

## Week 8: Lecture Notes

Topics: Dynamic programming  
Longest Common Subsequence  
Graphs  
Prim's Algorithms  
Graph Search.

### Dynamic Programming

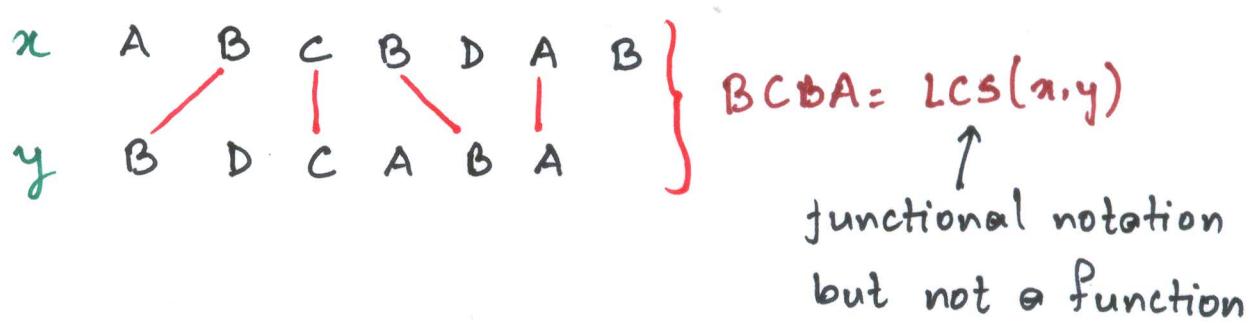
Design technique:

Like divide-and-conquer

Example:

### Longest Common Subsequence (LCS)

Given two sequences  $x[1 \dots m]$  and  $y[1 \dots n]$ ,  
find  $\underline{a}$  longest subsequence common to them both.  
 $\underline{a}$  not 'the'



## Brute-Force LCS Algorithm

Check every subsequence of  $\alpha[1 \dots m]$  to see if it is also a subsequence of  $\gamma[1 \dots n]$ .

### Analysis:

- Checking =  $O(n)$  time per subsequence
- $2^m$  subsequences of  $\alpha$  (each bit-vector of length  $m$  determines a distinct subsequence of  $\alpha$ ).

Worst case running time =  $O(n2^m)$

= exponential time

## Towards a better algorithm

### Simplification:

1. Look at the length of the longest-common subsequence.
2. Extend the algorithm to find the LCS of sequence  $s$ .

### Notation:

- Denote the length of a sequence  $s$  by  $|s|$

### Strategy:

- Consider prefixes of  $\alpha$  and  $\gamma$ .
- Define  $c[i,j] = |\text{LCS}(\alpha[1 \dots i], \gamma[1 \dots j])|$
- Then  $c[m,n] = |\text{LCS}(\alpha, \gamma)|$ .

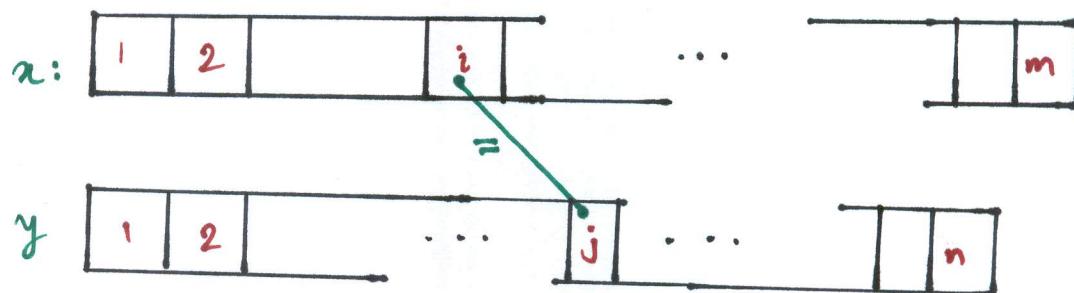
## Recursive Formulation

Theorem:

$$c[i,j] = \begin{cases} c[i-1, j-1] + 1 & \text{if } x[i] = y[j] \\ \max\{c[i-1, j], c[i, j-1]\} & \text{otherwise} \end{cases}$$

Proof:

Case  $x[i] = y[j]$ :



Let  $z[1 \dots k] = \text{LCS}(x[1 \dots i], y[1 \dots j])$ , where  $c[i, j] = k$ .

Then,  $z[k] = x[i]$ , or else  $z$  could be extended.

Thus,  $z[1 \dots k-1]$  is CS of  $x[1 \dots i-1]$  and  $y[1 \dots j-1]$

Claim:  $z[1 \dots k-1] = \text{LCS}(x[1 \dots i-1], y[1 \dots j-1])$ .

Suppose  $w$  is longer CS of  $x[1 \dots i-1]$  and  $y[1 \dots j-1]$ , that is  $|w| > k-1$ .

Then cut and paste:  $w||z[k]$  ( $w$  concatenated with  $z[k]$ ) is a common subsequence of  $x$  and  $y$  with  $|w||z[k]| > k$ .

Contradiction, proving claim.

Thus  $c[i-1, j-1] = k-1$ , which implies that  $c[i, j] = c[i-1, j-1] + 1$ .

Other cases are similar.

# Dynamic Programming hallmark #1

## Optimal Substructure:

An optimal solution to a problem (instance) contains optimal solutions to subproblems.

