CS3510 Design & Analysis of Algorithms Section B

Fall 2016 Final Exam Solutions

Instructor: Richard Peng *In class, Friday, Dec 9, 2016*

Problem	Title	Points	Parts	Grade	Initials
0	**Name / student number on top of every page**	1	1		
1	Master Theorem	4	4		
2	Scrooge's Knapsack	4	1		
3	Sorting by Reversals	4	3		
4	Formulating Linear Programs	4	2		
5	NP-Hardness	5	2		
6	Approximating Diameter	4	3		
Bonus 1	Infinite / Very Long Chase	1	2		
Total		26			

- You may use a sheet with notes on both sides, which will be handed in at the end of the test.
- This booklet contains $6 + 2$ questions on ? pages (due to solutions), including this one. Please check the number of pages that you have before starting, and notifying proctors immediately of missing pages.
- Place your GT ID on the desk. The proctors will come around to check it during the test.
- Do not open this quiz booklet until you are directed to do so. Read all the instructions first.
- At the start of the test, write down your name and student number on **every** page and your sheet of notes.
- You may use a calculator, but not any device with transmitting functions.
- Note that $\log_a b \leq c$ if and only if $b \leq a^c$. You may leave answers in the form $\log_a b$.
- You may assume that $n \leq O(m)$ for all graphs, aka. there are no degree 0 vertices.
- You have 170 minutes to earn $1 + 25$ points, the test is graded out of 25. Do not spend too much time on any one problem. Three times a problem's point value is a reasonable estimate of how many minutes to spend on it (on the first pass).
- Check your answers. During the review session, Richard lost at least a total of 5 points to small mistakes :-(.
- Write your solutions in the space provided. If you run out of space, continue your answer on the back of the same sheet and make a note on the front of the sheet.

- You may use any of the theorems/facts/lemmas/algorithms covered in class, homeworks, or on textbook without re-proving them unless explicitly stated otherwise.
- If necessary make reasonable assumptions but please be sure to state them clearly
- When we ask you to give an algorithm in this test, describe your algorithm in English or pseudocode, and provide a short argument for correctness and running time. You do not need to provide a diagram or example unless it helps make your explanation clearer.
- You can use the back of the pages for scratch work.
- Good luck!
- **Disclaimer:** no coyotes, or graphs, were harmed in the making of this exam and its solutions.

0 (/1 point) **Write your name and student id on top of every page.**

1. (4 points) Master Theorem

   Solve the following runtime recurrences using the master theorem, or another method of your choice. In each of the cases, you may assume $T(n) = 1$ for all $n \leq 100$ as the base case.

   Your can express your answers using big-O, but they should be tight up to constant factors. Only the final answer will be marked.

   (a) $T(n) = 4T(n/4) + n^2$
   SOLUTION:
   This fits into the recurrence $T(n) = aT(n/b) + n^c$ with $a = 4$, $b = 4$, $c = 2$. $\log_b a = 1 < 2$, so Master theorem gives $T(n) = O(n^2)$.

   (b) $T(n) = 8T(n/2) + n^3$
   SOLUTION:
   This fits into the recurrence $T(n) = aT(n/b) + n^c$ with $a = 8$, $b = 2$, $c = 3$. $\log_b a = 3 = c$, so Master theorem gives $T(n) = O(n^3 \log n)$.

   (c) $T(n) = 3T(n/3) + 1$
   SOLUTION:
   This fits into the recurrence $T(n) = aT(n/b) + n^c$ with $a = 3$, $b = 3$, $c = 0$. $\log_b a = 1 > 0$, so Master theorem gives $T(n) = O(n)$.

   (d) $T(n) = 5T(n/5) + n$
   SOLUTION:
   This fits into the recurrence $T(n) = aT(n/b) + n^c$ with $a = 5$, $b = 5$, $c = 1$. $\log_b a = 1 = c$, so Master theorem gives $T(n) = O(n \log n)$.

2. (4 points) Knapsack

   Scrooge McDuck is trying to put together a holiday gift knapsack for Willie Coyote. He has n items to choose from, each with infinitely many copies (aka. knapsack with replacement). Item i has weight w_i, and value v_i.

