Kruskal's MST Algorithm
with step-by-step execution

Daniel Michel Tavera
Student at National Autonomous University of Mexico (UNAM)
Mexico
e-mail: daniel_michel@ciencias.unam.mx

Social service project director: Dr. Patricia Esperanza Balderas Cañas
Full time professor at National Autonomous University of Mexico (UNAM)
Mexico
e-mail: balderas.patricia@gmail.com

Introduction

Kruskal's MST Algorithm is a well known solution to the Minimum Spanning Tree (MST) problem, which consists in finding a subset of the edges of a connected weighed graph, such that it satisfies two properties: it maintains connectivity, and the sum of the weights of the edges in the set is minimized.

In this work we utilize the definition of Kruskal's MST algorithm given by Cook et. al. (see References) which is as follows:

"Keep a spanning forest $H = (V,F)$ of G, with $F = \emptyset$ initially.
At each step add to F a least-cost edge $e \notin$ F such that H remains a forest.
Stop when H is a spanning tree."

This work is part of a social service project consisting in the implementation of several graph theory algorithms with step-by-step execution, intended to be used as a teaching aid in graph theory related courses.

The usage examples presented were randomly generated.

Module usage

The KruskalMST module contains only a single procedure definition for Kruskal(G, *stepByStep*, *draw*), as follows:

Calling Kruskal(...) will attempt to calculate the MST for graph G using Kruskal's Algorithm.

The parameters taken by procedure Kruskal(...) are explained below:

- G is an object of type Graph from Maple's *GraphTheory* library, it is the graph for which the MST will be computed. Regardless of how it is defined, G will always be treated as though it is undirected.
  This parameter is not optional

- *stepByStep* is a true/false value. When it is set to *true*, the procedure will print a message reporting whenever an edge is added to the MST or discarded because it would create a loop. When it is *false*, only the final result will be shown.
  This parameter is optional, and its default value is *false*.

- *draw* is a true/false value. When it is set to *true*, the resulting MST will be displayed after computation finishes; if both *stepByStep* and *draw* are *true* then the graph *G* will be drawn at every step, highlighting the edges in the MST in green and the discarded edges in red. When *draw* is set to *false*, the graphs will not be displayed, and the procedure will only print the total weight of the MST and return the edge list for the MST.
  This parameter is optional, and its default value is *true*.

The return value can be one of three possibilities as follows:
- If *draw* is *true*, the procedure returns a graph *H* such that *H* is an MST for *G*.
- If *draw* is *false*, the procedure will return the edge list for *H*, this is so the value reported by Maple contains more useful information.
- If *G* is not a connected graph, the procedure will return the string "ERROR".

