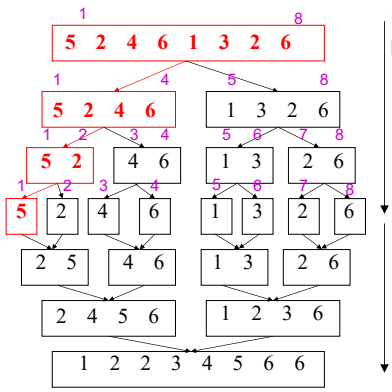
2024-2025, Semestre d'automne
L3, Licence Sciences et Technologies
Université Lyon 1

LIFAPC: Algorithmique, Programmation et Complexité

Chaine Raphaëlle (responsable semestre automne)
E-mail : raphaelle.chaine@liris.cnrs.fr
http://liris.cnrs.fr/membres?idn=rchaine

Retour sur le tri fusion



Profondeur récursion $\log_2(n)$

Empilement des indices des bornes des sous-parties à trier

A chaque profondeur, au moment de la remontée :
• n affectations d'elts (précédées ou non d'une comparaison)

Retour sur le tri fusion

- Remarque : La version récursive du tri fusion étudiée aujourd'hui diffère de celle que vous aviez découverte en LIFAPSD.
- Il s'agissait d'une version où on ne coupe pas la séquence S des éléments à trier de la même manière!
  - La taille de la séquence n'est pas connue à priori
  - Version adaptée au tri des séquences rangées dans des fichiers ou des listes chaînées

- Algorithme itératif qui traite la séquence initiale comme une séquence de monotonies de longueur 1 sans qu'on connaisse leur nombre
- A chaque passage dans la boucle
  - Répartition des monotonies dans 2 autres séquences S1 et S2, à raison d'1 monotonie sur 2 (éclatement)
  - Fusion de la kième monotonie de S1 avec la kième monotonie de S2 et réécriture dans S qui contient ainsi des monotonies de longueur double (sauf peut-être la dernière)


```
procédure TriFusionItératif(
    donnée-résultat S : séquence)
variables S1,S2 : séquence
début
    lgmono←1
    répéter
        Éclatement( S , lgmono , S1 , S2 )
        Fusion( S1 , S2 , lgmono , S , nbmono )
        lgmono=lgmono*2
    tantque nbmono≠1
fin
```

Données

Résultat

- Tri fusion sur un tableau :
  - TriFusionItératif est applicable à un tableau de taille n, moyennant l'utilisation de 2 tableaux supplémentaires de taille max n
  - **TriFusionRec et TriFusionItératif ne sont pas des algorithmes de tri sur place lorsqu'on les applique à des tableaux**. Besoin d'espace supplémentaire pour opérer la fusion
  - Pour **TriFusionRec** il ne faut pas oublier l'espace requis pour gérer la récursion!
- Le tri fusion n'est donc pas l'algo à privilégier pour trier les tableaux

146

146

- Tri fusion itératif sur un fichier
  - Besoin de deux fichiers supplémentaires, pour y recopier les données en les dispatchant, avant de les fusionner dans le fichier de départ.
  - Ce n'est **pas un algo de tri sur place**

147

147

- Tri fusion itératif sur des listes chainées
  - Le dispatch des éléments dans les deux listes supplémentaires peut se faire par rectification de chaînage, sans dupliquer les Celllules ☺
  - La fusion peut également s'opérer par rectification de chaînage
  - **Possibilité d'une implantation sur place du tri fusion sur les listes chainées.**

148

148

Types abstraits (rappels)

- **Type abstrait de données (TAD):**
  - Ensemble des valeurs codées par le type
  - Ensemble des opérations que l'on peut effectuer sur les valeurs et variables de ce type
- Il n'est pas nécessaire de connaître la manière dont les valeurs et les opérations sont codées pour pouvoir les utiliser.
- On utilise les types de façon abstraite, sans connaître leur implantation interne
- Ex : Entier + - * /

149

149

- **Description des Types Abstraits dans le cadre de modules**
- Module :
  Regroupe un ensemble de définitions de **types**, de **procédures** et de **fonctions** qui forment un ensemble cohérent (éventuellement aussi des constantes et des variables globales).
  - **Interface du module (partie visible)**
    Présentation claire des **types**, **procédures**, **fonctions**, constantes et variables offertes par le module
  - **Implantation du module (partie cachée)**
    Mise en œuvre des **types**, **procédures** et **fonctions** proposées dans l'interface. Définition des constantes et variables globales du module.

