CS/ECE 374 P23

Jiawei Tang, Pengxu Zheng

TOTAL POINTS

30 / 100

QUESTION 1

1 23.A. 10 / 30

- **0 pts** Correct
- **30 pts** Wrong
- **22.5 pts** IDK
- **5 pts** No English Description
- **10 pts** Slower but correct algorithm
- **15 pts** Using BFS but wrong algorithm.
- ✓ **- 15 pts** **Error in graph construction, but got the general idea right.**
- ✓ **- 5 pts** **Missing or wrong run time analysis**
- **5 pts** Minor mistake

How do you determine if there is an obstacle in the path? It's possible to do it in O(n^4), but it's non-trivial. Otherwise, brute-force takes O(n^2) per move.

Also, not all moves are reversible, so you need a directed graph. (Suppose you start in the middle and move towards a wall. You can't go back.)

QUESTION 2

2 23.B. 0 / 30

- **0 pts** Correct
- ✓ **- 30 pts** **Wrong**
- **22.5 pts** IDK
- **5 pts** No English Description
- **10 pts** Slower but correct algorithm
- **15 pts** Some error, but got the general idea right.
- **5 pts** Missing or wrong run time analysis
- **5 pts** Minor mistake

What is I? You're not given any start state, so I can only assume it's all non-target states.

Also, the pseudocode only updates the distances for states adjacent to each target state?

QUESTION 3

3 23.C. 20 / 40

- **0 pts** Correct
- **40 pts** Wrong
- **30 pts** IDK
- **10 pts** No English Description
- **15 pts** Slower but correct algorithm
- ✓ **- 20 pts** **Some error, but got the general idea right.**
- **10 pts** Missing or wrong run time analysis
- **5 pts** Minor mistake

How would you determine which states are "stuck" though? Be more specific.

HW8 Solution

CS/ECE 374: Algorithms & Models of Computation, Spring 2019 Version: **1.0**

Submitted by:

- ≪**Pengxu Zheng**≫: ≪**pzheng5**≫
- ≪**Jiawei Tang**≫: ≪**jiaweit2**≫

23

Solution:

23.A.

We build an undirected graph G(V, E) to represent the state space of robot motions, where each vertex V is represented by the configuration of (R_i, B_i) such that R_i and B_i stand for the coordinates of robots. Each edge in G stands for a valid transition of states each time. The upper bounds for the total number of vertices is n^4 since there are total of n^2 possible states for each of the red and blue robots. Since each bot is capable of moving along 4 directions (up, down, left, and right), and the upper bound for the total number of the edges is $2 * 4n^4$. To find the minimum number required for one robot to reach the target, we run a BFS on G from the initial configuration (R_1, B_1) (which is given as a parameter) to the configuration that one of the robots reached the specified target grid block (if it exists), under the constraints of the must-stop conditions. An empty search tree will be generated and filled during BFS. The smallest number of steps taken, which is the number of nodes in the search tree, will be returned. The run time for BFS is $O(|V| + |E|)$ on a undirected graph, where |V| in this case is n^4 and |E| is $8n^4$. Thus, we conclude that our algorithm to find the minimum number of steps required to reach the target runs in $O(n^4)$ time.

23.B.

For part B, we build a graph $G'(V', E')$ that is similar to part A. Instead of using undirected edges, we use directed edges to link each valid transitions between reachable states. Thus, G' becomes a DAG. To determine the initial conditions of the robots such that the number of turns required to reach the target state is maximized, we use Topological sort to obtain the topological order of edges. We use two sets, T and I, which stand for target states and initial states, respectively. Since there will be multiple starting and target states, the longest path should reside in a path from a $i \in I$ to a $t \in T$. We also need to initialize an search table, dist[], to hold the distances for each $t \in T$ to a feasible location on the map; or in other words, possible paths from I to T. We start searching from T back to I.

for all $t \in T$, do:
    dist[t] = +inf
set the distance of the starting node to itself to 0
for all $t \in T$, do:
    for all $v \in Adj[t]$, do:
        if dist[v] < dist[t] + l(t, v), then update dist[v] = dist[t] + l(t, v).

Where l(t, v) stands for the length of edge t, v. Once the algorithm runs into $v \in T$, the After the loop ends, loop through dist[] and return the maximum number in dist[]. In terms of run time, since the dominant factor is the procedure of conducting topological sort that requires $O(|V'| + |E'|)$ $= O(n^2 * n^2 + 2n^2 * 2n^2) = O(n^4)$

23.C.

1 23.A. **10 / 30**

- **0 pts** Correct
- **30 pts** Wrong
- **22.5 pts** IDK
- **5 pts** No English Description
- **10 pts** Slower but correct algorithm
- **15 pts** Using BFS but wrong algorithm.