Module definition and initialization

```
> restart:
  with(GraphTheory):
  KruskalMST := module()
  option package;
  export Kruskal;

  Kruskal := proc (G::Graph, stepByStep::truefalse := false,
  draw::truefalse := true)
  local H :: list, V :: set, E :: set, e :: list, g::Graph ,
  s::symbol, a::list, c::set, c1::set, c2::set, discarded::set,
  total::int, components::int:

  #variable initialization
  H:={}:   #List of edges of the MST
  E:=Edges(G,weights):   #backup of G's edge list, used in
  destructive operations
  components:=nops(Vertices(G)):   #number of distinct connected
  components

  V:={}:   #list of connected components of the MST
  for s in Vertices(G)do
    V:=V union {{s}}:   #initially each set is its own connected
  component
  end do:
```

```
if draw and stepByStep then
  printf("key: yellow = vertices, blue = original graph edges,
\n\tgreen = MST edges, red = discarded edges.\n");
  discarded:={}:   #discarded edge set, used only when drawing
the graph
end if:

total:=0:   #total weight of the edges in the MST

while nops(E)>0 do:   #continue while there are unprocessed
edges
 e:={}:   #assume no edge is added to the MST
 for a in E do:   #for each edge
  for c in V do:  #for each connected component
    if a[1][1] in c then
      if a[1][2] in c then
        E:=E minus {a}:   #if it would cause a loop in the MST,
discard the edge
        if stepByStep then   #report discarded edge if the
option is enabled
          printf("discarded edge (%a,%a) as it would cause a
loop\n", a[1][1], a[1][2]):
          if draw then   #draw resulting graph if the option is
enabled
            discarded:=discarded union {a}:
            g:=Graph(Vertices(G), discarded):
            HighlightSubgraph(G, g, red, yellow):
            print(DrawGraph(G));
          end if:
        end if:
      else
        if e={} or a[2]<e[2] then   #if no loop is formed, take
the minimum weight edge
