Minimum Spanning Tree

CSE 326

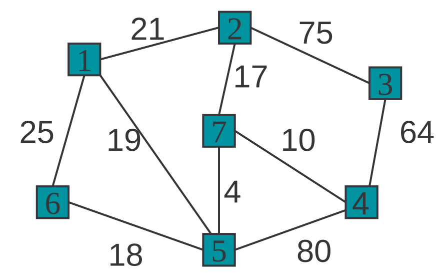
Data Structures

Unit 14

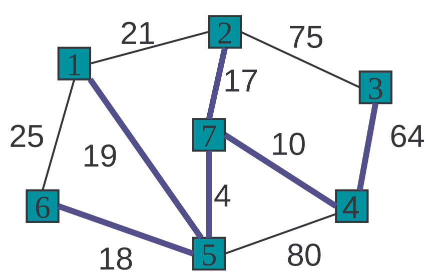
Reading: Chapter 9.5

Minimum Spanning Tree

- Each edge has a cost.
- Find a minimal-cost subset of edges that will keep the graph connected. (must be a ST).



Example of a Spanning Tree



Price of this tree = 18+19+4+10+17+64

Minimum Spanning Tree Problem

- Input: Undirected connected graph $G = (V,E)$ and a cost function C from E to the reals. $C(e)$ is the cost of edge e.
- Output: A spanning tree T with minimum total cost. That is: T that minimizes

$$C(T) = \sum_{e \in T} C(e)$$

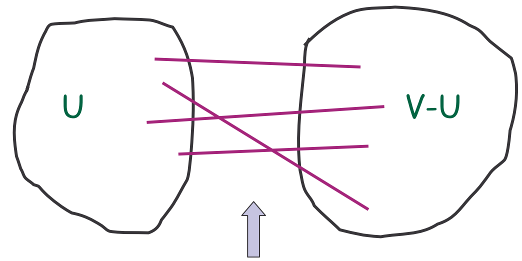
- Another formulation: Remove from G edges with maximal total cost, but keep G connected.

Minimum Spanning Tree

- Boruvka 1926
- Kruskal 1956
- Prim 1957 also by Jarnik 1930
- Karger, Klein, Tarjan 1995
  - Randomized linear time algorithm
  - Probably not practical, but very interesting

Minimum Spanning Tree Problem

- Definition: For a given partition of V into U and V-U, the cut defined by U is the set of edges with one end in U and one end in V-U.



The cut defined by U

An Algorithm for MST

- The algorithm colors the edges of the graph. Initially, all edges are black.
- A blue edge – belongs to T.
- A red edge – does not belong to T.
- We continue to color edges until we have n-1 blue edges.
- How do we select which edge to color next? How do we select its color?

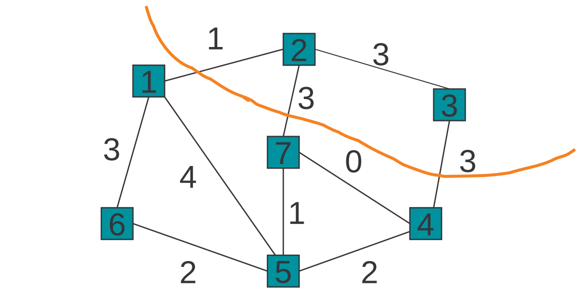
The Blue/Red Edge-coloring Rules

- The blue rule: Find a cut with no blue edge. Color blue the cheapest black edge in the cut.
- The red rule: Find a cycle with no red edge. Color red the most expensive black edge in the cycle.

Ø These rules can be applied in any order.

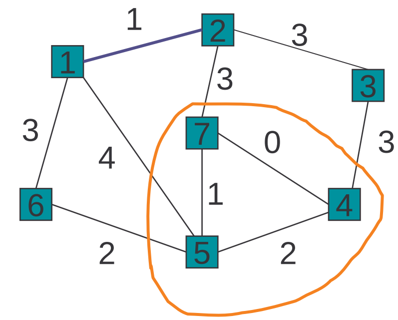
Ø We will see two specific algorithms.

