Minimum Spanning Trees

- Raghav Goel

Goal

To understand

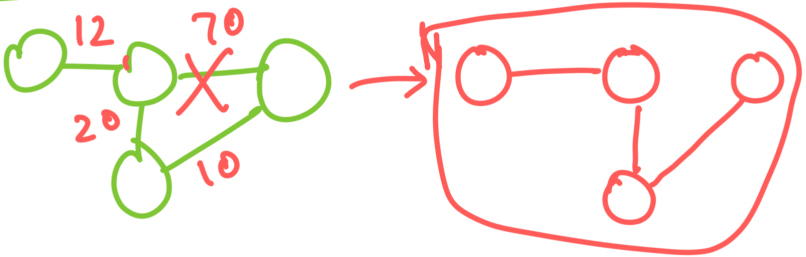
- Minimum Spanning Trees ✓
- Kruskal's Algorithm
- Prim's Algorithm

DSU → merge/unite( )
→ isSameComponent( )

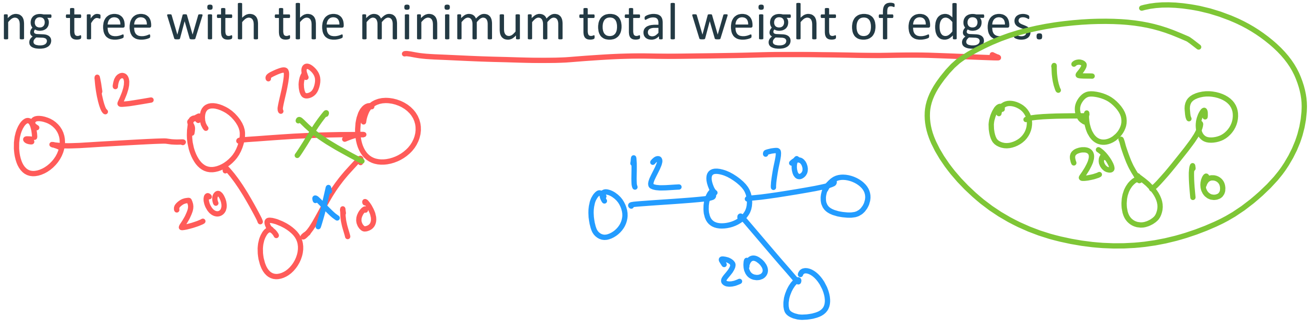
Dijkstra

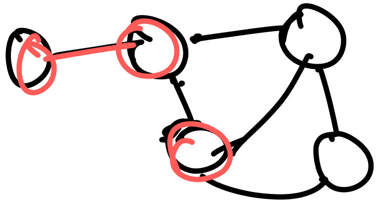
Minimum Spanning Tree

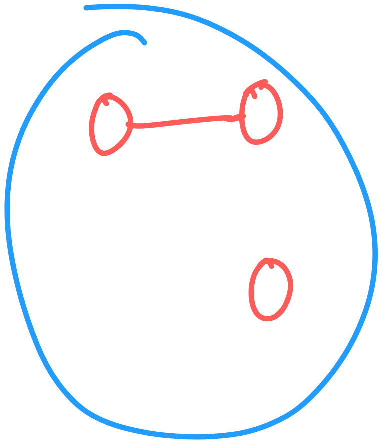
- Spanning Tree:
  - A subgraph that connects all nodes in a graph without forming any cycles.



- Minimum Spanning Tree:
  - A spanning tree with the minimum total weight of edges.



