Cours

Technique d'algorithmie :

- Algorithme Glouton
- Diviser pour régner, rencontre au millieu, dichotomie
- Programmation dynamique : Calcul de bas en haut par mémoisatioon, reconstruction de solution.
- Recherche Brute-force, Retour sur trace

Algorithme du Cours :

- Recherche Dichotomique dans un tableau trié
- Tri fusion
- Tri insertion/tri selection (?? c'est au programme meme??)
- Tri par tas
- Tri rapide (????)
- Algorithme de Boyer-Moore (recherche textuelle)
- Algorithme de Rabin-Karp (recherche textuelle)
- Algorithme de Huffman (compression)
- Algorithme de Lempel-Ziv-Welch (compression)
- Parcours en largeur, profondeur dans un graphe
- Kosaraju pour les CFC
- Floyd-Warshall
- Dijsktra

Exercices cité dans le programme :

- Huffman, sélection d'activité, ordonnancement de taches unitaires
- recherche dichotomique dans un tableau trié
- tri fusion
- comptage du nombre d'inversions dans une liste
- calcul des deux points les plus proches dans une ensemble de points
- recherche d'un sous-ensemble d'un ensemble d'entiers dont la somme des éléments est donnée
- problème de la somme d'un sous-ensemble
- ordonnancement de tâches pondérées
- plus longue sous-suite commune
- distance d'édition

Merge de liste

Soit L_1, L_2 deux listes triées. Proposer un algoritme en OCaml qui calcule la liste trié des éléments contenue dans L_1 et L_2.

Nombre d'inversions dans une liste

Soit L une liste de longueur n, on dit que (i, j) pour $0 \leq i < j < n$ est une inversion si $L[i] > L[j]$. On cherche à compter le nombre d'inversion.

En vous inspirant du tri fusion, proposer une approche dichotomique en $O(n \log n)$.

Points les plus proches

On pose $P = \{p_1, ..., p_n\} \subseteq \mathbb{R}^2$ un ensemble de points du plan. On écrit pour tout $1 \leq i \leq n$ $p_i = (x_i, y_i)$. On cherche deux points p_i, p_j distincts qui minimise

$$\|p_i - p_j\|_2 = \sqrt{\left(x_i - x_j\right)^2 + \left(y_i - y_j\right)^2}$$

Montrer qu'il existe un algorithme en $O(n \log n)$.

Indication: Trier P selon les $(x_i)_i$, puis couper P en deux pour s'apeller récursivement sur deux sous-ensemble de points. Quel cas manquent-t'ils? Pour ces cas-làs, considérer un tri selon les $(y_i)_i$

Sous-tableau connexe

Soit A un tableau de n entiers relatifs, on cherche un algorithme en $O(n)$ qui calcule le sous-tableau connexe qui maximise la somme de ses éléments, i.e. le couple (i, j) avec $0 \leq i \leq j \leq |T|$ tel que $\sum_{k \in [\![i;j]\!]} A[k]$ soit maximal.

Le glouton par défault[1]

Étant donné un ensemble $\{x_1, ..., x_n\}$ de n points sur une droite, décrire un algorithme qui détermine le plus petit ensemble d'intervalles fermés de longueur 1 qui contient tous les points donnés.

Prouver la correction de votre algorithme et donner sa complexité.

[1] Algo 1 ENS Lyon

SUBSET-SUM

Soit $\{x_1, ..., x_n\} \subseteq \mathbb{N}$ un sous-ensemble d'entiers et $S \in \mathbb{N}$. On cherche à savoir s'il existe un $I \subseteq [\![1; n]\!]$ tel que

$$\sum_{i \in I} x_i = S$$

Donner un algorithme de programmation dynamique en $O(nS)$ qui répond au problème. On pourra considérer le tableau T de booléens de longeur S tel que $T[k] = \text{true}$ si et seulement si il existe $I \subseteq [\![1; n]\!]$ dont $\sum_{i \in I} x_i = k$.

Elements Majoritaire

Soit T un tableau d'entier de longeur N, on dit qu'un entier K est *majoritaire dans* T s'il est présent strictement plus que $\lfloor \frac{N}{2} \rfloor$ fois.

1. Donner un algorithme en C `bool is_majo(int* t, int l);` qui teste si un élément est majoritaire ou non.

On note $\text{head}(P)$ l'élement en tete d'une pile P, et on considère l'algorithme suivant:

$P \longleftarrow$ pile vide
Pour i **allant de** 1 à N:
    si P est vide **alors:**
        Empiler $T[i]$ à P
    sinon si $\text{head}(P) = T[i]$ **alors**:
        Empiler $T[i]$ à P
    sinon:
        Dépiler P
    Fin si
Fin Pour

2. Montrer que à toute étape elle contient que les meme valeurs. Elle peut donc etre implémentée par un couple $(\text{valeur de tete}, \text{compteur})$. Donner un code en C qui execute cet algorithme avec cette optimisation.
3. Montrer que si il y a un élément majoritaire dans T, alors à la fin de l'algorithme P est non vide et $\text{head}(P)$ est majoritaire.
4. En déduire un algorithme linéaire qui teste si un élément majoritaire est dans P

On dit qu'un élément est C-majoritaire s'il est présent strictement plus de $\lfloor \frac{N}{C} \rfloor$ fois.

5. Pour $C \in \mathbb{N}$ est fixé, donner un algorithme linéaire pour calculer les (au plus C) éléments majoritaires de T.
6. Quelle est la complexité en fonction de C et N ?

