# CE2101/ CZ2101: Algorithm Design and Analysis

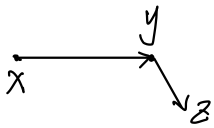
## Greedy Algorithms;

## Dijkstra’s Algorithm; Prim’s Algorithm

Ke Yiping, Kelly

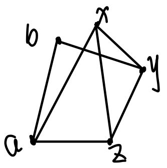
# Lemmas & Proofs

proof by contradiction.

Lemma 1: 

$\begin{cases} \text{weighted graph } G \\ x \to z \text{ is shortest path} \end{cases} \Rightarrow \begin{cases} x \to y \text{ and} \\ y \to z \text{ is shortest path} \end{cases}$

Theorem 1: Rephrasing Dijakstra's Algo (Greedy choice is optimal)



Let G = (V, E, W) be a weighted graph with nonnegative weights. Let S be a subset of V and let s be a member of S. Assume that d[y] is the shortest distance in G from s to y, for each y in S. Let z be the next vertex chosen to go into S. If edge (y, z) is chosen to minimise d[y] + W(y, z) over all edges with one vertex in S and one vertex in V – S, then the path consisting of a shortest path from s to y followed by the edge (y, z) is the shortest path from s to z.

⇓ 数学语言表达:

① Let P be a shortest path from s to y [distance: d(y)]
Let W(P) = distance traveled along P

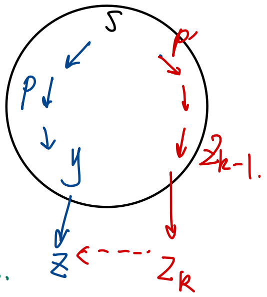
② $d[y] + W(y,z)$ is the min of all left node (minimization)

try to prove: W(P) from $s \to z$ is the shortest path

Proof 1: Let P' = any shortest path different from P

$P' = s, z_1, \dots z_k, \dots z$

Assume $z_k$ is the first vertex in P' not in set S.



$W(P) = d[y] + W(y, z)$

$W(P') = d[z_{k-1}] + W(z_{k-1}, z_k) +$ distance from $z_k$ to $z$.

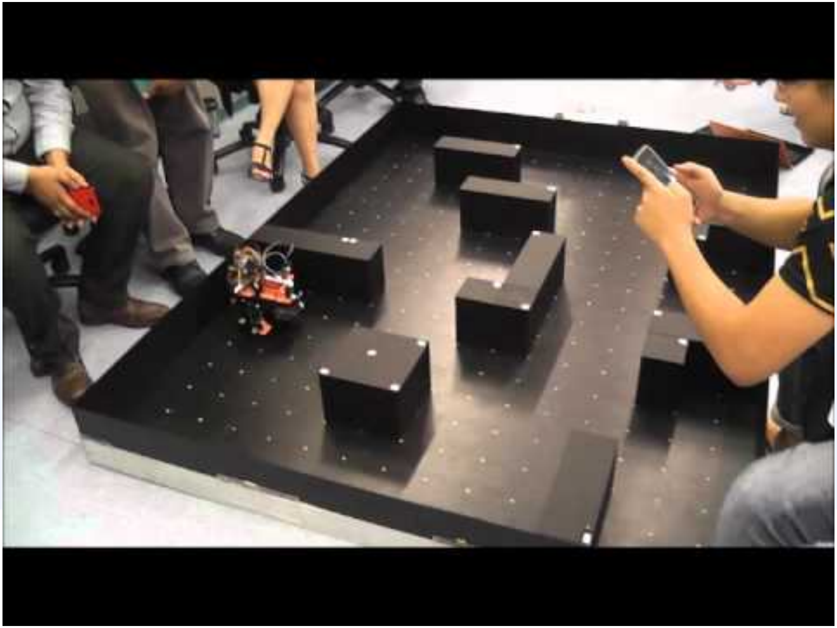
From ②, we know $d[y] + W(y,z) \le$ any other path in S + [V − S]

i.e. $d[y] + W(y,z) \le d[z_{k-1}] + W(z_{k-1}, z_k)$

Since distance from $z_k$ to $z$ is non-negative, therefore $W(P) \le W(P')$

NANYANG TECHNOLOGICAL UNIVERSITY

## Greedy Algorithms



## Learning Objectives

At the end of this lecture, students should be able to:

