Assignment 5

CS 514 – Algorithms

Submitted By: Aman Pandita

Onid: panditaa@oregonstate.edu

Compare the running times of the two algorithms as a function of the graph sizes (number of edges). Show the running times a table and a plot in the report.

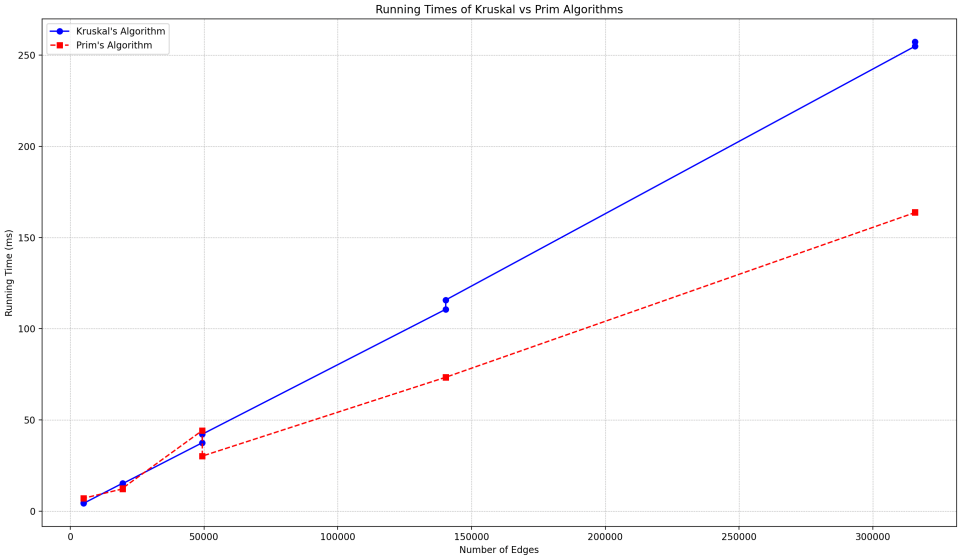
The graph sizes are given as follows:

	number of nodes	number of edges
test case 1 and 2:	1000	315745
test case 3 and 4:	1500	140359
test case 5 and 6:	2000	49370
test case 7 and 8:	1000	19607
test case 9 and 10:	1500	4994

My results for the testcases given on canvas module (changed the name of testcases so that it was easy to print):

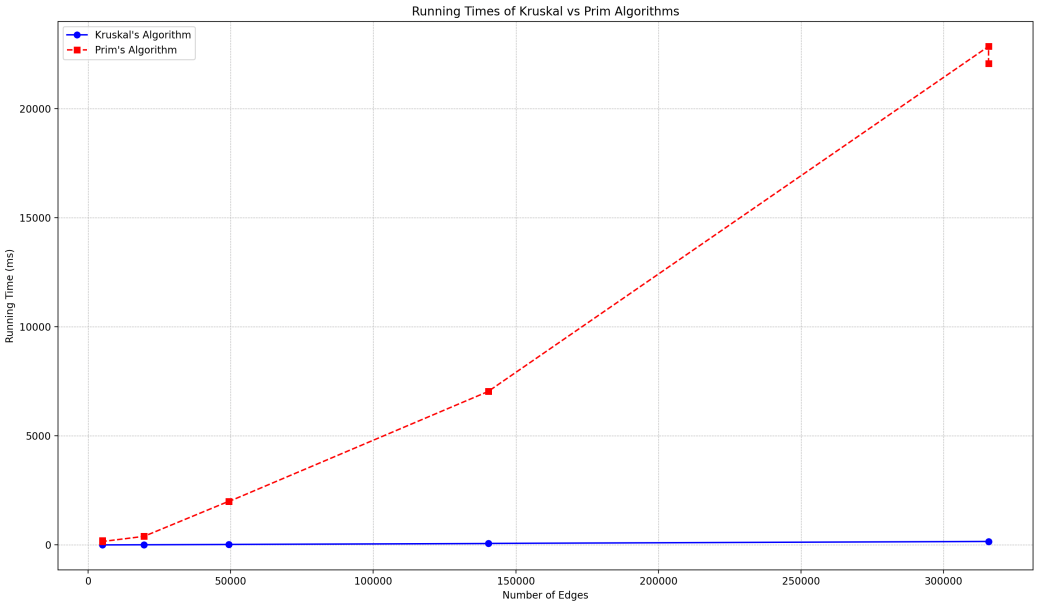
```
amanpandita@10-249-161-161 Hw5 % python3 testcases.py

Test_File       Node_count              Edge_count              Krusal's Algorithm (ms)   Prim's Algorithm (ms)     Total Time taken (ms)
1.txt           1000                    315745                  257.268                   163.923                   4348.590999999999
2.txt           1000                    315745                  254.878                   163.697                   3855.8340000000003
3.txt           1500                    140359                  115.79                    73.417                    1685.885
4.txt           1500                    140359                  110.691                   73.426                    1589.8419999999999
5.txt           2000                    49370                   42.211999999999996        30.203                    586.8050000000001
6.txt           2000                    49370                   37.537                    44.242                    528.267
7.txt           1000                    19607                   15.110999999999999        12.619                    211.189
8.txt           1000                    19607                   15.342                    12.224                    207.641
9.txt           1500                    4994                    4.3                       7.093                     47.114999999999995
10.txt          1500                    4994                    4.471                     7.007000000000001         61.051
amanpandita@10-249-161-161 Hw5 %
```

