CS-202

ASSIGNMENT-05

S. SAMARTH REDDY
B19109
GROUP - 25

(1) While computing an MST of a graph, we first sort all the edges and then take them one by one. If we already have paths connecting to the two vertices, then we ignore the current edge and continue the process until all vertices are connected.

It can be clearly se e algorithm, that we take the imum edges between any two consecutive paths, thus the cost to travel from A e minimum as cost to travel is the maximum cost of all roads from A to B.

2

Proof: Every Minimum Spanning Tree (MST) is a Minimum bottleneck Spanning Tree (MBST).

Let T be MST of G(V, E) and T' be it's MBST.

Consider max weighted edge of T and T'.

Case 1: Both edges are the Same. Then, T is a MBST i.e, every MST is an MBST, due to arbitrary choice of G.

Case 2: Both edges are different. weight of T's (max. edge) cannot be greater the weight of T, Since T' is an MBST.

Case 3: Assume T has maximum weighted edge (p, q) whose weight is greater than that of T'.

- Let X ⊂ V in T, that can be reached from p without going to q.
- Let Y ⊂ V in T, that can be reached from q without going through p.
- Since G is a connected graph, there should be a cut edge between X and Y. The only edge that can be added across this cut is the one of minimum weight.
- we know that (p, q) is the only Possible cut edge of minimum weight.

⇒ However, we have a MBST T' with lesser weight than $w(p,q)$.

- This a contradiction, as MBST is itself a spanning tree and it must have an edge across this cut. And it will be of lesser weight than $w(p,q)$

⇒ ∴ Our assumption was wrong and the only possibility is Case 1, ie, max weight edge of both T and T' are the same.

∴ It has been shown that every MST is a MBST

Now, to proove that every MBST is not a MST, we consider a counter example. Consider a weighted triangle with 2 edges of weight 2 and one edge of weight 1.

Clearly, the MST ~~be~~ will be the path with weight 1 and either path with weight 2. But a tree formed by the weight 2 edges will be MBST with bottleneck cost (=2) same as an any MST, but has strictly more total cost, and therefore is not a MST.

(3) DFS can be used to verify whether a graph has cliqs or not by identifying if the directed graph G has atleast one strongly connected component of size ≥ 2 using Kosaraju's algorithm

The Kosaraju's algorithm to detect cliqs can be defined as follows:-

1) Initialise all vertices of the graph G as unvisited
2) Run a sequence of DFS over graph G which will return vertices in increasing order of their exit times "t_{out}"
3) Reverse direction of all arcs to obtain the transposed graph G^T.
4) Run a series of DFS in order determined by decreasing order of their exit times "t_{out}". Every set of vertices reached after the next search, will be the next strongly connected component.
5) Now, if the count of strongly components having size ≥ 2 is atleast one, then the graph has cliqs.

(4) (a) Inorder to calculate the out degree of vertices in a directed graph,

we visit each vertex, which takes $O(V)$ time, and then calculate the length of the corresponding adjajency list containing the adjacent edges, which takes $O(E_i)$ time for the i^{th} edge).

$\therefore$ Total Time Taken $= O(V) + \sum_{i=1}^{V} O(E_i) = \underline{O(V+E)}$.

(b) To calculate the indegree of each vertex, we need to scan the entire adjacency lists just like the previous part and maintain an array ~~of~~ which stores the indegree of the vertices. Each time we come across a vertex in the adjacency list, we update or increment the value by 1 in the corresponding place of the array. This also takes the same time as the previous part, ie $\underline{\text{Time Taken} = O(V+E)}$

(5) Given an adjacency matrix, we can check in constant time whether a given edge exists. To discover whether there is an edge $(u, w) \in G^2$, for each possible intermediate vertex v we check whether (u,v) and (v,w) exist in $O(1)$ time [(u,v) exists if $a_{uv} = 1$ in adjacency matrix]

Since there are at most n intermediate vertices to check, and n^2 pairs of vertices to ask about, this takes $O(n^3)$ total time.

There is another way. We can calculate adjacency matrix of the square graph by computing the square of the adjacency matrix, which is why this graph is called the square graph.