- Explain the strategy of Greedy algorithms
- Solve single-source shortest paths problem using Dijkstra’s algorithm
- Prove the correctness of Dijkstra’s algorithm
- Describe Prim’s algorithm for finding minimum spanning trees (MSTs)
- Prove the correctness of Prim’s algorithm

NANYANG TECHNOLOGICAL UNIVERSITY

## Greedy Algorithms

- In optimisation problems, the algorithm needs to make a series of choices whose overall effect is to minimise the total cost, or maximise the total benefit, of some system.

only make the best short-term decision

- There is a class of algorithms, called the **greedy algorithms**, in which we can find a solution by using only knowledge available at the time when the next choice (or guess) must be made.

- Each individual choice is the best within the knowledge available at the time.

## Greedy Algorithms

- Each individual choice is not very expensive to compute.
- A choice cannot be undone, even if it is found to be a bad choice later.
- Greedy algorithms cannot guarantee to produce the optimal solution for a problem.

NANYANG TECHNOLOGICAL UNIVERSITY

## Dijkstra’s Algorithm

**Shortest Path Problem:**

The problem of finding the shortest path from one vertex in a graph to another vertex. "Shortest" may be the least number of edges, or the least total weight, etc.

**Dijkstra's Algorithm:**

This is an algorithm to find the shortest paths from a single source vertex to all other vertices in a **weighted**, **directed** graph. All weights must be **nonnegative**.

**Dijkstra's algorithm keeps two sets of vertices:**

- **S** ---- the set of vertices whose shortest paths from the source node have already been determined [they form the tree]
- **V – S** ---- the remaining vertices

**The other data structures needed are:**

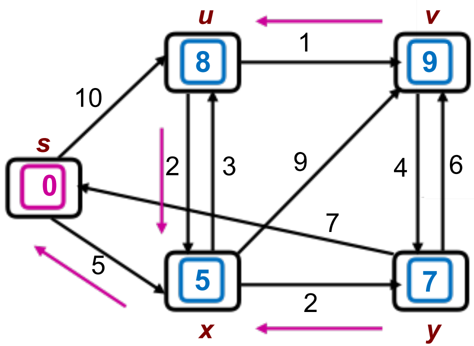
- **d** ---- array of estimates for the lengths of shortest paths from source node to all vertices
- **pi** ---- an array of predecessors for each vertex

## Basic Steps

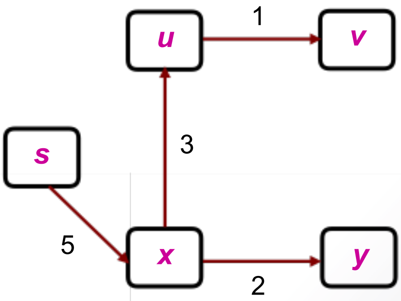
**The basic steps are:**

1. Initialise **d** and **pi**
2. Set **S** to empty
3. While there are still vertices in **V** – **S**
   - i. Move ***u***, the vertex in **V** – **S** that has the shortest path estimate from source, to **S**
   - ii. For all the vertices in **V** – **S** that are connected to ***u***, update their estimates of shortest distances to the source

# A Toy Example



Shortest paths from s to other vertices



## Pseudocode of Dijkstra's Algorithm

```
Dijkstra_ShortestPath ( Graph G, Node source ) {
  for each vertex v {
     d[v] = infinity;
     pi[v] = null pointer;
     S[v] = 0;
   }
  d[source] = 0;
  put all vertices in priority queue, Q, in d[v]'s increasing order;
  while not Empty(Q) {
        u = ExtractCheapest(Q);
        S[u] = 1; /* Add u to S */
```

## Pseudocode of Dijkstra's Algorithm

```
    for each vertex v adjacent to u
        if (S[v] ≠ 1 and d[v] > d[u] + w[u, v]) {
            remove v from Q;
            d[v] = d[u] + w[u, v];
            pi[v] = u;
            insert v into Q according to its d[v];
        }
  } // end of while loop
}
```

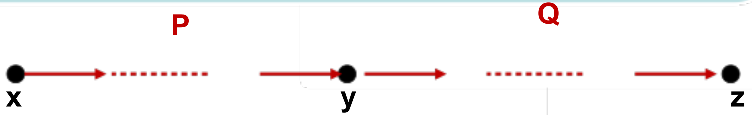
**Worst case time complexity of Dijkstra's algorithm is *O(|V|2)* (analysis not required).**

NANYANG TECHNOLOGICAL UNIVERSITY

## Property of Shortest Path

最短路径的子路径也是最短的.