          e:=a:
        end if:
        if e=a then
          c1:=c:
        end if:
      end if:
    else
      if a[1][2] in c then
        if e={} or a[2]<e[2] then   #if no loop is formed, take
```

```
the minimum weight edge
          e:=a:
        end if:
        if e=a then
          c2:=c:
        end if:
      end if:
    end if:
  end do:
 end do:

 if e<>{} then   #if an edge of the MST was found, add it to the
MST
  V:= V minus {c1,c2} union {c1 union c2}:
  H:=H union {e}:
  E:=E minus {e}:
  total:= total+e[2]:
  components:=components-1:
  if stepByStep then   #report added edge if the option is
enabled
   printf("added edge (%a,%a) with weight %a to the MST\n", e[1]
[1], e[1][2], e[2]):
   if draw then   #draw resulting graph if the option is enabled
     g:=Graph(Vertices(G), H):
     HighlightSubgraph(G, g, green, yellow):
     print(DrawGraph(G));
   end if:
  end if:
  if components=1 then   #algorithm ends when all vertices are
in the same connected component
   if stepByStep then   #report end of computation if the option
is enabled
    printf("Finished MST construction.\n"):
    break:
   end if:
  end if:
 else
  if(E<>{})then   #if there are unprocessed edges, but none of
them belongs to the MST, report an error
   printf("ERROR: unable to construct MST, graph may be
disconnected");
   return "ERROR":
```

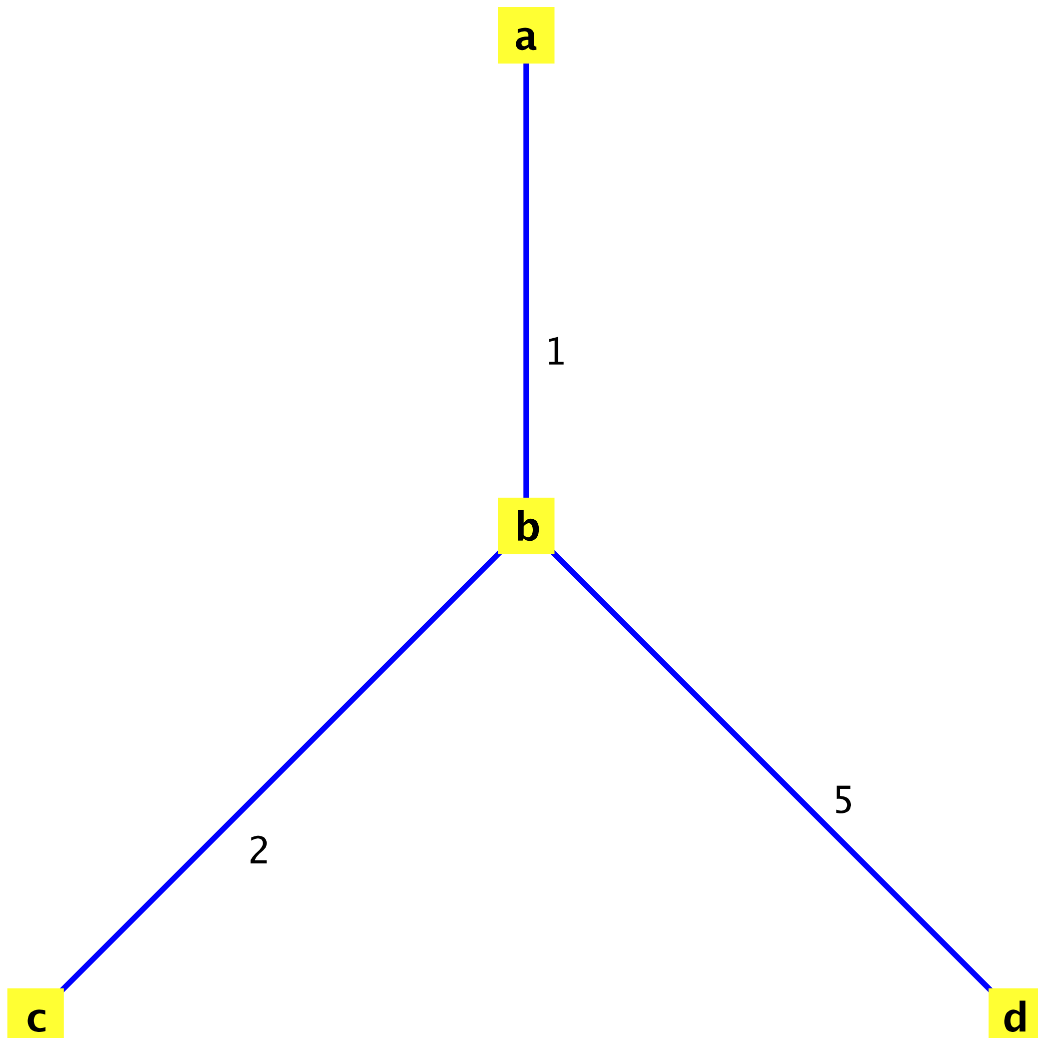
```
  end if:
 end if:
end do:

if (draw) then   #print MST if the option is enabled
 g:=Graph(Vertices(G),H):
 if stepByStep then
   printf("graph for the obtained MST:\n", a[1][1], a[1][2]):
 end if:
 print(DrawGraph(g));
 printf("total weight of the MST: %a\n",total):   #report total
MST weight
 return g:   #return graph for the MST
else
 printf("total weight of the MST: %a\n",total):   #report total
MST weight
 return H;   #return list of edges for the MST
end if:

end proc:
end module:

with (KruskalMST);
```

