NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

CX2001 Algorithms **2019/20 Semester 1**

Tutorial 4: Graph

School of Computer Science and Engineering *Nanyang Technological University*

Week 10 (Q1-Q3): use Adj list

Q1 The degree of a vertex v of a graph is the number of edges incident on v. Design an algorithm to compute vertex degrees using adjacent lists and using adjacency matrix respectively. Compare the time complexity of the two algorithms in terms of n and m, the number of vertices and the number of edges of a graph.

Q2 Manually execute breadth-first search on the undirected graph in Figure 4.1, starting from vertex s. Then, use it as an example to illustrate the following properties:

(a) The results of breadth-first search may depend on the order in which the neighbours of a given vertex are visited.

(b) With different orders of visiting the neighbours, although the BFS tree may be different, the distance from starting vertex s to each vertex will be the same.

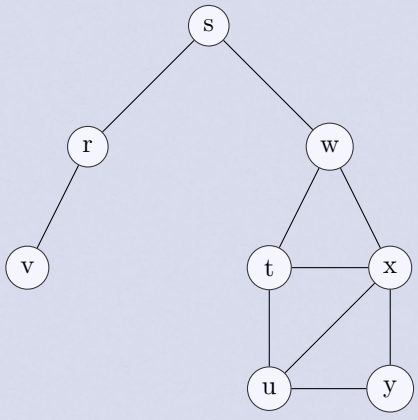


Figure 4.1: Graph for Q2

Q3 Prove the following proposition: Let v be the root of a breadth-first search tree for a graph G. For any vertex x in the BFS tree, the path from v to x using the tree edges is always a shortest path from v to x in the graph. [Note: Here, the length of a path from u to x is measured by the number of edges in the path, and the length of the shortest path is called the distance from v to x.]

Week 11 (Q4-Q6):

Q4 Design an algorithm to find a simple path connecting two given vertices in an undirected graph in linear time (Hint: Use depth-first search).

Q5 A backtracking algorithm may be used to determine and print all possible sequences in ascending positive integers that are summed to give a positive integer C. For example, if the input value for C is 6, the

output should be

1 2 3
1 5
2 4
6

Give a pseudocode of a recursive backtracking algorithm for the function sumToC(sequence s, int C). Sequence s, initially empty, is the (partial) sequence the algorithm has computed so far. Integer C is an input parameter of the function.
Draw the tree representing the (partial) solution space that will be searched by the backtracking algorithm for C = 10. Identify solutions and dead-ends in the tree.

Q6 Apply the Dijkstra's algorithm on the graph represented by the following adjacency matrix to find the shortest distances and the shortest paths from vertex 1 to the other vertices. Show the contents of arrays S, d and pi after each iteration of the while loop.

vertex	1	2	3	4	5
1	0	4	2	6	8
2	∞	0	∞	4	3
3	∞	∞	0	1	∞
4	∞	1	∞	0	3
5	∞	∞	∞	∞	0

Week 12 (Q7-Q9):

Q7 Let $G = (V, E, W)$ be a weighted graph, and let s and z be distinct vertices. In the graph, there may be more than one shortest path from s to z. Explain how to modify Dijkstra's shortest-path algorithm to determine the number of distinct shortest paths from s to z. Assume all edge weights are positive.

Q8 Execute by hand the Prim's algorithm for finding minimum spanning tree (MST) on the graph in Figure 4.2, starting from vertex G. Show the contents of arrays S, d and pi after each iteration of the while loop when a vertex is added to the MST.

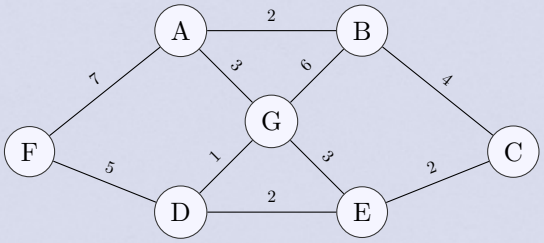


Figure 4.2: Graph for Q8

Q9 In a weighted undirected graph, is the path between two vertices in a minimum spanning tree always the shortest path (i.e. a path with the minimum weight) between the two vertices in the graph? If your answer is yes, give a proof; otherwise, give a counterexample.