   Scrooge wants to pick some items (possibility with duplicates) so their total weight is **exactly** W, while **minimizing** the total value of the items picked.

   To solve this via dyanmic programming, we can create the state of

$$OPT[w],$$

   which is the minimum total value needed to make a weight of w, for every $0 \leq w \leq W$. Of course, we set the base case to $OPT[0] = 0$.

   Then the transition of the dynamic program is:

$$OPT[w] = \min_{i:w \geq w_i} \left(OPT[w - w_i] + v_i \right).$$

   Suppose we have $W = 8$, and three items:

$$w_1 = 1, v_1 = 3$$
$$w_2 = 3, v_2 = 2$$
$$w_3 = 5, v_3 = 1$$

   Compute the values of $OPT[1] \ldots OPT[8]$: the minimum value needed to make up weights $1 \ldots 10$ respectively. Each correct value is worth 1/2 points.

   SOLUTION:

$$OPT[0] = 0$$
$$OPT[1] = 3$$
$$OPT[2] = 6$$
$$OPT[3] = 2$$
$$OPT[4] = 5$$
$$OPT[5] = 1$$
$$OPT[6] = 4$$
$$OPT[7] = 7$$
$$OPT[8] = 3$$

3. (4 points) Sorting by Reversals.

The paper 'transforming cabbage into turnip' showed that the mitochondrial DNA of cabbage can be transformed via three reversals into the mitochondrial DNA of cabbage. Formally, a reversal of a string of length n, $s_1 \ldots s_n$, on the interval $l \ldots r$, would transform it into

$$s_1, s_2, \ldots s_{l-1}, s_r, s_{r-1}, \ldots s_{l+1}, s_l, s_{r+1}, \ldots s_n.$$

As an example, reversing the interval $[3 \ldots 5]$ turns the string ABCDEF into ABEDCF, and reversing the interval $[1, 5]$ of the string COYOTE turns it into into TOYOCE.

(a) (1 point) Show that the minimum number of reversals needed to transform A into B is the same as the minimum number of reversals needed to transform B into A.

SOLUTION:
If we can get from A to B using the reversal $[l, r]$, doing them same reversal again on B would give us A. Therefore, the graph is undirected, so the distance from A to B is the same as the distance from B to A.

(b) (1 point) Show that for any string S of length n, the number of strings that can be obtained from S via a single reversal is at most $O(n^2)$.

SOLUTION:
There are only $O(n^2)$ choices of l and r.

(c) (2 points) Devise an algorithm that takes two binary strings of length n, A and B, and computes in $O(2^n \times n^{20})$ time the minimum number of reversals needed to transform the first string to the second.

SOLUTION:
By the degree bound given in part b), a graph on these 2^n vertices has $m = O(2^n n^2)$ edges. Once we have this graph, computing the shortest path from A to B gives the answer. This can be done in either $O(m)$ or $O(m \log n)$ time using BFS/Dijkstra's algorithm, both of whose runtimes are below $O(2^n \times n^{20})$ (note that $\log(2^n) = O(n)$).

4. (4 points) Formulating linear programs.

   ACME Corp is currently trying to fill a large order of rockets (for launching coyotes), catapults (for launching stuff at coyotes), and rubber bands (for tripping coyotes), measured in tons. Maunfacturing each ton of material requires a certain amount of time, and a certain amount of grey goo, both of which are limited.

   Note that it is acceptable to manufacturing a fraction of a ton (e.g. $0.3t$) of material.

   Specifically (for both parts of this problem), ACME corp currently has 8 hours of time, and 20 units of grey goo. Manufacturing each ton of the three products requires:

product	time (hours)	grey goo (units)
rocket	3	3
catapult	1	10
rubber band	2	5

   in all of your solutions, please use r, c, and b to denote the amount (in tons) of rockets, catapults, and rubber bands respectively.

