CSCI-GA.1170-001/002 Fundamental Algorithms November 28, 2019

Solutions to Problem 1 of Homework 10 (11 Points)

Name: jingshuai jiang (jj2903) *Due: 5 pm on Wednesday, November 27th*

Collaborators: NetID1, NetID2

An $n \times n$-grid is a graph G with n^2 vertices $\{(i,j) \mid 1 \leq i,j \leq n\}$ and edges such that two vertices u and v are connected if and only if they are at distance 1.

The game of Sokoban is formalized as follows:[1] Let $S = (V, E)$ be a subgraph of the $n \times n$-grid, i.e., S can be obtained by deleting all vertices of the grid not contained in V and the edges adjacent to them. Imagine that at the onset of the game, some vertices $I \subseteq V$ on subgrid S have a crate on them and that there is one special vertex $w \in V \setminus I$ where Waldo stands. Moreover, there is a set of target positions $T \subseteq V$ with $k := |T| = |I|$ for the crates. Waldo's goal is to move the crates in such a way that in the end there is a crate on each vertex in T.

At any point, Waldo can move freely about the subgrid vertices not covered by crates. He can move an adjacent crate by standing behind it and pushing by one vertex, provided there is no other crate in the way. For example, if Waldo is at $(i,j) \in V$, there is a crate at $(i+1,j) \in V$ and no crate at $(i+2,j)$, he can push the crate to $(i+2,j) \in V$; Waldo ends up at $(i+1,j)$ in such a case.

(a) (3 points) Show that the number of different states of this game is at most $\left(n^2\right)^{k+1}$.

Solution: There are k caters and one people. Then the man can have n^2 positions. The k caters should have $(n^2-1)\cdot(n^2-2)...\cdot(n^2-k)$ positions. Then the total state should be $n^2\cdot(n^2-1)\cdot(n^2-2)...\cdot(n^2-k) \leq \left(n^2\right)^{k+1}$ This equation will be achieved when $k=0$.

So the number of different states of this game is at most $\left(n^2\right)^{k+1}$. □

(b) (3 points) Describe a winning state of this game and argue that there are at most $n^2k!$ of them.

Solution: In this game, the wining state of this game will be : There is a crate on each vertex in T,this permutations will have $k!$. And the man can be in any other vertex, which is n^2-k.So the toal number of winning state should be $(n^2-k)k! \leq n^2k!$. □

(c) (5 points) Devise an $O\left(\left(n^2\right)^{k+1}\right)$-algorithm that on input $S = (V, E)$ figures out the fastest way for Waldo to solve this problem (i.e., to move the crates from their initial positions I to their target positions T) or reports that there is no solution.

Argue why your algorithm works and achieves the desired running time.

(**Hint**: Using one of the algorithms you have seen in class, explore a state graph corresponding to Sokoban, in which the vertices are the various states and two states are connected if and only if in one move of the game (either Waldo only or Waldo and a crate change positions) one can get from one to the other.)

[1] Feel free to watch the animation on Wikipedia.

Solution: I will use the BFS algorithm. BFS will find the minimal distance from the start point to any other vertices. And we can use this to achive this by turning this whole process into a BFS search. This whole process will being like this: 1. Think of this problem as finding the minimum distance from the initialized state to the winning state. 2. Get the initialize state of both waldo and crates and mark this state as the starting point. 3. all states $u.d = \infty$ and $u.\pi = nill$. 4. and all of the state(vertices) will have whilte,gray,black states as shown in our books. 5. use the BFS algorithm start from the starting point and keep looking for the winning state. IF we get the winning state, the whole process will be finished.

Here is the sudocode.

```
1     SOKOBAN(G, s)
2         s.color = gray,s.d = 0,s.π = nil, Q is empty
3         Enqueue(Q,s)
4         while Q is not empty
5             u ← Dequeue(Q)
6             for all of v adj to U
7                 if v.color = white
8                     v.color = gray
9                     if v is winningstate then return v
10                    v.d = u.d+1
11                    v.π = u
12                    Enqueue(Q,v)
13                u.color = black
14        return nil
```

Run time: Using BFS to traversing the whole graph will take O(m+n), in this algorithm we have $O((n^2)^{k+1})$ vertices, and each vertices will only have constant edges with the surrounding vertices. This means we have $O(c(n^2)^{k+1})$ edges. Then in this way the whole time will be $O((n^2)^{k+1})$. □

CSCI-GA.1170-001/002 Fundamental Algorithms November 28, 2019

Solutions to Problem 2 of Homework 10 (11 points)

Name: jingshuai jiang (jj2903) *Due: 5 pm on Wednesday, November 27th*

Collaborators: NetID1, NetID2

You are given a directed graph $G = (V, E)$ on n nodes and m edges, where the node set $V = A \cup B$ consists of two disjoint subsets A and B of sizes n_1 and n_2 (so $n = n_1 + n_2$). Nodes in A are "healthy", while nodes in B are "infected". Given a source $s \in A$, your goal is to compute the shortest distance from s to every other healthy node which can pass through *at most one* infected node (i.e., if the path from s to v contains at most one infected u, this is OK, but if it contains two or more, this path is not allowed when computing the shortest distance).

