CSEP 521 – Applied Algorithms

Graph Algorithms

Minimum Spanning Tree
Graph Coloring

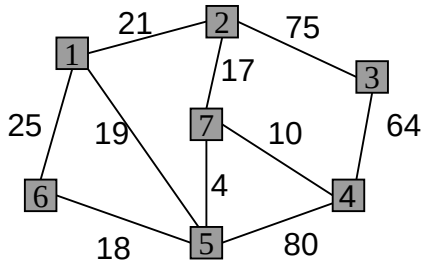
Graph Algorithms -reading

Minimum Spanning Tree:
CLR: chapter 24 (1st ed.) or 23 (2nd ed.)
Skiena: section 4.7

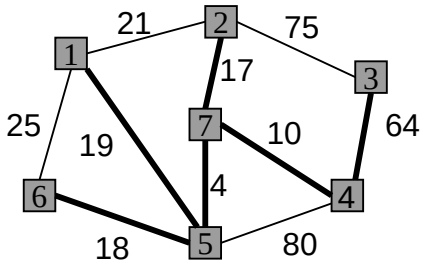
Graph Coloring:
Skiena: section 4.5.3 (very brief)

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

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

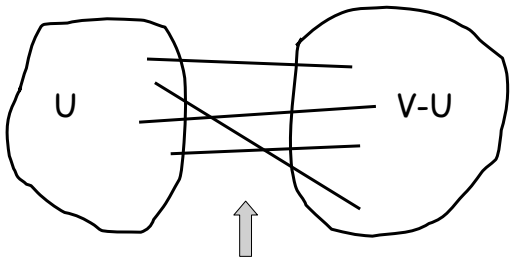
- Another formulation: Remove from G edges with maximal total cost, but keep G connected.

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

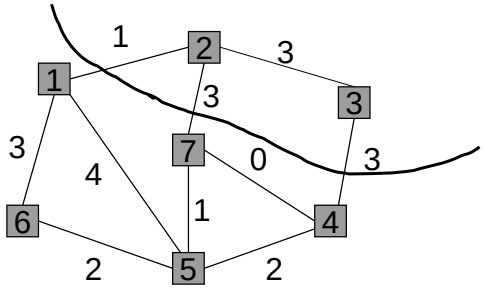
- The algorithm colors the edges of the graph. Initially, all edges are black.
- A blue edge – belongs to T.
- A red edge – does not belong to T.
- We continue to color edges until we have n-1 blue edges.
- How do we select which edge to color next? How do we select its color?

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.

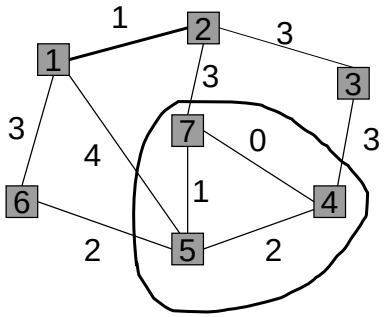
- These rules can be applied in any order.
- We will see two specific algorithms.

Example of Blue/Red rules (1)



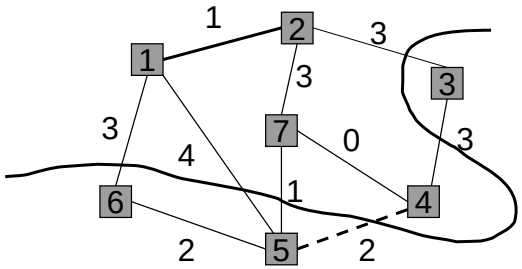
Consider the cut defined by {2,3}
- color (1,2) blue

Example of Blue/Red rules (2)



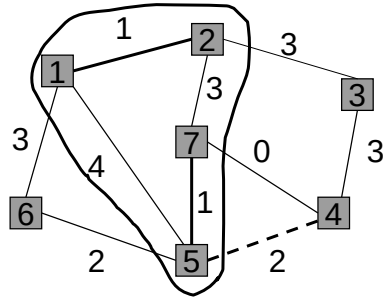
Consider the cycle (7-5-4)
- color (4,5) red

Example of Blue/Red rules (3)



