CS2010 – Data Structures and Algorithms II

Lecture 06 – Minimum Spanning Tree

stevenhalim@gmail.com

National University of Singapore

School of Computing

Outline

- What are we going to learn in this lecture?
  - *Continuation of leftover materials from Lecture 05*
  - Minimum Spanning Tree (MST), CP2.5 Section 4.3
    - Motivating Example & Some Definitions
  - Algorithms to solve MST (you have a choice!)
    - Prim's
      - Introduction to greedy algorithm
      - Priority Queue!!
    - Kruskal's
      - Sorting graph edges based on weight
      - Data structure to prevent cycle: Union-Find Disjoint Sets
        - (This UFDS is not examinable)

Admins

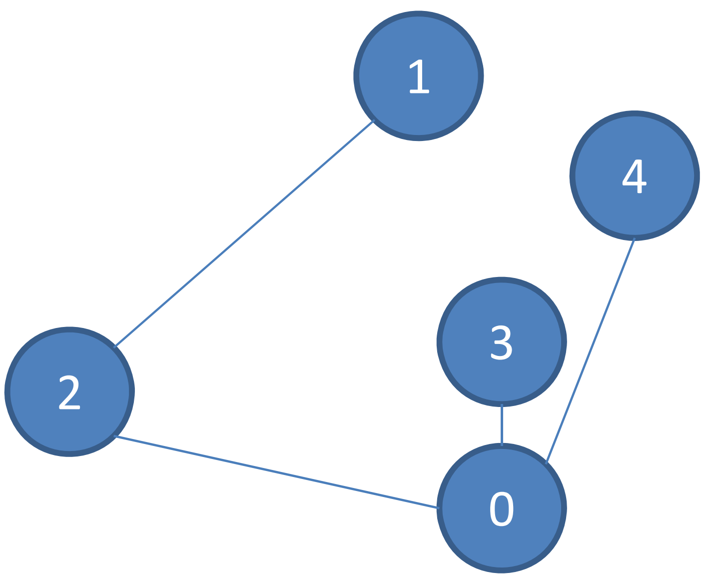
- PSBonus (Name TBA) will be opened on Saturday, 22 September 2012, after Quiz 2 until Saturday, 29 September 2011, 8am
  - Non CS2010R students do NOT have to do it…
  - This PS is optional and very? hard!
  - You can get to level 15 without touching this PS at all!
- PS4 (Out For a Walk) will be opened on Thursday, 27 September 2012 but PS4 will only due on Week08, Tuesday, 09 October 2012, 8am
  - You have time to deal with your other modules' midtests during Week07 ☺

Review

- Definitions that we have learned before
  - **Tree T**
    - **T** is a **connected graph** that has **V** vertices and **V-1** edges
    - One unique path between any two pair of vertices in **T**
  - **Spanning** Tree **ST** of connected graph **G**
    - **ST** is a tree that spans (covers) every vertices in **G**
    - Recall the **BFS and DFS Spanning Tree**
- Sorting problem & several sorting algorithms
  - Rearrange set of objects so that every pair of objects **(a, b; a < b)** in the final arrangement satisfies that **a** is before **b**

Is This A Tree?

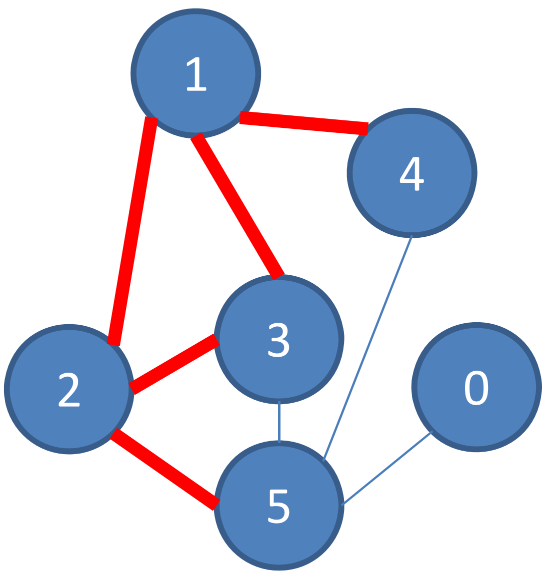
1. Yes, why ___________
2. No, why ___________



0 of 120

Are the edges highlighted in red part of a spanning tree of the original graph?

1. Yes, why ___________
2. No, why ___________



0 of 120

Motivating Example

- Government Project
  - Want to link rural villages with roads
  - The cost to build a road depends on the terrain, etc
  - You only have limited budget
  - How are you going to build the roads?

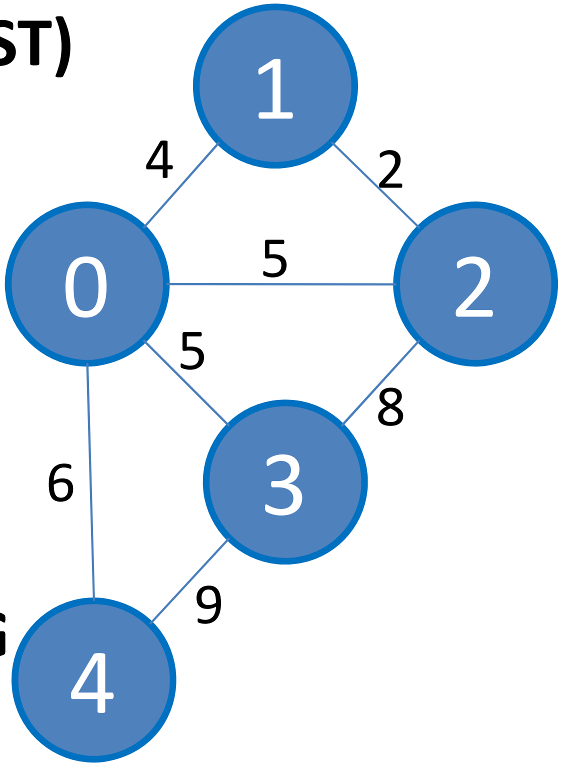
More Definitions (1)

- **Weighted** Graph: **G(V, E), w(u, v): E→R**
  - See below for **w(u, v)**
- Vertex **V** (e.g. street intersections, houses, etc)
- Edge **E** (e.g. streets, roads, avenues, etc)
  - Generally undirected (e.g. bidirectional road, etc)
  - Weighted (e.g. distance, time, toll, etc)
- Weight function **w(u, v): E→R**
  - Sets the weight of edge from **u** to **v**
- **Connected** undirected graph **G**
  - There is a path from any vertex **u** to any other vertex **v** in **G**

More Definitions (2)