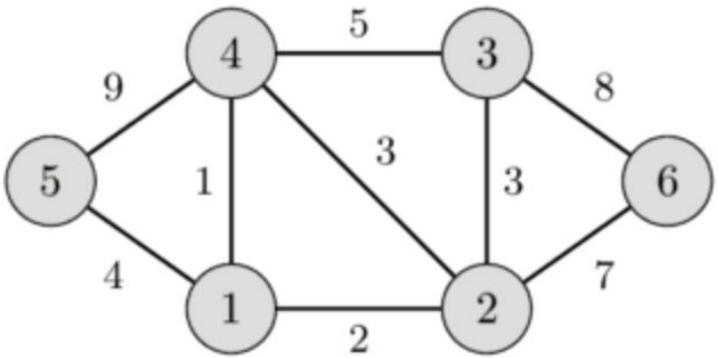




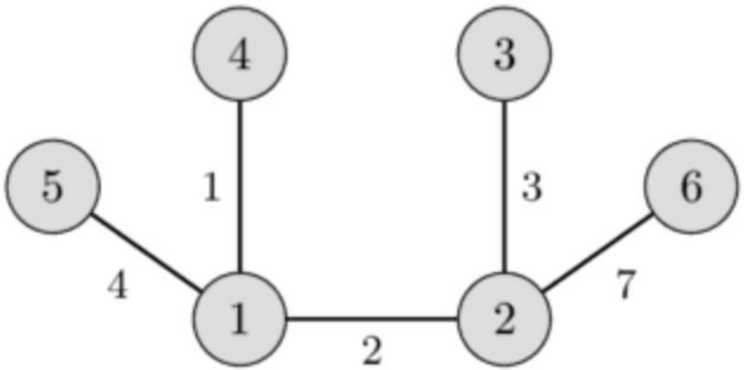
$V' \in V$

$E' \in E$

Example



Graph



Minimum Spanning Tree

Algorithms for finding MST

- Kruskal's Algorithm
- Prim's Algorithm

Kruskal's Algorithm

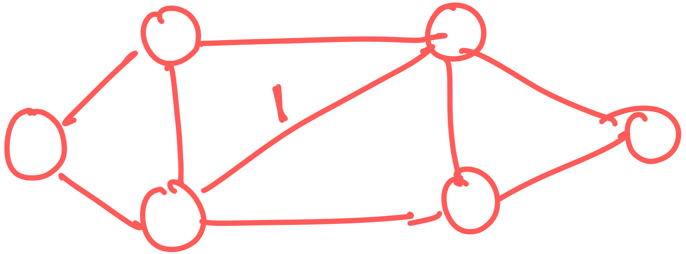
- Core Idea:
  - Greedily select edges in increasing order of weights
  - Avoid creating cycles.

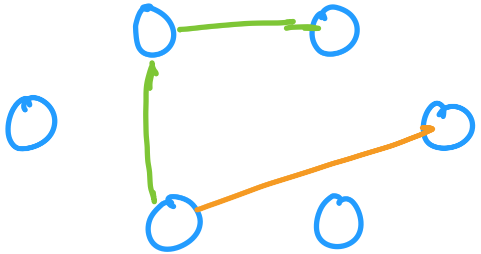
- Complexity Analysis:
  - Time Complexity: O(V + E log E) // sorting edges
  - Space Complexity: O(V + E)

Kruskal's Algorithm

greedily pick smallest weight edge

$O(V)$ 1. Imagine the graph as N isolated nodes. V
2. Sort all the edges in ascending order. $O(E \log E)$
3. Start picking up edges one by one from lower weight to higher weight $O(E)$
   a. If current edge connects A with B, check if both A and B lie in the same component.
   b. How to check if A and B are in same component quickly? DSU → $O(1)$
   c. If both A and B lie in same component, don't use this edge.
   d. If no, use this edge and connect A with B.