Q1

Q1 The degree of a vertex v of a graph is the number of edges incident on v. Design an algorithm to compute vertex degrees using adjacent lists and using adjacency matrix respectively. Compare the time complexity of the two algorithms in terms of n and m, the number of vertices and the number of edges of a graph.

Handshaking theorem

sum of vertex degree = 2 × No. of edges.

> Each edge has 2 endpts, so it contributes 1 to each degree of its endpts
> Each degree adds 2 to the total vertex degree

comparisons:

degree 1: $\Theta(n^2)$

degree 2: $\Theta(n+m)$

complete graph: A simple graph with n vertices has max $\frac{n(n-1)}{2}$ edges

e.g 

If input is complete graph
both degree 1 & degree 2 $\Theta(n^2)$ ⇒ No diff

However real world graph, not complete.
If $m \ll \frac{n(n-1)}{2}$, deg 2 more eff than deg 1

> layerwise manner
1. move horizontally at visit all nodes of current layer
2. move to next layer.

Q2 Manually execute breadth-first search on the undirected graph in Figure 4.1, starting from vertex s. Then, use it as an example to illustrate the following properties:

(a) The results of breadth-first search may depend on the order in which the neighbours of a given vertex are visited.

(b) With different orders of visiting the neighbours, although the BFS tree may be different, the distance from starting vertex s to each vertex will be the same.

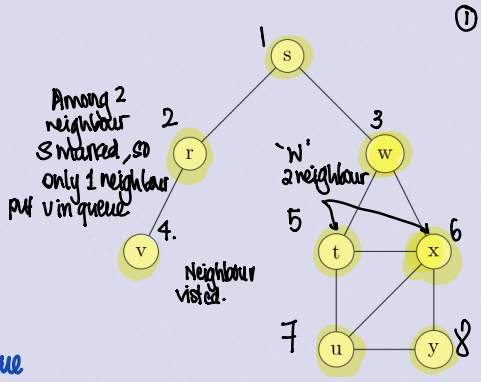
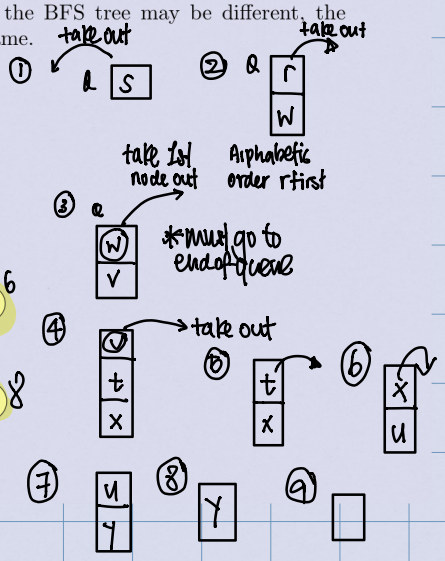
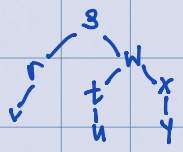


Figure 4.1: Graph for Q2

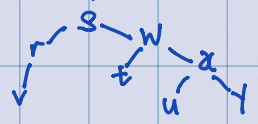


1. Push source node into queue
2. Take first node, v, out of queue
3. mark v as visited
4. Access v neighbours (following pre-defined order)
5. If neighbour has not been visited yet, add it to end of queue
6. Repeat line 2 until queue empty.

Alphabetical order



Reverse alphabetical order

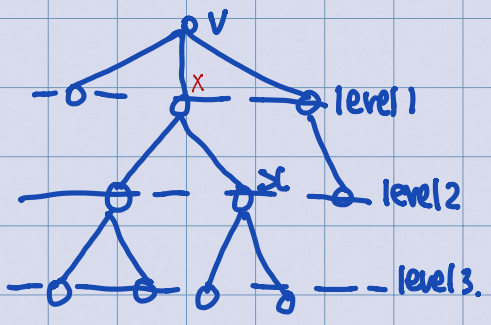


1. two trees diff in that vertex u is adj with t in the left tree, but adj with x in right tree

2. distance from starting vertex s to each other vertex is equal in the 2 trees.