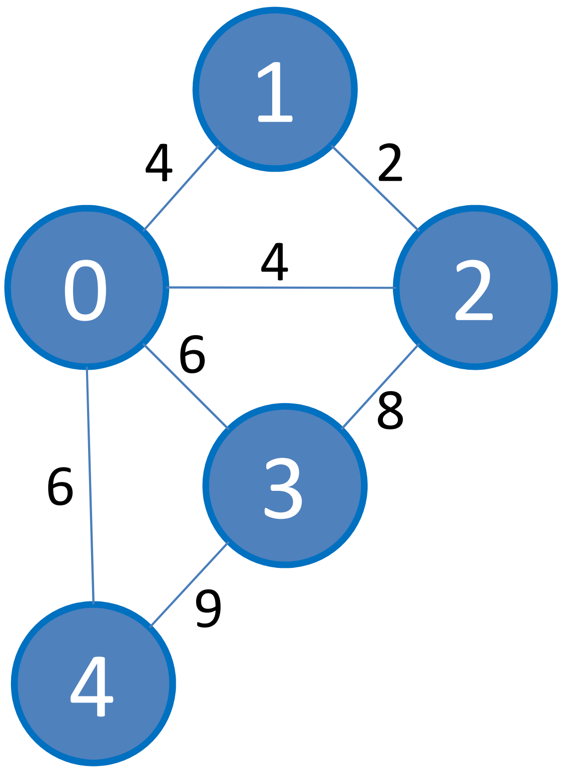
$$w(ST) = \sum_{(u,v)\in ST} w(u,v)$$

- Spanning Tree **ST** of **G**
  - Let **w(ST)** denotes the total weight of edges in **ST**
- **Minimum Spanning Tree (MST)** of connected undirected weighted graph **G**
  - **MST** of **G** is an **ST** of **G** with min possible **w(ST)**
- **The (standard) MST Problem**
  - Input: A connected undirected weighted graph **G(V, E)**
  - Select some edges of **G** such that the graph is still connected, but with min total weight
  - Output: Minimum Spanning Tree **(MST)** of **G**

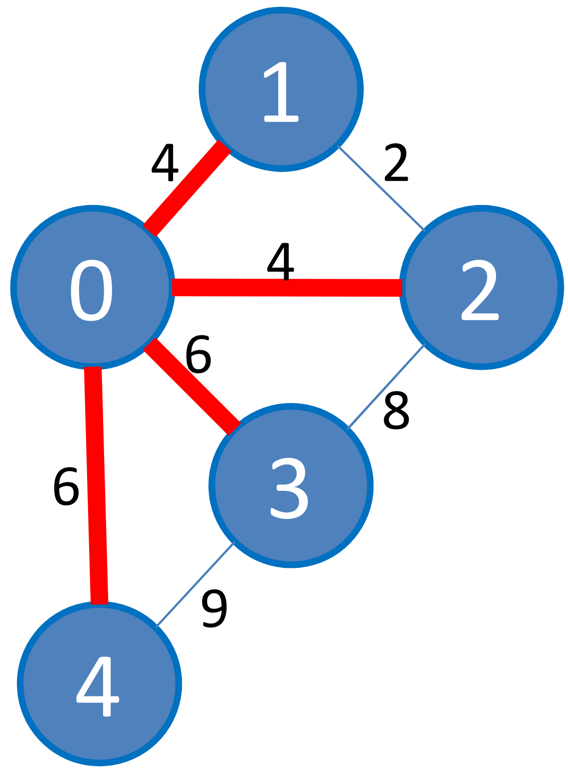


Example

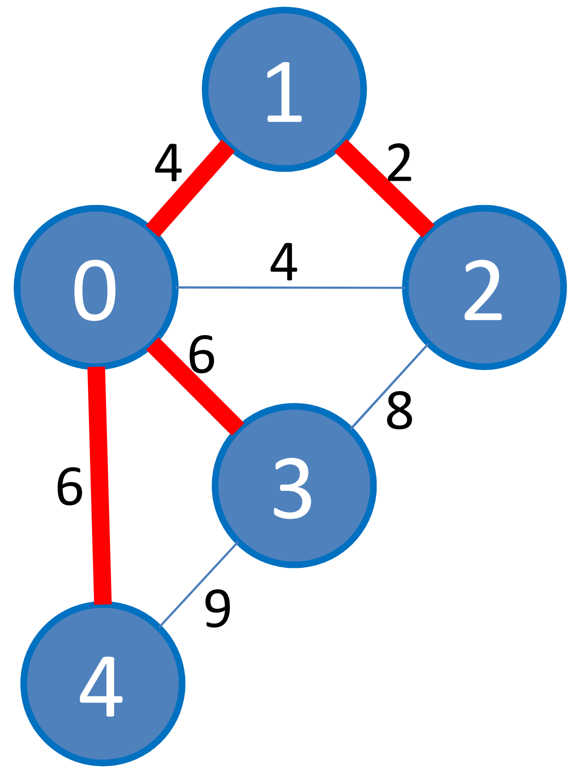
The Original Graph



A Spanning Tree
Cost: 4+4+6+6 = 20

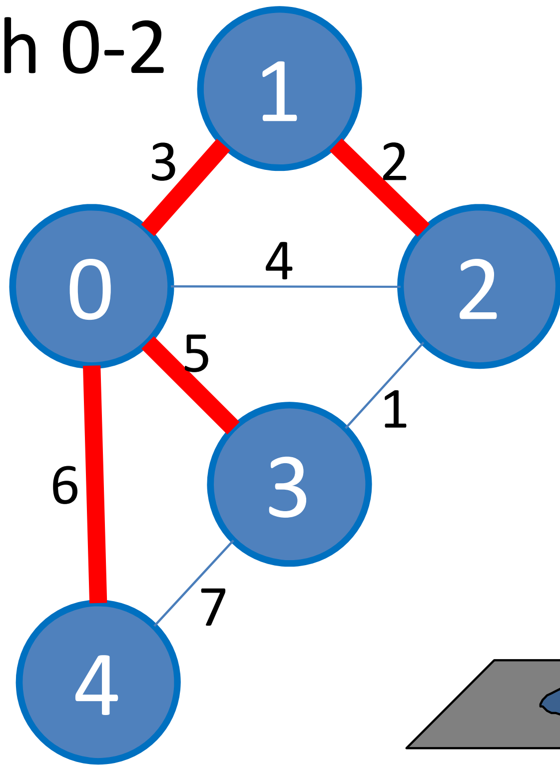


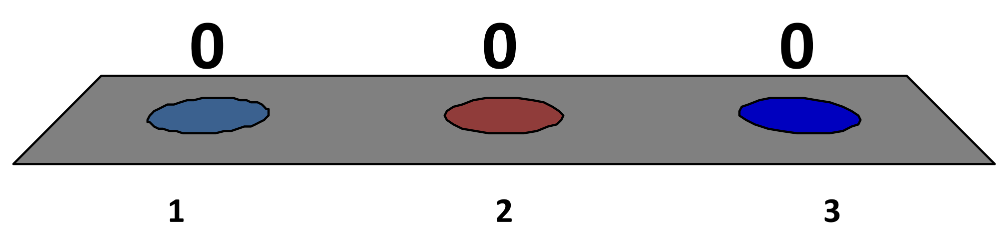
An MST
Cost: 4+6+6+2 = 18



Are the edges highlighted in red part of an MST of the original graph?

