MSTKruskals

August 7, 2020

1 Minimum Spanning Tree - Kruskal's Algorithm

https://opendsa-server.cs.vt.edu/ODSA/Books/CS3/html/Kruskal.html

1.0.1 Table of Contents

1.1 Kruskal's Algorithm

- another algorithm for finding Minimum Spanning Tree (MST)
- also a greedy algorithm
  - makes the optimal choice at each step as it attempts to find the overall optimal way to solve the entire problem
- use Parent Pointer Tree to find and join disjoint sets
- algorithm steps:
  1. partition the set of vertices into V disjoint sets
     - each set contains one vertex
  - process the edges in order of weight (sort, or use min heap priority queue)
  - if an edge connects two vertices in dfferent disjoint sets (FIND):
    * add the edge to the MST
    * combine the sets (UNION)
  - if the graph is connected, MST will have $|V| - 1$ edges

1.1.1 visualize Kruskal's algorithm here: https://opendsa-server.cs.vt.edu/ODSA/Books/CS3/html/Kruskal.html

[1]:
```
#include <iostream>
#include <vector>
#include <queue> // priority_queue
#include <climits> // sizes of integral types
#include <sstream>
#include <list>
#include <algorithm>
```

```
using namespace std;

using iPair = pair<int, int>;
```

[2]:
```
// general Parent-Pointer Tree implementation for UNION/FIND
class ParPtrTree {
  private:
    vector<int> parents; // parent pointer vector
    vector<int> weights; // weights for weighted union
  public:
    // constructor
    ParPtrTree(size_t size) {
        parents.resize(size); //create parents vector
        fill(parents.begin(), parents.end(), -1); // each node is its own root␣
↪to start
        weights.resize(size);
        fill(weights.begin(), weights.end(), 1);// set all base weights to 1
    }

    // Return the root of a given node with path compression
    // recursive algorithm that makes all ancestors of the current node
    // point to the root
    int FIND(int node) {
        if (parents[node] == -1) return node;
        parents[node] = FIND(parents[node]);
        return parents[node];
    }

    // Merge two subtrees if they are different
    void UNION(int node1, int node2) {
        int root1 = FIND(node1);
        int root2 = FIND(node2);
        // MERGE two trees
        if (root1 != root2) {
            if (weights[root1] < weights[root2]) {
                parents[root1] = root2;
                weights[root2] += weights[root1];
            }
            else {
                parents[root2] = root1;
                weights[root1] += weights[root2];
            }
        }
    }

    string toString() {
        string nodes = "nodes:\t";
```

```
        string prts = "parents:\t";
        for (int i=0; i < this->parents.size(); i++) {
            prts += to_string(this->parents[i]) + '\t';
            nodes += " \t " + to_string(i);
        }
        return prts + "\n" + nodes;
    }
};
```

[3]:
```
// a structure to represent a weighted edge in graph
struct Edge
{
    int src, dest, weight;
    // for min priority queue
     bool operator<(const Edge &other) const {
        return this->weight > other.weight;
    }
};
```

1.2 Representing Graph using Vector of Edge

[4]:
```
// a structure to represent undirected
// and weighted graph
struct Graph
{
    // V -> Number of vertices, E -> Number of edges
    int V, E;
    // graph is stored in a min heap priority_queue
    // Kruskal algo requires working with edges with smallest to highest weight
    priority_queue<Edge, vector<Edge> > edges;
    // constructor
    Graph(int v, int e) {
        V = v;
        E = e;
    }

    void addEdge(int u, int v, int w) {
        edges.push({u, v, w});
    }
};
```

1.2.1 Kruskal's Algorithm Implementation

[5]:
```
// function to construct MST using Kruskal's algorithm
// returns the total weight of MST
// edges forming MST are stored in MST vector
```

```
int KruskalMST(Graph& graph, vector<iPair> & MST)
{
    if (graph.E == 0)
        return 0;

    int numMST = graph.V;  // initially V disjoint classes
    ParPtrTree unionfind(graph.V);
    int weight = 0;

    while (numMST > 1 && !graph.edges.empty())
    {
        // pick the smallest edge
        Edge edge = graph.edges.top();
        graph.edges.pop();
        int x = unionfind.FIND(edge.src); // root of src
        int y = unionfind.FIND(edge.dest); // root of dest
        // if src and dest nodes are in different sets
        if (x != y)
        {
            int u = edge.src;
            int v = edge.dest;
            // add weight
            weight += edge.weight;
            // the ordering is not required, but...
            if (u > v) swap(u, v);
            // add u->v edge to MST
            MST.push_back({u, v});
            // combine equiv classes
            unionfind.UNION(u, v);
            numMST--; // one less MST
        }
    }
    return weight;
}
```

1.2.2 Test Kruskal's Algorithm

[6]:
```
// represent undirected graph shown in above diagram
// A->0, B->1, C->2, D->3, E->4, F->5
Graph graph(6, 8);
vector<iPair> MST;
```

[7]:
```
// 8 undirected edges
graph.addEdge(0, 2, 7);
graph.addEdge(0, 4, 9);
graph.addEdge(1, 2, 5);
graph.addEdge(1, 5, 6);
```

```
graph.addEdge(2, 3, 1);
graph.addEdge(2, 5, 2);
graph.addEdge(3, 5, 2);
graph.addEdge(4, 5, 1);
```

[8]:
```
int wt;
```

[9]:
```
wt = KruskalMST(graph, MST);
```

[10]:
```
cout << "total cost of MST = " << wt << endl;
cout << "MST edges:\n";
for(auto &p:MST)
    cout << char(p.first+65) << "-" << char(p.second+65) << '\n';
```

```
total cost of MST = 16
MST edges:
C-D
E-F
C-F
B-C
A-C
```

1.3 Analysis of Kruskal's Algorithm

- dominated by the time required to process the edges
- if path compression and weighted union is used, union/find takes nearly constant time
- total cost $\Theta(|E|log|E|)$ in the worst case when nearly all edges must be processed
- most often need to process only about $|V|$ edges
  - so, cost is $\sim \Theta(|V|log|E|)$ in the average case

1.4 Comparison with Prim's Algorithm

- if heap (priority queue) is used and the graph is sparse, cost is $\Theta((|V| + |E|)log|E|)$
- if graph is dense, cost can be $\Theta(|V|^2 + log|E|) = \Theta(|V|^2 log|V|)$

1.5 Exercises

1. Minimum Spanning Tree problem: https://open.kattis.com/problems/minspantree

- A Feast For Cats - https://open.kattis.com/problems/cats
  - Hint: Use Kruskal's
  - if M >= C + TotalMST Weight -> yes!
- Island Hopping - https://open.kattis.com/problems/islandhopping
  - Hint: distance between two points is the weight (float)
- Lost Map - https://open.kattis.com/problems/lostmap
  - much faster compared to Prim's
- Driving Range - https://open.kattis.com/problems/drivingrange
  - Hint: last edge that formed MST

```
[ ]:
```