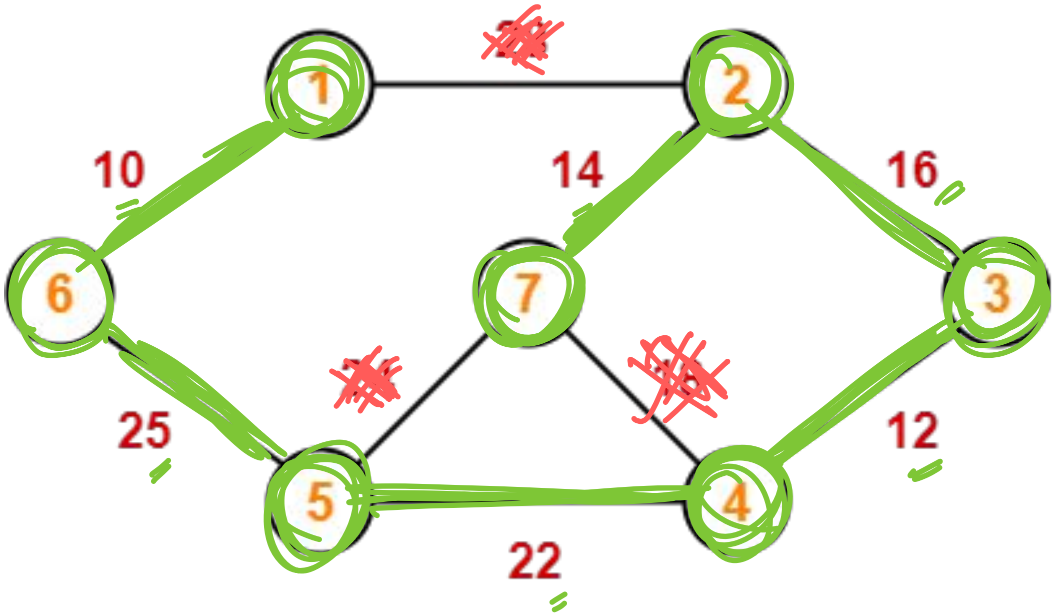




Formal Mathematic Proof →

- CP Algorithms
- Chatgpt

Kruskal's Algorithm Visualise



$10 + 12 + 14 + 16 + 22 + 25$

{
{1, 6, 10},
{3, 4, 12},
{2, 7, 14},
⋮
}

wt

Implementing Kruskal's Algorithm

DSU ...

```cpp
vector<vector<int>> kruskal_algorithm(int n, vector<vector<int>> &edges) {
  vector<vector<int>> mst;
  Dsu dsu(n);
  sort(edges.begin(), edges.end(),
       [](auto &l, auto &r) { return l[2] < r[2]; });
  for (auto &edge : edges) {
    int u = edge[0], v = edge[1], wt = edge[2];
    if (dsu.find(u) == dsu.find(v)) continue;
    dsu.unite(u, v);
    mst.push_back(edge);
  }
  return mst;
}
```

n isolated nodes

sorting

dsu.same(u,v)

O(V)

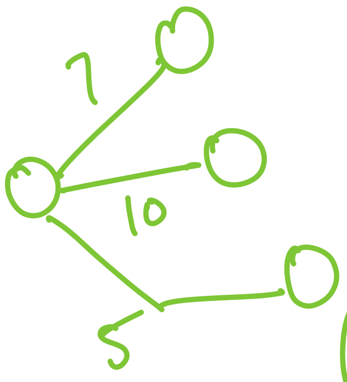
O(E logE)

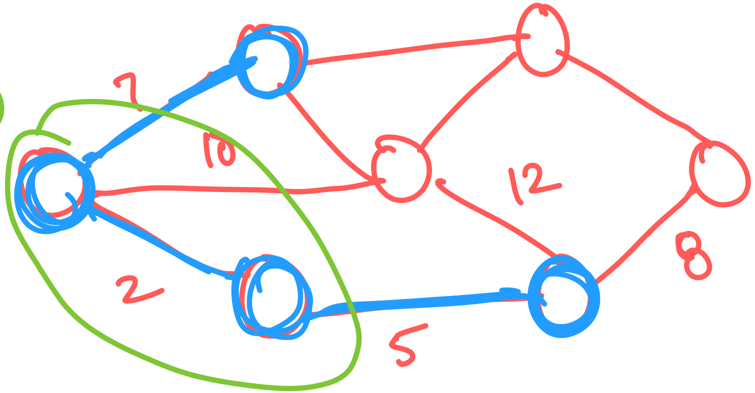
O(E)

Prim's Algorithm

- Core Idea:
  - Start with an arbitrary node. consider it as the component
  - Greedily add the minimum-weight edge connecting the current tree to a node outside the tree. expanding greedily