1. No, we must replace edge 0-3 with edge 2-3
2. No, we must replace edge 1-2 with 0-2
3. Yes





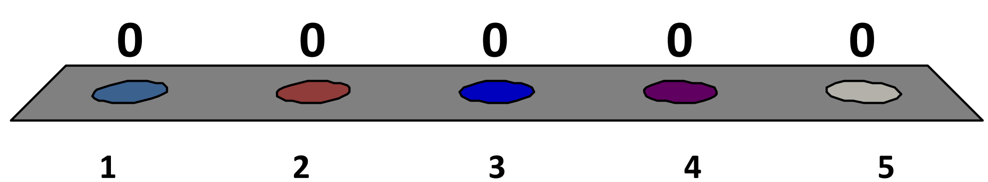
0 of 120

MST Algorithms

- MST is a well-known Computer Science problem
- Several efficient (polynomial) algorithms:
  - Jarnik’s/Prim’s greedy algorithm
    - We will use PriorityQueue Data Structure taught in Lecture04!
  - Kruskal’s greedy algorithm
    - We will learn two new Data Structures :O
  - Boruvka’s greedy algorithm (not discussed here)
  - And a few other more advanced variants/special cases…

Do you still remember Prim's/Kruskal's algorithms from CS1231?

1. Yes and I also know how to ***implement*** them
2. Yes, but I have not try implementing them yet
3. I forgot that particular CS1231 material… but I know it exists
4. Eh?? These two algorithms were covered before in CS1231??
5. I haven't took CS1231 ☹



0 of 120

Prim's Algorithm

- Pseudo code (very simple)

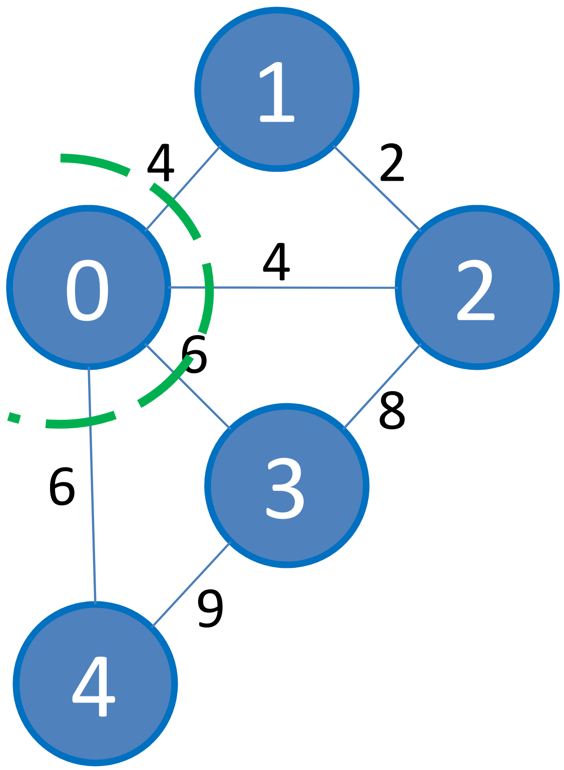
```
T ← {s}, a starting vertex s (usually vertex 0)
enqueue edges connected to s (only the other ending
   vertex and edge weight) into a priority queue PQ
   that orders element based on increasing weight
while there are unprocessed edges left in PQ
  take out the front most edge e
  if vertex v linked with this edge e is not taken yet
    T ← T ∪ v
    enqueue edges connected to v (as above)
T is an MST
```

- Let's see how it works first…

Prim’s Animation (1)

