Part 4

1 Breadth-First Search

1.1 Graphs

- A graph G=(V,E) is a pair of sets: vertices V and edges E
- To give an adjacency list representation of a graph, for each vertex v list all the vertices adjacent to v
- To give an adjacency matrix representation of a graph create a square matrix A and label the rows and columns of the vertices: the entry in row i column j is 1 if vertex j is adjacent to vertex i and 0 if it is not

1.2 Breadth-First Search

- BFS maintains a queue that contains vertices that have been discovered but are waiting to be processed
- BFS colours the vertices:
  - **White** indicates that a vertex is undiscovered
  - **Grey** indicates that a vertex is discovered but unprocessed
  - **Black** indicates that a vertex has been processed
- The algorithm maintains an array d (distance)
  - d[s]=0 where s is the source vertex
  - If we discover a new vertex v while processing u, we set d[v]=d[u]+1

Listing 1: BFS(G,s)

```
for each vertex u∈ V[G]-{s}
    do colour[u]← WHITE
        d[u]← ∞
            π[u] ← NIL
colour[s]=GREY
d[s]← 0
π[s] ← NIL
Q← ∅
ENQUEUE(Q,s)
while Q ≠ ∅
    do u←DEQUEUE(Q)
        for each v∈ Adj[u]
            do if colour[v]=WHITE
                then colour[v]=GREY
                    d[v]←d[u]+1
                    π[v] ← u
                    ENQUEUE(Q,v)
        colour[u]← BLACK
```

1.3 Analysis of running time

- We want an upper bound on the worst-case running time
- Assume that it takes constant time for each operation such as to test and update colours, to make changes to distance and to enqueue and dequeue
- Initialisation takes time $O(V)$
- Each vertex enters (and leaves) the queue exactly once. So queueing operations take $O(V)$

- In the loop the adjacency lists of each vertex are scanned. Each list is read once, and the combined lengths of the lists is $O(E)$
- This the total running time is $O(V + E)$

2 Depth-First Search

- Initialize: source vertex grey, others white; source discovered at time 1
- Repeat:
  - Increment the time
  - If there is a white neighbour of the current vertex, then it is coloured grey and its discovery time noted and it becomes current
  - Else colour the current vertex black, not its finish time and return to its predecessor or jump to an undiscovered vertex

Listing 2: DFS(G)

```
for each vertex u∈ V[G]
    do colour[u]← WHITE
        π[u] ← NIL
time ← 0
for each vertex u ∈ V[G]
    do if colour[u] = WHITE
        then DFS-VISIT(u)
```

Listing 3: DFS-VISIT(u)

```
colour[u]← GREY                          [vertex u has just been discovered]
time←time+1
d[u]←time
for each vertex v ∈ Adj[u]                           [explore edge (u,v)]
    do if colour[v]=WHITE
        then π[v] ← u
            DFS-VISIT(v)
colour[u]←BLACK                                     [u has been processed]
f[u]←time←time+1
```

- Initialisation takes time $O(V)$
- Time $O(V)$ spent on incrementing time, colouring vertices and updating d and f
- Each vertex in each adjacency list is considered at most once. This takes time $O(E)$
- Total time is $O(V + E)$

2.1 Classification of the edges

- **Tree** edges are those edges in the DFS-forest
- **Back** edges are edges that join a vertex to an ancestor
- **Forward** edges are edges not in the tree that join a vertex to its descendant
- **Cross** edges: all other edges

The classification is ambiguous for undirected graphs (back edges and forward edges are the same thing)

Let us redefine the definition: suppose that e is an edge that joins a vertex u to its descendant v

- e is a forward edge if DFS first considers e from u
- e is a back edge if DFS first considers e from v

In an undirected graph, every edge is a tree edge or a back edge

2.2 Using depth first search

- Every edge in an undirected graph is either a tree edge or a back edge
- A graph is connected if each pair of vertices is joined by a path
- A cycle is a sequence of edges that start and end at the same vertex
- An articulation point is a vertex whose removal disconnects the graph

3 Minimum Spanning Trees

The minimum spanning tree problem

Find a tree that spans the vertices and has minimum cost

Basic properties of MSTs

- Have $|V| - 1$ edges
- Have no cycles
- Might not be unique

3.1 Kruskal's algorithm

1. Sort the edges by weight
2. Let $A = \emptyset$
3. Consider edges in increasing order of weight. For each edge e, add e to A unless this would create a cycle

Running time is $O(E \log V)$

3.2 Prim's algorithm

1. Let $U = \{u\}$ where u is some vertex chosen arbitrarily
2. Let $A = \emptyset$
3. Until U contains all vertices: find the least-weight edge e that joins a vertex v in U to a vertex w not in U and add e to A and w to U

Running time is $O(V \log V + E)$