- If  $z = \text{LCS}(x, y)$ , then any prefix of  $z$  is an LCS of a prefix of  $x$  and a prefix of  $y$ .

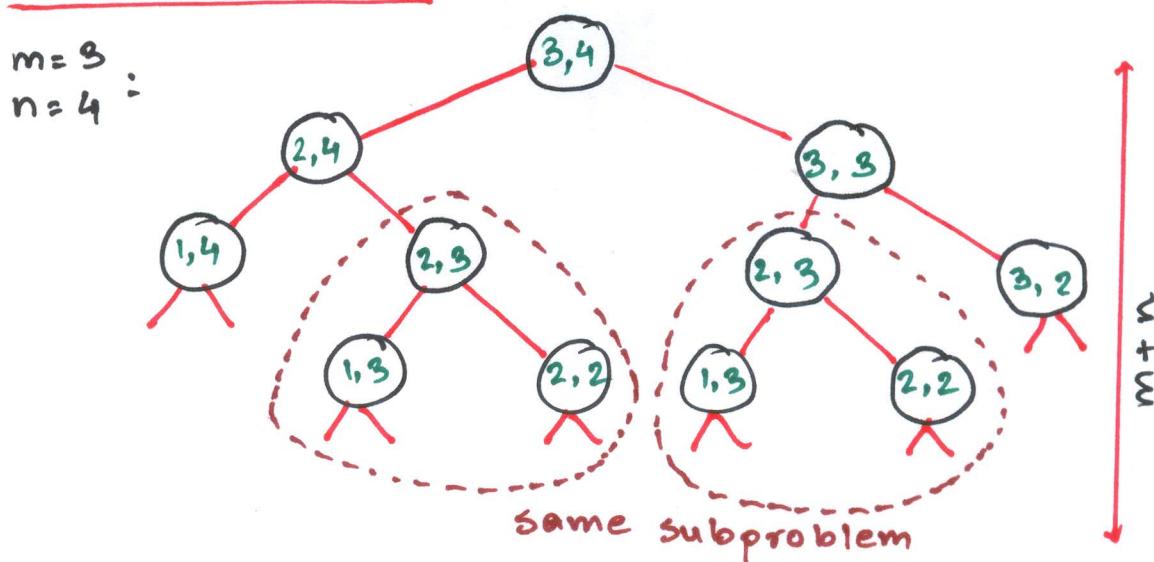
## Recursive Algorithm for LCS

$\text{LCS}(x, y, i, j)$

1. if  $x[i] = y[j]$
2. then  $c[i, j] \leftarrow \text{LCS}(x, y, i-1, j-1) + 1$
3. else  $c[i, j] \leftarrow \max \{ \text{LCS}(x, y, i-1, j), \text{LCS}(x, y, i, j-1) \}$

**Worst case:**  $x[i] \neq y[j]$ , in which case the algorithm evaluates two subproblems, each with only one parameter decremented.

## Recursion Tree



Height =  $m+n \Rightarrow$  work potentially exponential, but we are solving subproblems already solved.

## Dynamic Programming hallmark #2

### Overlapping subproblems:

A recursive solution contains a "small" number of distinct subproblems repeated many times.

- The number of distinct LCS subproblems for two strings of lengths m and n is only  $mn$ .

### Memoization Algorithm

#### Memoization:

After computing a solution to a subproblem, store it in a table. Subsequent calls check the table to avoid redoing work.

$\text{LCS}(x, y, i, j)$

if  $c[i, j] = \text{NIL}$

then if  $x[i] = y[j]$

then  $c[i, j] \leftarrow \text{LCS}(x, y, i-1, j-1) + 1$

else  $c[i, j] \leftarrow \{\text{LCS}(x, y, i-1, j),$

$\text{LCS}(x, y, i, j-1)\}$

} same  
as  
before

Time:  $\Theta(mn)$  = constant work per table entry

Space:  $\Theta(mn)$

## Graphs (review)

### Definition

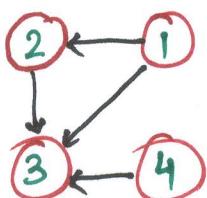
- A **directed graph (digraph)**,  $G = (V, E)$  is an ordered pair consisting of:
  - a set  $V$  of vertices (singular vertex)
  - a set  $E \subseteq V \times V$  of edges.
- In an **undirected graph**,  $G = (V, E)$ , the edge set  $E$  consists of unordered pair of vertices.
- In either case, we have  $|E| = O(|V|^2)$
- Moreover if  $G$  is connected, then  $|E| \geq |V| - 1$ , which implies that  $\log |E| = \Theta(\log V)$

### Adjacency matrix representation

The **adjacency matrix** of a graph  $G = (V, E)$ , where  $V = \{1, 2, \dots, n\}$ , is the matrix  $A[1 \dots n, 1 \dots n]$  given by

$$A[i,j] = \begin{cases} 1 & \text{if } (i,j) \in E \\ 0 & \text{if } (i,j) \notin E \end{cases}$$

### Example.



A	1	2	3	4
1	0	1	1	0
2	0	0	1	0
3	0	0	0	0
4	0	0	1	0

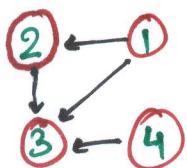
$O(V^2)$  storage

$\Rightarrow$  **dense representation**.