**Lemma 1**: In a weighted graph G, suppose that a shortest path from x to z consists of a path P from x to y followed by a path Q from y to z. Then P is a shortest path from x to y and Q is a shortest path from y to z.



**Proof (By Contradiction):** (用反证法证明)
Assume that P is not the shortest path from x to y. Then there will be another path from x to y, P' which is shorter than P. As a result P' followed by Q will be a path **shorter** than P followed by Q. But it was known that P followed by Q is the **shortest** path. Contradiction. Same can be said about Q.

Main idea: the extension of solution set by dijaskta is always correct.

## Greedy choice is optimal

A rephrase of Dijsktra

**Theorem D1**: Let G = (V, E, W) be a weighted graph with nonnegative weights. Let S be a subset of V and let s be a member of S. [= every vertex is in the solution] Assume that d[y] is the shortest distance in G from s to y, for each y in S.] Let z be the next vertex chosen to go into S. If edge (y, z) is chosen to minimise d[y] + W(y, z) (= finding minimum d.) over all edges with one vertex in S and one vertex in V – S, then the path consisting of a shortest path from s to y followed by the edge (y, z) is the shortest path from s to z.

**Proof:**

We will show that there is no other path from s to z that is shorter.

## Proof of Theorem D1 (continued)

src vertex

**P:** s →y → z (shortest path for z)

in total 3 edge connected to y.

W(P) = d[y] + W(y, z)

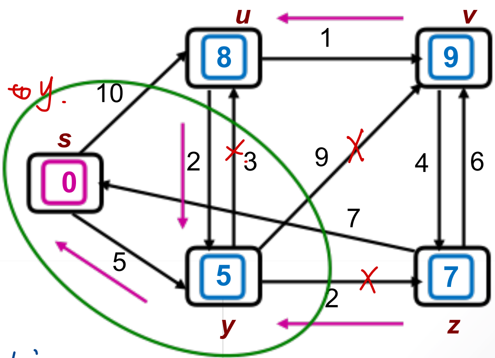
choosing edge[y,u]

**P':** s → y → u → … → z
(an alternative shortest path)

W(P') = d[y] + W(y, u)
+ distance from u to z

greedy choice

Because d[y] + W(y, u) $\overset{>}{□}$ d[y] + W(y, z),
and distance from u to z is nonnegative,
therefore W(P) □ W(P').



Edge (y, z) is chosen to minimise d[y] + W(y, z) over all edges with one vertex in S and one vertex in V – S

Greedy choice is optimal

## Proof of Theorem D1 (continued)

Let P be a shortest path from **s** to **y** followed by edge (y, z)

Let W(P) = the distance travelled along P

Let P’ = any shortest path different from P, i.e., P’ = s, z1, …, zk, …, z

Assume that zk is the first vertex in P’ not in set S.

W(P) = d[y] + W(y, z)

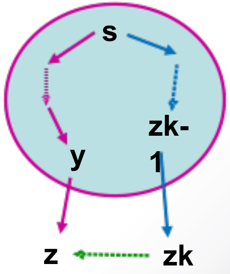
W(P’) = d[zk–1] + W(zk–1, zk) + distance from zk to z

Greedy Choice.

**Note that:** d[zk–1] + W(zk–1, zk) $\geq$ d[y] + W(y, z)

Since distance from zk to z is non-negative,

therefore, **W(P) <= W(P’).**



## Theorem D2 and Proof

**Theorem D2**: Given a directed weighted graph G with nonnegative weights and a source vertex s, Dijkstra's algorithm computes the shortest distance from s to each vertex of G that is reachable from s.

**Proof (By induction):**
We will show by induction that as each vertex v is added into set S, d[v] is the shortest distance from s to v.

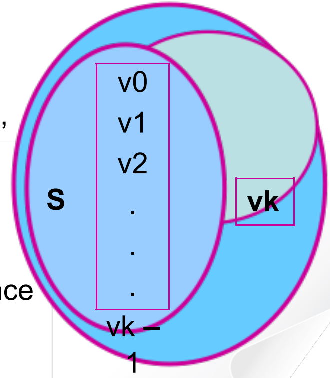
**Basis**: (Base Case)
The algorithm assigns d[s] to zero when the source vertex s is added to S. So d[s] is the shortest distance from s to s when S has the first vertex in it.

