EECS 376: Foundations of Computer Science **Due 8:00pm, February 14**
University of Michigan, Winter 2024 **Homework 5** **(accepted until 9:59 pm, no credit after)**

This homework has 9 questions, for a total of 100 points and 10 extra-credit points.
Unless otherwise stated, each question requires *clear*, *logically correct*, and *sufficient* justification to convince the reader.
For bonus/extra-credit questions, we will provide very limited guidance in office hours and on Piazza, and we do not guarantee anything about the difficulty of these questions.
We strongly encourage you to typeset your solutions in LaTeX.
If you collaborated with someone, you must state their name(s). You must *write your own solution* for all problems and *may not use any other student's write-up.*

(0 pts) 0. **Before you start; before you submit.**

If applicable, state the name(s) and uniqname(s) of your collaborator(s).

Solution:

(10 pts) 1. **Self assessment.**

Carefully read and understand the posted solutions to the previous homework; you may also find the video "walkthroughs" in the Canvas Media Gallery helpful. Identify one part for which your own solution has the most room for improvement (e.g., has unsound reasoning, doesn't show what was required, could be significantly clearer or better organized, etc.). Copy or screenshot this solution, then in a few sentences, explain what was deficient and how it could be fixed.

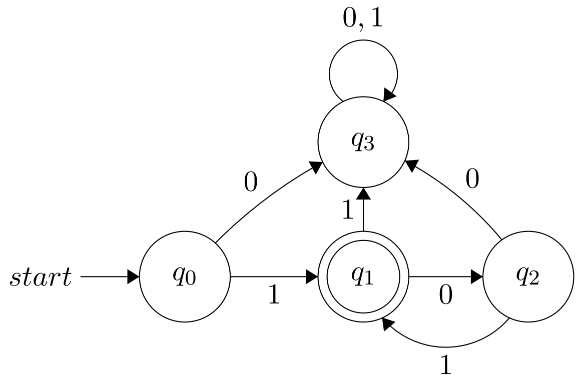
(Alternatively, if you think one of your solutions is significantly *better* than the posted one, copy it here and explain why you think it is better.)

If you didn't turn in the previous homework, then (1) state that you didn't turn it in, and (2) pick a problem that you think is particularly challenging from the previous homework, and explain the answer in your own words. You may reference the answer key, but your answer should be in your own words.

Solution:

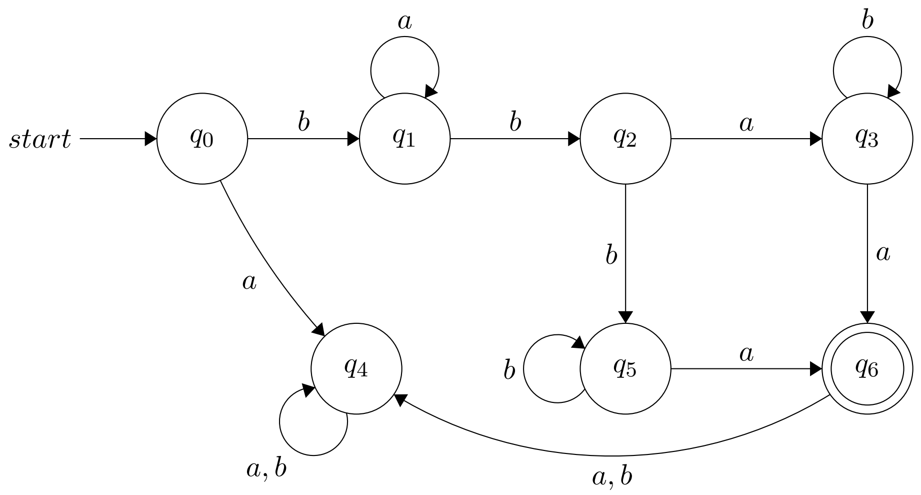
2. **Short answer.**

(4 pts) (a) Write the state-transition function (called δ in the lecture and the notes) for the following DFA, as a table. Also give a regular expression (using the "string notation" from class) for the language this DFA decides. No justification is required for either of these. For reference on formatting the state-transition function, see the lecture notes on Finite Automata.



Solution:

(4 pts) (b) Give a regular expression (using the "string notation" from class) for the language decided by the following DFA. No justification is required.



Solution:

(4 pts) (c) Consider the following claim: a minimum spanning tree in a graph is *unique* if the edge weights are *distinct* (all different).

