Deepanshu 2019CS50427

COL106 : Nov-26 (Graph-14)

Page :
Date :

Ans-1 ⇒ We know that $\sum_{all\ v \in V} deg(v) = 2|E|$ since every edge contributes twice in the summation (each edge is counted twice, one by each vertex on which it is incident)

Now since the summation is even, number of odd values contributing in the summation should be even.

Hence, there is an even number of odd-degree vertices in any undirected graph.

Hence Proved

Ans-2 ⇒ (a) Matrix to List

Algorithm matrixToList(Matrix m, n) [size of matrix = n×n]

```
list vertices[n]        // initialise the vertex array
for(i ← 0 to n)
    vertices[i] ← i

for(i ← 0 to n)
    for(j ← 0 to n)
        if(m[i][j] = 1)     // edge found
            vertices[i].append(j)   // appending in the list.
return vertices
```

- We are assuming that nodes are represented as numbers.
- Initialising the list takes $O(n)$ time as we are putting vertex index in list.
- We have to traverse through each entry of array which will take $O(n^2)$ time.

Page:
Date:

Deepanshu 2019CSS0427

- Assuming that appending to a list takes constant time (which can be achieved by maintaining tail pointer in the implementation of list).

Total time taken will be $O(n^2 + n)$ which is $O(n^2)$

(b) List to matrix

We will go through each and every edge in the list corresponding to each vertex.

```
Algorithm listToMatrix(List l, n)        // n ← No. of vertices in list
    Matrix m[n][n]                       // Initialise the matrix.
    for(i ← 0 to n)
        for(j ← 0 to n)
            m[i][j] = 0                  // Make each entry as 0
    for(i ← 0 to n)                      // Visiting each vertex
        v ← l[i]
        for each x in list               // Edge corresponding to v
            m[v][x] = 1                  // [can also be m[u][v] = 1, depending on how we define (esp. for directed)]
    return m
```

- Visiting each edge will take $O(|E|)$ time.
- Declaring the matrix and making each entry as 0 takes $O(n^2)$ time.
- Total time taken is $O(n^2 + |E|) = O(n^2)$ because $|E|$ can be at max $\frac{n(n-1)}{2}$ assuming no multi-edge and no self loops.
- Time take is hence $O(n^2)$.

Page :
Date :

Deepanshu 2019CS50427

Ans-3 ⇒ Algorithm reverse(List l, n) ← size of list l

```
List l' = list[n]

for(i ← 0 to n)        // Adding vertices to new list
    l'[i] = l[i]

for(i ← 0 to n)
    v ← l[i]
    for each x in list // ■ edge from v to x
        l'[x].append(v) // in l', edge from x to v
return l'
```

- Assuming that appending takes constant time, above algorithm takes time proportional to number of edges as we are looking at each edge (exactly once) and appending.
- Total time = $O(n + |E|)$ where n is number of vertices.

Ans-4 ⇒ Plan (discussed in flipped lecture)

- We will carry out BFS. While doing BFS, if two nodes at the same level are connected directly by an edge, then graph would have a cycle of odd length which implies graph is not bipartite.
- If there is no such ~~level~~ edge at any level, then we can come up with 2 disjoint subsets of V with alternate levels ~~merged~~ taken together.

Note :- Here, level refers to the level number while doing BFS.

Page :
Date :

Deepanshu 2019CS50427

Algorithm checkBipartite (G, s)

```
for each vertex u ∈ V[G] - [s]
    colour [u] ← white
    d[u] ← ∞
    π[u] ← NIL
```

} Init all vertices

```
colour[s] ← gray
d[s] ← 0
π[s] ← NIL
Q ← [s]
```

} Init BFS with source s.

```
while Q ≠ ∅ do
    u ← head[Q]
    for each v ∈ adj[u] do
        if colour[v] = white then
            colour[v] ← gray
            d[v] ← d[u] + 1
            π[v] = u
            Enqueue(Q, v)
    for each v ∈ adj[u] do
        if colour[v] = gray and d[u] = d[v]
            return false
    Dequeue[Q]
    colour[u] ← black
return true
```

[u & v are in same level and are directly connected by an edge]

Page :
Date :

Deepanshu 2019CS50427

We claim that graph is not bipartite if two vertices in same level are directly connected.

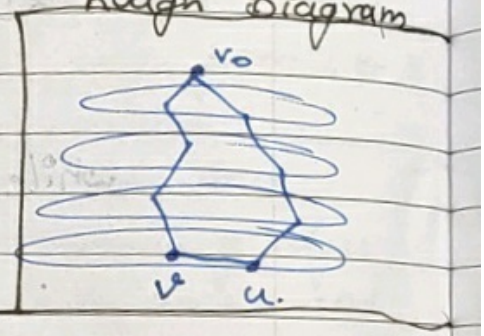
Proof by implication

Let v and u be at same level in BFS Tree.
Let v_0 be the nearest common ancestor. Note that they will always have a common ancester because they are vertices of tree (Bfs Tree).

Let v_0 be k levels above v and u.

