CMSC 451:Fall 2017 Dave Mount

Practice Problems for the Midterm

The midterm will be on **Thu, Oct 26** in class. The exam will be closed-book and closed-notes, but you will be allowed one sheet of notes (front and back).

Disclaimer: These are practice problems, which have been taken from old homeworks and exams. They **do not** reflect the actual length, difficulty, or coverage for the exam.

Problem 0. You should expect one problem in which you will be asked to work an example of one of the algorithms we have presented in class on a short example. (Likely candidates: DFS (including applications such as cut vertices), Dijkstra's algorithm, Huffman's algorithm, any greedy algorithm from the lecture notes, LCS.)

Problem 1. Short answer questions:

(a) As a function of n, what is the asymptotic running time of the following function? (Express your running time using Θ notation.)

```
void f(int n) {
  i = n;
  while (i > 0) {
    for (j = 1 to i) print("hello!\n");
    i = i/2;
  }
}
```

(b) Assume that G is an undirected connected graph.

(i) True or False: There is a graph G that has a cut vertex but no cut edges.

(ii) True or False: There is a graph G that has a cut edge but no cut vertices.

(c) What is the maximum number of edges in an undirected graph with n vertices, in which each vertex has degree at most k? (For full credit, express your answer as an exact function of n and k. For partial credit give it in big-O notation.)

(d) Consider the code $a = \langle 00 \rangle$, $b = \langle 01 \rangle$, $c = \langle 11 \rangle$, $d = \langle 101 \rangle$.

(i) Is this a prefix code? Explain.

(ii) Could this code be generated by Huffman's algorithm? Explain.

(e) G is a directed graph with negative weight edges but no negative weight cycles. Which of the following hold for *Dijkstra's algorithm* on G (as given in class):

(i) The output is correct, and it will run in the same time bound as given in class.

(ii) The output is correct, but it will take longer than the time bound given in class.

(iii) It will terminate, but may produce an incorrect result.

(iv) It may not terminate.

(f) Repeat problem (e), but for the *Bellman-Ford algorithm* (as given in class).

(g) Repeat problem (e), but for the *Floyd-Warshall algorithm* (as given in class).

Problem 2. Recall the *Interval Scheduling Problem.* A set of n requests is given, each with a given start and finish time, $[s_i, f_i]$. The objective is to compute the maximum number of activities whose corresponding intervals do not overlap. In class we presented an greedy algorithm that solves this problem. We will consider some alternatives here.

(a) **Earliest Activity First** (EAF): Schedule the activity with the earliest start time. Remove all activities that overlap it. Repeat until no more activities remain.

Give an example to show that EAF is not optimal. Your example should show not only that it is not optimal, but its approximation ratio can be *arbitrarily high.*

(b) **Shortest Activity First** (SAF): Schedule the activity with the smallest duration $(f_i - s_i)$. Remove all activities that overlap it. Repeat until no more activities remain.

Give an example to show that SAF is not optimal. (Challenge: Show that it has an approximation ratio of at most two.)

Problem 3. Given a DAG $G = (V, E)$, a path of G is said to be *maximal* if it ends at a vertex with outdegree zero. (If u is a vertex of outdegree zero then the path consisting of just u itself is a maximal path.) For each vertex u, let $P(u)$ denote the number of maximal paths that start at u. (If u has outdegree zero, then $P(u) = 1$.) Describe an $O(n + m)$ algorithm which, given a DAG, $G = (V, E)$ as an adjacency list, computes $P(u)$ for each $u \in V$. (Hint: Use DFS.)

Problem 4. Professor Farnsworth drives from College Park to Miami Florida along I-95. He starts with a full tank and can go for 100 miles on a full tank. Let $x_1 < \ldots < x_n$ denote the locations of the various gas stations along the way, measured in miles from College Park. Present an algorithm that determines the *fewest number* of gas stations he needs to stop at to make it to Miami without running out of gas along the way. Justify the correctness of your algorithm. (You may assume that the gap between consecutive stations never exceeds 100 miles.)

Problem 5. A pharmacist has W pills and n empty bottles. Let b_i denote the number of pills that can fit in bottle i. Let v_i denote the cost of purchasing bottle i. Given W, b_i's and v_i's, we wish to compute the cheapest subset of bottles into which to place all W pills. (You may assume that $\sum_i b_i \geq W$.)

(a) Suppose that you only pay for the *fraction* of the bottle that is used. For example, if the ith bottle is half filled with pills, you pay only $v_i/2$. Present an algorithm for this version of the problem. (Hint: Use greedy.) Justify your algorithm's correctness.

(b) Suppose that this assumption does not hold. That is, you must buy the entire bottle, even if only a portion of it is used. Present an algorithm for this version of the problem. (Hint: Use DP. It suffices to give the recursive formulation.) Justify your algorithm's correctness.

Problem 6.

(a) Describe a greedy algorithm for making change for a coin system based on powers of 2. In particular, you have denominations $\{1, 2, 4, 8, 16\}$. Given any amount A, determine the minimum number of coins from this set whose sum is A. (Challenge: Do this for the USA system of coins $\{1, 5, 10, 25\}$.)

(b) The greedy algorithm is not optimum for all choices of coin denominations. Give a set of coin denominations (which must include a 1-cent coin) for which the greedy algorithm does not always give the minimum number of coins. Explain briefly. (Challenge: Find an example, possibly from history, of an *actual* coin system where greedy fails.)

