The University of Newcastle
School of Electrical Engineering and Computer Science

COMP2230/6230 Algorithms

Tutorial Week 6 Solutions

23rd – 27th August 2021

Tutorial

1. Using the graph in Figure 1, list the order in which depth-first search visits the vertices, assuming that the vertices are listed in ascending order, starting the search at 1.

   Solution: 1,2,3,4,8,7,6,5,9,10,13,14,11,12,16,15

2. Using the graph in Figure 1, list the order in which breadth-first search visits the vertices, assuming that the vertices are listed in ascending order, starting the search at 1.

   Solution: 1,2,5,6,3,7,9,10,15,4,8,11,13,14,16,12

3. Using the directed acyclic graph in Figure 2, trace Algorithm 4.4.1 (Topological Sort).

   Solution: Recall that a Topological Sort is just a depth first search, where we list in reverse order the vertices that we backtrack on when they become a dead end.

   7 5 8 2 1 3 6 9 10 4

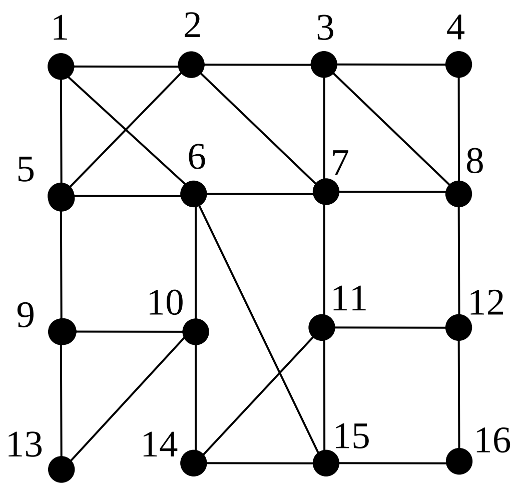

Figure 1: A Simple Graph

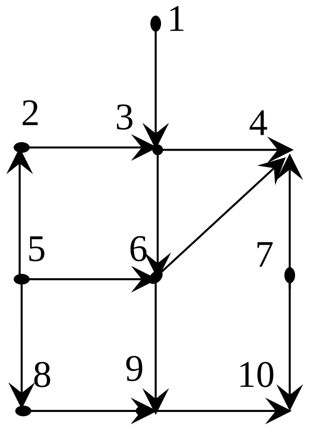

Figure 2: A Directed Acyclic Graph

4. A shortest path *δ(u, v)* between two vertices, *u* and v, in a simple graph has a property that for any vertex k not on the path $\delta(u, v) \leq \delta(u, k) + \delta(k, v)$. That is, the shortest path is the minimum number of edges that have to be traversed to get from *u* to *v* (and vice versa in a simple undirected graph). Use either depth-first or breadth-first search to calculate the shortest paths from a fixed start vertex s.

   Solution idea: A breadth first search tree should be used to determine the shortest path from the root to every other vertex. Simply keep track of which level the current vertex is one with respect to the root node, with the root being on level 0.

5. Write a backtracking algorithm that prints all permutations of the numbers *1, . . . , n.*

   Solution:

```
perm(n){
    for k=1 to n
       used[k]= false;
    reperm(1, n)
}

reperm(k,n) {
   for s = 1 to n
       if (!used[s]) {
           used[s]=true;
           x[k]=s;
           if (k==n) {
              for i= 1 to n
                  printf(x[i]+"");
           }
           else
              reperm(k+1, n);
           used[s]=false;
       }//if
}// reperm
```

6. Let T [0, 1, . . . , n-1] be a sorted array of distinct integers (they may be negative). Give an algorithm that finds an index i such that T [i] = i, if it exists. What is your algorithm's running time?

   Solution:

```
altbsearch(int[] T){

    int i = 0;
    int j = T.length-1;
    while (i <= j){
        int k = (i+j)/2;
        if (T[k] == k) return k;
        if (T[k] < k) i = k+1;
        else j = k-1;
    }
    return -1;
}
```

Homework

7. Write a non-recursive version of depth-first search.

   Solution:

```
dfs(root){
    s = stack_init( );
    mark root as visited;
    s.push(root);
    while (there are unvisited vertices){
       if (s.empty( ))
          mark an unvisited vertex as visited and push onto the stack;
       v = s.top( );
       if (v has an unvisited neightbour u){
          mark u as visited;
          s.push(u);
       }
       else
          s.pop();

}
```

