Lecture 9: Trees

Xiaoxing Ma

Nanjing University

xxm@nju.edu.cn

November 25, 2014

Acknowledgement: These Beamer slides are totally based on the textbook *Discrete Mathematical Structures*, by B. Kolman, R. C. Busby and S. C. Ross, and Prof. Daoxu Chen's PowerPoint slides.

At the Last Class

1. Finite Boolean Algebra
    - Boolean algebra: a special type of lattice
    - Substitution rule for Boolean algebra
2. Logical Design
    - Boolean expressions
    - Circuit Design

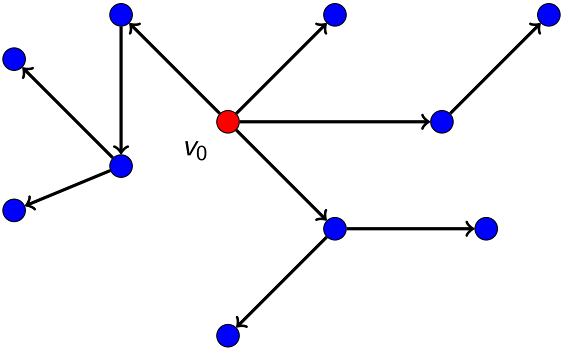
Overview

1. Rooted Trees
   - Basic properties of rooted tree
   - Labeled tree and its representation
   - Tree searching

2. Undirected Trees
   - Undirected graph: as a symmetric closure
   - Basic properties of undirected tree
   - Minimal spanning tree and its algorithm

Rooted Tree

Let A be a set, and let T be a relation on A. T is a **rooted tree** if there is a vertex v_0 in A with the property that there exists a unique path in T from v_0 to every other vertex in A, but no path from v_0 to v_0.



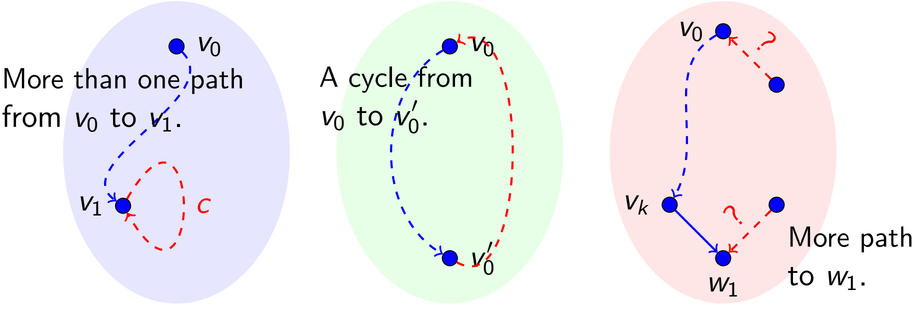
Properties of Rooted Tree

Let (T, v_0) be a rooted tree. Then

(a) There are no cycle in T.

(b) v_0 is the only root of T.

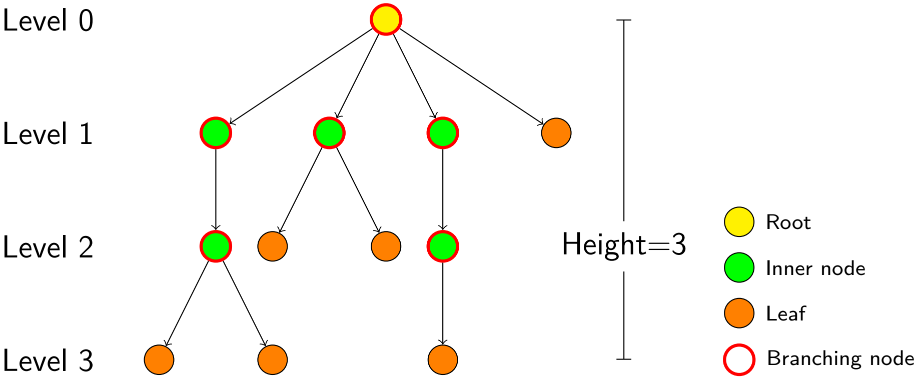
(c) Each vertex other than the root has in-degree one, and the root has in-degree 0



(a) (b) (c)

