Algorithmique

Fiche

0.1 Introduction

tab titre

...	...	...

0.2 Types de Données Abstraits

0.2.1 Les arbres binaires de recherche

Définition : un arbre binaire est un A.B.R. si pour tout nœud s, les contenus des nœuds du sous-arbre gauche de s sont inférieurs ($\leq$) au contenu de s et les contenus du sousarbre droit sont supérieurs ($>$) au contenu de s.

Accesseurs : a.contenu; a.sAG; a.sAD.
Opérations usuelles :
a.insérer(x) : insère l'élément x dans l'arbre;
a.maximum() et a.minimum()
a.supprimer(x) : supprime un élément.
a.rechercher(x)

0.2.2 Les tas

Définition : un tas est un arbre binaire presque complet tel que pour tous nœuds n sauf la racine on a : n.pere.contenu $\geq$ n.contenu

Accesseurs : t.contenu; t.fG; t.fD; t.pere
Opérations usuelles : t.insérer(x) : insère l'élément x dans le tas;
t.maximum() : retourne l'élément maximum;
t.extraire() : supprime un élément maximum.

Les tas : relation de filiation
L'implémentation d'un tas par un tableau admet quelques propriétés :
racine : noeud 1;
parent du nœud i : nœud(i Div 2);
fils gauche du nœud i : noeud(2i);
fils droit du noeud i : noeud(2i + 1)

Definition:
Un type de données abstrait est composé d'un ensemble d'objets, similaires dans la forme et dans le comportement, et d'un ensemble d'opérations sur ces objets.
L'implémentation d'un T.D.A. ne suis pas de schéma préétabli. Il dépend des objets manipulés et des opérations disponibles pour leur manipulation

0.2.3 Les Contraintes d'implémentation

L'implémentation d'un type de données abstrait doit respecter deux contraintes :
- Utiliser un minimum d'espace mémoire;
- Exécuter un nombre minimal d'instructions pour réaliser une opération.

0.2.4 Les Ensembles dynamiques

Définition : On appelle ensemble dynamique e un ensemble fini d'éléments issus d'un ensemble discret (entiers, chaîne de caractères,...) et muni d'une relation d'ordre.
Opérations :
e.inserer(x) ajoute un élément x à e;
e.supprimer(x) un élément x de e;
e.rechercher(x)
e.maximum() retourne l'élément maximum de e;
e.minimum() retourne l'élément minimum de e;
e.prédécesseur(x)
e.successeur(x)

0.2.5 Les Dictionnaires

Définition : On appelle dictionnaire un ensemble dynamique d dont on a restreint l'ensemble des opérations :
Opérations : d.insérer(x) : insère l'élément x dans d;
d.rechercher(x) : recherche l'élément x dans d;
d.supprimer(x) : supprime l'élément x de d.

Dictionnaire : Implémentation

Structure de données	Rechercher	Insérer	Supprimer
Tableau non ordonné	O(n)	O(1)	O(1)
Liste non ordonnée	O(n)	O(1)	O(1)
Tableau ordonné	O(log n)	O(n)	O(n)
Liste ordonnée	O(n)	O(1)	O(1)
Arbre de recherche	O(h)	O(h)	O(h)
Tas	O(n)	O(h)	O(h)

0.2.6 Les Piles

Définition : Une pile est un ensemble dynamique tel que la suppression concerne toujours le dernier élément inséré. Une telle structure est aussi appelé LIFO (last-in, first out).
Opérations : p.empiler(x) insère un élément à l'entrée de la pile;
p.dépiler() retourne et supprime l'élément en entrée de pile;

Applications
La pile d'exécution : les appels des méthodes dans l'exécution d'un programme sont gérés par une pile.

Éditeur de texte : une pile est fournie par les éditeurs de texte évolués qui possèdent le couple d'actions 'annuler-répéter'.

Implémentation
On peut implémenter une pile par un couple composé d'un tableau et d'un entier.
Schema Pile + taille
Inconvénient majeur : il faut fixer à l'avance la taille maximale de la pile.

0.2.7 Les Files

Définition : Une file est un ensemble dynamique tel que les insertions se font d'un coté (l'entrée de file) et les suppressions de l'autre coté (la sortie de file). Une telle structure est aussi appelé FIFO (first-in, first out).
Opérations :
f.enfiler(x) ajoute un élément en entrée de file;
f.défiler() supprime l'élément situé en sortie de file.