Now since v and u are already connected by an edge, a cycle of length $2k+1$ is formed.

Rough Diagram



And since a cycle of odd length is found, graph is not bipartite.

If this is not the case, then graph formed is bipartite.

Proof by construction

We have to come up with 2 sets (call them A and B) such that $A \cap B = \phi$, $A+B = V$ and there is no edge between vertices in A (same for B). Define A and B as follows -

$A = \{ v : d[v]$ from s is odd$\}$.
$B = \{ v : d[v]$ from s is even$\}$.
Since we came up with 2 such sets, graph is bipartite.

Deepanshu 2019CS50427

Page :
Date :

Ans5=> **Proof by induction**

P(n): Any undirected graph on n nodes has at least n-1 edges.

Base Case: n = 2

Graph has (at least) 2-1 = 1 edge. P(2) is true.

Induction Hypothesis

Assume P(n) is true for all $n \leq k$

For n = k+1

Pick any vertex v and remove it. We can have following 2 cases:

Case 1: Graph still remains connected.

Note that deg(v) ≥ 1 [as original graph is connected].

Apply induction hypothesis on new graph, k nodes implies k-1 edges.

Hence our original graph has at least k-1 + deg(v) edges

Hence total edges is at least k.

Case 2: Graph now has k connected components.

Removing v made k different connected components. Hence, deg(v) $\geq k$.

Let each component has number of vertices $n_1, n_2 \dots n_k$.

Apply induction hypothesis to get number of vertices for new graph as $(n_1-1)+(n_2-1) \dots (n_k-1)$

at least $(n_1+n_2 \dots n_k) - k$

at least $n - k$ [$n_1+n_2 \dots n_k = n$ = total vertices].

Total edges are at least $n-k+deg(v) \geq n$

Hence P(k+1) is true whenever P(1) ... P(k) is true.

By principle of mathematical induction, P(n) is true for all n.

Hence Proved.

Deepanshu 2019CS50427

Page :
Date :

Ans=>

```
1  Algorithm allSource(list l, n)
2
3      boolean source[n]   // initialise boolean array
4      for(i<-0 to n)
5          source[i] = true
6
7      for(i<-0 to n)
8          for each vertex x in list   // if vertex is
9              source[x] = false       in list of any vertex,
10                                     make its source
11     list sourceVertices;            entry as false.
12     for(i<-0 to n)
13         if(source[i])
14             sourceVertices.append(l[i])
15     return sourceVertices
```

Ans=>

Running time.

- We first initialise source array by making each entry as true. This takes O(|V|) time.
- Now we visit each edge and marks its corresponding entry as false. This takes O(|E|) time.
- In the end, we return another list containing all source vertices. Building this list takes O(|V|) time in worst case.

Total time = O(|V|+|E|)

Page :
Date :

Deepanshu 2019CS50427

Proof of correctness

Proof by implication

- Pick any vertex v from the list sourceVertices.
- By line 13, we know that its source array value is true.
- From line 7 & 8, we can say that every vertex going in this loop has to have its source value as false.
- This means that v did not go inside that loop which means that v is not in the linked list of any vertex.
- This means that there is no edge going into it which implies that v is a sourceVertex.

This shows that every value present in the returned list is a sourceVertex.

Hence Proved.

Deepanshu 2019CS50427

(b) Running the algorithm for given directed graph.

We will be showing the entries of source array and finally the array "sourceVertices".

Array just after line 5 (after initialisation).

T	T	T	T	T	T	T	T	T	T	T
A	B	C	D	E	F	G	H	I	J	K

After successive iteration of outer loop (line 7) the source array is as follows.

Iteration 1:

T	T	T	F	T	T	T	T	T	T	T
A	B	C	D	E	F	G	H	I	J	K

Iteration 2:

T	T	F	F	F	T	T	T	T	T	T
A	B	C	D	E	F	G	H	I	J	K

Iteration 3:

T	T	F	F	F	F	T	T	T	T	T
A	B	C	D	E	F	G	H	I	J	K

Iteration 4:

T	T	F	F	F	F	T	T	T	T	T
A	B	C	D	E	F	G	H	I	J	K

Iteration 5:

F	T	F	F	F	F	T	T	T	T	T
A	B	C	D	E	F	G	H	I	J	K

Iteration 6:

F	T	F	F	F	F	T	T	F	F	T
A	B	C	D	E	F	G	H	I	J	K

Page :
Date :

Deepanshu 2019CS50427

Iteration 7:

A	B	C	D	E	F	G	H	I	J	K
f	T	f	f	f	f	T	T	f	f	f

Iteration 8:

A	B	C	D	E	F	G	H	I	J	K
f	T	f	f	f	f	f	T	f	f	f

Iteration 9:

A	B	C	D	E	F	G	H	I	J	K
f	T	f	f	f	f	f	T	f	f	f

Iteration 10:

A	B	C	D	E	F	G	H	I	J	K
f	T	f	f	f	f	f	T	f	f	f

Iteration 11:

A	B	C	D	E	F	G	H	I	J	K
f	T	f	f	f	f	f	f	f	f	f

Source vertices : B