Q3 Prove the following proposition: Let v be the root of a breadth-first search tree for a graph G. For any vertex x in the BFS tree, the path from v to x using the tree edges is always a shortest path from v to x in the graph. [Note: Here, the length of a path from u to x is measured by the number of edges in the path, and the length of the shortest path is called the distance from v to x.]

prove distance v & x = level of x in BFS tree



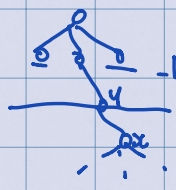
prove by induction on K, the shortest distance

① base case: K=1, the distance btw root v and a vertex is 1
meaning v & x are connected of x is v's neighbour, x is on the first level of BFS tree

② Induction hyp: let y be an arbitrary node whose distance to v is K-1, then Y is on the (K-1)th level of BFS tree.

- consider node X, whose distance to v is K. Among x's neighbours, there exists a vertex, say u, whose dist= to v is K-1
- By our hyp: u is on the (K-1)th level of the BFS tree. During BFS, x is listed AFTER u has been visited, & X cannot be on the same level as u

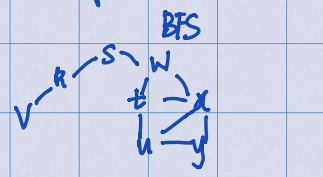
∴ x is on the Kth level of BFS tree

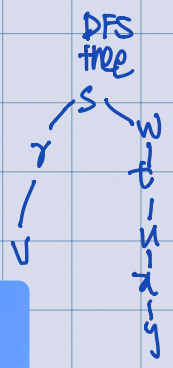


distance(x,u) = K
∃u which is x's neighbour
Shortest distance(u,v) = K-1

Q4 Design an algorithm to find a simple path connecting two given vertices in an undirected graph in linear time (Hint: Use depth-first search).

BFS compute shortest path but DFS does not.





DFS → compute a simple path

BFS → shortest path.

added edges during visiting

- DFS

```
void DFS(Graph G, vertex s)
    label s as discovered
    for all edges from s to w that are in G.adjacentEdges(s) do
        if vertex w is not labeled as discovered then
            DFS(G, w) // recursive call
            add edge sw to the tree
```

A naïve solution:
Step 1: Call DFS for source vertex s and compute the entire DFS tree
Step 2: Backtrack from t in the DFS tree to obtain the path

A faster algorithm: stop traversing when the path is found

```
bool DFS_path(Graph G, vertex s, vertex t)
    if (s == t) return true;
    label s as discovered
    for all edges from s to w that are in G.adjacentEdges(s) do
        if vertex w is not labeled as discovered then
            if (DFS_path(G, w, t) == true) then
                print (w, s) // output the edge (s, w)

    return false;
```

output recursive

source nodes:

visit s
visit r
visit v
add(v,r)
add(r,s)
visit w
visit t
visit u
visit x
visit y

add(y,x)
add(u,x)
add(t,u)
add(w,t)
add(s,w)

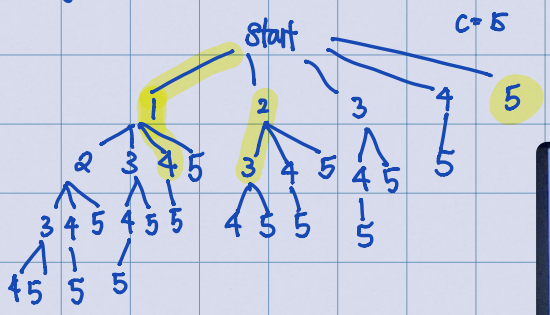
Q5 A backtracking algorithm may be used to determine and print all possible sequences in ascending positive integers that are summed to give a positive integer C. For example, if the input value for C is 6, the

output should be

1 2 3
1 5
2 4
6

Give a pseudocode of a recursive backtracking algorithm for the function sumToC(sequence s, int C). Sequence s, initially empty, is the (partial) sequence the algorithm has computed so far. Integer C is an input parameter of the function.
Draw the tree representing the (partial) solution space that will be searched by the backtracking algorithm for C = 10. Identify solutions and dead-ends in the tree.

