Greedy Algorithms

Dr. Jeevani Goonetillake

Minimum Spanning Trees

Dr. Jeevani Goonetillake

Spanning Tree

A spanning tree for a connected undirected graph G=(V,E) is a subgraph of G that is a tree and contains all the vertices of G

Thus a spanning tree for G is a graph, T = (V', E') with the following properties:

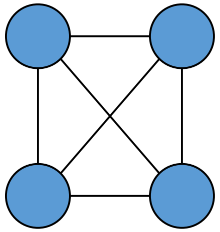
- V' = V
- T is connected
- T is acyclic.

A spanning tree is called a tree because every acyclic undirected graph can be viewed as a general, unordered tree. Since the edges are undirected, any vertex may be chosen to serve as the root of the tree.

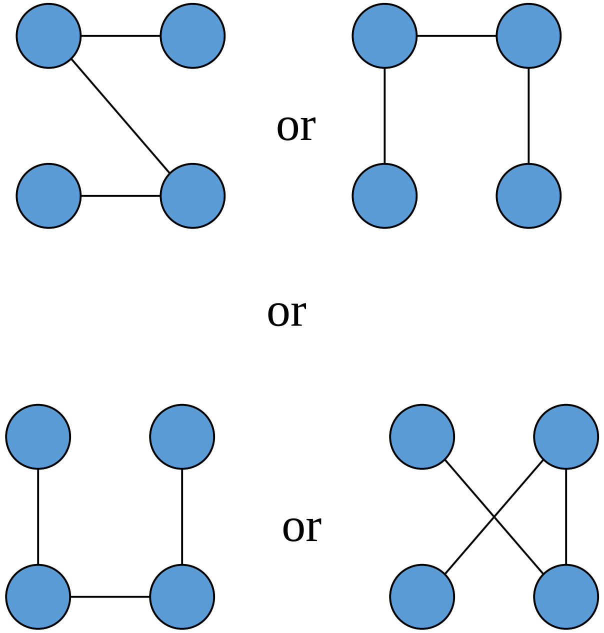
Spanning Tree

A graph may have many spanning trees.

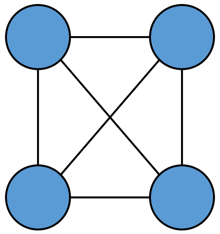
Graph A



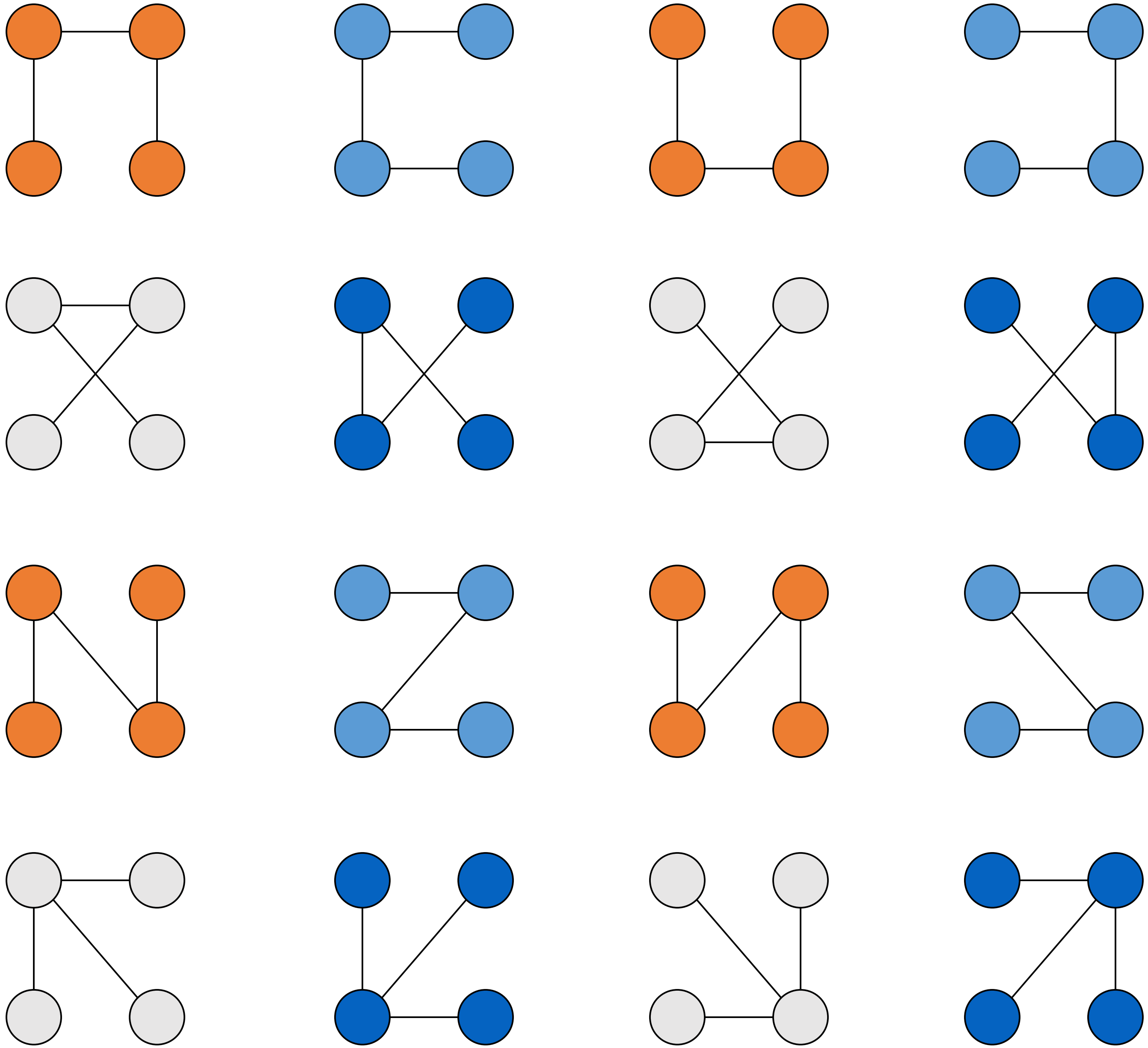
Some Spanning Trees from Graph A



Complete Graph



All 16 of its Spanning Trees

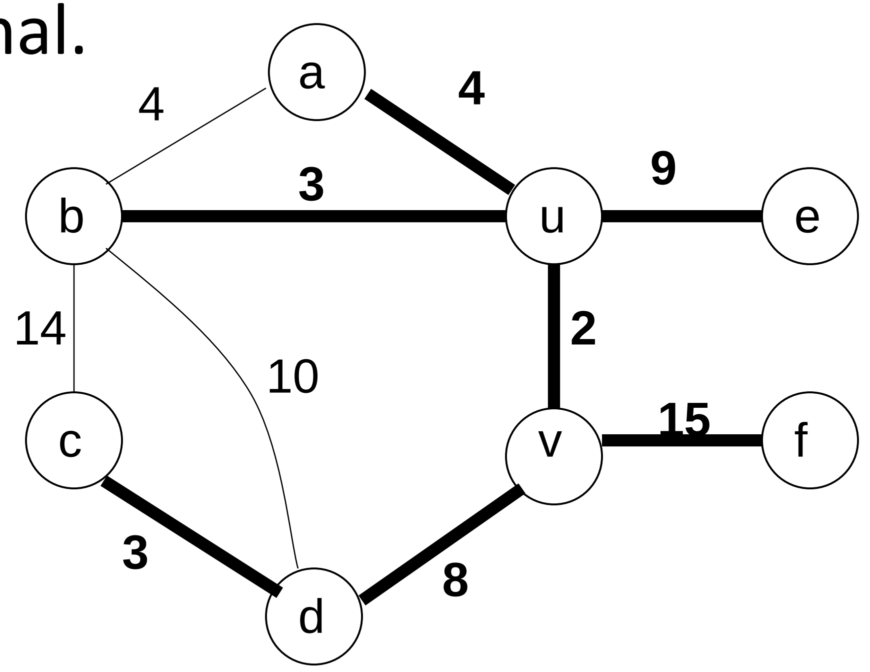


Minimal Spanning Tree (MST)

A minimal spanning tree of a (connected undirected) weighted graph $G = (V, E, w)$ is a sub-graph $T = (V', E')$ of G such that

- T is a spanning tree and
- weight $w(T) = \sum_{e \in E'} w(e)$ is minimal.

* Can a graph have more then one minimum spanning tree?



Applications of Minimum Spanning Trees

Minimum-cost spanning trees have many applications. Example:

- Building cable networks that join n locations with minimum cost.

Translating a Problem into a MST

- Each location of the network must be connected using the least amount of cables.

Modeling the Problem

- The graph is a complete, undirected graph $G = (V, E ,W)$, where V is the set of locations, E is the set of all possible interconnections between the pairs of locations and $w(e)$ is the length of the cable needed to connect the pair of vertices.
- Find a minimum spanning tree.

Underlying Principles

Recall two of the defining properties of a tree:

- Removing an edge from a tree breaks it into two separate sub-trees.
- Adding an edge that connects two vertices in a tree creates a unique cycle.

MST Algorithm – Cut Property

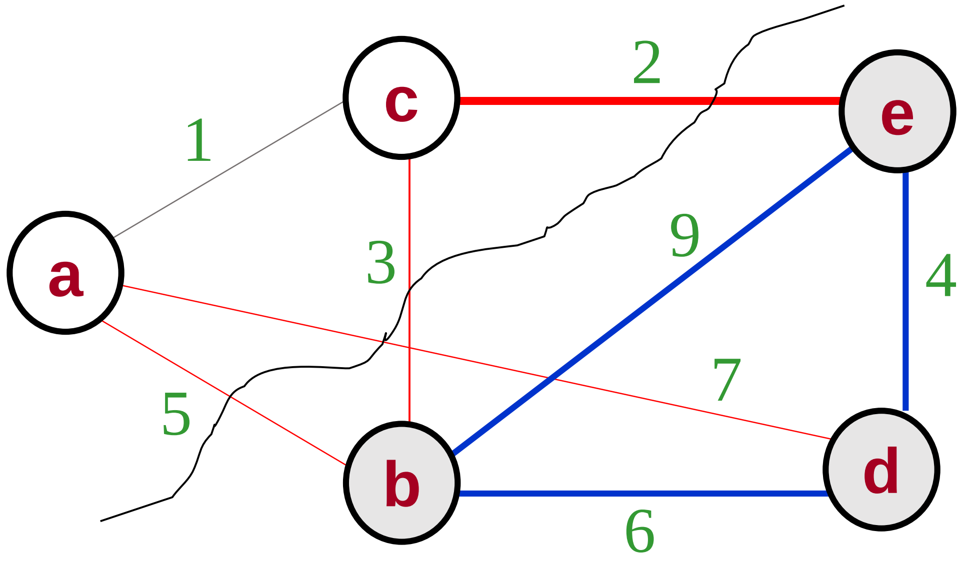
Given any cut in an edge-weighted graph (for simplicity assume that all edge weights are distinct), the crossing edge of minimum weight is in the MST of the graph.

MST Construction

The cut property yields a simple greedy algorithm for finding an MST.

- Start with an empty set T of edges.
- Let E′ be the set of edges relevant to a cut which do not contain any edge from T
- As long as T is not a spanning tree, add a minimal-cost edge from E′ (known as light edge) to T.
- Different choices of E′ lead to different specific algorithms.

Light Edge



A cut (S, V-S) of an undirected graph G = (V,E) is a partition of V.

Vertices in the set S - gray Vertices in (V-S) – white
Edges crossing the cut are connecting white vertices and black vertices.

MST Construction

Given any cut the crossing edge of minimum weight is in the MST.

Proof: Suppose min-weight crossing edge e is not in the MST.

- Adding e to the MST creates a cycle.
- Some other edge e’ in cycle must be a crossing edge.
- Removing e’ and adding e is also a spanning tree
- Since w(e) is less than w(e’) that spanning tree is lower weight.
- Contradiction*

Generic Algorithm for MST

Input : connected weighted graph, G

Output : MST, T, for graph G

Greedy strategy in the generic algorithm

- Grow the MST one edge at a time.
- Manage a set of edges A, that is prior to each iteration, A is a subset of some MST
    - At each step determine an edge (u,v) that can be added to A without violating this invariant.
    - We call such an edge a **safe edge** for A, since it can be safely added to A while maintaining the invariant.

Generic Algorithm

```
Generic-MST(G,w)
A ← 0
while A does not form a spanning tree
  do find an edge (u,v) that is safe for A
        A ← A U { (u,v)}
return A
```

Safe edge - a light edge satisfying a given property.

Greedy Choice

Two algorithms to build a minimum spanning tree.

- MST can be grown from a forest of spanning trees : find a safe edge to be added to the growing forest by finding, of all the edges that connects two distinct trees in the forest. The choice is greedy because at each step it adds to the forest an edge of least possible weight. (**Kruskal's algorithm**)

- MST can be grown from the current spanning tree: Each step adds to the tree A a light edge that connects A to an isolated vertex—one on which no edge of A is incident. (**Prim's algorithm**)

Kruskal's Algorithm

MST-Kruskal(G,w)
1. $A \leftarrow 0$
2. For each vertex $v \in V[G]$
3. do Make-Set(v)
4. sort the edges of E into nondecreasing order by weight w
5. for each edge $(u,v) \in E$, taken in nondecreasing order by weight w
6. do if Find-Set(u) $\neq$ Find-Set(v)
7. then $A \leftarrow A \cup \{(u,v)\}$
8. Union (u,v)
9. return A

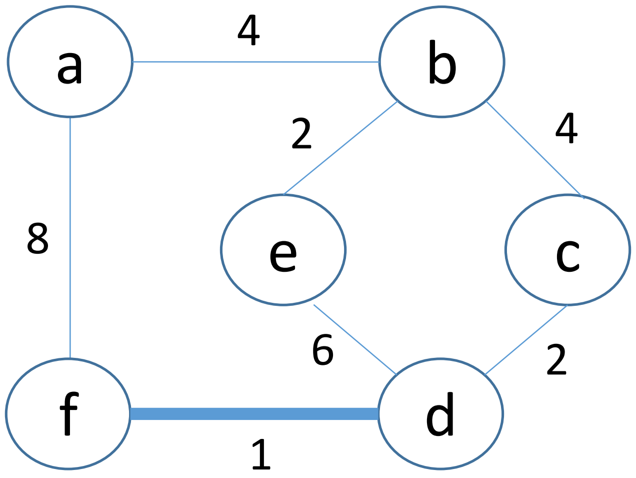
A – Tree

w - weight

Kruskal’s Algorithm

- Initialize the set A to the empty set
- Create |V| trees (sets), one containing each vertex.
- Sort the edges in increasing order of weight.
- Take the edges in the sorted order.
- For each edge (u,v), check whether the endpoints (vertices) u and v belong to the same tree.
- It is safe to connect two vertices if they belong to different trees. If so the edge (u,v) is added to A.
- Vertices in the two trees are merged.

