Student name : Yixiang Fan

Subject code : CSCI803

Student number : 5083898

Email ID : yf874@uowmail.edu.au

Introduction :

MST can be used to optimise the cable connection layout of large-scale offshore wind farms[1]. The vertices in graph represent the wind turbines and the weight of edge is defined as the levelised production cost(LPC), which includes the cable cost, energy production and energy losses related to the edge.

In addition, MST algorithm can be also used to find the connected minimum weighted edges for secondary users(SU) in cognitive radio network(CRN)[2]. The primary users in CRN are licensed users of certain frequency band. The secondary users are unlicensed users that opportunistically access the spectrum when no primary users occupy that frequency band. The vertices in this algorithm. The edge are the channels that can used to connect SU. The weight of edges are the cost setting up a connection between 2 SU.

Besides, MST algorithm also can optimise the ship transport[3]. In this case, vertices represent sea ports and the edges represent the transport routes between two ports. The weight of an edge represents energy consumed to drive the boat between two ports.

In this report, I regard the first case as the background and purpose of my experiments. This is because the matrix generated by generateAdjM.cpp represents a fully undirected weighted graph, which is almost the same as the first case. In the second case, the channels (edge) may change. So it need a dynamic MST algorithm. In the third case, if the target is just to find the smallest weights routes between two ports, then a modified or part of MST algorithm is the best solution. In other words, the MST of a graph may not be the best route for two vertices.

Method :

Brute Force :

At first, I read the size and create appropriate array mat and other elements to store the data. In the first method, brute force, I create an edge structure for each edge and store them in the edge list. Then I try every combination consist of n-1 different edges in the edge list. I adopt iterative function. In each iteration, one edge is selected from the edge list. Of course ,there are some criterion of selecting edges. For example, this edge must haven't been selected; its start vertice is in the set of chosen vertices and its end vertice is out of the set. I use a for loop to try every possible edge in a iteration. Each time when I get a combination of edges, I compare it with the previous minimum combination and choose the minimum one. In the end , I can get the minimum spanning tree.

Prim Matrix :

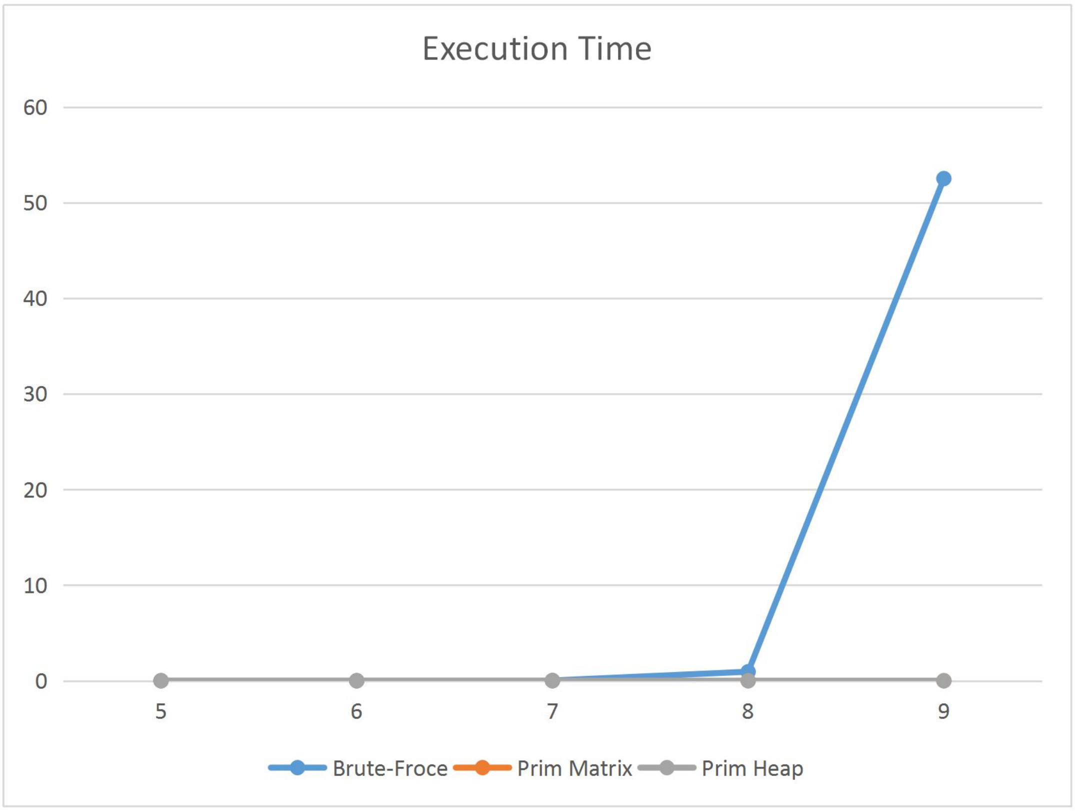
This method has been clarified quite clearly in lecture. Firstly, create 2 arrays, nearest and mindist. In each iteration, read the value from mat in the column related to the vertice just put into the set which consists of the chosen vertices. For example, if vertice 2 was chosen in the last iteration, read the values, which are the weights of edges, in column 2 of the matrix which is , actually , a float array, mat, in my code. This float array, mat , was created in the brute force method. If the values of that column are smaller than the corresponding weights in the array mindist, then update mindist and nearest. After that, find the smallest weight in mindist and record corresponding vertices , the start and the end. If the number of chosen edges < n - 1, then use the end vertice mentioned above as the parameter to start next iteration. After n-1 times iteration, I can get the MST.