We can see that when the number of edges cross 5000, that is for the test cases 1,2,3,4,5,6,7 and 8, the **prim's algorithm** tends to become more efficient since all the test files other 9 and 10 have dense graphs whereas 9 and 10 have sparse graphs. The Algorithm performs in O(E) + O(V) + O(E log V) = ***O(E log V) time complexity*** and performs better than kruskal's algorithm for dense graph's. Whereas Kruskal performs better in sparse graphs.



For comparison:
I Also implemented the more common implementation of prims algorithm vs Kruskal algorithm where my prims algorithm was using a ***O(V²)*** time complexity by using an unsorted list as a priority queue.

This was just for comparison and the program I submitted has a prims and Kruskal algorithm running in **O(E log V) time complexity**



2. (10 points) You are driving on a long highway with gas stations at distances $d_1 < \ldots < d_n$ miles from the starting location *S*. Your car can run *M* miles with a full gas tank. You start with a full gas tank and want to reach the final location which is at d_n miles from *S*. How would you choose the gas stations to minimize the number of refueling stops? Argue that no other choice can make fewer stops.

The following greedy approach works:

1. **Initialize variables**: Start your trip from "***S***" with a full tank. Initialize variables like current distance from the starting location, List of gas stations visited, Number of refueling stops.

2. **Sort gas stations**: Check your map to determine the farthest away gas station in your route within ***"M"*** miles.

3. **Traverse gas stations:** Stop at that gas station, fill up your tank and check your map again to determine the farthest away gas station in your route within ***"n"*** miles from this stop.

   Using below logic:

   if current distance from the starting point + M < current station distance: Refuel at the previous gas station since you won't be able to reach the current station without refueling Update the number of stations visited and refueling stops accordingly Reset current distance from the starting point to the distance of the previous gas station ***else:*** Continue driving without refueling. Update Reset current distance from the starting point to the current station distance, ***if i == n-1:*** Reached the final destination without refueling again.

4. Repeat the process until you get to ***"d_n"*** miles.

Proof:

No other choice can make fewer stops because stopping earlier would only decrease the distance covered per tank, potentially increasing the total number of refueling stops. This strategy ensures that you take the fewest possible stops because at each refueling stop, you have maximized the distance that you can travel before the next stop. If there were a solution with fewer stops, it would imply there is a station beyond the range of ***"M"*** miles from the previous stop, which contradicts the approach of driving to the farthest reachable station.

The key assumption here is that the distance between any two consecutive gas stations is not greater than "***M***", as otherwise, it would be impossible to travel between them with a full tank.

