**Practice Problems for the Midterm**

(Updated: Tue, Apr 1.) The midterm will be on **Thu, Apr 3, 7-9pm in IRB 0324**. 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 computing strong components), Dijkstra's algorithm, any greedy algorithm from the lecture notes, or any of the DP algorithms from class.)

**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 $\Theta$ notation.)

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

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

(c) Suppose that you perform a DFS on an undirected graph $G = (V, E)$, and for each vertex $u \in V$, you compute the discovery time $d[u]$ and finish time $f[u]$. Let $u$ be a descendant of $v$ in the DFS tree. What (if anything) can be inferred about the relative orders of $d[u]$, $f[u]$, $d[v]$, and $f[v]$?

(d) We know that Dijkstra's algorithm may fail on graphs with negative edge weights. Prof. DM devises a trick to fix this. He adds a sufficiently large positive constant to every edge in the graph so that the edge weights are now *all positive*. He then runs Dijkstra on the result.

Will this algorithm correctly generate shortest paths for the original graph? Explain why or why not.

(e) Recall the variant of the Bellman-Ford algorithm given in class, which performs relax operations on all the edges until the $d[v]$ values converge. Suppose that you run the algorithm on some graph, and it goes into an infinite loop. What can you infer about this graph?

(f) Gonzalez's algorithm (the greedy $k$-center heuristic) is run on a set $P$ of $n = 100$ points in the plane. For $i \geq 1$, let $C_i$ denote the set of centers after $i$ iterations. Let $\Delta_i$ denote the maximum distance of any point of $P$ to its closest center in $C_i$. Let $\Gamma_i$ denote the minimum distance between any two centers of $C_i$. Which of the following statements necessarily holds? (Select all that apply.)

(i) $\Delta_4 \leq \Delta_3$
(ii) $\Gamma_4 \leq \Gamma_3$
(iii) $\Gamma_4 \leq \Delta_3$
(iv) $\Gamma_4 \geq \Delta_3$

(g) Suppose that the capacities of the edges of an $s$-$t$ network are all integers that are evenly divisible by 3. (E.g., $3, 6, 9, 12, \ldots$.) What can be inferred about the maximum flow value in this network?

**Problem 2.** Let $G = (V, E)$ be an directed acyclic graph (DAG) (see Fig. 1(a)). Define the *length* of a path in $G$ to be the number of edges on the path.

Present an efficient algorithm that labels each vertex $u$ with the length $L[u]$ of the longest path ending at $u$ (see Fig. 1(b)). Your algorithm should run in time $O(n+m)$, where $n = |V|$ and $m = |E|$. Briefly justify your algorithm's correctness and derive its running time. (Hint: It may be helpful to some preprocessing in $O(n + m)$ time.)

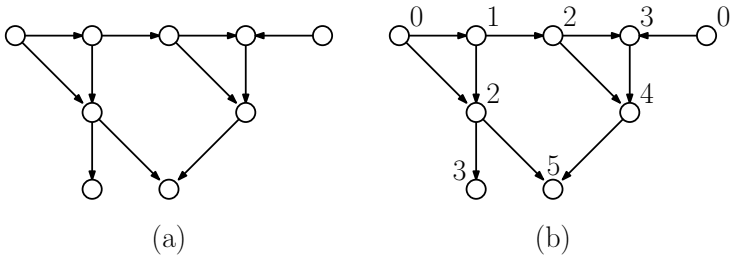


Figure 1: Longest path ending at a vertex in a DAG.

**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 $P(u) = 1$, which counts the trivial path $\langle u \rangle$.)

Present 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.** Prof. DM drives from College Park to Miami Florida along I-95 to seek the lovely and elusive Prothonotary Warbler. 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.** You are given a collection of files $\{f_1, \ldots, f_n\}$ files that are to be stored on a tape. File $f_i$ is $s_i$ bytes long. The tape is long enough to store all the files. The probability of accessing file $f_i$ is $p_i$, where $0 \leq p_i \leq 1$, and $\sum_{i=1}^{n} p_i = 1$. The tape is rewound before each access, and so the time to access any file is proportional to the distance from the front of the tape to the end of the file.