Briefly explain what is wrong with the following bogus "proof" of this claim: *because edge weights are distinct, all choices in Kruskal's algorithm are completely determined—there are no "ties" between edge weights that the algorithm can choose how to break. So, the algorithm has only one possible output. Because Kruskal's algorithm is correct for the MST problem, its unique output must be the only minimum spanning tree in the graph.*

Solution:

(4 pts) (d) Give a correct proof for the claim in the previous part.

Solution:

3. **Greedy knapsack filling.**

Recall that in the "0-1" knapsack problem, you are given n items, where the ith item has weight w_i and value v_i (both positive), and a non-negative weight capacity C of the knapsack. An optimal solution is a subset $S \subseteq \{1, 2, \ldots, n\}$ of the items having maximum total value

$$\sum_{i \in S} v_i \ ,$$

under the constraint that the total weight is at most the knapsack capacity, i.e.,

$$\sum_{i \in S} w_i \leq C \ .$$

In this problem, we introduce the *fractional* variant of the knapsack problem, where one may take any fraction $p_i \in [0, 1]$ of any item i. Naturally, a p_i-fraction of item i weighs $p_i \cdot w_i$, and has value $p_i \cdot v_i$. The goal is to maximize the total value of the selected fractional items.

(3 pts) (a) Briefly explain why the optimal value for the fractional knapsack problem is *at least as large* as that of the original 0-1 knapsack problem (for the same weights, values, and knapsack capacity).

Solution:

(4 pts) (b) Consider the following two greedy algorithms for the fractional knapsack problem:

- Algorithm A: While there is still some unused knapsack capacity, choose an item having the *largest value* (among those not considered yet), and add the largest possible fraction of that item (up to the entire item) that will fit within the remaining knapsack capacity.
- Algorithm B: While there is still some unused knapsack capacity, choose an item having the *smallest weight* (among those not considered yet), and add the largest possible fraction of that item (up to the entire item) that will fit within the remaining knapsack capacity.

Suppose we run *both* algorithms A and B, and output a best one of their two outputs. Is this a correct algorithm for the fractional knapsack problem? If so, prove it. Otherwise, give an input for which this algorithm's output is not optimal, and show why it is not.

Solution:

(7 pts) (c) Consider the following greedy algorithm for the fractional knapsack problem:

- Algorithm C: While there is still some unused knapsack capacity, choose an item having the largest *relative value* v_i/w_i (among those not considered yet), and add the largest possible fraction of that item (up to the full item) that will fit within the remaining knapsack capacity.

Here is the skeleton of an inductive proof that Algorithm C outputs an optimal solution. Without loss of generality, suppose that the items are in sorted order by relative value $r_i = v_i/w_i$, from largest to smallest (the algorithm considers them in this order). View any selection of fractional items as a sequence $p_1, p_2, \ldots, p_n$, where p_i is the fraction of item i. Let $g_1, g_2, \ldots, g_n$ be the sequence of fractions that the greedy algorithm selects. We aim to prove the following claim:

> For *every* $k = 0, \ldots, n$, the first k values $g_1, \ldots, g_k$ of this sequence are a prefix of (i.e., are the first k item fractions in) *some* optimal solution.

Given this claim, the full sequence (for $k = n$) is itself an optimal solution, because there are no more items that can be (fractionally) selected.

Prove the claim by induction. Specifically, establish the base case $k = 0$, and then prove the inductive step, which says: for any $k < n$, if $g_1, \ldots, g_k$ is a prefix of *some* optimal solution $OPT = o_1, \ldots, o_n$, then $g_1, \ldots, g_k, g_{k+1}$ is a prefix of *some* optimal solution $OPT' = o'_1, \ldots, o'_n$. Use an exchange argument to modify OPT into a valid selection OPT', without reducing the value (so that it remains optimal).

Solution:

(8 pts) (d) Now consider a new variant of the fractional knapsack problem, in which there are ℓ knapsacks, where the jth knapsack has capacity C_j, and we aim to maximize the total value across all the knapsacks. Give a greedy algorithm that solves this variant.

Solution:

4. **Greedy touring.**

PETROBLUE, in support of the M Bus Formula One Team, has innovated a modular fuel pack for a bus that will transport EECS 376 students on a tour through various towns in Michigan. This fuel pack allows the vehicle to travel a certain distance before refueling. Each town has a service station where a fuel pack can be reloaded: the current pack is simply swapped out for a fresh full one. (Any remaining fuel in the removed pack will not be used by the bus.) The goal is to visit the towns in a specified order, while minimizing the number of fuel pack reloads.