Example of Blue/Red rules (1)



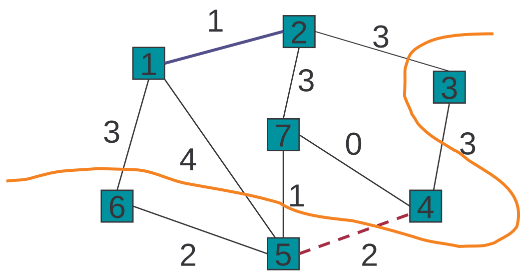
Consider the cut defined by {2,3}
- color (1,2) blue

Example of Blue/Red rules (2)



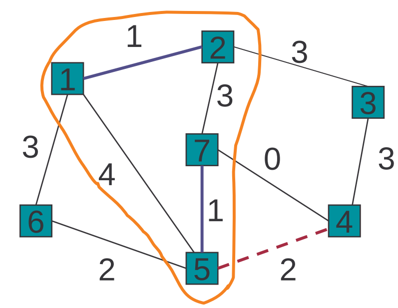
Consider the cycle (7-5-4)
- color (4,5) red

Example of Blue/Red rules (3)



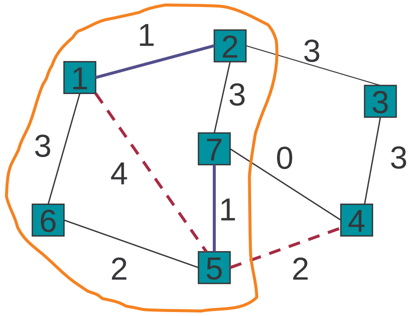
Consider the cut defined by {3,5,6}
- color (5,7) blue

Example of Blue/Red rules (4)



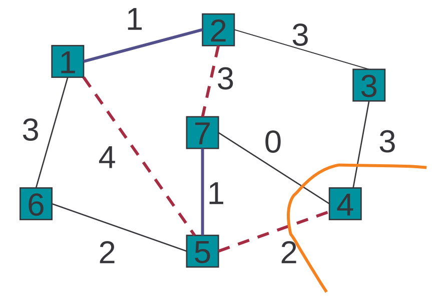
Consider the cycle (1-2-7-5)
- color (1,5) red

Example of Blue/Red rules (5)



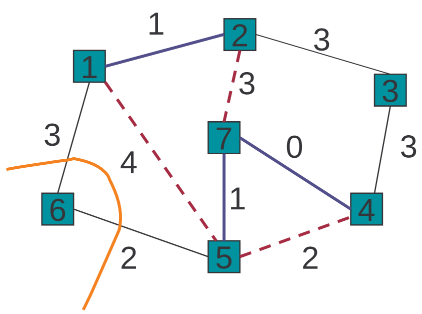
Consider the cycle (1-2-7-5-6)
- color (2,7) red.

Example of Blue/Red rules (6)



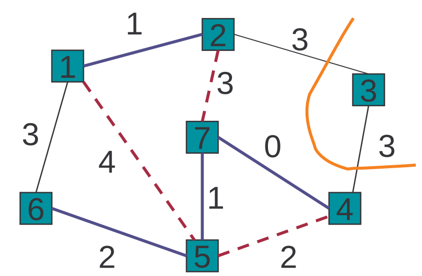
Consider the cut defined by {4}
- color (4,7) blue

Example of Blue/Red rules (7)



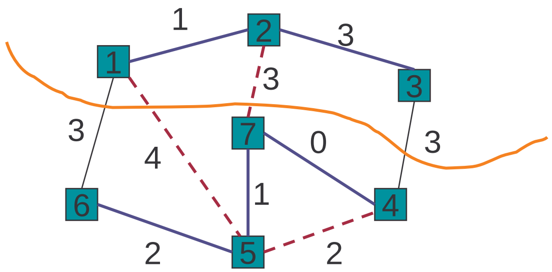
Consider the cut defined by {6}
- color (5,6) blue

