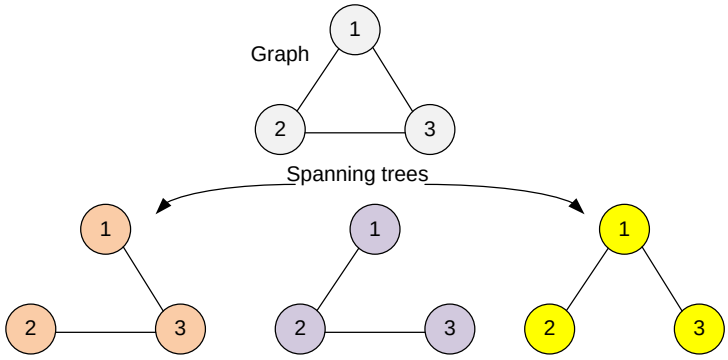
Minimum spanning tree

A ***spanning tree*** is a subset of Graph, which has all the vertices covered with minimum possible number of edges. Hence, a spanning tree does not have cycles and it cannot be disconnected.



We found *three* spanning trees of one complete graph. A complete undirected graph have n^{n-2} number of spanning trees, where n is the number of nodes. In the above example $n = 3$, hence $3^{3-2} = 3$ spanning trees are possible.

General Properties of Spanning Tree

We now understand that one graph can have more than one spanning tree. Following are a few properties of the spanning tree connected to graph.

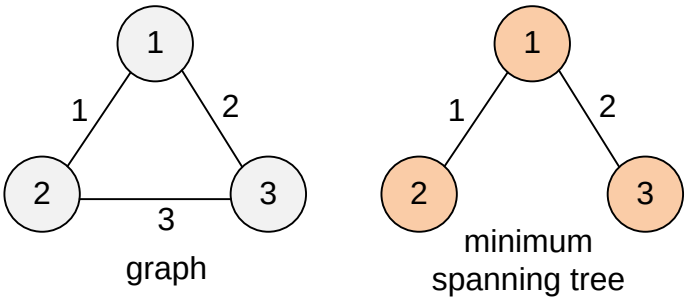
- A connected graph can have more than one spanning tree.
- All possible spanning trees of graph, have the same number of edges and vertices.
- The spanning tree does not have any cycle (loops).
- Removing one edge from the spanning tree will make the graph disconnected, i.e. the spanning tree is minimally connected.
- Adding one edge to the spanning tree will create a circuit or loop, i.e. the spanning tree is maximally acyclic.

Mathematical Properties of Spanning Tree

- Spanning tree has $n - 1$ edges, where n is the number of nodes (vertices).
- From a complete graph, by removing maximum $e - n + 1$ edges, we can construct a spanning tree.
- A complete graph have n^{n-2} number of spanning trees.

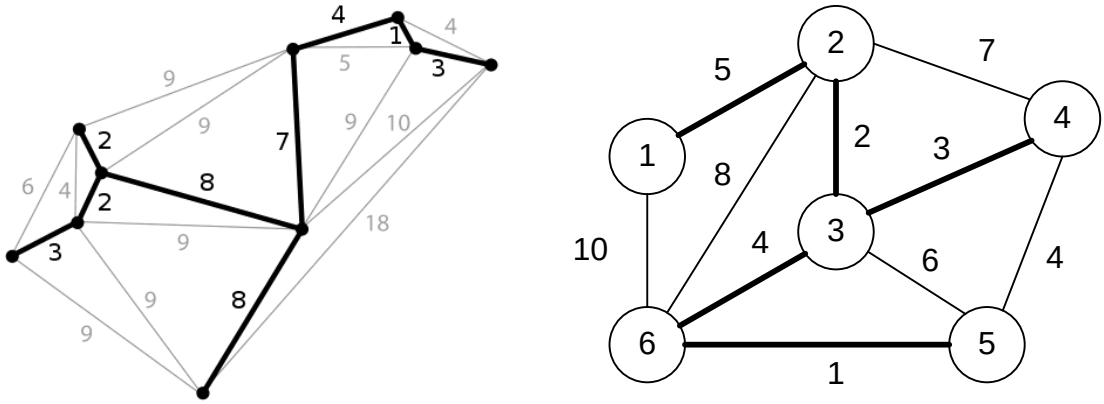
Minimum Spanning Tree (MST)

In a ***weighted graph***, a **minimum spanning tree** is a spanning tree that has *minimum weight* than all other spanning trees of the same graph. In real-world situations, this weight can be measured as distance, traffic load or any arbitrary value denoted to the edges.



More generally, any edge-weighted undirected graph (not necessarily connected) has a minimum spanning forest, which is a union of the minimum spanning trees for its connected components.

Below given graphs and its minimum spanning trees.



Kruskal's algorithm

Assume that we have a connected, undirected graph G = (V, E) with a weight function $w : E \rightarrow R$, and we wish to find a minimum spanning tree for G.

