TD d'algorithmique numéro 6
Daniel Hirschkoff et Natacha Portier
semaine du 8 au 12 novembre 1999

Question 1 ⋆ du rab sur le coloriage ⋆

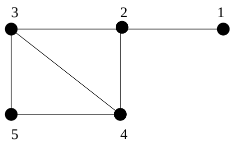
L'objet de cet exercice est de démontrer le résultat surprenant suivant : le nombre moyen de K-coloriages d'un graphe de n sommets est borné par une constante indépendante de n. Commençons par quelques résultats préliminaires.

Question 1.1 Soient K nombres positifs $s_1, s_2, \ldots, s_K$ dont la somme est L. Montrer que la somme de leurs carrés vaut au moins $\frac{L^2}{K}$.

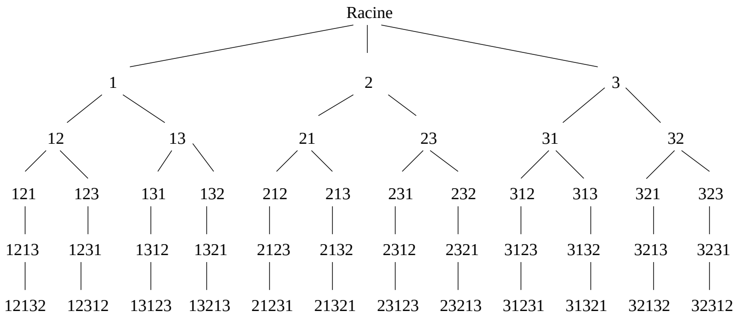
Question 1.2 Soit C une des K^L façons de colorier avec K couleurs un ensemble de L sommets virtuels. Montrer que le nombre de graphes G dont l'ensemble des sommets est cet ensemble et pour lesquels C est un coloriage correct est au plus $2^{\frac{L^2}{2}(1-\frac{1}{K})}$.

Question 1.3 Montrer que le nombre total de coloriages corrects avec K couleurs de tous les graphes de L sommets est au plus $K^L \times 2^{\frac{L^2}{2}(1-\frac{1}{K})}$.

On introduit maintenant la recherche "backtracking" pour le coloriage des graphes. On construit l'arbre de toutes les possibilités de coloriage, où l'étiquette de chaque noeud décrit les couleurs déjà affectées aux premiers sommets. Voici un exemple de graphe :



et son arbre "backtracking"de coloriage avec $K = 3$ couleurs :



Dans l'arbre, l'étiquette 212 signifie que les trois sommets 1, 2 et 3 ont été respectivement coloriés en 2, 1 et 2. Le nom de "backtracking" vient de l'algorithme de parcours : on descend tant qu'on peut dans l'arbre, et après la fin d'une exploration (couronnée de succès ou terminant par un échec) on remonte l'arbre le moins possible pour poursuivre l'exploration.

Le niveau L de l'arbre de coloriage avec K couleurs d'un graphe G (la racine est au niveau 0) représente tous les coloriages corrects en K couleurs du sous-graphe $H_L(G)$ induit par les L premiers sommets de G. Le nombre de noeuds à ce niveau est donc égal au nombre de coloriages corrects de $H_L(G)$ avec K couleurs : on note ce nombre $P(K, H_L(G))$ (P représente le polynôme chromatique, mais c'est une autre histoire).

Question 1.4 Montrer que le nombre moyen $A(n, K)$ de noeuds dans un arbre de coloriage "backtracking" pour colorier les graphes de n sommets avec K couleurs est borné par une constante $h(K)$ indépendante de n.

Question 2 ⋆ B-arbres ⋆

Encore une structure de données, encore une famille d'arbres : les B-arbres sont utilisés pour stocker des données sur des supportts tels que les disques magnétiques, dont les temps d'accès sont considérablement plus lents que pour la mémoire à accès aléatoire. Dans ce contexte, un facteur déterminant pour la performance est le nombre d'accès disque effectués, qui augmente avec la hauteur de l'arbre (on essaye donc de limiter cette dernière).

Un *B-arbre* T est caractérisé par les données suivantes :

- sa racine $racine[T]$;
- pour chaque nœud x :
  - $n[x]$: nombre de clés du nœud (les B-arbres ne sont pas binaires)
  - $n[x]$ clés $k_1[x] \leq k_2[x] \leq \cdots \leq k_{n[x]}[x]$
  - un booléen $feuille[x]$ qui indique si x est une feuille
  - si x n'est pas une feuille, $n[x] + 1$ pointeurs $c_1[x] \dots c_{n[x]+1}[x]$ vers des sous-arbres, construits selon la définition des arbres binaires de recherche (les $n[x]$ clés définissent $n[x] + 1$ intervalles, et quand on descend dans un fils on n'y trouve que des clés qui sont dans l'intervalle correspondant).
- toutes les feuilles ont la même profondeur $h[T]$
- à chaque B-arbre est associé un entier t appelé son *degré minimum* (souvent fixé en fonction des caractéristiques du support de stockage utilisé). À t sont associées les deux contraintes suivantes :
  - Tout nœud autre que la racine contient au moins $t - 1$ clés (et donc s'il n'est pas feuille, il a au moins t fils). Si l'arbre n'est pas vide, la racine doit posséder au moins une clé.
  - Tout nœud peut contenir au plus $2t - 1$ clés.

  Remarque : un B-arbre avec $t = 2$ est appelé arbre 2-3-4 (et est apparu avant les B-arbres dans la littérature).

Pour l'étude de la complexité des algorithmes sur les B-arbres, on sera amené à raisonner en distinguant

- le nombre d'accès au disque, et
- le temps (de calcul) CPU.

En effet, on travaille avec la convention selon laquelle un nœud de l'arbre a la taille d'une *page mémoire* pour l'architecture avec laquelle on travaille. Ainsi, le traitement d'une page se fera typiquement selon le scénario suivant :

...
$x \leftarrow$ un pointeur sur un objet de la mémoire
`lire_disque`(x) *charger la page en mémoire principale*
blablabla(x) *opérations sur x*
`écrire_disque`(x) *on recopie la page en mémoire*

On s'attachera donc à mentionner explicitement les points où sont faits des accès mémoire (coûteux), et on calculera le nombre d'accès mémoire nécessités par un algoithme avant d'en évaluer le temps CPU.

Question 2.1 Calculer la hauteur au pire des cas d'un B-arbre en fonction du nombre de clés n qu'il contient et de t.

Question 2.2 Écrire l'algorithme de recherche et en calculer la complexité.

Pour créer un B-arbre vide, on lance la fonction suivante (qui est en $O(1)$) :

Créer_B-arbre (T)
x := `Allouer-noeud()`
$feuille[x]$:= vrai
$n[x]$:= 0
`Écrire-disque`(x)
$racine[T]$:= x
...

Question 2.3 Pour définir l'algorithme d'insertion, on commence par définir le découpage, qui, à partir d'un nœud complet ($2t-1$ clés), fait remonter la clef médiane et produit deux nœuds à $t-1$ clés (on remarque donc que le père, qui “reçoit” la clé qui remonte, ne doit pas être complet : c'est une condition que l'on va assurer lors des appels à cette fonction de découpage).

Écrire la fonction `découper_fils`(x, i, y), où x est un nœud incomplet, et $y = c_i[x]$ est un nœud complet (fils de x).

Écrire l'algorithme d'insertion, qui, en progressant, commence par couper en deux un nœud complet avant de descendre dans l'arbre. Évaluer la complexité.

Question 2.4 Suppression : à défaut d'écrire l'algorithme en entier, indiquer les différents cas de figure qui peuvent se présenter.