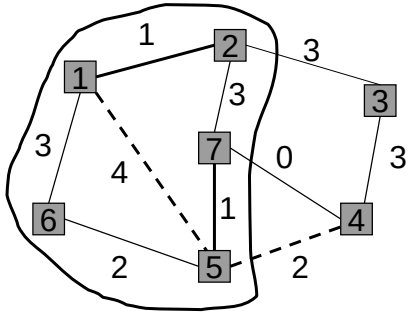
Consider the cut defined by {3,5,6}
- color (5,7) blue

Example of Blue/Red rules (4)



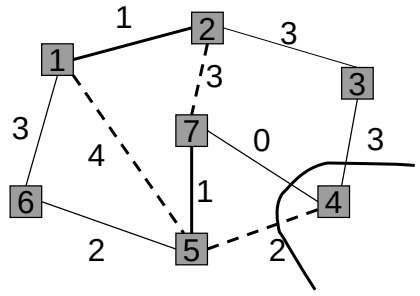
Consider the cycle (1-2-7-5)
- color (1,5) red

Example of Blue/Red rules (5)



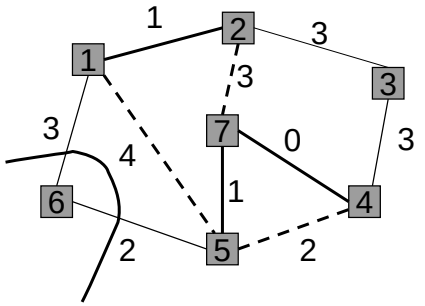
Consider the cycle (1-2-7-5-6)
- color (2,7) red.

Example of Blue/Red rules (6)



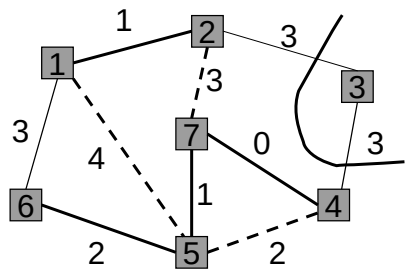
Consider the cut defined by {4}
- color (4,7) blue

Example of Blue/Red rules (7)



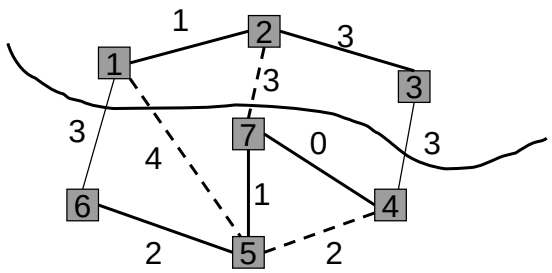
Consider the cut defined by {6}
- color (5,6) blue

Example of Blue/Red rules (8)



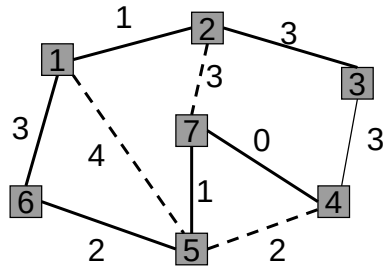
Consider the cut defined by {3}
- color (2,3) blue

Example of Blue/Red rules (9)



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

Example of Blue/Red rules (10)