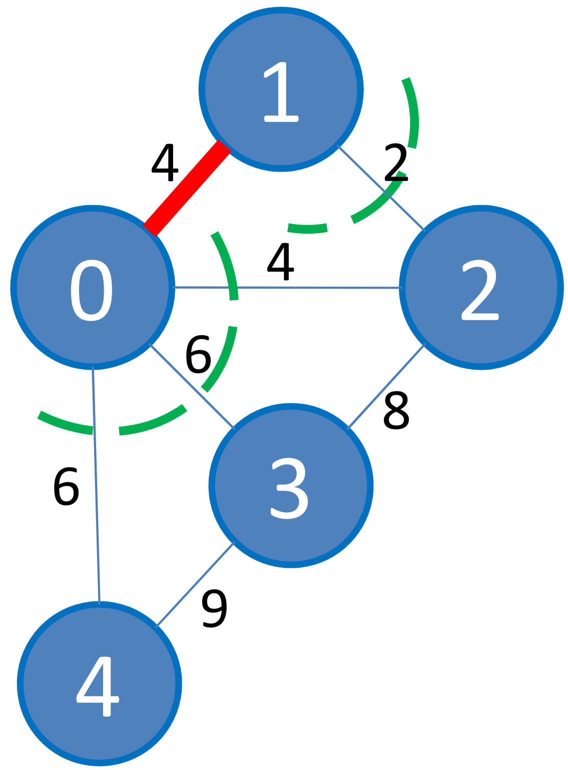
PQ = {(4,1),(4,2),(6,3),(6,4)}

The original graph,
start from vertex 0



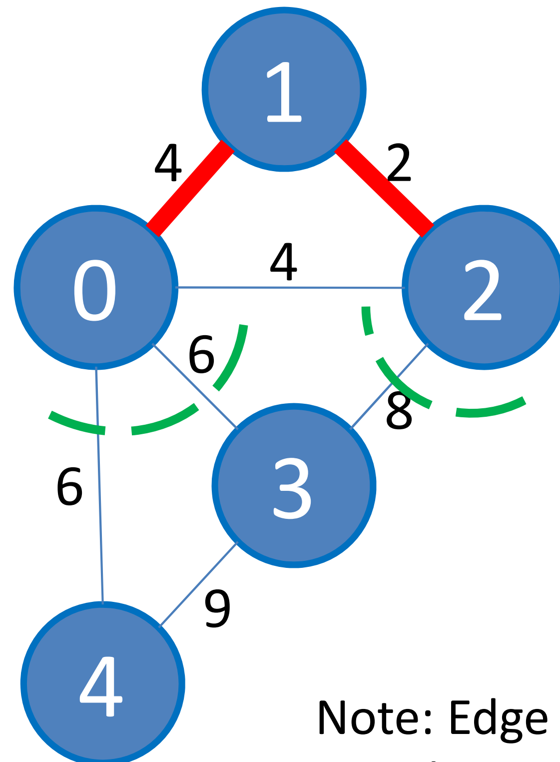
PQ = {(2,2),(4,2),(6,3),(6,4)}

Connect 0 and 1
As this edge is smallest



PQ = {~~(4,2)~~,(6,3),(6,4),(8,3)}

Connect 1 and 2
As this edge is smallest



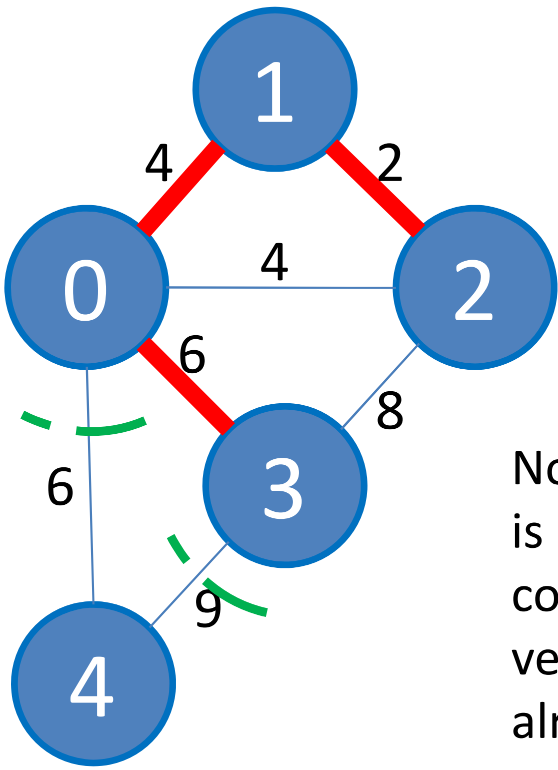
Note: Edge 0-2 is no longer considered as vertex 2 is already taken

Note: The sorted order of the edges determines how the MST formed. Observe that we can also choose to connect vertex 0 and 2 also with weight 4!

Prim’s Animation (2)

PQ = {(6,4),(8,3),(9,4)}

Connect 0 and 3
As this edge is smallest

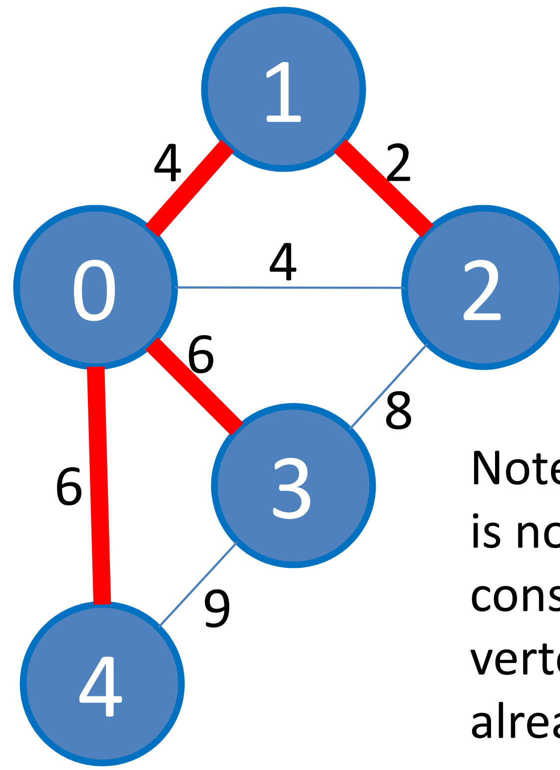


Note: Edge 2-3 is no longer considered as vertex 3 is already taken

Note: The sorted order of the edges determines how the MST formed. Observe that we can also choose to connect vertex 0 and 4 also with weight 6!

PQ = {~~(8,3),(9,4)~~}

Connect 0 and 4
MST is formed



Note: Edge 3-4 is no longer considered as vertex 4 is already taken

Visualization

- Let’s take a look at MST visualization:

  www.comp.nus.edu.sg/~stevenha/visualization/mst.html

Java Implementation

- You just need to use two known Data Structures to be able to implement Prim's algorithm:
  - A priority queue (we can use Java PriorityQueue), and
  - A Boolean array (to decide if a vertex has been taken or not)
- With these DSes, we can run Prim's in O(E log V)
  - We process each edge once, O(E)
  - Each time, we Insert/ExtractMax from a Binary Heap/ Priority Queue in O(log E) = O(2 log V) = O(log V)
- Let's have a quick look at PrimDemo.java

Why Prim's Works? (1)

- First, we have to realize that **Prim's algorithm** is a **greedy algorithm**
  - This is because **at each step**, it always try to select the next valid edge e with **minimal weight** (greedy!)
  - Greedy algorithm is usually simple to implement
  - However, it usually requires "proof of correctness"
  - You will see such proof like this again in CS3230
  - Here, we will just see a quick proof

Why Prim's Works? (2)

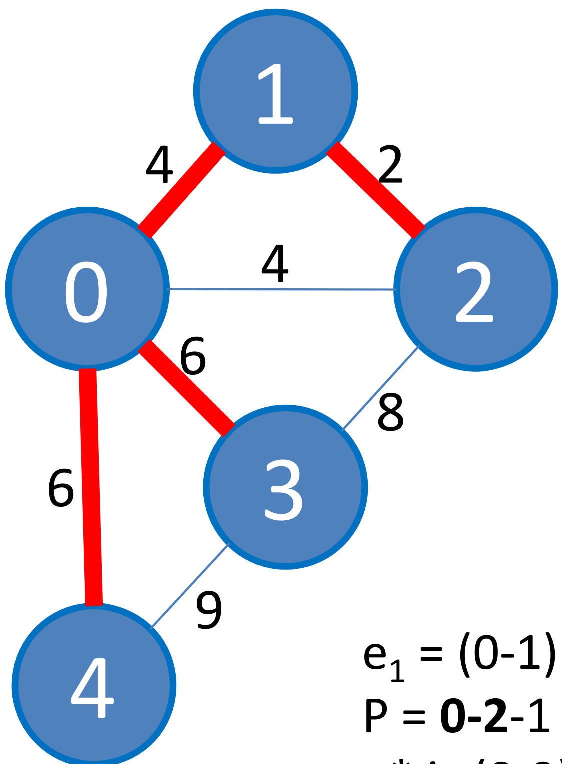
- Let T be the spanning tree of graph G generated by Prim's algorithm and T* be the spanning tree of G that is known to have minimal cost
- If T == T*, we are done
- If T != T*
  - Let $e_k = (u, v)$ be the first edge chosen by Prim's algorithm at the *k-th* iteration that **<u>is not</u>** in T*
  - Let *P* be the path from *u* to *v* in *T**, and let *e** be an edge in *P* such that one endpoint is in the tree generated at the (*k*−*1*)-th iteration of Prim's algorithm and the other is not
    - i.e. one endpoint of *e** is *u* **or** one endpoint is *v*,
      but the endpoints are not *u* **and** *v*