Example of Blue/Red rules (8)



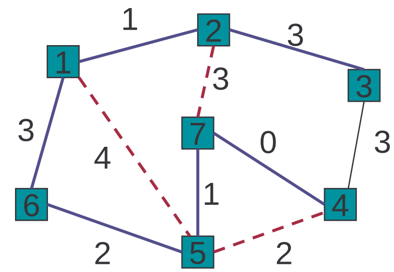
Consider the cut defined by {3}
- color (2,3) blue

Example of Blue/Red rules (9)



Consider the cut defined by {1,2,3}
- color (1,6) blue

Example of Blue/Red rules (10)



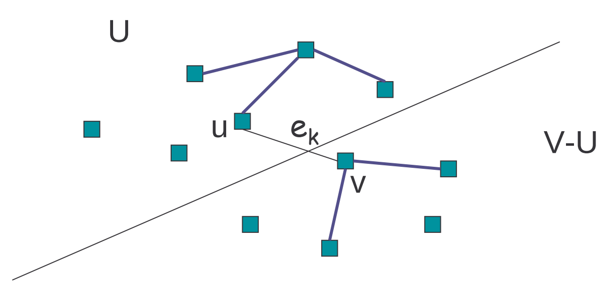
Final MST

Proof of Blue/Red Rules

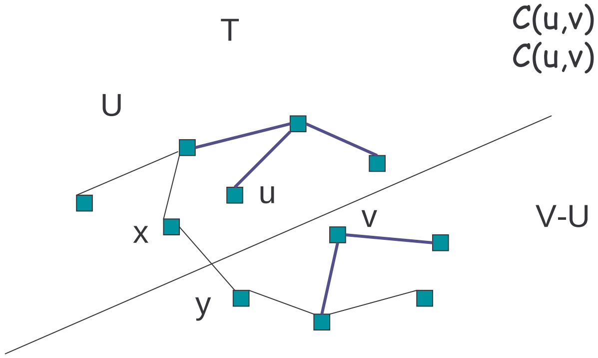
- Claim: for any $k \geq 0$, after we color k edges there exists an MST that includes all the blue edges and none of the red edges.
- Proof: By induction on k.
- Base: $k=0$ trivially holds.
- Step: Assume this is true after we color $k-1$ edges $e_1, e_2, \ldots, e_{k-1}$. Consider the coloring of e_k.

Case 1: Applying the Blue Rule

C(u,v) is minimal



Case 1: Applying the Blue Rule

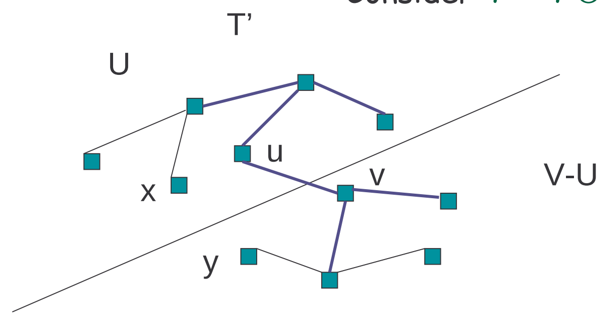


C(u,v) is minimal
C(u,v) $\leq$ C(x,y)

If (u,v)∉T, then T must includes some other edge (x,y) in the cut defined by U (T is connected, so there is a path u—v).