Disco-world[2]

A DiscoWorld, quand tu as une réduction, elle s'applique à tout... meme aux réductions ! Alice, fane de réductions, veut toutes les acheter, et ce demande quel est la quantité minimale d'argent qu'elle devra dépenser pour obtenir toutes les réductions.

Version 1 On pose $R = \{(c_1, v_1), ..., (c_n, v_n)\}$ un ensemble de réductions, tel que c_i soit le cout de la i-ème réduction et v_i soit la valeure qui sera soustraite à tout les achats suivants (pour un cout pottentiellement négatif). On ne peut pas acheter 2 fois la meme réduction.

Par example, pour $R = \{(5, 4), (10, 4), (5, 2)\}$, Alice n'aura besoin de que 8€ pour tout acheter : elle achete d'abord la réduction 1 pour 5€, puis la réduction 2 pour 6€, puis elle achete la réduction 3 pour... -3€. Elle aura donc au total seulement utilliser $5 + 6 - 3 = 8$€. La réponse est donc 8€ dans ce cas.

Donner un algorithme calculant le prix/gain que Alice devrais dépenser/obtiendra en achetant toutes les réduction. On pourra faire par programmation dynamique.

Version 2 DiscoWorld se rendant compte qu'il commençait à perdre de l'argent, on décider maintenant que une fois une réduction (c_i, v_i) achetée, elle **diviserai** le prix de toutes les futures réductions par v_i. On ne peut toujours que acheter les réduction qu'en un seul exemplaire.

Donner un algorithme polynomial calculant la quantité d'argent que Alice devrais dépenser pour acheter toutes les réductions.

Indication: Comparer l'achat de la réduction i puis j avec l'achat de j puis i. En déduire qu'il suffit de trier les réductions selon une certaine formule.

[2]Cours de programmation compétitive ENS Lyon

Recherche dans une matrice

Question 1 Donner le code C d'une fonction `int find(int *arr, int n, int key);` qui prend en argument un tableau `arr` trié de longueur `n` et qui trouve un indice `i` tel que `arr[i] == key`.

On cherche à faire un algorithme efficace aussi dans le cadre de matrice trié. On dit qu'une matrice M de taille $n \times m$ est *linéairement trié* si chaque ligne est trié de haut en bas et que chaque colone est trié de gauche à droite.

Question 2 Donner le code d'un fonction C `bool is_ordered(int **mat, int n, int m)` qui à une matrice `mat` de taille $n \times m$ teste si elle est linéairement trié.

Question 3 Donner le code d'une fonction `is_in(int **mat, int n, int m, int key)` qui teste si `key` est présent dans la matrice `mat` linéairement trié de taille $n \times m$ par diviser pour reigner.

Question 4 Meme question en $O(n + m)$

Question 5 Donner le code d'une fonction C qui en $O(n)$ trouve un minimum local dans une matrice $n \times n$

Multiplication rapide de polynome[3]

Ici on considère des polynômes d'entiers $\mathbb{Z}[X]$. Soient $P, Q \in \mathbb{Z}_n[X]$, leur produit $R = PQ \in \mathbb{Z}_{2d}[X]$

On représente un polynome par un tableau d'entiers T, tel que pour $P \in \mathbb{Z}_d[X]$, si on écrit $P = \sum_{i \leq d} a_i X^i$ on a $T[i] = a_i$.

1. Donner une fonction C `int* multiply(int* p, int* q, int n)` qui prend deux polynômes de degré $\leq n$ représenter par deux tableaux de longueur n et qui renvoie le polynome produit.
2. Soit $n = 2m$, pour $P \in \mathbb{Z}_n[X]$, montrer que on peut décomposer $P = P_1 + X^m P_2$ avec $P_1, P_2 \in \mathbb{Z}_m[X]$.

Soient $P, Q \in \mathbb{Z}_n[X]$, on décompose $P = P_1 + X^m P_2$ et $Q = P_1 + X^m P_2$. On définit alors $R_1 = P_1 Q_1$, $R_2 = P_2 Q_2$ et $R_3 = (P_1 + P_2) \times (Q_1 + Q_2)$.

3. Exprimer $P \times Q$ en fonctions de R_1, R_2, R_3.
4. En déduire un algorithme récursif pour calculer le produit de polynome.
5. Quelle est la complexité de cet algorithme ?

[3] Algo 1 ENS Lyon

Tableaux auto-référents

Pour T un tableau de longueur n, on définit l'histogramme de T noté H_T comme le tableau de longueur n tel que $H_T[i]$ correspond au nombre de i dans T. Formellement, $H_T[i] = |\{j : T[j] = i\}|$

On dit que T est auto-référent si $H_T = T$. Par exemple, $[1, 2, 1, 0]$ est un tableau autoréférent.

Question 1 Donner le code d'une fonction `int* histogramme(int *t, int n);` qui, à un tableau t de longueur n renvoie H_t son histogramme.

Question 2 Donner tous les tableaux auto-référent de longueur 4 et 5

Question 3 Donner le code d'une fonction `bool is_autoref(int *t, int n);` qui teste si un tableau t est auto-référent.

Question 4 Donner le code d'une fonction de recherche exhaustive `int count_autoref(int n);` qui donne le nombre de tableaux auto-référents de longueur n.

Question 5 Montrer que pour tout $n > 6$ il existe un tableau autoréférent de longueur n

Question 6 $(*)$ Montrer qu'il est unique