[*Kruskal*]

Usage examples

Default Behavior: print resulting MST, without step-by-step reports.

```
> vertices:=["a","b","c","d"]:
  edges:={[{"a", "b"}, 1],[{"a", "c"}, 3],[{"b", "c"}, 2],[
  {"b", "d"}, 5],[{"c", "d"}, 9]}:
  g := Graph(vertices,edges):
  Kruskal(g);
```



total weight of the MST: 8

Graph 1: an undirected weighted graph with 4 vertices and 3 edge(s)

Shows step-by-step reports, but doesn't print the MST

```
> vertices:=[1,2,3,4,5,6]:
  edges:={[{1,2},6],[{1,3},2],[{1,4},5],[{2,3},6],[{2,4},4],[
  {2,5},5],[{3,4},6],[{3,5},3],[{3,6},2],[{4,5},6],[{5,6},2]}:
  g := Graph(vertices,edges):
  Kruskal(g, true, false);
```

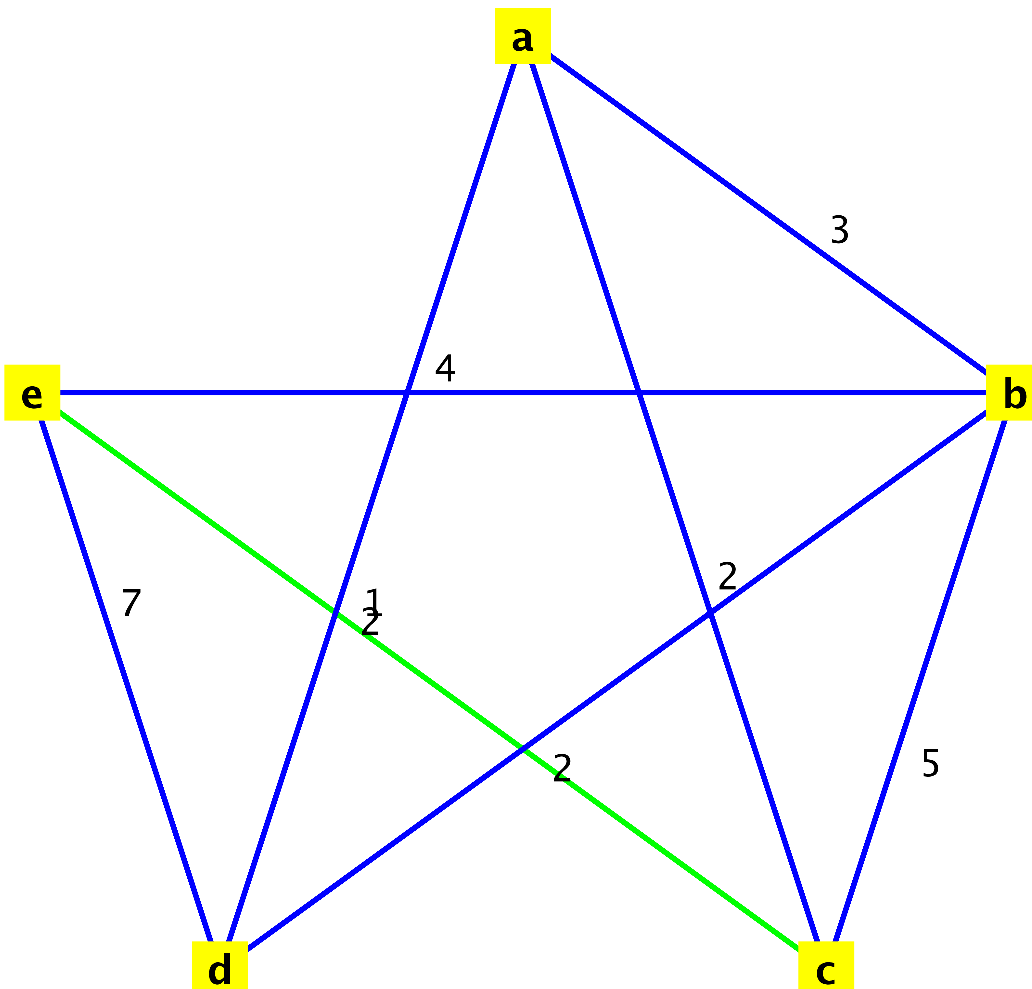
```
added edge (1,3) with weight 2 to the MST
added edge (3,6) with weight 2 to the MST
added edge (5,6) with weight 2 to the MST
discarded edge (3,5) as it would cause a loop
added edge (2,4) with weight 4 to the MST
added edge (1,4) with weight 5 to the MST
Finished MST construction.
total weight of the MST: 15
```