Case 1: Applying the Blue Rule

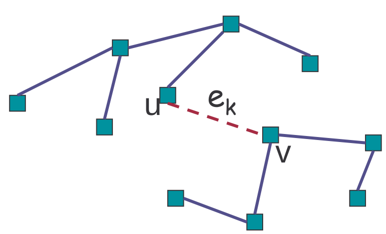
Consider T' = T ∪ (u,v) -(x,y)



C(T') = C(T) + C(u,v) - C(x,y)
C(T') $\leq$ C(T)

T' is also a minimum spanning tree, and it includes e_k

Case 2: Applying the Red Rule

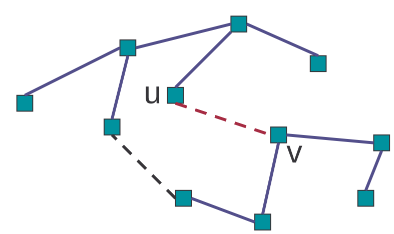


C(u,v) is maximal in some cycle

Assume (u,v)∈T.

By removing (u-v) from T we get two components.

Case 2: Applying the Red Rule



C(u,v) is maximal in some cycle

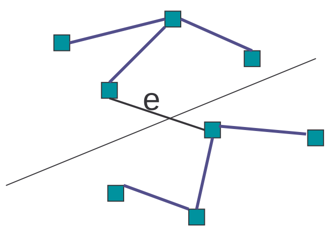
The cycle that causes us to color (u-v) red includes an edge connecting the two component (whose cost is at most c(u,v).

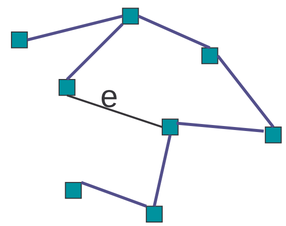
Ł There is an alternative MST, that does not include e_k

One more point: We can always proceed

Select an edge e.

- If e connects two blue sub-trees, then there is a cut without any blue edge and we can run the blue rule on this cut.
- Otherwise, e closes a cycle in which e is the most expensive edge (why?) so we can color e red.



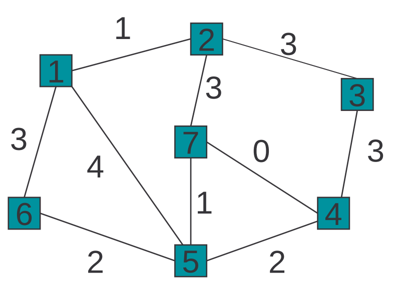


Kruskal's Greedy Algorithm

```
Sort the edges by increasing cost;
Initialize T to be empty;
For each edge e chosen in increasing order do
   if adding e does not form a cycle then
      add e to T
```

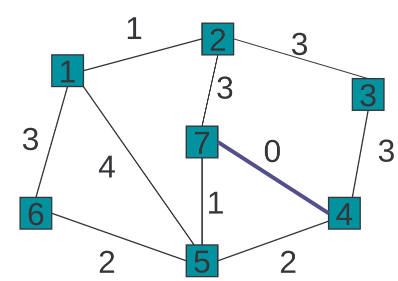
Proof: The algorithm follows the blue/red rules:
- If e closes a cycle – apply the red rule (by the sorting, e is the most expensive in this cycle).
- Otherwise – apply the blue rule (e connects two components, consider the cut defined by any of them. e is the cheapest edge in this cut)

Example of Kruskal 1



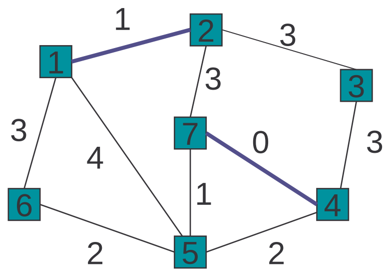
{7,4}	{2,1}	{7,5}	{5,6}	{5,4}	{1,6}	{2,7}	{2,3}	{3,4}	{1,5}
0	1	1	2	2	3	3	3	3	4

