Structures complexes

Exemples

- Sous-ensembles d'un ensemble
- Arbres (binaires ou autres...)
- Graphes
- Permutations

Si un programme construit et utilise une seule structure, des méthodes *ad hoc* suffisent;
si le programme en utilise beaucoup, il faut penser à l'efficacité des opérations nécessaires

Opérations utiles

- Boucler sur toutes les structures
- Stocker une structure dans l'espace minimum possible ($code$)
- Récupérer une structure stockée ($décode$)
- Compter les structures
- Stocker des informations sur toutes dans un tableau

souvent après avoir choisi une taille de structure (disons n); sinon la boucle ou le tableau serait impossible

Sous-ensembles et permutations

Pas de gros problème

- Sous-ensemble de n objets $\approx$ entier en $0 \cdots 2^n - 1$
- Permutation? considérer une permutation comme n entiers en $0 \cdots n-1$ donnerait une représentation par un entier en $0 \cdots n^n - 1$
- Mais tous ces entiers ne correspondent pas à une permutation (n^n entiers mais $n!$ permutations)
- Interpréter suite $a_1,\ a_2,\ \cdots,\ a_n$ avec $1 <= a_i <= i$: échanger les éléments n et a_n, ensuite $n-1$ et a_{n-1} et ainsi de suite; chaque permutation est générée une seule fois

- Facile à générer la suite des a à partir d'un entier en $0 \cdots n! - 1$

- (Remarquer l'existence d'algorithmes plus astucieux qui génèrent toutes les permutations avec une seule transposition à chaque itération)

Générer tous les arbres binaires (sans valeurs aux sommets)

(de taille n)

- Pour toute taille i possible du sous-arbre gauche $(0 \cdots n-1)$

- générer tous les sous-arbres gauches (taille i)i

- générer tous les sous-arbres droits (taille $n-1-i$)

- mais complication de programmation; il faut chaque combinaison possible d'un sous-arbre gauche et droit; donc, on pourrait stocker deux listes des sous-arbres gauches et droites et les croiser; mais solution trop gourmande en espace

- une solution possible: procédures/fonctions `init` et `prochain`
  `init` donne le premier arbre (d'une taille donnée) et `prochain(T)` donne le successeur de `T` dans la liste
  définies récursivement!

  par exemple: $prochain(T) =$
  $(T{\cdot}gauche,\ prochain(T{\cdot}droite)$ si le $prochain$ n'échoue pas
  ou $prochain(T{\cdot}gauche),\ init(taille(T{\cdot}droite)))$
  si ...
  ou $init(taille(T{\cdot}gauche)+1),\ init(taille(T{\cdot}droite)-1))$ si $taille(T{\cdot}droite) > 0$
  ou échoue
  et $init(n)$ est une chaîne à droite.

Compter les arbres (binaires)

Dans ce cas il existe une formule simple mais...

- Soit $T(n)$ le nombre d'arbres de taille n
- Considérer le nombre avec sous-arbre gauche de taille i
- $T(n) = \sum_{i=0}^{n-1} T(i) * T(n-1-i)$
- Programmation dynamique!
- $T(0) = 1$

Coder un arbre

(de taille n, entier en $0 \cdots T(n)-1$)

- Ordonner les arbres selon la taille de leur sous-arbre gauche, et ensuite selon le code de ce sous-arbre et enfin le code du sous-arbre droite

- $code(A) = \sum_{i=0}^{taille(A.gauche)-1} T(i) * T(n-1-i) + code(A.gauche) * T(A.droite) + code(A.droite)$

- $code(arbre.vide) = 0$

Décoder

Etant donné le code c d'un arbre de taille n, trouver

- La taille du sous-arbre gauche (min i tel que la somme est au moins c)

- Le code du sous-arbre gauche $((c - la\ somme)/T(A.droite))$

- Le code du sous-arbre droit (et le reste de la division)

- Et continuer récursivement

En général

Un algorithme de génération donne des algorithmes de codage et décodage

- Ordonner les objets selon les choix faits par l'algorithme

- considérer le nombre d'objets susceptibles d'être générés par chaque choix
  il faut que ce nombre soit calculable efficacement

- ajouter pour coder

- soustraire pour décoder

Des cas plus compliqués

Par exemple de vrais arbres (selon les matheux, pas les arbres binaires des informaticiens)

- Arbre = vide ou racine + ensemble de sous-arbres

- ensemble, donc sans ordre

- pour simplifier, je considère les cas de degré max 2

- Alors, un arbre peut être representé par un arbre binaire

- Mais, de plusieurs façons! (à cause de l'ordre des sous-arbres)

Représentations canoniques

- Quand une structure a plusieurs représentations, on en choisit arbitrairement une comme la "vraie" ou *canonique*

- Par exemple ordonner les arbres (binaires) par taille, sous-arbre gauche, sous-arbre droit

- et dire que la rep. canonique d'un arbre est le premier arbre binaire qui en est une représentation

- Fonction de décider si une rep. est canonique

- Récurrence pour le nombre d'arbres de taille n ($n > 1$): $G(n) = G(n - 1)$

$+ \sum_{i=1}^{n/2-1} G(i) * G(n-1-i) +$
$(si\ n\ impair) G((n-1)/2) + \begin{pmatrix} G((n-1)/2) \\ 2 \end{pmatrix}$
(à noter: arrondir vers le bas dans la division $n/2$)
(les quatre parties sont: un seul sous-arbre (forcément de taille $n-1$, deux sous-arbres de taille différente, le plus petit de taille i, deux sous-arbres identiques, deux sous-arbres différents de la même taille)

- Mêmes principes pour codage et décodage

Des Problèmes Difficiles

Les problèmes de codage et décodage sont parfois très compliqués:
exemple:

- graphes sur n sommets de degré maximum (par exemple) 3

- On peut les générer en considérant chaque sommet à son tour et ajoutant un nombre d'arêtes jusqu'à 3 $-$ degré actuel

- mais contraintes: pas d'arête à un sommet qui a déjà degré 3

- donc, combien de façons de compléter?

Graphes et isomorphisme!

Souvent, on considère deux structures comme identiques si on peut tranformer l'une dans l'autre par une permutation des composants

- donc, on veut générer (ou compter) un representant canonique de chaque classe d'équivalence

- mais les classes ne sont pas toutes de la même taille!

- difficile même de savoir si une structure est canonique

- ou de décider si deux structures sont isomorphes (Pb NP-complet)