Why Prim's Works? (3)

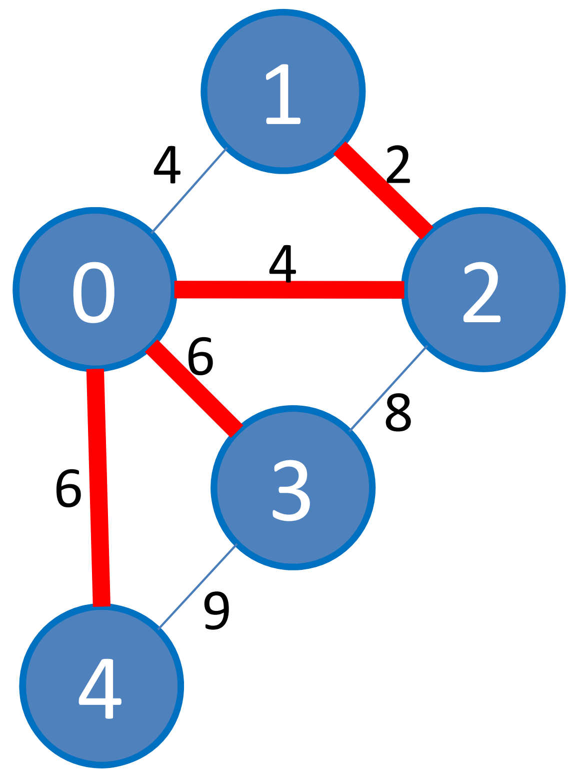
- If T != T* (continued)
  - If the weight of e^* is less than the weight of e_k, then Prim's algorithm <u>would have chosen</u> e^* on its k-th iteration
    - So, it is certain that $w(e^*) \geq w(e_k)$
    - When e^* has weight equal to that of e_k, the choice between the e^* or e_k is arbitrary
    - Whether the weight of e^* is greater than or equal to e_k, e^* can be substituted with e_k while preserving minimal total weight of T^*
  - This process can be repeated until T^* is equal to T
    - Thus we can show that the spanning tree generated by any instance of Prim's algorithm is a minimal spanning tree

Visual Explanation

Our Prim's algorithm reports this MST T



Suppose that this is the optimal MST T*



e_1 = (0-1) at iteration 1
P = **0-2**-1 in T*
e* is (0-2)
If we substitute e_1 with e*, we can transform T to T*

After this…, two more “new” data structures :O

5 MINUTES BREAK

Kruskal's Algorithm (1)

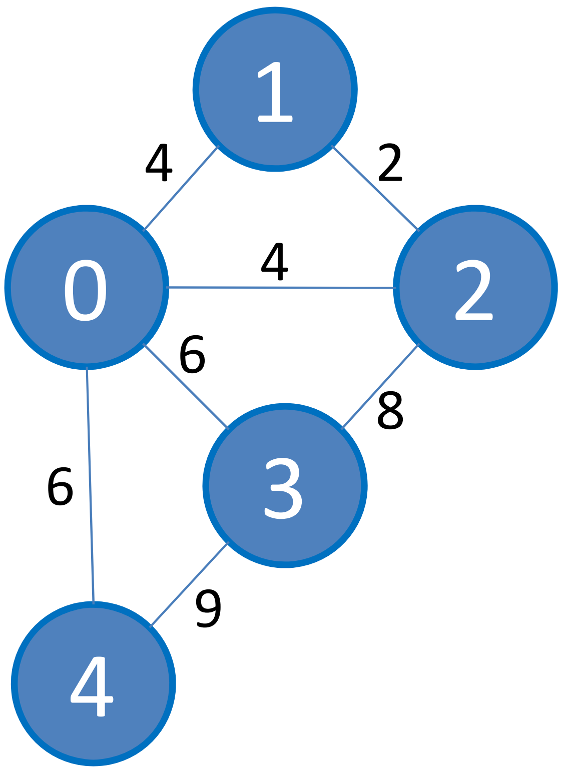
- Pseudo code (very simple)

```
sort E edges by increasing weight
T ← {}
while there are unprocessed edges left
  pick an unprocessed edge e with min cost
  if adding e to T does not form a cycle
    add e to T
T is an MST
```

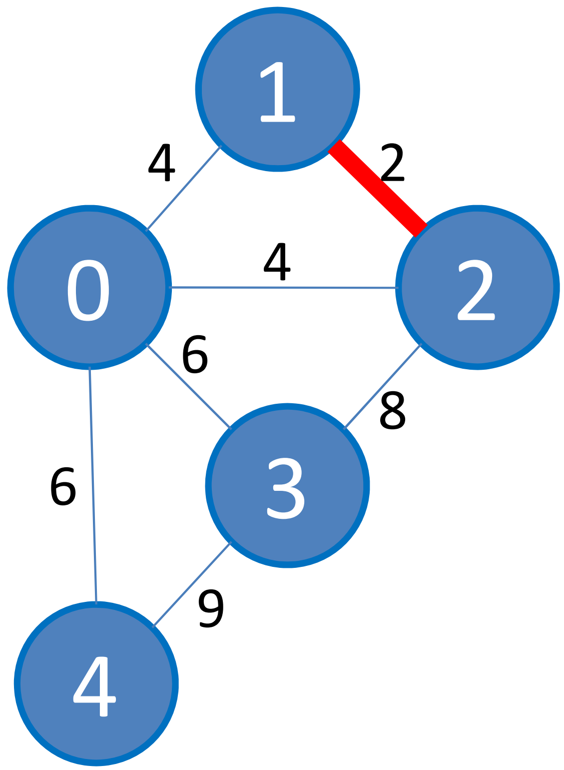
- Let's see how it works first…

Kruskal’s Animation (1)