Example of Kruskal 2



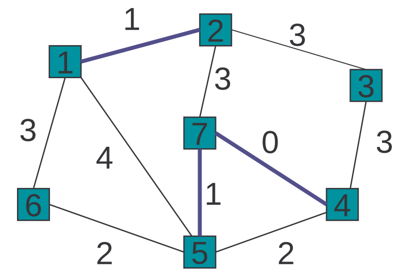
~~{7,4}~~	{2,1}	{7,5}	{5,6}	{5,4}	{1,6}	{2,7}	{2,3}	{3,4}	{1,5}
~~0~~	1	1	2	2	3	3	3	3	4

Example of Kruskal 2



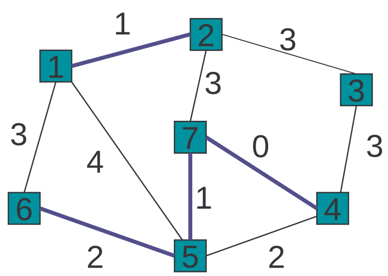
~~{7,4}~~	~~{2,1}~~	{7,5}	{5,6}	{5,4}	{1,6}	{2,7}	{2,3}	{3,4}	{1,5}
~~0~~	~~1~~	1	2	2	3	3	3	3	4

Example of Kruskal 3



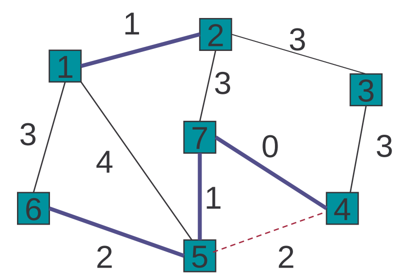
~~{7,4}~~	~~{2,1}~~	~~{7,5}~~	{5,6}	{5,4}	{1,6}	{2,7}	{2,3}	{3,4}	{1,5}
~~0~~	~~1~~	~~1~~	2	2	3	3	3	3	4

Example of Kruskal 4



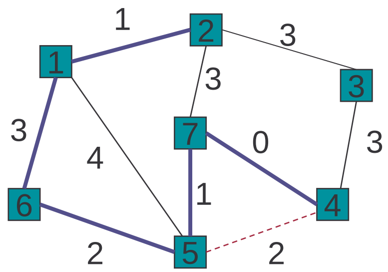
~~{7,4}~~	~~{2,1}~~	~~{7,5}~~	~~{5,6}~~	{5,4}	{1,6}	{2,7}	{2,3}	{3,4}	{1,5}
~~0~~	~~1~~	~~1~~	~~2~~	2	3	3	3	3	4

Example of Kruskal 5



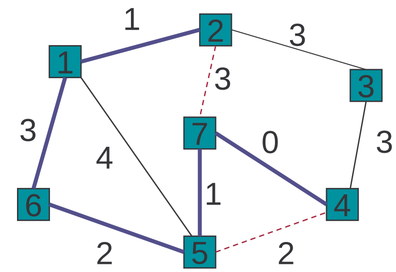
~~{7,4}~~	~~{2,1}~~	~~{7,5}~~	~~{5,6}~~	~~{5,4}~~	{1,6}	{2,7}	{2,3}	{3,4}	{1,5}
~~0~~	~~1~~	~~1~~	~~2~~	~~2~~	3	3	3	3	4

Example of Kruskal 6



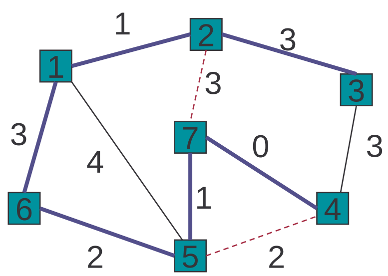
~~{7,4}~~	~~{2,1}~~	~~{7,5}~~	~~{5,6}~~	~~{5,4}~~	~~{1,6}~~	{2,7}	{2,3}	{3,4}	{1,5}
~~0~~	~~1~~	~~1~~	~~2~~	~~2~~	~~3~~	3	3	3	4

