THE UNIVERSITY OF BRITISH COLUMBIA
CPSC 203: FINAL EXAMINATION – December 11, 2019

Full Name: _____ CS Account: _____

Signature: _____ UBC Student #: |_|_|_|_|_|_|_|_|_|

Important notes about this examination

1. You have 150 minutes to complete this exam.
2. No notes or electronics of any kind are allowed.
3. Good luck!

Student Conduct during Examinations

1. Each examination candidate must be prepared to produce, upon the request of the invigilator or examiner, his or her UBCcard for identification.
2. Examination candidates are not permitted to ask questions of the examiners or invigilators, except in cases of supposed errors or ambiguities in examination questions, illegible or missing material, or the like.
3. No examination candidate shall be permitted to enter the examination room after the expiration of one-half hour from the scheduled starting time, or to leave during the first half hour of the examination. Should the examination run forty-five (45) minutes or less, no examination candidate shall be permitted to enter the examination room once the examination has begun.
4. Examination candidates must conduct themselves honestly and in accordance with established rules for a given examination, which will be articulated by the examiner or invigilator prior to the examination commencing. Should dishonest behaviour be observed by the examiner(s) or invigilator(s), pleas of accident or forgetfulness shall not be received.
5. Examination candidates suspected of any of the following, or any other similar practices, may be immediately dismissed from the examination by the examiner/invigilator, and may be subject to disciplinary action:
   i. speaking or communicating with other examination candidates, unless otherwise authorized;
   ii. purposely exposing written papers to the view of other examination candidates or imaging devices;
   iii. purposely viewing the written papers of other examination candidates;
   iv. using or having visible at the place of writing any books, papers or other memory aid devices other than those authorized by the examiner(s); and,
   v. using or operating electronic devices including but not limited to telephones, calculators, computers, or similar devices other than those authorized by the examiner(s)—(electronic devices other than those authorized by the examiner(s) must be completely powered down if present at the place of writing).
6. Examination candidates must not destroy or damage any examination material, must hand in all examination papers, and must not take any examination material from the examination room without permission of the examiner or invigilator.
7. Notwithstanding the above, for any mode of examination that does not fall into the traditional, paper-based method, examination candidates shall adhere to any special rules for conduct as established and articulated by the examiner.
8. Examination candidates must follow any additional examination rules or directions communicated by the examiner(s) or invigilator(s).

Please do not write in this space:

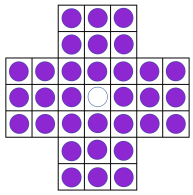
CPSC 203 2019W1: Final Exam

December 11, 2019

1 Who gets the marks? [1 mark] Write your 4 or 5 digit CSID here

2 Solo Noble [15-marks]

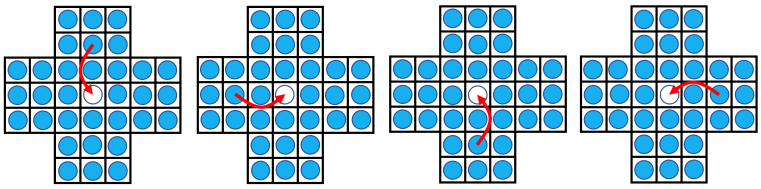
Solo Noble is a peg solitaire game played on a board. In this picture, the purple circles are pegs, and the blank circle is an empty hole into which a peg could be placed.



The object of the game is to find a sequence of moves that 1) removes all but one peg from the board, and 2) leaves the remaining peg in the center hole.

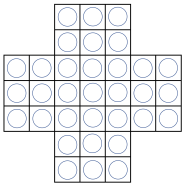
A move is a "jump" by one peg over one other, into an empty spot on the board. Jumps can be up/down or left/right, but not diagonal. When a peg is jumped over, it is removed from the board. Pegs may not jump over empty locations.

For example, given the starting board, there are exactly 4 different moves that can be made:



In this problem we will (partially) design a Solo Noble solver.

1. As a warm-up, sketch the state of the board after any one of the valid initial moves illustrated above.



2. The board is not a grid! Explain how you would represent the Solo Noble board in a program, including a scheme for naming the peg locations. If you use some kind of list or dictionary to represent the board, be sure to tell me what kind of data it contains. You may use sketches to answer this question, but you should do so in a way that helps me understand the code you would write. Place your answers in the relevant spaces below.

