Ollscoil na hÉireann
The National University of Ireland

Coláiste na hOllscoile, Corcaigh
University College, Cork

Summer Examination 2013

CS4407 Algorithm Analysis

Prof. G. Provan
Prof. B. O'Sullivan (HoD)
Prof. Ian Gent (extern)

Attempt all questions

Total marks: 80

90 minutes

Please answer all questions
Points for each question are indicated by [xx]

1. **[15]** We are given as input two sorted arrays A and B, each of length *n* and containing integers. Our task is to find if there exists *i* and *j* such that A[*i*] + B[*j*] = c, for some integer c.
   a. **[5]** Define an algorithm to solve this problem in time O(*nlogn*).

   The pseudo-code is as follows:
   INPUT: A, B, c
   Do for i=1 to n such that A[i] ≤c
   Apply a binary search algorithm on B, testing if A[*i*] + B[*j*] = c to select the half of array B we continue searching
   If A[*i*] + B[*j*] = c return (i,j)

   The outer loop over A is O(*n*). The inner-loop, a binary search over B, is O(log*n*). Hence the total complexity is O(*nlogn*).

   b. **[5]** Define an algorithm to solve this problem in time O(*n*).

   The pseudo-code is as follows:
   D-search (A, B, c)
   n ← length[A]
   i ← 1
   j ← n
   (i*,j*) ← dual-search (i,j)
   if i*< n OR j*<n return true
   else return false

   Dual-search (i,j)
   Do while i,j ≤ n
   if A[*i*] + B[*j*] = c return (i,j)
   else if A[*i*] + B[*j*] > c Dual-search (i,j-1)
   else if A[*i*] + B[*j*] < c Dual-search (i+1,j)
   end do

   The loop in Dual-search over A peforms at most O(*n*) while performing at most O(*n*) steps over B. Hence the total complexity is O(*n*).

   c. **[5]** Prove that your O(*n*) algorithm is correct.

   We use the loop invariance method for our proof. Our invariant is that the unsearched portion of the array contains the solution, if it exists.
   - At the start, by definition our loop invariant must be true.
   - In our loop, we take one of 3 options: (a) we have found the solution and return it, or we increment i when A[*i*] + B[*j*] > c, or we decrement j when

A[i] + B[j] < c. The 3 options are exhaustive, and none violates the loop invariant.

- Exit condition: upon exit we either return the solution, or we have reached a point where the search space is empty and no solution exists.

2. **[20]** Consider a knapsack problem where we want to pack a collection of different-sized balls, each with integer-valued volume *V* and weight *W*, into a knapsack of maximum volume such that the weight of the knapsack is minimal. We assume that we have a source of balls with set $\mathcal{V}$ of volumes $\{V_1, V_2,\ldots,V_n\}$, and each ball weighs the same.

   a. **[5]** Define a greedy algorithm to solve this knapsack problem if we apply the heuristic of always taking the maximum-volume ball possible at each step.

1. Sort the items in decreasing order of their ratios.
2. Set TotalVolume = 0 and j = 1. V* is the knapsack's capacity
3. while (TotalVolume < V*) do for the sorted items $I_1,\ldots,I_n$
   Let r = W - TotalWeight. (Variable r represents the remaining capacity of the knapsack.)
   if (V_j < r) then
   Add I_j to knapsack.
   Set TotalVolume = TotalVolume + V_j.
   j = j + 1.
4. Output the contents of Knapsack.

   b. **[4]** Is this algorithm optimal? This algorithm is optimal, but only for particular sets of volumes $\mathcal{V}$. (See part (c).)

   c. **[4]** Will this algorithm work for all possible sets of volumes $\mathcal{V}$? No, and a single counter-example suffices to show this is not true. Consider the case where there are 3 types of balls, with large=6cc, medium=4cc and small =1cc. If the total volume is 9cc, the algorithm will return 1 large and 3 small, with weight 4, whereas the optimal solution is 2 medium and 1 small, with weight 3.

   d. **[4]** Is there a selection heuristic that is optimal for this problem? No, there is no optimal heuristic.

   e. **[3]** If the balls vary in weight, is there an optimal selection heuristic? Yes, the algorithm is as below:

1. For each item I_j , compute the ratio v_j=w_j (i.e., value per unit weight), $1 \leq j \leq n$.
2. Sort the items in decreasing order of their ratios. Rename the items if necessary so that the sorted order of items is [$I_1,I_2,\ldots, I_n$].
3. Set TotalWeight = 0 and j = 1.
4. while (TotalWeight < W) do
   Let r = W- TotalWeight. (Variable r represents the remaining capacity of the knapsack.)
   if (w_j > r) then
   Add r lbs of item I_j to knapsack.
   Set TotalWeight = W.
   else
   Add I_j to knapsack.
   Set TotalWeight = TotalWeight + w_j.
   j = j + 1.
5. Output the contents of Knapsack.