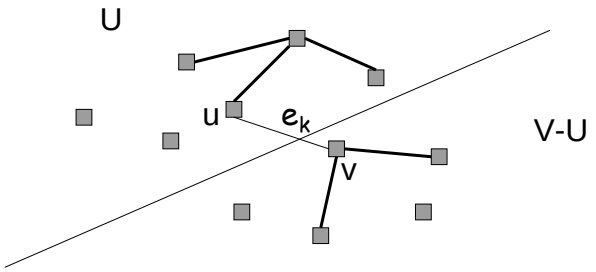
Final MST

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

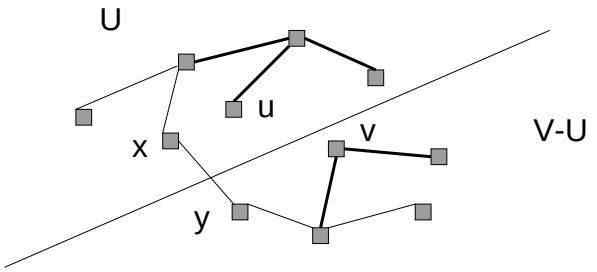
C(u,v) is minimal



Case 1: Applying the Blue Rule

T

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

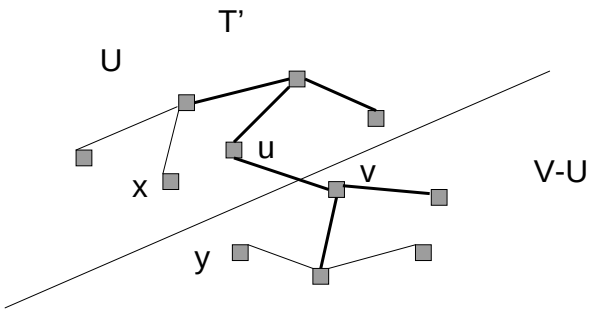


If $(u,v) \notin T$, then T must includes some other edge (x,y) in the cut defined by U (T is connected, so there is a path u—v).

Case 1: Applying the Blue Rule

Consider $T' = T \cup (u,v) - (x,y)$

T'

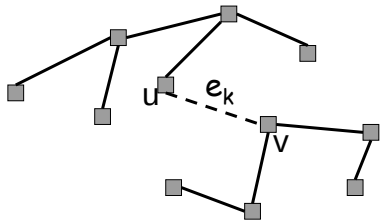


$C(T') = C(T) + C(u,v) - C(x,y)$
$C(T') \leq C(T)$

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

Case 2: Applying the Red Rule

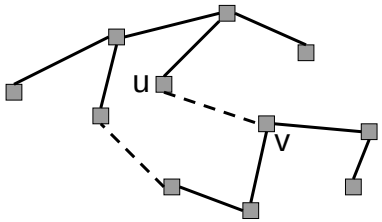
C(u,v) is maximal in some cycle



Assume $(u,v) \in T$.

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).

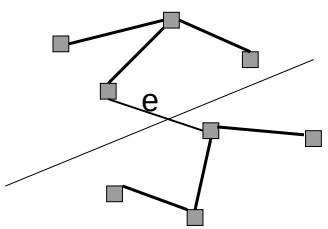
➔ There is an alternative MST, that does not include e.

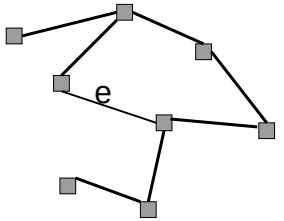
One more point: We can always proceed

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.



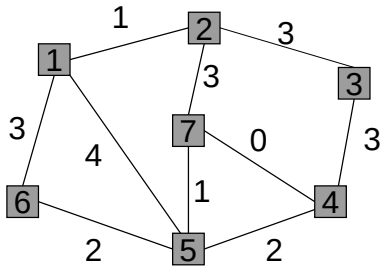


Kruskal's Greedy Algorithm

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

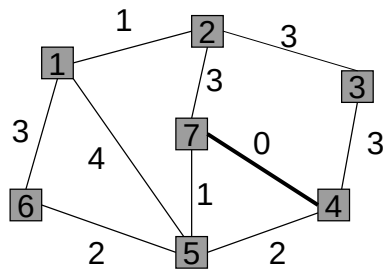
Proof: The algorithm follows the blue/red rules:
•If e closes a cycle – apply the red rule (by the sorting, e is the most expensive in this cycle).
•Otherwise – apply the blue rule (e connects two components, consider the cut defined by any of them. e is the cheapest edge in this cut)

Example of Kruskal 1



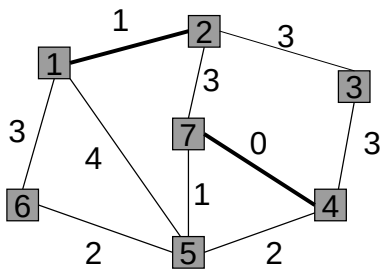
{7,4}	{2,1}	{7,5}	{5,6}	{5,4}	{1,6}	{2,7}	{2,3}	{3,4}	{1,5}
0	1	1	2	2	3	3	3	3	4

