LECTURE #12

Not on Final: P vs NP, Halting problems

Could be on Final: Random select algorithm, something that could have been on first midterm...

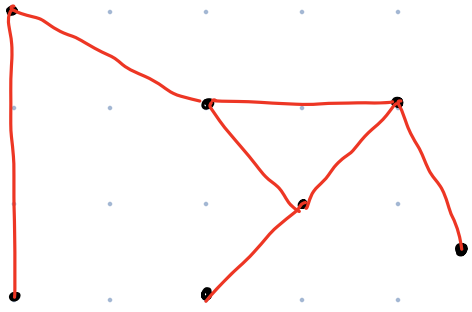
ASSIGNMENT #7:

Be careful not to return anything that leads to privacy leaks...

ASSIGNMENT #9:

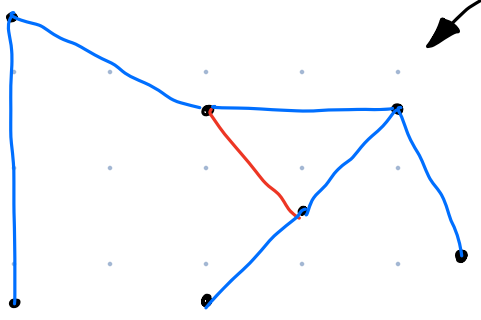
Is EC & replaces the lowest scoring program...

Spanning Trees:



connected graph!

- Any connected graph has a spanning tree!!!



Spanning tree is a tree that is a subgraph of a connected graph that touches every node of the graph G.

Minimum Spanning Tree: Applies to a graph with weighted edges. The minimum weight subgraph in which all nodes

are connected.

Basic Idea:

$A = \{\}$ // Solution set of edges

While A is NOT a minimum spanning tree,

find an edge (u, v) that is safe for A

$A = A \cup (u, v)$

- How do we find a safe edge? thats what both algorithms do differently!
- A cut $(S, V-S)$ is a partition into disjoint sets S, $V-S$
- Any edge (u, v) either crosses cut or respects cut.
- A cut respects A if no edge in A crosses the cut
- An edge is a light edge crosses cut with minimum weight.

Greedy Algorithms:

- Kruskal's $O(E \log(V))$
- Prims $O(E \log(V))$ as "implemented by us" (2in heap + adjacency list.) priority queue = 2in heap...

$O(V^2)$ if done with adj matrix

$O(E + V \log V)$ if done with fibonnachi heap

KRUSKALS:

- needs a disjoint set...

graph

```
kruskals (G) {
    A = { };
    for each node v ∈ V {
        make_set(v);
    }
    sort E in increasing order;
    for each edge (u, v) from the sort (in order) {
        if (find_set(u) != find_set(v)) {
            A = A ∪ (u, v);      // add (u, v) to solution
            Union(u, v);         // update disjoint set
        }
    }
}
```

PRIMS:

starting

graph node

```
Prims (G, S) {
    for each v ∈ V {
        δ[v] = ∞ ;      // distance
        p[v] = ∅ ;      // prent
        PQ.insert(v);   // indexed by distance...
    }

    PQ.decrease key (s, ∅);  // start node distance zero...
    δ[s] = 0;  //zero

    While (!PQ.empty()) {
        U = PQ.extractMin();
        for each node v ∈ Adj[u] {
            if (PQ.contains(v) && weight(u,v) < δ[v]) {
                P[v] = u;
                δ[v] = weight(u,v);
                PQ.decrease key (v, δ[v]);
            }
        }
    }
}
```

Initialization