Problem 7. You operate a business that has two offices, one in Washington DC and one in Los Angeles. Each week, you need to decide whether you want to work in the DC office or the LA office. Depending on the week, your business makes more profit by having you at one office or the other. You are given a table of weekly profits, based on your location. Here is an example:

Week	1	2	3	4	5
DC	\$400	\$100	\$200	\$50	\$1100
LA	\$210	\$900	\$100	\$1500	\$20

Clearly, you would prefer to work at the location where you get the greater profit, but here is the catch. It costs \$1000 to fly from one office to the other. (For example, if you do the first job in DC, the next three in LA, and the last in DC, the total profit will be $\$400 - \$1000 + (\$900 + \$100 + \$1500) - \$1000 + \$1100 = \$2000$.

You are given two lists of length n, DC[$1..n$] and LA[$1..n$], where DC[i] is the profit for spending week i in DC, and LA[i] is the profit for spending week i in LA. Present an efficient algorithm, which given these two arrays, determines your maximum overall profit. You must *start* and *end* in DC, but you may travel back and forth any number of times. Briefly justify your algorithm's correctness and derive its running time.

Hint: $O(n)$ time is possible using dynamic programming. It suffices to give just the recursive rule. You will need to find a way to keep track of where you were the previous week.

Problem 8. The objective of this problem is to write a dynamic programming algorithm to play a game. Two players, called Jen and Ben alternate in taking moves, with Jen always going first. Initially the board consists of three piles of diamonds, which we denote (A, B, C), meaning that there are A diamonds in the first pile, B in the second pile, and C in the third. The board always consists of three numbers that are nonnegative. During a move a player can do any one of the following:

(1) Remove 1 diamond from pile 1.
(2) Remove either 1 or 2 diamonds from pile 2.
(3) Remove either 2 or 3 diamonds from pile 3.

The first player who cannot make a move loses. (And the winner gets all the diamonds.) That is, if it is a player's turn to move and the board is either $(0, 0, 0)$ or $(0, 0, 1)$ then he/she loses.

Given the initial configuration, (A, B, C), and with the assumptions that Jen plays first and both players play as well as possible, determine which of the two players can force a win. (Since there is no element of chance, and the game is finite in length, one of the two can always force a win.)

For example, suppose that the initial board is $(0,1,4)$. In this case Jen can force a win. She uses rule (3) to remove 2 diamonds from pile 3, resulting in the board $(0,1,2)$. Ben's only choices are to remove 1 from pile 2 or 2 from pile 3, resulting in the possible boards $(0,0,2)$ and $(0,1,0)$. In either case, Jen can remove the final diamonds (using either rules (3) or (2), respectively) leaving Ben without a move.

(a) Derive a (recursive) dynamic programming rule to determine the winner, given the initial board (A, B, C). (Be sure to include a description of the basis cases.) Justify the correctness of your formulation. (For this part I do not want a full algorithm, just the recursive rule.) You are *not* allowed to use mathematical results from the game of Nim to find a short-cut solution.

(b) Present an implementation of recursive rule of part (a). (You may use memoization or the bottom-up method.) Express your running time as a function of A, B, and C.

Problem 9. Given a graph $G = (V, E)$, a subset of vertices $V' \subseteq V$ is called a *dominating set* if every vertex of G is either in the set V' or is a neighbor of a vertex in V'. For example, in Fig. 1(a) the set $\{4, 8\}$ is a dominating set of size two. In the *dominating set problem* you are given a graph G and the objective is to compute a dominating set of minimum size.

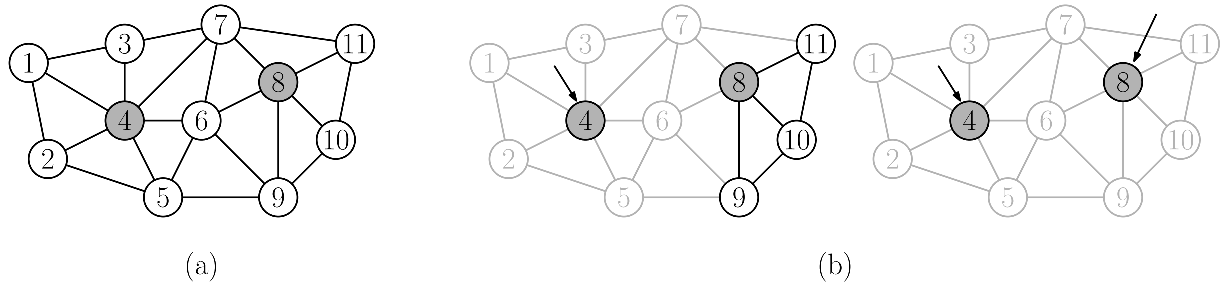

Figure 1: (a) Dominating set and (b) the greedy heuristic.

Consider the following greedy heuristic for this problem. Initially, all vertices are marks as *undominated*. Select a vertex that is adjacent to the maximum number of undominated vertex. Add this vertex to the dominating set. Mark it and all its neighbors as *dominated*. Repeat until no more undominated vertices remain. (For example, in Fig. 1(b) the heuristic would first select 4 which is adjacent to six undominated vertices, and then it could select either vertex 8 or 10, each of which is adjacent to three undominated vertices.)

(a) Present an example to show that this greedy heuristic is not optimal.

(b) Show that the greedy heuristic achieves an approximation factor of $\ln n$, where $n = |V|$. That is, if there exists a dominating set of size k, then the greedy heuristic outputs a dominating set of size at most $k \cdot \ln n$.

(Aside: This problem is known to be NP-hard.)