## Adjacency-list representation

An adjacency list of a vertex  $v \in V$  is the list  $\text{Adj}[v]$  of vertices adjacent to  $v$ .

Example:



$$\text{Adj}[1] = \{2, 3\}$$

$$\text{Adj}[2] = \{3\}$$

$$\text{Adj}[3] = \{4\}$$

$$\text{Adj}[4] = \{3\}$$

For undirected graphs:

$$|\text{Adj}[v]| = \text{degree}(v)$$

For digraphs,  $|\text{Adj}[v]| = \text{out-degree}(v)$

Handshaking Lemma:

$$\sum_{v \in V} = 2|E| \text{ for undirected graphs}$$

$\Rightarrow$  adjacency list use  $\Theta(V+E)$  storage

(a sparse representation for either type of graph)

## Minimum Spanning Tree (MST)

**Input:** A connected, undirected graph  $G = (V, E)$  with weight function  $w: E \rightarrow \mathbb{R}$ .

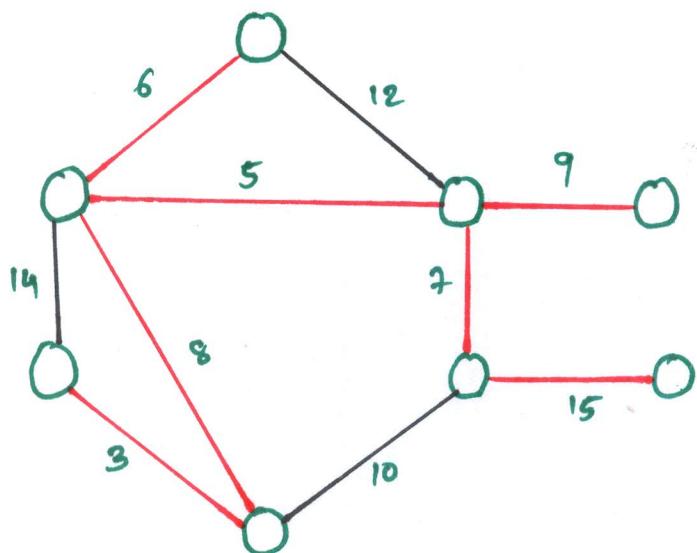
- For simplicity, assume that all edge weights are distinct.

**Output:**

A spanning tree  $T$  - a tree that connects all vertices - of minimum weight.

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

## Example of MST

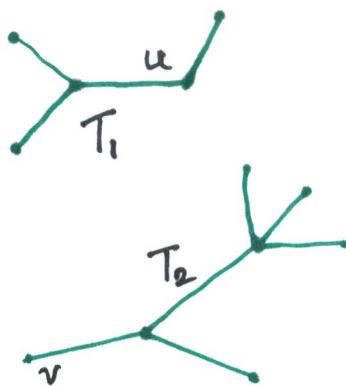


## Optimal Substructure

Consider MST  $T$  of graph  $G$ .

Remove any edge  $(u, v) \in T$ .

Then  $T$  is partitioned into two subtrees  $T_1$  and  $T_2$ . (other edges of  $G$  are not shown)



**Theorem:**

The subtree  $T_1$  is an MST of  $G_1 = (V_1, E_1)$ , the graph induced by the vertices of  $T_1$ :

$$V_1 = \text{vertices of } T_1$$

$$E_1 = \{(x, y) \in E : x, y \in V_1\}$$

Similarly for  $T_2$ .

**Proof:**

Cut and paste:

$$w(T) = w(u, v) + w(T_1) + w(T_2)$$

If  $T'$  were a lower-weight spanning tree than  $T_1$  for  $G_1$ , then  $T' = \{(u, v)\} \cup T'_1 \cup T_2$  would be a lower-weight spanning tree than  $T$  for  $G$ . ~~for  $G$~~

Do we also have overlapping subproblems?

- Yes
  - Great, then dynamic programming may work
- Yes, but MST exhibits another powerful property which leads to an even more efficient algorithm.

Q61

## Hallmark for Greedy Algorithm

### Greedy Choice Property:

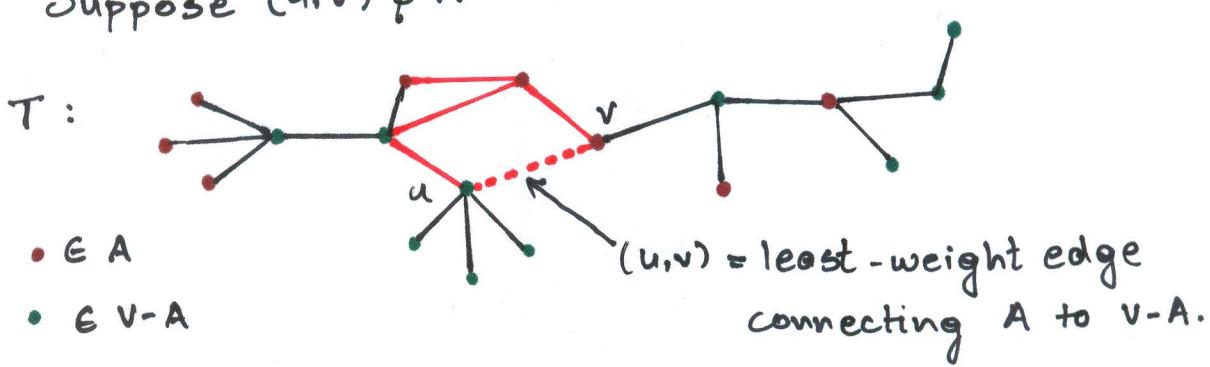
A locally optimal choice is globally optimal