Example of Kruskal 7



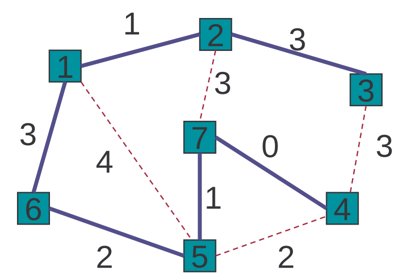
~~{7,4}~~	~~{2,1}~~	~~{7,5}~~	~~{5,6}~~	~~{5,4}~~	~~{1,6}~~	~~{2,7}~~	{2,3}	{3,4}	{1,5}
~~0~~	~~1~~	~~1~~	~~2~~	~~2~~	~~3~~	~~3~~	3	3	4

Example of Kruskal 8



~~{7,4}~~	~~{2,1}~~	~~{7,5}~~	~~{5,6}~~	~~{5,4}~~	~~{1,6}~~	~~{2,7}~~	~~{2,3}~~	{3,4}	{1,5}
~~0~~	~~1~~	~~1~~	~~2~~	~~2~~	~~3~~	~~3~~	~~3~~	3	4

Example of Kruskal 9



~~{7,4}~~	~~{2,1}~~	~~{7,5}~~	~~{5,6}~~	~~{5,4}~~	~~{1,6}~~	~~{2,7}~~	~~{2,3}~~	~~{3,4}~~	~~{1,5}~~
~~0~~	~~1~~	~~1~~	~~2~~	~~2~~	~~3~~	~~3~~	~~3~~	~~3~~	~~4~~

Data Structures for Kruskal

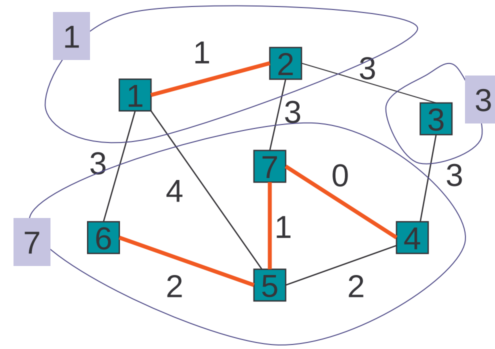
- Sorted edge list

{7,4} {2,1} {7,5} {5,6} {5,4} {1,6} {2,7} {2,3} {3,4} {1,5}
0 1 1 2 2 3 3 3 3 4

- Disjoint Union / Find
  - Union(a,b) - union the disjoint sets named by a and b
  - Find(a) returns the name of the set containing a

Remark: The set name is one of its members

Example of DU/F (1)

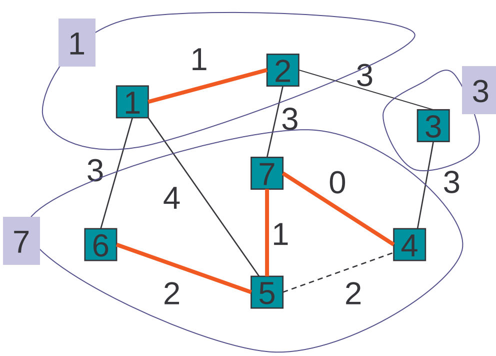


Find(5) = 7
Find(4) = 7

{7,4} {2,1} {7,5} {5,6} {5,4} {1,6} {2,7} {2,3} {3,4} {1,5}
0 1 1 2 2 3 3 3 3 4

u,v in the same set à (u,v) is not added to T

Example of DU/F (2)