The greedy strategy is captured by the following "generic" algorithm, which grows the minimum spanning tree one edge at a time. The algorithm manages a set of edges A, maintaining the following loop invariant:

Prior to each iteration, A is a subset of some minimum spanning tree

At each step, we determine an edge (u, v) that can be added to A without violating this invariant, in the sense that $A \cup \{(u, v)\}$ is also a subset of a minimum spanning tree. We call such an edge a ***safe edge*** for A, since it can be safely added to A while maintaining the invariant.

GENERIC-MST(G, *w*)
A ← ∅
while A does not form a spanning tree
  do find an edge (*u*, *v*) that is safe for A
    A ← A ∪ {(u, v)}
return A

Kruskal's algorithm is based directly on the generic minimum-spanning-tree algorithm. It finds a *safe edge* to add to the growing forest by finding, of all the edges

that connect any two trees in the forest, an edge (u, v) of ***least weight***. Kruskal's algorithm is a *greedy algorithm*, because at each step it adds to the forest an edge of least possible weight.

The implementation of **Kruskal's algorithm** is like the algorithm to compute connected components. It uses a *disjoint-set data structure* to maintain several disjoint sets of elements. Each set contains the vertices in a tree of the current forest. The operation FIND-SET(u) returns a representative element from the set that contains u. Thus, we can determine whether two vertices u and v belong to the same tree by testing whether FIND-SET(u) equals FIND-SET(v). The combining of trees is accomplished by the UNION procedure.

MST-KRUSKAL(G, w)
A ← ∅
for each vertex v ∈ V[G]
  do MAKE-SET(v)

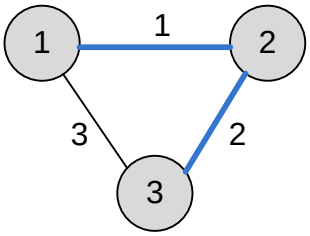
sort the edges of E into nondecreasing order by weight w

for each edge (u, v) ∈ E, taken in nondecreasing order by weight
  do if FIND-SET(u) ≠ FIND-SET(v)
    then A ← A ∪ {(u, v)}
      UNION(u, v)
return A

E-OLYMP **981. Minimum spanning tree** Find the weight of minimum spanning tree for a weighted undirected connected graph.

► In this problem you must find the weight of the minimum spanning tree using Kruskal's algorithm.

The graph given in the sample has a form:



Declare the structure of the graph edge (a pair of vertices and the weight of the edge). Declare the vector of edges *e*.

```
struct Edge
{
  int u, v, dist;
};

vector<Edge> e;
```

Declare an array *parent* used by the disjoint set system.

```
vector<int> parent;
```

The function ***Repr*** finds a representative of the set that contains vertex *n*.

```
int Repr(int n)
{
  while (n != parent[n]) n = parent[n];
  return n;
}
```

Function ***Union*** unites sets that contain elements *x* and *y*.

```
int Union(int x, int y)
{
  x = Repr(x); y = Repr(y);
  if (x == y) return 0;
  parent[y] = x;
  return 1;
}
```

The function ***lt*** is a comparator for sorting edges.

```
int lt(Edge a, Edge b)
{
  return (a.dist < b.dist);
}
```

The main part of the program. Initialize the *parent* array.

```
scanf("%d %d",&n,&m);
parent.resize(n + 1);
for (i = 1; i <= n; i++) parent[i] = i;
```

Read the edges of the graph.

```
e.resize(m);
for (i = 0; i < m; i++)
  scanf("%d %d %d", &e[i].u, &e[i].v, &e[i].dist);
```

Sort the edges in increasing order of their weights.

```
sort(e.begin(), e.end(), lt);
```

Start the Kruskal algorithm that constructs the minimum spanning tree.

```
res = 0;
for(i = 0; i < m; i++)
  if (Union(e[i].u,e[i].v)) res += e[i].dist;
```

Print the weight of the minimum spanning tree.

```
printf("%d\n",res);
```

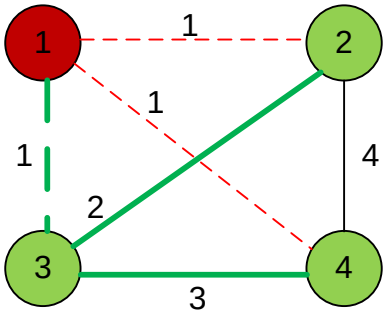
E-OLYMP 7531. Landline telephone network The mayor of RMRCity wants to create a secure landline telephone network for emergency use in case of serious disasters when the city is cut off from the outside world. Some pairs of buildings in the city can be directly connected with a wire telephone line and the municipality engineers have prepared an estimate of the cost of connecting any such pair.

The mayor needs your help to find the cheapest network that connects all buildings in the city and satisfies a particular security measure that will be explained shortly. A call from a building A to another building B may be routed through any simple path in the network (i.e., a path that does not have any repeated building). There are also some insecure buildings that one or more persons with serious criminal records live in. The mayor wants only communications intended for these insecure buildings to reach them. In other words, no communication from any building A to any building B should pass through any insecure building C in the network (where C is different from A and B).

► Find the value of the minimum spanning tree for all buildings except unsafe ones. Then connect each unsafe building to the nearest safe one. The required network cannot be constructed if the graph is disconnected.