#### Theorem:

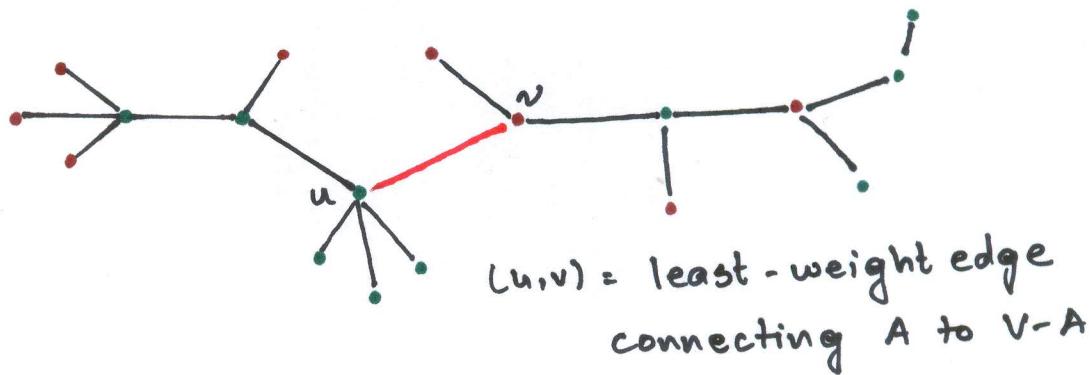
Let  $T$  be the MST of  $G = (V, E)$ , and let  $A \subseteq V$ . Suppose that  $(u, v) \in E$  is the least-weight edge connecting  $A$  to  $V - A$ . Then  $(u, v) \in T$ .

#### Proof:

Suppose  $(u, v) \notin T$ .



Consider the unique simple path from  $u$  to  $v$  in  $T$ .  
Swap  $(u, v)$  with the first edge on this path that connects a vertex in  $A$  to a vertex in  $V - A$ .



A lighter-weight spanning tree than  $T$  results. //

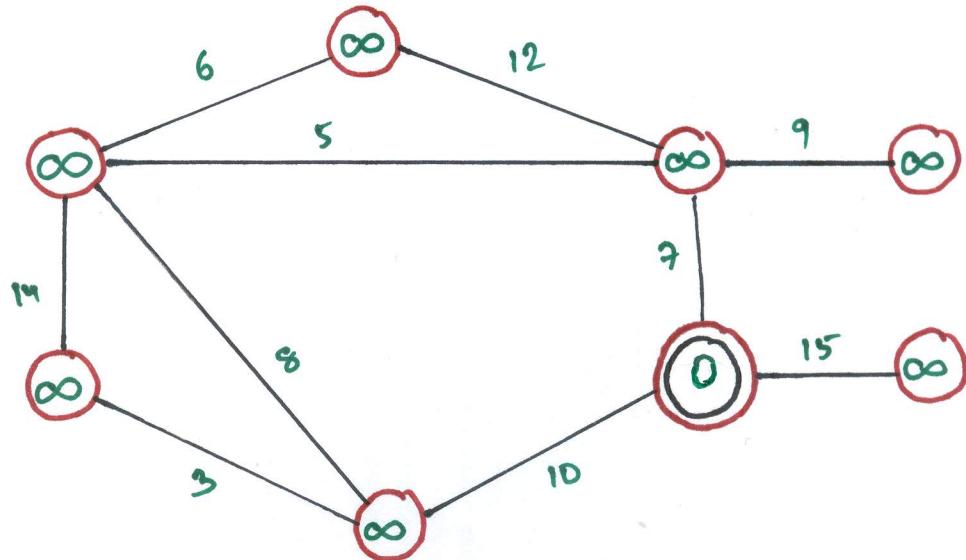
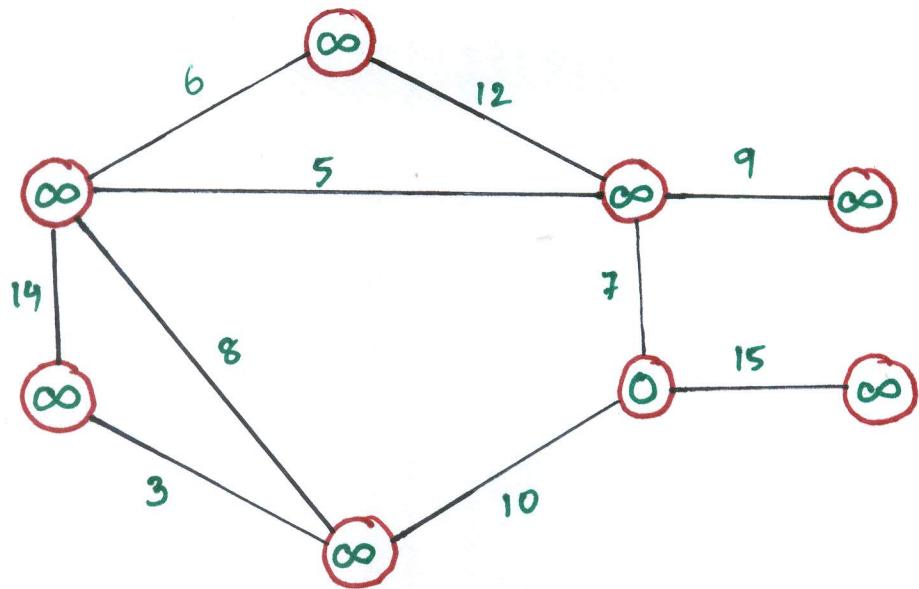
## Prim's Algorithm

**Idea:** Maintain  $V - A$  as a priority queue  $Q$ . Key each vertex in  $Q$  with the weight of least-weight edge connecting it to a vertex in  $A$ .