150

150

Retour sur le type abstrait **séquence (ou liste)**

- Les structures linéaires / séquences permettent de stocker une suite d'élément
- Cette liste peut être amenée à évoluer…
- Il existe 2 sortes de Séquences (ou Listes)
  - Les listes/séquences avec accès récursif :
    - on n'accède pas directement à l'emplacement du ième élément
    - l'accès au i+1 ème élément se fait à partir de l'accès au ième élément
  - Les listes/séquences avec accès itératif
    - Il existe une fonctionnalité d'accès direct au ième élément
    - **Si** les éléments sont en outre garantis être **contigus** on obtient le Type Abstrait Tableau Dynamique

151

151

Interface du TAD Liste Récursive

- **module** ListeRec
  - **importer**
    Module Element, Entier
  - **exporter**
    Type **ListeRec, Emplacement**
    procédure **initialisation**(Résultat s : ListeRec)
    {Préc° : s- non initialisée , Postc° : s+ ListeRec vide}
    procédure **ajoutEltTete**(Donnée-résultat s : ListeRec, Donnée e : Element)
    {Préc° : s- initialisée, Postc° : e inséré en 1ère position}
    pointeur sur Emplacement fonction **emplacementTete** (s : ListeRec)
    {Préc° : s initialisée, Résultat : adresse 1er Emplacement, nul sinon}
    pointeur sur Emplacement fonction **emplacementSuivant** (s : ListeRec, p : pointeur sur Emplacement)
    {Préc° : s initialisée, p adresse d'un Emplacement de s
    Résultat : adresse Emplacement suivant, nul sinon}
    Element fonction **contenuEmplacement** (s : ListeRec, p : pointeur sur Emplacement)
    {Préc° : s initialisée non vide, p adresse d'un Emplacement de s
    Résultat : Element contenu dans l'Emplacement concerné}
    booléen fonction **testerListeVide**(s : ListeRec)
    {Préc° : s initialisée, Résultat : vrai si s vide, faux sinon}
    procédure **suppressionEltTete**(Donnée-résultat s : ListeRec)
    {Préc° : s- initialisée non vide Postc° : le 1er elt a disparu de s}

finmodule

Les Fichiers répondent à cette interface

152

Interface du TAD Séquence Itérative

- **module** SequenceIter
  - **importer**
    Module Element, Entier
  - **exporter**
    Type **Sequencelter**
    procédure **initialisation**(Résultat s : Sequencelter)
    {Préc° : s- non initialisée , Postc° : s+ Sequencelter vide}
    Entier fonction **longueur**(s : Sequencelter)
    {Préc° : s initialisée, Résultat : nombre d'elts présents dans s}
    Element fonction **consulterIèmeElt** (s : Sequencelter, i : Entier)
    {Préc° : s initialisée, 0<i<=longueur(s) , Résultat : ième Element}
    procédure **insertionElt**(Donnée-résultat s : Sequencelter, Donnée e : Element, i : Entier)
    {Préc° : s- initialisée, 0<i<=longueur(s)+1, Postc° : e inséré en ième pos}
    procédure **suppressionElt**(Donnée-résultat s : Sequencelter, Donnée i : Entier)
    {Préc° : s- initialisée, 0<i<=longueur(s), Postc° : ième elt disparu de s}
    pointeur sur Element fonction **rechercheElt**(s : Sequencelter, e :Element)
    {Préc° : s initialisée , Résultat: adresse de e dans s, nul sinon}