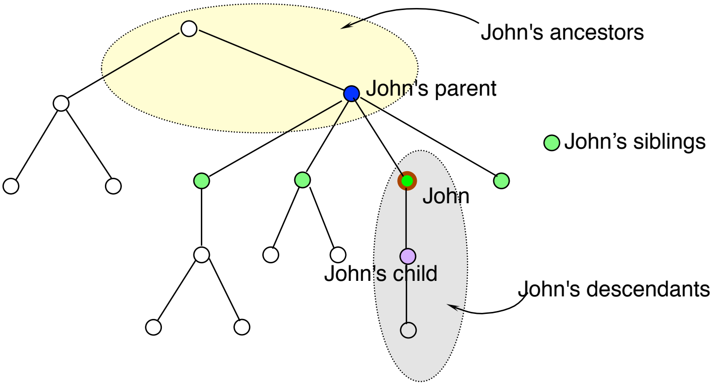
Drawing a Rooted Tree by Levels

All edges downward



Rooted Tree and Family Relations

It is easy to describe the family relations, and on the other hand, terms about family relations are used in rooted trees.



Some Terms about Rooted Tree

Ordered tree: the ordering is assumed on vertices in each level;

n-tree: every vertex has at most n offspring;

Complete n-tree: every vertex, other than leaves, has exactly n offspring;

Binary tree: 2-tree.

Subtree of a Rooted Tree

Theorem

*If (T, v_0) is a rooted tree and $v \in T$. Let $T(v)$ be the set of v and all its descendants, then $T(v)$ and all edges with their two ends in $T(v)$ is a tree, with v as its root. (It is called a **subtree** of (T, v_0))*

Subtree of a Rooted Tree

Theorem

If (T, v_0) is a rooted tree and $v \in T$. Let $T(v)$ be the set of v and all its descendants, then $T(v)$ and all edges with their two ends in $T(v)$ is a tree, with v as its root. (It is called a ***subtree*** *of (T, v_0))*

Proof.

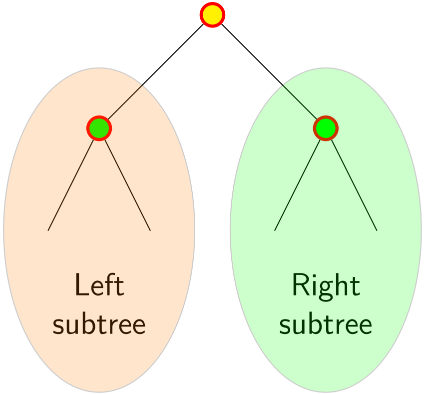
- There is a path from v to any other vertex in $T(v)$ since they are all the descendants of v;
- There cannot be more than one path from v to any other vertex w in $T(v)$, otherwise, in (T, v_0), there are more than one path from v_0 to w, both through v;
- There cannot be any cycle in $T(v)$, since any cycle in $T(v)$ is also in (T, v_0)

$\square$

Subtrees of Binary Tree

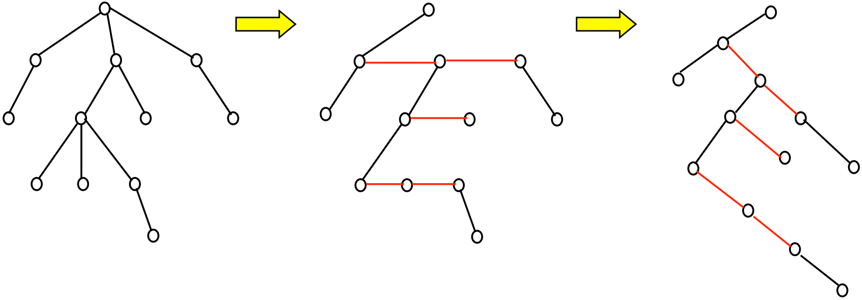
In a ordered binary tree, a subtree is a left subtree or a right subtree.

Even if a vertex has only one offspring, its subtree can be identified as left or right by its location in the digraph.



Ordered Binary Tree

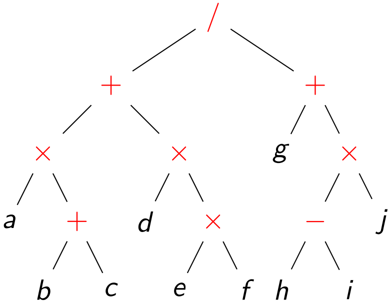
Any ordered tree can be converted into a ordered binary tree.