1.  $Q \leftarrow V$
2.  $\text{key}[v] \leftarrow \infty$  for all  $v \in V$
3.  $\text{key}[s] \leftarrow 0$  for some arbitrary  $s \in V$
4. while  $Q \neq \emptyset$
5.     do  $u \leftarrow \text{EXTRACT-MIN}(Q)$
6.         for each  $v \in \text{Adj}[u]$
7.             do if  $v \in Q$  and  $w(u,v) < \text{key}[v]$
8.                 then  $\text{key}[v] \leftarrow w(u,v)$   
 $\pi[v] \leftarrow u$
- 9.

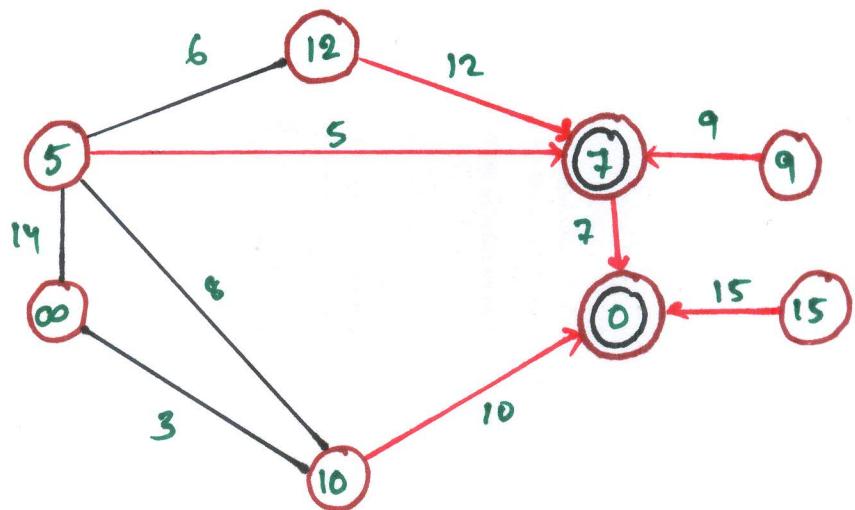
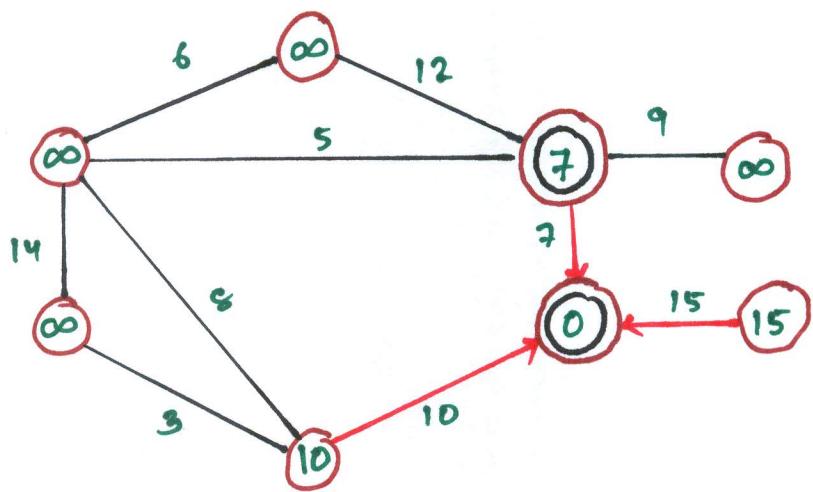
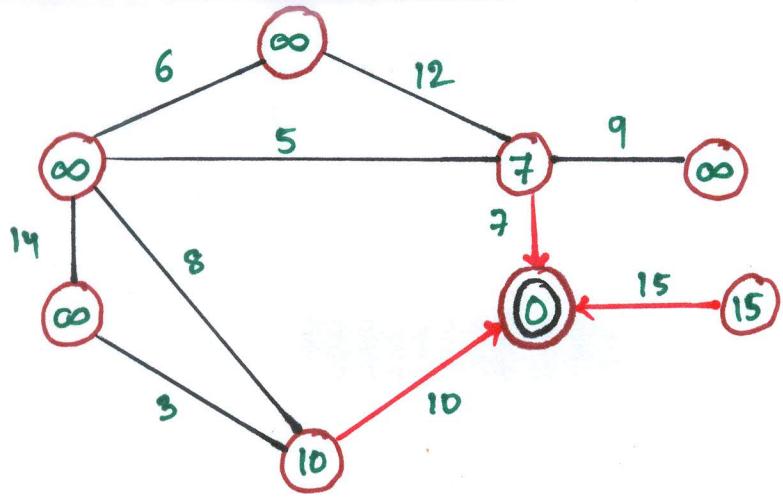
At the end  $\{v, \pi[v]\}$  forms the MST.