Example of Kruskal 2



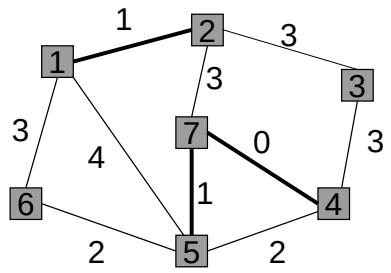
{7,4} {2,1} {7,5} {5,6} {5,4} {1,6} {2,7} {2,3} {3,4} {1,5}
0 1 1 2 2 3 3 3 3 4

Example of Kruskal 2



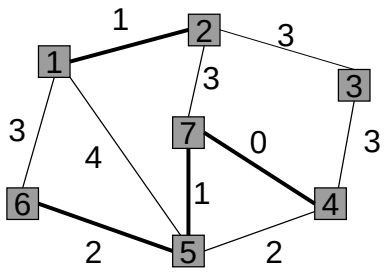
{7,4} {2,1} {7,5} {5,6} {5,4} {1,6} {2,7} {2,3} {3,4} {1,5}
0 1 1 2 2 3 3 3 3 4

Example of Kruskal 3



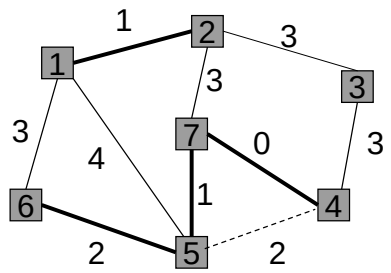
{7,4} {2,1} {7,5} {5,6} {5,4} {1,6} {2,7} {2,3} {3,4} {1,5}
0 1 1 2 2 3 3 3 3 4

Example of Kruskal 4



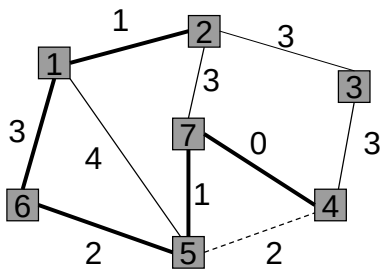
{7,4} {2,1} {7,5} {5,6} {5,4} {1,6} {2,7} {2,3} {3,4} {1,5}
0 1 1 2 2 3 3 3 3 4

Example of Kruskal 5



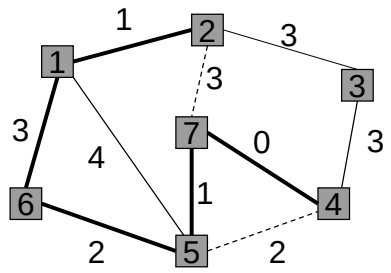
{7,4}	{2,1}	{7,5}	{5,6}	{5,4}	{1,6}	{2,7}	{2,3}	{3,4}	{1,5}
0	1	1	2	2	3	3	3	3	4

Example of Kruskal 6



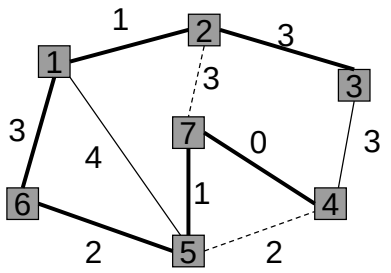
{7,4}	{2,1}	{7,5}	{5,6}	{5,4}	{1,6}	{2,7}	{2,3}	{3,4}	{1,5}
0	1	1	2	2	3	3	3	3	4

Example of Kruskal 7



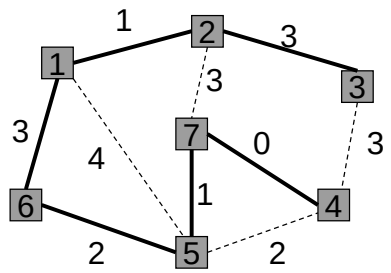
{7,4}	{2,1}	{7,5}	{5,6}	{5,4}	{1,6}	{2,7}	{2,3}	{3,4}	{1,5}
0	1	1	2	2	3	3	3	3	4