finmodule

153

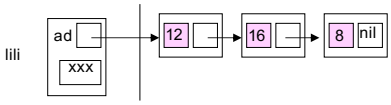
Attention : Ne pas oublier d'ajouter aux modules ListeRec et Sequencelter :

procédure **testament**(Donnée-résultat s : Séquence)
{Préc° : s- initialisée , Postc° : s+ prêt à disparaître}
procédure **initialisation**(Résultat s1 : Séquence, Donnée s2 : Séquence)
{Préc° : s1- initialisée , Postc° : s1+ est une copie de s2}
procédure **affectation**(Donnée-Résultat s1 : Séquence, Donnée s2 : Séquence)
{Préc° : s1- initialisée , Postc° : s1+ est une copie de s2}

154

Implantations possibles

- Utilisation d'une structure chaînée



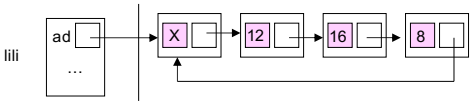
  - Eléments dispersés dans l'espace mémoire
  - Eléments encapsulés dans une structure emplacement (ou Cellule) qui contient également un accès à la place suivante
  - Attention : Cette implantation n'est pas valable si on désire obtenir le TAD Tableau Dynamique

xxx : nombre d'Elements, pointeur sur le dernier emplacement, etc. (optionnel)

155

Implantations possibles

- Variantes :
  - Structure chaînée circulaire (avec éventuellement une cellule bidon)

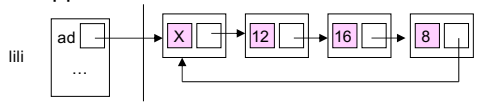


  - On remplace le pointeur à *nil* de la dernière cellule de la liste par un pointeur sur une cellule bidon, laquelle contient un pointeur sur la première cellule
  - Utile pour insérer en queue sans parcourir toute la liste

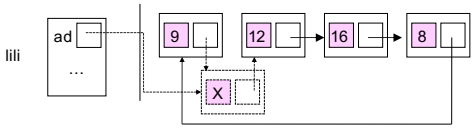
156

Implantations possibles

- Rappel astuce :



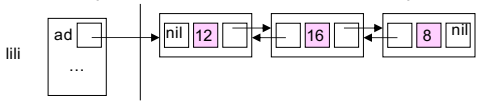
  - Utilisation de la cellule bidon pour y stocker le nouvel Elément
  - Insertion de 9 en queue de la liste <12, 16, 8>



157

Implantations possibles

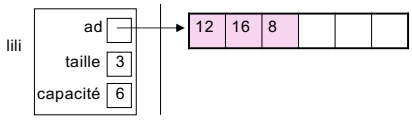
- Variantes :
  - Structure doublement chaînée
    - Chaque emplacement comporte un accès à l'emplacement suivant et un accès au précédent



    - Utile pour les parcours à l'envers

Implantations possibles

- Utilisation d'une représentation contiguë



  - Eléments rangés les uns à la suite des autres dans la mémoire
  - Nécessité de savoir où se finit la séquence (tableau dynamique)
    - Stockage du **nombre d'éléments**
    - Pour éviter les déménagement d'Eléments à chaque insertion on peut prévoir un espace plus grand (**capacité** jusqu'à laquelle il n'y aura pas de réorganisation de la mémoire)

Complexité opérations sur les séquences

- En fonction du nombre n d'Elements

• Quizz	Structure chaînée	Structure chaînée circulaire (avec elt bidon)	Représentation contiguë (Tableau Dynamique)
ajoutEnTete			
ajoutEnQueue			
insertionIèmeElt			
rechercheElt (liste non triée)			
rechercheElt (liste triée)			
insertionElt (liste triée)			

