CSE 101, Winter 2018

Discussion Section
Week 5

February 5 - February 12

Topics

1. Minimum spanning trees
2. Prim’s algorithm
3. Kruskal’s algorithm
4. Binary search the answer
5. Greedy algorithms

Minimum Spanning Tree

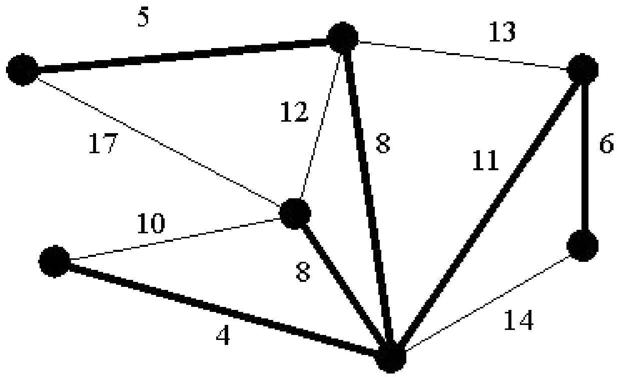
A minimum spanning tree (MST) of an undirected weighted graph G = (V, E) is a subgraph G’ = (V’, E’), such that:

- V’ = V (covers all the nodes in G)
- G’ is connect and doesn’t have any cycles (G’ is a tree)
- The total sum of all weights in G’ is as small as possible

Example:

The cost of the MST is

5 + 4 + 8 + 8 + 11 + 6 = 42



Minimum Spanning Tree

Statement. You are given an undirected weighted graph G = (V, E). Output MST of this graph.

There are two popular greedy algorithms that solve this problem: Prim’s algorithm and Kruskal’s algorithm.

Prim's algorithm

1. Pick any node s in V. Initialize G' = (V' = {s}, E' = {})
2. Among all edges (u, v) in E, such that u belongs V' and v doesn't belong V', choose the one that has the smallest weight.
3. Add v to V' and (u, v) to E': G' = (V' add {v}, E' add (u, v)}
4. If |V'| < |V| go to step 2.
5. G' is a MST of G.

Can be implemented in $O(|V|^2)$, $O(|E|\log|V|)$ or in $O(|V|\log|E|)$

(Example on the board)

Kruskal’s algorithm

1. Initialize G’ = (V’ = V, E’ = {})
2. Sort all edges in E in an increasing order of their weights.
3. For every (u, v) in E (in an increasing order of their weights)
    a. Add edge (u, v) to G’, if G’ won’t have any cycles afterwards

Can be implemented in $O(|E| \log |V|)$ with Disjoint-Set-Union data structure

(Example on the board)

Binary search the answer

Recall question from the midterm: you are given an undirected weighted graph G = (V, E), two nodes s and t, and capacity L. You can go from node u to v, if there is an edge (u, v) and its weight is no greater than L. Determine whether there is a valid path from s to t.

Solution. Run DFS/BFS on the graph starting from s and ignore all edges with edge weights greater than L. If, at some point, we visit t, then there is a valid path from s to t.

Binary search the answer

Now, let's find find the minimum possible L, such that there is a valid path from s to t in the graph G.

We can answer whether we can get from s to t in G for a fixed L. Let's say we have some function f(L, G, s, t) which returns true, if there is a valid path from s to t, and false otherwise.

Can we find minimum L using this function?

Binary search the answer

Let’s find the range for L. If s = t, then the answer is 0. If s != t, then optimal value for L will be equal to some edge weight (think why). Let M be the maximum of all edge weights.

We could run our function f(L, G, s, t) for all integer values

L: 0 <= L <= M. The moment the function returns “true”, we take that value for L as an answer. The overall complexity will be O(M(|V| + |E|)).

Can we optimize this algorithm?

Binary search the answer

We can run our function f(L, G, s, t) for values that are weights of some edges. We can iterate through weights of edges in an increasing order (adding value zero to the beginning) and apply the same algorithm. The complexity will be $O(|E|(|V| + |E|))$. This may be better, if $|E| < M$.

Binary search the answer

The better optimization is the following: we can see the following property of function f:

If f(L, G, s, t) == true, then f(L’, G, s, t) == true for any L’ >= L

If f(L, G, s, t) == false, then f(L’, G, s, t) == false for any L’ <= L

We can perform binary search using parameter L over the range [0, M]

Binary search the answer

```
Ans = M
BinarySearch(G, s, t, Left, Right):
    If Left > Right:
        return
    L = (Left + Right) / 2
    If f(L, G, s, t) == true:
        Ans = L
        BinarySearch(G, s, t, Left, L - 1)
    Else:
        BinarySearch(G, s, t, L + 1, Right)
```

Binary search the answer

The complexity of the binary search is $O(\log M)$. At each step of the binary search, we run function $f(L, G, s, t)$, which runs in $O(|V| + |E|)$.

So, the overall time complexity is $O((|V| + |E|)\log M)$.

Greedy algorithms: exchange method

- Make the decision that gives you the most immediate benefit
- Not always optimal, but can be useful for providing an approximation when solving the exact optimal solution would be prohibitively expensive or very complicated
- Identify greedy parameter - the value you want to increase the most with each iteration
- Figure out how you want to order the elements you will be selecting
- Define your selection procedure