   (a) (2 points) Write down a linear program that determines the maximum amount of products (in terms of tonnage) that ACME corp can make.
   SOLUTION:

$$\begin{aligned} \text{max} \quad & r + c + b \\ \text{subject to:} \quad & 3r + c + 2b \leq 8 \\ & 3r + 10c + 5b \leq 20 \\ & r, c, b \geq 0 \end{aligned}$$

   (b) (2 points) Due to the potential danger posed by grey goo, ACME corp would like to use up as much of its supply grey goo, while:

   i. spending at most 8 hours
   ii. manufacturing **a total** of at least 2 tons of rockets plus rubber bands.

   Formulate this as a linear program. **Note that we now want to maximize the amount of goo used, not the total tonnage of products produced.**
   SOLUTION:

$$\begin{aligned} \text{max} \quad & 3r + 10c + 5b \\ \text{subject to:} \quad & 3r + c + 2b \leq 8 \\ & 3r + 10c + 5b \leq 20 \\ & r + b \geq 2 \\ & r, c, b \geq 0 \end{aligned}$$

GRADING:

(for both parts)
maximum of 0.5 for everything correct, or -0.5 for mistakes (including forgetting the ≥ 0).

5. (5 = 2 + 3 points) NP-Hardness

   In this problem, we will show that the maximum 2-SAT problem is NP-hard. A 2-SAT instance contains:

   (a) a set of varaibles $x_1 \dots x_n$,

   (b) m clauses, each containing the OR of two literals of the form x_i, $\overline{x}_i$:

   $$l_{i_1} \vee l_{i_2}.$$

   The decision version of the maximum 2-SAT probelm, MAX2SAT takes an 2-SAT instance and a variable g, and asks whether there exists an assignment to the x_is that satisfies at least g of the clauses.

   The goal of this problem is to show that MAX2SAT is NP-complete.

   (a) Show that MAX2SAT $\in$ NP.

   SOLUTION:

   We can create a verifier that takes an assignment of the variables, and counts the number of clauses that are satisfied.

   This runs in $O(m)$ time since it needs to loop through all the clauses once, therefore the verifier is in P.

   GRADING:

   1 point for stating what the verifier is, 1 point for relating it to the definition of NP.

   -0.5 for not comparing the number of satisfied clauses with g.

   (b) Consider the maximum cut problem, whose decision version is given an undirected, unweighted, graph $G = (V, E)$ and a guess g, determine if there is a cut S s.t. there are at least g edges between S and $V \setminus S$.

   In the textbook, it was shown that MAXCUT is NP-complete. Reduce maximum cut to maximum 2-SAT. Specifically, show:

   $$\text{MAXCUT} \rightarrow \text{MAX2SAT}$$

   Hint: have a variable x_u for each vertex u that denotes whether it's in S. For each edge uv, design a few clauses such that whether the edge is cut corresponds to the number of these clauses that are satisfied.

   SOLUTION:

   Given a graph G, consider the following instance of max-2-SAT: for each edge u, v, have two clauses of the form

   $$x_u \vee x_v \tag{1}$$
   $$\overline{x}_u \vee \overline{x}_v \tag{2}$$

   If $x_u \neq x_v$, two of these clauses are satisfied, while if $x_u = x_v$, only one of them is satisfied.

Therefore, asking maximum 2-SAT with the goal of $g + m$ where m is the number of edges will produce the same result as $\textsc{MaxCut}(G, g)$.

SOLUTION:

1 point for having the right reduction framework (edges to clauses, vertices to variables.)

2 points for giving the right reduction function, this includes building clauses from the edges, and changing the value of g.

-0.5 for any minor issues, including an incorrect goal value g in the new Max-2-SAT instance (this may not be $g + m$ depending on the reduction.)

-0.5 for not stating what the goal value given to the new Max-2-SAT instance is.

6. (4 points) Approximating Diameter of a Graph

   The distance between two vertices u and v in a graph is the length of the shortest path between them.

