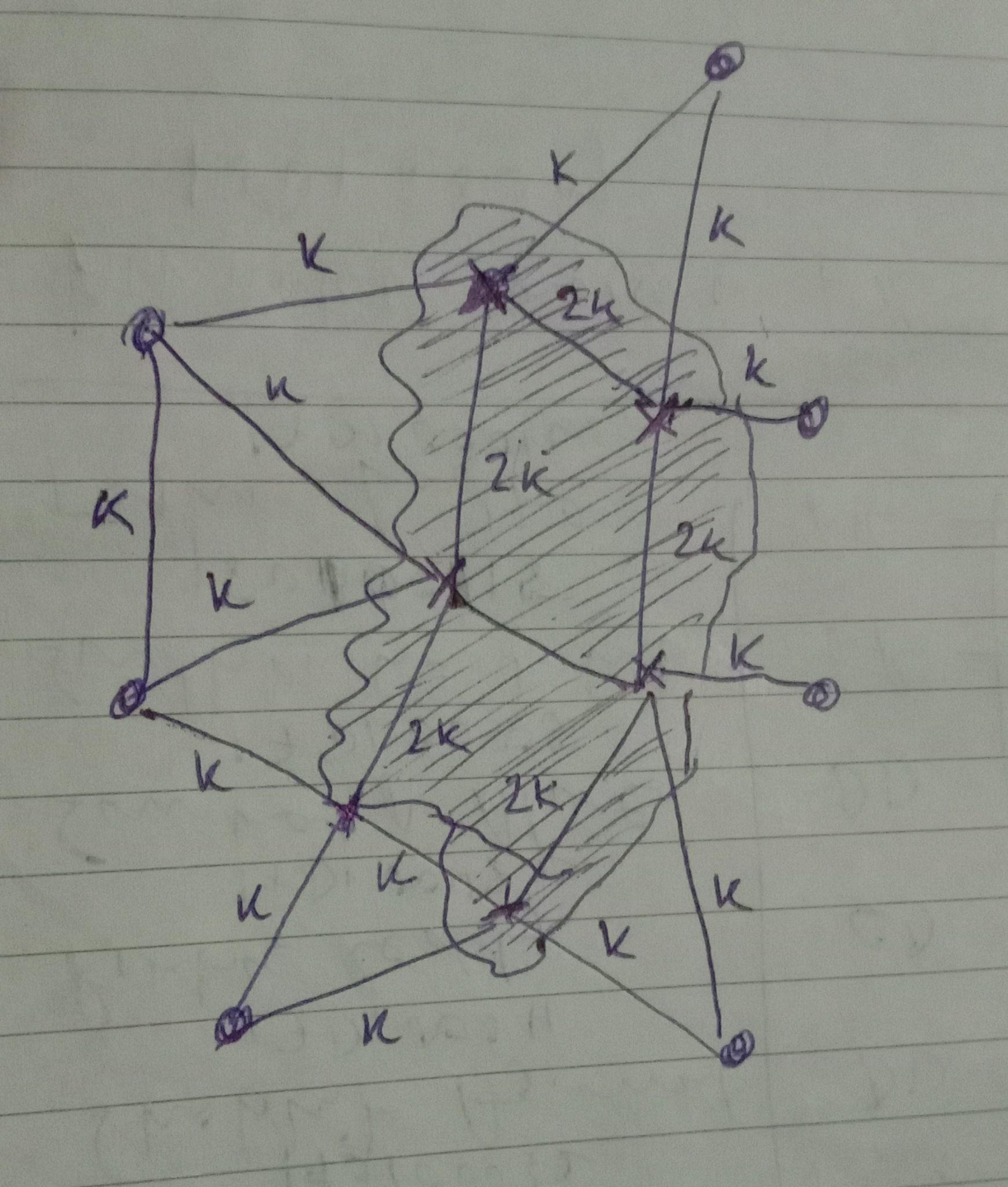
**Transportation Networks**

Considering the network in the image below we can construct a steiner tree i.e a minimum spanning tree with the cost of the network paths. For the construction of the minimum spanning tree we will consider using Bellman Ford method( single source shortest path algorithm ).



Using Bellman Ford in this situation is a better approach because it has a less time complexity compared to the Floyd Marshall technique that takes time with complexity of In the Bellman Ford method we can simply calculate the path from the start vertex to the other vertices as needed. It makes the computation of the cost easier as it does not require the computation of the minimum cost for the other vertices. In the Bellman ford we classify the ports as necessary and optional. Using this, we compute the minimum cost by taking some of the temporary and then choose the temp ports that are resulting in the minimum cost.

