CSCI-GA.1170-001/002 Fundamental Algorithms November 17, 2014

Solutions to Problem 1 of Homework 9 (18 Points)

Name: Sahil Goel *Due: Wednesday, November 19*

You have an undirected graph $G = (V, E)$ and two special nodes $r, d \in V$. At time 0, node r is republican, node d is democratic, while all the other nodes $v \notin \{r, d\}$ are initially "undecided". For every $i = 1, 2, 3, \ldots$, the following 2-stage "conversion" process is performed at time time i. At the first stage, all republicans at time $(i-1)$ look at all their neighboring nodes v which are still undecided, and convert those undecided nodes to become republican. Similarly, at the second stage, all democratic nodes at time $(i-1)$ look at all their neighboring nodes v which are still undecided by the end of the first stage above, and convert those undecided nodes to become democratic. The process is repeated until no new conversions can be made. For example, if G is a 5-cycle $1, 2, 3, 4, 5$ where $r = 1$, $d = 5$, after time 1 node 2 becomes republican and node 4 becomes democratic, and after time 2 the last remaining node 3 becomes republican (as republicans move first). On the other hand, if the initial democratic node was $d = 3$ instead, then already after step 1 nodes 2 and 5 become republican, and node 4 becomes democratic, and no step 2 is needed.

Assume each node v have a field $v.color$, where *red* means republican, *blue* means democratic, and *white* means undecided, so that, at time 0, $r.color = red$, $d.color = blue$, and all other nodes v have $v.color = white$.

(a) (5 points) Using two BFS calls, show how to properly fill the final color of each node.

Solution: The same procedure of BFS can be followed to fill the colors of nodes correctly if we call it two times.
Overall idea:
1. Initially make source as {R} and completely traverse the graph and calculate shortest distance of every node from republican. Turn every visited node into Red except Democratic node. Instead of filling the correct values of v.d in every node, fill it will negative of that distance value. After this step every node will have its v.d field already filled with value which is equal to negative of shortest distance from Republican Node
2. Now make source as {D} and completely traverse the graph. In this case we will consider nodes with negative distance and infinity distance as not visited yet. Now we will calculate the shortest distance of every node and check if it is smaller than abs(v.d). If it is strictly smaller we will change its color to Blue and update its v.d to shortest distance otherwise if it is greater or equal we will just make its v.d field positive so that we don't traverse it again and keep its color as red.

First BFS call
BFS(G,{R})
S=R
for each vertex u ∈ G.V - {S} {
u.color = WHITE
u.d = Infinity

u.π= NIL
}
End For
S.color=GRAY
S.d=0
S.π=NIL
Q={}
enqueue(Q,s)
while Q$\neq$ {}
u=dequeue(Q)
for each v $\in$ G.Adj[u]
if v.color == WHITE
v.d=abs(u.d) + 1
v.π = u
enqueue(q,v)
end for
u.color=RED
end BFS

Second BFS call
BFS(G,{D})
S=D
S.d=0
S.π=NIL
Q={}
enqueue(Q,S)
while Q$\neq$ {}
u=dequeue(Q)
for each v $\in$ G.Adj[u]
if v.d < 0
if(abs(u.d) + 1 <abs (v.d))
v.color=BLUE
v.d=abs(u.d) + 1
v.π = u
enqueue(q,v)
end for
end BFS

□

(b) (8 points) Show how speed up your procedure in part (a) by a factor of 2 (or more, depending

on your implementation) by directly modifying the BFS procedure given in the book. Namely, instead of computing distances from the root node, you are computing the final colors of each node, by essentially performing a *single*, appropriately modified BFS traversal of G. Please write pseudocode, as it is *very* similar to the standard BFS pseudocode, and is much easier to grade. But briefly explain your code.

Solution: We can speed up our algorithm in part (a) by a factor of 2 by just running BFS once with following changes
(a). Instead of initializing with only one source, initialize BFS with 2 sources, R and D by enqueuing R first and then D
Enqueue(Q,R)
Enqueue(Q,S)
(b). Change the color of the node to the color of node which is recently popped from the QUEUE while adding them to the QUEUE. That is, if we have popped a red color node from Q, we will color all its adjacent nodes to the color Red and then add them to Q

```
BFS(G,{R},{D})
S1=R, S2=D
for each vertex u ∈ G.V - {S1,S2} {
u.color = WHITE
u.d = Infinity
u.π= NIL
}
End For
S1.color=RED
S1.d=0
S1.π=NIL
S2.color=BLUE
S2.d=0
S2.π=NIL

Q={}
enqueue(Q,S1)
enqueue(Q,S2)
while Q≠ {}
u=dequeue(Q)
for each v ∈ G.Adj[u]
if v.color == WHITE
v.color=u.color v.d=u.d + 1
v.π = u
enqueue(q,v)
end for
end BFS
```

□

(c) (5 points) Now assume that at time 0 more than one node could be republican or democratic. Namely, you are given as inputs some disjoint subsets R and D of V, where nodes in R are initially republican and nodes in D are initially democratic, but otherwise the conversion process is the same. For concreteness, assume $|R| = |D| = t$ for some $t \geq 1$ (so that parts (a) and (b) correspond to $t = 1$). Show how to generalize your solutions in parts (a) and (b) to this more general setting. Given parts (a) and (b) took time $O(|V| + |E|)$ (with different constants), how long would their modifications take as a function of t, $|V|$, $|E|$? Which procedure gives a faster solution?