   The diameter of a graph is the maximum distance between a pair of vertices

$$D = \max_{u,v} \textsc{dist}(u, v).$$

   In this problem we will give an 2-approximation to the the diameter of an undirected graph with positive edge weights. That is, we will try to produce a pair of vertices, x, y, such that

$$\textsc{dist}(x, y) \geq \frac{1}{2}D.$$

   (a) (2 points) Use the fact that distances obey the triangle inequality, aka

$$\textsc{dist}(u, v) \leq \textsc{dist}(u, w) + \textsc{dist}(v, w)$$

   for any vertices u, v, w to show that for any vertex s,

$$D \leq 2 \cdot \max_{u} \textsc{dist}(s, u).$$

   SOLUTION:
   Let the vertex that maximizes the distance from s be t. Then we have for any u

$$\textsc{dist}(s, u) \leq \textsc{dist}(s, t).$$

   Putting this into triangle inequality involving u, v, and s gives:

$$\textsc{dist}(u, v) \leq \textsc{dist}(s, u) + \textsc{dist}(s, v) \leq 2\textsc{dist}(s, t)$$

   (b) (2 points) Use part a) to give an algorithm that on a connected graph with m edges, produces an 2-approximation of the diameter in $O(m \log^{10} n)$ time.
   Specifically, it should output in $O(m \log^{10} n)$ time a pair of vertices, x, y such that

$$\textsc{dist}(x, y) \geq \frac{1}{2}D.$$

   You may use the fact that on any weighted graph with positive edge weights, for any vertex s, Dijkstra's algorithm computes in $O(m \log n)$ time the values of $\textsc{dist}(s, u)$ for all vertices u.

   SOLUTION:
   The algorithm is:

   i. Pick an arbitrary vertex s.
   ii. Compute distances from s to all vertices u by running Dijkstra's algorithm with s as the source vertex.
   iii. Let t be the vertex that maximizes $\textsc{dist}(s, u)$, return s and t.

   The approximation factor of this algorithm follows from part b), while its running time follows from that of Dijkstra's algorithm ($O(m \log n)$), plus the cost of doing a linear scan on the distance values.

Bonus 1 (1 point)

Bonus, do not attempt until you are sure that you've fully attempted all other questions (the point value of this is too low)

Infinite / Very Long Chase

As you may remember from the homeworks, many things in the Wild West (e.g. coyotes) have been observed chasing other things (e. g. roadrunners).

As an intern at ACME research (which, as you may recall, is the equivalent of Amazon.com Research for Coyotes), you're trying to investigate the chasing relations between objects.

Story ends here, mathematical description of problem starts below.

You're given n types of objects, as well as m observations of the form 'an object of type i was observed chasing an object of type j'.

Under the assumption that these are the only possible pairs of i chasing j, you would like to investigate the possibility of 'chasing chains', that is, a sequence of (possibly repeating) objects $i_1, i_2 \ldots i_t$ such that i_j was spotted chasing i_{j+1} for every $1 \leq j < t$.

(a) (1 point) Give an $O(n^{10})$ time algorithm for finding the longest chasing chain in a graph, or determine if its length is infinite.

SOLUTION:

This problem is similar to problem 6 in that the longest chasing chain can be thought of as the diameter of a directed graph $G(V, E)$ where the n objects are the vertices and the m instances of 'object i chasing object j' are the directed edges (i, j). One caveat is that a cycle is an infinite chasing chain, so we might first determine whether the graph contains a cycle, and if not, find its diameter.

Recall the DFS (depth-first search) procedure starting from some $v \in V$, and modify it slightly to store how many times each vertex is visited by incrementing its *visited* label instead of simply assigning it *true*. Perform DFS for each $v \in V$, and if at any time a vertex is visited twice, this means that G contains a cycle, and we have an infinite chase.

Also modify DFS to store the depths of the DFS trees in $DEPTH$ for each $v \in V$. Easy implementation of this only adds constant time to the DFS procedure. If no cycles are found, then the following is the longest chasing chain:

$$\max_{v \in V} \{DEPTH[v]\}$$

Runtime: DFS is $O(n + m)$. Since in any graph, $m = O(n^2)$, DFS is $O(n + n^2) = O(n^2)$. Since we run DFS $O(n)$ times, and our modifications only add constant time for each iteration, our total time complexity is $O(n^3)$.

GRADING:

1 for correct algorithm and runtime justification.

-.5 for missing runtime or minor error.
Note that other solutions are possible.