Difference between the shortest path techniques is cited below:

https://iopscience.iop.org/article/10.1088/1742-6596/1087/2/022011/pdf

**Sample code:**

/\*

let us take a sample network/graph and then let us create a matrix with the cost and let the points of the graph

be assigned the coordinates in the matrix{ by coordinates I mean that let the matrix be 3\*3 and then divide the

matrix be divided into 9 parts such that the (1,0), (1,1), etc are the coordinates of the mid point of the divided

squares}. The cost will be mentioned in each of the individual squares

\*/

#include<bits/stdc++.h>

using namespace std;

#define R 3 // define the number of rows in the matrix made by the cost of the path length of the graph

#define C 3 // define the number of columns in the matrix made by the cost of the path length of the graph

int min(int x, int y, int z);

// Returns cost of minimum cost path

// from (0,0) to (m, n) in mat[R][C]

int minCost(int cost[R][C], int m, int n)

{

if (n < 0 || m < 0) // base case {to eliminate the condition that the rows and the coloumn number is negative}

return INT\_MAX;

else if (m == 0 && n == 0) // if the destination is the same as the source

return cost[m][n];

else

return cost[m][n] + //use of the bellman ford technique to calculate the MIN cost

min(minCost(cost, m - 1, n - 1),

minCost(cost, m - 1, n),

minCost(cost, m, n - 1));

}

// Min function to calculate the minimum cost of 3 numbers{in this case it is used for the bellman ford technique}

int min(int x, int y, int z)

{

if (x < y)

return (x < z) ? x : z;

else

return (y < z) ? y : z;

}

// testing code

int main()

{

int cost[R][C] = { { 1, 2, 3 }, // random costs initiliazed in the cost function( for the path lenghts of the graph)

{ 4, 8, 2 },

{ 1, 5, 3 } };

cout << minCost(cost, 2, 2) << endl;

return 0;

}

/\*

the above code is an implementation of the simple and uniform network.

This code can be again harnessed for complex networks with

\*/

**Algorithm:**

* The code is simply the computation of the shortest cost from one vertex to another vertex in the graph network.
* The logic used is that the ports( compulsory points) are assigned the centers of the sub-matrices of a bigger matrix.
  + Let there be a square matrix of dimension 3 and the matrix be divided into 9 equal matrices.
  + The 9 equal matrices centers are assigned the ports of the graph network.
  + The cost of the ports traversal is written on the lines connecting the centers of the small matrices.
* The bigger matrix is then filled with the cost values of the ports.
* The shortest distance from a chosen vertex to the other vertices is computed using the Bellman-Ford method.

*Why did I choose the Bellman-Ford method for minimum cost computation?*

*Answer:*

*If the weighted graph contains the negative weight values, then the Dijkstra algorithm does not confirm whether it produces the correct answer or not. In contrast to the Dijkstra algorithm, the bellman ford algorithm guarantees the correct answer even if the weighted graph contains negative weight values.*

* This algorithm only works for simpler networks, needs to be developed further for more complex networks.

ALGORITHM IN STEPS:

1. minCost function:
   1. Compare the rows and columns with 0 and check that both are greater than 0.
   2. If the destination is the source itself then the cost of that matrix is returned i.e. 0.
   3. In the other cases, the summation of the cost of that matrix and the minimum of the upper, right, left matrices' cost is returned.
   4. The above function operates in a recursive way, with the source and the destination coordinates as arguments.
2. A minimum of 3 numbers is computed in the min function using the ternary operator.
3. In the main function, a sample network is taken as input and then it is passed to the minCost function and the starting and the destination matrix coordinate is given as argument in the function.

Filtering the complex network into a more simple network for cost computation

The minimum spanning tree can be made with the help of the Kruskal algorithm. A minimum spanning graph will make the cost computation more simple as it eliminates the paths through which the cost is much higher comparatively.

Q: Now, why might we need the Kruskal Algorithm in this situation?

A: This will help in the final cost computation of the network using the bellman ford method. It is because eliminating some of the paths will reduce the number of paths the algorithm needs to check and thus reducing the running time of the code and eventually making the code more time-efficient

Algorithm:

***1.*** *Sort all the edges in non-decreasing order of their weight.*

***2.*** *Pick the smallest edge. Check if it forms a cycle with the spanning tree formed so far. If cycle is not formed, include this edge. Else, discard it.*

***3.*** *Repeat step#2 until there are (V-1) edges in the spanning tree.*