COMP2521 Week 4 Tutorial

Graph Traversal and Graph Algorithms

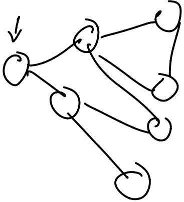
Jeremy Le

2026T0

Announcements

- Quiz04 due 12pm Wednesday
- Lab03 handmarking (complexity analysis) due this week!
- Lab04 due 12pm Monday Week 5
- Assignment due 8pm Monday Week 5
- Sample Exam in Week 5 lab

Graph Traversal



Breadth-first search (BFS)

- Prioritises exploring widely over exploring deeply
- Implemented iteratively (using a queue)

Depth-first search (DFS)

- Prioritises exploring deeply over exploring widely
- Implemented recursively **or** iteratively (using a stack)

Graph Traversal - Code

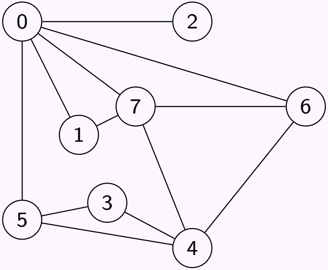
```c
void breadthFirst(Graph g, int src) {
    bool *visited = calloc(g->nV, sizeof(bool));
    int *pred = calloc(g->nV, sizeof(int));
    Queue q = QueueNew();

    visited[src] = true;
    QueueEnqueue(q, src);
    while (!QueueIsEmpty(q)) {
        int v = QueueDequeue(q);


        printf("%d\n", v);
        for (int w = 0; w < g->nV; w++) {
            if (g->edges[v][w] && !visited[w]) {
                visited[w] = true;
                pred[w] = v;
                QueueEnqueue(q, w);
            }
        }
    }

    free(visited);
    free(pred);
    QueueFree(q);
}
```

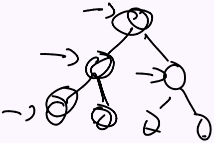
```c
void depthFirst(Graph g, int src) {
    bool *visited = calloc(g->nV, sizeof(bool));
    int *pred = calloc(g->nV, sizeof(int));
    Stack s = StackNew();

    StackPush(s, src);
    while (!StackIsEmpty(s)) {
        int v = StackPop(s);

        if (visited[v]) continue;
        visited[v] = true;

        printf("%d\n", v);
        for (int w = g->nV - 1; w >= 0; w--) {
            if (g->edges[v][w] && !visited[w]) {
                pred[w] = v;
                StackPush(s, w);
            }
        }
    }

    free(visited);
    free(pred);
    StackFree(s);
}
```

Graph Traversal - Example

Question 1

Consider the breadth-first and depth-first traversal algorithms below and the following graph:





Trace the execution of the traversal algorithms, and show the state of the VISITED and PRED arrays and the QUEUE (BFS) or STACK (DFS) at the end of each iteration, for each of the following function calls:

- BREADTHFIRST(G, 0);
- BREADTHFIRST(G, 3);
- DEPTHFIRST(G, 0);
- DEPTHFIRST(G, 3);

Graph Traversal - Example Working

For BREADTHFIRST(G, 0):

#	Printed	visited								pred								Queue (front at left)
		0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	
0		1	0	0	0	0	0	0	0									0
1	0	1	1	1	0	0	1	1	1		0	0			0	0	0	1 2 5 6 7
2	1	1	1	1	0	0	1	1	1		0	0			0	0	0	2 5 6 7
3	2	1	1	1	0	0	1	1	1		0	0			0	0	0	5 6 7
4	5	1	1	1	1	1	1	1	1		0	0	5	5	0	0	0	6 7 3 4
5	6	1	1	1	1	1	1	1	1		0	0	5	5	0	0	0	7 3 4
6	7																	
7	3																	
8	4																	

For BREADTHFIRST(G, 3):

#	Printed	visited								pred								Stack (top at right)
		0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	
0																		
1																		
2																		
3																		
4																		
5																		
6																		
7																		
8																		

Graph Traversal - Example Working

For DEPTHFIRST(G, 0):

#	Printed	visited								pred								Stack (top at right)
		0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	
0																		
1																		
2																		
3																		
4																		
5																		
6																		
7																		
8																		

For DEPTHFIRST(G, 3):

#	Printed	visited								pred								Queue (front at left)
		0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	
0																		
1																		
2																		
3																		
4																		
5																		
6																		
7																		
8																		

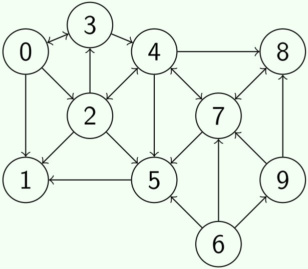
Graph Algorithms

Reachability

The ability to get from one vertex to another within a graph.
A vertex s can reach a vertex t (and t is reachable from s) if there exists a sequence of adjacent vertices (i.e. a walk) which starts with s and ends with t.

Example

In the following graph:



Vertices 1, 2, 3, 4, 5, 7 and 8 are reachable from vertex 0.