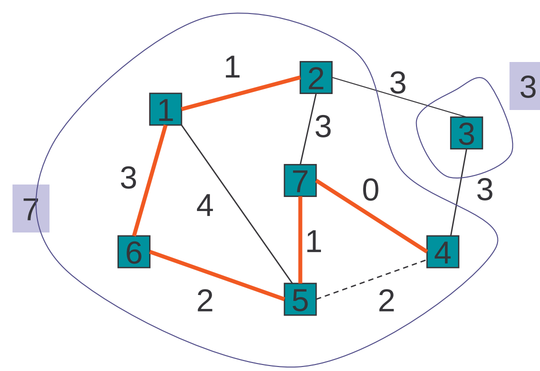
Find(1) = 1
Find(6) = 7

{7,4} {2,1} {7,5} {5,6} {5,4} {1,6} {2,7} {2,3} {3,4} {1,5}
0 1 1 2 2 3 3 3 3 4

u,v in different sets à add (u,v) to T, union the sets.

Example of DU/F (3)

Union(1,7)



{7,4} {2,1} {7,5} {5,6} {5,4} {1,6} {2,7} {2,3} {3,4} {1,5}
0 1 1 2 2 3 3 3 3 4

Kruskal's Algorithm with DU / F

```
Sort the edges by increasing cost;
Initialize T to be empty;
for each edge {i,j} chosen in increasing order do
   u := Find(i);
   v := Find(j);
   if (u ≠ v) then
      add {i,j} to T;
      Union(u,v);
```


Amortized Complexity

- Disjoint union/find can be implemented such that the average time per operation is essentially a constant.
- An individual operation can be costly, but over time the average cost per operation is not.
- On average, each U/F operation takes $O(m\,\alpha(m,n))$ time.

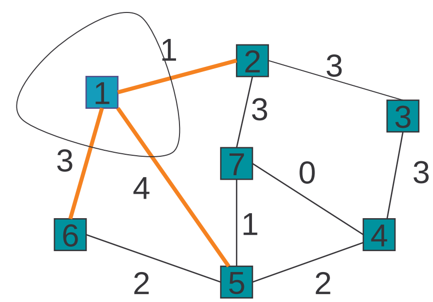
Ekerman function. Practically, this is a constant.

Evaluation of Kruskal

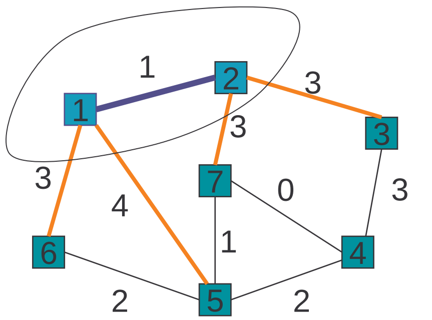
- G has n vertices and m edges.
- Sort the edges - $O(m \log m)$.
- Traverse the sorted edge list using efficient UF - $O(m\,\alpha(m,n))$.
- Total time is $O(m \log m)$.

Prim's Algorithm

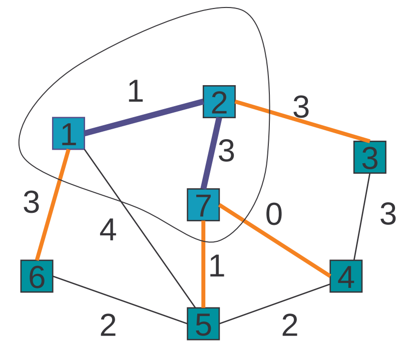
- We maintain a single tree.
- Initially, the tree consists of one vertex.
- For each vertex not in the tree maintain the cheapest edge to a vertex in the tree (if exists).