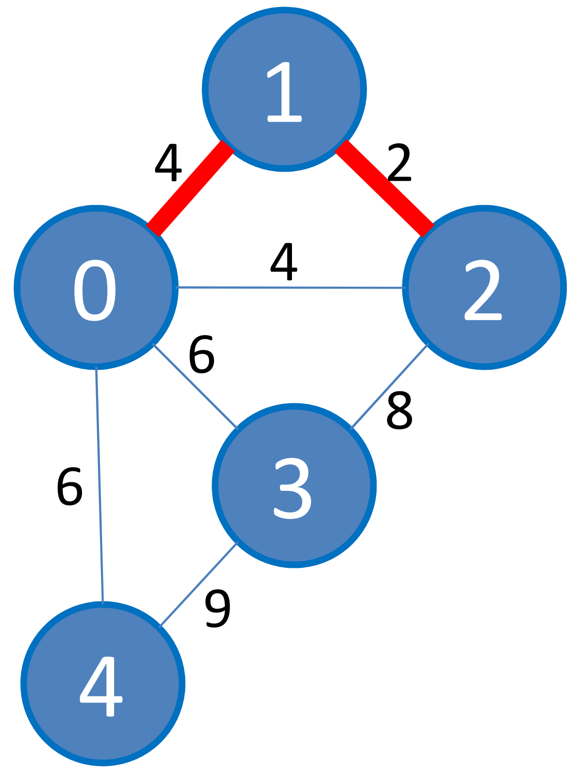
The original graph,
no edge is selected



Connect 1 and 2
As this edge is smallest



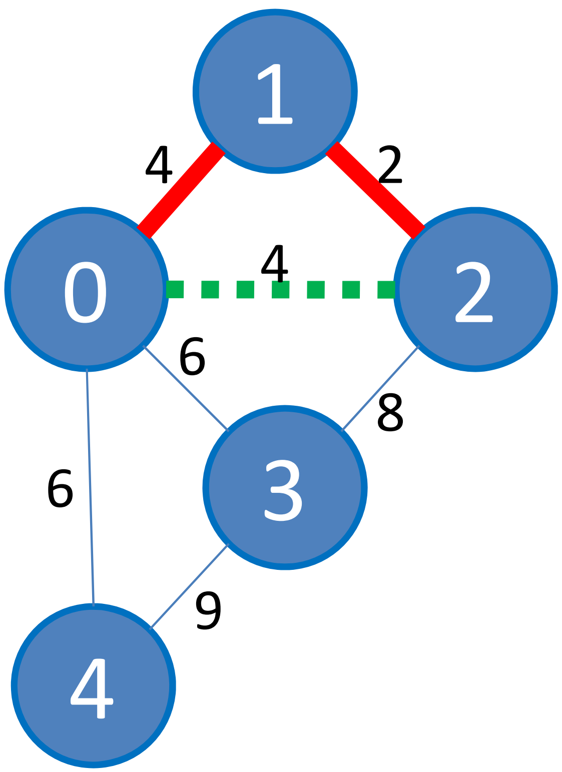
Connect 1 and 0
No cycle is formed



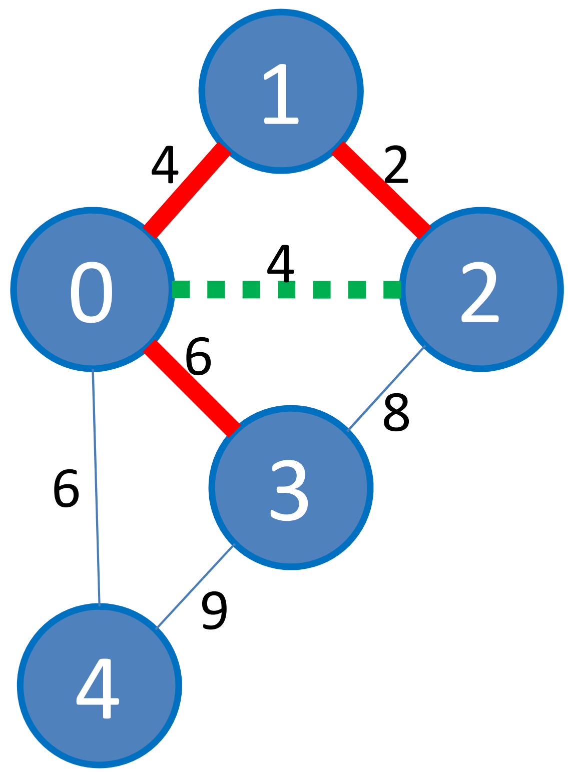
Note: The sorted order of the edges determines how the MST formed. Observe that we can also choose to connect vertex 2 and 0 also with weight 4!

Kruskal’s Animation (2)

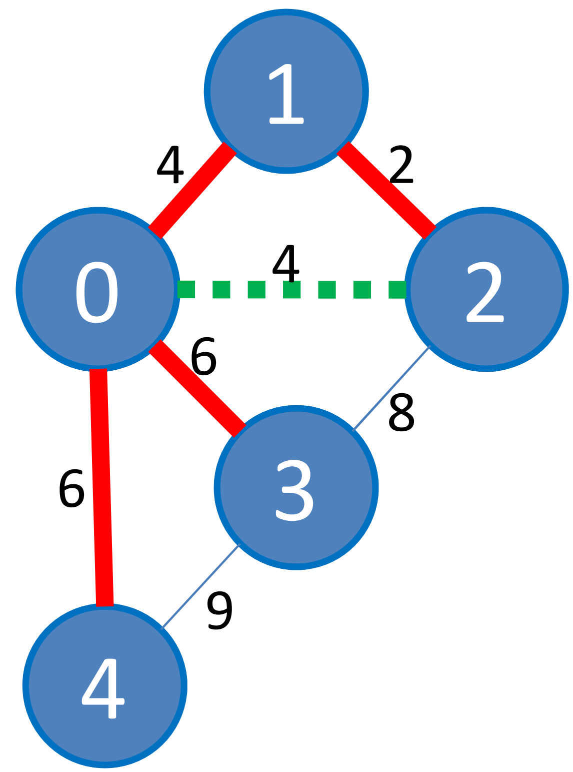
Cannot connect 0 and 2
As it will form a cycle