Solution: We can generalize the second solution for any number of nodes
Instead of enqueuing only 2 nodes initially we can enqueue t nodes of Republican first and then t nodes of Democratic and follow the same algorithm as of second part

BFS(G,$\{R_1, R_2, \ldots, R_t\}, \{D_1, D_2, \ldots, D_t\}$)
S1=$\{R_1, R_2, \ldots, R_t\}$
$S2 = \{D_1, D_2, \ldots, D_t\}$
for each vertex u ∈ G.V - {S1,S2} {
u.color = WHITE
u.d = Infinity
u.π= NIL
}
End For
Q={}
For each u in S1
u.color=RED
u.d=0
u.π=NIL
Enqueue(Q,u)
end FOR
For each u in S2
u.color=BLUE
u.d=0
u.π=NIL
Enqueue(Q,u)
end FOR
while Q$\neq$ {}
u=dequeue(Q)
for each v ∈ G.Adj[u]
if v.color == WHITE
v.color=u.color v.d=(u.d) + 1
v.π = u

enqueue(q,v)
end for
end BFS

Analyzing the time complexity for both the solutions
For first algorithm
We will have to traverse the graph with 2*t BFS calls, once for every initial democratic and republican node. Hence total complexity in that case will be $O(t * (|V| + |E|)$
For second algorithm
We will just have to traverse the graph once and the time complexity will be $O(|V| + |E|)$
□

CSCI-GA.1170-001/002 Fundamental Algorithms November 17, 2014

Solutions to Problem 2 of Homework 9 (5 Points)

Name: Sahil Goel *Due: Wednesday, November 19*

Consider an $n \times n$ chessboard. In one move, a knight can go from position (i, j) to (k, ℓ) for $1 \leq i, j, k, \ell \leq n$ if either $|k - i| = 1$ and $|j - \ell| = 2$ or $|k - i| = 2$ and $|j - \ell| = 1$. However, a knight is not allowed to go to a square that is already occupied by a piece of the same color. You are given a starting position (s_x, s_y) and a desired final position (f_x, f_y) of a black knight and an array $B[1 \ldots n][1 \ldots n]$ such that $B[i][j] = 1$ if (i, j) is occupied by a black piece, and 0, otherwise. Give an $O(n^2)$ algorithm to find the smallest number of moves needed for the knight to reach from the starting position to the final position.

Solution: This solution can be solved with BFS with following changes in the algorithm
We will make n^2 nodes. Every node will represent a coordinate on the board(i,j).
We can also have an n*n boolean matrix Check[n][n] which will have Check(i,j)=0 if its not visited and Check(i,j)=1 if it has been visited once. Initially all the values will be 0.
Instead of adding nodes from adjacency list, we will enqueue nodes, that are reachable from the current position and not yet visited. For example if current and initial position is (3,3), we will add to queue (4,5), (4,1), (2,5), (2,1), (5,4), (5,2), (1,4), (1,2).
Algorithm

```
BFS(G,S,F)
For i = 1 to n
For j = 1 to n
Check[i][j]=0
end FOR
end FOR
Q={}
Enqueue(Q,S)
Check[S.x][S.y]=1
while Q≠ {}
u=dequeue(Q)
For each possible position (i,j) which can be visited from (u.x,u.y) and check[i][j]=0
v= new node(i,j)
v.d=u.d+1
if(i==F.x and j==F.y)
print "v.d"
exit
else
enqueue(q,v)
end for
check[x][y]=1
end BFS
```

Since there are maximum n^2 positions on the chess board, the maximum possible states are n^2 and hence if each position is visited once before finally finding the final position, the total worst case complexity will be $O(n^2)$

□

CSCI-GA.1170-001/002 Fundamental Algorithms November 17, 2014

Solutions to Problem 3 of Homework 9 (6 points)

Name: Sahil Goel *Due: Wednesday, November 19*

An undirected graph is said to be connected if there is a path between any two vertices in the graph. Given a connected undirected graph $G = (V, E)$, where $V = \{1, \ldots, n\}$, give an algorithm that runs in time $O(|V| + |E|)$ and finds a permutation $\pi : [n] \mapsto [n]$ such that the subgraph of G induced by the vertices $\{\pi(1), \ldots, \pi(i)\}$ is connected for any $i \leq n$. Which of BFS or DFS gives a better algorithm for this problem?

