INFO0902-1 - Structure de données et algorithmes

Matière d'examen

Pierre Geurts

Année académique 2017-2018

Prélude

Pour chaque algorithme mentionné au début de chaque section, vous devez être capable :

- de décrire l'algorithme en utilisant le formalisme de votre choix (pseudo-code, langage c ou langage naturel, pour autant que vous soyez précis). Exemple :
  - Décrivez l'algorithme de tri par insertion dans le formalisme de votre choix
- d'illustrer l'exécution de l'algorithme sur une instance particulière. Exemple :
  - Illustrez les différentes étapes du tri par fusion sur le tableau $[31, 41, 59, 26, 41, 58]$.
  - Montrez l'arbre binaire de recherche obtenu par l'insertion successive des clés $[3, 7, 4, 20, 15, 5]$.
- de restituer sa complexité (en temps et en espace) le plus précisément possible et de pouvoir donner des exemples correspondants aux meilleur et pire cas. Exemple :
  - Caractérisez de manière aussi précise que possible la complexité du tri rapide
  - Donnez un exemple d'arbre binaire de recherche correspondant au pire cas pour la recherche d'une clé
- de le comparer aux autres algorithmes vus au cours pour résoudre le même problème. Exemple :
  - Quel algorithme de tri est le plus approprié dans le cas où on désire minimiser la complexité en espace ? Justifiez.

Sauf dans quelques cas mentionnés dans la liste ci-dessous, on ne vous demandera pas :

- de prouver formellement la correction de ces algorithmes
- de prouver formellement les formules de complexité (mais bien de les expliquez)

Partie 1 : introduction et récursivité

Algorithmes : tri par insertion, tri par fusion

Questions :

1. Qu'est-ce qu'un algorithme ? Comment peut-on le décrire ?
2. Qu'est-ce qu'un algorithme correct, partiellement correct, approximatif ?

3. Qu'est-ce qu'une structure de données ? Un type de données abstrait ? Donnez un ou plusieurs exemples.
4. Qu'est-ce qu'un algorithme récursif ? La récursivité multiple ? La récursivité terminale ? Donnez à chaque fois un exemple.
5. Représentez l'arbre des appels récursifs d'un algorithme donné.

Partie 2 : outils d'analyse

Questions :

1. Comment vérifie-t-on qu'un algorithme itératif/récursif est correct ?
2. Montrez qu'un algorithme donné itératif/récursif est correct (pour des algorithmes simples, du type de ceux vus aux répétitions)
3. Expliquez le principe d'une preuve par induction. Montrez par induction qu'une propriété simple donnée est correcte.
4. Qu'entend-on par complexité en temps et complexité en espace ?
5. Que signifient les notations $O(.)$, $\Theta(.)$, $\Omega(.)$?
6. Qu'est-ce que la complexité au pire cas, au meilleur cas, en moyenne ? Quel cas est le plus pertinent en pratique ?
7. Calculez et justifiez la complexité d'un algorithme donné (itératif ou récursif).
8. Un algorithme de complexité X est-il systématiquement meilleur/moins bon qu'un algorithme de complexité Y ? Justifiez.
9. Montrez que le problème de tri est $\Omega(n)$. Montrez qu'il est aussi $O(n^2)$.

Partie 3 : tri

Algorithmes : tri rapide, construction d'un tas (Build-Max-Heap), tri par tas

Questions :

1. Qu'est-ce que le problème de tri ? Qu'est-ce qu'un algorithme de tri en place/itératif/récursif/stable/comparatif ? Donnez à chaque fois un exemple.
2. Qu'est-ce qu'un arbre ? Qu'est-ce qu'un arbre binaire/ordonné/binaire entier/binaire parfait/binaire complet ? Qu'est-ce que la hauteur d'un arbre ?
3. Etablissez en utilisant les notations asymptotiques le lien qui existe entre la hauteur d'un arbre binaire (entier ou non) et le nombre de nœuds. Justifiez.
4. Qu'est-ce qu'un tas binaire ? Comment implémente-t-on un tas dans un vecteur ?
5. Construisez l'arbre de décision correspondant à un algorithme de tri X sur un tableau de taille Y.
6. Montrez que le problème de tri (comparatif) est $\Omega(n \log n)$.