   (a) Board representation:

(b) Location description:

(c) Data description for each location:

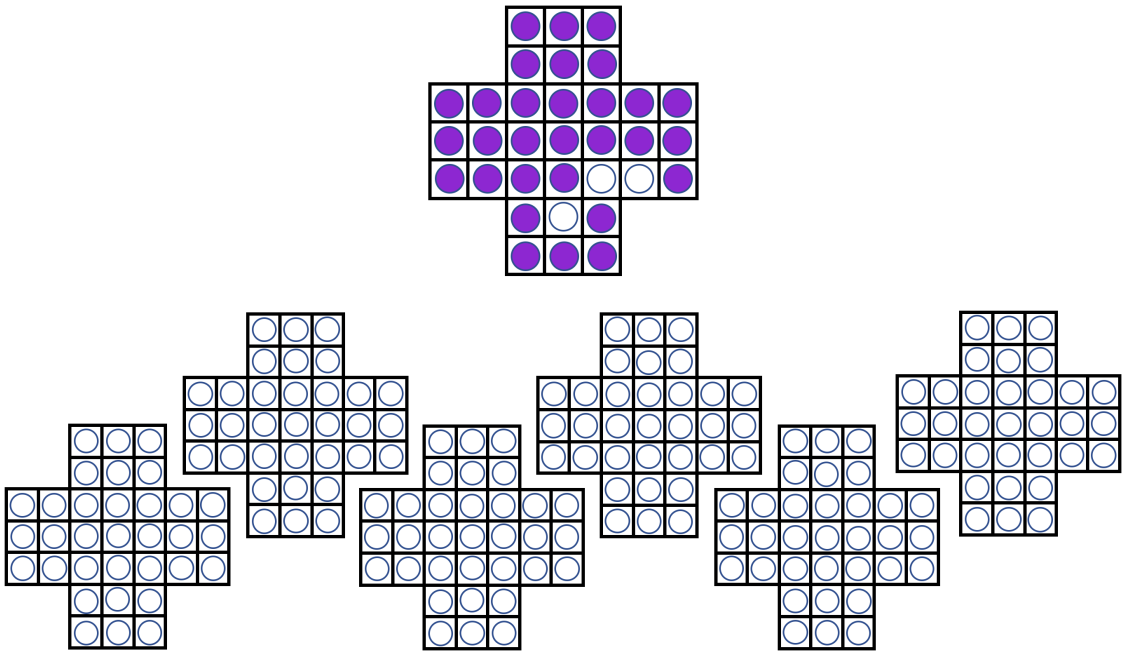
3. Use your location representation from the previous part to fill in the blanks:

   If an empty location is represented by _____ then the four pegs in locations _____, _____, _____, and _____, *might* be able to jump into it.

4. Why did we say *might* in the previous part? State two situations in which the locations you listed above would not produce a valid jump.

   -
   -

5. Suppose you have a partially solved board in the state shown below. Draw all board configurations that could result after one additional move. (You might not use all of the empty boards that we've given you.)



6. How many moves will there be in any winning solution?

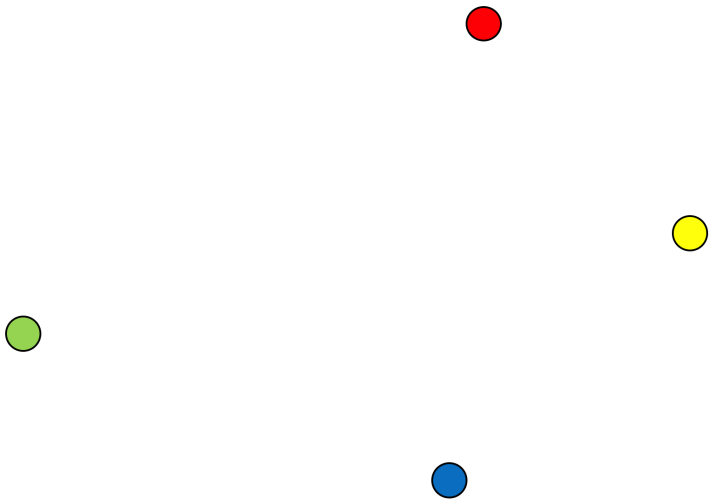
7. If you were asked to write a Python program to solve this puzzle, which example from CPSC203 would provide a good starting point? (One or two words is enough.)

3 Capture the Frog [25-marks]

The game of *Capture the Frog* is played by n players in a park. Each player has a *base*, at which they have stashed a colorful, beanbag frog. A player wins the game by collecting as many of the other players' frogs as possible, while protecting their own from capture.