## Theorem D2 and Proof (Continued)

**Inductive Hypothesis:** ✓ level 1: Proof by contradiction
Assume the theorem is true when S has **k** vertices. That is, assume v0, v1, v2, …, vk–1 are added where d[v1], d[v2]… are the shortest distances.

When vk is chosen by Dijkstra's algorithm, it means an edge (vi, vk), where i ∈ {0, 1, 2, …, k – 1}, is chosen to minimise d[vi] + W(vi, vk) among all edges with one vertex in S and one vertex not in S.

By Theorem D1, d[vk] is the shortest distance from source to vk. So the theorem is true when S has **k + 1** vertices.



NANYANG TECHNOLOGICAL UNIVERSITY

## Minimum Spanning Tree

## Minimum Spanning Tree

### Definition of Spanning Tree

$|E| = |V| - 1$

(No cycle).

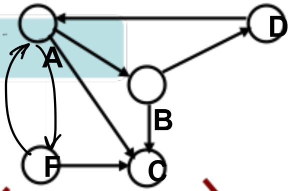
A connected, acyclic subgraph containing all the vertices of a graph.

$\Downarrow$ can't create new edge

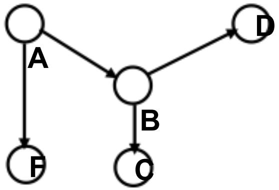
### Definition of Minimum Spanning Tree

A minimum-weight spanning tree in a weighted graph.

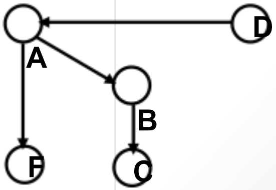
# Spanning Tree



Spanning Tree

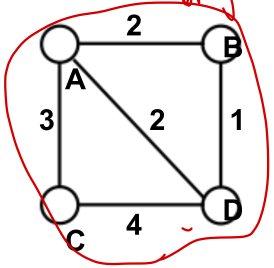


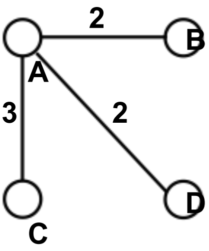
Another Spanning Tree



# Minimum Spanning Tree

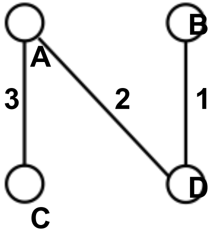
original graph.





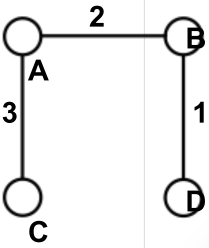
total weight = 7

Spanning Tree



total weight = 6

Minimum Spanning Tree



Minimum Spanning Tree

## Main Idea of Prim's Algorithm

**Prim's Algorithm**

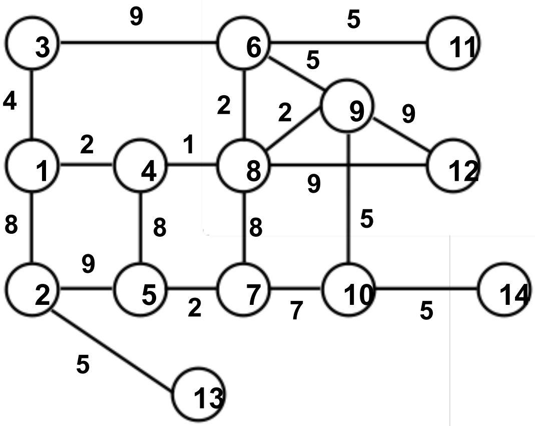
- It works on undirected graph.
- It builds upon a single partial minimum spanning tree, at each step adding an edge connecting the vertex nearest to but not already in the current partial minimum spanning tree.
- At first a vertex is chosen, this vertex will be the first node in ***T***.
- Set ***P*** is initialised: ***P*** = set of vertices not in tree ***T*** but are adjacent to some vertices in ***T***.