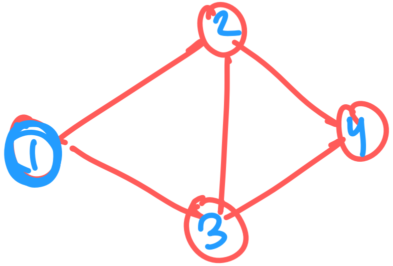
- Complexity Analysis:
  - Time Complexity: O(V + E log V) // using a min-heap
  - Space Complexity: O(V + E)

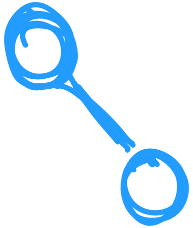




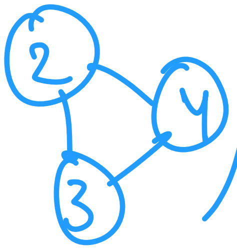
intuitive

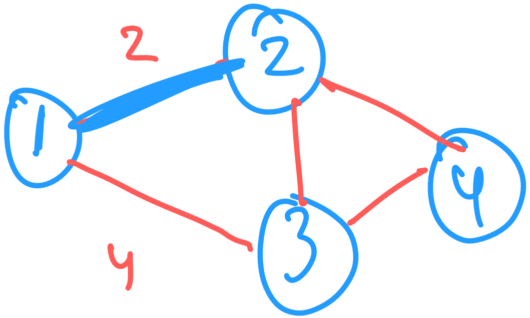
vague

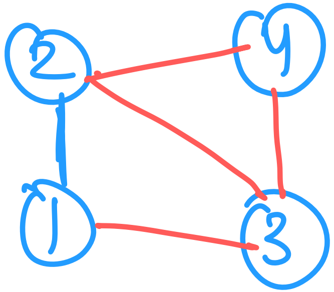


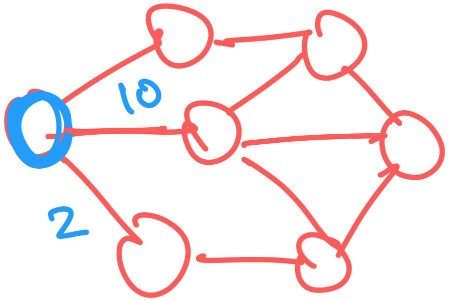


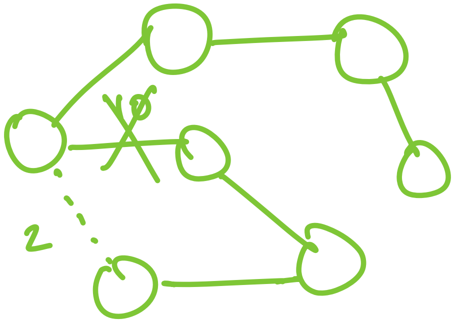
MST of whole graph

= 1 + MST()