More detailed pseudocode, using adjacency lists – adj is an array of size n such that adj[i] contains a pointer to the linked list of neighbours of i:

```
dfs(adj){
    n = adj.last
    for i = 1 to n
        visit[i] = false
    s = stack_init( );
    for i = 1 to n {
        if (!visit[i]){
            visit[i] = true;
            s.push(i);
            while (!s.empty( )){
                neighbour = adj[s.top( )]
                while (neighbour != NULL){
                    if (!visted[neighbour.vertex]){
                        visit[neighbour.vertex]=true;
                        s.push(neighbour.vertex);
                        neighbour = adj[neighbour.vertex];
                    }
                    else neighbour = neighbour.next
                }\\while
                s.pop( )
            }\\while
    }\\for
}\\dfs
```

8. What is the best case time for depth-first search?

   Solution: The asymptotic time for depth first search is Θ(|V| + |E|), the best and worst case for this depends on how many edges there are. If the graph is a tree, or a forest (note that this includes a disconnected graph and a path as subcases), then are at most a linear number of edges, thus the running time is Θ(|V|). If the graph is a complete graph, or close, then there are roughly |V|2 edges, giving a running time of Θ(|V |2).

9. Show that Algorithm 4.4.1 (Topological Sort) runs in time Θ(|V | + |E |) for a graph G = (V , E).

   Solution: Topological is just a depth first search, in which we do Θ(n) additional operations in total over the whole search (storing the vertices in the list as they become a dead end). Thus the running time remains Θ(|V | + |E |).

10. Give a directed acyclic graph with at least 4 vertices that has a unique topological sort of

the graph.

Solution: Any graph with a Hamiltonian path has a unique topological sort.

11. Write a backtracking algorithms that prints all subsets of the set {1, 2, . . . , n}. ***Solution:*** The solution array x contains 0's and 1's. If x[k]=0, k is not included in the subset. If x[k]=1, k is included in the subset.

```
all_subsets(n) {

        rec_all_subsets(n,1)

}

rec_all_subsets (n, k) {

            for s = 0 to 1
                x[k]=s;
                if (k==n) {
                    for  i=1 to n
                         if  (x[i] ==1)
                              print (i+" ");
                }
                else
                     rec_all_subsets (n, k+1);
}
```

Extra Questions

12. Write a version of depth-first search in which the input graph is in the form of an adjacency matrix. What is the worst case time of your algorithm?

Solution:

```
dfs (adjm, start){
        for i = 1 to adjm.last
           visit[i]=false;
        dfs_recurs(adjm, start)
}

dfs_recurs (adjm, start) {
       println(start)
       visit[start]=true;
       for v = 1 to adjm.last
           if (adjm[start][v]= =1 && !visit[v])
                     dfs_recurs(adjm,v)     }
```

The worst-case time is $\Theta(n^2)$

13. Give an example of a directed acyclic graph with at least 4 vertices in which every permutation of the vertices is a topological sort of the graph.

Solution: Consider the graph with four vertices and no edges.

14. Show all solutions to the 4-Queens problem.

Solution:

		*	
*			
			*
	*		

	*		
			*
*			
		*	

15. Trace a depth-first and breadth-first search for the graphs in exercises 4.2.1 and 4.2.2 in the text.

Solution:
BFS for exercise 4.2.1: **1, 2, 4, 3, 5**
DFS for exercise 4.2.1: **1, 2, 3, 4, 5**

DFS for exercise 4.2.2: **3, 2, 1, 4, 5, 6, 9, 8, 7, 10, 11, 12. (Starting vertex 3)**
BFS for exercise 4.2.2: **1,2,4,5, 3, 7, 6, 8, 9, 10, 11, 12 (Starting vertex 1)**

16. Trace a depth-first and breadth-first search for the directed graphs in exercises 4.4.1, 4.4.2 and 4.4.3 in the text.

Solution:

DFS for 4.4.1: **1, 3, 4, 6, 9, 10, 2, 5, 8 ,7 (Starting vertex 1)**
DFS for 4.4.2: **5, 2, 1, 3, 4 (Starting vertex 5)**
DFS for 4.4.3: **7, 3, 4, 6, 8, 1, 5, 2 (Starting vertex 7)**
BFS for 4.4.1: **1, 3, 4, 6, 9, 10, 2, 5, 8, 7 (Starting vertex 1)**
BFS for 4.4.2: **5, 2, 3 , 4, 1 (Starting vertex 5)**
BFS for 4.4.3: **7, 3, 4, 6, 8, 1, 5, 2 (Starting vertex 7)**