$\{[\{1,3\},2],[\{1,4\},5],[\{2,4\},4],[\{3,6\},2],[\{5,6\},2]\}$

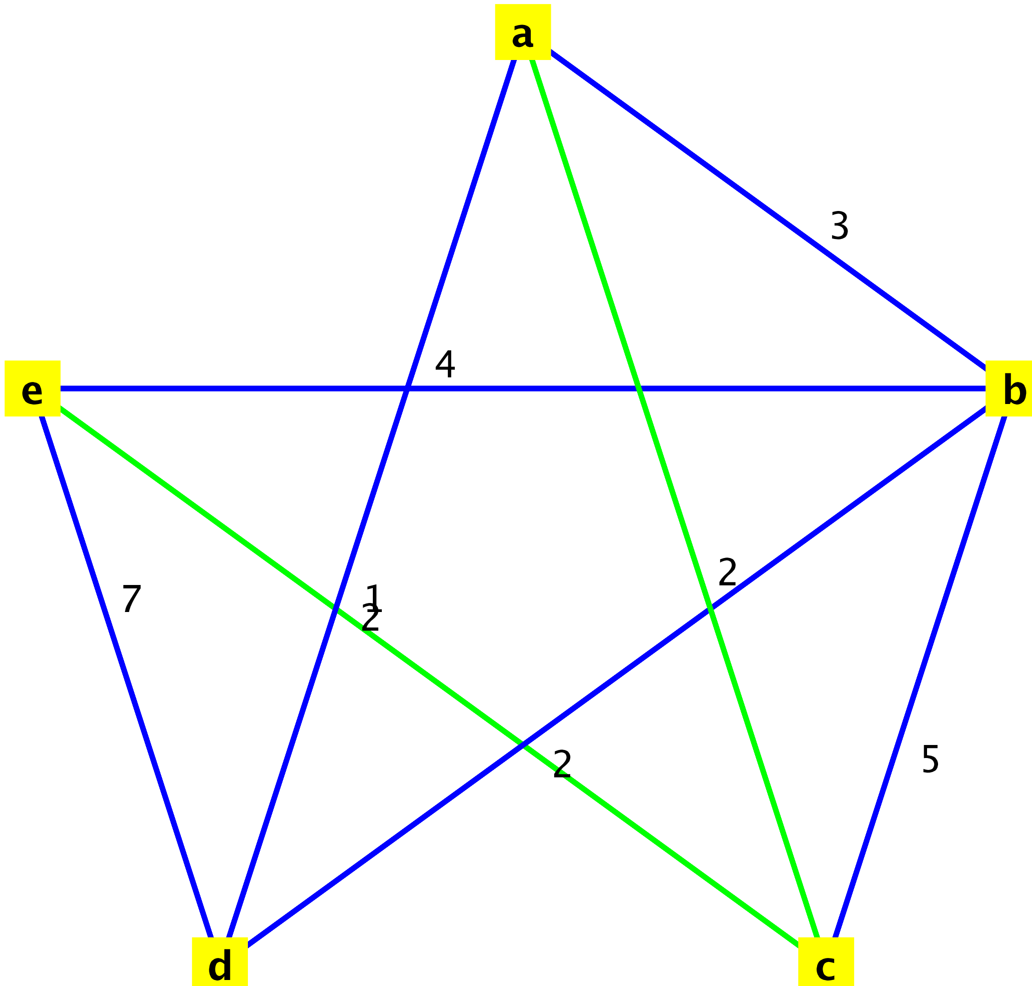
Shows step-by-step process with graphs for each step

```
> vertices:=["a","b","c","d","e"]:
  edges:={[{"a","b"},3],[{"a","c"},2],[{"a","d"},2],[{"b","c"},
  5],[{"b","d"},2],[{"b","e"},4],[{"c","e"},1],[{"d","e"},7]}:
  g := Graph(vertices,edges):
  Kruskal(g, true);
```

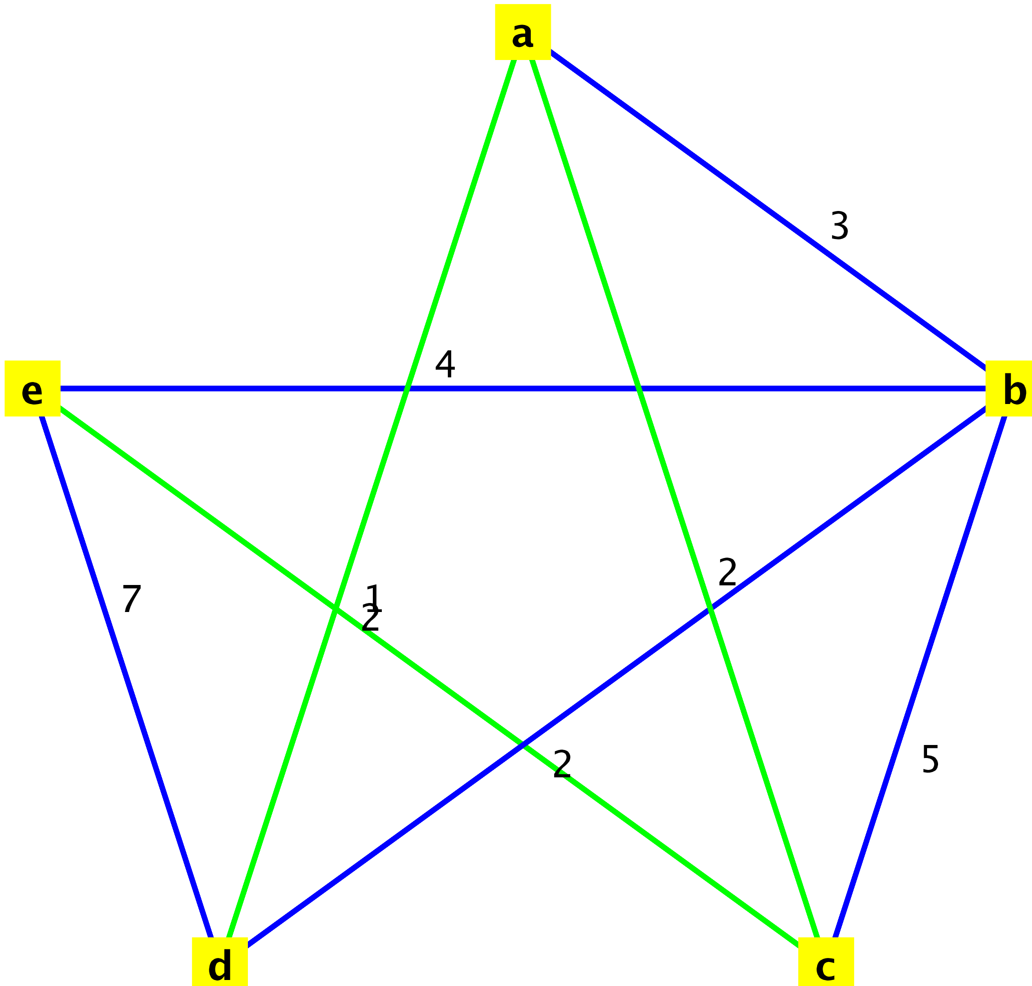
```
key: yellow = vertices, blue = original graph edges,

= MST edges, red = discarded edges.
added edge ("c","e") with weight 1 to the MST
```