In this problem, you will strategize for optimal game play, given the map of the park. We will make the simplifying assumption that all players run the same speed. Your base is the green base, and in our game, there are 3 other players whose bases are red, yellow, and blue.

1. The first step is to stake out your *home region*. Your home region consists of all places in the park from which you can reach your own base faster than you can reach any other. Sketch your home region on the map:



Which of the following did you adapt to solve the problem?

◯ DFS ◯ Voronoi Diagram ◯ BFS ◯ TSP ◯ Dijkstra's

◯ True ◯ False The home regions for all players will be the same size, no matter where they are placed in the park.

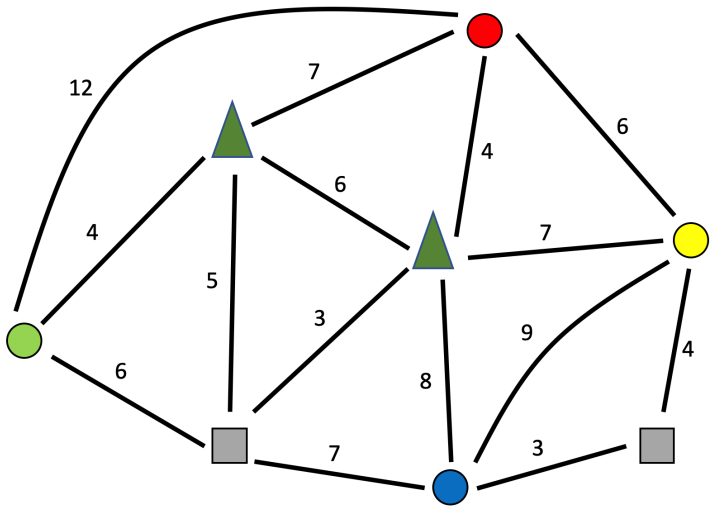
◯ True ◯ False The best algorithm to find the home regions of all the players requires the same amount of time, no matter how many players there are.

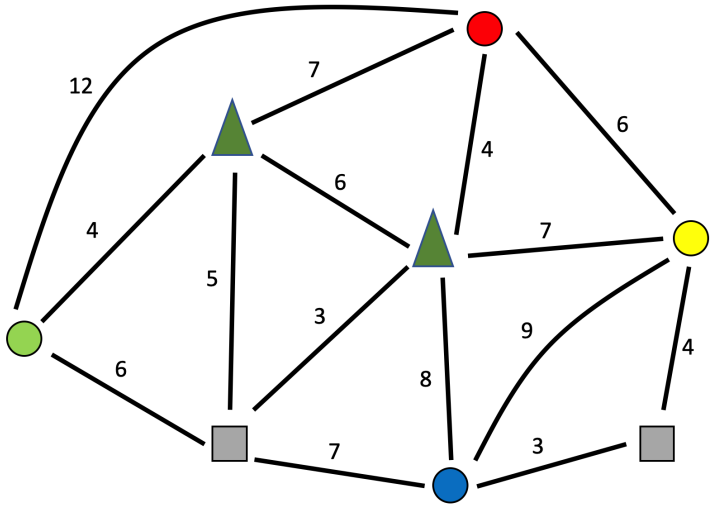
2. In addition to the bases, the park has trees (shown as triangles) and rocks (shown as squares). You would like to plan routes from your base to each of the other bases in a way that minimizes the time it takes you to get from your base to each other base. The diagram below is the map of the park, augmented with paths along which you may travel during the game. Every path is labelled with its travel time.

   The first copy of the map is for your scratch work. Place your answer on the second as follows:

   - ● to ● Label the route from your base to the red base with ∿∿∿
   - ● to ● Label the route from your base to the yellow base with -------
   - ● to ● Label the route from your base to the blue base with · · · · · · · ·

   Note: a) some paths may participate in more than one route, and b) it is possible that the shortest route from your base to another will pass through some other base.