We model this problem as follows. The bus can travel a distance up to R on a single fuel pack. There are n towns to visit in sequence. For $i = 1, \ldots, n$, the distance from the $(i-1)$st town (or from the starting point, when $i = 1$) to the ith town is $d_i \leq R$. Given R and an array $D[1, \ldots, n] = [d_1, \ldots, d_n]$ as input, the goal is to find a *smallest* set $S \subseteq \{1, \ldots, n\}$ of towns (specified by their indices) where refueling should occur, so that the bus can reach all n towns without running out of fuel at any point. The bus starts out with a full fuel pack, and no refueling is needed at the final town.

(8 pts) (a) Give a greedy algorithm (including pseudocode) that solves this problem. Defer a correctness argument to the next part, but provide everything else involved in "giving an algorithm" here.

Solution:

(8 pts) (b) Give an inductive proof using an exchange argument (like the ones presented in lecture and in the previous problem) that your algorithm from the previous part is correct.

Solution:

(4 pts) (c) Now, suppose that the price of a fuel pack may differ from town to town. For example, a fuel pack might cost \$8 in the first town, \$3 in the second town, etc. So, the input now also includes an array $C = [c_1, \ldots, c_n]$, where c_i is the cost of refueling in the ith town. The goal now is to complete the tour while *minimizing the total cost* of the fuel packs that are purchased along the way.

PETROBLUE claims that we should use the same greedy algorithm that you proposed above to solve this problem. Determine whether this claim is true or not. If it is, give a proof. Otherwise, provide a small counterexample: give a specific input, the output of your greedy algorithm on that input, and a better solution for that input; also, *very briefly* explain where your proof from the previous part fails for this problem.

Solution:

(3 EC pts) (d) *Optional extra credit.* Give an efficient dynamic programming algorithm (with recurrence and bottom-up implementation) that solves the problem from the previous part.

Solution:

5. **Selecting suitable skis.**

Suppose there are n skiers of various heights, and n pairs of skis having various lengths. (Both skis in a pair have the same length.) We want to match each skier with a pair of skis so as *minimize the total difference* between each skier's height and the length of their skis.

Formally, we are given a *sorted* array $P[1, \ldots, n]$ of the skiers' heights (where $P[i] \leq P[i+1]$ for all $i < n$), and a (not necessarily sorted) array $S[1, \ldots, n]$ of the skis' lengths. We want to return a rearrangement (permutation) $S'[1, \ldots, n]$ of S that minimizes

$$\sum_{i=1}^{n} \big|P[i] - S'[i]\big|.$$

(3 pts) (a) Give an $O(1)$-time algorithm for the case $n = 2$.

Solution:

(7 pts) (b) For general n, consider an algorithm that just returns S in sorted order. We can view this as a greedy algorithm that just repeatedly selects a pair of skis of shortest length (among those that remain). So, it makes exactly n selections $s'_1, s'_2, \ldots, s'_n$ in sorted order.

Give an inductive proof using an exchange argument (like the ones presented in lecture and in the previous problems) that this algorithm is correct.

Hint: Use the previous part as an ingredient of your proof.

Solution:

6. **Decision problems and languages.**

Recall that the goal of a *decision problem* is to determine whether a given input "object" has a certain "property", e.g., determine whether a given integer is prime, or whether a given string is a palindrome. In class, we said that any decision problem is equivalent to a *membership problem* for a corresponding *language*. That is, determining whether a given object has the property is equivalent to determining whether its encoding (as a string) is a member of the language.

For each of the following decision problems, (i) define a reasonable (finite) alphabet Σ, (ii) give the encoding (over the alphabet) of a representative input object, (iii) analyze the length of the encoding in terms of the value of the input, and (iv) define a language L for the corresponding decision problem. Use the notation $\langle X \rangle$ for the encoding of object X as a string over the alphabet. As examples, we have provided solutions to the first two parts.

(0 pts) (a) "Does a given non-negative integer k have 3 as its last digit, when written in base 10?"

Solution:

(i) A reasonable alphabet consists of the decimal digits, $\Sigma = \{\texttt{0}, \dots, \texttt{9}\}$.

