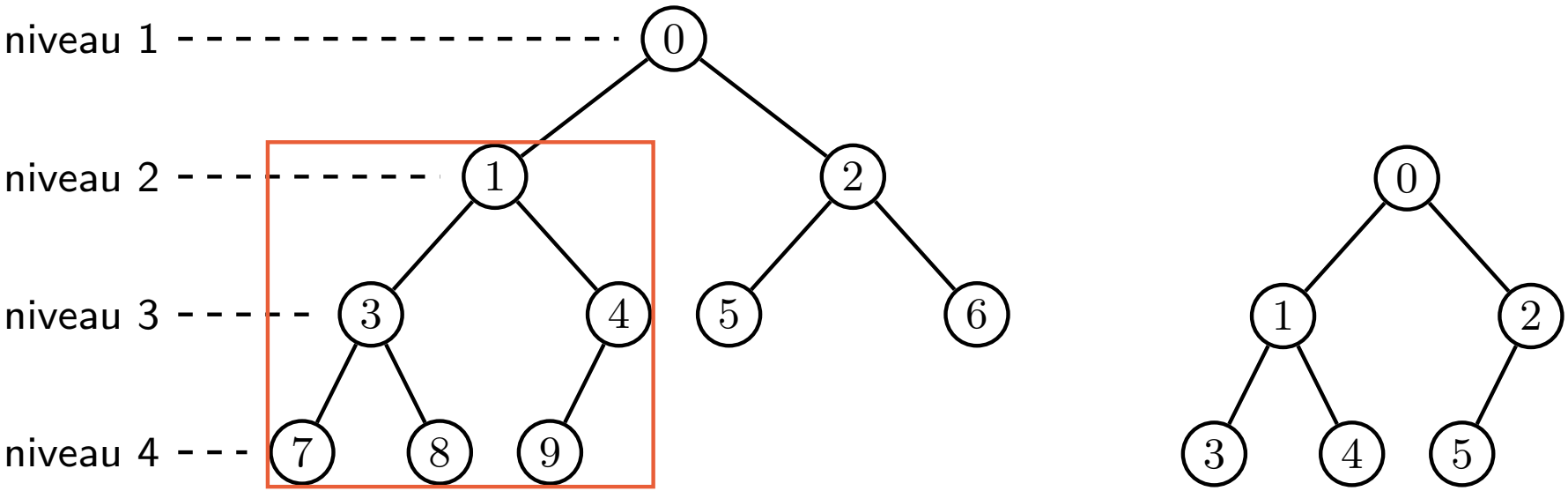
Arbres Binaires Complets

- **Structure récursive**
- Tous les niveaux sont remplis à l'exception éventuelle du dernier
- Au dernier niveau les feuilles sont alignées à gauche



Arbres Binaires Complets : Insertion

- position à insérer : dernier niveau, le plus à gauche possible
- comment guider la recherche de la position à insérer ?
  - ⇒ avec le numéro de la position (nb d'éléments après l'insertion - 1)
  - ⇒ on peut déduire le niveau à insérer et le nombre de feuilles à ce niveau (après l'insertion)
  - ⇒ si ce niveau sera rempli moins que la moitié, alors on va à gauche, sinon on va à droite



- **Procédure récursive** : le numéro de la position doit être mis à jour

Arbres Binaires Complets

- Autres opérations :
  - extraire la donnée du dernier nœud et le supprimer
  - consulter le dernier nœud
- Même principe que pour l'insertion

Complexité ?

Présentation du canevas...

- `util.h` et `util.c` (code fourni)
- `list.h` et `list.c`
- `tree.h` et `tree.c`
- `heap.h` et `heap.c`