Solution: This problem can be solved by simply traversing through either of the two DFS or BFS. Both will take $O(|V| + |E|)$ time.
Final result will be stored in arr[n].
DFS(G)
i=1; create new array arr of size n
For all u ∈ v color[u] = white
parent[u] = nil
end for
for all u belonging to v (u ∈ v)
if(color[u] = white)
DFS visit(G,u)
}

DFS-Visit(G,u)
{
color[u] = gray
arr[i++]=u
for all v∈ adj[u]
{
if color[v] = white
parent[v] = u
DFS visit(G,v) }
Color[u]=black

for i=1 to n
print arr[i]
end for
}

Proof by induction
for i=1, initially there will be only one node and it will be connected
Now suppose the graph is connected till i=k-1 and we are adding k^{th} node then we know that we have reached k^{th} node and we must have reached it from adjacency lists of one of the previous k-1

nodes. Since previous k-1 nodes were connected and there is a path to kth node from one of those k-1 nodes(say q), then therefore there will be path from every node to kth node via q. Hence the complete graph till k will be connected. Similar will be the argument for BFS traversal.Therefore any of those will give the correct solution in time $O(|V| + |E|)$

□

CSCI-GA.1170-001/002 Fundamental Algorithms November 17, 2014

Solutions to Problem 4 of Homework 9 (8 points)

Name: Sahil Goel *Due: Wednesday, November 19*

The class teacher of a kindergarten class wishes to divide the class of n children into two sections. She knows that some students pairs of students are friends with each other, and she wants to try to split the two sections in such a way that in each section all students are friends of each other. Can you help her find an efficient algorithm to form the two sections given as input n, and m statements of the form 'i and j are friends with each other'. What is the running time of your algorithm?

(**Hint**: Assume that the first student goes into the first section. Which section should the students who are friends of the first student go to? Which section should those that are not his friends go to? Try to carefully form a graph and use BFS to solve this problem.)

Solution: Initially we will form a graph of students(each node represent a student) in which edge between node i and node j will represent that i and j are friends. For each node we will represent a field section which will initially be 0 and later on will be either 1 or 2 depending on student's allotted section. We will represent the graph as Adjacency Matrix A. So if i and j are friends then A[i][j]=A[j][i]=1 else A[i][j]=A[j][i]=0

Overall Idea of the algorithm:
1. Initially add first student to section 1, add all his friends to the queue.
2. Now start popping from the queue one by one and for each node(i) such that node(i).section=0 do
(a). Check if this node is friends with all the other nodes in section 1 i.e. A[i][k]=1 where k $\in$ section 1 students. If yes add this node to section 1 otherwise
(b). Check if this node is friends with all other nodes in section 2 i.e. A[i][k]=1 where k $\in$ section 2 students. If yes add this node to section 2 otherwise exit and print "Not possible"

3. Check if queue is empty. If queue is empty and there still nodes to whom section is not assigned add them to queue otherwise exit.
Algorithm

```
SplitStudents
{
For all students s in Class
create node (s)
s.section = 0
end For

Initialize A[1..n][1..n]={0};
For all m statements
if ( i and j are friends)
A[i][j]=A[j][i]=1
end For
```

```
S ={1}
S.section=1 (% Adding first student to 1st section)
Q={}
Enqueue(Q,S)
While(Q!={})
{
u = Dequeue (Q);
For each i ∈ G.Adj[u]
if(i.section==0)
if(i is friends with all of the students in section 1) i.e. A[i][k]==1 for all k where k ∈ section 1
i.section=1
else if(A[i][k]==1) for all k where k ∈ section 2
i.section=2
else
{
print("Not Possible")
exit;
}
enqueue(Q,v)
end For
if(Q=={})
If(There are nodes j such that j.section=0)
Enqueue(Q,j)
else exit
end if
}
```

Time Complexity of the solution will be $O(n^2)$ where n is number of students because for each student when we are assigning the section we are checking its compatibility with all the students already in section (1) and (2). Hence total complexity will be $O(n^2)$.

□

CSCI-GA.1170-001/002 Fundamental Algorithms November 17, 2014

Solutions to Problem 5 of Homework 9 (8 points)

Name: Sahil Goel *Due: Wednesday, November 19*

(a) (4 points) Explain how a vertex u of a directed graph can end up in a depth-first tree containing only u, although u has both incoming and outgoing edges.

Solution: It can happen because all the nodes v which are reachable from u (i.e. u $->$ v is an edge) may have already been traversed by some other node. Example:
Suppose this is the adjacency list
1 $->$ 2, 3, 4
5 $->$ 2,3
6 $->$ 5
u=5
In this case, Suppose we start the traversal from 1, we will visit all the nodes which are reachable from 1 that are 2,3,4,5. Now when we start with 5, and check if 2 and 3 are visited. They are already visited so there will be only 5 in its DFS tree.

□

(b) (4 points) Assume u is part of some directed cycle in G. Can u still end up all by itself in the depth-first forest of G? Justify your answer.
(**Hint**: Recall the White Path Theorem.)

Solution: No it isn't possible. There are 2 possible cases
Case 1
Suppose we have traverse all the nodes v(before processing u) such that u$->$v is an edge. Now since u is a part of cyclic graph G, there will be some v for which there will a path from v to u and hence it must have been traversed too. Hence it won't be the only one in its DFS Tree
Case 2
Suppose now we don't traversed any one or more of the nodes v which have in edges from u. In this case we know that there will be at least v in u's DFS tree. Hence it won't be the only node.

□