## Prim's Algorithm (Cont.)

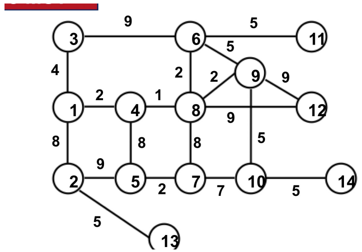
- In every iteration in the Prim's Algorithm, a new vertex ***u*** from set ***P*** will be connected to the tree ***T***. The vertex ***u*** will be deleted from the set ***P***. The vertices adjacent to ***u*** and not already in ***P*** will be added to ***P***.
- When all vertices are connected into ***T***, ***P*** will be empty. This means the end of the algorithm.
- The new vertex in every iteration will be chosen by using **greedy** method, i.e. among all vertices in ***P*** which are connected to some vertices already inserted in the tree ***T*** but themselves are not in ***T***, we choose one with the minimum cost.

# An Example of Prim's Algorithm

## Prim's MST

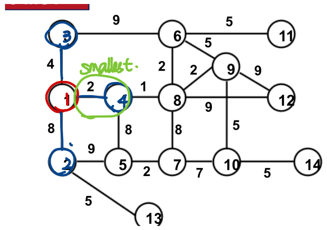


Black vertices: unseen vertices | Pink vertices: tree vertices | Blue vertices: fringe vertices

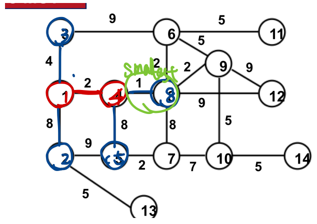


visited = { }.

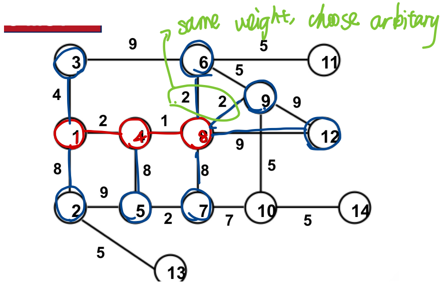
1. Choose an arbitary start.

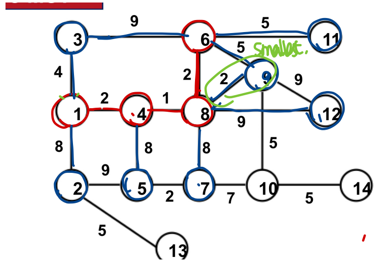


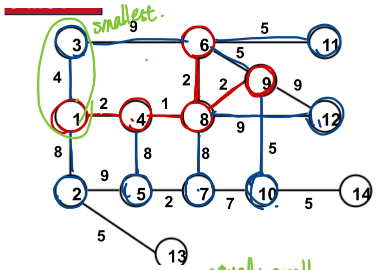
2. Make the greedy choice.

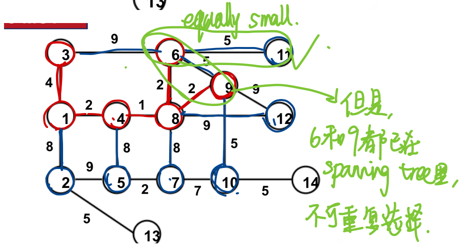


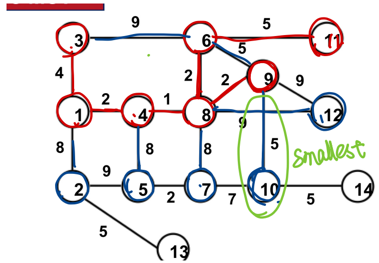
3. Repeat.