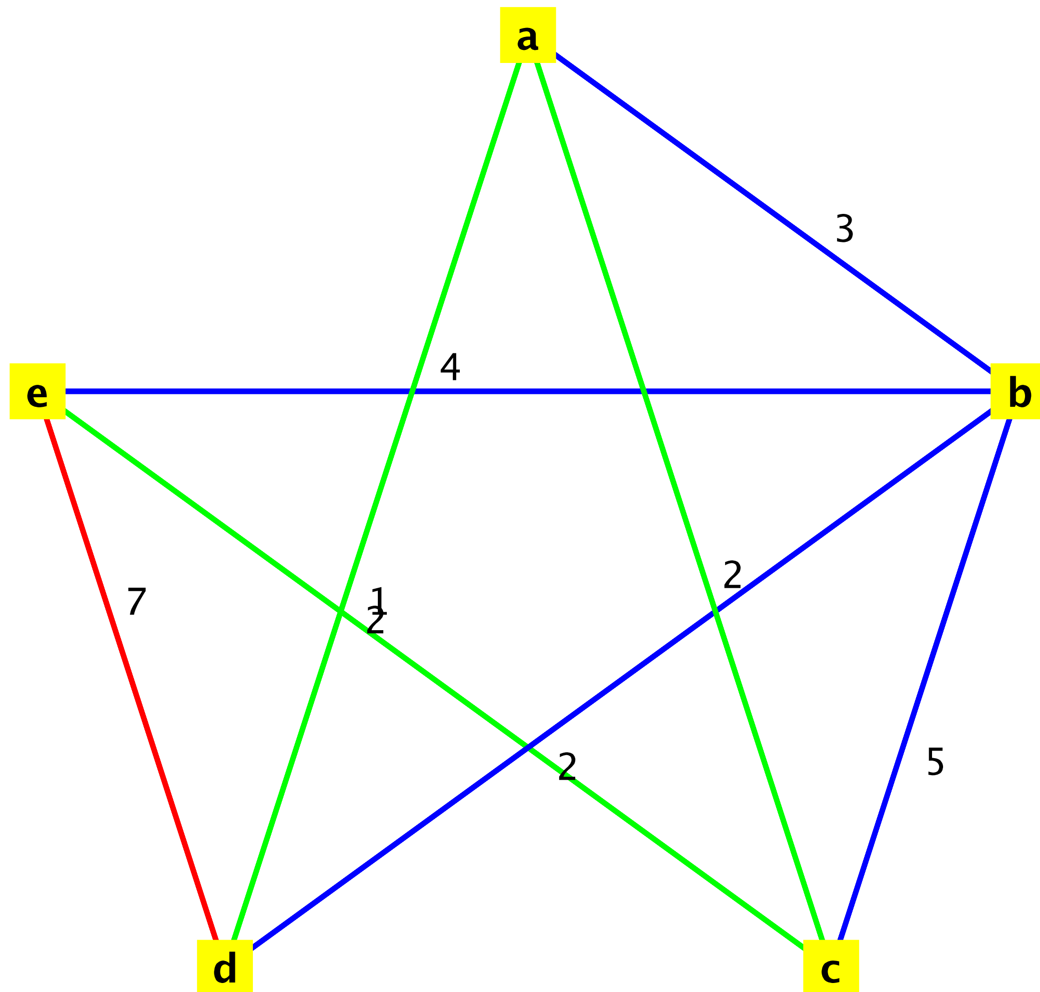

```
added edge ("a","c") with weight 2 to the MST
```