Separately, analyze the case when there are no safe buildings in the city. If there is only one unsafe building, then the answer is 0, if there are two unsafe buildings, then the answer equals to the distance between them. If there are more than 2 unsafe buildings, then the desired network does not exist.

Let's look at the first sample. The unsafe house has number 1. Build the MST for the vertices 2, 3, 4 – its value is 5. Connect the house number 1 to MST. The cost of the cheapest network is 6.



Let's consider the second sample. Unsafe house numbers are 1 and 2. Unsafe house number 1 must be directly connected to a safe house, which is impossible for the given network. Therefore, the answer is **impossible**.

E-OLYMP 6576. Road In one country, there aren cities, numbered 1 to *n*. A civil engineer has to build public roads that connect all the cities together, i.e. it must be possible to travel from all cities to any other cities, maybe going through multiple cities. His team has surveyed several routes (candidate road between any two cities). Each

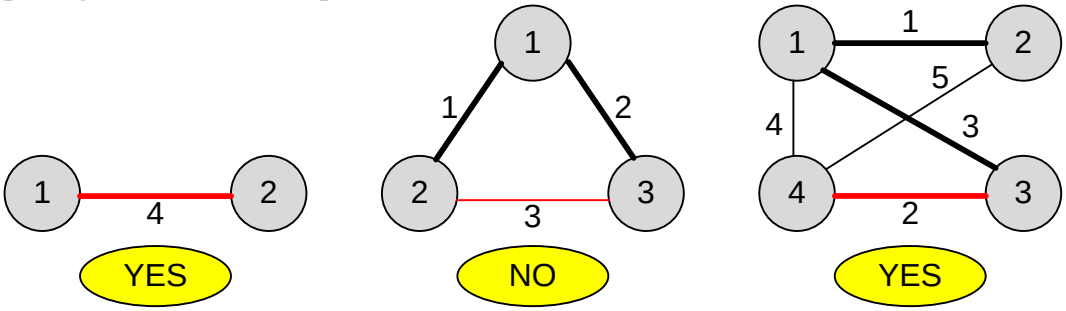
route is a bidirectional connection between two cities. He can build a bidirectional road on the surveyed route for a specific cost (The shorter the route is the cheaper the road).

This engineer has never planned a road system in advance. He would just pick one of the routes based on his preference, and build a road until all the cities are connected.

Right now this engineer is going to build a road from the city *p* to the city *q*. With pressure from the government to reduce the cost, he asks you to write a program to decide, if he should build this road or not. Your program should say yes if the building of this road guaranteed that it can be part of the shortest road system that connects all cities together. Otherwise, your program should say no.

► Run the Kruskal's algorithm on the input graph. In the set *s* include the edges of the graph that belong to the minimal spanning tree (MST). Next, check whether the given edge (*p*, *q*) belongs to the MST. To do this, check whether the pair (*p*, *q*) or (*q*, *p*) is in the set *s*.

Graphs, given in the samples, have the form:



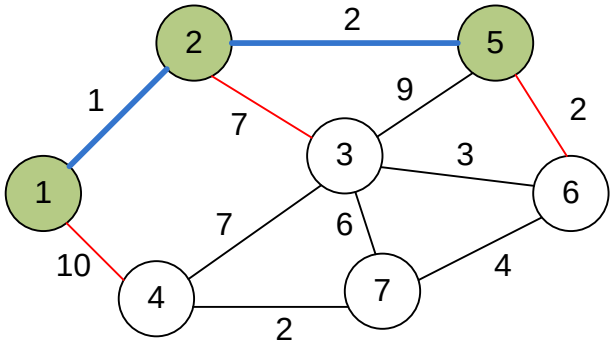
Prim's algorithm

Let V = {1, 2, …, *n*} be the set of graph vertices. Let's construct a set U from which a spanning tree will grow. First, set U = {1} (the MST starts to be built from the first vertex). At each step of the algorithm, an edge of minimum cost(*u*, *v*) is found such that $u \in U$ and $v \in V \setminus U$, after which the vertex *v* is moved from the set V \ U to U. This process continues until the set U will not be equal to V.

In the next pseudocode T denotes the set of edges included in MST.

```
MST_Prim(G)
T = ∅;
U = {1};
while U ≠ V do
begin
   find such edge (u, v) of minimum weight, that u ∈ U and v ∈ V \ U;
   T = T ∪ {(u, v)};
   U = U ∪ {v};
end;
return T;
```

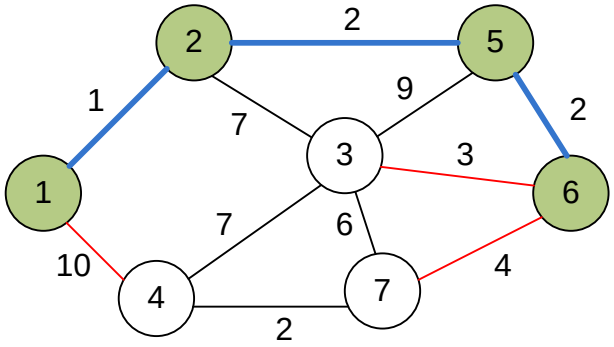
For each not yet selected vertex $v \in V \setminus U$, we'll store the minimum edge that runs to already selected vertex $u \in U$. Then, in order to select the minimum edge at the current step, you just need to look at these minimum edges for each vertex from $V \setminus U$, the asymptotics is $O(n)$.