Kruskal’s Algorithm Example



Initially A = { }

Sets – {a} {b} {c} {d} {e} {f}

E – Sorted in Ascending Order

(f, d)	(b, e)	(c, d)	(a, b)	(b, c)	(e, d)	(a, f)

Step 1

Take (f,d) ; Find-Set(f) $\neq$ Find-Set(d) => **add (f,d) to A**

A = {(f,d)}

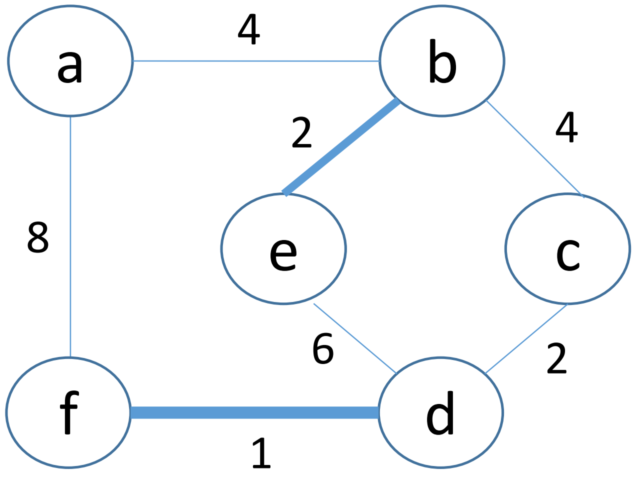
Combine Set(f) & Set(d)

Sets - {a} {b} {c} {e} {f,d}

Kruskal’s Algorithm Example

Step 2

(f, d)	(b, e)	(c, d)	(a, b)	(b, c)	(e, d)	(a, f)



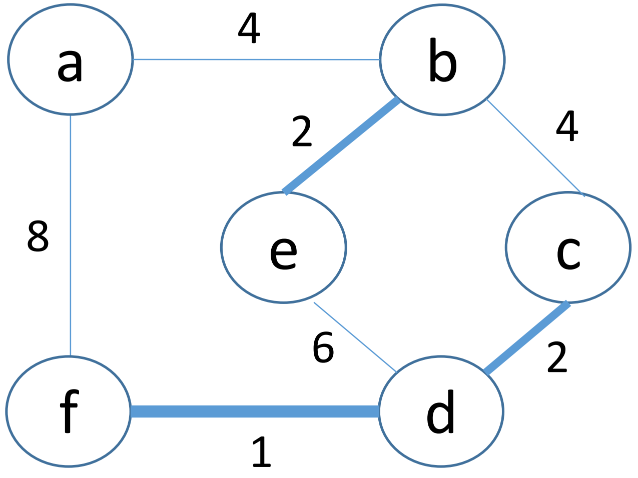
Take (b,e) ; Find-Set(b) ≠ Find-Set(e) => **add (b,e) to A**

A = {(f,d), (b,e)}

Combine Set(b) & Set(e)

Sets - {a} {b,e} {c} {f,d}

Kruskal's Algorithm Example



Step 3

(f, d)	(b, e)	(c, d)	(a, b)	(b, c)	(e, d)	(a, f)

Take (c,d) ; Find-Set(c) ≠ Find-Set(d) => **add (c,d) to A**

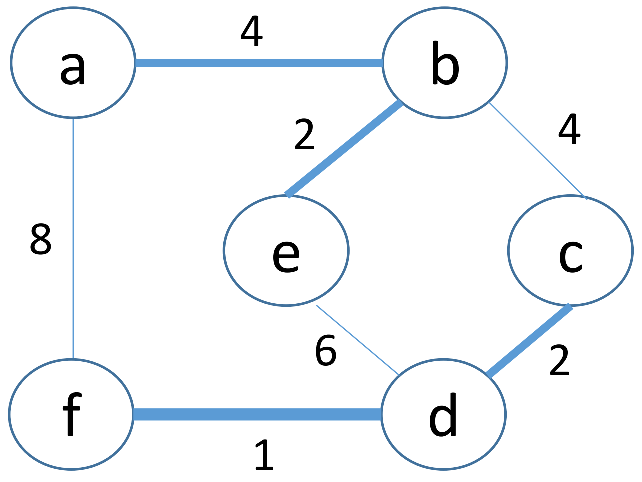
A = {(f,d), (b,e),(c,d)}

Combine Set(c) & Set(d)

Sets - {a} {b,e} {f,d,c}

Kruskal’s Algorithm Example

Step 4



(f, d)	(b, e)	(c, d)	(a, b)	(b, c)	(e, d)	(a, f)

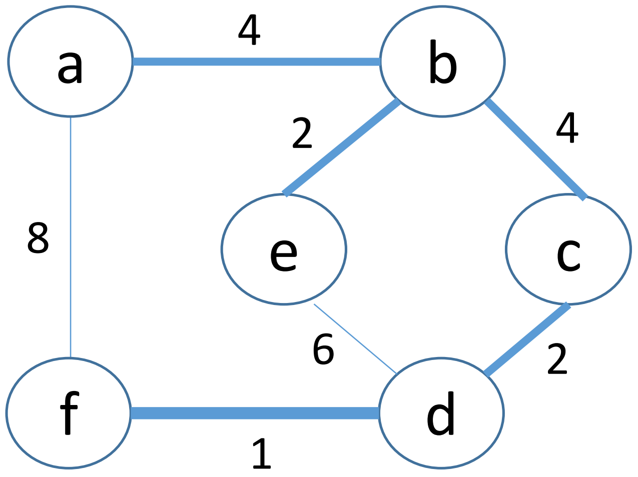
Take (a,b) ; Find-Set(a) $\neq$ Find-Set(b) => **add (a,b) to A**

A = {(f,d), (b,e),(c,d),(a,b)}

Combine Set(a) & Set(b)

Sets - {b,e,a} {f,d,c}

Kruskal’s Algorithm Example



Step 5

(f, d)	(b, e)	(c, d)	(a, b)	(b, c)	(e, d)	(a, f)

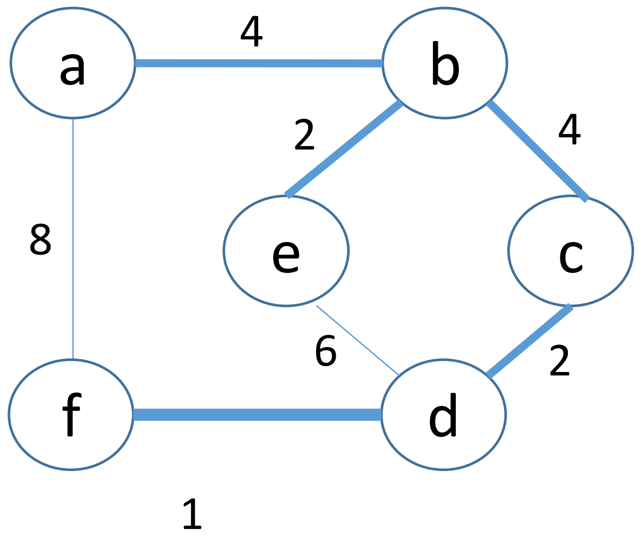
Take (b,c) ; Find-Set(b) ≠ Find-Set(c) => **add (b,c) to A**

A = {(f,d), (b,e),(c,d),(a,b),(b,c)}

Combine Set(b) & Set(c)

Sets - {b,e,a,f,d,c}

Kruskal's Algorithm Example



Step 6

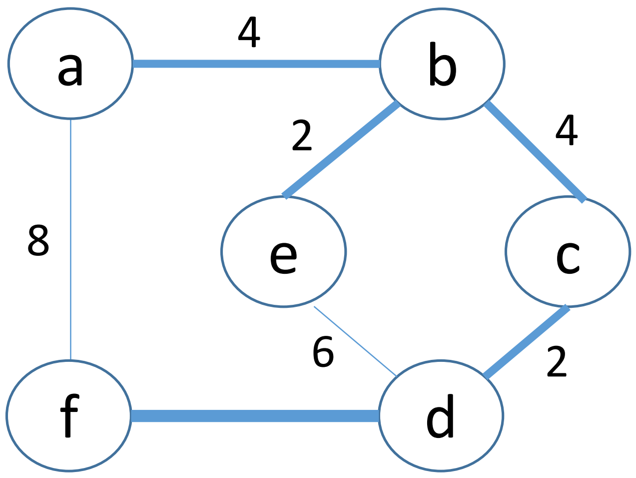
(f, d)	(b, e)	(c, d)	(a, b)	(b, c)	(e, d)	(a, f)

Take (e,d) ; Find-Set(e) $\neq$ Find-Set(d) => **Ignore**

A = {(f,d), (b,e),(c,d),(a,b),(b,c)}

Sets - {b,e,a,f,d,c}

Kruskal's Algorithm Example



Step 7

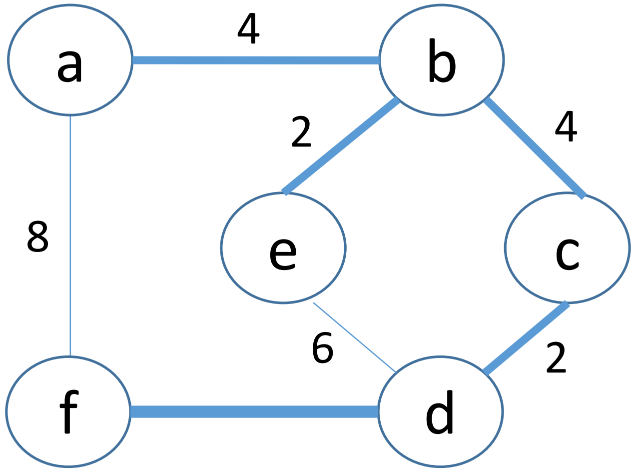
(f, d)	(b, e)	(c, d)	(a, b)	(b, c)	(e, d)	(a, f)

Take (a,f) ; Find-Set(a) $\neq$ Find-Set(f) => **Ignore**

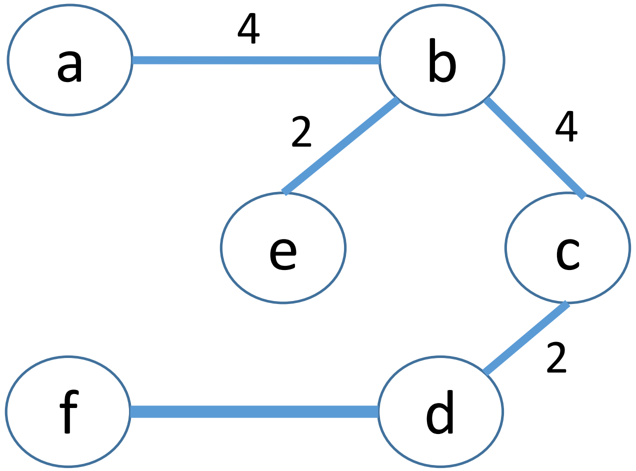
A = {(f,d), (b,e),(c,d),(a,b),(b,c)}

Sets - {b,e,a,f,d,c}

Kruskal’s Algorithm Example

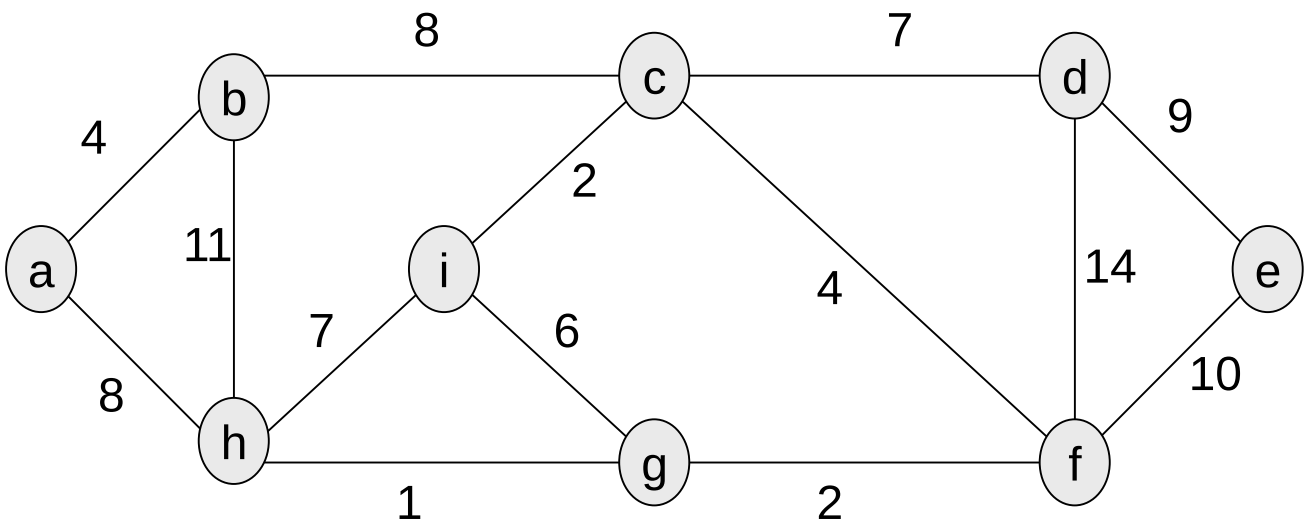


Graph



MST

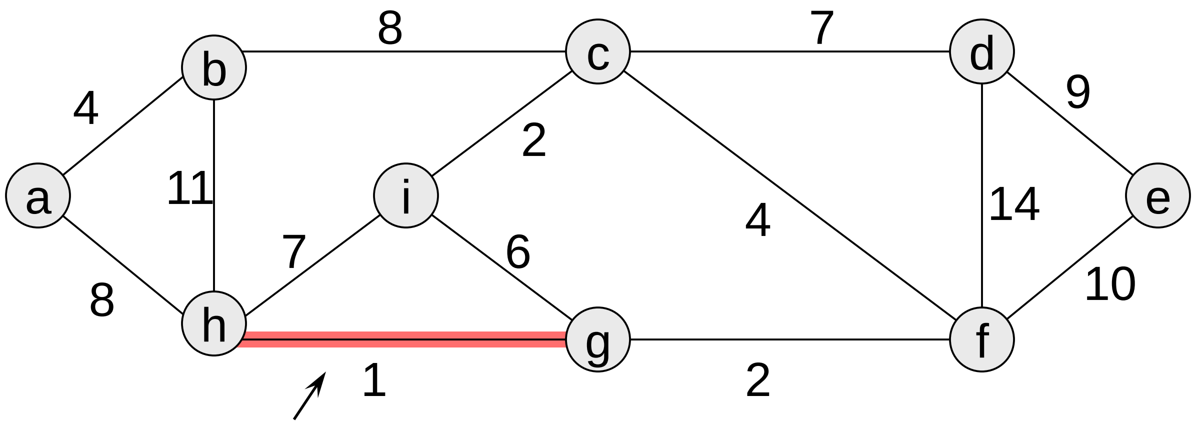
Kruskal's Algorithm - Problem



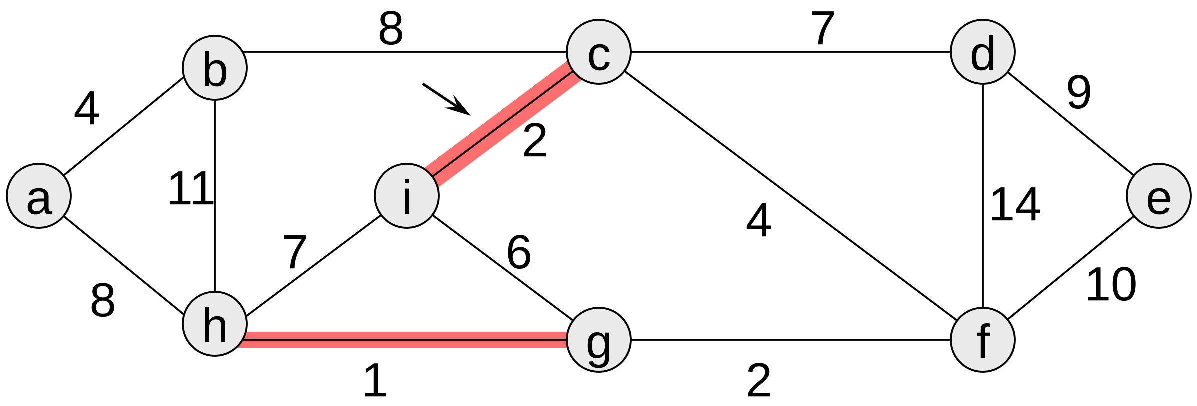
* Same example is given for Prim's in order to understand how the MST is constructed with respect to the two algorithms.

