Third Year B.Tech CSF : Sem-5

Design and Analysis of Algorithm

Exercise 11/10/2021

PRN : 2019BTECS00043
Name: Krishna Babar Mali
Batch: T2

Exercise Task 1:

Find Complexity of Prims and Kruscal's algorithms of O notations?

Ans The complexity of the Kruscals algorithm is : $O(E \log(V))$ where E is number of vertices inside the graph
The reason for this Complexity is due to the sorting cost.

The Complexity of Prims algorithm is $O(E+V.\log(V))$ where E is the number of edges and V is number of vertices inside the graph

Q1 Kruskal's algorithm can return different spanning trees for same input graph G, depending on how it breaks ties when the edges are sorted into order. Show that for each minimum spanning tree T of G there is a way to sort the edges of G in Kruscal's algorithm so that the algorithm returns T

Ans We should start by sorting the edges in of G in non-descending order. In addition we would want that among the edges of same weight, the edges which are

contained in T are placed in first positions.

The claim here is that Kruscals algorithm will return T when run on E if sorted in the above mentioned manner

Proof:

T: $e_1 \leq e_2 \leq e_3 \leq \ldots\ldots \leq e_m$

T': $e'_1 \leq e'_2 \leq e'_3 \leq \ldots\ldots \leq e'_m$

weight of e_i = Weight of e'_i where $1 \leq i \leq m$

Since the algorithm always places edges of T first the edges of T will be choosen to T'.

Q2 Suppose that we represent the graph $G = (V, E)$ as an adjacency matrix Give a simple implementation of Prim's algorithm for this case that runs in $O(V^2)$ time

Ans

We would need to sort through V twice. On line 8 rather than adjacency list we would loop from 1 to V

Here's what it would look like

```
MST - PRIM (G, r) /* G = (V, E)*/
1] For i = 1 to V
2]    Dist[i] = ∞
3]    Pred[i] = ∞
4]    Dist[r] = 0
5]    Create minimum priority queue Q for indexes of
      vertices
6]    If Q not empty
7]    i = EXTRACT - MIN (Q)
```

8] For j=1 to V
9] If m[i,j] = 1
10] If j is in Q and weight(i,j) < dist[j]
11] Pred[j] = weight(i,j)

Q3 For a sparse graph G = (V, E), where |E| = Θ(V) is the implementation of prims algorithm with a Fibonacci heap asymptotically faster than the binary heap implementation? What about for a dense graph where |E| = Θ(V²)? How must the sizes |E| & |V| are related for the fibonacci-heap implementation to be asymptotically faster than the binary heap implementation?

Ans Consider the running times of prims algorithm implemented with either a binary heap or a Fibonacci heap

Suppose |E| = Θ(V) then the running times are
Binary O(E log V)
Fibonacci: O(E + V log V) = O(V log V)

If |E| = Θ(V²) then
Binary: O(E log V) = O(V² log V)
Fibonacci: O(E + V log V) = O(V²)

The Fibonacci heap beats the binary heap Implementation of prims algorithm when |E| = ω(V) since O(E + V log V) = O(V log V) ↯ for |E| = O(V log V) but O(E log V) = ω(V log V)
For |E| = ω(V) for |E| = ω(V log V)
The Fibonacci version clearly has a better running time than the ordinary version

Page No.
Date

Q4 Suppose that all edge weights in a graph are integers in the range from 1 to |V|. How fast can we make Kruskal's algorithm run? What if the edge weights are integers in the range from 1 to W for some constant W?

Ans If w is a constant we can use Counting Sort

Sorting the edges : $O(E \lg E)$ times

$O(E)$ operations on a disjoint-set forest taking $O(E \alpha(V))$

The Sort dominates and hence the total time is $O(E \lg E)$
Sorting using counting sort when the edges fall in the range $1, \ldots |V|$ yields $O(V+E) = O(E)$ time sorting
The total time is then $O(E \alpha(V))$.

If the edges fall in the range $1, \ldots, W$ for any Constant W we still need to use $\Omega(E)$ time for sorting and the total running time can't be improved further.

Q5 Suppose that all edge weights in a graph are integers in the range from 1 to |V|. How fast can you make Prims algorithm run? What if the edge weights are integers in the range from 1 to W for some constant W?

Ans The running time of Prims algorithm is composed
$O(V)$ initilization
$O(V \cdot$ time for EXTRACT-MIN$)$
$O(E \cdot$ time for DECREASE-KEY$)$

If the edges are in the range $1, \ldots |V|$ the Van Emde Boas priority queue can speed up EXTRACT-MIN and

DECREASE-KEY to $O(\log\log V)$ thus yielding a total running time of $O(V \log\log V + E \log\log V) = O(E \log\log V)$

If the edges are in the range from 1 to W we can implement the queue as an array $[1 \ldots\ldots W+1]$ where the ith slot holds a doubly linked list of the edges with weight i.e.

- The $(W+1)^{st}$ slot contains ∞ EXTRACT-MIN now runs in $O(W) = O(1)$ time since we can simply scan for the first non empty slot and return the 1^{st} element of the list.

- DECREASE-KEY runs in $O(1)$ time as well since it can be implemented by moving an elements from one slot to another.