Monday, November 2021 2:31 PM

```
void DAGSetMax (DAG* max, v) {
    for (int i=0; i<v.size(); i++) {
        max[i] = 0;
        v[i] = 0;
    }
}

void updateMax (max, u, v) {
    int num_u = max[u];
    int num_v = max[v];

    if (num_u > num_v) {
        max[v] = max[u] + 1;
        v[v] = max[u];
    } else {
        max[v] = max[v] + 1;
        v[v] = max[v];
    }
}
```

Union with Path Compression

max(5)

max[5] = max [max[5]]

→ avoids inequality time max of 5, set each 5 to point to its grandparent

(Analyze holding two path lengths).

when union, we are union vertices, union the tallest the first vertex depending on the tall of v.

```
max (max, i) {
    while (max[i] != i) {
        max[i] = max[max[i]];
        i = max[i];
    }
    return i;
}
```

BFS → (level log (n))

BFS can be used for find the shortest path for any two nodes.

unweighted graph

parent dist level 1 1 0 2 2

→ Dijkstra

[v_1 v_2 v_3]

[v_1 v_2 v_3]

[v_1 v_2 v_3]

↑ nodes ↑ edges ⇒ space

→ Visiting a weighted graph.

Shortest Path Algorithms

1. Dijkstra Algorithm → single source shortest path algo
2. Bellman Ford
3. Floyd Warshall → APSP (pair shortest path algo)

Dijkstra Algorithm

→ greedy algo.

→ relaxed in 1st first time shortest path. any long long indicate infinity.

Relaxation

if (d(u) + c(u,v) < d(v))
    d(v) = d(u) + c(u,v)

d(s) = 0
c(u,v) ≥ 0
d(v) = ∞

0 → 4 → ∞
d(v) = 4

a → b 4
a → c 6
a → d 5
a → e 8
a → f 8

Pseudocode

Dijkstra → BFS + Priority Queue

visited = []
q = Priority Queue
q.push (make_pair (0, s))
while (! q.isEmpty()) {
    top = q.pop();
    dist = top.first;
    node = top.second;
    visited [node] = true;
    for every child v of node {
        if (visited [v] = false)
            continue;
        if (dist + wt (node → v) < wt (v,child))
            q.push (make_pair (dist + wt(node,v), child));
    }
}
}

Applications

1. shortest path
2. negative cycle detection
3. find location in a map

Discussion

TC → (V+E) log V

Bellman Ford Algorithm

→ Dynamic programming

→ works for +ve weights as well.

Each edge is relaxed V-1 times

V = 3
↓
Each edge will be relaxed 2 times

edgelist → (s,b), (s,c), (b,d), (b,e), (c,b), (c,d), (d,e), (e,b), (e,d)

```
function setBellmanFord (G, s)
    for each vertex V in G
        dist[v] = ∞
        prev[v] = null
        dist[s] = 0
    for each vertex in G
        for every edge (u,v) in G,
            tempDist = dist[u] + edgeWeight(u,v)
            if ( tempDist < dist[v])
                dist[v] = tempDist;
                prev[v] = u;

    for each edge (u,v) in G,
        if ( dist[v] + edgeWeight(u,v) < dist[u])
            "negative cycle exists"
```

A negative weight cycle is a cycle with weight that can be a negative number.

TC → O(E V) (Bellman Ford)