Kruskal’s Algorithm - Solution

(a)

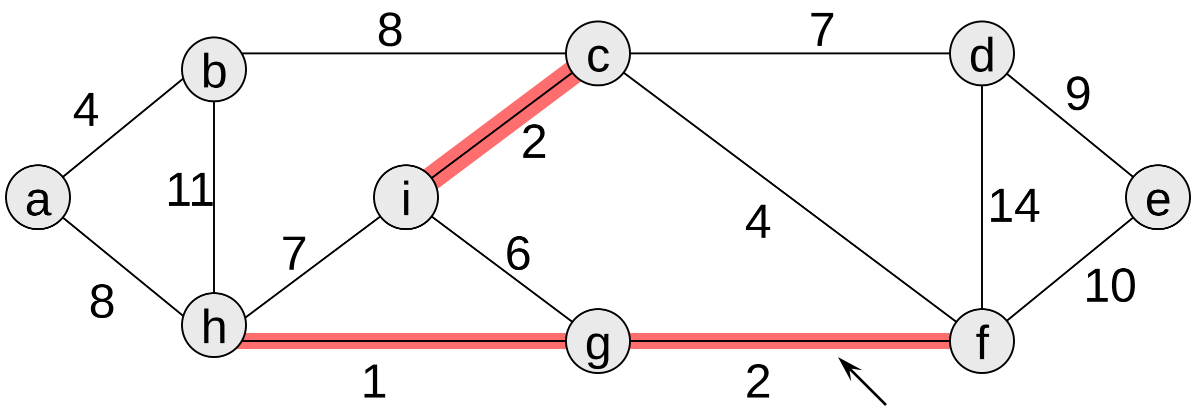


(b)

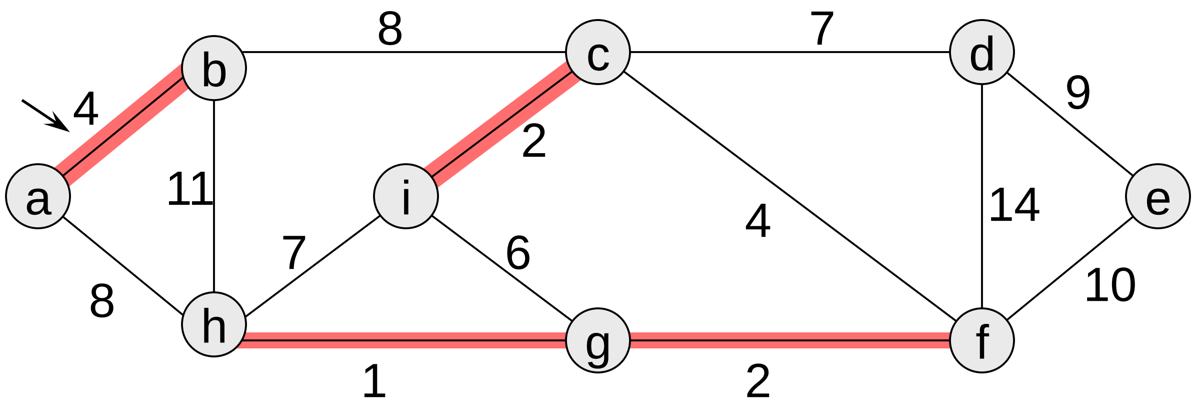


Kruskal's Algorithm - Solution

(c)

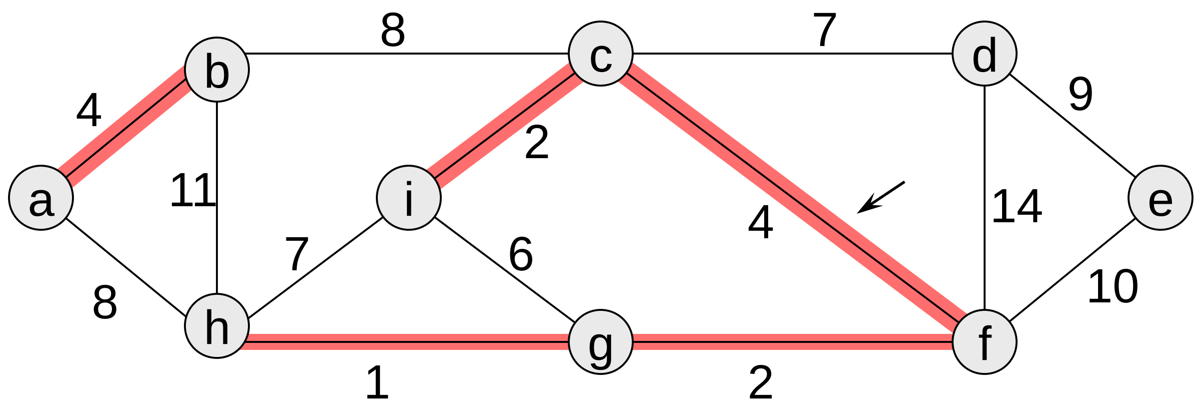


(d)

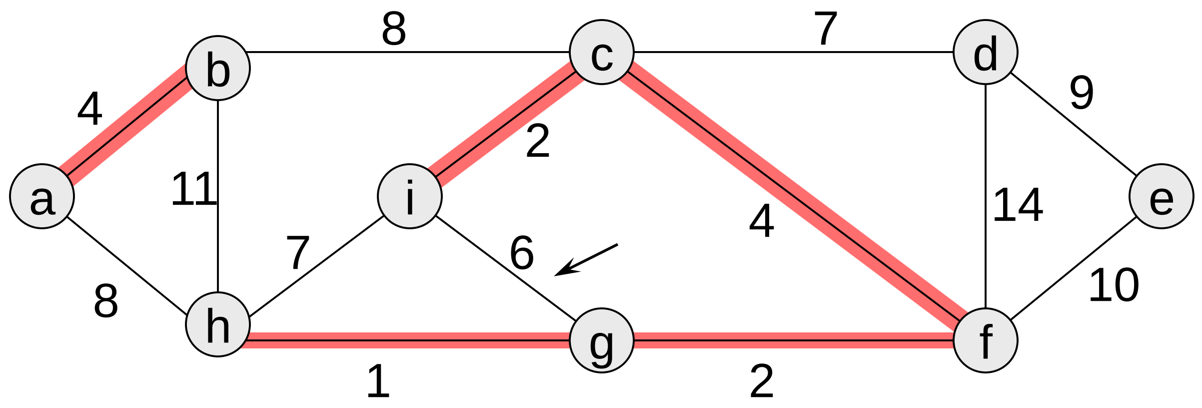


Kruskal’s Algorithm - Solution

(e)

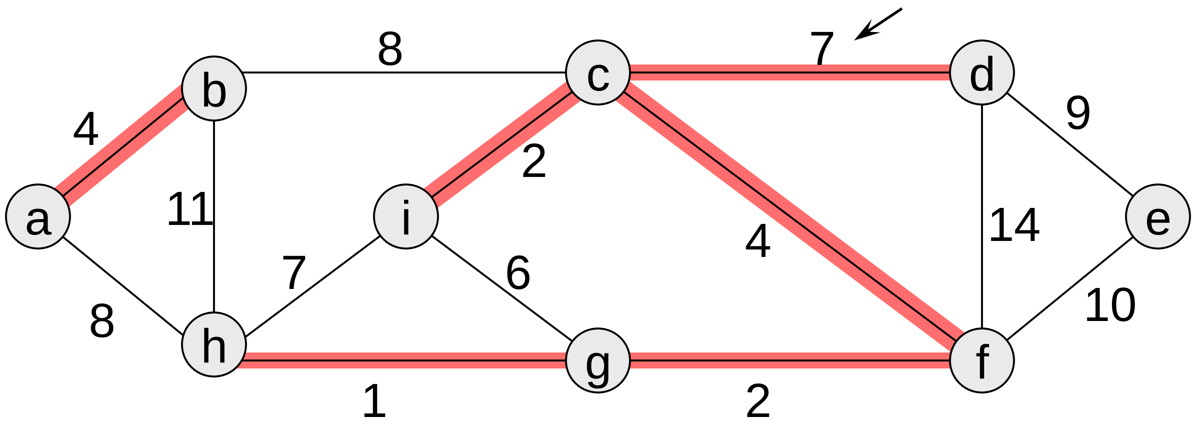


(f)

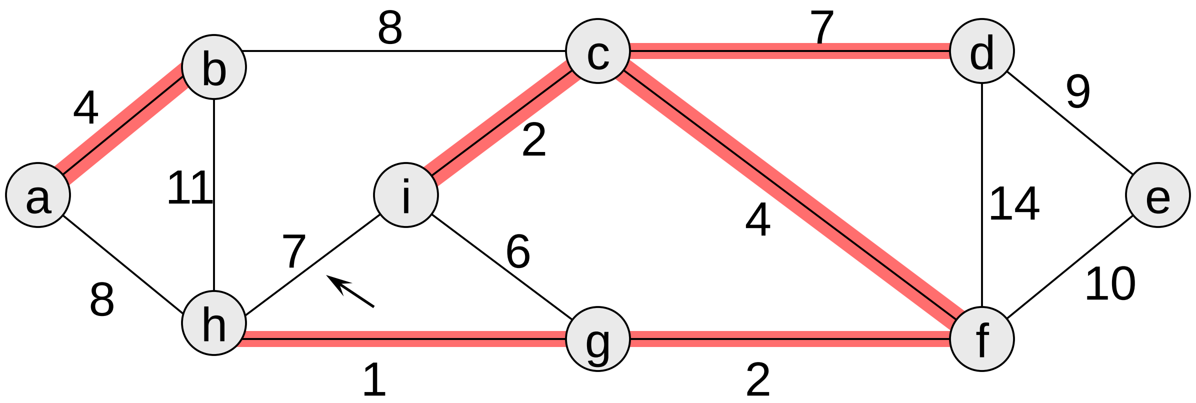


Kruskal’s Algorithm - Solution

(g)

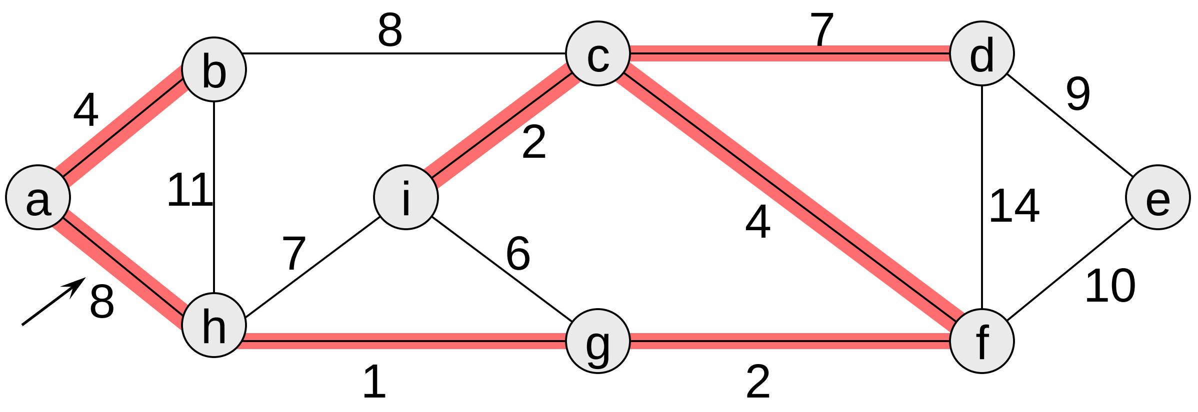


(h)

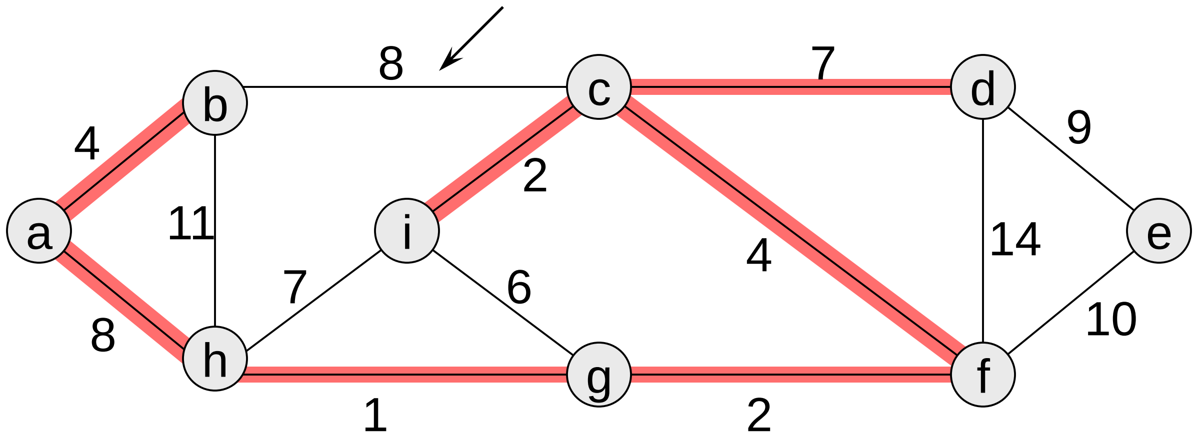


Kruskal's Algorithm - Solution

(i)