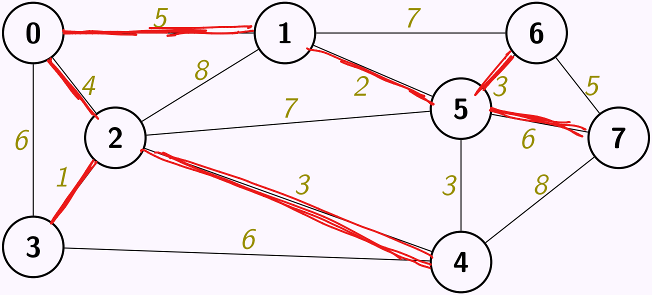
Graph Algorithm - Dijkstra's Algorithm

- Finds the shortest path (single source to all other vertices) in a weighted graph with non-negative weights
  - For unweighted graphs, we can use BFS
  - For weighted graphs with negative weights, we use Bellman-Ford (not taught in this course)
- Data structures used in Dijkstra's algorithm:
  - Distance array (DIST)
    - To keep track of shortest currently known distance to each vertex
  - Predecessor array (PRED)
    - Same purpose as in BFS/DFS
    - To keep track of the predecessor of each vertex on the shortest currently known path to that vertex
    - Used to construct the shortest path
  - Set of vertices
    - Stores unexplored vertices

Graph Algorithm - Dijkstra's Algorithm Example

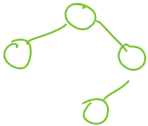
Question 3

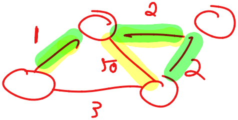
Trace the execution of Dijkstra's algorithm on the following graph to compute the minimum distances from source node 0 to all other vertices:

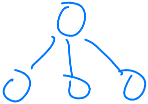


Graph Algorithm - Dijkstra's Algorithm Example Working

Graph Algorithm - Minimum Spanning Tree







Spanning Tree

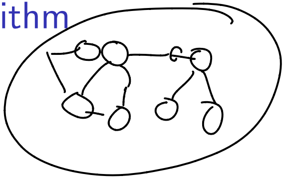
A spanning tree T of an undirected graph G is a subgraph that is a tree which includes all of the vertices of G.

Minimum Spanning Tree

A subset of the edges of a connected, edge-weighted undirected graph that connects all the vertices together, without any cycles and with the minimum possible total edge weight.

Graph Algorithm - Kruskal's Algorithm

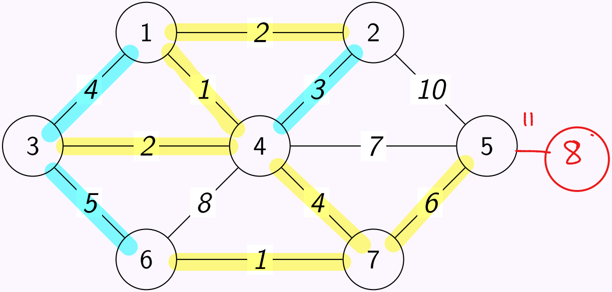
union find



- This algorithm effectively constructs the MST by gradually joining together the connected graphs in a forest that starts with each subgraph being a single node.
- On each iteration, it add a new edge to the forest, and reduces the number of subgraphs by one.

Graph Algorithm - Kruskal's Algorithm Example

Question 4



- How many edges did we have to consider? 9 out of 12
- For a graph $G(V, E)$, what is the least number of edges we might need to consider? $|V| - 1$
- What is the most number of edges we might have to consider? $|E|$
- Add another edge to the above graph to force Kruskal's algorithm to the worst case.

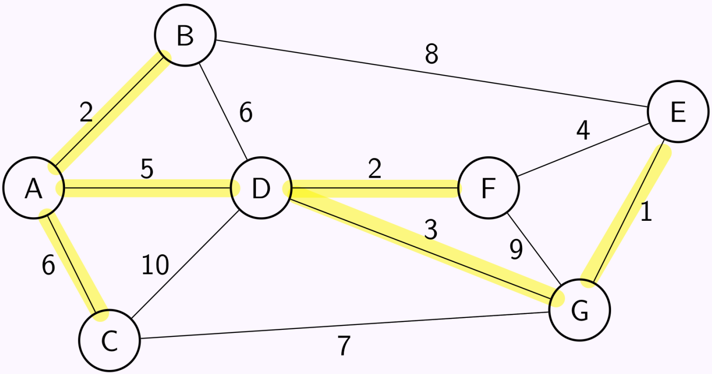
Graph Algorithm - Prim's Algorithm

1. Start from any vertex v and empty MST
2. Choose edge not already in MST, satisfying
   - incident on a vertex s already in MST
   - incident on a vertex t not already in MST
   - with minimal weight of all such edges
3. Add chosen edge to MST
4. Repeat until MST covers all vertices

Graph Algorithm - Prim's Algorithm Example

Question 5

Show how Prim's algorithm produces an MST on the graph below:



$V = \{A\}$ $E = \{\}$

$V = \{A, B\}$ $E = \{AB\}$

$V = \{A, B, D\}$ $E = \{AB, AD\}$

$V = \{A, B, D, F\}$ $E = \{AB, AD, DF\}$

Graph Algorithm - Prim's Algorithm Example Working