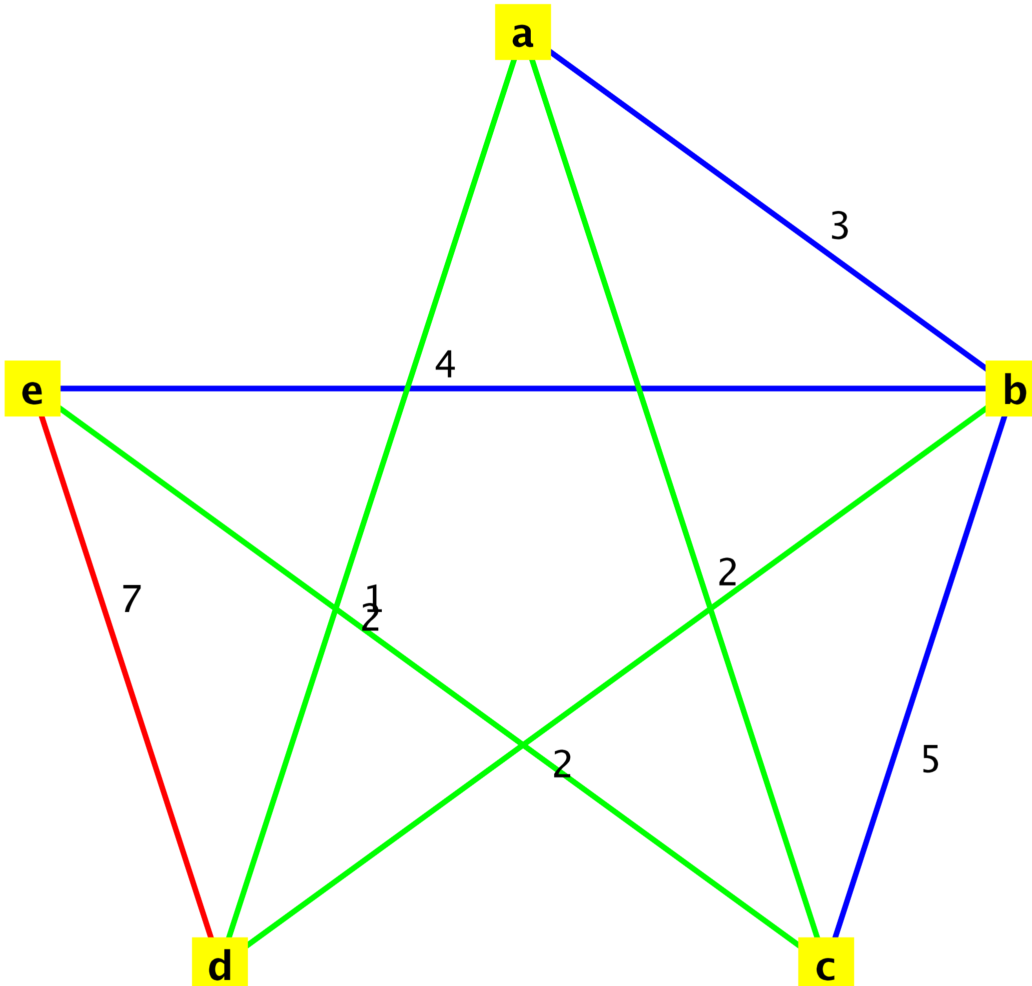
added edge ("a","d") with weight 2 to the MST