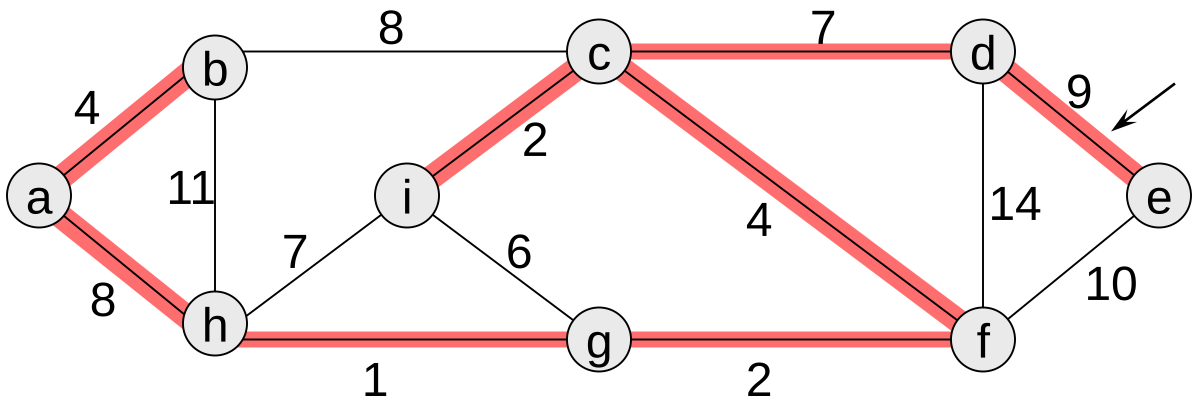


(j)

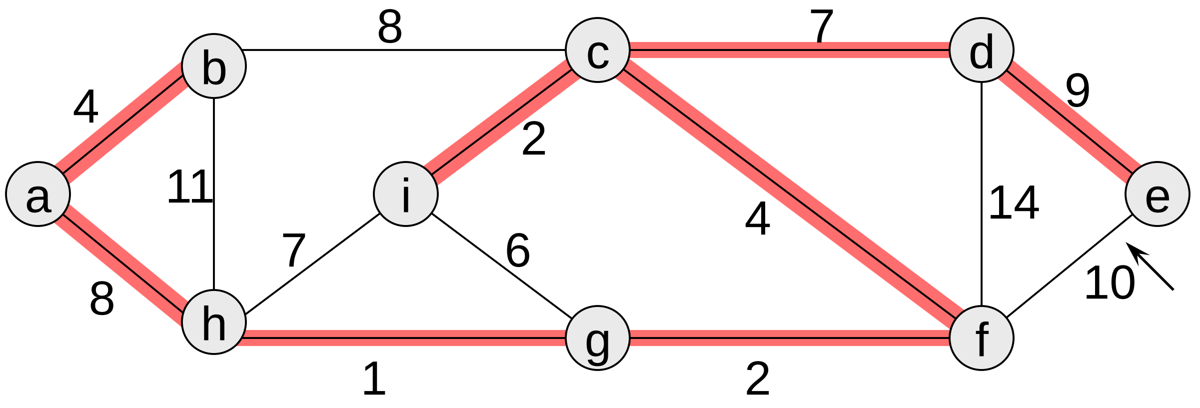


Kruskal's Algorithm - Solution

(k)

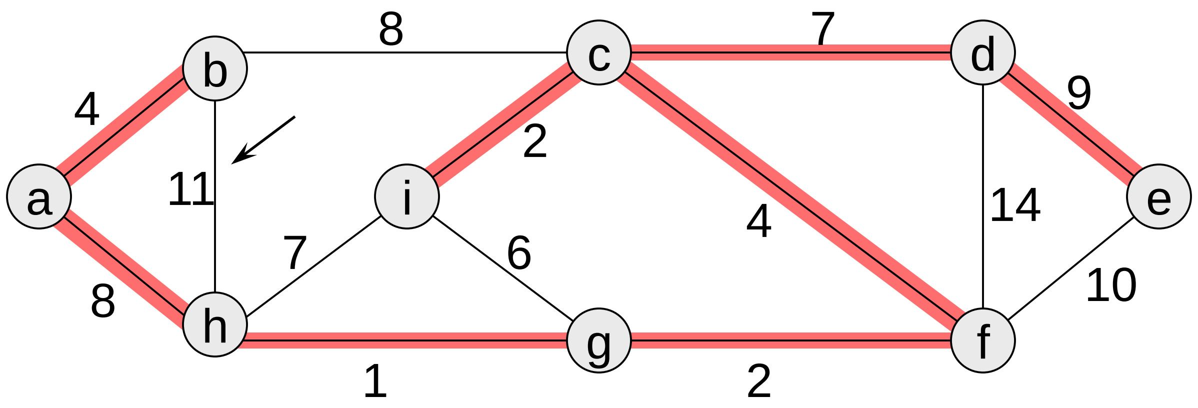


(l)

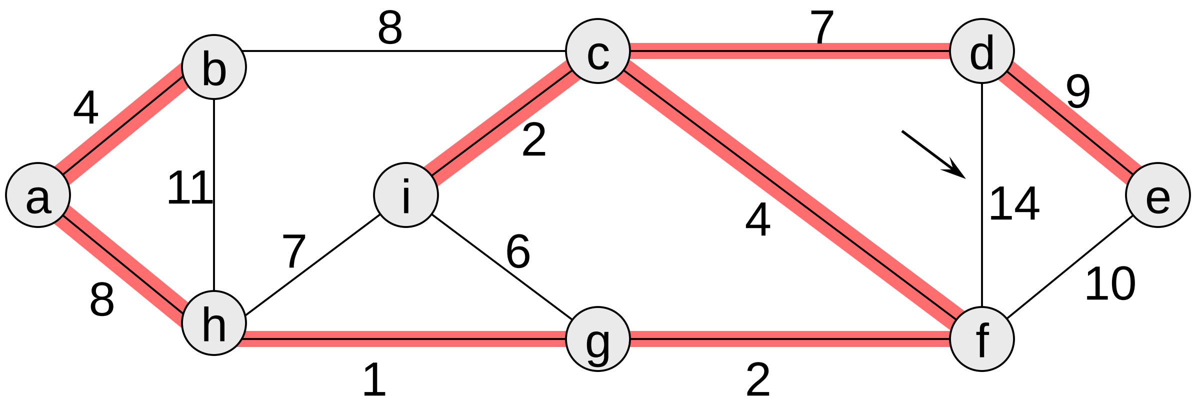


Kruskal's Algorithm - Solution

(m)



(n)



Kruskal's Algorithm – Runtime Analysis

Run time depends on the operations on the disjoint sets data structure:

- Initialize the set A: O(1)
- First for loop: |V| MAKE-SETs
- Sort E: O(E lg E)
- Second for loop: O(E) FIND-SETs and UNIONs => O(E lg E)

- Therefore, total run time is O(E lg E).
  $|E| \le |V|^2 \Rightarrow \lg |E| = O(2 \lg V) = O(\lg V)$.

- Hence, O(E lg V) time.

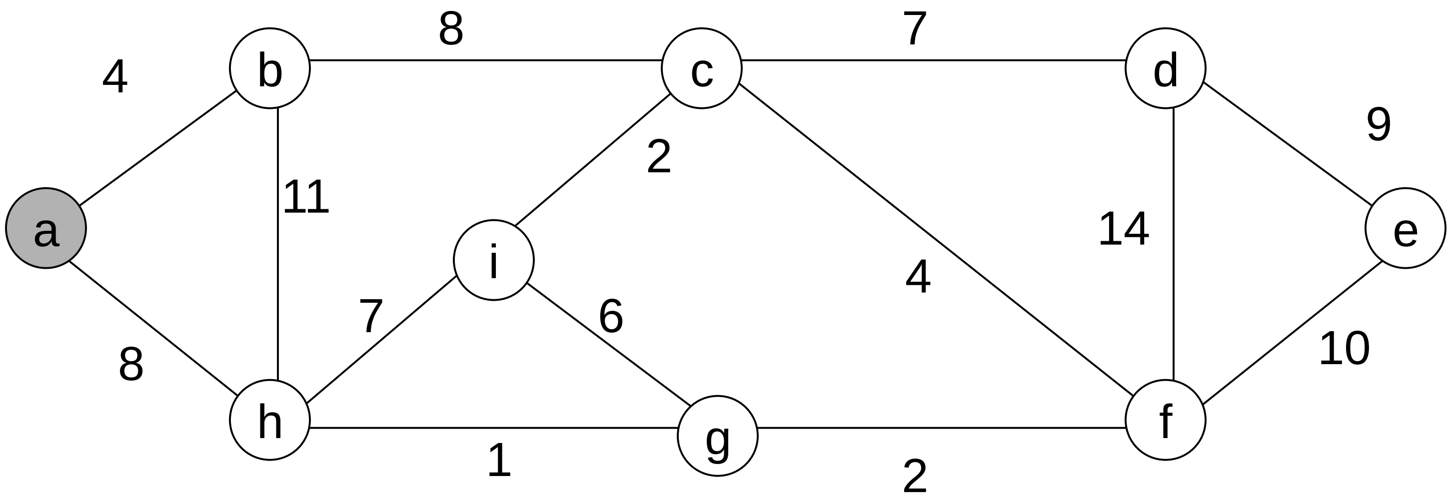
Prim’s Algorithm

- Edges in the set A always form a single tree.
- Tree starts from an arbitrary root vertex r and grows until the tree spans all the vertices in V.
- At each step a light edge is added to the tree A. The algorithm implicitly maintains the set A.
- This strategy is greedy.

Prim’s Algorithm

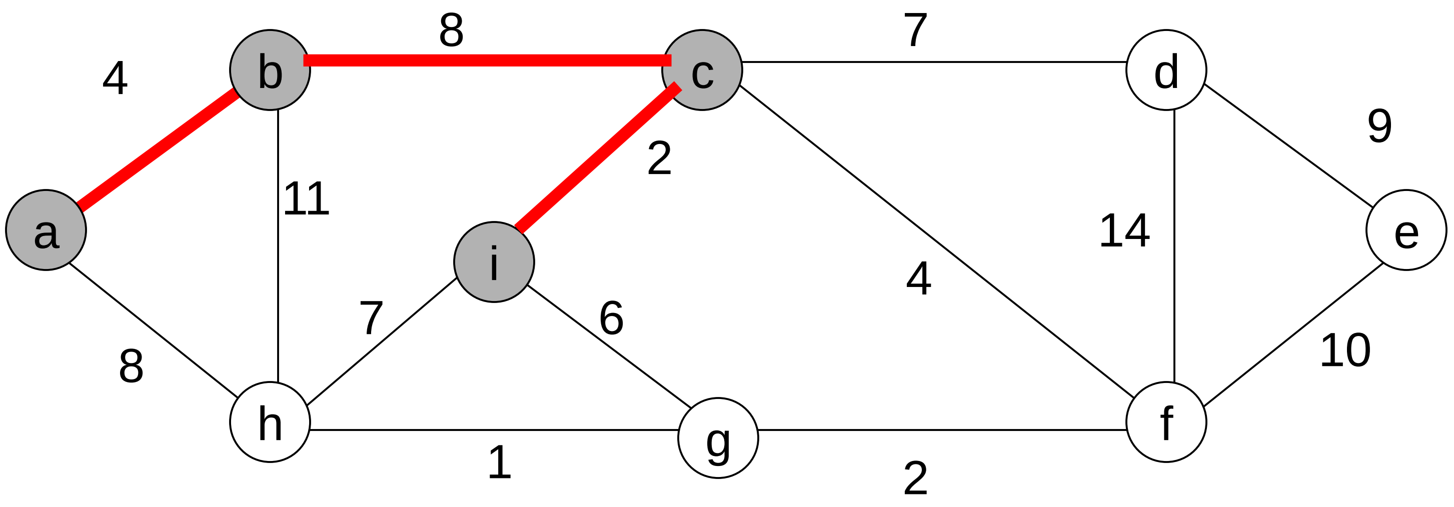
```
MST-PRIM (G, w, r)
for each u ∈ V [G]
    do key [u] ← ∞
       π [u] ← NIL
key [r ] ← 0
Q ← V [G]
while Q ≠ ∅
    do u ← EXTRACT-MIN (Q)
       for each v ∈ Adj[u]
          do if v ∈ Q and w (u, v) < key [v]
                then π [v] ← u
                     key [v] ← w (u, v)
```

Prim’s Algorithm - Problem

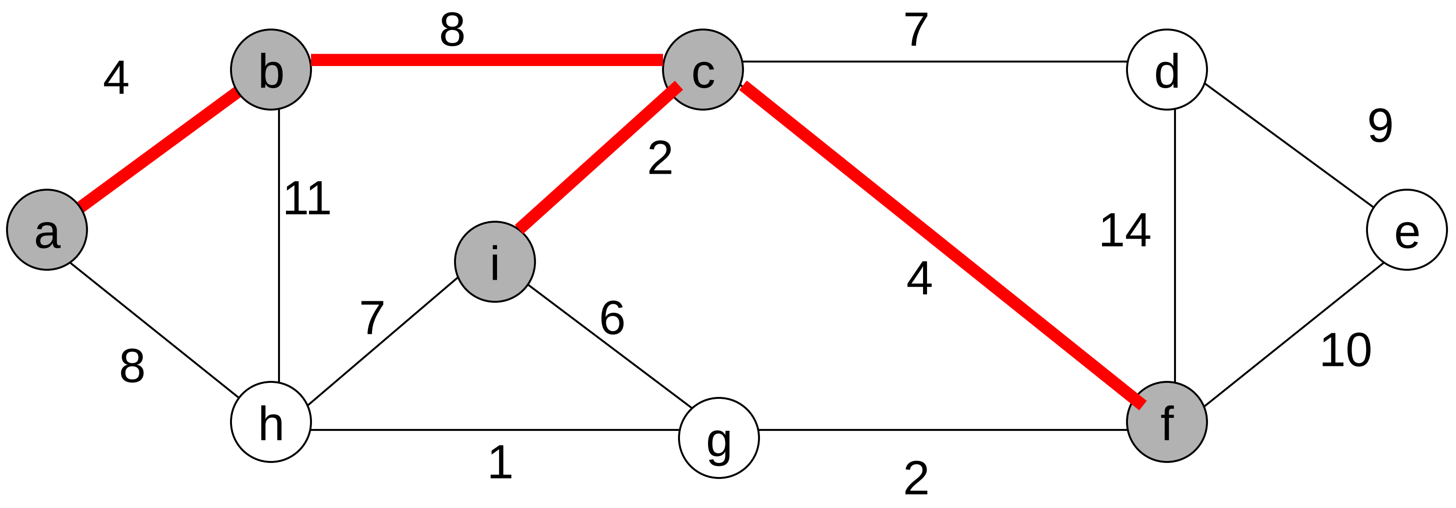


Prim’s Algorithm - Solution

d)