Prim Heap :
In this part, I use an associated list to store graph and use a heap to find the shortest edge. Firstly, create an associated list representing the graph. Then push the edge list of vertice 1 into an empty heap. After that, I use a while loop to find the MST. In each loop , I pop the top edge of the heap. The start vertice of this edge is in the set of chosen vertice certainly. If the end vertice of this edge is in the set, then discard this edge, update the heap and pop again. After getting an appropriate edge(start vertice is in the set and end vertice is not), record this edge and go to the next loop. When all vertice are chosen, this algorithm is finished. The MST is the edges recorded in each loop.

Result analysis

Brute Force :
Time complexity : O(n^4)
Space complexity : O(n^2)
Prim Matrix :
Time complexity : O(n^2)
Space complexity : O(n^2)
Prime Heap :
Time complexity : O(n^2logn)
Space complexity : O(n^2)

In the brute force, the number of edges is about n^2. Assume m = n^2 , then I need to try at most C^n_m. This is about O(n^4) which is quite large. As to Prim matrix, the outer loop has n times. So does the inner loop. So the time complexity of Prim Matrix is O(n^2). As to Prim Heap, the outer loop is O(n). The inner loop is n * logn. This is because in each inner loop, I need to push n edges at most into the heap. Each pushing cost logn. So Prim Heap is O(n^2 * logn) totally.



The execution time of Prim Matrix and the one of Prim Heap are almost the same, so the line of Prim Matrix is covered by the line of Prim Heap. The unit of vertical axis is second. The execution time of brute force increases rapidly from 8. When the size of graph is 9, the execution time of brute force is over 50s. And when it comes to 10, my laptop breakdown.

The following table shows the MST built by each method based on different size of graph.

Size of graph : 5		
Brute Force	Prim Matrix	Prim Heap
1-2 1.8 1-5 3.5 5-4 2 4-3 2.9 weight : 10.2	1-2 1.8 1-5 3.5 5-4 2 4-3 2.9 weight : 10.2	1-2 1.8 1-5 3.5 5-4 2 4-3 2.9 weight : 10.2
Size of graph : 6		
Brute Force	Prim Matrix	Prim Heap
1-2 2 2-5 3.3 1-4 4.7 4-3 2.3 4-6 3.9 weight : 16.2	1-2 2 2-5 3.3 1-4 4.7 4-3 2.3 4-6 3.9 weight : 16.2	1-2 2 2-5 3.3 1-4 4.7 4-3 2.3 4-6 3.9 weight : 16.2
Size of graph : 7		
Brute Force	Prim Matrix	Prim Heap

1-2 2.2 2-6 2.5 6-3 4.5 2-4 3.8 4-7 3.9 1-5 4.7 weight : 21.6	1-2 2.2 2-6 2.5 2-4 3.8 4-7 3.9 6-3 4.5 1-5 4.7 weight : 21.6	1-2 2.2 2-6 2.5 2-4 3.8 4-7 3.9 6-3 4.5 1-5 4.7 weight : 21.6
Size of graph : 8		
Brute Force	Prim Matrix	Prim Heap
1-2 2.4 2-5 2.7 5-6 4.7 2-3 4.2 3-8 4.4 3-4 5 4-7 4.7 weight : 28.1	1-2 2.4 2-5 2.7 2-3 4.2 3-8 4.4 5-6 4.7 3-4 5 4-7 4.7 weight : 28.1	1-2 2.4 2-5 2.7 2-3 4.2 3-8 4.4 5-6 4.7 3-4 5 4-7 4.7 weight : 28.1
Size of graph : 9		
Brute Force	Prim Matrix	Prim Heap
1-2 2.6 2-8 5.5 8-7 6.3 8-5 3.9 2-4 3 4-6 5.1 6-3 4.8 1-9 4.6 weight : 35.8	1-2 2.6 2-4 3 1-9 4.6 4-6 5.1 6-3 4.8 2-8 5.5 8-5 3.9 8-7 6.3 weight : 35.8	1-2 2.6 2-4 3 1-9 4.6 4-6 5.1 6-3 4.8 2-8 5.5 8-5 3.9 8-7 6.3 weight : 35.8

Conclusion :

Brute force , Prim’s algorithm based on matrix and Prim’s algorithm based on heap can solve the MST problem. The efficiency of these three methods are almost the same when the graph is small. While the graph increase to certain extend, the execution time of brute force will rocket incredibly. But the execution time of Prim’s algorithm based on matrix and that of Prim’s algorithm based on heap keep stable in the experiments. This means Prim’s algorithm is more universal than brute force on solving MST problem.

References

[1]Peng, H, Weihao, H, Cong, C, & Zhe, C 2016, 'Optimisation of offshore wind farm cable connection layout considering levelised production cost using dynamic minimum spanning tree algorithm', IET Renewable Power Generation, vol. 10, no. 2, pp. 175-183. Available from: 10.1049/iet-rpg.2015.0052. [1 October 2016].

[2]Murmu, MK, Firoz, AM, Meena, S, & Jain, S 2016, 'A Distributed Minimum Spanning Tree for Cognitive Radio Networks', Procedia Computer Science, vol. 89, no. Twelfth International Conference on Communication Networks, ICCN 2016, August 19- 21, 2016, Bangalore, India Twelfth International Conference on Data Mining and Warehousing, ICDMW 2016, August 19-21, 2016, Bangalore, India Twelfth International Conference on Image and Signal Processing, ICISP 2016, August 19-21, 2016, Bangalore, India, pp. 162-169. Available from: 10.1016/j.procs.2016.06.030. [1 October 2016].

[3]Antoš, K 2016, 'The Use of Minimal Spanning Tree for Optimizing Ship Transportation', [Korištenje minimalnog razgranatog stabla radi optimizacije brodskog prijevoza], Nase More, vol. 63, pp. 81-85.