Graph- algorithms

Sunday, 22 October 2023 7:43 AM

① Topological Sorting of a Directed Acyclic Graph

— It is linear ordering of vertices such that for every directed edge uv, vertex u comes before v.

— Topological sorting is not possible for a graph if it not a DAG.

— In a DAG, we can have vertices which are unrelated to each other because of which we can order them in many ways.

Naive $O(V!)$ → worst case is when there are no edges in graph

Topological Sorting using DFS

— In DFS, we push a vertex then recursively call DFS for adjacent vertices.

In topological sorting, we need to print vertex before adjacent vertex.

DFS → $S2, 3, 1, 0, 4, 5, 2, 3, 1, 4$

TC → $S + O2, 3, 1$

Modification of DFS for Topo-Sorting:

1) Use a temp stack
2) Push point vertex immediately
3) Recursively call topo sorting for all adj vertices, then push it to stack
4) Finally print content of stack

Note: A vertex is pushed to stack only when all its adjacent vertices (and their adjacent vertices and so on) are already in the stack.

$O(V+E)$ time.

$O(V)$ extra space for stack

```
def topSort(self,V,adj):
    stack = []
    visited = [False]*len(adj)
    for node in range(len(adj)):
        if visited[node] == False:
            self.dfs(node,visited,stack)
    return stack[::-1]
def dfs(self, v,visited,stack):
    visited[node] = True
    for node in adj[v]:
        if visited[node] == False:
            self.dfs(node, visited,stack)
    stack.append(node)
```

Kahn's Algorithm for topological sorting (BFS manner)

Steps 1) Compute indegree for each vertex and initialize count of visited nodes as 0.

2) Pick all vertices with indegree as 0 and add them to queue.

3) Remove vertex from queue and then:

a) Increment count of visited nodes by 1

b) Decrease indegree by 1 for all neighboring nodes

c) If indegree of neighboring nodes is reduced to 0, enqueue it.

4) Repeat step 3 until queue is empty

5) If count of visited nodes is not equal to no. of nodes then topological sort is not possible (contains cycle)

Applications of Kahn's Algo

— Course segmenting

— Management of software dependencies

— Scheduling tasks

— Data processing

— Circuit design

```
def topSort(self,V,adj):
    indegree = [0]*len(adj)
    q = deque()
    visited = 0, ans = []
    for node in range(len(adj)):
        for neighbour in adj[node]:
            indegree[neighbour]+=1
    for node in range(len(adj)):
        if indegree[node] == 0:
            q.append(node)
    while q:
        node = q.popleft()
        visited += 1
        ans.append(node)
        for neighbour in adj[node]:
            indegree[neighbour]-=1
            if indegree[neighbour]==0:
                q.append(neighbour)
    if visited != len(adj):
        cycle present
    else: return ans
```

② Minimum Spanning Tree

— defined as spanning tree that has minimum weight among all possible spanning tree.

Properties

— no. of vertices (V) in graph and spanning tree is same.

— fixed no. of edges in spanning tree which is equal to one less than total no. of vertices ($E = V-1$)

— Spanning tree should not be disconnected, as there is only one single source of component not more than that.

— spanning tree should be acyclic, which means there would not be any cycle.

— total cost (or weight) is defined as sum of edges

— can be many possible spanning tree

Algorithms

Kruskal's Minimum Spanning Tree

— greedy approach (network design, clustering and data analysis)

Prim's Min Spanning Tree

— greedy (image segmentation, routing)

③ Kruskal Min Spanning Tree

① Sort all edges in increasing order of their weight

② Initialize : MST = {}, res = 0

③ Do following for every edge 'e' while MST size does not become V-1

Ⓐ If adding e to MST does not cause a cycle

MST = MST ∪ {e}

res += e.weight

④ Return res

Sorted Edges

B C — 3

A C — 5

A B — 6 X

C D — 7

D E — 10

B C — 12 X

MST

res = 2+5+7+10 = 25

Implementation

Simple : Create a separate graph for MST

Graph representation for Kruskal's Algo

Class Edge:

```
def __init__(self,u,v,w):
    self.u = u
    self.v = v
    self.w = w
```

— Using disjoint set Data structure

↳ to find if cycle is being formed quickly

```
V = 5
parent = [i for i in range(V)]      — O(V)
rank = [0 for i in range(V)]        — O(V)
def Kruskal(all):
    all.sort(key=lambda e:e.w)      — O(ElogE)
    res = 0
    for e in all:
        ve = find(e.u)
        ve = find(e.v)
        if ve != ve:
            res += e.w
            union(ve, ve)
    return res
```

Time

$O(ElogE + V + E\alpha(V)) = O(ElogE)$

Space — $O(V)$

If we are not allowed to modify the input array, then space $O(V+E)$

1-3 → 3

2-3 → 4

1-2 → 5

0-2 → 8

0-1 → 10

2-4 → 12

3-4 → 15

Edge 1-3

res = 3

Edge 2-3

res = 2+4

Edge 1-2 X

Edge 0-2

res = 2+4+8

Edge 0-1 X

Edge 2-4

res = 2+4+8+12

Edge 2-4 X

④ Prim's Algorithm for MST

— group of edges that connect two sets of vertices in a graph is called cut-in graph theory.

— at every step of Prim's Algo, find a cut and pick minimum weight edge from the cut, and include vertex in MST set.

— idea is to maintain a two set of vertices that contains vertices already in MST, and other contains vertices not yet included.

At every step, it considers all edges that connect the two set and picks the minimum edge from these edges.

After picking the edge it moves other endpoint of edge to set containing MST.

⑤ Dijkstra's Algorithm (find shortest distance from single to all other vertices)