S4 2019-2020

Nathan TONNELLE

Algorithmique

INFO0401

Table des matières

[La note du TP : 2](#_Toc32991845)

[TP : tous les algos du TD 2](#_Toc32991846)

[Les tris 4](#_Toc32991847)

[Le tri à bulle 4](#_Toc32991848)

[Principe 4](#_Toc32991849)

[Algorithme : 4](#_Toc32991850)

[Tri couleur 4](#_Toc32991851)

[Algo 5](#_Toc32991852)

[Tri rapide 5](#_Toc32991853)

[Algo 5](#_Toc32991854)

[Tri topologique 6](#_Toc32991855)

[Algo 7](#_Toc32991856)

[Algo 7](#_Toc32991857)

[Algo 8](#_Toc32991858)

[Tri par tas 8](#_Toc32991859)

# La note du TP :

A faire dans un document

Tous les algos qu'on fait pendant l'année sont programmé et dans un dossier (algo TD, TP)

Divisée en 3 parties :

Document 40% note TP

Partie exécutive : 30% note TP

Oral : 30% note TP

Question sur un algo tirer au hasard (montrer son exécution)

# TP : tous les algos du TD

Ecrire l'algo : (langue algo)

Donner son utilisation

Quelques propriétés

Donner sa complexité (estimé)

Code

Exemple d'exécution

Horaire

Piles :

* Implémentation d'une pile
* Différents algos
  + - Remplir une pile
    - Vider une pile
    - Les fonctions de bases pile :
      * Est\_Vide(p)
      * Sommet(p)
      * Eunparler(p)
      * Dépiler(p)
    - Egalite de 2 piles
    - Tuer une pile
    - Recherche d'une pile occurrente de x depuis une pile
    - Fonction Ackermann
* Liste chainée
  + - Construction d'une liste chainée
    - Vider une liste chainée
    - Insertion d'un élément dans une liste chainée (au début, milieu, fin)
    - Soustraire un élément dans une liste (3 config)
    - Concaténation de 2 listes chainée
    - Les opérations logiques sur liste chainée (et ou intersection, union)
    - Ajouter en fin de liste (l)
    - Addition de 2 polys en utilisant les listes chainées
    - Trier une liste chainée
    - Fonction existante : est-ce qu'une liste est contenue dans une liste

Extrait (C1, C2) : Boolean

* Les 2 premières valeurs de C1, C2 sont égales
* Les2 dernières valeurs de C1, C2 sont égales
* Des autres sont égales 2 à 2

Tris :

* Tris à bulles
* Tris rapides
* Tri topologique
* Tri couleur
* Tri par parties

Les arbres

* Les différents parcours
* Construction d'un arbre
* Recherche d'un élément dans un arbre binaire
* La hauteur d'un arbre binaire
* St-h équilibré
* Affichage d'un arbre
* Calcul des nœuds internes et externes d'un arbre
* Calcul des nombres de feuilles
* Egalité entre 2 arbres binaires
* L'arbre d'Hoffmann
* Un arbre 2 3

# Les tris

Organiser une collection d'objets selon un ordre déterminé. Les objets à trier sont des éléments d'un ensemble muni d'un ordre total. Souvent ces objets sont des entiers. On utilise le plus souvent un tableau non dimensionnel pour réaliser ce tri.

## Le tri à bulle

### Principe

#### Ev

13 14 15 21 6 7 3 2 5 1

Comparer 2 valeurs adjacentes et les échanger jusqu'à remonter la plus grande valeur.

A la 1ere itération : on remonte la plus grande valeur

A la 2eme itération : on remonte la 2eme plus grande valeur

A la 3eme itération : on remonte la 3eme plus grande valeur

… jusqu'à la dernière valeur contenue dans la 1ere case du tableau qui constitue la primitive de base égale à elle même

1ere T 13 14 15 6 7 3 2 5 1 **21**

2eme T 13 14 6 7 3 2 5 1 **15 21**

3eme T 13 6 7 3 2 5 1 **14 15 21**

4eme T 6 7 3 2 5 1 **13 14 15 21**

5eme T 6 3 2 5 1 **7 13 14 15 21**

6eme T 3 2 5 1 **6 7 13 14 15 21**

7eme T 2 3 1 **5 6 7 13 14 15 21**

8eme T 2 1 **3 5 6 7 13 14 15 21**

9eme T 1 **2 3 5 6 7 13 14 15 21**

10eme T **1 2 3 5 6 7 13 14 15 21**

### Algorithme :

Données :

T tableau d'entiers

I, inf, sup, taille : entier

Début :

Lire (sup)

Lire (t)

Taille <- sup

Tant que taille > 1 faire

Pour i allant de 1 à taille-1 faire

Si T[i] > T[i+1] alors

Echanger (t[i],T[i+1])

finSi

FinPour

Taille <- taille-1

FinTantQue

Afficher (T)