A *layout* of files on the tape is given by a permutation $\pi = \langle \pi_1, \ldots, \pi_n \rangle$ of the numbers $\{1, \ldots, n\}$. (For example, Fig. 2 shows the layout $(4, 2, 1, 3)$.) Given a layout $\pi$, the *expected cost* of accessing the $i$th file on the tape is the product of its access probability and the distance from the start of the tape to the end of the file. The *total cost* of a layout $\pi$ is the sum of the expected costs for all the files, denoted $T(\pi)$.

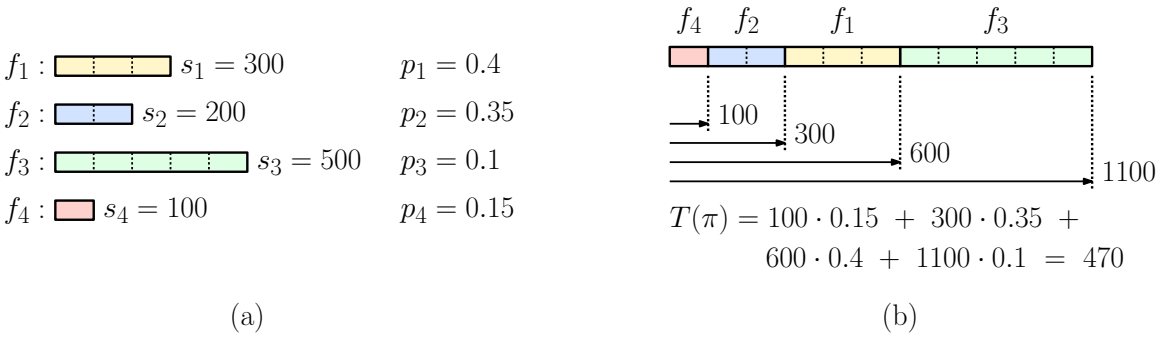


Figure 2: Placing files on a tape to minimize access time.

(a) Present a (short) counterexample so show that laying out the files on the tape in increasing order of size ($s_i$) is not optimal.

(b) Present a (short) counterexample so show that laying out the files on the tape in decreasing order of access probability ($p_i$) is not optimal.

(c) Present an algorithm, which given $s_i$'s and $p_i$'s, determines a layout $\pi$ of minimum total cost. Prove your algorithm's correctness and derive its running time. (Hint: Use a greedy approach.)

**Problem 6.** Consider a variant of the LCS problem where symbols are not required to match exactly. You are given two sequences of digits $X = \langle x_1, \ldots, x_m \rangle$ and $Y = \langle y_1, \ldots, y_n \rangle$ where $x_i, y_j \in \{0, \ldots, 9\}$ (see Fig. 3(a)). You wish to match some subsequence of $X$ with a subsequence of $Y$ where slight mismatches are allowed.

When matching two symbols $x$ and $y$, if $x = y$, this match counts as $+2$. If $|x - y| = 1$, then the match counts as $+1$, and if $|x - y| \geq 2$, then the symbols cannot be matched. (For example, in Fig. 3(b) we show a correspondence with total weight 4.5.).

Define the *LCS with Mismatches* (LCSM) to be the maximum weight achievable by matching corresponding symbols of two equal-sized subsequences of $X$ and $Y$ using the above weighted criterion.

Present a recursive DP formulation for this problem. (Don't forget the basis case and be sure to indicate which table entry yields the final answer.) Provide a *brief* justification (not a formal proof).