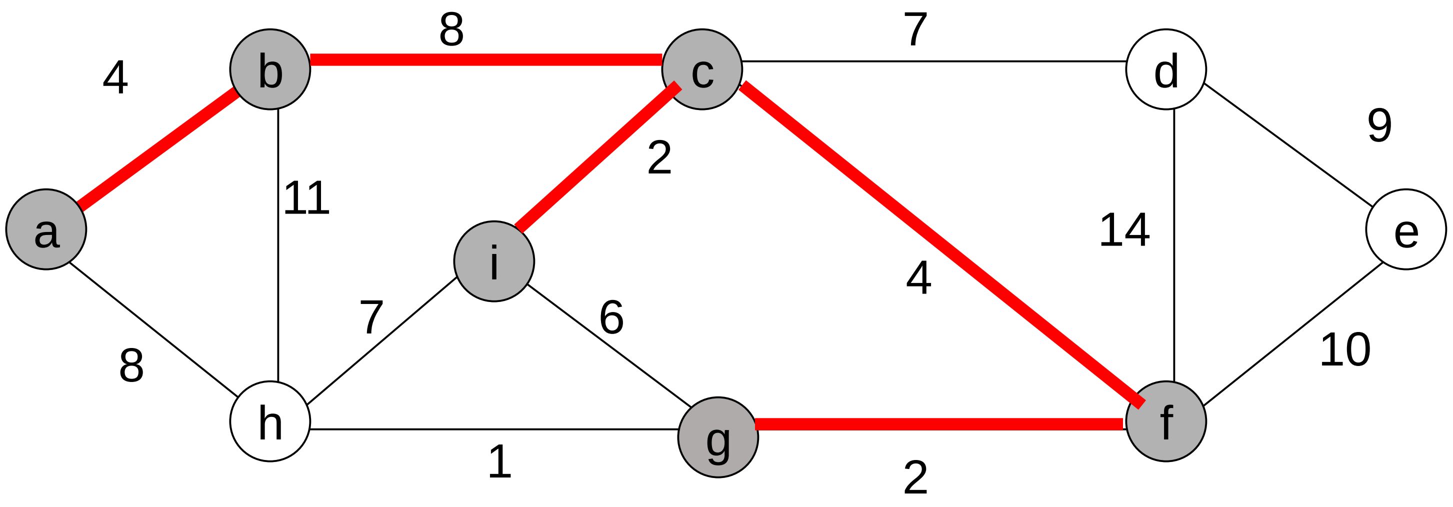


e)

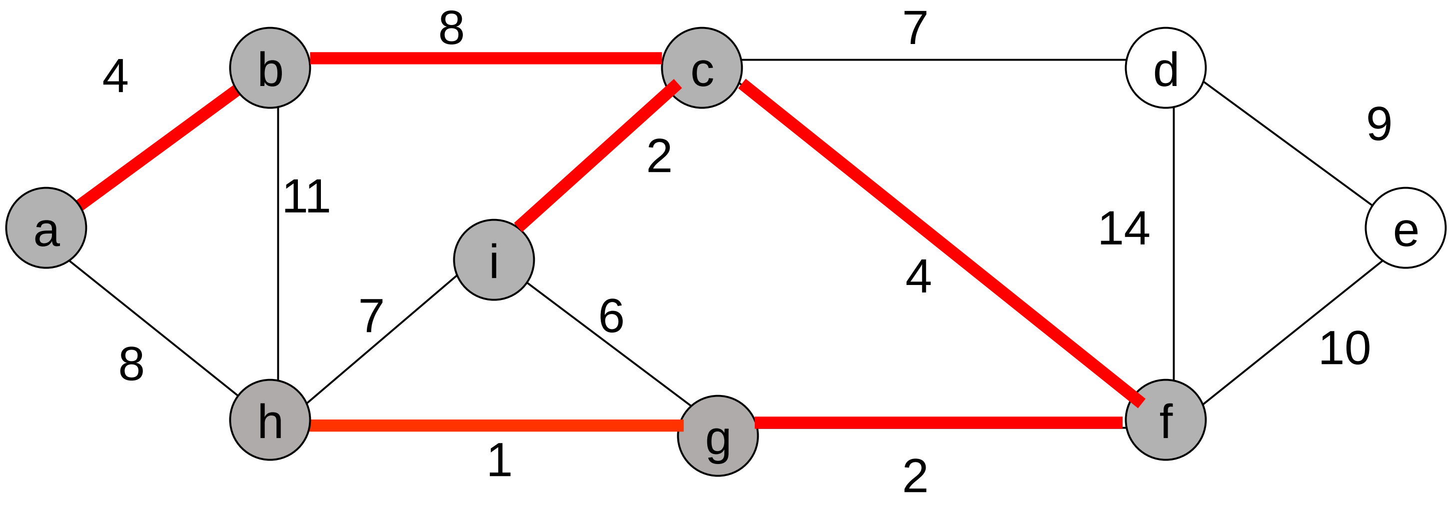


Prim's Algorithm - Solution

f)

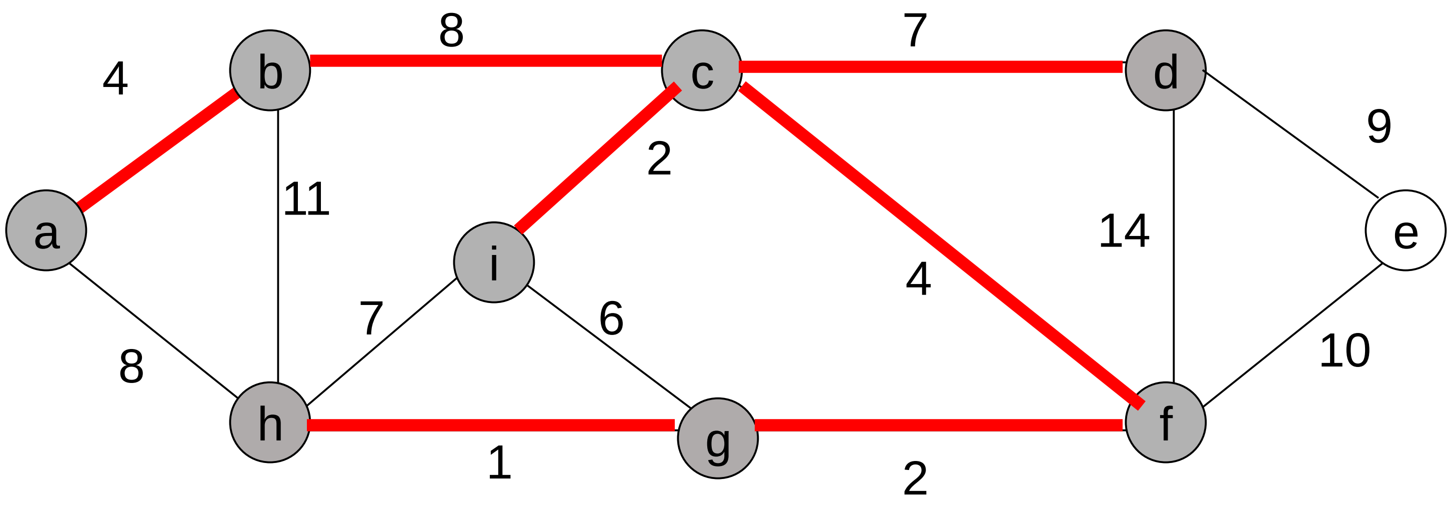


g)

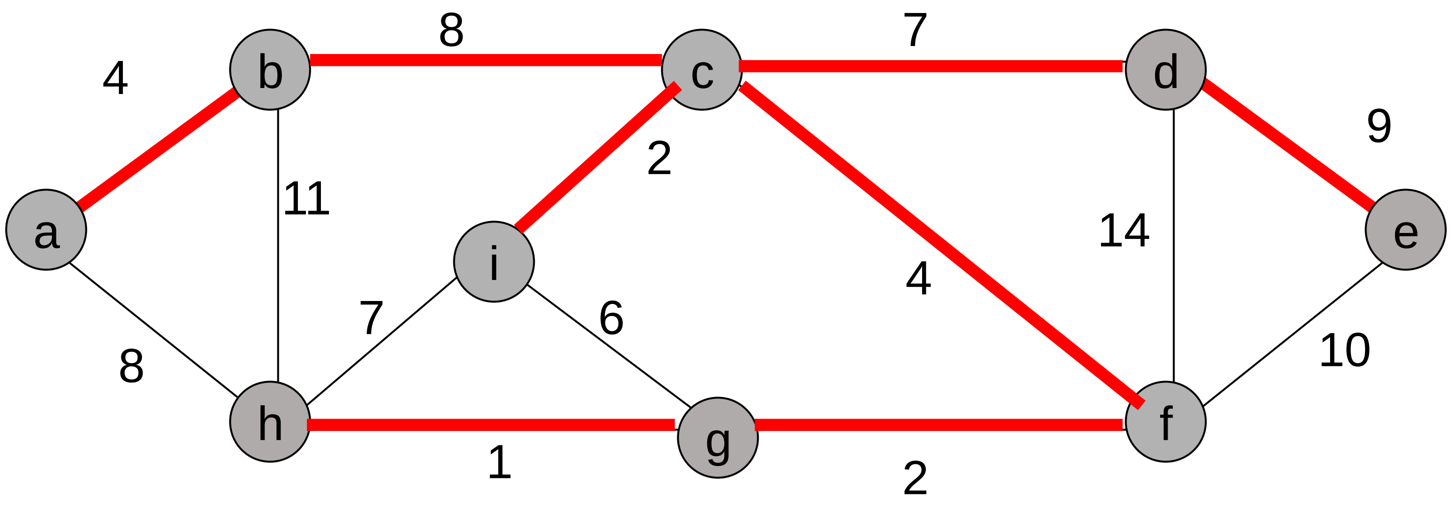


Prim's Algorithm - Solution

h)



i)



Prim's Algorithm - Analysis

- Maintains a min-priority queue by calling three priority queue operations:
  - INSERT
  - EXTRACT-MIN
  - DECREASE-KEY

- Running time of Prim's Algorithm depends on how the min-priority queue is implemented.

Prim's Algorithm - Analysis

- Binary min-heap

  Building min binary heap O(V)

  DECREASE-KEY O(lg V) - E

  EXTRACT-MIN O(lg V) - V

  Total time O ((V + E) lg V)

  = O (E lg V)