Connect 0 and 3
The next smallest edge



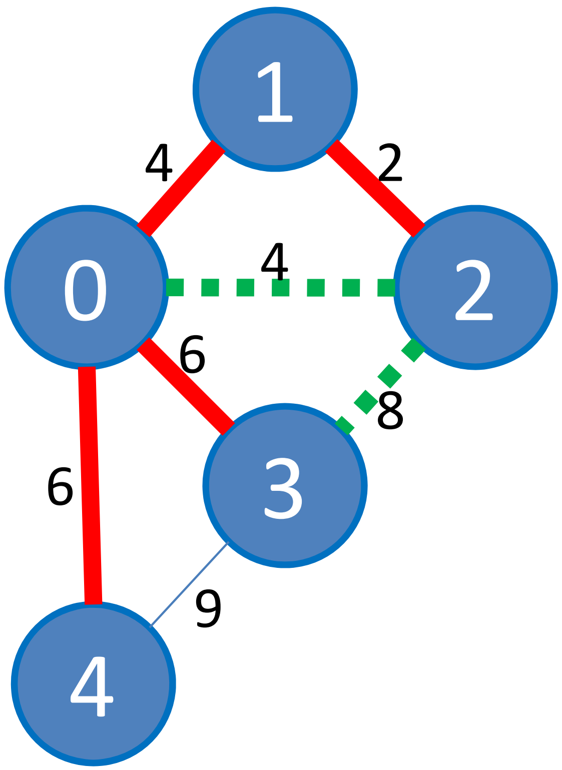
Connect 0 and 4
MST is formed…



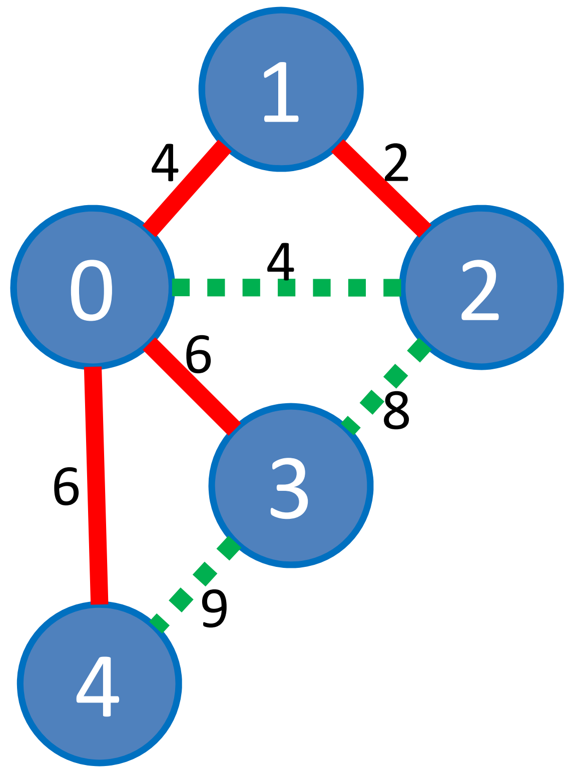
Note: Again, the sorted order of the edges determines how the MST formed;
Connecting 0 and 4 is also a valid next move

Kruskal’s Animation (3)

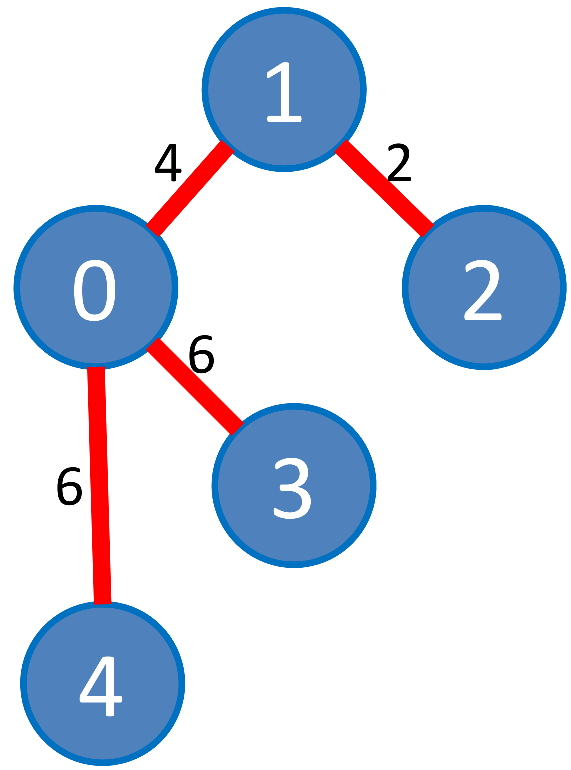
But (standard) Kruskal’s algorithm will still continue



However, it will not modify anything else



This is the final MST with cost 18



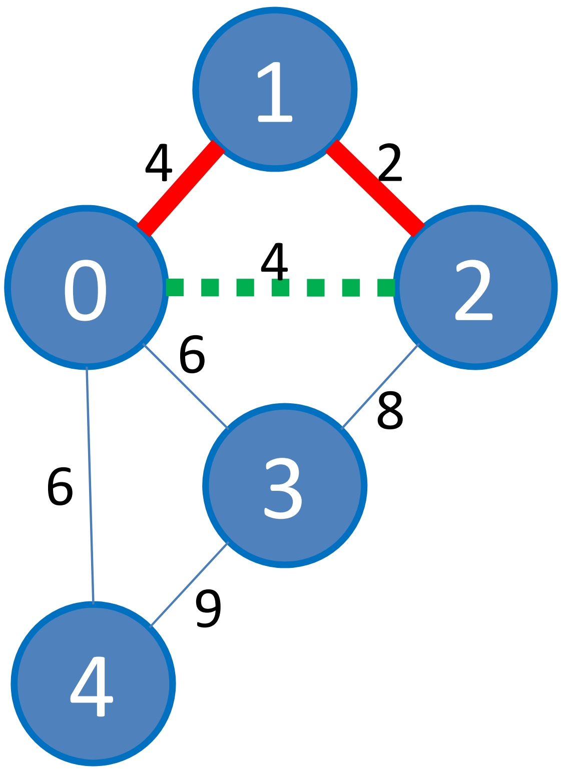
www.comp.nus.edu.sg/~stevenha/visualization/mst.html

Why Kruskal's Works? (1)

- **Kruskal's algorithm** is also a **greedy algorithm**
  - This is because **at each step**, it always try to select the next unprocessed edge e with **minimal weight** (greedy!)
- Simple proof on how this greedy strategy works
  - Loop invariant: Every edge **e** that is added into **T** by Kruskal's algorithm is part of the MST

Why Kruskal's Works? (2)

Cannot connect 0 and 2
As it will form a cycle



Loop invariant: Every edge **e** that is added into **T** by Kruskal's algorithm is part of the MST.

```
sort E edges by increasing weight
T ← {}
while there are unprocessed edges left
  pick an unprocessed edge e with min cost
  if adding e to T does not form a cycle
    add e to T
T is an MST
```

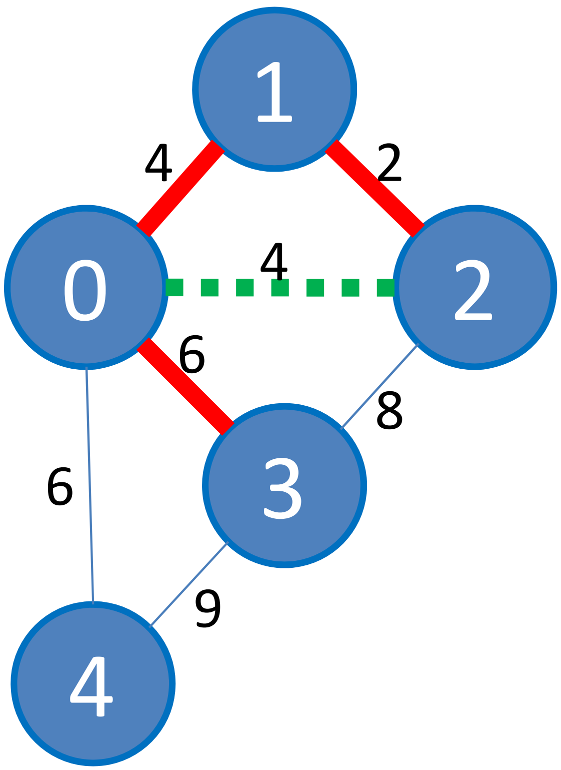
Kruskal's algorithm has a special **cycle check** before adding an edge **e** into **T**. Edge **e** will never form a cycle.