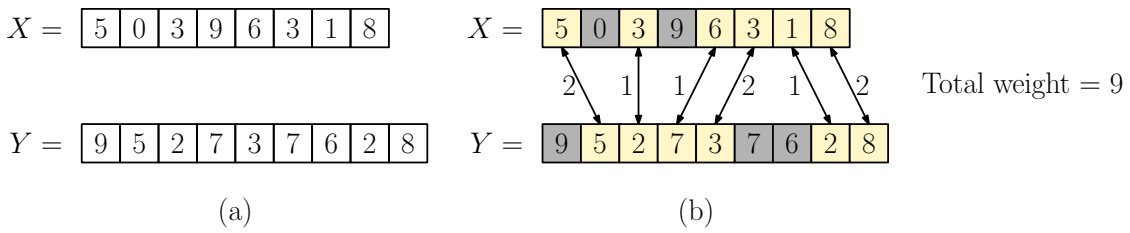


Figure 3: LCS with mismatches.

**Problem 7.** 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 $i$th 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 8.** In the long forgotten past (before Venmo) there were curious, round things called *coins*, which could be used for buying stuff. In these quaint times, when someone paid too much for an item, an overworked, underpaid person, called a *cashier*, would *make change* by counting out the smallest number of coins to make up the difference.

(a) Consider an infinite coin system based on powers of 3, that is, the coin values are $\{1, 3, 9, 27, \ldots\}$. Describe a greedy algorithm for making change in such a system. That is, given any amount $R$, determine the minimum number of coins from this set whose sum is $R$.

(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 may *fail* to yield the smallest number of coins. Explain briefly.

**Problem 9.** Prof. DM runs a side business selling photos of the lovely and elusive Prothonotary Warbler. His business has two offices, one in Washington DC and one in LA. Each week, he needs to decide whether to work in the DC office or the LA office. Depending on the week, his business makes more profit by having him at one office than the other. He is given a table of weekly profits, based on his location. Here is an example:

| Week | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| DC | \$40 | \$10 | \$20 | \$5 | \$110 |
| LA | \$21 | \$90 | \$10 | \$150 | \$2 |

Clearly, he would prefer to work at the location where he receives the greater profit, but here is the catch. It costs \$100 to fly (on Spirit Airlines) from one office to the other. (For example, if he does the first job in DC, the next three in LA, and the last in DC, his total profit will be $\$40 - \$100 + (\$90 + \$10 + \$150) - \$100 + \$110 = \$200$.

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 his maximum overall profit. You must *start* and *end* in DC, but he 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 formulation. You will need to find a way to keep track of where he was the previous week.

**Problem 10.** You are given a directed network $G = (V, E)$ with a root node $r$ and a set of terminal nodes $T = \{t_1, \ldots, t_k\}$. Present a polynomial time algorithm to determine the minimum number of edges to remove so that there is no path from $r$ to any of the terminals (see Fig 4). Prove that your algorithm is correct.

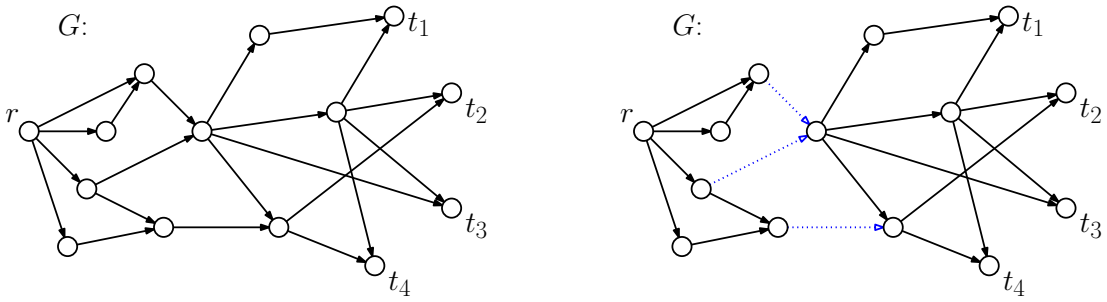


Figure 4: Eliminating edges to separated $r$ from terminals.

**Problem 11.** An edge of a flow network is said to be *critical* if decreasing the capacity of this edge results in a decrease in the maximum flow value. Present an efficient algorithm that, given an $s$-$t$ network $G$ finds any critical edge in a network (assuming one exists)