Prim's Algorithm - Analysis

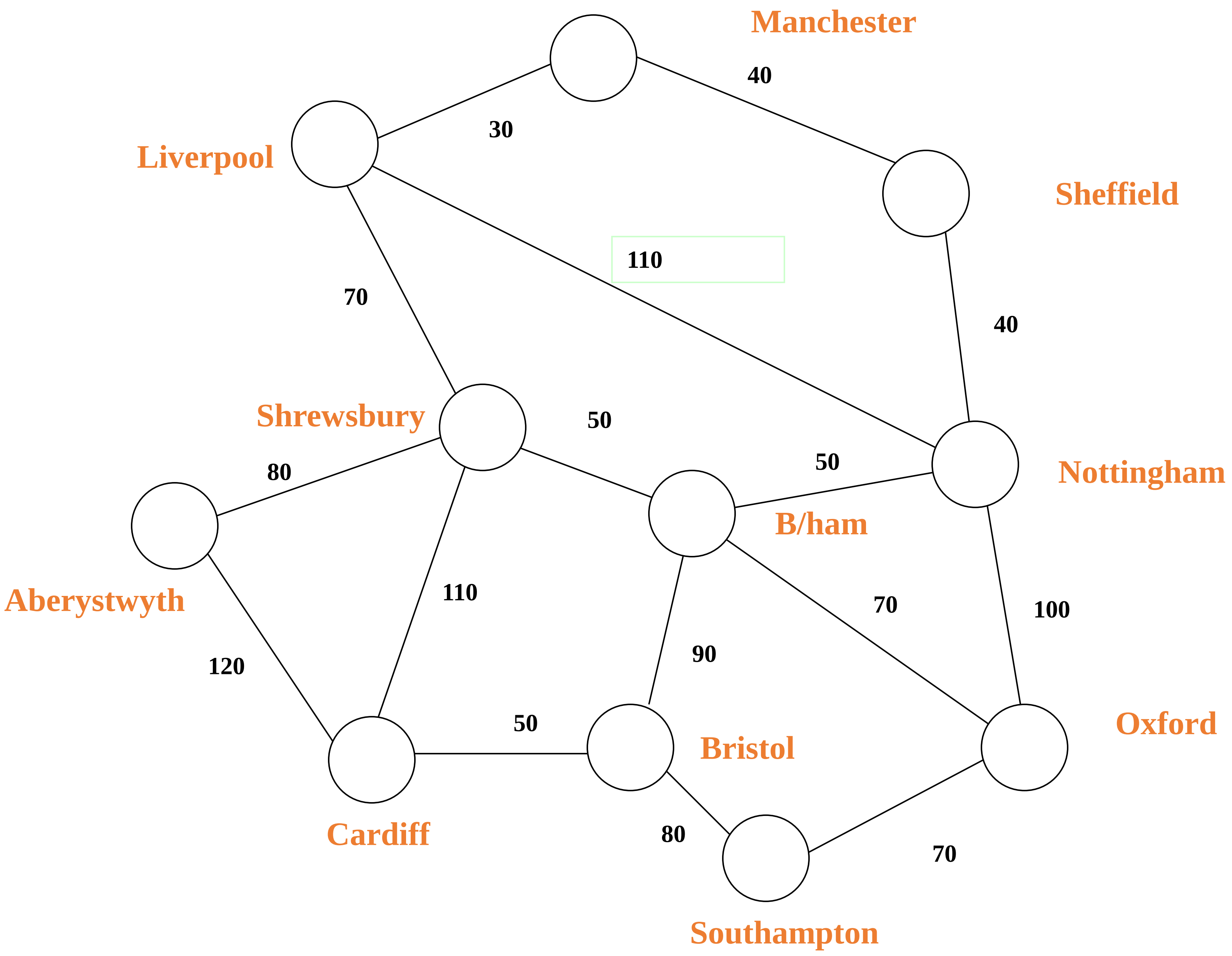
- Fibonacci heap

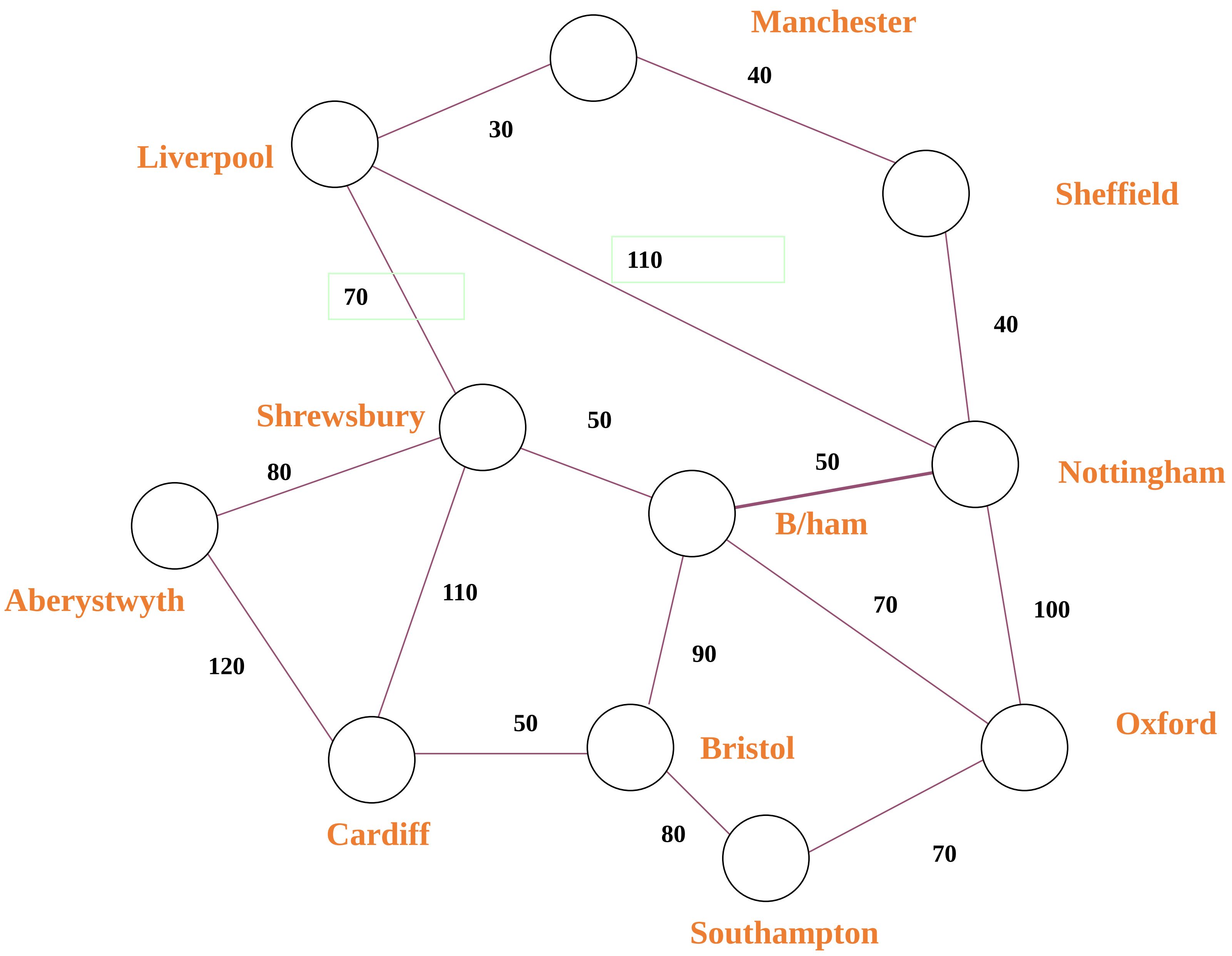
  Building Fibonacci heap O(V)

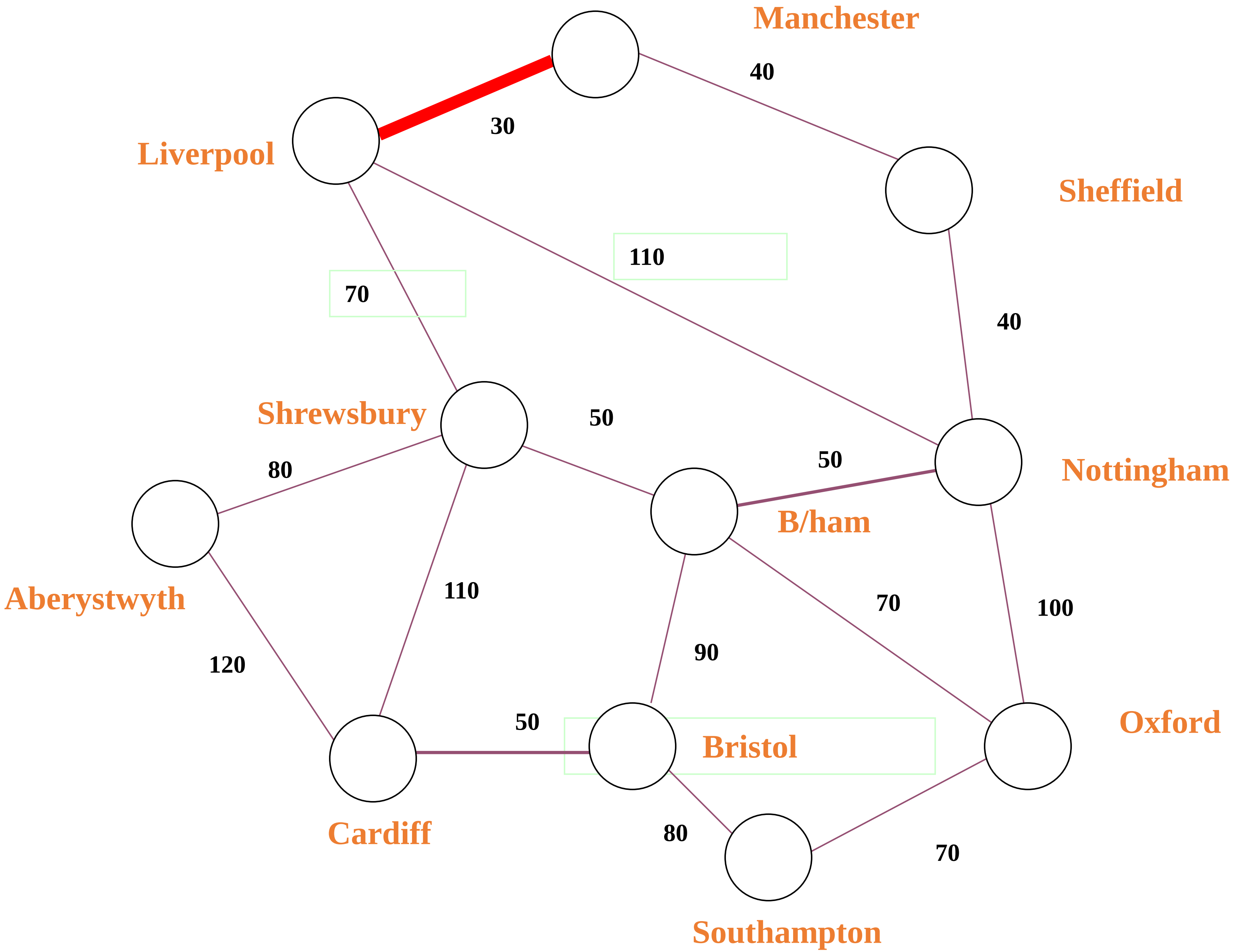
  DECREASE-KEY O(1) - E

  EXTRACT-MIN O(lg V) - V

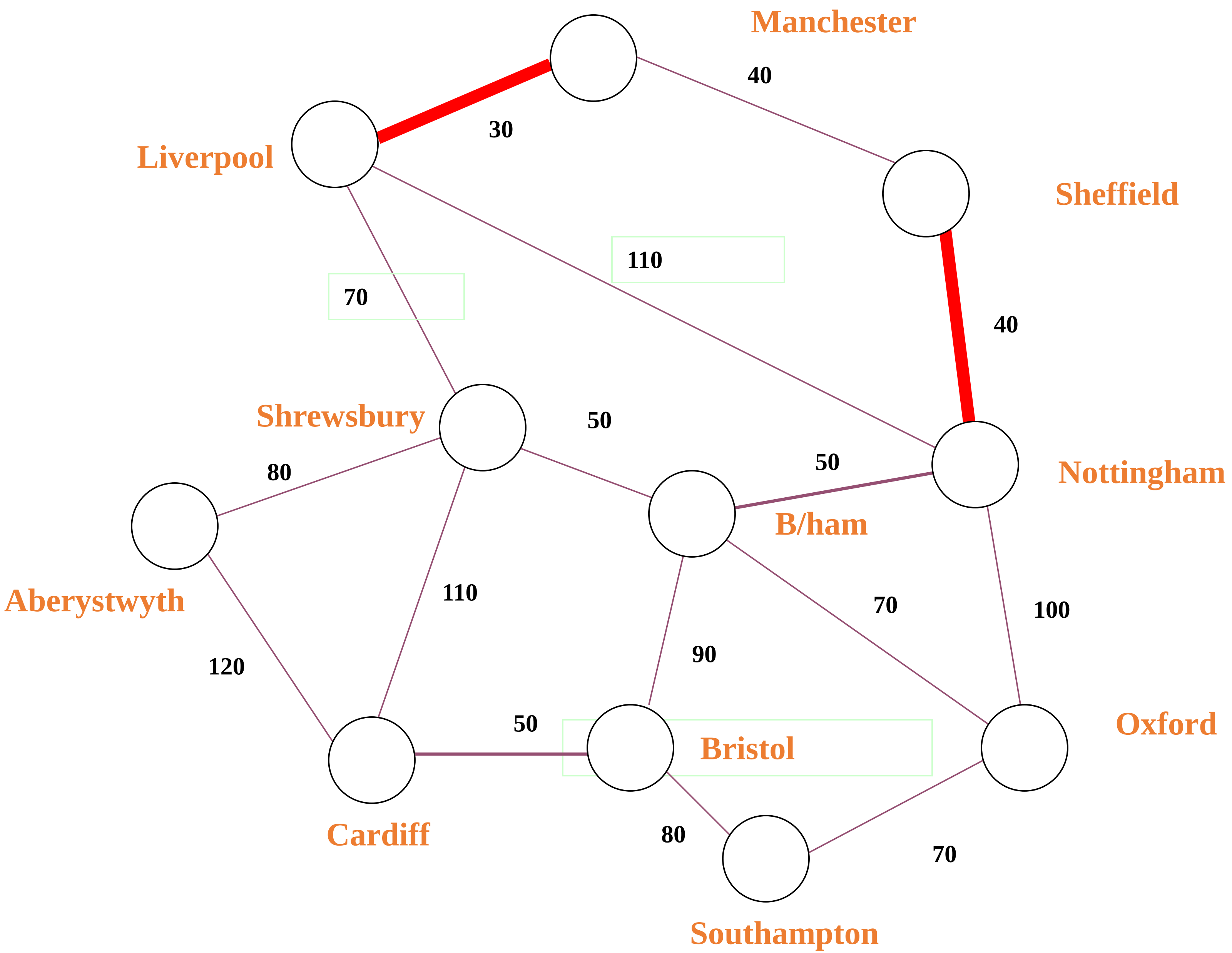
  Total time O (V lg V + E)

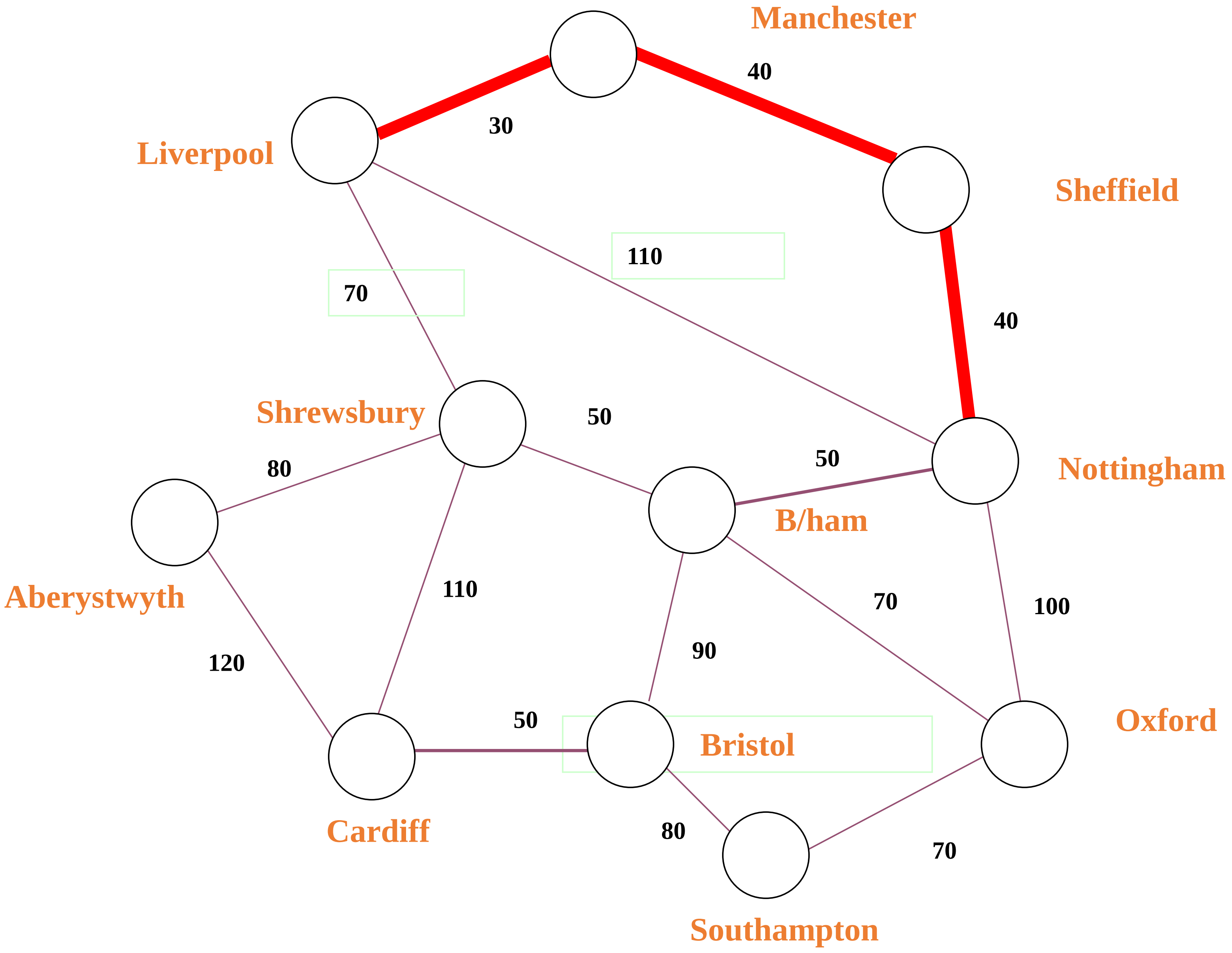

Manchester
40
30
Liverpool
Sheffield
110
70
40
Shrewsbury
50
50
80
Nottingham
B/ham
Aberystwyth
110
70
100
90
120
Oxford
50
Bristol
Cardiff
80
70
Southampton