10
2


2

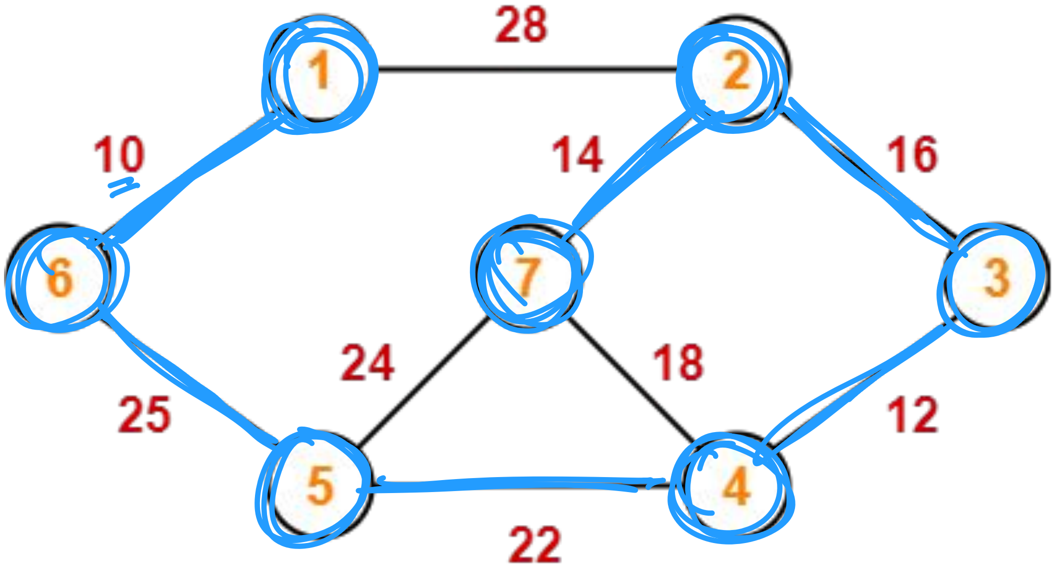
Prim’s Algorithm

1. Pick up any node (A) form the graph.
2. Pick up the smallest edge which connects A to another node B.
3. Pick up the smallest edge that goes out from selected nodes [A, B] and reaches a non-selected node C.
4. Pick up the smallest edge that goes out from selected nodes [A, B, C] and reaches a non-selected node D.
   a. How to find the smallest edge every time? Heap/Set
5. Repeat this process until all the nodes in the graph end up in the selected nodes.

Prim’s Algorithm Visualise

$10 + 25 + 22 + 12 + 16 + 14$

99



Implementing Prim’s Algorithm

same as dijkstra

```
vector<vector<int>> prims_algorithm(vector<vector<pair<int, int>>> &adj) {
  int n = adj.size();
  vector<vector<int>> mst;
  vector<int> vis(n);
  vector<int> min_cost(n, INT_MAX);
  priority_queue<array<int, 3>, vector<array<int, 3>>, greater<array<int, 3>>> pq;
  min_cost[0] = 0;
  pq.push({0, 0, -1});
  while (!pq.empty()) {
    auto [cost, u, p] = pq.top();
    pq.pop();
    if (vis[u]) continue;
    vis[u] = 1;
    if (p != -1) mst.push_back({p, u, cost});
    for (auto [v, wt] : adj[u]) {
      if (!vis[v] && wt < min_cost[v]) {
        min_cost[v] = wt;
        pq.push({wt, v, u});
      }
    }
  }
  return mst;
}
```

→ init

Prim's Algorithm vs Kruskal's Algorithm

- Prim's Algorithm has a better time complexity than Kruskal's Algorithm in case of dense graphs.
- Kruskal's Algorithm is simpler.

Prim $O(V + \underbrace{E \log V}_{\text{heap}})$

Kruskal $O(V + \underbrace{E \log E}_{\text{sorting}})$

$E \leq V^2$

$O(V + E \log V)$

DSU

Prim's TC $\geq$ Kruskal's TC

MST Kruskal

Properties of MST

- A minimum spanning tree of a graph is unique, if the weight of all the edges are distinct. Otherwise, there may be multiple minimum spanning trees.

MST

- e_{max} should be minimized
- all the weights are distinct

Kruskal's → unique MST

Properties of MST

- Minimum spanning tree is also the tree with minimum product of weights of edges. (It can be easily proved by replacing the weights of all edges with their logarithms)

$[w_1 + w_2 + w_3 + w_4 + w_5] \to \min$

$w_i = \log k_i$

$\log a + \log b = \log(ab)$

$[\log k_1 + \log k_2 + \log k_3 + \ldots]$

$[\log(k_1 * k_2 * k_3 * k_4 * k_5)] \to \min$

Properties of MST

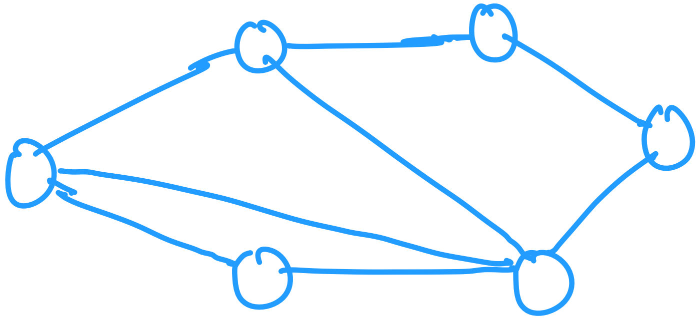
- In a minimum spanning tree of a graph, the maximum weight of an edge is the minimum possible from all possible spanning trees of that graph. [Kruskal?]

