CS 430 Project Report

Kai Yao A20347045
Ke Yao A20359070

Problem Definition

Algorithms, implementation and asymptotic behavior for finding spanning trees.

Description of problem

We are going to choose "finding spanning trees" as our project, there are two algorithms for this project. One is Kruskal algorithm, the other is Prim algorithm.

Kruskal algorithm can be described as following:
Suppose there is a graph G, the edge is represented by e and vertex is represented by v, the minimum spanning tree is T. each edge have start vertex and end vertex and its weight.

1. Sort all the edge in increasing order based on its weight.
2. T ←φ, edge ← 0.
3. While(edge < ｜T｜- 1)
4. Pick smallest edge e_i.
5. If e_i do not form a cycle in T add e_i to T.
6. edge++

From this algorithm we can see that we have multiple ways to determine if one edge form a cycle or not. One way is called union set, another way is called DFS.
In union set basically we give each vertex a different set, when picking an edge we see if this edge's two vertices are in the same set, if it is not, pick this edge, else ignore.
e.start means start vertex, e.end means end vertex, just for clarity.

1. Pick one edge e_i
2. Find e_i start vertex, and e_i end vertex in which set.
3. If two vertices in this edge are not in any set, union them. (do not have a cycle)
4. else If e.start in one set, e.end is not in any set, union them. (do not have a cycle)
5. else If e.end in one set, e.start is not in any set, union them. (do not have a cycle)
6. else If e.start and e.end are in different set, union them. (do not have a cycle)
7. else e.start and e.end are in the same set (have a cycle).

In DFS when picking an edge, we do DFS from any vertex in this edge based on the edges has been choose and connected to this vertex. if we can find another vertex in this edge by DFS, then ignore this edge else pick.

1. Pick an edge e_i
2. start ← e.start, destination← e.end
3. flag←DFS(start, destination, T)

```
4. if(flag)
5.         have a cycle
6. else
7.         do not have a cycle
```

DFS:

```
1. DFS(start, destination, T)
2. mark start
3. if start == destination
4.         return true
5. for each adjacent vertex of start w in T
6.         if w is unmarked
7.                 DFS(start, destination, T)
8. return false
```

Prim algorithm can be described as following:

I implement the Prim Algorithm by two kinds of data structures, the first one is linear array list, and another one is the priority queue.

For the array list, I travel the graph each time to find the shortest edge connect to the tree(the vertices collection that I have found), and then add the vertex connected with the edge to the MST. Here is the pseudocode:

```
1. Prim(G, w)
2. Pick a vertex u and T←u
3. Q = Q - u
4. For each vertex u in the Q
5.       For each v in G.Adj[u]
6.            Find the minimum edge in w(u,v)
7.       T = T + v
8.       Q = Q – v
9. Return T
```

For the priority queue:

```
1.  Make a queue (Q) with all the vertices of G (V);
2.  For each member of Q set the priority to INFINITY;
3.  Only for the starting vertex (u) set the priority to 0;
4.  The parent of (u) should be NULL;
5.  While Q isn't empty
6.       Get the minimum from Q – let's say (u); (priority queue);
7.       For each adjacent vertex to (v) from (u)
8.            If (v) is in Q and weight of (u, v) < priority of (v) then
9.                 The parent of (v) is set to be (u)
10.                The priority of (v) is the weight of (u, v)
```

Time complexity for two algorithms

Time complexity for Kruskal algorithm with union set

Because we have to do sort based on edges, so the time complexity is at least ElogE.
For each edge | E | (at most | V | - 1), we determine if it form a cycle, time complexity is V + ElogV
Because for each edge we have to find two vertices' set in heap O(2logV), this takes logV and if they are in different set, we have to union them O(1),(number of loop for union set is V)
So the total time complexity is O(ElogE + V + ElogV)

Time complexity for Kruskal algorithm with DFS

Because we have to do sort based on edges, so the time complexity is at least ElogE.
For each edge(at most V- 1), we determine if it form a cycle, time complexity is (V – 1)(V + E)
Because in each vertex, you have to do a DFS, whose time complexity is V + E, in this case should be V + V
So the total time complexity is $O(ElogE + V^2)$

Time complexity for Prim algorithm with array list

Because i use array list, and for each vertex the algorithm would scan the all the vertex near by, so the time complexity will be $O(V^2)$

Time complexity for Prim algorithm with priority queue

1, Time required for one call to EXTRACT-MIN(Q)=O(log V) and is called V times.so total time required for EXTRACT-MIN(Q)=O(VlogV).
2, Time required for executing using DECREASE_KEY operation on the queue. So total time required to execute O(Elog V).
3, Using BUILD_HEAP procedure each O(1) for V times, it will require O(V) times
Total time complexity =O(VlogV+ElogV+V)=O(ElogV)

Implementation details

In this project, we use adjacent list to generate vertices and edges.

For star graph, we just generate edges from first vertex to all the rest edges.

For Line graph, we generate edges from 1 to 2, 2 to 3, 3 to 4 and so on so forth. Except for last vertex. It doesn't need an edge.

For random graph, we generate edges for each vertex, each vertex's may have 1-10(inclusive) number of edges, which is generated randomly.

Result

Kruskal algorithm Result:

	Vertices	Edges	Union set	DFS
Star	1000	999	40ms	2597ms
Line	1000	999	64ms	540ms
Radom	1000	5518	15ms	3422ms

Prim algorithm Result:

	Vertices	Edges	ArrayList	PriorityQueue
Star	1000	999	2757ms	1756ms
Line	1000	999	26ms	1677ms
Radom	1000	5518	10ms	485ms

Analysis of Result

For the Kruskal Algorithm:

We can see from result that union set algorithm is better than DFS in any input graph. I want to make following several observations:

1, In union set method. Line graph is slowest because there will be a big chance that you have to create two different set for each node and combine them one by one. However, in star graph, each time you are going to add new vertex to an existed set. In random graph, because the graph may be disconnected, so it is the fastest one.

2, In DFS, star graph is slow because you may test each branch in graph in one DFS probe, but in Line, you have no branch. In random graph, both the depth and branch number may be larger than Line and Star.

For the Prim Algorithm:

1, Star: Because in the Arraylist data structure, every time we discover a vertex, we will travel the whole graph on time. So it's higher than the PriorityQueue.

2, Line: Because it's line so the ArrayList is very fast, it can finish in O(n). But in the PriorityQueue, it will spend more time on the data structure operation, so the time is higher.

3, Random: the ArrayList is less because the graph we generate randomly may not a full graph, all the vertex may not in one graph, so the time complexity is much less than the previous.