Let U = {1, 2, 5} – set of vertices of current MST. The minimum edge for

- the vertex 4 is (1, 4) of length 10;
- the vertex 3 is (2, 3) of length 7;
- the vertex 6 is (5, 6) of length 2;
- the vertex 7 no such edges;

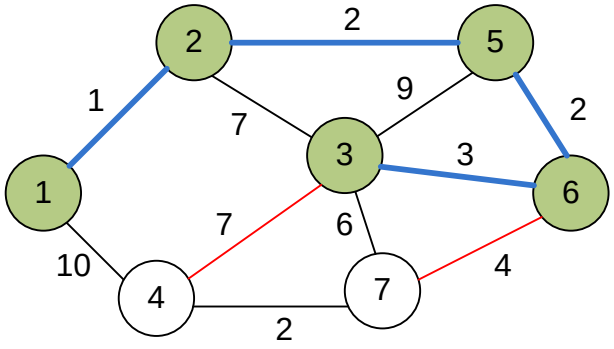
The shortest distance from current MST to vertex from $V \setminus U$ is min{10, 7, 2} = 2, this minimum is achieved for the edge (5, 6). So at the current step we add vertex 6 to U, now U = {1, 2, 5, 6} and edge (5, 6) is added to MST.



The minimum edge for

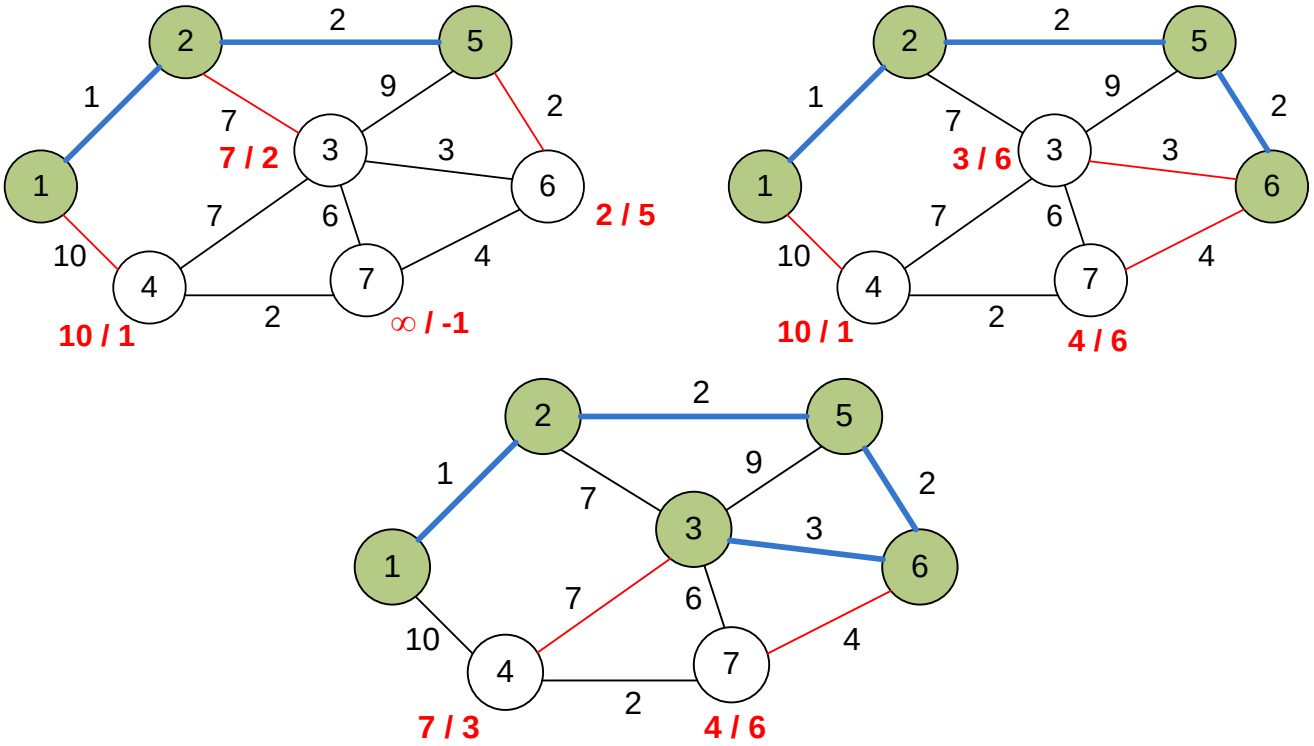
- the vertex 4 is (1, 4) of length 10;
- the vertex 3 is (6, 3) of length 3;
- the vertex 7 is (6, 7) of length 4;

The shortest distance from current MST to vertex from $V \setminus U$ is min{10, 3, 4} = 3, this minimum is achieved for the edge (6, 3). So at the current step we add vertex 3 to U, now U = {1, 2, 3, 5, 6} and edge (6, 3) is added to MST.