Fin

## Tri couleur

Hollandais : **000**0000000

T 000**0**00**0**000**00**000**0**0**0** initial

Plusieurs approches pour réaliser ce tri couleur

T **000** 000 **00**00000**0**0 0000

Classé classé non classé classé

i j j+1 m

### Algo

Données :

T : tableau

I, j, k : entier

N : entier

Début :

Lire(T)

I<-0

J<-0

K<-n

Tant Que (k=/= j) faire

Si t[j+1]=bleue alors

Echange(t[j+1],t[i+1])

I++

J++

Sinon si (t[j+1]=rouge) alors

Echange( t[j+1],t[k])

k—

sinon

j++

finSi

FintTantQue

Fin

T : A B C A C B C B i=0, j=0, k=taille

T : A B C A C B C B i=0, j=1, k=8

T : B A C A C B C B i=0, j=2, k=8

…

## Tri rapide

Ex : 1 3 15 5 14 6 21 2 13 7

Sur une partie du tableau à traiter avant la séparation en 2 parties

2 4 5 3 8 1 21 16 9

Progression de i tant que t[i]<pivot

### Algo

I, j, k, droite, gauche, pivot, taille, sup, inf : entier

Début :

Droite<-sup;

Gauche <- inf

Pivot<-T[droite]

J<-droite-1

I<-inf

Tant Que i =/= j faire

Tant Que (T[i]<pivot) & (i=/= j)faire

I<-i+1

Fin tant que

Tant que (T[j]>pivot)&(i=/=j) faire

j<-j+1

fin tant que

echanger (T[i], T[j])

fin tant que

si T[i]>pivot alors

echanger (T[i], pivot)

finSi

si i> gauche +1

T1 rapide (T,inf,i)

Sinon

T1 rapide (T,i+1,sup)

Fin Si

Fin

1 3 15 5 14 6 21 2 17 7

1 3 2 5 14 6 21 15 13 7

1 3 2 5 6 14 21 15 13 7

1 3 2 5 6 7 21 15 13 14

## Tri topologique

On utilise une matrice ou une liste d'adjacence (vecteur de pointeurs sur listes chainées). Par tri topologique, on entend gérer un projet (ensemble de données organisées selon leurs exécutions).

Ce tri découle d'une organisation de sommets d'un graphe.

A🡪B A--B 🡺on peut aller de A à B ou de B à A 🡸🡺 A🡨🡪B

(arc) on peut arête

aller uniquement

de A à B

Graphe orienté :

Aucune dépendance,

Pas d'arc entrant

🡪 S1 🡪 S2 🡪 S3 🡪 S5 🡪S6

🡪 S4 🡪 S5

🡪 S4

|  |  |
| --- | --- |
| G | P |
| 1 | 🡪2|\ |
| 2 | 🡪3|4|\ |
| 3 | 🡪5|\ |
| 4 | 🡪3|5|\ |
| 5 | 🡪6|\ |
| 6 | 🡪\ |

🡺 Matrice de liens, liste d'adjacence i

Graphe qui n'est pas dense

Plus rapide 🡨

S2

S1

Sommet sommet

Arc sortant arc entrant

Vect-, V+

Entrant sortant

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | S1 | S2 | S3 | S4 | S5 | S6 | V+ |
| S1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 |
| S2 | 0 | 0 | 1 | 1 | 0 | 0 | 2 |
| S3 | 0 | 0 | 0 | 0 | 1 | 0 | 1 |
| S4 | 0 | 0 | 1 | 0 | 1 | 0 | 2 |
| S5 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| S6 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| V- | 0 | 1 | 2 | 1 | 2 | 1 |  |

**Matrice**

### Algo

Données

V+ vecteur d'arc sortant

V- vecteur d'arc de dépendance

G matrice de lien

I, j entiers

Début :

Initialisation V+ à 0

Initialisation V- à 0

Pour i allant de 1 à n faire

Pour j allant de 1 à n faire

Si G[i,j]=1 alors

V+[i]🡨V+[i]+1

V-[j]🡨V-[j]+1

Fin Si

Fin Pour

Fin Pour

Fin

### Algo

Algo de calcul V- et V+ en utilisant les listes d'adjacence

Données

G vecteur entier

Nb de sommet, j, i entier

V+, V- vecteur entier

\*p pointeur entier

Début

Initialiser (V+ et V-) à 0

Pour i allant de 1 à nb\_de\_sommets faire

P🡨G[i]

Tant que p!=NULL faire

V+[i]🡨V+[i]+1

V-[p🡪val]🡨V-[p🡪val]+1

P🡨(p🡪suivant)