3. (10 points) Let 'maximum spanning tree' be defined as a spanning tree with the maximum total weight. Define the *cut property* for maximum spanning tree as follows. Suppose *X* is a set of edges in a maximum spanning tree. Choose a set of vertices *S* such that no edges in *X* cross from nodes in *S* to nodes in *V-S*. Let *e* be the heaviest edge not in *X* that crosses from *S* to *V-S*. Show that *X union {e}* is a subset of a maximum spanning tree.

Let's use Proof by contradiction:

1. Let $G = (V, E)$ be a connected, weighted graph.
2. Let T be a maximum spanning tree of G with edge set $X \subset E$. Consider a cut in G that separates the vertices into two sets S and $V-S$, such that no edge in X crosses this cut.
3. Let e be the heaviest edge across the cut that is not in X.

Let's claim that $X \cup \{e\}$ is a subset of some maximum spanning tree T' of G. For the sake of contradiction, assume that $X \cup \{e\}$ is not a subset of any maximum spanning tree. This would mean that there exists a maximum spanning tree T', different from T, that does not contain e.

Since T' is a maximum spanning tree, it must have the maximum possible weight among all spanning trees of G, which is also the weight of T. Now consider adding e to T'. This action creates a cycle because T' is already a spanning tree. Within this cycle, there must be at least one edge f that connects S to $V-S$ (because e does so and e has just been added to create the cycle).

Since T' did not originally include e, and e is the heaviest edge crossing the cut, the edge f must have a weight less than or equal to the weight of e (otherwise, e would not be the heaviest edge not in X that crosses from S to $V-S$).

Now remove f from the cycle, leaving T'' which is still a spanning tree of G. The total weight of T'' is greater than or equal to the weight of T', because we removed an edge that is lighter than e and added e. This means T'' is at least as heavy as T, implying that T'' is also a maximum spanning tree of G that contains $X \cup \{e\}$, contradicting our initial assumption.

Conclusion:

Therefore, we conclude that $X \cup \{e\}$ must be a subset of some maximum spanning tree T' of G, and this satisfies the cut property for maximum spanning trees.

4. (10 points) A barber shop serves n customers in a queue. They have service times t_1, ..., t_n. Only one customer can be served at any time. The waiting time for any customer is the sum of the service times of all previous customers. How would you order the customers so that the total waiting time for all customers will be minimized? Carefully justify your answer.

Such a problem can be solved by using the **SHORTEST JOB FIRST ALGORITHM**. It is also a type of Greedy Algorithm. It states that, "Given a set of jobs (in this case, customers with their respective service times) to be scheduled, the job with the shortest processing time should be done first".

Justification:

If there is a queue where the longer job is ahead of the shorter job, the shorter job would require to wait more than the longer. Instead, if we change the order of the job execution and the shorter job is executed before the longer job, the longer job would have to wait comparatively less.

Mathematical Proof:

- Let there be two customers, ***A*** and ***B***, with service times ***"tA"*** and ***"tB"*** respectively, and ***"tA" < "tB"***. If ***A*** is served first, ***B*** waits for ***"tA"*** time units. The total waiting time is ***"tA"***. If "***B***" is served first, ***A*** waits for ***"tB"*** time units, and the total waiting time is ***"tB"***. Since ***"tA" < "tB"***, serving ***A*** first reduces the total waiting time.

- Extending this to ***"n"*** customers, it becomes clear that sorting customers in increasing order of their service times minimizes the sum of their waiting times because each customer's waiting time is the sum of the service times of all the customers before them. By minimizing each individual waiting time, we minimize the total.

Proof By Contradiction:

Assume there is an optimal order that is not sorted by shortest processing time first. This means there are at least two customers, **i** and **j**, where **i** is before **j** in the queue, but ***ti > tj***. If we swap ***i*** and ***j***, the waiting time for customer ***i*** is increased by ***`tj`***, but the waiting time for all customers before ***i***, including ***j***, is decreased by ***ti - tj***, which is a net decrease since ***ti > tj***. This contradicts our assumption that we had an optimal order, thus the order must be shortest processing times first.

Therefore, the optimal way to order the customers to minimize the total waiting time is to arrange them in ascending order of their service times, ***t1, ..., tn***.