Idea: apply PFS to find all paths whose elements add up to C.



Instead of constructing searching tree explicitly, use list to maintain path

```
// a naïve method   Have to search entire tree
vector<int> L; // initially, L is empty
void SumToC(vector<int>& L, int C) {
        if (Sum(L) == C) {
            Print(L);  // sequence found !!
        } else {
            // get the value of the last element in the current path
            int last = (L.size()==0) ? 0 : L.back();
            for (int j = last + 1; j <= C; j ++) {
                        L.push_back(j); // add "j" to the end of the path
                        SumToC(L, C);
                        L.pop_back(); // remove the last element and backtrack
                }
        }
}
```

```
// A faster algorithm
vector<int> L; // initially, L is empty
void SumToC(vector<int>& L, int C) {
        if (Sum(L) == C) {
            Print(L);  // sequence found !!
        } else {
            // get the value of the last element in the current path
            int last = (L.size()==0) ? 0 : L.back();
            for (int j = last + 1; j <= C; j++) {
                if (Sum(L) + j <= C) {
                    L.push_back(j);
                    SumToC(L, C);
                    L.pop_back();
                }
            }
        }
}
```

Q6 Apply the Dijkstra's algorithm on the graph represented by the following adjacency matrix to find the shortest distances and the shortest paths from vertex 1 to the other vertices. Show the contents of arrays S, d and pi after each iteration of the while loop.

vertex	1	2	3	4	5
1	0	4	2	6	8
2	∞	0	∞	4	3
3	∞	∞	0	1	∞
4	∞	1	∞	0	3
5	∞	∞	∞	∞	0

Shortest distances in unweighted graph: BFS (FIFO)

Shortest distance in weighted graph: Dijkstra's alg (priority queue)

S: set of vertices whose shortest path from source node have already been determined (they finished

V-S remaining vertices distance are unknown

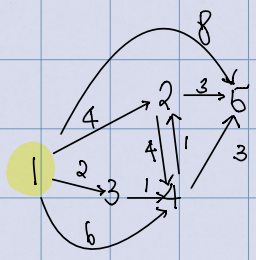
d: array of estimates for the shortest path from source node to all vertices.

pi: array of predecessors for each vertex

A priority queue (binary heap): top node is the unvisited vertex w the least distance to source.

Step 1. Mark all nodes unvisited.
Step 2. Assign zero to the source node and infinity to all other nodes. Push the source node into a priority queue (binary heap: the shorter the distance, the higher the priority).
Step 3. Pop up the top element, say u, from the priority queue.
- For each of u's unvisited neighbors, say v, calculate its *tentative* distance using the weight of edge (u,v)
- Compare the newly calculated *tentative* distance to the current assigned value of v; if it is smaller, assign the new distance. ← one have, push to queue
- When we are done considering all of the unvisited neighbors of u, mark u as visited.

Step 4. If the priority queue is empty, stop the algorithm. Otherwise, go back to step 3.



After initialization

	1	2	3	4	5
S	0	0	0	0	0
d	0	∞	∞	∞	∞
pi	null	11	11	11	11

1st iteration
- Node1 marked as visited.
- update distance of node 1 neighbours.
- update predecessors for unvisited neighbours

	1	2	3	4	5
S	1	0	0	0	0
d	0	4	2	6	8
pi	null	1	1	1	1

2nd iteration
- 3 shortest dist
- mark Node3 as visited.
- update dist of Node3 unvisited neighbours.
- update predecessors for unvisited neighbours

	1 ✓	2	3 ✓	4	5
S	1	0	1	0	0
d	0	4	2	3	8
pi	null	1	1	3	1

3rd iteration
- 4 ,,

4th iteration
- 2 ,,

5th iteration
- popeNode 5 out of

	1	2	3	4	5
S	1	1	1	1	1
d	0	4	2	3	6

shortest path
Node 5 parent 4,

priority.

pi null 1 1 3 4 Max 4 parent 3
" 3 1