Implementation:
On peut implémenter une file par un triplet composé d'un tableau et de deux entiers.
Schema File + entre + sortie
-Inconvénient majeur : il faut fixer à l'avance la taille maximale de la file.

-On peut implémenter une pile par un couple de listes chaînées.
Schema File Chaine → debut/fin
Comparaison d'implémentation
Nous avons vu que l'implémentation par les tableaux impose de définir par avance la taille de la file. Ce qui n'est pas cas avec les listes chaînée.
Quelque soit le choix d'implémentation, ce choix n'apparaît pas pour le programmeur puisqu'il n'aura accès à ce type de données que par l'intermédiaire d'un ensemble de méthodes. La file devient alors un type de données abstrait.

0.2.8 Les Files de priorité

Définition : Une file de priorité est une structure de données permettant de gérer un ensemble f d'éléments, chacun ayant une priorité associée appelée clé.
Opérations :
f.insérer(x,clé) : insère l'élément x dans f;
f.maximum() : retourne l'élément de plus grande clé;
f.extraireMax() : retourne et supprime l'élément de f de plus grande clé.

File de priorité : Implémentation

Structure de données	Insérer()	Maximum()	extraireMax()
Tableau non ordonné	O(1)	O(n)	O(n)
Liste non ordonnée	O(1)	O(n)	O(n)
Tableau ordonné	O(n)	O(1)	O(1)
Liste ordonnée	O(n)	O(1)	O(1)
Tas	à étudier	à étudier	à étudier

0.2.9 Les Famille d'ensembles

Définition : Soit X un ensemble muni d'une relation d'ordre <x, on appelle collection (ou famille) un ensemble F de sousensembles de X.
Opérations :
c.insérer(s) : insère le sous-ensemble s dans c;
c.appartient(s) : vérifie si le sous-ensemble s est dans c;
c.supprimer(s) : supprime le sous-ensemble s de c.

0.2.10 Implémentation et complexité

Question : Quelle structure de données permettrait de proposer des algorithmes pour les opérations d'insertion, de vérification d'appartenance et de suppression admettant une complexité indépendante de la taille de la famille?
...+ schema

0.2.11 L'arbre lexicographique

Définition : soit F une famille de sous-ensembles de X, nous associons à F un arbre T(F) lexicographique unique tel que :

- chaque arête de l'arbre est étiqueté par un élément de X;
- à chaque nœud notifié de l'arbre correspond un mot de F;
- à chaque mot de F correspond un chemin unique dans l'arbre tel que ce mot corresponde à la concaténation des étiquettes de ce chemin;
- l'ordre des arêtes d'un chemin coïncident avec l'ordre <x;
- l'ordre des arêtes sortant d'un nœud coïncident avec l'ordre <x.

Implémentation
Remarque : une représentation d'une collection par un arbre lexicographique correspond à un mapping!
En Java un mapping est implémenté par des tables de Hachage.
Plusieurs implémentations différentes d'un arbre lexicographique peuvent être proposées en fonction de la façon dont l'ensemble des fils sont représenté :
- Par un tableau;
- Par des listes chaînées;

0.2.12 La Gestion de partition

Définition : Une partition p d'un ensemble e est un ensemble de parties non vides de e, deux à deux disjointes et dont la réunion est égale à e.
Opérations :
p.trouverClasse(e) : retourne la classe de e dans p;
p.union(c1,c2) : fusionne les deux classes c1 et c2 dans p;
SCHEMA

0.2.13 Pour résumer

• Nous avons défini un T.D.A. comme un ensemble d'objets cohérent muni d'opérations données. Nous avons dit que l'implémentation d'un T.D.A. devait respecter des contraintes d'efficacité (en espace et en temps).
• Nous avons défini les T.D.A. : ensemble dynamique, dictionnaire, pile, file, file de priorité, collection, gestion de partition.
• L'implémentation de chacun de ces T.D.A. repose sur des structures de données évoquées au chapitre précédent : liste, tableau, arbre, tas, arbre binaire de recherche.

0.3 Algorithme Glouton

0.3.1 Le choix glouton

Principe : les algorithmesgloutonssontdes algorithmes pour lesquelsàchaqueitérationon fixe la valeur d'uneou plusieursvariables décrivant le problèmesans remettreen cause les choix antérieurs.
Précisément Le principe consiste àfaire localement le choix qui semble le meilleur, pour se ramener ensuite àla résolution d'un sous problème identique.