## Example of Prim's Algorithm



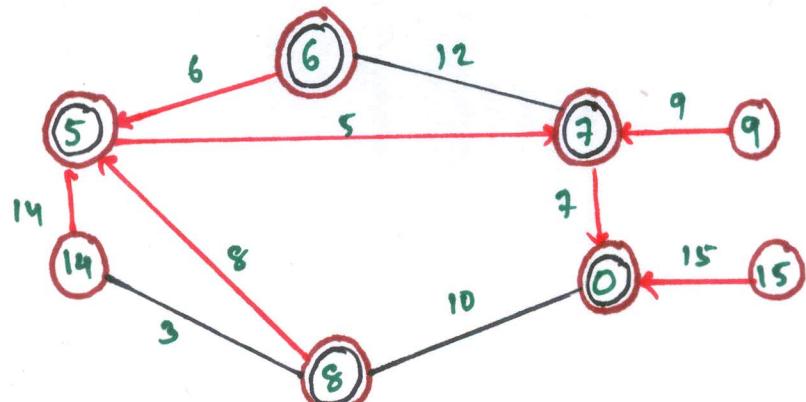
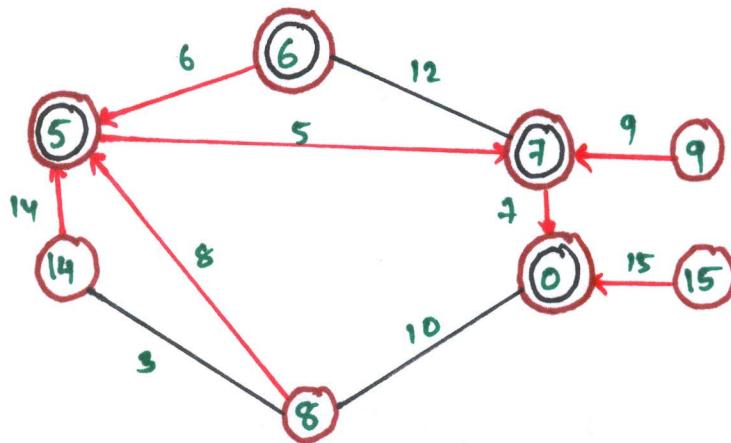
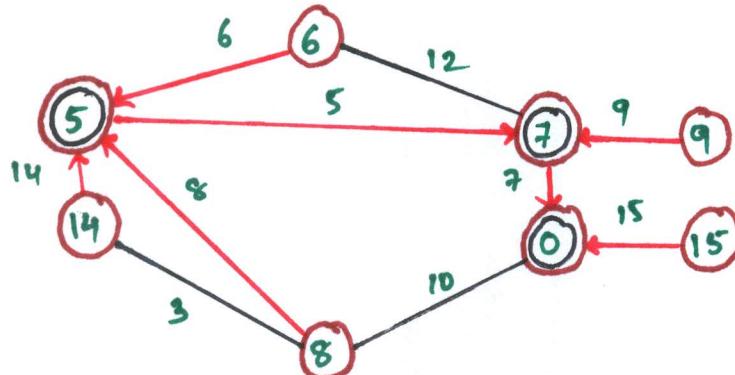
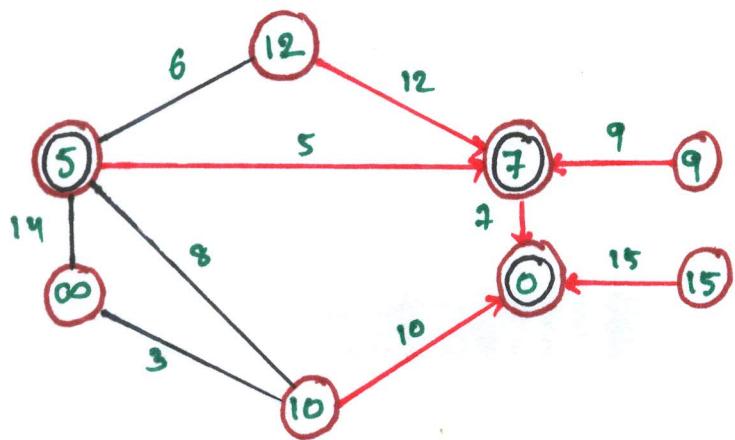
$\textcircled{0} \in A$