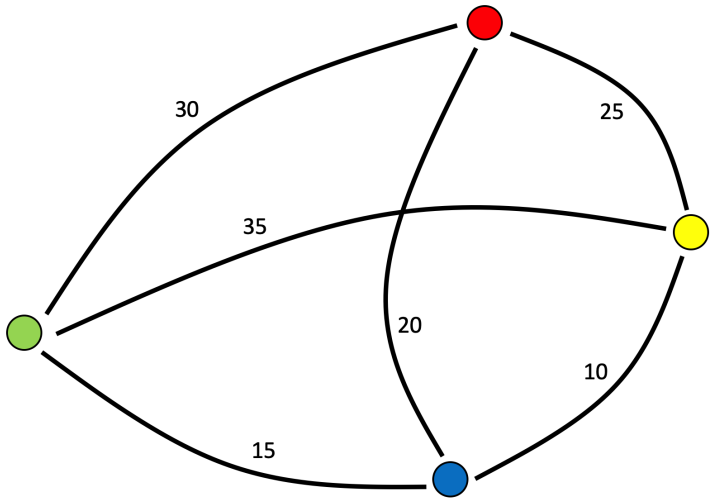


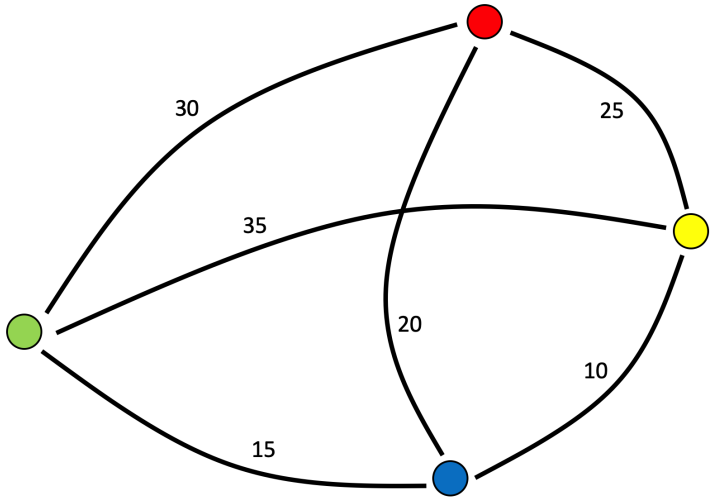


   Which of the following did you use to solve the problem?

   ○ DFS ○ Voronoi Diagram ○ BFS ○ TSP ○ Dijkstra's

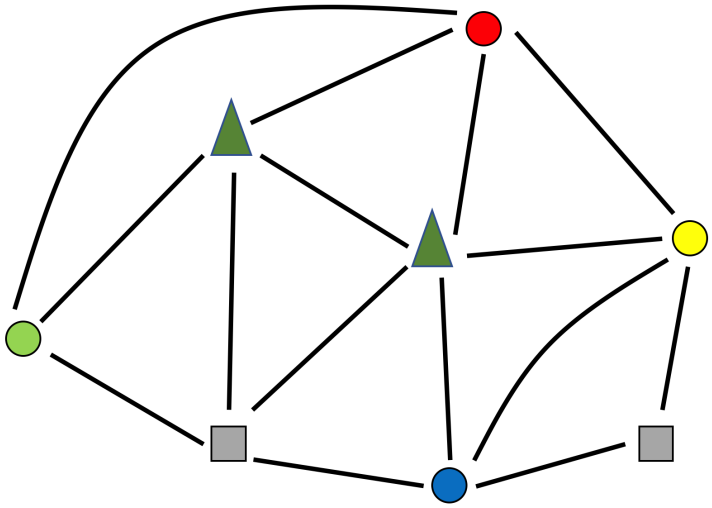
3. After some consideration, you decide that the best strategy is to think of the task as an instance of the Travelling Salesperson Problem on the graph below. Use the first graph for your scratchwork, and place your final answer on the second.

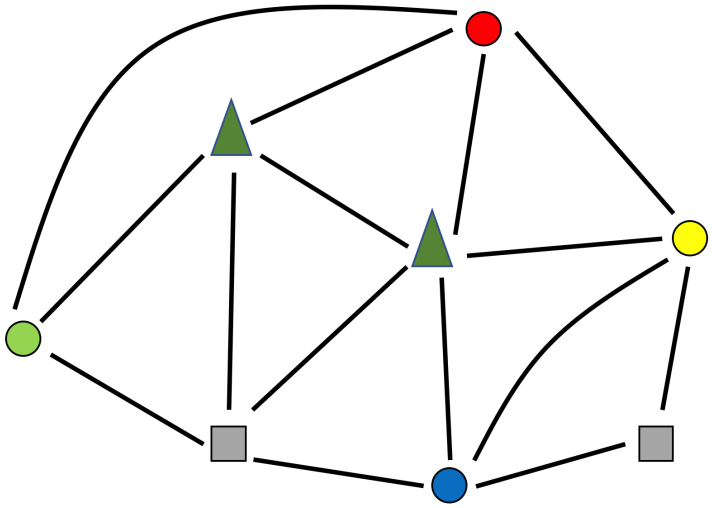




Briefly explain why this strategy would not be a good idea if there were 100 players, instead of 4.