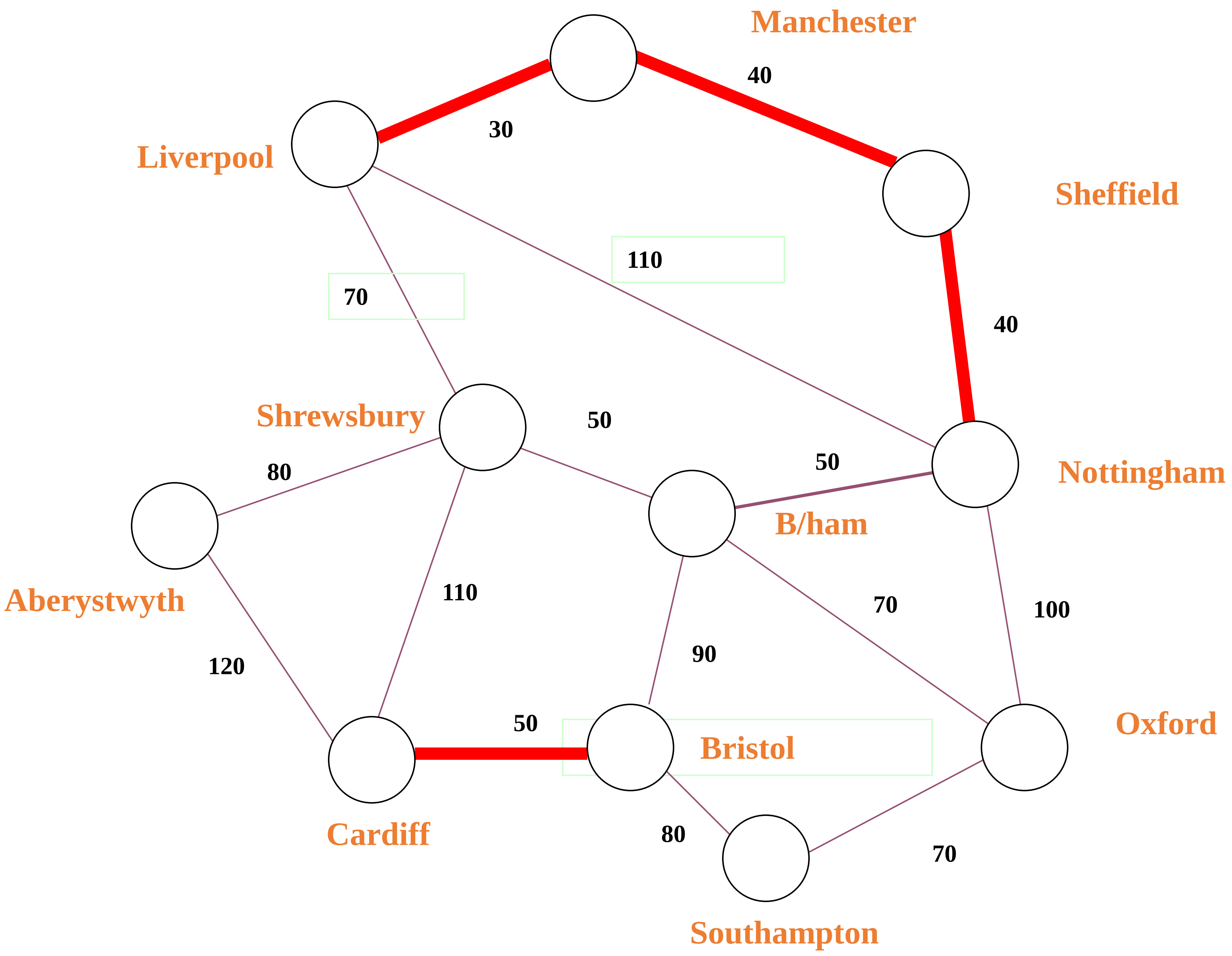

Manchester
40
30
Liverpool
Sheffield
110
70
40
Shrewsbury
50
50
Nottingham
80
B/ham
Aberystwyth
110
70
100
90
120
50
Oxford
Bristol
Cardiff
80
70
Southampton

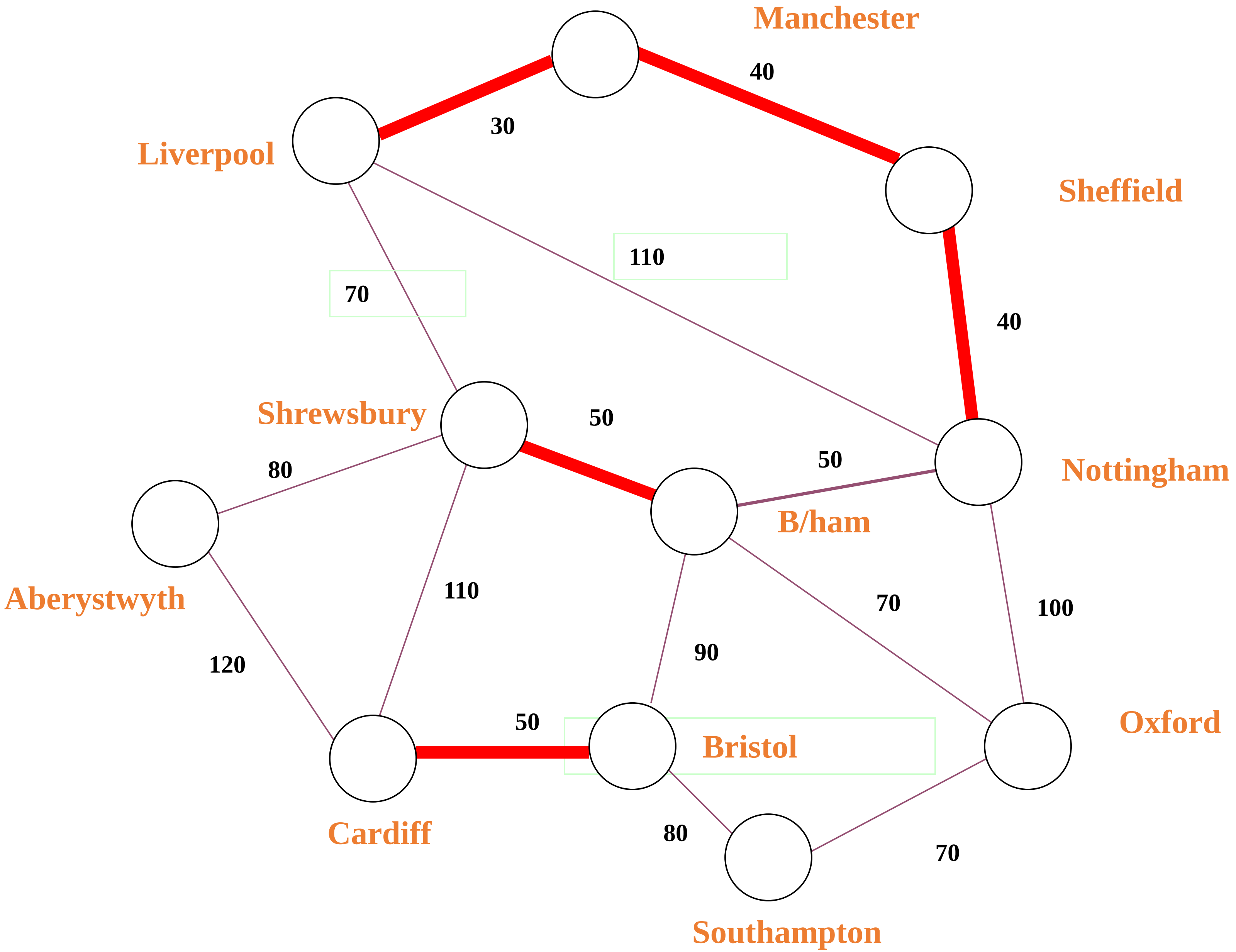


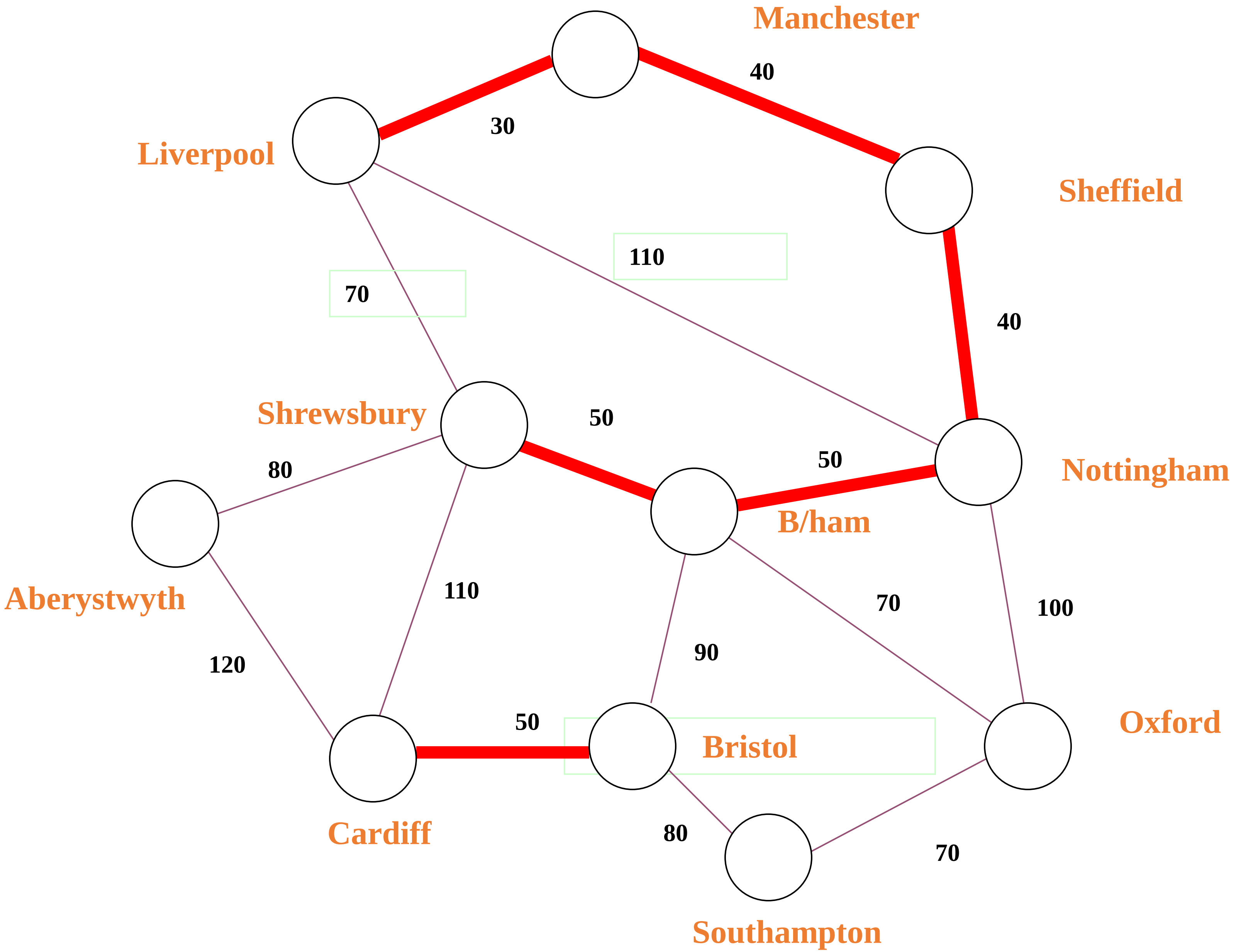
Manchester
40
30
Liverpool
Sheffield
110
70
40
Shrewsbury
50
50
80
Nottingham
B/ham
Aberystwyth
110
70
100
90
120
50
Oxford
Bristol
Cardiff
80
70
Southampton


Manchester
40
30
Liverpool
Sheffield
110
70
40
Shrewsbury
50
50
Nottingham
80
B/ham
Aberystwyth
110
70
100
90
120
Oxford
50
Bristol
Cardiff
80
70
Southampton

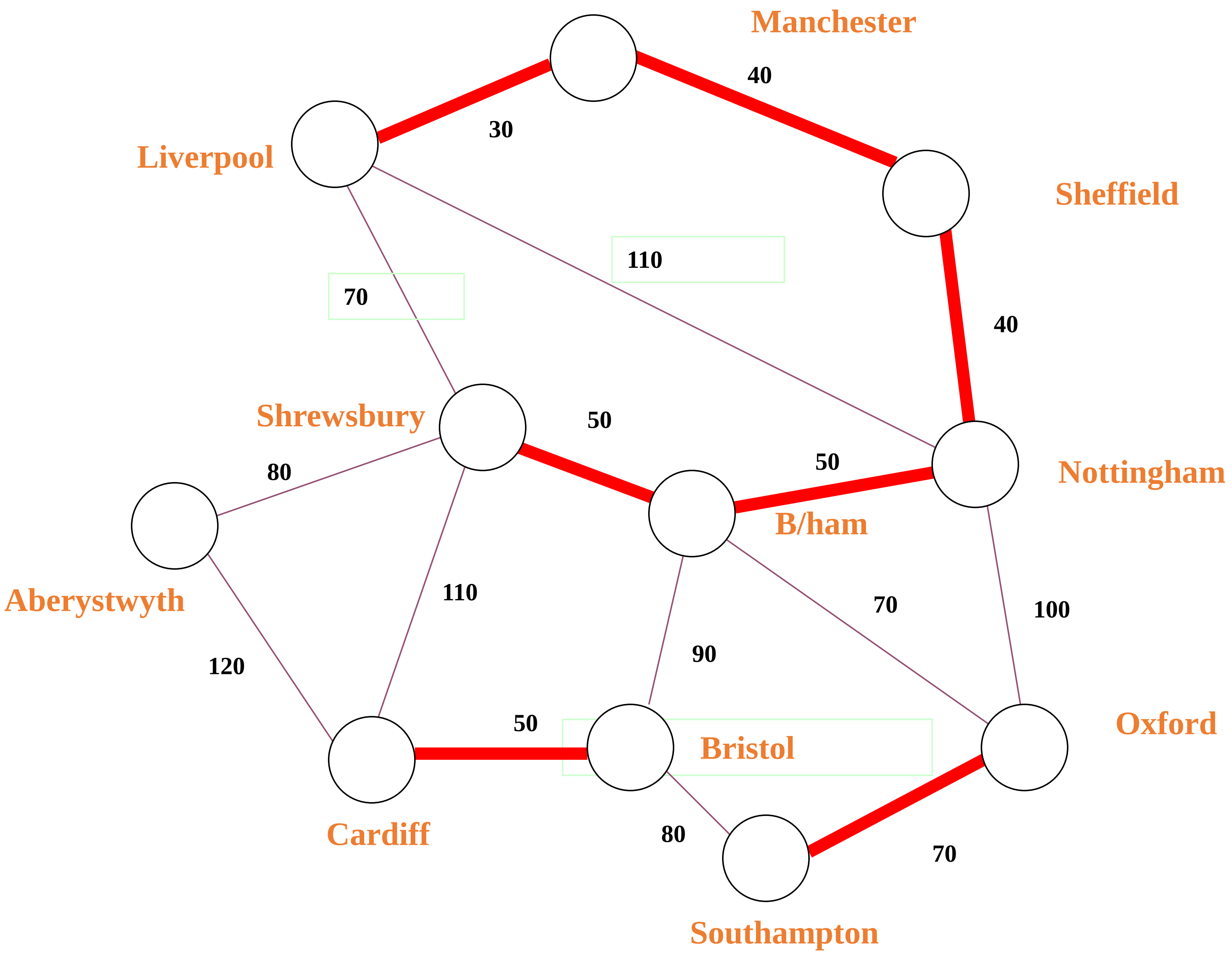

Manchester
40
30
Liverpool
Sheffield
110
70
40
Shrewsbury
50
50
Nottingham
80
B/ham
Aberystwyth
110
70
100
90
120
Oxford
50
Bristol
Cardiff
80
70
Southampton