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

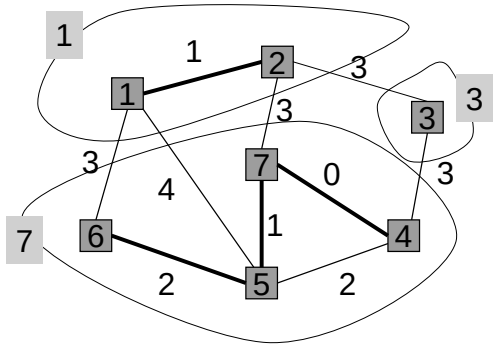
- Sorted edge list

{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)

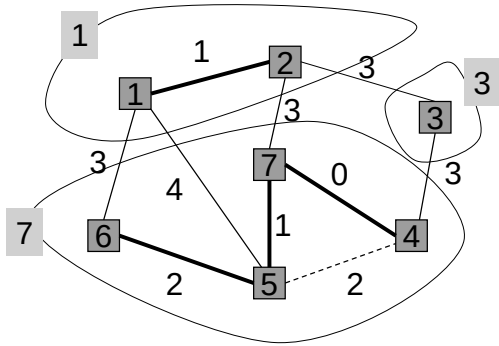


Find(5) = 7
Find(4) = 7

{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 → (u,v) is not added to T

Example of DU/F (2)

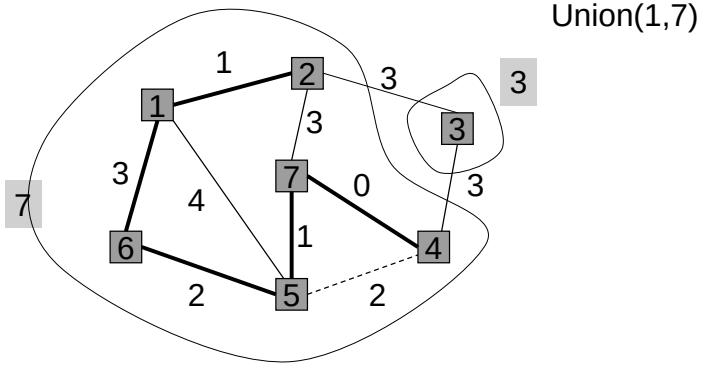


Find(1) = 1
Find(6) = 7

{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 → add (u,v) to T, union the sets.

Example of DU/F (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

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.

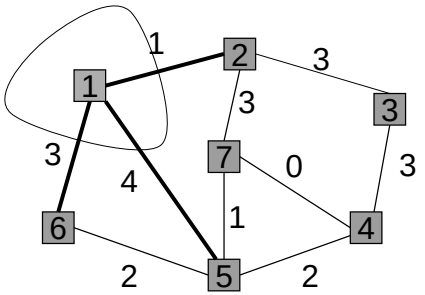
Ekerman function. Practically, this is a constant.

Evaluation of Kruskal

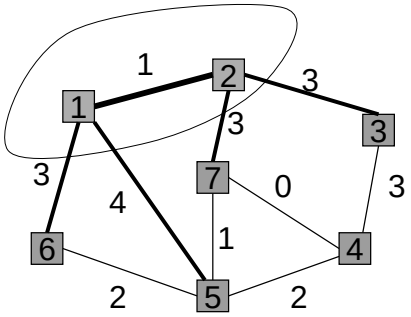
- G has n vertices and m edges.
- Sort the edges - $O(m \log m)$.
- Traverse the sorted edge list using efficient UF - $O(m\,\alpha(m,n))$.
- Total time is $O(m \log m)$.

Prim's Algorithm

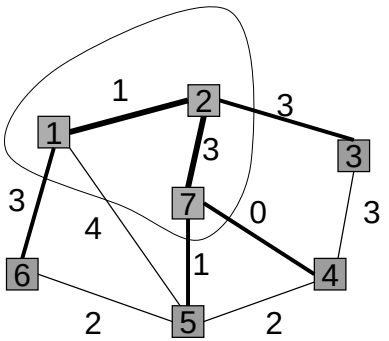
- We maintain a single tree.
- Initially, the tree consists of one vertex.
- For each vertex not in the tree maintain the cheapest edge to a vertex in the tree (if exists).