Complexité opérations sur les séquences

	Structure chaînée	Structure chaînée circulaire (avec elt bidon)	Représentation contiguë (Tableau Dynamique)
ajoutEnTete	Θ(1)	Θ(1)	Θ(n) décalage + déménagement parfois
ajoutEnQueue	Θ(n)	Θ(1)	Θ(1) (parfois déménagement Θ(n))
insertionIèmeElt	Θ(i)	Θ(i)	Θ(n-i) (parfois déménagement Θ(n))
rechercheElt (liste non triée)	Θ(n) en moyenne	Θ(n) en moyenne	Θ(n) en moyenne
rechercheElt (liste triée)	**Θ(n) en moyenne**	**Θ(n) en moyenne**	**Θ(lg_2n) en moyenne** Dichotomie
insertionElt (liste triée)	**Θ(n) en moyenne**	**Θ(n) en moyenne**	**Θ(n) en moyenne** Dichotomie+décalage

Rêvons un peu…

- Si on pouvait bénéficier du pouvoir de la recherche dichotomique avec les implantations chaînées
  - On effectuerait les recherches d'éléments et les insertions dans les séquences/listes triées en θ(lg_2n)
  - Les performances seraient donc meilleures qu'avec une implantation contiguë pour laquelle l'insertion dans une séquence/liste triée se fait en θ(n)

Rappel

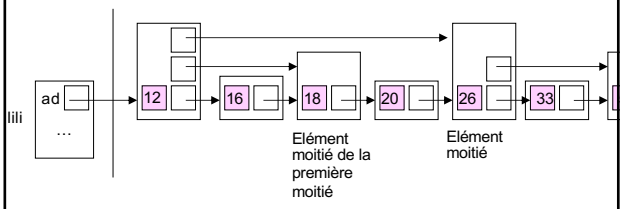
- Recherche dichotomique d'un élément dans une séquence triée avec accès direct :
  - Comparaison de l'élément par rapport à l'élément moitié.
  - La recherche d'un élément dans une séquence de taille n se ramène alors à la recherche dans la première ou seconde moitié
  - **Equation de récurrence :**
    $T_D(n)=T_D(n/2)+T$(accès à l'élément moitié)$+T$(comparaison)
  - Master Theorem, cas 2 (a=1, b=2, $lg_2(1)=0$) :
    - $T_D(n)=lg_2(n)$

Le rêve est-il inaccessible?

- Pour bénéficier du pouvoir de la dichotomie sur les listes chaînées, il faudrait :
  - avoir un accès direct à l'élément milieu,
  - puis un accès direct à l'élément milieu de la première moitié,
  - à l'élement milieu de la seconde moitié,
  - et ainsi de suite...
- Cela peut-être réalisé à l'aide d'ajouts de pointeurs supplémentaires (raccourcis vers l'avant)

164

164



- Exemple d'une Liste de 9 éléments (pas la place de l'écrire en entier sur ce slide!)
- Une telle structure de donnée serait dure à maintenir au fur et à mesure des insertions/retraits d'éléments…

165

165

- Il existe une variante de cette idée basée sur un principe probabiliste : les ***skip-lists***
- Les *skip-lists* entrent dans la catégorie des algorithmes et des structures de données randomisées
- Skip-Lists :
  - Hiérarchie de séquences, chaque séquence correspondant à un échantillonnage de la précédente
  - Probabilité *p* pour un élément d'un niveau, d'appartenir au niveau au dessus
- On reviendra sur cette structuration efficace en TD et en TP ☺

166

166

TAD Ensemble

- Pour un ensemble d'éléments, l'ordre des éléments n'a pas d'importance.
- On veut pouvoir
  - tester l'appartenance d'un élément à un ensemble,
  - ajouter ou supprimer un élément,
  - tester si un ensemble est vide, …

Le plus efficacement possible!