Let **min_e**[*i*] stores the weight of the smallest possible edge runing to the vertex *i*, and **end_e**[*i*] contains the number of the vertex from which this smallest edge runs.

Next to each vertex *v* we'll write the labels **min_e**[*v*] / **end_e**[*v*]. At the next step, the vertex from V \ U with the smallest value min_e[*v*] is included to MST.



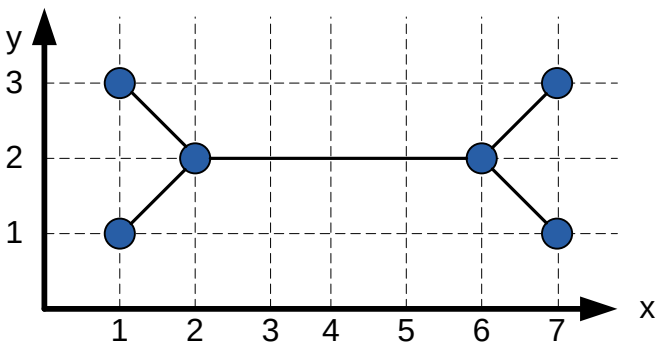
E-OLYMP **2967. Unification day** Byteland has *n* cities, but no one road. The king of the country, Waldemar de Bear, decided to remedy this situation, he wants to connect some cities with roads so that along these roads it will be possible to reach any city from any other city. When construction will be completed, the king plans to celebrate the Unification Day.

Unfortunately, the treasury in Byteland is almost empty, so the king needs to save money and minimize the total length of constructed roads.

► To solve the problem, you need to find the minimum spanning tree (MST). Implement Prim algorithm.

Example

Construct the minimum spanning tree for the graph given in the sample.



The weight of the minimum spanning tree is $4 + 4\sqrt{2} \approx 9.65$.

Declare the arrays. Store the city coordinates in (x[*i*], y[*i*]).

```
#define MAX 5001
int x[MAX], y[MAX];
int used[MAX], min_e[MAX], end_e[MAX];
```

The function ***dist2*** computes the squared distance between cities *i* and *j*.

```
int dist2(int i, int j)
{
  return (x[j] - x[i])*(x[j] - x[i]) + (y[j] - y[i])*(y[j] - y[i]);
}
```

The main part of the program. Read the coordinates of the cities.

```
scanf("%d",&n);
for(i = 0; i < n; i++)
  scanf("%d %d",&x[i], &y[i]);
```

Initialize the arrays.

```
memset(min_e,0x3F,sizeof(min_e));
memset(end_e,-1,sizeof(end_e));
memset(used,0,sizeof(used));
```

The size of MST will be calculated in the *dist* variable.

```
dist = min_e[1] = 0;
for (i = 0; i < n; i++)
{
```

Look for the vertex *v* with minimum value of min_e[*v*] among the vertices that are not yet included in MST (for which used[*v*] = 0).

```
  v = -1;
  for (j = 0; j < n; j++)
    if (!used[j] && (v == -1 || min_e[j] < min_e[v])) v = j;
```

Include the vertex *v* into MST. Add an edge (*v*, end_e[*v*]) to MST.

```
  used[v] = 1;
  if (end_e[v] != -1) dist += sqrt((double)dist2(v,end_e[v]));
```

Recompute the labels for the edges outgoing from *v*.

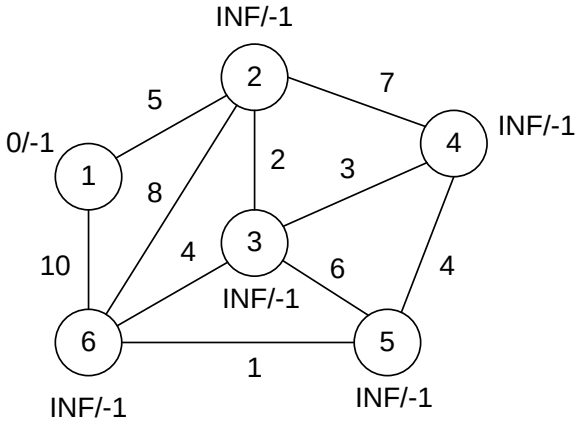
```
  for (to = 0; to < n; to++)
  {
    int dV_TO = dist2(v,to);
    if (!used[to] && (dV_TO < min_e[to]))
    {
      min_e[to] = dV_TO;
      end_e[to] = v;
    }
```

```
    }
}
```

