Exercise – 11/10/2021

2019BTECS00035

Q1) Kruskal's algorithm can return different spanning trees for the 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 Kruskal's algorithm so that the algorithm returns T.

Q1] → suppose that we want to pick T as our minimum spanning Tree, then to obtain tree with kruskal's algorithm. we will order the edges according to their weights but then we will resolve the tie in edge weights by picking edge first if it is contained in the MST and treating all the ~~dee~~ edges aren't in the T as a beig slightly longer even they have same weight. with this order we still be able for find cost of MST. there may be chance of multiple MST possible in this case.

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

Q2] → At each step of algorithm we will add an edge from vertex in tree created so far to next vertex not in the tree, such that this edge has minimum weight

* Implementation,

```
vector <bool> selected (n, false);
min_Edge [0] = 0      // min weight

for (i=0 ; i<n ; i++)
{
      v = -1
      for (j=0 ; j<n ; j++)
      {  if (!selected[j] && (v==-1 ||
            min_Edge[j] < min_Edge[v])
         {  v = j ;
         }
      }
      cost += min_Edge[v];
      selected [v] = true;
}
```

Q3) For a sparse graph $G = (V, E)$ where $|E| = \Theta(V)$, is the implementation of Prim's algorithm with a Fibonacci heap asymptotically faster than the binary-heap implementation? What about for a dense graph, where $|E|=\Theta(V^2)$? How must the sizes $|E|$ and $|V|$ be related for the Fibonacci-heap implementation to be asymptotically faster than the binary-heap implementation?

Q3] → prims Algorithm implemented using binary heap has runtime $O(V + E\log V)$ which in sparse graph $O(V\log V)$.

the implementation of fibonacci heap is

$O(E + V\log V) = O(V + V\log V) = O(V\log V)$

- In the sparse case two algorithms have same time complexity
- In dense case binary heap has

$O((V+E)\log V) = O((V+V^2)\log V)$

$= O(V^2\log V)$

- for fibonacci implementation

$O(E + V\log V) = O(V^2 + V\log V)$

$= O(V^2)$

$\therefore O(V^2) < O(V^2\log V)$

$\therefore$ Fibonacci heap is faster than binary heap for dense graphs.

Fibonacci heap will be faster if $E = \omega(V)$

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

Q4] → Kruskal algorithm uses the ~~d~~ edge weights in sorting order. If the edge weights are integers from range 1 to $|V|$ we can use counting sort to sort edges in $O(V+E)$.

If the edge weights are in range 1 to W. (constant) we can use counting sort to sort edges in $O(W+E)$ time & kruskal algorith will run in $O(V+E+V\log V)$

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

Q5] → For the first case Edmas boas priority queue can be used to get min weight in $O(\log\log V)$ yeilding total running time $O(E\log(\log V))$.

For second case we can use collection of doubly linked list each corresponding to an edge weight. this improves bound to $O(E)$