167

167

Interface du TAD Ensemble

- **module** Ensemble
  - **importer**
    Module Element
  - **Exporter**
    Type **Ensemble**
    procédure **initialisation**(Résultat f : Ensemble)
    {Préc° : f- non initialisé , Postc° : f+ Ensemble vide}
    procédure **ajouterElt**(Donnée-résultat f : Ensemble, Donnée e : Element)
    {Préc° : f- initialisé, Postc° : e présent dans f+}
    procédure **supprimerElt**(Donnée-résultat f : Ensemble, Donnée e : Element)
    {Préc° : f- initialisé, Postc° : e non présent dans f+}
    booléen fonction **rechercherElt** (f : Ensemble, e : Element)
    {Préc° : f- et e initialisé, Résultat : vrai si e dans f, faux sinon}
    entier fonction **nombreElement**(f : Ensemble)
    {Préc° : f initialisé, Résultat : nbre d'éléments contenus dans f}
    Éventu-ellement
    Ensemble fonction **union**(f1 : Ensemble, f2 : Ensemble)
    {Préc° : f1 et f2 initialisés, Résultat : elts appartenant à f1 ou f2}
    Ensemble fonction **intersection**(f1 : Ensemble, f2 : Ensemble)
    {Préc° : f1 et f2 initialisés, Résultat : elts appartenant à f1 et f2}
    procédure **testament**(Donnée-résultat f : Ensemble)
- **Finmodule**

168

168

Implantations possibles

- Représentation par des tableaux de booléens
  - **Lorsque l'univers des valeurs possibles des Eléments sont en nombre fini et raisonnable**, et peuvent permettre d'indicer un tableau,
  - Utilisation d' **un tableau**
    - contenant vrai dans chaque case correspondant à un Elément de l'Ensemble,
    - faux sinon
  - Quelle est la complexité des opérations sur les ensembles avec cette implantation?

169

169

Implantations possibles

- Représentation par des tableaux de booléens
  - **Lorsque l'univers des valeurs possibles des Eléments sont en nombre fini et raisonnable**, et peuvent permettre d'indicer un tableau,
  - Utilisation d' **un tableau**
    - contenant vrai dans chaque case correspondant à un Elément de l'Ensemble,
    - faux sinon
  - Quelle est la complexité des opérations sur les ensembles avec cette implantation?
    - TEMPS CONSTANT POUR L'AJOUT, LA RECHERCHE ET LA SUPPRESSION

170

170

Autres implantations possibles

- Représentation d'un Ensemble par une Séquence/Liste de ses Eléments
- Si le type des Eléments bénéficie d'une relation d'ordre total,
  - on peut utiliser une Séquence/Liste triée (par exemple implantée avec une *skip-list*)
  - ou un Arbre Binaire de Recherche

171

171

Notion de clé

- En général, quand on stocke et recherche des éléments complexes, les opérations de comparaison ne sont pas effectuées directement sur les éléments, mais sur une clé unique qui leur est associée.
  - Exemple : numéro de sécurité sociale d'une personne.
- Avantages :
  - Il est souvent plus rapide de comparer 2 éléments sur la base de leurs clés, que sur la base des informations qui leur sont associées.
  - Il est possible que le type Elément ne bénéficie pas d'une relation d'ordre total, mais que ce soit le cas pour le type Clé
  - On peut alors bénéficier de structurations performantes

Un élément = Une clé et des informations associées

172

172

TAD Table

- Table : Ensemble de clés (auxquelles on peut associer une valeur)
- Vocabulaire :
  - Si la clé c est présente dans une table t
  - On dit qu'il existe une entrée dans la table t pour la clé c
- Chaque clé est (généralement) unique
  - Information identifiante et discréminante permettant de distinguer les entrées de la table

173

173

- Les opérations sur les Tables sont similaires à celles sur les Ensembles
  - Recherche, insertion et suppression de Clés/Eléments
- Opérations supplémentaires
  - Fournir l'information associée à une clé présente dans la table
  - Modifier l'information associée à une clé présente dans la table
  - La suppression de l'entrée correspondant à une clé doit s'accompagner de la suppression de l'information associée

174

174