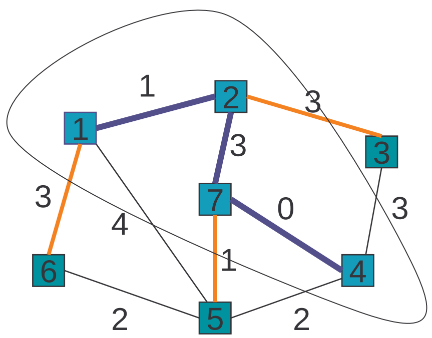
Prim's Algorithm 2



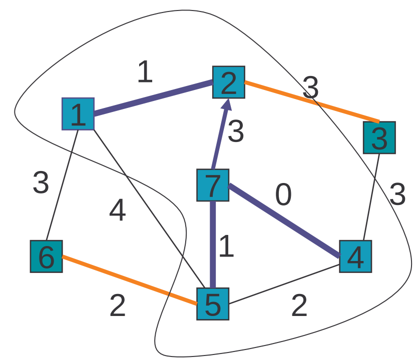
Prim's Algorithm 3



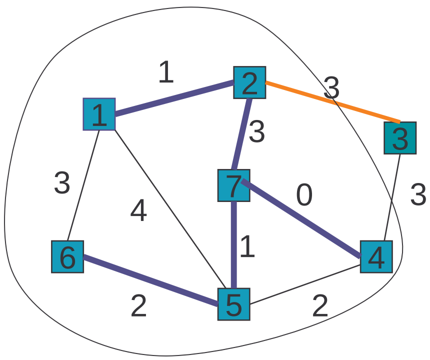
Prim's Algorithm 4



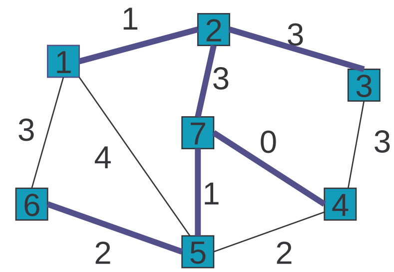
Prim's Algorithm 5



Prim's Algorithm 6



Prim's Algorithm 7

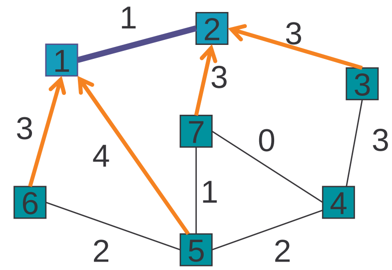


Correctness Proof for Prim

- Repeatedly executes the blue rule (n-1 times).
- In each step we consider the cut defined by the vertices that are already in T.

Data Structures for Prim

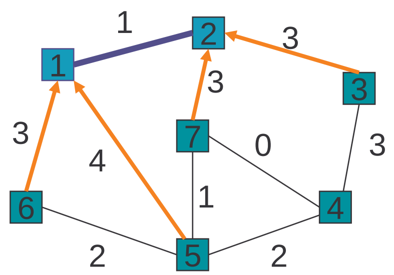
- Adjacency Lists - we need to look at all the edges from a newly added vertex.
- Array for the best edges to the tree.



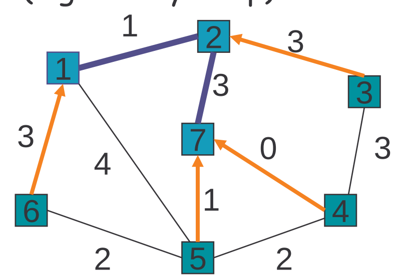
	1	2	3	4	5	6	7
to			2		1	1	2
cost			3		4	3	3

Data Structures for Prim

- Priority queue for all edges to the tree (orange edges).
  - Insert, delete-min, delete (e.g. binary heap).



	1	2	3	4	5	6	7
to			2		1	1	2
cost			3		4	3	3



	1	2	3	4	5	6	7
to			2	7	7	1	
cost			3	0	1	3	

Evaluation of Prim

- n vertices and m edges.
- Priority queue O(log n) per operation.
- O(m) priority queue operations.
  - An edge is visited when a vertex incident to it joins the tree.
- Time complexity is O(m log n).
- Storage complexity is O(m).

Kruskal vs Prim

- Kruskal
  - Simple
  - Good with sparse graphs - O(m log m)
- Prim
  - More complicated
  - Perhaps better with dense graphs - O(m log n)

Note: O(log n)= O(log m)   (since $m<n^2$)