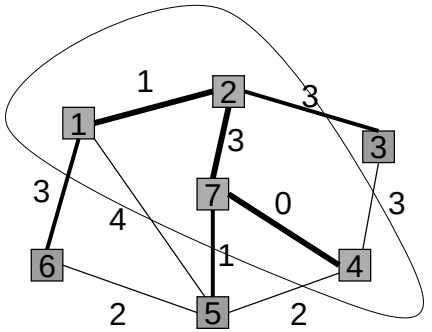
Prim's Algorithm 2



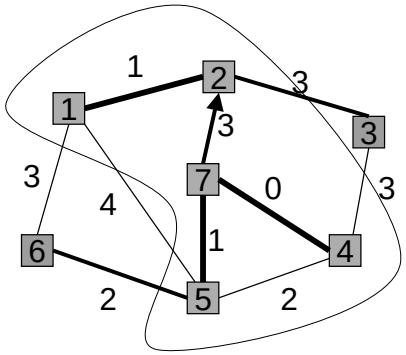
Prim's Algorithm 3



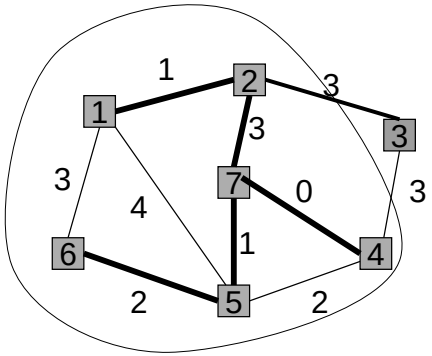
Prim's Algorithm 4



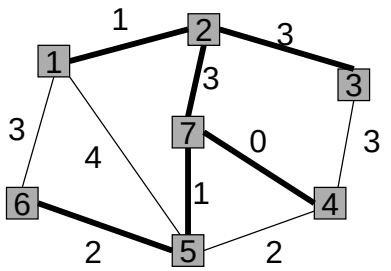
Prim's Algorithm 5



Prim's Algorithm 6



Prim's Algorithm 7

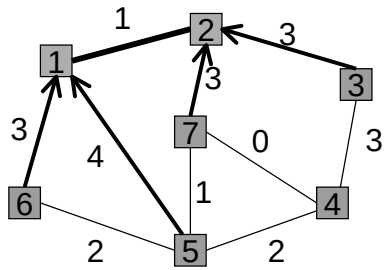


Correctness Proof for Prim

- 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

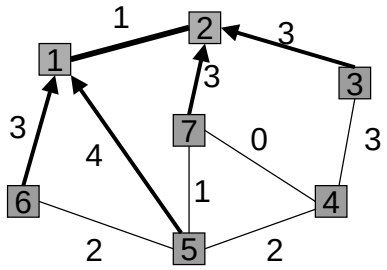
- Adjacency Lists - we need to look at all the edges from a newly added vertex.
- Array for the best edges to the tree.



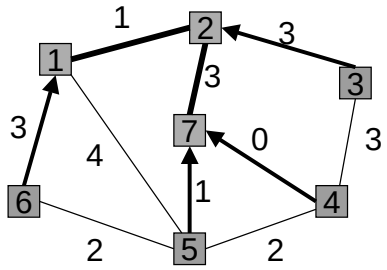
	1	2	3	4	5	6	7
to			2		1	1	2
cost			3		4	3	3

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 $m<n^2$)

Graph Coloring

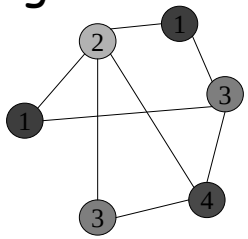
A problem that has lots of applications:

- Resource Allocation
- VLSI design
- Parallel computing

Definition: A coloring of a graph G(V,E) is a function c:V→ N such that for any edge $(u,v) \in E$, $c(v) \neq c(u)$

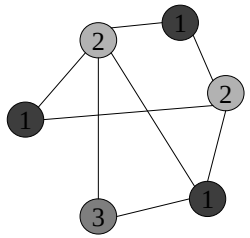
Graph Coloring

Example: coloring with 4 colors.