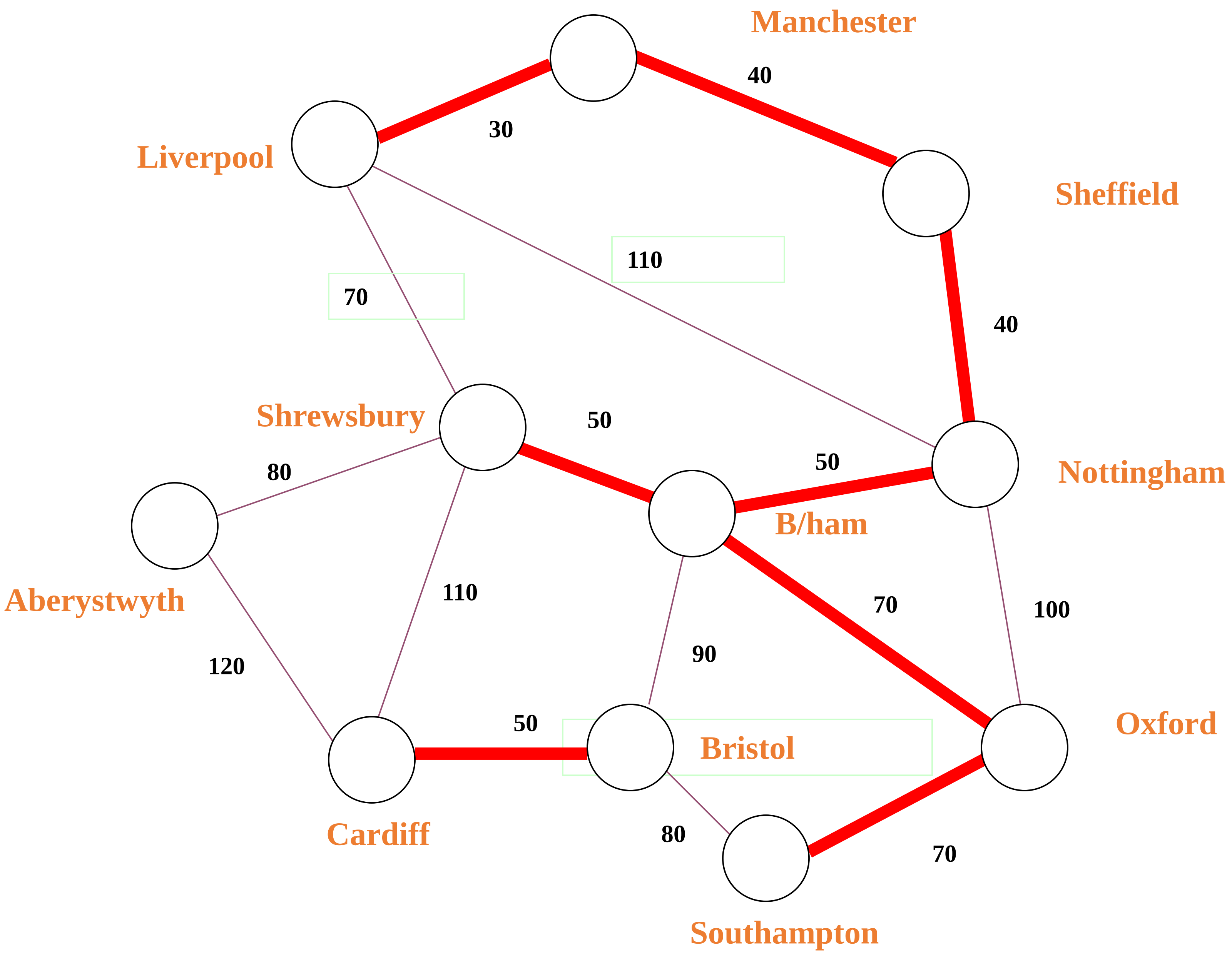

Manchester
40
30
Liverpool
Sheffield
110
70
40
Shrewsbury
50
50
Nottingham
80
B/ham
Aberystwyth
110
70
100
90
120
50
Oxford
Bristol
Cardiff
80
70
Southampton

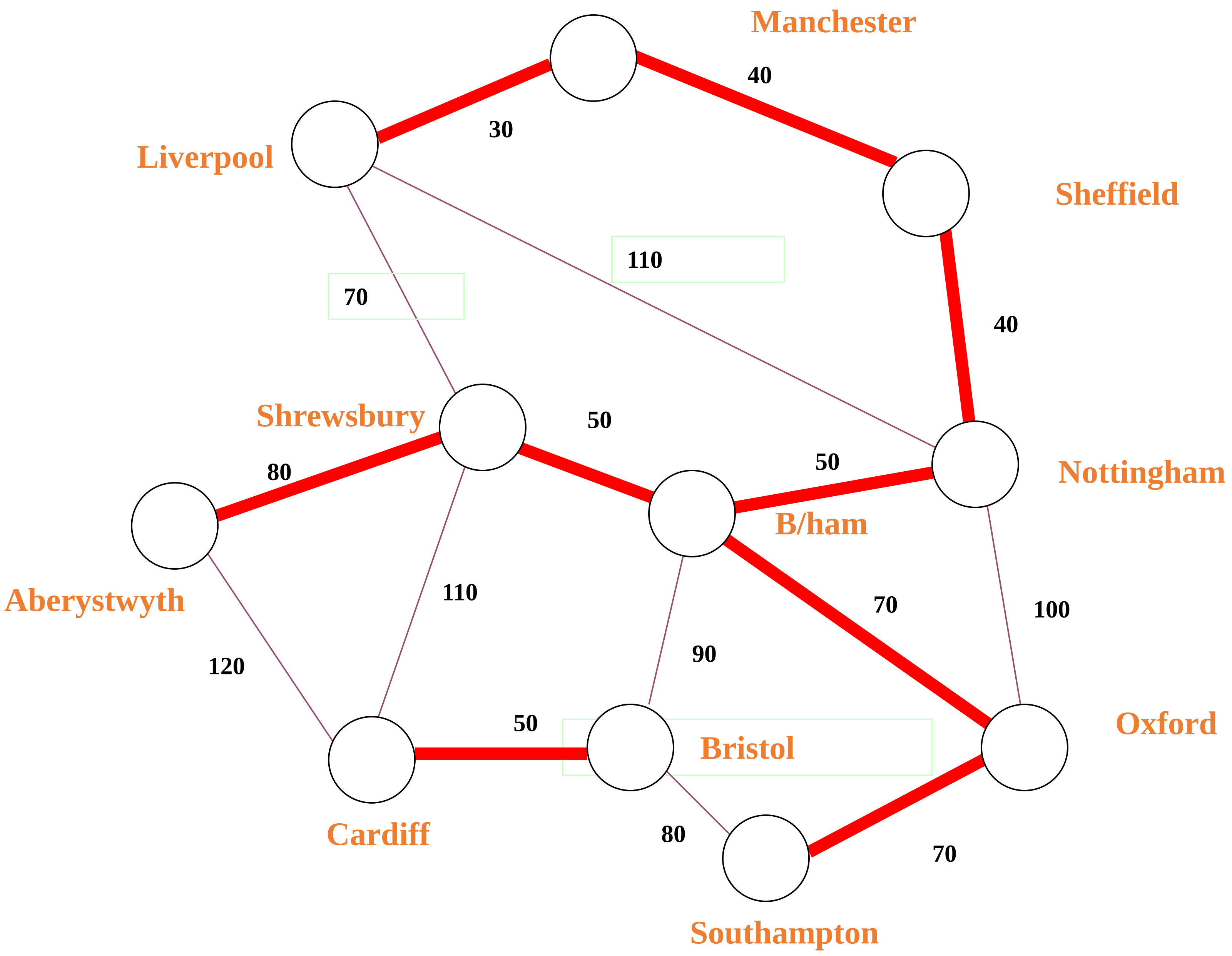

Manchester
40
30
Liverpool
Sheffield
110
70
40
Shrewsbury
50
80
50
Nottingham
B/ham
Aberystwyth
110
70
100
90
120
50
Bristol
Oxford
Cardiff
80
70
Southampton

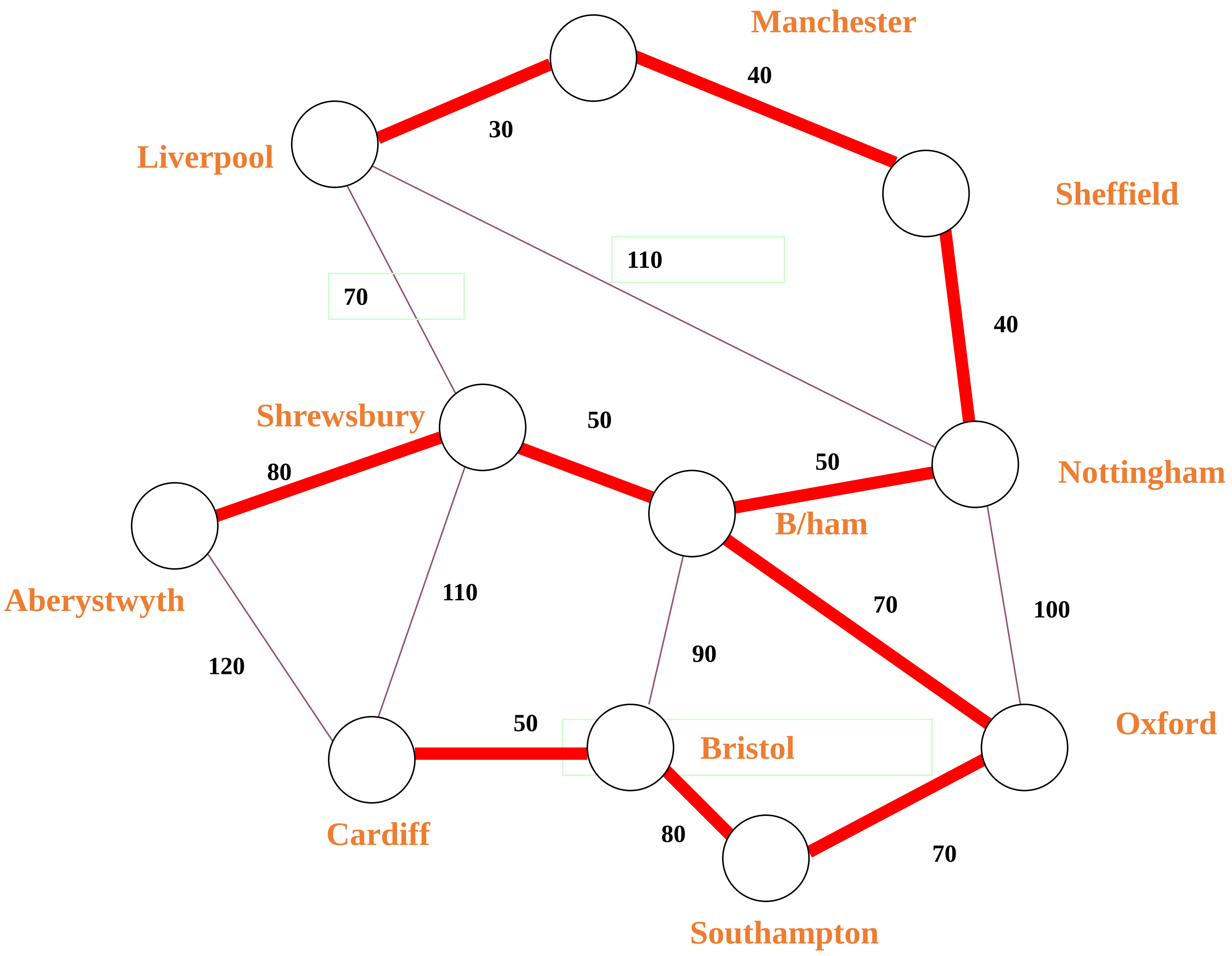


Manchester
40
30
Liverpool
Sheffield
110
70
40
Shrewsbury
50
50
80
Nottingham
B/ham
Aberystwyth
110
70
100
90
120
50
Oxford
Bristol
Cardiff
80
70
Southampton


Manchester
40
30
Liverpool
Sheffield
110
70
40
Shrewsbury
50
50
Nottingham
80
B/ham
Aberystwyth
110
70
100
90
120
Oxford
50
Bristol
Cardiff
80
70
Southampton


Manchester
40
30
Liverpool
Sheffield
110
70
40
Shrewsbury
50
50
80
Nottingham
B/ham
Aberystwyth
110
70
100
90
120
Oxford
50
Bristol
Cardiff
80
70
Southampton


Manchester
40
30
Liverpool
Sheffield
110
70
40
Shrewsbury
50
50
80
Nottingham
B/ham
Aberystwyth
110
70
100
90
120
50
Oxford
Bristol
Cardiff
80
70
Southampton


Manchester
40
30
Liverpool
Sheffield
110
70
40
Shrewsbury
50
50
Nottingham
80
B/ham
Aberystwyth
110
70
100
90
120
Oxford
50
Bristol
Cardiff
80
70
Southampton