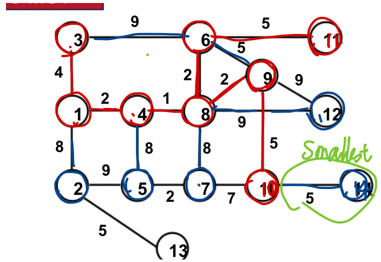


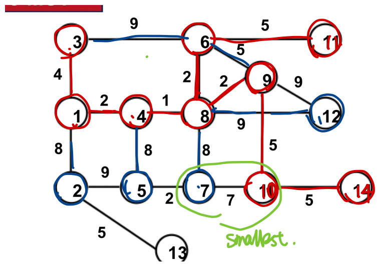


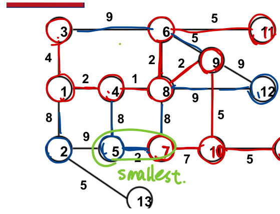


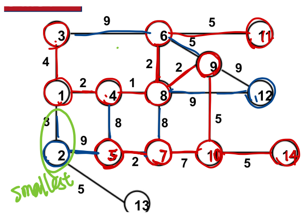


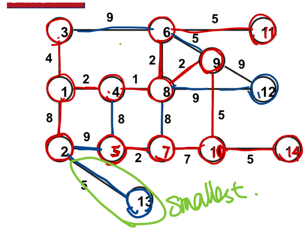


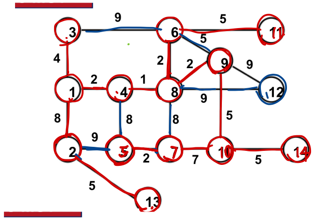


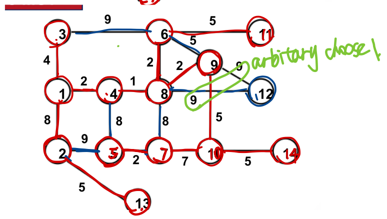


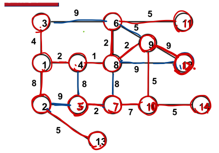








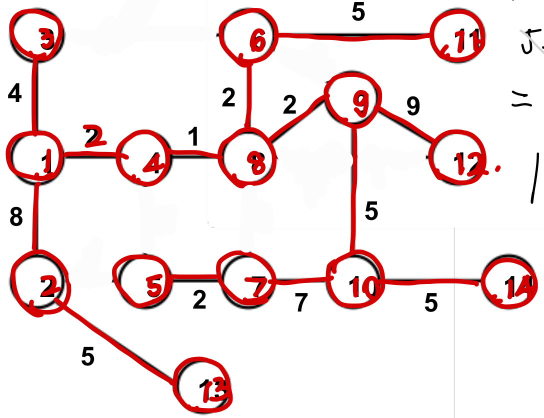




# An Example of Prim's Algorithm

Final Spanning

**Prim's MST**



total weight =

4+2+8+5+5+1+2+2+

5+2+7+5+9+5

= 57

|E| = |V| - 1

Black vertices: unseen vertices | Pink vertices: tree vertices | Blue vertices: fringe vertices

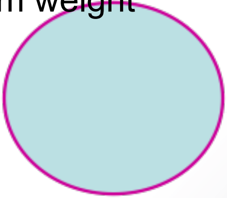
## 3 subsets of vertices

**Prim's Algorithm classifies vertices into three disjoint categories**:

- Tree vertices - in the tree being constructed so far
- Fringe vertices - not in the tree but adjacent to some vertices in the tree
- Unseen vertices - all others

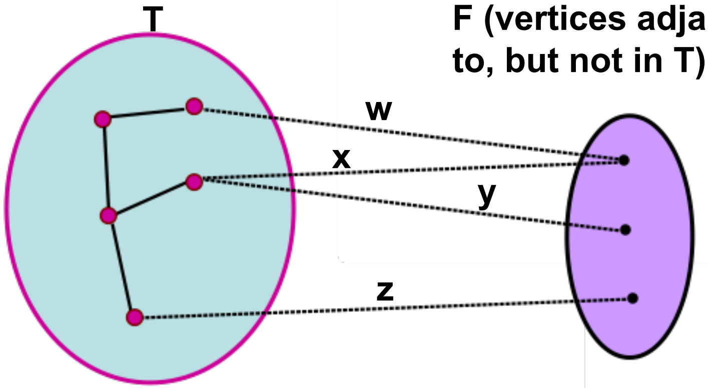
## Greedy choice of Prim's Algo

- Key step in the algorithm is the selection of a vertex from the fringe (which, of course, depends on the weights on incident edges).
- Prim's Algorithm always chooses a minimum weight edge from tree vertex to fringe vertex.