Spanning Trees

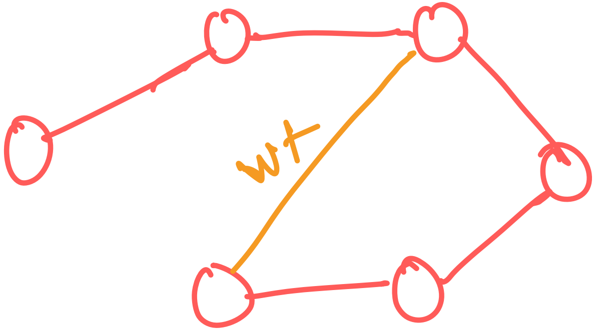
e_{max} 10 12 10 15

no MST in which $e_{max} > 10$

G



T^{MST}



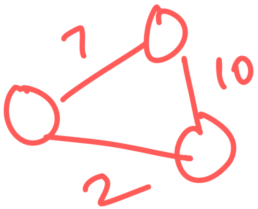
e_{max}

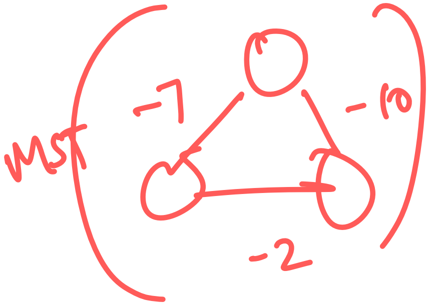
$wt > e_{max}'$

new wt $\rightarrow$ $\boxed{(T) + \underbrace{(wt - e_{max})}_{+ve}}$

Properties of MST

- The maximum spanning tree of a graph can be obtained similarly to that of the minimum spanning tree, by changing the signs of the weights of all the edges to their opposite and then applying any of the minimum spanning tree algorithm.





dijkstra

Problem: Shichikuji and Power Grid

(x_i, y_i)

Shichikuji is the new resident deity of the South Black Snail Temple. Her first job is as follows:

There are n new cities located in Prefecture X. Cities are numbered from 1 to n. City i is located x_i km North of the shrine and y_i km East of the shrine. It is possible that $(x_i, y_i) = (x_j, y_j)$ even when $i \neq j$.

Shichikuji must provide electricity to each city either by building a power station in that city, or by making a connection between that city and another one that already has electricity. So the City has electricity if it has a power station in it or it is connected to a City which has electricity by a direct connection or via a chain of connections.

- Building a power station in City i will cost c_i yen;
- Making a connection between City i and City j will cost $k_i + k_j$ yen **per km of wire** used for the connection. However, wires can only go the cardinal directions (North, South, East, West). Wires can cross each other. Each wire must have both of its endpoints in some cities. If City i and City j are connected by a wire, the wire will go through any shortest path from City i to City j. Thus, the length of the wire if City i and City j are connected is $|x_i - x_j| + |y_i - y_j|$ km.

Shichikuji wants to do this job spending as little money as possible, since according to her, there isn't really anything else in the world other than money. However, she died when she was only in fifth grade so she is not smart enough for this. And thus, the new resident deity asks for your help.

And so, you have to provide Shichikuji with the following information: minimum amount of yen needed to provide electricity to all cities, the cities in which power stations will be built, and the connections to be made.

If there are multiple ways to choose the cities and the connections to obtain the construction of minimum price, then print any of them.