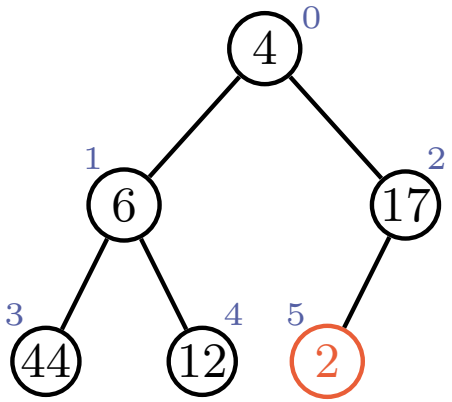
Tas

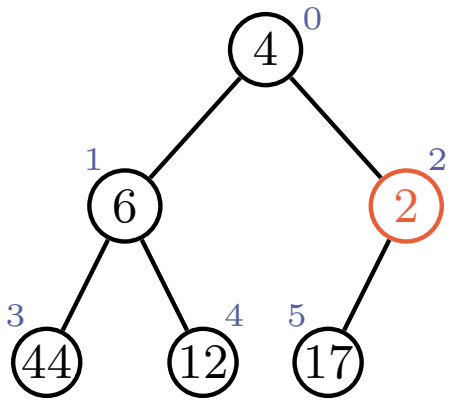
- Tas implémenté avec **un tableau** (`ArrayHeap`)
- Tas implémenté avec **des pointeurs** (arbre binaire complet, `CBTHeap`)
- 2 primitives de correction
  - correction vers le haut

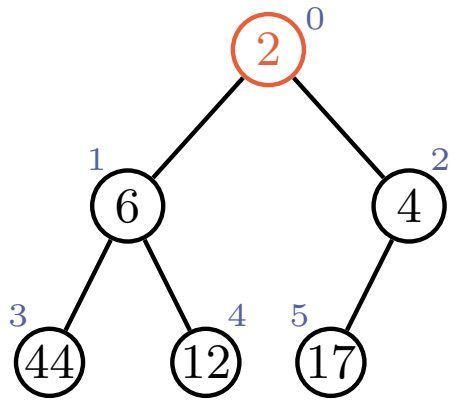
0	1	2	3	4	5
4	6	17	44	12	2

0	1	2	3	4	5
4	6	2	44	12	17

0	1	2	3	4	5
2	6	4	44	12	17



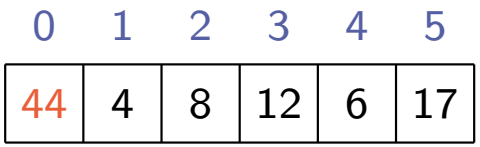


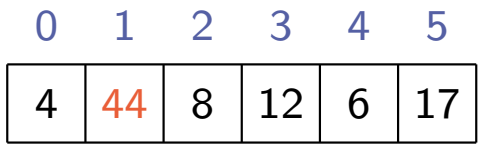


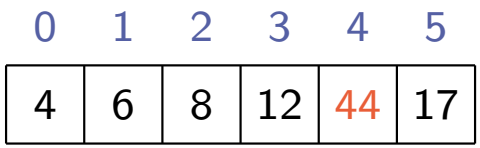
- Implémentation un peu plus difficile avec les arbres, car on n'a pas accès direct vers le nœud concerné (approche similaire à l'insertion dans un arbre binaire complet)

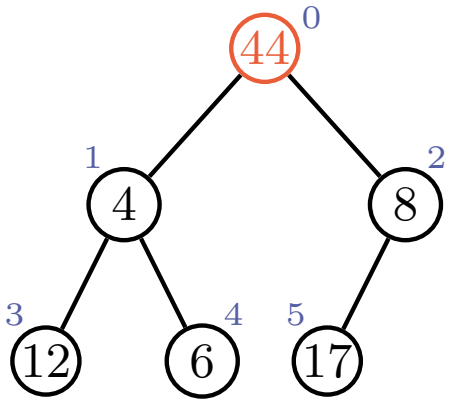
Tas

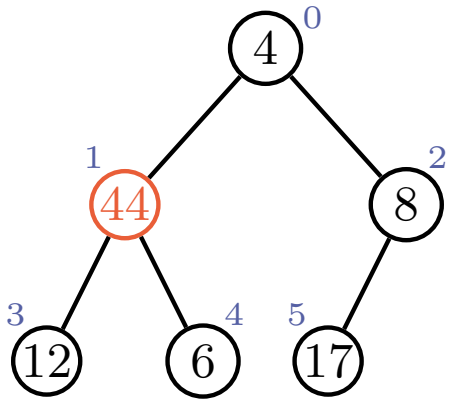
- Tas implémenté avec **un tableau** (`ArrayHeap`)
- Tas implémenté avec **des pointeurs** (arbre binaire complet, `CBTHeap`)
- 2 primitives de correction
  - correction vers le haut
  - correction vers le bas