0.3.2 Le problème du choix d'activité

Description : soitun ensemble S de n activités concurrentes qui souhaitentutiliseruneressource commune qui ne peutêtre allouée quepour uneactivité à la fois.
Chaque activité possèdeun horairede début di et de fin fi.
Question: Trouver l'ensemble le plus grand possible d'activités compatibles entre elles.

Exemple d'instance
Considérons les activités suivantes:
a1: [5-9] a5: [5-7] a9:[8-11]
a2: [2-13] a6: [3-8] a10:[3-5]
a3: [0-6] a7: [12-14] a11[8-12]
a4: [1-4] a8: [6-10]

Question: Trouver l'ensemble le plus grand possible d'activités compatibles entre elles.

```
    Algorithme choixActivité();
Données : s : liste d'activités; i,j : entier;
Résultat : A : liste d'activités;
Début
n ← s.longueur(); A ← {1}; j ←1;
pour ( i ← 2 à n) faire
        si (di >= fj) alors
                A ← A union i;
                j ← i;
fin pour
Retourner A;
Fin
```

Optimalité
Théorème : l'algorithmechoixActivité() calcule l'ensemble de taillemaximum d'activitéscompatibles.
Elément de démonstration :
- Montrer qu'il existe une solution optimale qui intègre le choix glouton (c'est-à-dire l'activité 1);
- Montrer que le même raisonnement peut être conduit sur S - {1}

Démonstration 1
Montrer qu'il existe une solution optimale qui intègre le choix glouton :
Soit A une solution optimale ordonnées par horaire de fin croissante, soit k la première activité de A :
Si k=1, A est optimale et intègre le choix glouton;
Sinon soit B = A -{k} + {1} avec f1 <= fk
Les activités de B sont disjointes et comme B possède autant d'activités que A, B est optimale.

Démonstration 2
Montrer que si A est une solution optimale sur S, alors A'=A-{1} est une solution optimale sur S'= {i dans S : di >= f1}
Par l'absurde :
si A' n'est pas optimale, alors il existe une solution B' pour S' contenant plus d'activité que A'.
B' union {1} est alors une solution pour S contenant plus d'activités que A.
Ce qui contredit l'hypothèse que A était optimale sur S.

0.3.3 Elément de stratégie gloutonne

La propriétédu choixglouton: on peutarriver à unesolution globalementoptimaleen effectuant un choix localement optimal.
La propriétéde sous structure optimale: un problème fait apparaître unesous structure optimalesiunesolution optimaledu problèmecontient la solution optimaledes sous problèmes.

Validation de la démarche gloutonne
Techniquement :
- Etudier une solution globalement optimale puis montrer que cette solution peut être modifiée pour qu'un choix glouton soit effectué à la première étape;
-Enfin, pour montrer qu'un choix glouton nous ramène à l'étude d'un problème similaire mais plus petit, il suffit de s'assurer qu'une solution optimale fait bien apparaitre des sous structures optimales

0.3.4 Codage de Huffman

Problème Minimiser la taille du codage global d'un fichier texte
Définition Nous appelons codage préfixe un codage ou aucun mot de code n 'est aussi préfixe d'un autre mot de code.

Problème ' codage d'Huffman '
```
Algorithme Huffman();
Données : C : chaîne de caractères;
Résultat: arbre binaire;
Début
n ← C.taille; F ←C;
```

pour (i ← 1 à n-1) faire
z ← nouveauNoeud();
x ← F.extraireMin(); z.gauche ← x;
y ← F.extraireMin(); z.droite ← y;
f(z) ← f(x) + f(y);
F.inserer(z);
fin pour
Retourner F.extraireMin;
Fin

Définition
Soit un alphabet C et un caractère c, f(c) est la fréquence de c dans le fichier et dT(c) la profondeur de la feuille c dans l 'arbre T.
Le nombre de bits requis pour encoder un fichier vaut : B(T) = Σ c dans C f(c) . dT(c)

Propriété du choix glouton
Lemme : Soit C un alphabet et f unefréquence d 'apparition surC. Soient x et y de C ayant les fréquences les plus basses. Il existe alors un codage préfixeoptimal pour C danslequel les mots de code pour x et y ontla même longueuret ne diffèreque par le dernier bit.