Problem Link - https://codeforces.com/problemset/problem/1245/D

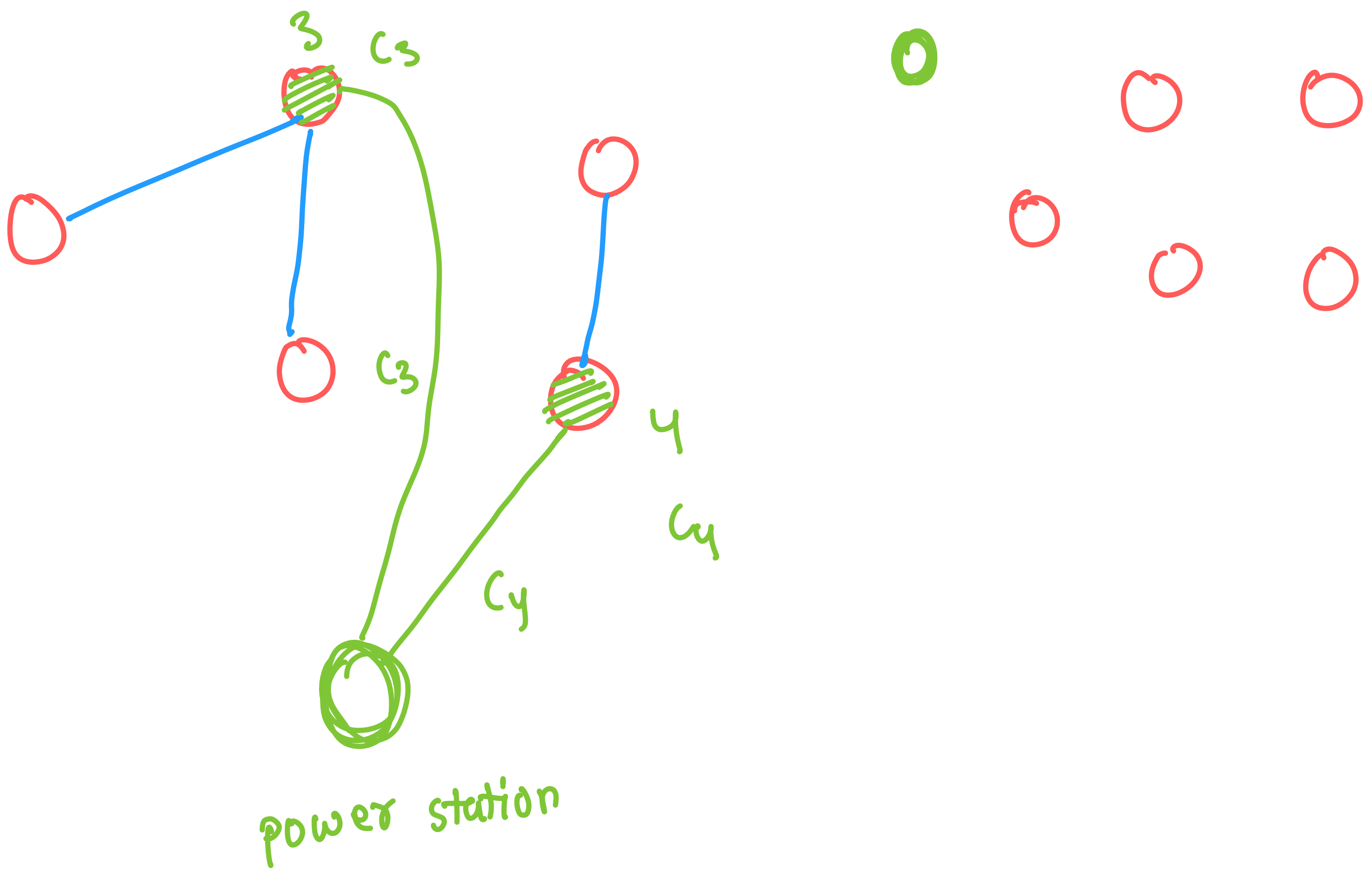
n cities

power station in city i

$\Rightarrow c_i$

connect
city i & j

$$f(i,j) = (k_i + k_j) * (|x_i - x_j| + |y_i - y_j|)$$


3
C3
C3
4
C4
C4
0
power station

$f(i,j) \rightarrow$ city i & j

$c_i \rightarrow$ cost of connecting city 0 & i

$\{\overbrace{\{u, v\}}, \underline{\underline{wt}}\}$

LUV CP