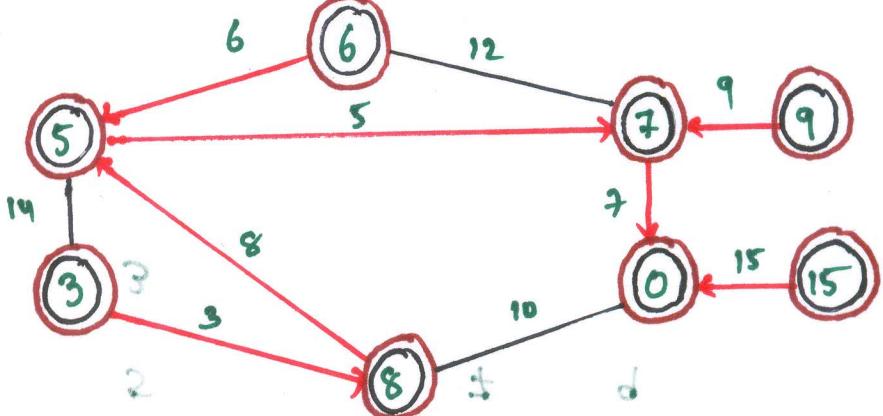
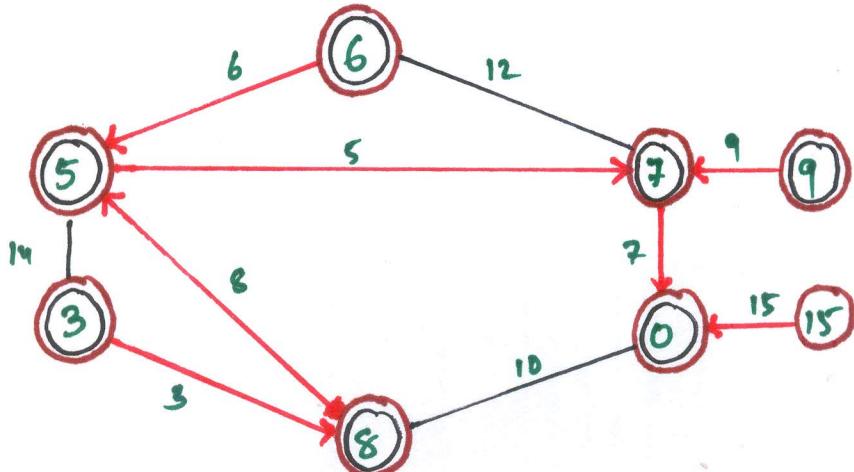
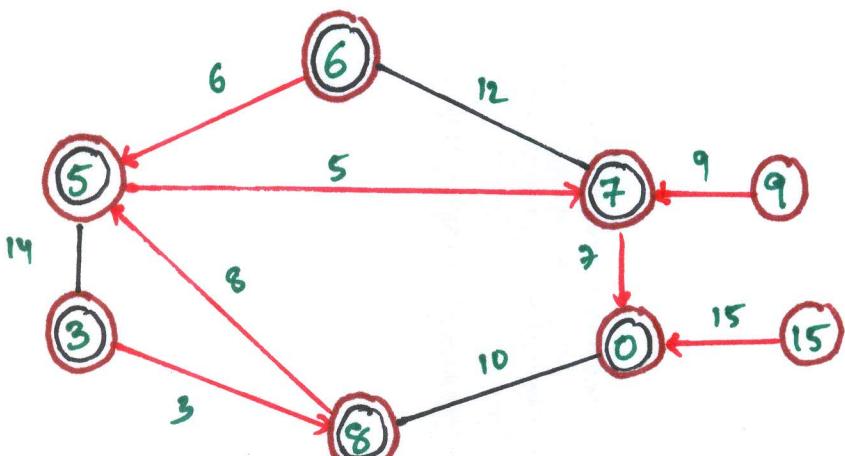
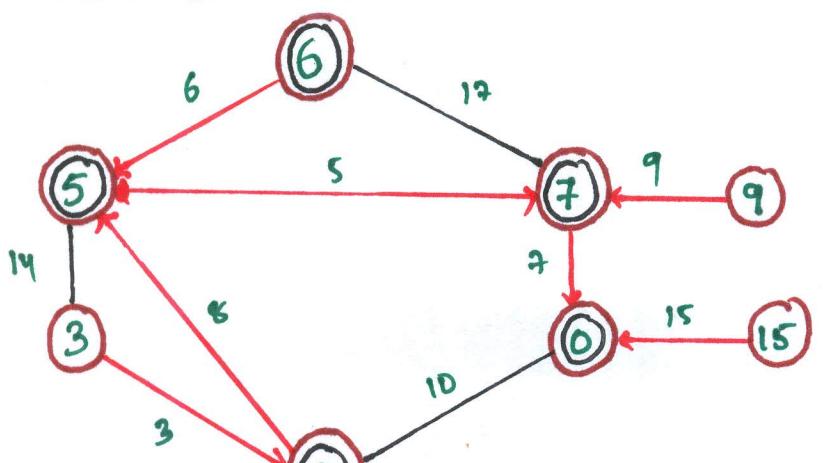
$\textcircled{\infty} \in V - A$



$\textcircled{0} \in A$

$\textcircled{0} \in V-A$





## Analysis of Prim

$O(V)$  total       $\left\{ \begin{array}{l} Q \leftarrow V \\ \text{key}[v] \leftarrow \infty \text{ for all } v \in V \\ \text{key}[s] \leftarrow 0 \text{ for some arbitrary } s \in V \end{array} \right.$   
 $|V|$  times      while  $Q \neq \emptyset$   
 $\quad \quad \quad$  do  $u \leftarrow \text{EXTRACT-MIN}(Q)$   
 $\quad \quad \quad$  for each  $v \in \text{Adj}[u]$   
 $\quad \quad \quad$  times      do if  $v \in Q$  and  $w(u,v) < \text{key}[v]$   
 $\quad \quad \quad$  then  $\text{key}[v] \leftarrow w(u,v)$   
 $\quad \quad \quad$              $\pi[v] \leftarrow u$

Handshaking lemma  $\Rightarrow \Theta(E)$  implicit DECREASE-KEY's

Time =  $\underline{\Theta(V) \cdot T_{\text{EXTRACT-MIN}}} + \underline{\Theta(E) \cdot T_{\text{DECREASE-KEY}}}$

Q	$T_{\text{EXTRACT-MIN}}$	$T_{\text{DECREASE-KEY}}$	Total
array	$O(V)$	$O(1)$	$O(V^2)$
binary heap	$O(\log V)$	$O(\log V)$	$O(E \log V)$
Fibonacci heap	$O(\log V)$ amortized	$O(1)$ amortized	$O(E + V \log V)$ worst case

## MST Algorithms

### Kruskal's algorithm:

- Uses the disjoint-set data structure
- Running time =  $O(E \log V)$

### Best to date:

- Karger, Klein, and Tarjan [1993]
- Randomized algorithm
- $O(V+E)$  expected time.