3. **[20]** Prove that the problem FEEDBACK VERTEX SET (FVS) is NP-complete. We define FVS as follows:

INSTANCE: A directed graph G = (V, E) and positive integer $k \leq |V|$.
QUESTION: Is there a subset $V' \subseteq V$ with $|V'| \leq k$ such that V' contains at least one vertex from every directed cycle in G?

(Assume that you need to define a reduction from one of the following NP-complete problems studied in class: HAMILTON CIRCUIT, VERTEX COVER, CLIQUE, INDEPENDENT SET, 3-SAT)

We first prove that FEEDBACK VERTEX SET (FVS) is in **NP**: this is because a subset can be guessed and verified for the desired property in polynomial time.

Second, we construct a transformation that maps the known NP-complete problem, VERTEX COVER (VC), into FVS. The transformation takes an instance I_1 of VC as $G_1=(V_1,E_1)$ and K_1, and constructs an instance I_2 of FVS as $G_2 = (V_2,A_2)$ and K_2 as follows:

$V_2 = V_1$
$A_2 = \{(u,v), (v,u) \mid (u,v) \in E_1\}$
$K_2 = K_1$

The transformation can clearly be done in polynomial time, because we are just doubling all the edges.
Finally we need to prove that a solution to VC is true iff the corresponding solution to FVS is true.
$\Rightarrow$ Suppose I_1 has a vertex cover. Then, for each arc in G_2, at least one of the "head" or the "tail" is covered by the vertex cover. Thus, any cycle includes a vertex in the vertex cover. This means that the vertex cover is also a feedback vertex set of G_2.
$\Leftarrow$ Suppose I_2 has a feedback vertex set V'. This means that any cycle in G_2 has at least one vertex in V'. In particular, any cycle of the form (u,v,u) has either u or v (or both) in V'. This means that at least one endpoint of edge (u,v) in G_1 is in V'. This means that V' is a vertex cover of G_1.

4. **[10]** Suppose that A is a polynomial-time randomized algorithm for Problem X, whose "yes" answers are always correct, and that on any member of X, A answers "yes" with probability at least $1/n^2$.
   a. [**10**] Show that there exists a polynomial-time randomized decision procedure B for X that is correct with probability at least 3/4 on any input, and B may be built from A without any unproven assumptions.

B will run A independently k times, where k will be computed below, and answer "yes" iff at least one of the A-answers is "yes". If the correct answer is "no", then each A run will answer "no" and B will answer "no" -- in this case B is correct with probability 1. If the correct answer is "yes", the probability that each run of A answers "no" is at most $1 - 1/n^2$. The probability that k consecutive runs of A all answer "no" is at most $(1 - 1/n^2)^k$. This is the only situation in which B is incorrect.

So it suffices to pick k so that $(1 - 1/n^2)^k$ is at most 1/4. Remember that $(1 - 1/n^2)^{n2}$ is very close to 1/e. So setting k to be $2n^2$, we get a probability about $1/e^2$ which is less than 1/4 because e is about 2.71828, greater than 2.

5. [**15**] A graph G(V,E) is *k*-colorable, for some integer *k>0*, if we can color the vertices with the *k* colours such that no two vertices that share an edge have the same colour.
    a. [**5**] Show that the decision problem of whether an undirected graph is 2-colorable lies in P. [Hint: Use a greedy algorithm or Breadth First Search.]
    b. [**5**] Prove that an undirected graph is 2-colorable iff it does not have a cycle with an odd number of vertices. [Hint: use a proof by contradiction.]
    c. [**5**] Using the above two parts, prove that your algorithm is correct.

I) BFS solution
If there is an non-tree edge between two vertices of the same layer then answer "no". Otherwise label alternate layers in alternate colours.
II) Greedy Solution
Start with any vertex and label it colour 1. Greedily label a vertex adjacent to a coloured vertex the opposite colour
Restart the above for every connected component
Output yes - if all edges connect vertices of opposite colours and no otherwise.

(b) Prove that an undirected graph is 2-colorable iff it does not have a cycle with an odd number of vertices
⇒ (proof by contrapositive):
If it has a cycle with an odd number of vertices, then you can't colour it with two colours because as you walk around the cycles, the colors have to alternate. Starting with colour 1 you will run into colour 1 again as you walk around the cycle while alternating colours.
⇒ (direct proof):
If the graph is 2 colourable, then all edges are between vertices of opposite colours. So the graph is bipartite. All cycle in such a graph have an even number of edges because they go left and right and equal number of times.

(constructive part of proof):
Both of the above algorithms never stop with answer no, because doing so implies the existence of an odd cycle. Since there is no odd cycle, the algorithms construct a 2-coloring.

(c) Using the above, prove that your algorithm is correct
I) BFS solution
The BFS answers no i_ it detects an intra-level non-tree edge i_ there is an odd cycle containing that edge and the first vertex of the connected component.
II) Greedy Solution
The greedy algorithm produces an inconsistent colouring along an edge, then this edge can be completed to an odd cycle. Also if there is no odd cycle, then the greedy algorithm cannot answer no because this would give rise to an odd cycle.