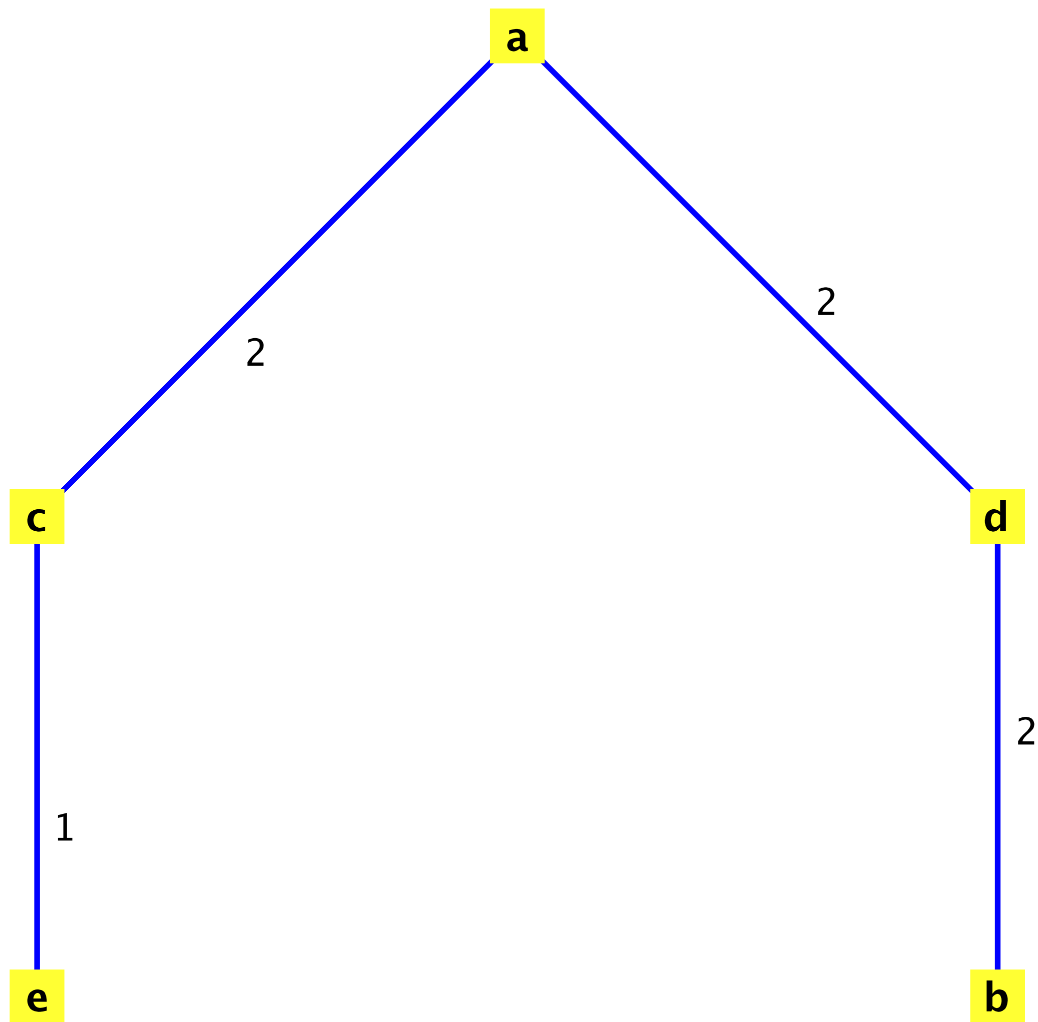
discarded edge ("d","e") as it would cause a loop



added edge ("b","d") with weight 2 to the MST




```
Finished MST construction.
graph for the obtained MST:
```




```
total weight of the MST: 7
```

Graph 2: an undirected weighted graph with 5 vertices and 4 edge(s) **(4.2.1)**

References

Cook, William J. et. al. *Combinatorial Optimization*. Wiley-Interscience, 1998. ISBN 0-471-55894-X