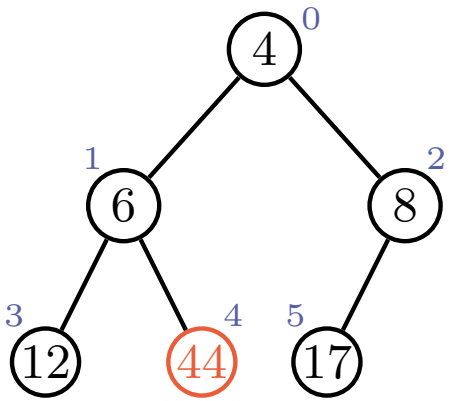












Tas

- Tas implémenté avec **un tableau** (`ArrayHeap`)
  - pas de primitive d'insertion
  - `ArrayToArrayHeap` : réorganisation en place du tableau en entrée afin de satisfaire la propriété du tas, en utilisant la correction vers le bas.
- Tas implémenté avec **des pointeurs** (arbre binaire complet, `CBTHeap`)
  - `CBTHeapInsert` : primitive d'insertion an utilisant la correction vers le haut.
- **Dans les deux cas**
  - primitive d'extraction de la valeur avec la plus grande priorité
    1. Échanger la racine avec le dernier élément.
    2. Corriger la position du nouveau élément à la racine en utilisant la correction vers le bas.
    3. Restituer l'élément extrait.

Présentation du canevas...

- `util.h` et `util.c` (code fourni)
- `list.h` et `list.c`
- `tree.h` et `tree.c`
- `heap.h` et `heap.c`
- `sort.h` et `sort.c`

Présentation du canevas...

- `util.h` et `util.c` (code fourni)
- `list.h` et `list.c`
- `tree.h` et `tree.c`
- `heap.h` et `heap.c`
- `sort.h` et `sort.c`
- `geometry.h` et `geometry.c`

Présentation du canevas...

- `util.h` et `util.c` (code fourni)
- `list.h` et `list.c`
- `tree.h` et `tree.c`
- `heap.h` et `heap.c`
- `sort.h` et `sort.c`
- `geometry.h` et `geometry.c`
- `algo.h` et `algo.c`

Format du fichier contenant un ensemble des points

- la premier ligne contient le nombre des points du fichier
- à partir de la deuxième ligne, chaque ligne decrit un seul point et contient l'abscisse et l'ordonnée du point.

```
12
0  5
4  7
5  10
3  6
7  6
4  3
10  5
6  7
7  4
6  3
3  4
5  0
```

Présentation du canevas...

- `util.h` et `util.c` (code fourni)
- `list.h` et `list.c`
- `tree.h` et `tree.c`
- `heap.h` et `heap.c`
- `sort.h` et `sort.c`
- `geometry.h` et `geometry.c`
- `algo.h` et `algo.c`
- `main.c`

Interface graphique

- JavaFX
- cours “Interfaces graphiques”

Interface graphique : qu'est-ce qu'on attend ?

- donner l'entrée du problème de l'enveloppe convexe (un ensemble des points) :
  - choisir un fichier contenant les points
  - créer un ensemble des points en cliquant avec la souris sur une fenêtre de l'interface et enregistrer ces points dans un fichier
- choisir l'algorithme qui calcule l'enveloppe convexe (1–SlowCOnvexHull, 2–ConvexHull, 3–RapidConvexHull)
- choisir l'algorithme de tri dans le cas où le choix 2–ConvexHull est sélectionné avant (1–tri par tas avec arbres, 2–tri par tas avec tableau, 3–tri par sélection)
- appeler la bonne fonction qui calcule l'enveloppe convexe implémentée en C avec les paramètres choisis (fichier d'entrée, algorithme de tri si besoin) ainsi que le nom du fichier où la solution va être enregistrée
- afficher les points d'entrée ainsi que la solution (par exemple, en désignant le polygone qui correspond à l'enveloppe convexe)

Comment appeler une fonction C de Java ?

- JNI : Java Native Interface
  - on fera un TP guidé après les vacances de printemps

`http://web.archive.org/web/20120419230023/http://java.sun.com/docs/books/jni/html/start.html`

Proposition d'organisation des séances TP

1. initiation du GIT & listes chaînées
2. listes chaînées & début arbres binaires complets
3. arbres binaires complets
4. tas
5. tri
6. enveloppe convexe
7. TP guidé JNI
8. interface graphique
9. interface graphique

Évaluation

- Utilisation du GIT : 5%
- Utilisation des primitives : 5%
- Listes doublement chaînées : 10%
- Arbres binaires complets : 10%
- Tas : 10%
- Tri : 10%
- Géométrie : 5%
- Enveloppe convexe : 10%

CCI 2
dépôt 1 : 29 avril
65% du projet
coef. 0.3

- Interface graphique + JNI : 15%
- Rapport : 12.5%
- Soutenance : 12.5% (semaine de 23 mai)

CCI 1
dépôt 2 : 20 mai
40% du projet
coef. 0.2

Évaluation

Il n'y aura pas de seconde chance !