✓ - **15 pts** **Error in graph construction, but got the general idea right.**

✓ - **5 pts** **Missing or wrong run time analysis**

- **5 pts** Minor mistake

How do you determine if there is an obstacle in the path? It's possible to do it in O(n^4), but it's non-trivial. Otherwise, brute-force takes O(n^2) per move.

Also, not all moves are reversible, so you need a directed graph. (Suppose you start in the middle and move towards a wall. You can't go back.)

HW8 Solution

CS/ECE 374: Algorithms & Models of Computation, Spring 2019 Version: **1.0**

Submitted by:

- ≪**Pengxu Zheng**≫: ≪**pzheng5**≫
- ≪**Jiawei Tang**≫: ≪**jiaweit2**≫

23

Solution:

23.A.

We build an undirected graph G(V, E) to represent the state space of robot motions, where each vertex V is represented by the configuration of (R_i, B_i) such that R_i and B_i stand for the coordinates of robots. Each edge in G stands for a valid transition of states each time. The upper bounds for the total number of vertices is n^4 since there are total of n^2 possible states for each of the red and blue robots. Since each bot is capable of moving along 4 directions (up, down, left, and right), and the upper bound for the total number of the edges is $2 * 4n^4$. To find the minimum number required for one robot to reach the target, we run a BFS on G from the initial configuration (R_1, B_1) (which is given as a parameter) to the configuration that one of the robots reached the specified target grid block (if it exists), under the constraints of the must-stop conditions. An empty search tree will be generated and filled during BFS. The smallest number of steps taken, which is the number of nodes in the search tree, will be returned. The run time for BFS is $O(|V| + |E|)$ on a undirected graph, where |V| in this case is n^4 and |E| is $8n^4$. Thus, we conclude that our algorithm to find the minimum number of steps required to reach the target runs in $O(n^4)$ time.

23.B.

For part B, we build a graph $G'(V', E')$ that is similar to part A. Instead of using undirected edges, we use directed edges to link each valid transitions between reachable states. Thus, G' becomes a DAG. To determine the initial conditions of the robots such that the number of turns required to reach the target state is maximized, we use Topological sort to obtain the topological order of edges. We use two sets, T and I, which stand for target states and initial states, respectively. Since there will be multiple starting and target states, the longest path should reside in a path from a $i \in I$ to a $t \in T$. We also need to initialize an search table, dist[], to hold the distances for each $t \in T$ to a feasible location on the map; or in other words, possible paths from I to T. We start searching from T back to I.

for all $t \in T$, do:
    dist[t] = +inf
set the distance of the starting node to itself to 0
for all $t \in T$, do:
    for all $v \in Adj[t]$, do:
        if dist[v] < dist[t] + l(t, v), then update dist[v] = dist[t] + l(t, v).

Where l(t, v) stands for the length of edge t, v. Once the algorithm runs into $v \in T$, the After the loop ends, loop through dist[] and return the maximum number in dist[]. In terms of run time, since the dominant factor is the procedure of conducting topological sort that requires $O(|V'| + |E'|)$ $= O(n^2 * n^2 + 2n^2 * 2n^2) = O(n^4)$

23.C.

2 23.B. **0 / 30**

- **0 pts** Correct
- ✓ **- 30 pts** **Wrong**
- **- 22.5 pts** IDK
- **- 5 pts** No English Description
- **- 10 pts** Slower but correct algorithm
- **- 15 pts** Some error, but got the general idea right.
- **- 5 pts** Missing or wrong run time analysis
- **- 5 pts** Minor mistake

What is I? You're not given any start state, so I can only assume it's all non-target states.

Also, the pseudocode only updates the distances for states adjacent to each target state?

To count for stuck states, we use another set S to count for states that both robots ran into "corners" where its directions of movement are restricted by the obstacles or walls. Since the list of obstacles' locations are given, under the constraints of robots' directions, elements of S could be finitely determined. We use the same algorithm described in part B but change the set T to set S in order to first find the longest paths that lead to the stuck state of both robots. After that, merge the result of the stuck states and success states, and iterate through dist[] to return the maximum. For run time, since the basic structure of our algorithm remain unchanged, we conclude that "longest stuck" algorithm still runs in $O(n^4)$ time.

3 23.C. **20 / 40**

- **0 pts** Correct
- **40 pts** Wrong
- **30 pts** IDK
- **10 pts** No English Description
- **15 pts** Slower but correct algorithm

✓ **- 20 pts** **Some error, but got the general idea right.**

- **10 pts** Missing or wrong run time analysis
- **5 pts** Minor mistake

How would you determine which states are "stuck" though? Be more specific.