(ii) Encoding: for an integer k, write it in base 10 in the usual way, as a string of digits. For example, if k is the integer forty-seven, then $\langle k \rangle = \texttt{47}$. We stress that this is a *string* of the characters `4` and `7`, which *represents* the number forty-seven.

(iii) The length of this encoding would be the number of digits in the value, which is $\Theta(\log k)$.

(iv) A corresponding language would be $L_{EndsWith3} = \{\langle k \rangle : k \bmod 10 = 3\}$. It would also be acceptable to copy the phrasing of the decision problem, i.e., $L_{EndsWith3} = \{\langle k \rangle : k \text{ written in base 10 has 3 as the last digit}\}$

Alternatively, we could also encode integers in binary, using the alphabet $\Sigma = \{0, 1\}$. For example, if $k = 5$, then $\langle k \rangle = \texttt{101}$. The length of this encoding is still $\Theta(\log k)$. The above two definitions of the language hold without modification, because they both describe membership only in terms of the *value* of k, not its encoding.

(0 pts) (b) "Is a given array of non-negative integers sorted?"

Solution:

(i) A reasonable alphabet Σ is the set of ASCII characters, or more selectively, the decimal digits along with some separator characters: $\Sigma = \{\texttt{0}, \dots, \texttt{9}, \texttt{[}, \texttt{,}, \texttt{]}\}$. (Notice that a comma is one of these characters.)

Note: we can't just use the decimal digits alone if we want to use some special symbols to indicate the start/end of the array and to separate the elements.

(ii) Example encoding: For an array A, encode its elements as in the previous part, and list those encodings with appropriate separators. For example, the array A with entries one, two, three, and four would have $\langle A \rangle =$ `[1,3,2,4]`.

(iii) The length of this encoding is $\Theta(n + e)$, where n is the number of elements in the array, and e is the total length of the encoding of the elements.

(iv) The corresponding language is

$$L_{sorted} = \{\langle A \rangle : A \text{ is a sorted array of non-negative integers.}\}.$$

(3 pts) (c) "Given an array S of integers, are all its elements distinct?"

Solution:

(3 pts) (d) "Is a given undirected graph complete?" *Hint*: Consider the adjacency matrix.

Solution:

(3 pts) (e) "Does a given C++ program halt when run on a given input?"

Solution:

7. **Deciding languages with DFAs.**

For each of the following languages, give a DFA that decides it. You may represent a DFA as a state-transition function or as a diagram, whichever you prefer.

(3 pts) (a) The language (over alphabet $\{a, b\}$) corresponding to the regular expression ab^* .

Solution:

(5 pts) (b) The language (over alphabet $\{a, b\}$) corresponding to the regular expression $b(aaa)^*b$.

Solution:

(5 pts) (c) The language of binary strings in which the number of 0s is divisible by 3 *and* the number of 1s is even.

For example, the strings ε, 000, 01010, and 11000 are in this language, while 1, 01, 001, 0011, and 000111 are not.

Solution:

(4 EC pts) (d) *Optional extra credit.*

Consider the following language:

$$L = \{x \in \{0,1\}^* : [x] \bmod 3 = 0\}.$$

where $[x] := \sum_{i=0}^{n} 2^i x_i$ for $x = x_0 x_1 \ldots x_n$. E.g., $[\varepsilon] = 0$ and $[01100000] = [011] = 6$. Note that the bits are given in 'reverse' order from what is typical (e.g., 6 in binary is usually written as 110).

Give a DFA that decides the language L.

Hint: Relate $[x_0 x_1 \ldots x_k]$ to $[x_0 x_1 \ldots x_{k-1}]$, x_k, and k. You can use the fact (which is easy to prove) that $2^{2m} \bmod 3 = 1$ and $2^{2m+1} \bmod 3 = 2$ for any non-negative integer m.

Solution:

(3 EC pts) 8. *Optional extra credit.*

Consider the alphabet $\Sigma = \{\texttt{(}, \texttt{)}\}$, which consists of the open- and close-parenthesis characters `(` and `)`. Say that a string $x \in \Sigma^*$ is *balanced* if x has an equal number of `(` and `)` characters, and every *prefix* of x has at least as many `(` characters as `)` ones.

For example, the empty string ε, `(()())`, and `((()))` are balanced, but `())(()` is not, because it fails the second condition (but not the first one).

Either give a DFA that decides the language of balanced strings over Σ, or rigorously prove that no such DFA exists.

Solution: