Theoretical Foundations of Computer Science

2021 Assignment

This assignment will not be submitted, but the questions in the assignment tests will be based directly on the questions below. Note that while some questions may be identical, the general idea is that the concepts covered in the questions are the same. In other words, you need to understand the answers to these questions, not just memorize them.

If you think that a question has multiple interpretations, you may want to prepare for all possible interpretations. If there is an obvious modification to a question, you may want to consider what the answer would be if that modification were made. If you need to make any assumptions in order to answer a question, be sure to state these if asked about that question and be ready to justify why they were needed. If you're unsure about a question, just do your best.

The assignment tests themselves will be vivas (individual verbal assessments) carried out via a platform such as Skype or Teams. More details will be available on Blackboard.

Computability Classification

For each of the following problems, classify it as being either Regular (Tier 1), Context-Free (Tier 2), Decidable (Tier 3), or higher (Tier 4+), and prove its membership of that category.

To prove membership, do the following:

- To show that something is Regular you should provide a DFA, NFA or RE that solves the problem.
- To show that something is Context-Free you should provide a proof that it is not Regular and should also provide a PDA or CFG that solves the problem.
- To show that something is Decidable you should provide a proof that it is not Context-Free and should also provide a decidable TM that solves the problem. If a pumping lemma proof is not appropriate, you may make an appropriate argument.
- If you believe that something is higher than Tier 3, prove this in an appropriate way. If you are not able to do this, consider providing something (such as a recognizer) if you can. If you are providing a reducibility proof, you may assume that A_{TM} is undecidable but will have to show anything further than that.

If you cannot prove or disprove membership formally, give a convincing argument. If you can't provide a full machine, provide enough to demonstrate that such a machine is possible. If you need to make assumptions, state these clearly. If you are not sure of a term, make sure that you look up a definition from a reliable source.

Complexity Classification

In addition, for each of the following problems, classify it as being a member of P, NP, NP-Complete and/or NP-Hard or none of these.

To show that a problem is in P you need to provide a machine (generally a TM but a non-deterministic PDA or DFA will do) that decides the problem and show that this completes in polynomial time. To show that something is in NP you can either provide a non-deterministic polynomial time decider or provide a deterministic decider that verifies an answer in polynomial time

To show that a problem is NP-Complete or NP-Hard your answer should involve a formal proof (either a direct proof or a reduction proof). If you are claiming equivalence to a meta-problem, you also need to provide a clear argument that the problem is equivalent to that problem. If you are using a reduction proof, you may assume that SAT and 3-SAT are NP-Complete.

You may make use of complexity classification theorems in the book that don't relate to specific meta-problems. If you cannot prove or disprove membership formally, giving a convincing argument can achieve part marks (or full marks if the argument is especially strong). If you need to make assumptions, state these clearly.

PROBLEM 1 – Simple Strings I

The string $01^x0^y1^z0^w$ where $x, y, z, w > 0$.

PROBLEM 2 – Simple Strings II

The string $01^x0^y1^x0^y$ where $x, y > 0$.

PROBLEM 3 – Simple Strings III

The string $01^x0^x1^y0^y$ where $x, y > 0$.

PROBLEM 4 – Combined Substrings

The language of binary strings that contain the sub-string 01^n0 and the sub-string 10^m1 where $0<n<m$.

PROBLEM 5 – Combined Substrings

The language of binary strings that contain either the sub-string 01^n0 or the sub-string 10^n1 where $0<n$.

PROBLEM 6 – Happy Cats

For L = {cat, dog, fish} find all strings with happy cats (which means that the string contains at least as many fish as cats and no dogs).

PROBLEM 7 – Happy Dogs

For L = {cat, dog, fish} find all strings with a happy dog (a dog is happy when it immediately follows a cat).

PROBLEM 8 – Counting Coins

A machine designed for counting coins is to have an additional circuit added. The existing machine will send a bit string representing each coin to the new circuit (11 is for \$2, 10 for \$1, 01 for 50c and 00 for 20c – the machine does not accept 10c or 5c coins). The new circuit is to check whether the machine receives more than twice as many \$1 and \$2 coins (combined) as \$50c coins in order to determine whether the size of the container for 50c coins needs to be decreased.

PROBLEM 9 – Skin in the Game

A player is studying the relative usage of his favourite Fortnite skins. He is particularly keen on Meowscles, Mandalorian and Deadpool. He has set up a bot to monitor Twitch Fortnite streams which reports whether any of those skins were used in that stream.

The bot will return three numbers, each corresponding to the number of players in that match that were spotted using the corresponding skin. So, if the bot returns 301 it means that three different players were spotted using Meowscles, none were spotted with Mandalorian and one was spotted wearing Deadpool.