4. After winning the game, you volunteer to clean up the park. To do so, you must start at your base and find a route through the park that follows every path exactly once. (The trees, rocks, and bases may be visited more than once.) Use the first graph for your scratchwork, and the second for your final answer. In your solution, label each edge with the order in which you travel over it, and circle the location at which you finish your work.





Though we did not discuss it, the solution to this problem is called an Eulerian Path. Which of the following most nearly matches your algorithm for finding the path?

◯ DFS ◯ Voronoi Diagram ◯ BFS ◯ TSP ◯ Dijkstra's

4 TFIDF [25-marks]

One of the ways in which search engines like google match the words in a query to webpages employs a statistical measure of word importance called "term frequency-inverse document frequency," or $tfidf$. In this problem we will explore and learn to compute that metric. While our treatment is a simplification, the underlying purpose and ideas are completely authentic.

1. Quickly read the following short passages of text, and for each, write a 3 word search-engine query for which you think the document would be a good match.

   (a) From powder to steeps to trees, from on-piste to off-piste to backcountry, the mountain resorts of B.C. give skiers and boarders access to world-class adventures and apres ski hangouts. With so many in our back yard it's all a matter of knowing where to go. Here are ten of the best. (Call this `docA`)

   (b) A chocolate chip cookie is a drop cookie that originated in the United States and features chocolate chips or chocolate morsels as its distinguishing ingredient. Circa 1938, Ruth Graves Wakefield added chopped up bits from a Nestlé semi-sweet chocolate bar into a cookie. (Call this `docB`)

   (c) Studies show that any act of altruism — a selfless act for others — is connected to positive physical and mental effects. According to the Cleveland Clinic, this includes lower blood pressure, increased self-esteem, less incidence of depression, lower stress levels, and even longer life and greater happiness. (Call this `docC`)

2. Essentially, search engines maintain a measure of the relevance of every web page to every possible query term. For us, the query will simply be a word (w), and a webpage will be a document (d) chosen from a collection of documents (D). In $tfidf$, the measurement comes from two distinct values:

   - $tf(w, d)$: the number of times word w appears in document d.
   - $idf(w, D)$: $\log \frac{\texttt{number of documents in } D}{\texttt{number of documents in } D \texttt{ containing } w}$.

   The importance of a word to a document is then the product of the two values: $tf(w, d) * idf(w, D)$.

   Answer the following questions using the three documents above for D. (Note that $\log 1 = 0$.)

   (a) What is $tf(\text{“}piste\text{”}, \texttt{docA})$?

   (b) What is $idf(\text{“}the\text{”}, D)$?

   (c) Which word has the highest $tfidf$ for `docB`?

3. In this part of the problem you will write the two functions `tf(w,d)` and `idf(w,D)`. You should assume that we have pre-processed the text so that each document is a list of words, and that the collection of documents is a list. Thus, the data structure storing the collection of documents is a list of lists of words. Finally, you may assume the existence of the log function: `log(float)->float`

   (a) Complete the function `tf(w:str, d:list(str))->int`.

```
def tf(w:str, d:list(str))->int:
''' return the number of times w appears in d '''
```

   (b) Complete the function `idf(w:str, D:list(list(str)))->float`.

```
def idf(w:str, D:list(list(str)))->float:
''' return the log of the (size of D/number of docs in which w appears. '''
```

4. The functions you wrote in the previous part of the problem will work fine, but it would be more efficient to pre-compute all the term frequencies and inverse document frequencies for every word in a collection of documents, so any query word could be handled by just looking up the pre-computed information.

   (a) Write a function `tfAll(D: list(list(str))->defaultdict(defaultdict(int))` that returns a dictionary containing the number of times word `w` appears in document `d` for all words and documents in `D`. Note that calling the function you wrote in the previous part is *not* the best way to accomplish this task, because it only handles one word at a time!

   (b) Write a function `idfAll(D:list(list(str))->defaultdict(float)` that returns a dictionary whose keys are words w, and whose values are the inverse document frequency of w in D.

This page intentionally left (almost) blank.
If you write answers here, you must CLEARLY indicate on this page what question they belong with AND on the problem's page that you have answers here.

This page intentionally left (almost) blank.
If you write answers here, you must CLEARLY indicate on this page what question they belong with AND on the problem's page that you have answers here.