Démonstration Pourquoi le cout de l'arbre n'est pas dégradé?
• On supposera f(b)<=f(c) et f(x)<=f(y) d'où f(x)<=f(b) et f(y)<=f(c)
• B(T) – B(T') = $\Sigma_{\ cdansC}$ F(c) . dT(c) - $\Sigma_{\ cdansC}$ F(c) . dT'(c)
= f(x)dT(x) + f(b)dT(b) - (f(x)dT'(x) + f(b)dT'(b))
= f(x)dT(x) + f(b)dT(b) - (f(x)dT(b) + f(b)dT(x))
= (f(b) – f(x)) (dT(b) - dT(x)) >= 0;
• De la même façon l'on pourra montrer que B(T') – B(T") est supérieur à 0;
Ainsi nous avons : B(T") <= B(T); T" est optimal;

Propriété de sous structure optimale
Lemme : Soit T un arbre binairereprésentant un codage préfixe optimal pour C, soient2 caractères x et y quelconquesqui apparaissent comme feuille sœurs dansT, et soitz leurpère. Alors, en considérant z de fréquence f(x) + f(y), l 'arbre T'=T-{x,y} représente un codage préfixeoptimal pour l'alphabet C '= C -{x,y} U {z}

Démonstration
Montrer que la procédure ' Huffman ' calcul un codage optimal sur l'alphabet C / x,y u z;
B(T) = σc dans C f(c) . dT(c)
B(T) = A + (f(x)+ f(y)) (dT(z)+ 1)
B(T') = A + (f(x)) + f(y)) dT(z)

Démonstration
Montrons que le cout B(T) de T peut être exprimé en fonction du cout B(T') de l'arbre T'.
• Rappel : B(T) = $\sigma_{\ cdansC}$ f(c) . dT(c)
• Pour tout c dans C / x,y, on a f(c)dT(c) = f(c)dT'(c)
• D'autre part dT(x) = dT(y) = dT'(z) + 1;
• f(x)dT(x) + f(y)dT(y) = (f(x) + f(y)) . (dT'(z) + 1) = f(z) dT'(z) + f(x) + f(y)
D'où B(T) = B(T') + f(x) + f(y) ;

Algorithme glouton générique
Données : G
Résultat R
Begin ensemble W=
tant que (w ≠ G)
choisir (d dans G w) R = R + traiter (d)
W = W union d
fin tant que
Retourner R
End

0.3.5 Pour résumer

• Le paradigme ' glouton ' consiste à faire un choix local qui maximise un critère donné à un instant donné. Ce choix ne sera jamais remis en cause.
• Le paradigme glouton est adapté aux problèmes pour lesquels les deux propriétés ' du choix glouton ' et de ' sous structure optimale ' sont vérifiées;
• Dans ce chapitre nous avons étudié deux problèmes : - Choix d'activité; - Codage de Huffman;

0.4 Programmation dynamique

'...toute politique optimale est composée de sous-politiques optimales' Bellman

0.4.1 Approches et définitions

Eléments de définition
Principe :La programmation dynamiquerésout les problèmes en combinant les solutions de sous-problèmes. Elle est applicable lorsque les sous-problèmes ne sont pas indépendants.
Un algorithme de programmation dynamique résout chaque sous problème une seule fois et mémorise sa solution dans un tableau. Cela évite ainsi le re calcule de la solution chaque fois que le sous-problème est rencontré.

Deux Approches:
Diviser pour régner
Programmation dynamiqu

0.4.2 Illustration

nombres de Fibonacci
Nombres de Fibonacci : F(0) = 1; F(1) = 1; F(n) = F(n-1) + F(n-2)
Fibonacci(n)
si n=1 alors retourner 1;
si n=2 alors retourner 1;
sinon retourner Fibonacci(n-1)+ Fibonacci(n-2);

approches recursive: (schema)

```
Fib(n)
Si Fib(n) est dans la table retourner table[n];
si n <=2 retourner 1;
sinon table[n]← Fib(n-1) + Fib(n-2); retourner table[n];                    Fib(n)
                    F[0] = 1; F[1] = 1;
                    Pour (i=2 à n) faire
                            F[i] = F[i-1] + F[i-2];
                    Retourner F[n];
```