While the bot may end up not seeing every player in each stream and may indeed scan a multiple match several times if it is streamed by different content creators, the player feels that this is still a representative sample. He plans to let the bot run over one week of streams and then see which one is the *least* popular. Your software should return the least popular skin, although you may use code to do so (e.g., 1 for Deadpool, etc).

PROBLEM 10 – Getting Bigger

A chip manufacturer is experiencing faults in their chipsets. Your task is to build a device to check that the increment is successful. The input will be the original input to the increment part of the circuit and the output from that circuit. For example, if the original input was A and the output was B (which is hopefully equal to an incremented A) then you will be given both A and B as inputs to your device. You may accept the digits of the inputs in any order but can only read each digit once. Your device will return accept if the output B is correct, given input A.

[Examiner’s note - While it would be entertaining to ask you to do this for the full 64-bit increment it is sufficient to look at the 8-bit increment. In practice, several 8-bit increment circuits are used in parallel or in sequence, so this isn’t unrealistic. Even looking at the 4-bit version will be sufficient, since the principle is the same.]

PROBLEM 11 – ASDF

Your input is a paragraph of English text. Accept if the number of times the phrase “I like trains” occurs is equal to the number of times the word “aargh” occurs (without quotes).

PROBLEM 12 – British Civilization

A You Tube content creator called the Spiffing Brit has asked your help with checking one of his exploits is working. He is testing yet another possible exploit in the game Sid Meier’s Civilization XIII.

Spiff has set up some bots to play Civ XIII and take a very specific series of actions with a Scout unit. The idea is that doing this then causes a nearby barbarian encampment to spawn an additional unit. This can be exploited using a proper setup that gives discounts on hiring barbarian units.

The output of the bots is somewhat complex, but if reduced to binary, the completion of the series of actions by the scout unit is shown by a 101 in the string and the spawn of a barbarian unit is shown by 001. The theory is that every time the scout finishes their actions, a barbarian unit will spawn quickly enough so that there are at most 4 binary digits between the two. If this happens every time, you are to accept the string.

Spiff will let the bots run many times, each time for an hour, which will then be input to your program as a binary string. Your software is to accept if his theory is correct for the given string.

PROBLEM 13 – Unending Code

A computing postdoc has invented a new programming language. They claim that the new language will successfully parse *any* text file input and create working code from it, thus eliminating debugging for all time.

Their supervisor has asked you to test the lexical parser for this language while she checks whether the compilation results of various files produce useful results. She has so far successfully compiled and run the lyrics of several of her favourite songs and Shakespeare's Sonnet 18, although the latter only printed some rude text.

You are to test whether the lexical parser truly does accept anything that is a finite text file.

PROBLEM 14 – The 1000

A rather complicated piece of software is supposed to be encoding a rather complicated biological theory. The idea is that the molecular structure of a compound (encoded as a binary string) is fed into the software, and it will accept if the string is a valid encoding of a compound belonging to a new family that the theory is based on and reject otherwise. This family are supposed to be the top 1000 structures meeting certain criteria.

The trouble is that encoding an existing molecular structure into a binary string takes a considerable amount of time and decoding it is just as bad. The research team has so far encoded 300 different molecular structures (including PG5, the largest known stable structure) which ends up being a binary string equating to the decimal number 10,372,843,723 (the number being exceptionally large because of its size). This was not accepted, which was seen as a success because no molecule this large should meet the required criteria.

The problem is that 250 of the other molecules HAVE been accepted, when the initial estimate was that only about half of them would be. The researchers now suspect that there is an error in the initial encoding that that is accepts more than the required 1000 structures.

Your team has been asked to write software that will test this software (and any future similar pieces of software) and confirm whether the software will indeed only return "accept" for at most 1000 structures.

You may assume that the current piece of software (and all future ones tested) will halt on all finite inputs.

PROBLEM 15 – The Infinite

The research team from Problem 14 have another piece of malfunctioning software relating to another project. This one is also accepting input that it is suspected it shouldn't. In this case there is no hard number of "acceptable" accepts but it is known to be finite, so you are asked to check if this machine (and any future ones tested) accept infinitely many inputs rather than some finite amount.

PROBLEM 16 – To Infinity and Beyond!

Your company has obtained a contract to be involved with the making of Toy Story IX and an argument has brought everything to a crashing halt. An AI has been trained to recognize "catchy" phrases and a small meme generator has been programmed to created phrases related to the upcoming movie that are likely to be popular, given the script of the movie itself.

The problem is that the owner insists that the very best phrase be used but that requires a complete listing. The company's IT team claims that the meme generator is likely to produce infinitely many phrases even given the relatively small set of initial combinations. Currently, the meme generator has produced over one million memes and is still producing new ones.

Your team has been asked to verify whether this is true or not. In fact, you've been asked to write some software to check whether the output of any future meme generator is finite for all inputs. You may assume that the meme generator will halt once it has finished listing all memes that are generated from its current input.