# Main Idea of Prim's Algorithm

F (vertices adjacent to, but not in T)



Choose min(w, x, y, z)

## Pseudocode of Prim's Algo

```
primMST(G, s, n)   // outline of Prim's algorithm
{
    Initialise all vertices as unseen.
    Reclassify s as tree vertex.
    Reclassify all vertices adjacent to s as fringe.
    While (there are fringe vertices)
    {
        Select an edge of minimum weight between a tree
                vertex t and a fringe vertex v;
        Reclassify v as tree; add edge tv to the tree;
        Reclassify all unseen vertices adjacent to v as fringe.
    }
}
```

## Implementing Prim's Algo

**Data Structures Used:**

not path. just weight

- Array **d**: distance of a fringe vertex from the tree
- Array **pi**: vertex connecting a fringe vertex to a tree vertex
- Array **S**: whether a vertex is in the minimum spanning tree being built
- Priority queue pq: queue of fringe vertices in the order of the distances from the tree

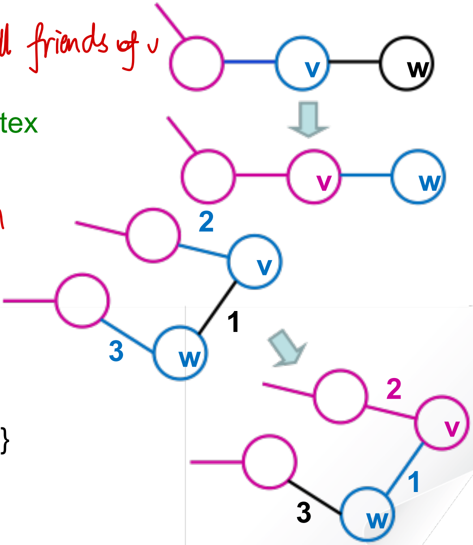
At the end of the algorithm, array pi has the minimum spanning tree.

## Implementing Prim's Algo

```
primMST(G, s, n)  {
   initialise priority queue pq as empty;
   for each vertex v {
       d[v] = infinity; S[v] = 0;
       pi[v] = null pointer; }   initialization
   d[s] = 0; S[s] = 1;
   insert(pq, s, 0);
   while (pq is not empty) {
       u = getMin(pq);    deleteMin(pq);
       S[u] = 1;
           updateFringe(pq, G, u);  }
}          update the friend of u.
```

## Update Fringe Set of Vertices

```
updateFringe(pq, G, v) {                          look at all friends of v
 for all vertices w adjacent to v {
   if (S[w] != 1) { //if w is not a tree vertex
      newWgt = weight of edge vw;
      if (d[w] == infinity) {                      w is unseen
         d[w] = newWgt; pi[w] = v;
         insert(pq, w, newWgt);
      } else if (newWgt < d[w]) {
         d[w] = newWgt; pi[w] = v;
         decreaseKey(pq, w, newWgt);}
   } // if w is not a tree vertex
 } // for all vertices
}
```



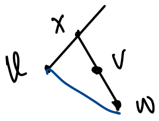
NANYANG TECHNOLOGICAL UNIVERSITY

MST Property

prove later

**Minimum Spanning Tree Property** $\Longleftrightarrow$ MST definition

Let *T* be a spanning tree of *G*, where *G* = (*V*, *E*, *W*) is a connected, weighted graph. Suppose that for every edge (*u*, *v*) of *G* that is not in *T*, if (*u*, *v*) is added to *T* it creates a cycle such that (*u*, *v*) is a maximum-weight edge on that cycle. Then *T* has the **Minimum Spanning Tree Property** (or **MST Property**, in short).



original tree edge $\begin{cases}(u,x)\\(x,v)\\(v,w)\end{cases}$

(u,w) max edge in cycle.

## Lemma 1 and Proof

**Lemma 1:** In a connected weighted graph $G = (V, E, W)$, if $T1$ and $T2$ are two spanning trees that have the MST property, then they have the same total weight.

**Proof by induction** on ***k*****, the number of edges in *T*1 but not *T2*** (there are also ***k*** edges in ***T2*** but not in ***T*1**).

**Basis:**

***k*** = 0; i.e. ***T*1** = ***T2***. Therefore, they have the same weight.

Weak hypothesis: 假设 case k-1 成立, 推 case k, 同样成立 ⇒ case k 成立

⇒ Strong hypothesis: 直接假设 case k 成立.

