Homework Assignment 2

Introduction to Data Structures and OO

Instructor: 王豐堅

Due: 15/05/2021

1) Develop and test a complete C++ template class for linked queues.

2) Write a non-recursive version of function *Postorder* below.

```
Template <class T>
Void Tree<T>::Postorder(TreeNode<T>*currentNode)
{//Workhorse.
        if(currentNode){
           Postorder(currentNode →leftChild);
         Postorder(currentNode →rightChild);
          Visit(currentNode);
        }
}
```

3) Devise an external representation for the formulas in propositional calculus. Write a function that reads such a formula and creates its binary tree representation. What is complexity of your function?

4) The worst-case number of comparisons performed during an insertion into a max heap can be reduced to O(loglog *n*) by performing a binary search on the path from the new leaf to the root. This does not affect the number of data moves though. Write an insertion algorithm that uses this strategy. Based on your experiments, what can you say about the value of this strategy over the one used in following program?

```
template <class T>
void MaxHeap<T>::Push(const T& e)
{ // Insert e into the max heap.
      if (heapSize == capacity) {// double the capacity
            ChangeSize 1D (heap, capacity, 2*capacity);
             capacity * = 2;

}

     int currentNode = ++heapSize;
    while (currentNode !=1 && heap [currentNode/2] <e)
    {// bubble up
       heap [currentNode ] = heap [currentNode /2]; // move parent down
       currentNode /= 2;

}
```

```
        heap [currentNode ] = e;
    }
```

5) Prove that the level-order traversal of a forest and that of its corresponding binary tree do not necessarily yield the same result.

6) Suppose we start with n sets, each containing a distinct element.
   a) Show that if u unions are performed, then no set contains more than u + 1 elements.
   b) Show that at most $n - 1$ unions can be performed before the number of sets becomes 1.

7)
   a) Prove that when function *DFS* is applied to a connected graph, the edges of T form a tree.
   b) Prove that when function *BFS* is applied to a connected graph, the edges of T form a tree.

```
virtual void Graph::BFS(int v)
{// A breadth first search of the graph is carried out beginning at vertex v.
 // visited [i ] is set to true when v is visited. The function uses a queue.
    visited = new bool [n ];
    fill (visited, visited + n, false);
    visited [v ] = true;
    Queue<int> q;
    q.Push (v);
    while (!q.IsEmpty ()) {
        v = q.Front ();
        q.Pop ();
        for (all vertices w adjacent to v) // actual code uses an iterator
            if (!visited [w ]) {
                q.Push (w);
                visited [w ] = true;
            }
    } // end of while loop
    delete [] visited;
}
```

```
virtual void Graph::DFS() // Driver
{
    visited = new bool [n ];
        // visited is declared as a bool* data member of Graph
    fill (visited, visited + n, false); // initialize visited to false
    DFS (0); // start search at vertex 0
    delete [] visited;
}

virtual void Graph::DFS(const int v) // Workhorse
{// Visit all previously unvisited vertices that are reachable from vertex v.
    visited [v ] = true;
    for (each vertex w adjacent to v) // actual code uses an iterator
        if (!visited [w ]) DFS (w);
}
```

8) Show that the number of spanning trees in a complete graph with n vertices is at least $2^{n-1} - 1$.

9) Refine the following program into a C++ function to find a minimum-cost spanning tree. The complexity of your function should be $O(n^2)$, where n is the number of vertices in the input graph. Show that this is the case.

```
//Assume that G has at least vertex.
TV = {0}; // start with vertex 0 and no edges
for (T = Θ; T contains fewer than n – 1 edges; add (u, v) to T)
{

    Let (u, v) be a least-cost edge such that u ∈ TV and v ∉ TV;
    if (there is no such edge) break;
```

add *v* to *TV;*

}

if (*T* contains fewer than *n* −*1* edges) **cout** <<"no spanning tree"<< **endl;**

10) Obtain a C++ function to find a minimum-cost spanning tree using Sollin's algorithm. What is the complexity of your function?

11) Define an iterator class *TopoIterator* for iterating through the vertices of a directed acyclic graph in topological order.

12) The *diameter* of a tree is the maximum distance between any two vertices. Given a connected, undirected graph, write an algorithm for finding a spanning tree of minimum diameter. Prove the correctness of your algorithm.