# Assignment overview

In this assignment you practice how to apply some algorithms that we have studied to new problems.

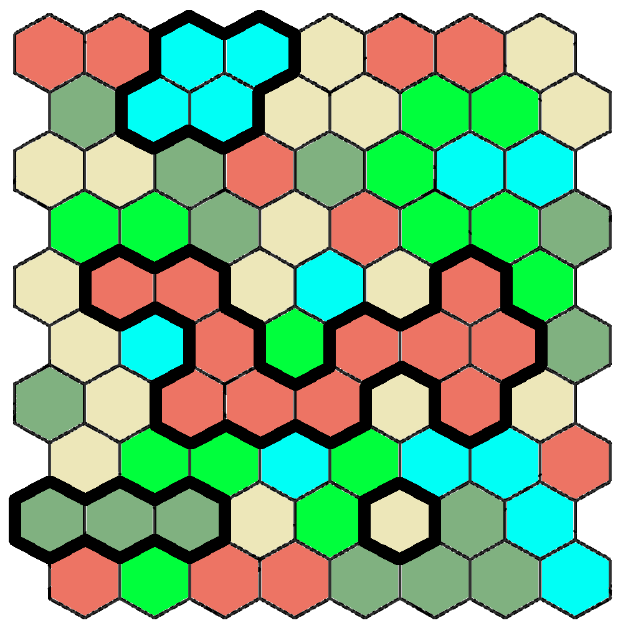
Write your answers to the questions here in this Word document and submit this document to Learning Hub when you are finished.

The due date for this assignment is Sunday 6pm as usual. The lab is worth 15 marks.

Note: There is no coding required on this assignment, but if you wish to write (and submit) any Java code for any of these problems I’ll certainly be happy to look at it.

# Problem 1:

A board game is played on a surface consisting of coloured hexagonal tiles. To set up a game, players select tiles randomly from a large supply of several colours, and lay them out in a grid. As a result, some tiles will be adjacent to other tiles of the same colour; this is called a *region*. Regions may be as small as one tile. The following diagram highlights just four of the many regions (shown with thick borders) on a sample map.



Given a map such as the one above, it is necessary to determine how many regions are on the map.

How can you model this problem as a graph?

1. What are the vertices?   
   **ANSWER:**

The vertices are the tiles

1. What are the edges? I.e., "There will be an edge between vertex u and v if and only if… **ANSWER:** There will be an edge if it is even and is :

(x, y +- 1), (x +- 1, y), (x + 1, y + 1), (x + 1, y – 1)

and is odd and:

(x - 1,y - 1), (x - 1, y + 1), (x, y +- 1), (x +- 1, y)

1. Choose an algorithm that we have studied and show how you would modify the algorithm to solve this problem.   
   **Note**: Copy the pseudocode algorithm from the book or lecture notes here, and make changes as needed. *Please highlight your changes in a different colour so they are easy to see.* Alternatively, write comments indicating where you have made changes.  
     
   **ANSWER:**

Algorithm Depth\_First\_Search(Graph G)

// Graph G = {V,E}

//initialize visited to false for all vertices

set all edges with the same color to True/1 and false/0 if they aren’t.

region\_count = 0

for each vertex v in V

if v has not been visited

dfs(v)

region\_count++

function dfs(Vertex v)

visit node v

for each vertex w in V adjacent to v

if w has not been visited and the edge of

(w, v) = 1

dfs(w)

# Problem 2:

The board game from the previous problem includes characters with traits such as *strength*, *wisdom*, *courage*, etc. that increase in value as the game progresses. At the beginning of the game the value of each trait is T1 = 1. In round 2 the value increases to T2 = 2. In all later rounds, the value is given by the following formula:

Write pseudocode for a *recursive* OR *non-recursive* function that uses *dynamic programming* to calculate Ti.

**ANSWER:**

T (n) {

Memo[1] = 1

Memo[2] = 2

For i = 2 to n do

Memo[i] = memo[I - 1] + (memo[I - 2] / 2)

Return memo[n]

}

# Problem 3:

Your wealthy Uncle Bob is giving you a fabulous graduation present when you complete your CST diploma: A free vacation on All World Airlines! (Vacation to be taken post-pandemic.) Uncle Bob’s condition is that you get to choose *any one city* that AWA flies to, as long as you can get there (one way) with at most $999. You’ll stay for two weeks, and then Uncle Bob will fly you home. Flying home is NOT part of the $999.

So you need to figure out a list of all the places that you can get to for $999 or less on AWA. Assume that the starting point is always Vancouver. Fortunately there is an online database that gives one-way ticket prices for all of AWA’s flights worldwide.

PART 1: Suppose you are only allowed to take *one flight*. Using one or two sentences, describe how you could produce a list of all the valid destinations.

**ANSWER:**

Check the ticket price to fly to that location is less than or equal to 999 for all flights. If it is less than or equal to $999, add it to the list.

PART 2: Now suppose you are allowed to use connecting flights, as long as the total cost of flights is at most $999.

How can you model this problem as a graph?

1. What are the vertices?  
   **ANSWER:** Each flight/price
2. What are the edges?   
   **ANSWER:** Edges are if the two flights can go together without exceeding the limit
3. What graph algorithm can you use to solve this problem? Describe how to set up the initial configuration for the problem, and how to answer the original question after the algorithm has completed execution.  
     
   **ANSWER:**

Using DFS that is modified to keep track of price, plug in Vancouver. It adds up the prices, if it doesn’t go over, add the node to the list.

# Problem 4:

Suppose G = (V, E) is a graph representing a communication network in which the edges are labelled either “slow” or “fast”. Using one of the algorithms discussed in class, briefly describe an algorithm (no need for pseudocode) for ﬁnding a path from vertex s to vertex t with the smallest number of slow edges. It does not matter how many fast edges there are in the minimum path as long as the number of slow edges is as small as possible.

**ANSWER:**

Floyd’s algorithm for finding the shortest path will be used, that has the Warshall parameter changed so that it keeps track of how many fast edges there are.

# Problem 5:

In the classic sliding number puzzle below, the goal is to arrange the numbered tiles in order from top to bottom, left to right. The only move you can make is to slide one tile into the empty space (after which the empty space is somewhere else).



Suppose you wish to write an algorithm to solve this puzzle using a backtracking strategy.

1. What would each state of the puzzle be?  
   **ANSWER:** A 2d array of the positions of each numbered tile.
2. How many choices would be available each time you make a decision? (If the answer is "it depends", what does it depend on?)  
   **ANSWER:** The empty square can move in one of the four directions, up, down, left, right if the square is in the internal. The empty square cannot move outside the board, so it can’t go up if it’s at the top row, can’t go down it’s at the bottom row, can’t go left if it’s at the left most column, and can’t go right if it’s at the right most column. Additionally, the move that puts the board state in the previous state is excluded.
3. How could you determine that there is no feasible "next move" from a given state, and that backtracking is required?  
   **ANSWER:** If the next move has the board in a state that already exists, and it is the only move, such as when its in the left corner, and it has just moved down, but moving right is a previous state in the puzzle. When it exceeds a user inputted max search depth.