**Inductive hypothesis**: For *k* > 0, assume the lemma holds when there are *j* differing edges where $0 \le j < k$. $\le k-1$

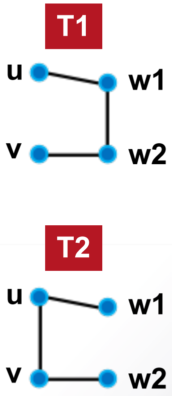
Let *uv* be the minimum weight edge among the differing edges (assume *uv* is in *T*2 but not *T*1).

Look at unique path in *T*1 from *u* to *v*.

Suppose it is made up of w0,w1,…,wp where
w0 = *u*, …, wp= *v*.
This path must contain some edge different from *T*2's.

Let wiwi + 1 be this differing edge.

By MST property of *T*1, wiwi + 1 cannot be > *uv*'s weight.



## Proof of Lemma 1 (continued)

But since *uv* was chosen to be the minimum weight among differing edges, wiwi+1 cannot have weight less than *uv*.

Therefore, W(wiwi+1) = W(*uv*).

Add *uv* to *T*1 (creating a cycle). Remove wiwi+1 leaving tree *T* '1 (which has the same weight as *T*1).

But *T* '1 and *T*2 differ only on *k* – 1 edges.

So by inductive hypothesis, *T* '1 and *T*2 have the same total weight. Therefore, *T*1 and *T*2 have same weight.

Page 392 Baase & Van Gelder

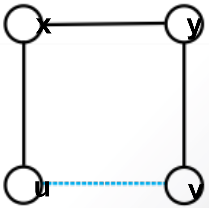
## Theorem 1 and Proof

**Theorem 1:** In a connected weighted graph G = (V, E, W), a tree *T* is a minimum spanning tree if and only if *T* has the MST property.

**Proof** (**Only if**): Assume *T* is an MST for graph G.

Suppose *T* does not satisfy the MST property, i.e. there is some edge *uv* that is not in *T* such that adding *uv* creates a cycle, in which some other edge *xy* has weight W(*xy*) > W(*uv*).

Then, by removing *xy* and adding *uv* ,we create a new spanning tree whose total weight is < W(*T*); This contradicts the assumption that *T* is an MST.



## Proof of Theorem 1 (continued)

**Theorem 1:** In a connected weighted graph G = (V, E, W), a tree *T* is a minimum spanning tree if and only if *T* has the MST property.

**Proof** (**Only if**): Assume *T* is an MST for graph G.

**(Cont.)**

**(If)** Assume *T* has MST property.

If *T*min is an MST, then *T*min has MST property by the first half of the proof.

By Lemma 1, W(*T*) = W(*T*min), so *T* is also an MST.

## Prim's Algorithm is Optimal

**Lemma 2:** Let G = (V, E, W) be a connected weighted graph; Let Tk be the tree with k vertices constructed by Prim's Algorithm, for k = 1, 2, …, n; and let Gk be the subgraph induced by the vertices of Tk. Then Tk has the MST property in Gk. (**Proof is not required**)

**Theorem 2:** Prim's Algorithm outputs a minimum spanning tree.

**Proof:**

- From Lemma 2, Tn has the MST property.
- By Theorem 1, Tn is a minimum spanning tree.

## Priority Queue for MST (Optional)

- Inserted by order of priority (not chronological, as in 'normal' queues – FIFO)
- Elements to be inserted have a 'key' - contains the priority; element with highest priority will be selected first. [priority can be largest value (e.g. if we're computing max profit) or smallest value (e.g. if we're interested in min cost)]
- Think of pq as a sequence of pairs: (id1,w1), (id2,w2),…, (idk,wk). The order is in increasing wi and id is a unique identifier for an element

## Methods of Priority Queue (Optional)

**The Priority Queue consists of:**

**Create:** Constructor to set up PQ
**isEmpty**; **getMin**; **getPriority:** Access functions
**insert**; **deleteMin**; **decreaseKey:** Manipulation procedures

**Insert(pq, id, w):** Inserts (id, w) into an existing pq - position depends on w

**decreaseKey(pq, id, neww):** Rearranges pq based on new wt of element id

**getMin(pq):** Returns id1;

**getPriorty(pq):** Returns weight of min element

## Summary

- Greedy algorithm is a general strategy to solve optimization problems
- Dijkstra's algorithm finds single-source shortest paths in a weighted graph of nonnegative edge weights
- Prim's algorithm finds the minimum spanning trees in weighted graphs
- Both are greedy algorithms, and use priority queue