PROBLEM 17 – Pair Programming

Your team is tasked with writing some software that is going to be used to assign backup teams in possible hostile scenarios. You will be given a map (encoded as an undirected graph) with each vertex representing a held position and each edge representing a possible reinforcement path between held positions.

Your aim is to divide all held positions into pairs such that each member of the pair can reinforce the other. If the number of teams is odd, three teams will end up being part of a trio rather than a pair.

You are to list the chosen pairs if your software is able to do a proper assignment and reject if no such assignment is possible.

PROBLEM 18 – Garfield Code

A conspiracy theorist has decided that Garfield cartoons form a secret code. The idea is that each panel in each cartoon is assigned a 'character number', which is the number of main characters (Garfield, Odie, Jon, and Nermal) who are clearly visible in that panel. The character numbers then form a string, which is then translated into binary and hence into English.

A linguistics expert has pointed out that if such a code were to exist, certain groupings would naturally occur in each episode (a collection of panels on one topic, generally at least 18 panels long). In particular, the sequence 001, 013 and 102 should all appear in every episode.

The problem is to write a program that checks a string of 'character numbers' representing one episode to see if the 'code' is found in the episode.

PROBLEM 19 – Company Tours

You are to prepare software for a tour company. You will receive GPS coordinates of a series of potential paths for each tour (each of which can consist of multiple roads or road segments, starting and ending with a significant place), which will start at the company's headquarters and end at a given point such as the post office or town hall. Each significant place is a location where the tour will pause for a description and photos.

Your problem is to find a tour that starts at the HQ and passes along each of the roads given, ending at the designated point, and visiting every significant place. Since this is for sightseeing tours, you do not want to repeat a place. (You may assume that you can abstract the input of this problem to being a weighted, undirected graph with vertices representing significant places, and a series of edges representing paths between those places.)

Your solution should *accept* if such a tour is possible and have the vertices visited written to a designated tape in the order that they are visited by the tour.

PROBLEM 20 – Not-So Secret Santa

You are to prepare software to organize teams of office workers for the annual Unsecret Santa (because the other way isn't evil enough). The idea is to label office staff with a number from 1 to 10. Each staff member with a 1 will buy something edible, those with a 2 will buy something alcoholic, those with a 3 will purchase something useful in the home, and so on.

A group of up to ten (each with a different number) will then get together and each gets to pick which of the three presents they wish to take home with them (in order of decreasing rank, then seniority.) The difficulty is that the company wishes to maximise the potential angst by ensuring that each person in such a group should be friends with at least one other group member and preferably with as many as possible

You are given a list of staff members (by staff ID) and pairs of staff who are friends (again, but staff ID). The company assumes that friendship always goes in both directions. Because the company calculates "friendship" fairly loosely, every staff member is listed as having many friends, however no person has more than nine friends.

To arbitrarily make life harder, the sadistic management team decide that this will be handled by two separate teams. Your team is to handle the labelling of workers. In order for the other team to have the best chance, you are to ensure that no two friends have the same number.

PROBLEM 21 – Chain Connections

A franchise owner wants software to compute the best location for a series of chain stores in different cities. The idea is to place them in areas that are well-connected. You will be given a map (assume it's in the form of a graph) and you are asked to check for well-connected locations.

A location (a vertex in the graph) is considered well-connected at size *C* if it is connected (by an edge of the graph) to at least *C* other locations which are themselves connected to each other. Your job is, given a map and an integer *C*, to find if a map contains one or more spots that is well-connected at size *C*.

PROBLEM 22 – Package Delivery

You are given a weighted undirected graph that represents the roads and junctions in a municipality. You are to return a set of edges so that, starting at a location *L* that represents a company's main warehouse in the area (also given), every delivery point *P* in the city (also given, a subset of the junctions) is connected with that location by a path of edges between *L* and *P* so that the total of all of the edges chosen is minimal.

PROBLEM 24 – Pokémon Party

A new Pokémon game has been released. Part of this involves collecting creatures called Pokémon who each have an integer called Combat Power (CP). Another part of the game involves gathering Pokémon into battle groups so that they can compete in battles against the groups of other players.

The trick is that competitions have limits on how many Pokémon can be in a battle group. The total CP (the sum of CP of all Pokémon in the group) has a limit *L* and players fairly much always want to be exactly on the limit to maximise their chances of winning. There are various limits, such as 1500 or 750, depending on the competition.

Your task is to write some software that will help players choose teams. Your software is to take a list of Pokémon (each is given as an ID number (an integer with 3 decimal digits) and their combat power (an integer with between 2 and 4 decimal digits) and a limit L. Your program is to accept if there is some subset of these Pokémon whose CP sums to equal L (and the IDs of these Pokémon should be written on an output tape).

PROBLEM 25 – Same Game

Playing (not writing) the game Same Game.