At the start of every loop, **T** is always part of **MST**.

At the end of the loop, we have selected **V-1** edges from a connected weighted graph **G** without having any cycle. This implies that we have a **Spanning Tree**.

Why Kruskal's Works? (3)

Connect 0 and 3
The next smallest edge



Loop invariant: Every edge **e** that is added into **T** by Kruskal's algorithm is part of the MST.

```
sort E edges by increasing weight
T ← {}
while there are unprocessed edges left
  pick an unprocessed edge e with min cost
  if adding e to T does not form a cycle
    add e to T
T is an MST
```

By keep adding the next unprocessed edge **e** with min cost, **w(T U e) ≤ w(T U any other unprocessed edge that does not form cycle)**.

At the start of every loop, **T** is always part of **MST**.

At the end of the loop, the Spanning Tree **T** must have minimal weight **w(T)**, so **T** is the final **MST**.

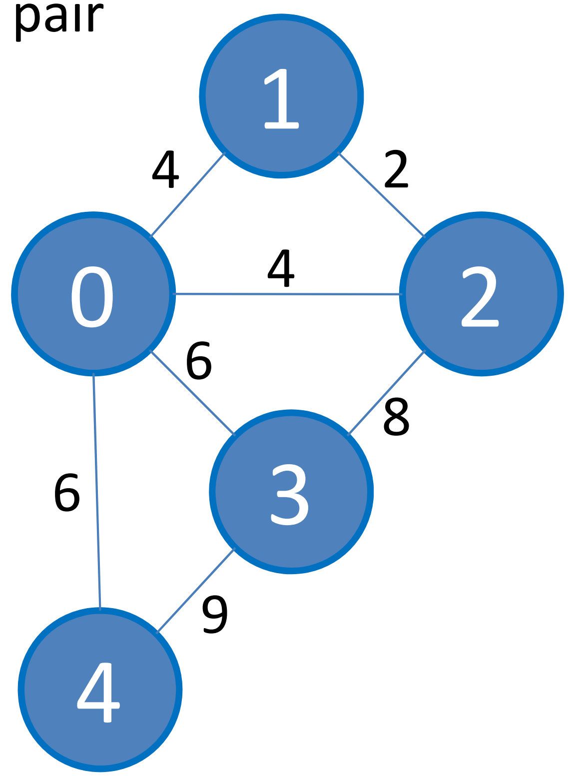
Kruskal's Algorithm (2)

```
sort E edges by increasing weight // O(E log E)
T ← {}
while there are unprocessed edges left // O(E)
  pick an unprocessed edge e with min cost // O(1)
  if adding e to T does not form a cycle // O(X)
    add e to the T // O(1)
T is an MST
```

- To sort the edges:
  - We use a **new** way to store graph information: **EdgeList**
  - Then use "any" sorting algorithm that we have seen before
- To test for cycles:
  - We will use a **new** data structure: **Union-Find Disjoint Sets**

Edge List

- Format: array **EdgeList** of E edges
- For each edge i, **EdgeList[i]** stores an (integer) triple {weight (u, v), u, v}
  - For unweighted graph, the weight can be stored as 0 (or 1), or simply store an (integer) pair
- Space Complexity: O(E)
  - Remember, $E = O(V^2)$
- Adjacency Matrix/List that we have learned earlier are *not suitable* for edge-sorting task!



i	w	u	v
0	2	1	2
1	4	0	1
2	4	0	2
3	6	0	3
4	6	0	4
5	8	2	3
6	9	3	4

Java Implementation (1)

- Introducing class **IntegerTriple** (similar as **IntegerPair**)
  - Used to store 3 attributes: weight(u, v), u, v
  - Class **IntegerTriple** implements **Comparable**
    - This allows a collection of this class to be *sorted*
  - Class **IntegerTriple** has **toString()** method to show its content
- To implement **EdgeList**, we can just use **Vector < IntegerTriple >**
- We can sort **EdgeList** by using ***one liner*** **Java Collections.sort :O**
  - After all efforts for teaching you merge/quick sort… :O

Java Implementation (2)

- Your Lab TA will present a bit more details about the non-examinable Union-Find Data Structure (UFDS) during Lab Demo of Week07
- For now, let's assume that the cost to test for cycle using UFDS is "very small", can be assumed as O(1)

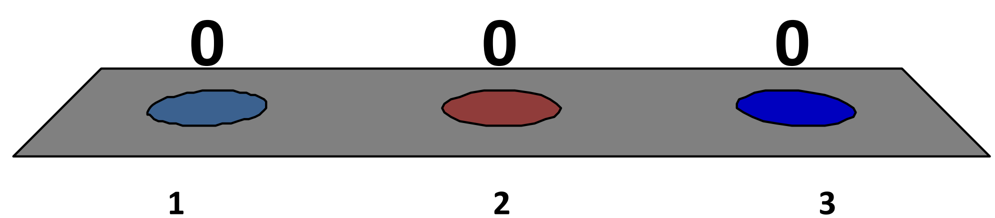
Kruskal’s Algorithm (3)

```
sort E edges by increasing weight // O(E log E)
T ← {}
while there are unprocessed edges left // O(E)
  pick an unprocessed edge e with min cost // O(1)
  if adding e to T does not form a cycle // O(α(V) = O(1)
    add e to the T // O(1)
T is an MST
```

- To sort the edges, we need O(E log E)
- To test for cycles, we need $O(\alpha(V))$ – small, assume constant O(1)
- In overall
  - Kruskal’s runs in $O(E \log E + \cancel{E\,\alpha(V)})$ // E log E dominates!
  - As $E = O(V^2)$, thus Kruskal’s runs in $O(E \log V^2)$ = **O(E log V)**
- Let’s have a quick look at KruskalDemo.java

If given an MST problem, I will…

1. Use/code Kruskal's algorithm
2. Use/code Prim's algorithm
3. No preference…



0 of 120

Summary

- Re-introducing the MST problem (covered in CS1231)
- Discussing the implementation of Prim's algorithm
  - Revisiting PriorityQueue ADT
- Discussing the implementation of Kruskal's algorithm
  - Introducing the EdgeList and technique to sort edges
  - A preview of Union-Find Disjoint Sets DS

– You *may* learn MST/Prim's/Kruskal's again in CS3230

Your To Do List around Recess Week

- Quiz 1 (15%) this Saturday
- PSBonus is released this Saturday
- PS3 is due next Tuesday
- *Steven is away during recess week (IOI 2012)*
- PS4 is released next Thursday
- PSBonus is due next Saturday
- Nothing due on Week07 for CS2010 ☺