Problem: Given a graph G, color G using the minimal number of colors.

Example: same graph, 3 colors.



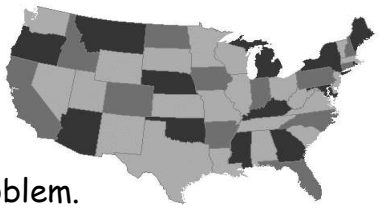
Definition: The chromatic number of a graph (denoted $\chi(G)$) is the minimal number of colors needed to color G.

The Map Coloring Problem

How many colors are needed in order to color a geographic map in such a way that neighboring countries get different colors?

The Map Coloring Problem

The map coloring problem can be reduced to finding the maximal chromatic number of a planar graph (a graph that can be drawn such that no two edges cross each other).



Vertices: the countries.

Edges: between neighboring countries.

For long, this was an open problem.

1852 – five colors are always enough, three is not enough (some maps require at least four colors).

1922 - four colors are enough for maps with at most 25 regions.

1976 - four colors are always enough.

Graph Coloring and Resource Allocation

Example: 9 groups of PMP students are learning 5 courses in a quarter.

The course CSEP501 is taken by groups 1,2,3

The course CSEP502 is taken by groups 6,7

The course CSEP503 is taken by groups 1,2,7,9

The course CSEP504 is taken by groups 4,6,8

The course CSEP505 is taken by groups 2,3,4,5

David Rispoli wants to schedule the exams such that no group will have more than one exam in one day, and the length of the exam period will be as short as possible.

Graph Coloring and Resource Allocation

Reduction to a coloring problem:

Vertices: courses.

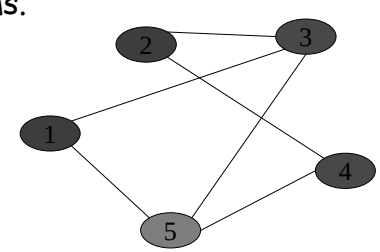
Edges: there is an edge between $CSEP_i$ and $CSEP_j$ if there is a group of students that needs to participate in both exams.

Possible solution:

Day 1: 501,502

Day 2: 505

Day 3: 503,504



Graph Coloring and Resource Allocation

Generally, given a resource allocation problem:

Vertices: processes that need resources.

Edges: between conflicting processes (that require a shared resource). This ensures that the two processes will not be scheduled simultaneously.

A coloring that uses k colors induces a partition of the processes into k phases.

Processes with the same color can be executed together – in the same phase.

2-colorable Graphs

Definition: A graph is k-colorable if it has a coloring with k colors.

Theorem: A graph is 2-colorable $\Leftrightarrow$ it does not include a cycle of odd length.

Proof:

1. ($\Rightarrow$) Let G be colored with the colors 1,2. Assume that G includes a cycle of length 2j+1. W.l.o.g v_1 is colored with 1. It must be that for any even i v_i is colored 2, and for any odd i v_i is colored 1. Therefore, the two endpoints of $(v_1\text{-}v_{2j+1})$ are colored 1. A contradiction.
2. ($\Leftarrow$) Homework…

The Chromatic Number

Let $\Delta(G)$ be the maximal degree of a vertex in G.

Theorem: For any graph G, $\chi(G) \leq \Delta(G)+1$

Proof: Lets color the graph using at mast $\Delta(G)+1$ colors:
Consider a list of the vertices in an arbitrary order. For each vertex in the list, determine c(v) to be the minimal integer which is not a color of any of the already-colored neighbors of v.

-This is a legal coloring: by definition, the color of v is different than the color of each of its neighbors.

- At most $\Delta(G)+1$ colors are used: when v is colored, at most $\Delta(G)$ neighbors of v are already colored.

The Chromatic Number

Remark: For some order of the vertices, this algorithm uses $\chi(G)$ colors.

Does it help us to find $\chi(G)$? practically, no!

We can check all the orderings, but this will take $O(n \cdot n!)$ which is a lot (next week we will define more formally why this is considered 'a lot').

Brook's Theorem: If G is not a complete graph nor a cycle of odd length, then $\chi(G) \leq \Delta(G)$.