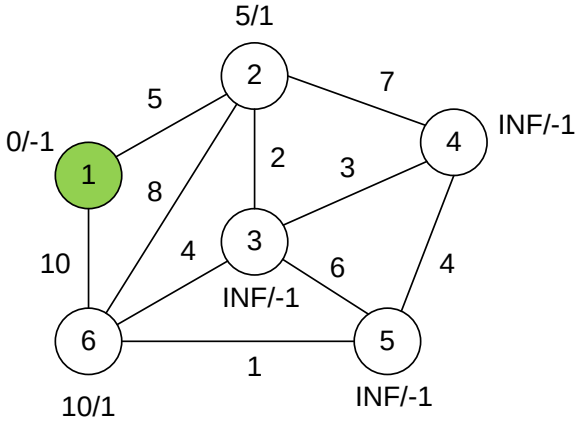
Print the value of MST.

```
printf("%.6lf\n",dist);
```

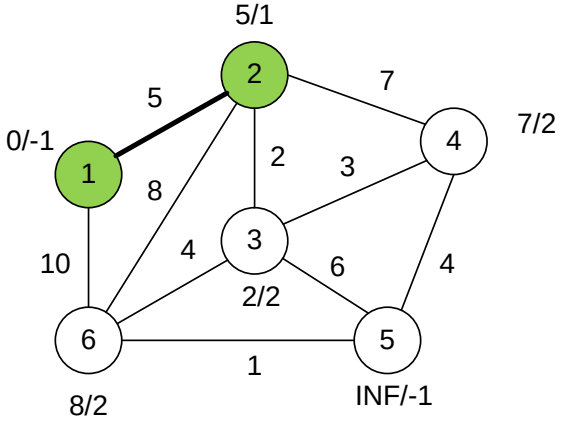
Example. Let's simulate the Prim's algorithm using the following example. Make an initialization:



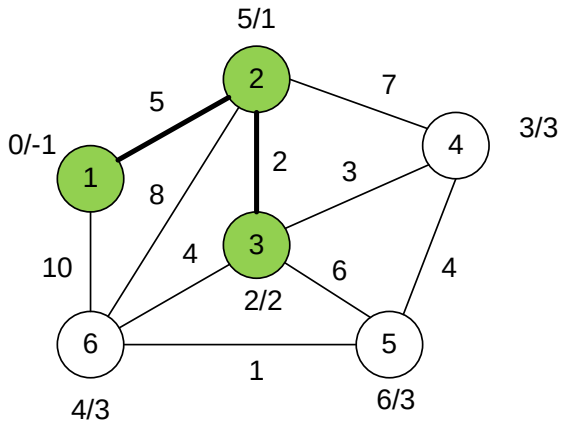
Vertex 1 is included to MST. Recalculate the labels for the edges outgoing from it.



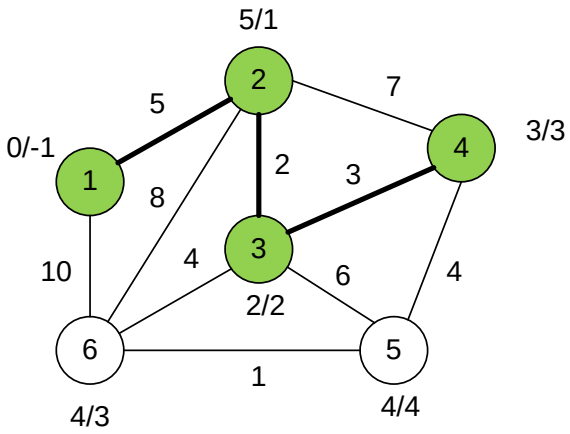
Vertex 2 is included to MST. Recalculate the labels for the edges outgoing from it.



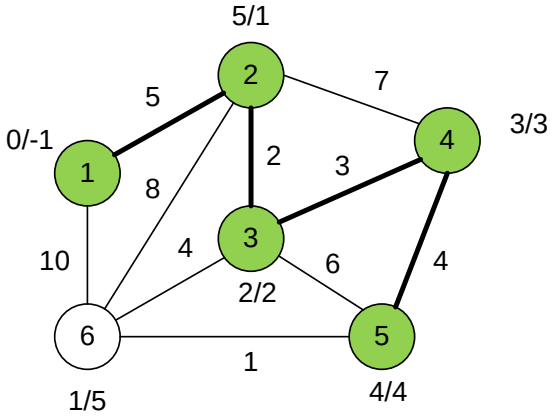
Vertex 3 is included to MST. Recalculate the labels for the edges outgoing from it.



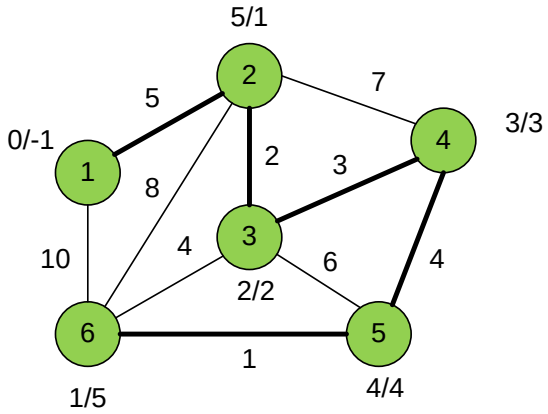
Vertex 4 is included to MST. Recalculate the labels for the edges outgoing from it.



Vertex 5 is included to MST. Recalculate the labels for the edges outgoing from it.