- Calcul de la note Seconde Chance de l'UE Interfaces Graphiques/Projet de Synthèse
  - $IG = 0.2 * CCI1 + 0.3 * CCI2$
  - $PS = 0.2 * CCI1 + 0.3 * CCI2$
  - $SCIG$: note de seconde chance pour le module Interfaces Graphiques
  - note finale SC $= \max\{IG + PS, SCIG + PS\}$

Les dépôts du projet

Dépôt 1

1. le répertoire `.git/`
2. le code en C (fichiers .h et .c, Makefile)
3. un README (facultatif) : comment lancer le programme C, explications divers, etc

Dépôt 2

1. le répertoire `.git/`
2. le code en Java (fichiers .java, Makefile)
3. un rapport de maximum 5 pages (format pdf)
4. un README : comment lancer/compiler l'interface Java

Les dépôts du projet

- un projet soumis par binôme (en deux parties)
- **dates limites**
  - dépôt 1 : vendredi 29 avril, 23h59
  - dépôt 2 : vendredi 20 mai, 23h59
- pénalité d'un point par jour de retard
- CCI 1 : ABI si aucune soumission jusqu'à 2 mai, 23h59
- **Attention** au plagiat !
  - les projets seront passer par un détecteur de plagiat de code

Soutenance

- semaine du 23 mai
- 30 minutes par binôme
  - 15 minutes de présentation
  - 15 minutes des questions et délibération
- tous les membres de l'équipe participeront
- différente note pour chacun possible
- il faut préparer une présentation (powerpoint)

Consignes pour le rapport

- **5 pages maximum**
- le rapport n'est pas une documentation du code
- introduction / conclusions
- expliquer par exemple :
  - Quels outils vous avez utilisé ?
  - Quelles fonctions importantes supplémentaires vous avez introduit et pourquoi ?
  - Quelles étaient les difficultés que vous avez rencontrées ?
  - Quelles améliorations envisageriez vous ?
- **évaluation de performance**
  - théorique – complexité
  - Quelle structure de données fonctionne le mieux en termes de temps d'exécution pour de petits instances ? Pour de instances moyennes ? Pour de grandes instances ? Pourquoi ?

Conseils générales

- utiliser GIT correctement et régulièrement
- partager les tâches avec vos coéquipiers
- écrire des commentaires utiles
- définir des nouvelles fonctions si besoin
- travailler en dehors des heures TP
- **ne pas modifier** les prototypes des fonctions et des structures fournies

Tester – tester – tester ! ! !
chaque fonction séparément et avant continuer

- A titre indicatif
  - .c du projet : 1370 lignes
  - fonctions de test : 2060 lignes

Définir

1. fonctions de test
2. données de test

Fonctions de test – exemple listes

```
static int compare_lists(List *l1, int* l2[], int size) {
    if (getListSize(l1) != size)
        return 0;
    if (listIsEmpty(l1))
        return 1;
    LNode* curr = Head(l1);
    int i = 0;
    while (curr != NULL) {
        if (getLNodeData(curr) != l2[i])
            return 0;
        curr = Successor(curr);
        i++;
    }
    curr = Tail(l1);
    i = size-1;
    while (curr != NULL) {
        if (getLNodeData(curr) != l2[i])
            return 0;
        curr = Predecessor(curr);
        i--;
    }
    return 1;
}
```

Données de test – exemple listes

```
void test_listInsertLast() {
    int *i1 = malloc(sizeof(int));
    int *i2 = malloc(sizeof(int));
    int *i3 = malloc(sizeof(int));
    *i1 = 1.1;
    *i2 = 2.2;
    *i3 = 3.3;

    List *L = newList(viewInt, freeInt);
    int* tab[3];
    tab[0] = i1; tab[1] = i2; tab[2] = i3;

    listInsertLast(L, i1);
    if (compare_lists(L, tab, 1) == 0) printf("problème");
    listInsertLast(L, i2);
    if (compare_lists(L, tab, 2) == 0) printf("problème");
    listInsertLast(L, i3);
    if (compare_lists(L, tab, 3) == 0) printf("problème");
}
```

Questions ? ? ?