CPSC 3620—Fall 2023
Assignment 4

Due: November 8, 2023 11:55pm

Written Questions (30%)

Please hand in the written portion of your assignment in Crowdmark.

All written questions are worth the same number of marks.

1. An undirected graph is called *bipartite* if the vertices can be partitioned into two sets S, $T \subseteq V$, $S \cap T = \emptyset$, and there are no edges connecting two vertices in S, and no edges connecting two vertices in T.

   Given a graph G, write an algorithm using depth-first search that determines if a graph is bipartite. What is the complexity of your algorithm?

2. Depth-first search can be used to detect cycles in undirected graphs (see lectures). Assuming an adjacency list representation of the graph, depth-first search has complexity $\Theta(n + m)$. Explain why the complexity of the cycle detection algorithm by depth-first search is actually $\Theta(n)$ regardless of what m is.

look at midterm Q6>

Programming Question (70%)

For this problem, you will write a program for the "Flip-Side" puzzle (see `http://www.jaapsch.net/puzzles/flipside.htm`). Your program will implement search using breadth-first search on the state-space graph of the game. Write your program in C++.

- Use a string of 10 characters (10 digits) to represent the states. Your program should ask the user for the initial state, and find the shortest solution to the end state "0123456789".
- Use an STL `map` to associate a distance (the number of moves from initial state) to each state. Note that the map can also be used to record whether a state has been seen—simply check if there is an entry corresponding to a given state.

- Use an STL `queue` to implement the breadth-first search.

Your program should ask the user for the initial state, and report to the user the smallest number of moves needed to reach the end state. Your program will be graded on correctness as well as readability.

Note: depending on the efficiency of your implementation, your program may take a few moments to finish (should not be more than 2 minutes of CPU time). You may wish to compile your program with optimization turned on. You can run a sample solution at:

```
cd /home/lib3620/as/as4
./flip-side
```

Note that it is acceptable for your solution to be slower than the sample solution.

You may find the following (incomplete) psuedocode useful:

```
push initial state to queue
set distance of initial state to 0
while (queue is not empty) {
  v = front of queue
  pop front of queue
  if v is the goal state, exit and report distance[v]
  for each state w generated from v in one move
    if w is not visited
      push w to queue
      distance[w] = distance[v] + 1
```

Bonus (10%)

In addition to reporting the number of moves to solve the puzzle, also report the actual moves required to solve the puzzle.

Submission

Put your source file(s) and a Makefile into a subdirectory named `as4`. Your program should compile correctly simply by typing `make`. Name your executable `flip-side`.

Select all source file(s) and Makefile and upload it to Moodle.