Fin tant que

Fin pour

Fin

Ce qui nous intéresse c'est le V- : vecteur de dépendance (arc entrant)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 1 | 2 | 1 |
| S1 | S2 | S3 | S4 | S5 | S6 |

V-

^-> c'est le sommet à exécuter le 1er car il n'a pas de dépendances

### Algo

Données

V- vecteur d'entier déjà calculé

Nb\_de\_sommets, i, j, l, h : entier

G : matrice

Début

Pour i allant de 1 à nb\_de\_sommet faire

J🡨1 //recherche de 0 dans V-

Tant que V-[j] 0 faire

J🡨j+1

Fin tant que

//on construit la liste avec j

L={1,2,…}

V-[j]🡨-1

Pour k allant de 1 à nb\_de\_sommet faire

Si G[j,h]=1 alors

V-[k]🡨V-[k]-1

Fin Si

Fin pour

Fin pour

Fin

## Tri par tas

En utilisant une structure arborescente

Tas minimum : clé(père) >= clés (fils)

Tas maximum : clé(père) >= clé (fils)

Minimum

Père

Père

10

Fils Droit

Fils Gauche

8

4

1

3

Recherche père : kaiv2

V 8 1 5 6 15 11

1 2 3 4 5 6

Recherche

-------------🡪

Fils : 2k

2k+1

Ex : père de 11 est 5

Fils de 1 sont à 2k et 2k+1 6 et 15

4 5 10 1 4 8 15 41

1 2 3 4 5 6 7 8

1er dormation au tas minimum clé (père)>clé (fils)

2eme echanger (T[1];T[n])

3eme preconstruire le tas

4-> 5-> 10-> 10-> 10-> … 🡪 41

5 4 4 4 5 14 5 12 14

10 5 1 1 4 10 4 5 8

14 1

41 12 14 10 4 5 8 1

Algorithme de détection d'un cigle {arbre dégénéré, graphe}

Cigle :

quand le graphe n'est pas orienté

circuit quand le graphe est orienté

schéma 0

une grande partie de l'algorithme ne fonctionne pas s'il y a cycle (circuit)

tels que tri topologique, recherche du plus court chemin

1---2 🡪 arrête

1🡪2

1🡨🡪2 arcs

1🡨2

Liste d'adjacence

Schéma 1

nom : detection de cigle

donnée :

g : pointeur sur liste chainée

p : pointeur

tvisite : tableau d'entiers

i, j, nbNoeuds : entiers

trouve : boolean

algo :

lire nbNoeuds

//initialisation tvisite à 0;

trouve <- faux;

i <- 1;

tant que nbNoeuds > 0 && trouve == faux faire

p <- G[i];

tant que p!=NULL && trouve == faux faire

j<-p->val

//on suppose que la valeur du noeud j > i

si j > i alors

si tvisite[i]==0 || tvisite[j]==0 alors

tvisite[i]<-1;

tvisite[j]<-1;

fin si

trouve<-vrai

fin si

p<-p->suivant;

i<-i+1;

fin tant que

nbNoeuds<-nbNoeuds-1;

fin tant que

afficher (trouve);

Arbre de recouvrement de poids minimum

Soit (G=(S,A) un graphe connexe dans lequel toute arrête de

A est pondéré par un poids P, un arbre de recouvrement

G est un arbre libre (graphe connexe sans cicle) joignant tous les sommets

G, le poids de cet arbre recouvrant est la somme des poids

Toutes les arrêtes qu'il contient

Schéma 2

**Connexe :** une seule entité 🡺 pas de sous graphe

**Fortement connexe :** chaque sommet peut être visité par tous les autres sommets

Schéma 3

**Matrice** G : graphe : donnée

A : arbre : résultat

Initialisation des 3 paramètres

Distance : 1ère ligne de G

Plus proche : tous proche de 1

SA = [1][0][0][0][0][0][0]

1

= max

Schéma 4

nom : arbre de recouvrement

complexité :

donnée :

G, A : matrice

distance, sa, plusProche : vecteur

jG, iA, j, nbSom, i, dis : entier

algo :

//initialisation tu tableau distance

pour j allant de 1 à nbSom faire

distance[j] <- G[1,j];

plusProche[j] <- 1;

fin pour

sa[1] <- 1;

pour j allant de 2 à nbSom faire

sa[j] <- 0;

fin pour

//recherche du min

pour i allant de 2 à nbSom faire

dis <- max; //999

jG <- 2;

pour j allant de 2 à nbSom faire

si sa[j]!=sa[1] && distance[j]<10 alors

dis <- distance[j];

jG <- j

fin si

fin pour

sa[jG] <- 1

iA <- plusProche[jG]

fin pour

A[iA,jG] <- G[iA,jG];

A[jG,iA] <- G[jG,iA];