Partie 4 : structures de données élémentaires

Algorithmes : insertion dans un tas (HEAP-INSERT), parcours d'arbres (4 algorithmes)

Questions :

1. Pour les structures suivantes :
   — Pile
   — File simple
   — File double
   — Liste
   — Vecteur
   — Arbre
   — File à priorité
   vous devez être capable :
   — d'expliquer le principe de la structure et les opérations standard de l'interface.
   — de décrire une manière de l'implémenter et de préciser la complexité des différentes opérations.
2. Montrez que les parcours d'arbres infixe, préfixe et postfixe sont $\Theta(n)$.
3. Montrez que le coût amorti d'une opération d'insertion à la fin d'un vecteur extensible dont on double la taille est $O(1)$ et qu'elle serait $\Theta(n)$ si on augmentait cette taille d'une constante c.
4. Etant donné un problème algorithmique, pouvoir déterminer quelle structure est la plus appropriée.

Partie 5 : dictionnaires

Algorithmes : recherche dichotomique, recherche, successeur, insertion et suppression dans un abre binaire de recherche.

Questions :

1. Qu'est-ce qu'un dictionnaire (principe, interface, exemples d'application) ? Enumérez au moins 4 manières de l'implémenter en précisant la complexité des opérations principales.
2. Comment peut-on implémenter un dictionnaire à l'aide d'un vecteur ?
3. Qu'est-ce qu'un arbre binaire de recherche ?
4. Comment peut-on trier avec un arbre binaire de recherche ? Comparez cet algorithme aux autres algorithmes de tri vu au cours.
5. Qu'est-ce qu'une table de hachage ? Décrivez les opérations d'insertion et de suppression et donnez leur complexité.
6. Qu'est-ce qu'une collision ? Qu'est-ce que le facteur de charge ? Expliquez le principe des deux méthodes principales de gestion des collisions, donnez leurs complexités (sans preuve) et comparez les.
7. Qu'est-ce que l'adressage ouvert ? Décrivez les différentes fonctions de sondages.

8. Qu'est-ce qu'une fonction de hachage ? Quelles sont les caractéristiques d'une bonne fonction de hachage ? Comment traiter des clés non numériques ?
9. Décrivez deux exemples de familles de fonctions de hachage
10. Comparez les arbres binaires de recherche et les tables de hachage en énumérant leurs avantages et défauts respectifs.

Partie 6 : résolution de problèmes

Questions :

1. Pour chaque technique de programmation (force brute, diviser-pour-régner, programmation dynamique et algorithme glouton) :
   — Expliquez le principe de la technique.
   — Donnez un exemple de problème et sa solution utilisant cette technique.
2. Etant donné un algorithme, déterminez quelle technique est utilisée.
3. Résolvez un problème donné (nouveau) en utilisant la technique X et donnez la complexité de votre solution.
4. Etant donné une équation récursive, dessinez le graphe des sous-problèmes et implémentez cette équation de manière efficace en utilisant l'approche ascendante/l'approche descendante avec mémoization.
5. Quelles sont les deux conditions nécessaires pour pouvoir appliquer la programmation dynamique ? Illustrez avec un exemple.
6. Comment prouver qu'une approche gloutonne est correcte ? Illustrez en montrant que l'algorithme CoinChangingGreedy est optimal pour les pièces $1, 2, 5, 10$ et 20.
7. Comparez la programmation dynamique au diviser-pour-régner et à l'approche gloutonne.

Partie 7 : graphes

Algorithmes : parcours en largeur, parcours en profondeur, Bellman-Ford, Dijkstra.

Questions :

1. Qu'est-ce qu'un graphe ? un graphe dirigé/non dirigé ? acyclique ? connexe ? une composante connexe d'un graphe ? le degré d'un graphe ? un graphe pondéré ?
2. Quelles sont les deux manières principales de représenter un graphe ? Donnez les complexités en fonction de $|V|$ et $|E|$ des principales opérations dans les deux cas. Comparez les deux représentations.
3. Comment modifier les algorithmes de parcours pour parcourir tous les sommets d'un graphe ? Appliquez cette idée au parcours en largeur/ en profondeur
4. Qu'est-ce qu'un chemin ? Le poids d'un chemin ? Le plus court chemin entre deux sommets ?
5. Comment gérer les cycles dans le cadre de la recherche d'un plus court chemin ?
6. Expliquez le problème de recherche du plus court chemin à origine unique, présentez le schéma général d'un algorithme et expliquez le principe du relâchement. Donnez l'invariant maintenu par l'algorithme et montrez qu'il est bien vérifié.