Coefficients binomiaux

$$\binom{n}{k} = \binom{n-1}{k} + \binom{n-1}{k-1}, \quad 0 > k > n$$

$$\binom{n}{k} = 1, \text{ sinon}$$

```
Binomial (n,k)
si ( k=0 ou k=n ) alors retourner 1;
sinon retourner Binomial(n-1,k-1) + Binomial(n-1,k)
```

Triangle de Pascal Tab

Arbre des appels récursifs

Algorithme de prog. dynamique

```
Binomial (n,k)
Données : n,k : entier;
Initialisation : B[1,0] ← 1; B[1,1] ← 1
pour (i=2 à n) faire
        pour (j=0 à min(i,k)) faire
                si (j=0 ou j=i) alors B[i,j] ← 1;
                sinon B[i,j] ← B[i-1,j-1] + B[i-1,j];
retourner B[n,k];
```

0.4.3 Principe et Processus

Principe d'optimalité
Pour un problème d'optimisation, le principe d'optimalité de Bellmans'applique lorsque la solution optimale peut être obtenue à partir des solutions optimales des sous-problèmes.

Processus de résolution
Le développement d'un algorithme de programmation dynamique peut être planifié dans une séquence de 4 étapes :
1)Caractériser la structure d'une solution optimale;
2)Définir récursivement la valeur d'une solution optimale;
3)Calculer la valeur d'une solution optimale;
4)Construire une solution optimale

0.4.4 Problème du plus court chemin

Problème :
Soit un graphe G=(S,A), S l'ensemble de sommets, et A l'ensemble d'arcs.
Le poids de l'arc a est un entier naturel noté l(a).
La longueur d'un chemin est égale à la somme des longueurs des arcs qui le composent.
Question :
Déterminer pour chaque couple de sommets (si,sj), le plus court chemin, s'il existe, qui joint si à sj

Algorithmes Floyd / Warshall
Etape 1 (Caractérisation de la structure d'une solution optimale):
Si f est un chemin de longueur minimale joignant x à y et qui passe par z, alors il se décompose en deux chemins de longueur minimale. Le premier joint x à z et le second joint z à y
On suppose les sommets numérotés: s1,s2,... sn et pour tout k > 0, on considère la propriété Pk suivante:
Pk(f): Tousles sommetsde f, autresque son origineet son extrémité, ontun indice strictementinférieur à k.

Expression Recusrive
Lemme (Etape 2: Définition récursive d'une solution optimale)
Les relations suivantes sont satisfaites par les :
$\forall$(si,sj), δ(si,sj)=l(si,sj) ou ∞.
$\delta k+1(si,sj)=\min[\delta k(si,sj),\delta k(si,sk)+\delta k(sk,sj)]$

Etape 3 : Calcul d'une solution optimale
Plus court Chemin

```
Données : G un graphe orienté valué;
Initialisation pour k=1;
pour (k=2 à n) faire
        pour (i=1 à n) faire
                pour (j=1 à n) faire
                        delta[i,j]=Min(delta[i,j],delta[i,k]+delta[k,j]
```

Etape 4 : Calcul effectif des chemins optimaux Plus court Chemin

```
Données : G un graphe orienté valué;
Initialisation pour k=1;
pour (k=2 à n) faire
        pour (i=1 à n) faire
                pour (j=1 à n) faire
                        si ( delta[i,j] > delta[i,k] + delta[k,j] ) alors delta[i,j] ← delta[i,k]+delta[k,j] suivant[i,j] ← suivant[i,k]
```

Etape 4 : Calcul effectif des chemins optimaux
Plus court Chemin () Données : suivant[n,n] matrice d'entiers; i,j :entier;
k ← i;

```
tant que (k != j) faire
        écrire (k, '');
        k ← suivant[k,j]
écrire j;
```

0.4.5 Pour résumer

- Nous avons introduit le principe d'optimalité de Bellman:

'toute solution optimale s'appuie elle-même sur des sous-problèmes résolus localement de façon optimale'

- Nous avons décrit un processus en 4étapes pour la conception d'algorithmes de programmation dynamique:
    - Structure d'une solution optimale;
    - Définition récursive d'une solution optimale;
    - Calcul de la valeur de la solution optimale;
    - Calcul de la solution optimale;
- Nous avons mis en oeuvre ces principes pour les problèmes du calcul des nombres de Fibonacci, du calcul des coefficients binomiaux et pour le calcul des chemin minimaux dans un graphe orienté valué.