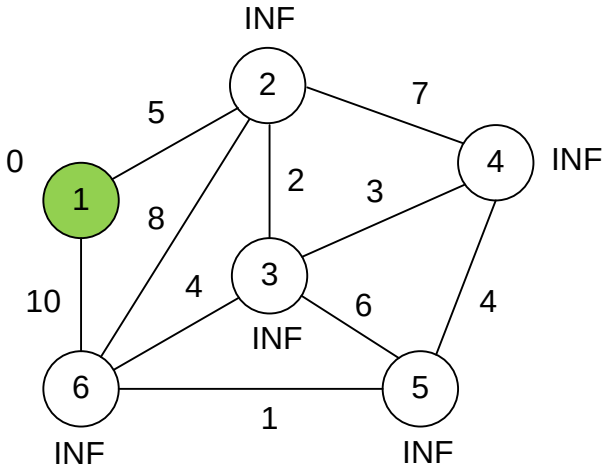


Vertex 6 is included to MST. End of the algorithm.

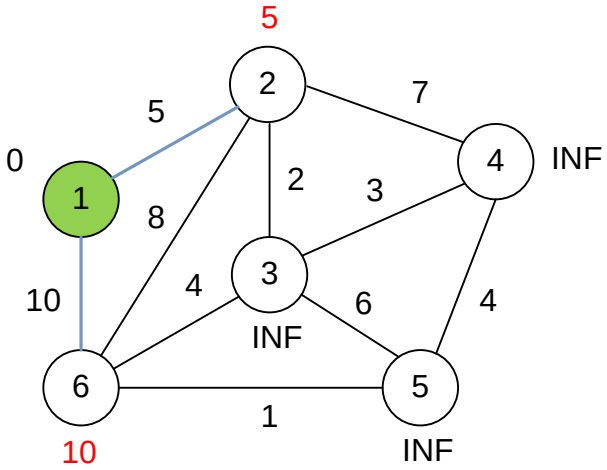


Example. Let's simulate the Prim's algorithm using the following example. Let *source* = 1. Near each vertex *v* the value of dist[*v*] is given. Start constructng MST from the vertex 1. Make an initialization:

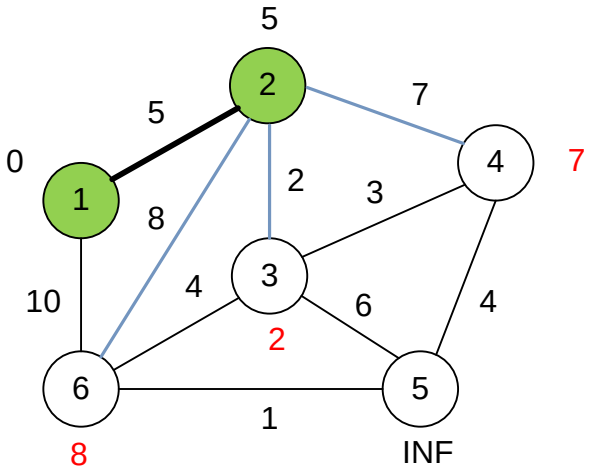
- dist[*v*] = ∞;
- dist[1] = 0;



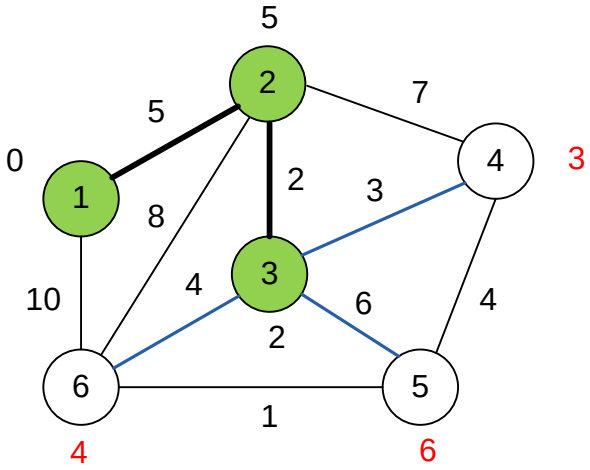
Vertex 1 is included to MST. Recalculate the labels for the edges outgoing from it.



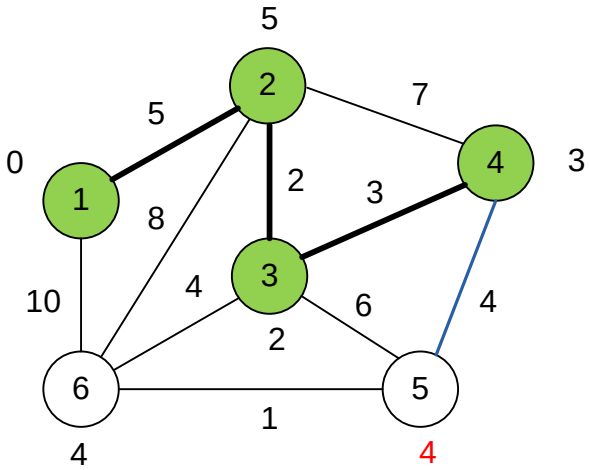
Vertex 2 is included to MST. Recalculate the labels for the edges outgoing from it.