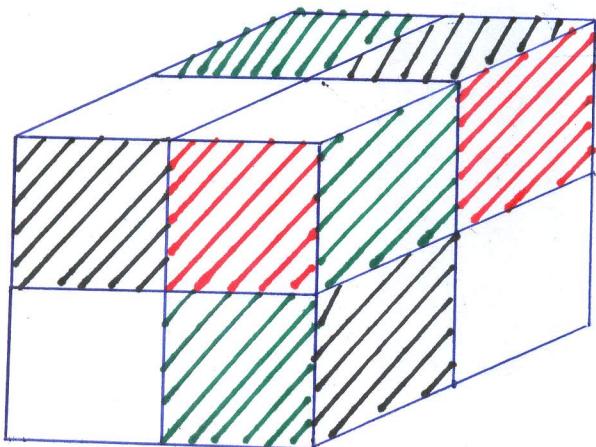
## Graph Search

"Explore a graph", e.g.:

- find a path from start vertex  $s$  to a desired vertex
- visit all vertices or edges of graph, or only those reachable from  $s$ .

## Pocket Cube

Consider a  $2 \times 2 \times 2$  Rubik's cube

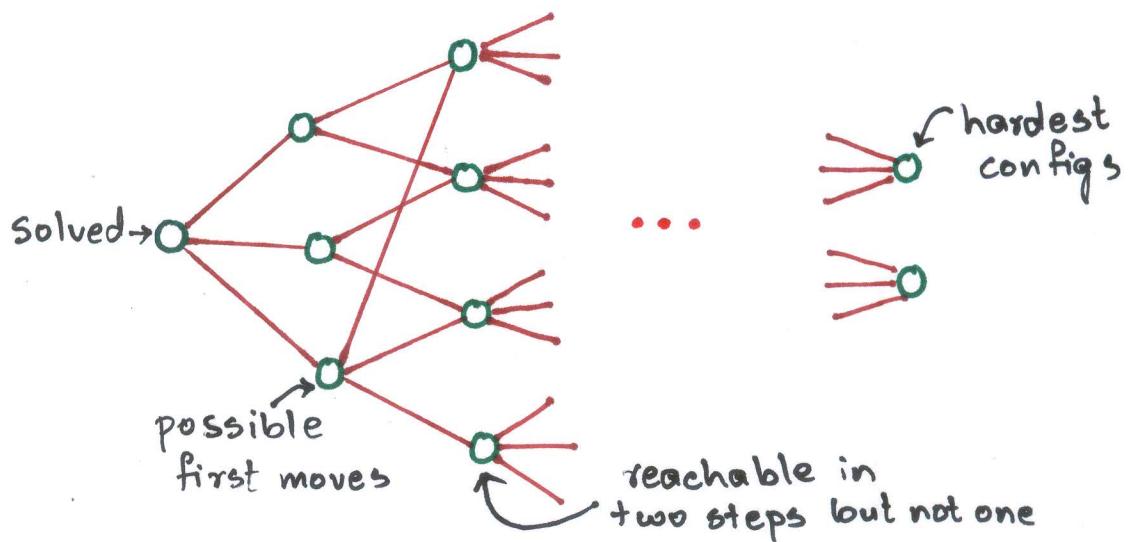


## Configuration graph:

vertex for each possible state

edge for each basic move (e.g.  $90^\circ$  turn) from one state to another.

undirected: moves are reversible.

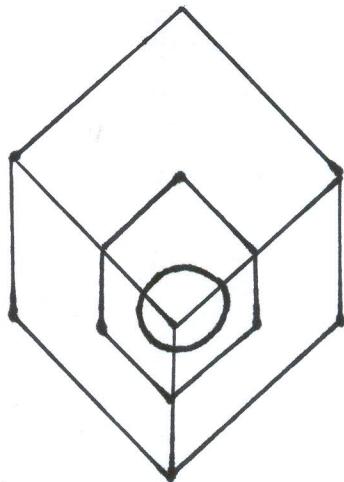


## Diameter ("God's Number")

- 11 for  $2 \times 2 \times 2$
- 20 for  $3 \times 3 \times 3$
- $\Theta(n^2/\log n)$  for  $n \times n \times n$  [Demaine, Demaine, Eisenstat, Lubiw Winslow 2011]

- Number of vertices =  $8! \cdot 3^8 = 264,539,520$

where  $8!$  comes from having 8 cubelets in arbitrary positions and  $3^8$  comes from as each cubelet has 3 possible twists.



This can be divided by 24 if we remove cube symmetries and further divided by 3 to account for actually reachable configurations (there are 3 connected components).