The problem is a bit tricky, so I will help you. Define the following *directed* graph $G' = (V', E')$ on $2n_1 + n_2$ nodes. The vertex set of V' of G' is $V' = A_1 \cup B \cup A_2$, where A_1 and A_2 are two copies of healthy nodes A. Two nodes in A_1 are connected in G' if and only if they are connected in G, and the same between two nodes in A_2. The nodes in B are more interesting. For every original incoming edge $(a, b) \in E$, where $a \in A$ and $b \in B$, we will put an edge (a_1, b) in E', where a_1 is the copy of a in A_1. Similarly, for every original outgoing edge $(b, a) \in E$, where $a \in A$ and $b \in B$, we will put an edge (b, a_2) in E', where a_2 is the copy of a in A_2.

(a) (4 points) Let n', m' be the number of vertices and edges in G'. Show that $n' \leq 2n$ and $m' \leq 2m$.

Solution: Since the vertex set of V' of G' is $V' = A_1 \cup B \cup A_2$. And A_2 is a copy of A. Then the number of vertices in G' will be $n_1 + n_2 + n_1 = n + n_1$. And apparently the number of vertices in A n_1 is smalller than or equal to n. Then $n + n_1 \leq n + n = 2n$

It is the same with the edges. The edges og G can be split into four parts$E_{aa}, E_{bb}, E_{ab}, E_{ba}$ and $E_{aa} + E_{bb} + E_{ab} + E_{ba} = m$ and $m' = E_{G'} = E_{aa} + E_{ab} + E_{ba'} + E_{a'a'} = m - E_{bb} + E_{aa} \leq 2m$ □

(b) (4 points) Recall our original problem of computing the required shortest distance in G from s to every other healthy node $a \in A$ which can pass through *at most one* infected node $b \in B$. Call this distance $a[dis]$. Let s_1 and s_2 be the two copies of s in G'. Using one "appropriate" BFS call on G', show how to compute the values $a[dis]$. Specifically, say what is the starting node (call it s') of your BFS call in G'. Also, after your BFS call computed shortest distances $v'.d$ from s' to v', for every $v' \in V'$, show how to compute the desired values $a[dis]$ for the problem at hand (i.e., write an explicit formula for $a[dis]$ using appropriate $v'.d$ values). Justify your algorithm.

Solution: The starting node s' should be s_1

Since there are no links in the new graph between nodes B. Then all the nodes of a' linked to this s' will all be satistied points. Then in this case we just need a BFS of all the node from

the starting point. And we will compare the distance of just using a to a and the distance of a to b to a' And this for every v as healthy nodes the $a[dis] = min(v.d, v'.d)$ □

(c) (3 points) Show that the running time of your procedure is $O(m + n)$.

Solution: By using BFS of a tree, which has at most 2n nodes and 2m edges, all the nodes will be added to the queue for only one time and all the edges will be visited for one time.The total running time will be $O(2m + 2n)$ which is $O(m + n)$ □

CSCI-GA.1170-001/002 Fundamental Algorithms November 28, 2019

Solutions to Problem 3 of Homework 10 (12 points)

Name: jingshuai jiang (jj2903) *Due: 5 pm on Wednesday, November 27th*

Collaborators: NetID1, NetID2

Assume you want to compute shortest distances $\delta(v)$ from a source s to all nodes v of a graph G. Normally, you would run BFS(s) and have $d(s) = \delta(s)$ at the end. Imagine that instead you run the following modification of DFS-Visit(v):

- You initialize $d(s) = 0$ and $d(v) = \infty$ for all $v \neq s$.
- For every *tree* edge (u, v) encountered by DFS-Visit, you set $d(v) = d(u) + 1$ (just like BFS).

(a) (1 point) What is the answer produced by this algorithm on a complete graph K_n, where $(u, v) \in E$ for all (u, v)?

Solution: The distance will be 0,1,2,...,n-1 □

(b) (2 points) Give an example of a graph G with at most $2n$ edges, and an ordering of its edges in the adjacency list, where the algorithm above would still produce the same (bogus) answer as in part (a)?

Solution:

□

(c) (2 points) Give an example of a graph G with $\Omega(n^2)$ edges and $\delta(v) < \infty$ for all v, and an ordering of its edges in the adjacency list, where the algorithm above would not only produce a correct answer for all n vertices v, but even produce exactly the same BFS tree as the BFS itself.

Solution: INSERT YOUR SOLUTION HERE □

(d) (4 points) Recall, a forward edge (u, v) connects u to some (not immediate) descendant v of u in the DFS forest. For each of the following assertions, either prove it is correct, or give a counter-example.

- If the algorithm above is correct on all nodes v, then DFS-Visit(s) did not encounter any forward edges.
- If the algorithm above is incorrect on at least one node v, then DFS-Visit(s) encountered at least one forward edge.

Solution: The first is True. Assume we have a forward edge and the algorithm is correct in all the node. Assume the distance from s to v_1 is d_1, the distance from s to v_2 is d_2. Since v_2 is a decentant of v_1 (not immediate) then we have $d_2 > d_1 + 1$. However we have this forward edge, which means we can write $d_2 = d_1 + 1$. Then this d_2 is wrong, then we have a contradict here. This means we do not encounter any forward edges.

□

(e) (3 points) Assume now we modify our procedure as follows: instead of setting $d(v) = d(u)+1$ only for tree edges (u, v) (color WHITE), we right away set $d(v) = \min(d(v), d(u) + 1)$ for every discovered edge (u, v) irrespective of the color of v. Give an example of a graph G, and an ordering of its edges in the adjacency list, where this algorithm is still wrong.

Solution:

□