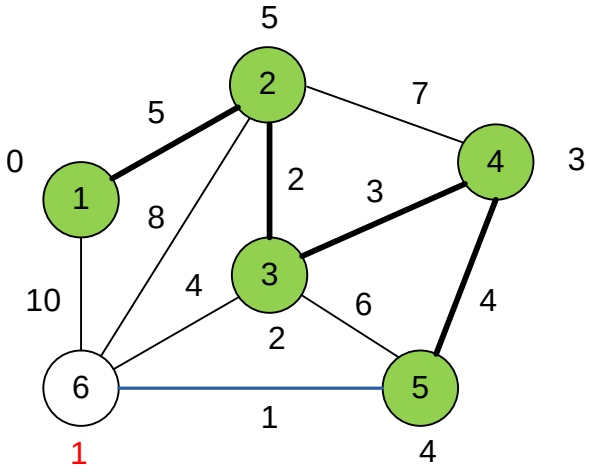
Vertex 3 is included to MST. Recalculate the labels for the edges outgoing from it.



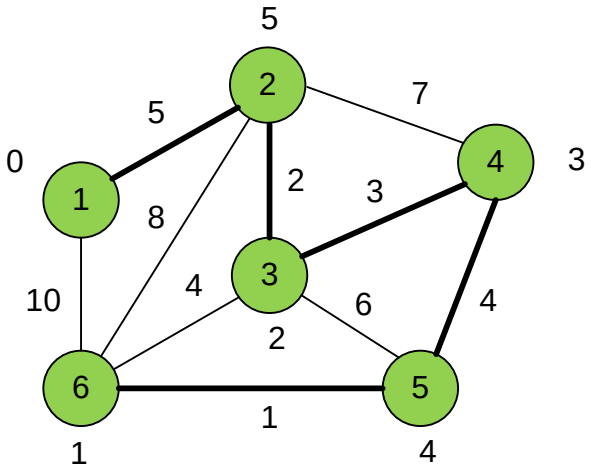
Vertex 4 is included to MST. Recalculate the labels for the edges outgoing from it.



Vertex 5 is included to MST. Recalculate the labels for the edges outgoing from it.



Vertex 6 is included to MST. End of the algorithm.

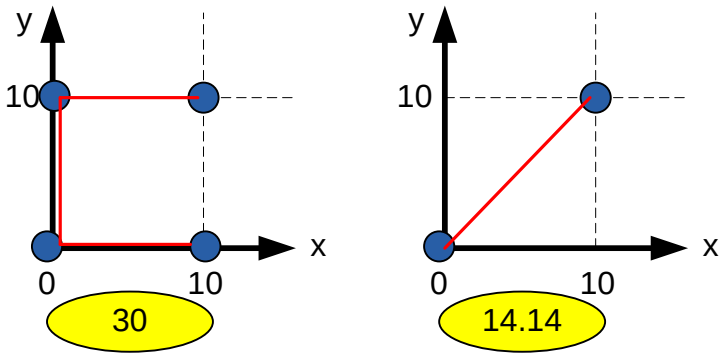


E-OLYMP **1387. Underground cables** A city wants to get rid of their unsightly power poles by moving their power cables underground. They have a list of points that all need to be connected, but they have some limitations. Their tunneling equipment can only move in straight lines between points. They only have room for one underground cable at any location except at the given points, so no two cables can cross.

Given a list of points, what is the least amount of cable necessary to make sure that every pair of points is connected, either directly, or indirectly through other points?

► To solve the problem, you need to find the minimum spanning tree (MST). Implement Prim algorithm.

Consider two graphs, gven in a sample.



E-OLYMP **9876. Jurassic Jigsaw** The famous Jurassic park biologist Dean O'Saur has discovered new samples of what he expects to be the DNA of a dinosaur. With the help of his assistant Petra Dactil, he managed to sequence the samples, and now they are ready for analysis. Dean thinks this dinosaur was affected with a particular disease mutating the DNA of some cells.

To verify his theory, he needs to compute the most likely evolutionary tree from the samples, where the nodes are the samples of DNA. Because there is no temporal data of the DNA samples, he is not concerned where the root of the tree is.

Dean considers the most likely evolutionary tree, the tree with smallest unlikeliness: the unlikeliness of a tree is defined as the sum of the weights of all edges, where the weight of an edge is the number of positions at which the two DNA strings are different.

As a world expert in data trees, he asks you to reconstruct the most likely evolutionary tree.

In the first sample, the optimal tree is AA – AT – TT – TC. The unlikeliness of the edge between AA and AT edge is 1, because the strings AA and AT differ in exactly 1 position. The weights of the other two edges are also 1, so that the unlikeliness of the entire tree is 3. Since there is no tree of unlikeliness less than 3, the minimal unlikeliness of an evolutionary tree for this case is 3.

► To solve the problem, you need to find the minimum spanning tree (MST). Implement Prim algorithm.