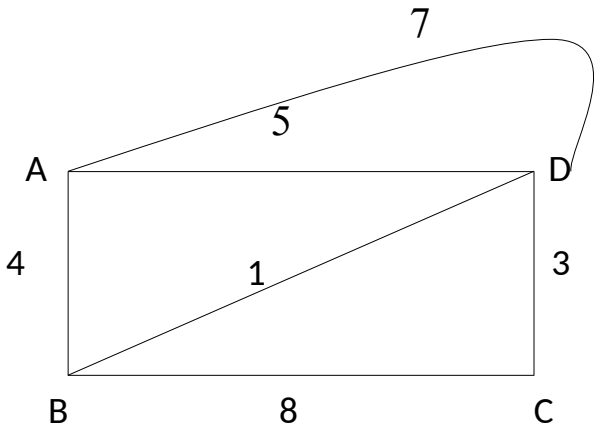
Prim's algorithm:

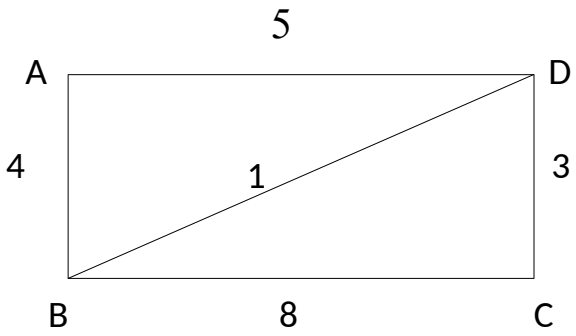
- T forms a single tree.
- The edge e added to T is always least-weight edge connecting the tree, T, to a vertex not in the tree

e.g



Step-1 Remove all loops and parallel edges.

In this step remove all loops and parallel edges from given graph. In case of parallel edges, keep the one which has the least cost associated and remove all others.



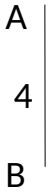
Step-2 Choose any arbitrary node as root node

In this case we choose A as root node.

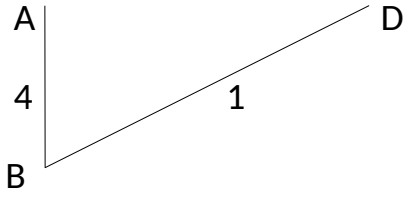
A

Step-3 Check outgoing edges and selects the one with less cost.

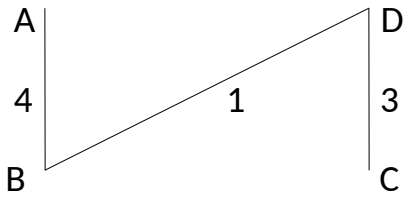
After choosing A as a root we have A-D and A-B two edges with weight 5 and 4 respectively. We choose A-B as it lesser weight than other edge.



Now we will check all the outgoing edges from vertices A,B (A-D ,B-D and B-C are the edges outgoing from vertices A,B). Among these edges B-D has least weight so connect B-D edges



Now we will check all the outgoing edges from vertices A,B and D(A-D ,D-C and B-C are the edges outgoing from vertices A,B). Choose the edge with minimum weight and that edge do not create cycle. Among these edges D-C has least weight so connect D-C edges.



All the vertices were covered with minimum weight, so cost for minimum spanning tree is 8.

Implementation:

- To choose the next edge to be included in T,

NEAR (i:n) array is used.

Procedure PRIM

```
Procedure PRIM (G, Cost, mincost)
/* Let n be # of vertices */
      Integer NEAR (1:n);
      Integer u,w,p,l;
1. Begin
2.          Choose an arbitrary vertex vo.
3.         mincost=o; NEAR (vo)=o
4.         For each vertex w vo do
5.                  NEAR (w)=vo;
6.         End for
7.          For I=1 to n-1 do /* fin n-1 edges of T */
8.         Choose a vertex w s.t.
9.         cost(w,NEAR(w) )= min (cost (u, NEAR(u)) )
10.         where NEAR (u) ¹ o
11.         mincost = mincost+ cost (w, NEAR(w));
12.         NEAR (w)=o
13.         For each vertex p do
14.         if NEAR(p)  o & cost (p, NEAR(p) )  > cost (p,w)
15.                 then NEAR (p)= w;
16.         endif
17.         end for
18.        End for
19. End.
```

Analysis

- The for loop between 4 and 6 takes O(n).
- Lines between 8 and 10 take O(n)
- The For loop between 13 and 17 takes O(n)
- Finally, the main For loop that starts at line 7 takes O(n)

The overall algorithm takes $O(n^2)$.