Labeled Tree: an Example

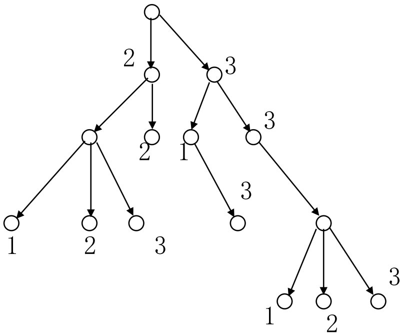
Using rooted tree to represent a arithmetic expressions:

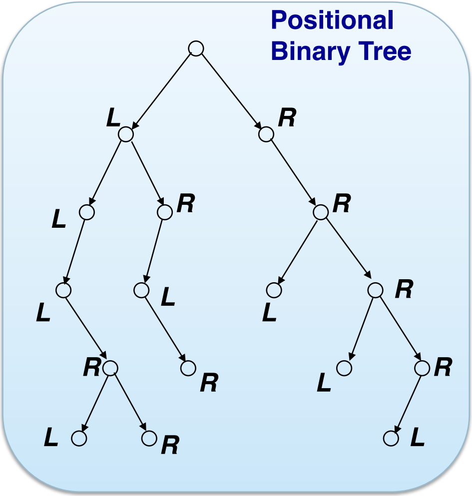
- branching vertices corresponding operators
- leaves corresponding operands



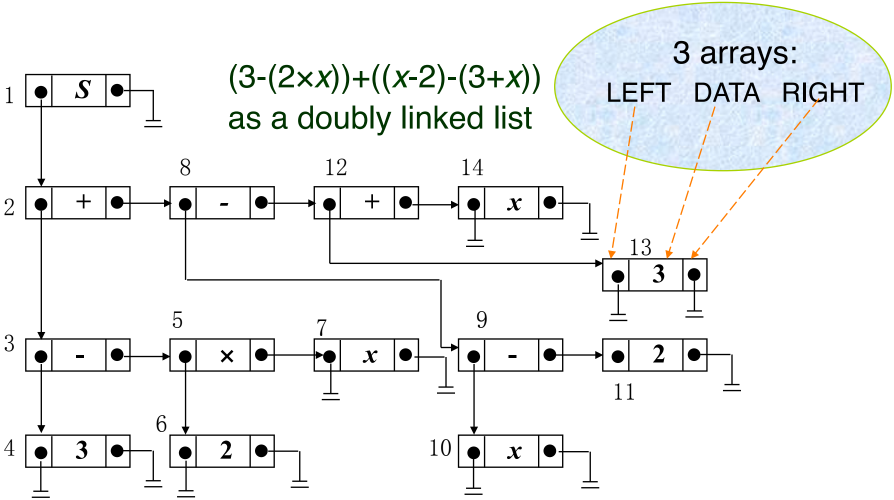
$$(a \times (b + c) + d \times (e \times f))/(g + (h - i) \times j)$$

Positional Trees





Computer Representation



Tree Searching

Tree recursive algorithm to search all vertices:

- Inorder: left, root, right

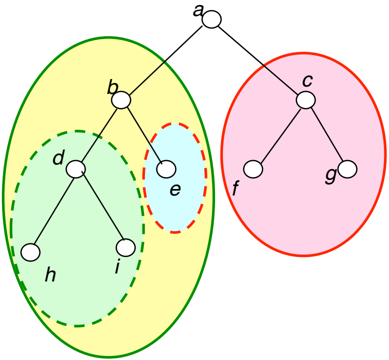
  $h, d, i, b, e, a, f, c, g$

- Preorder: root, left, right

  $a, b, d, h, i, e, c, f, g$

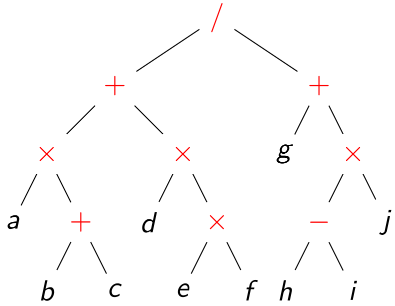
- Post order: left, right, root

  $h, i, d, e, b, f, g, c, a$



Reverse Polish Notation

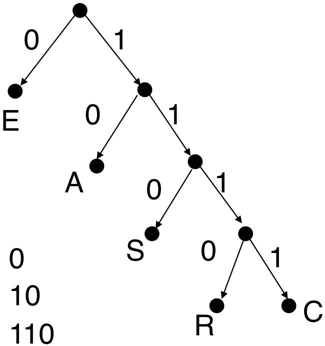
Example: $(a \times (b + c) + d \times (e \times f))/(g + (h - i) \times j)$



Searching in postorder: $abc + \times def \times \times + ghi - j \times + /$

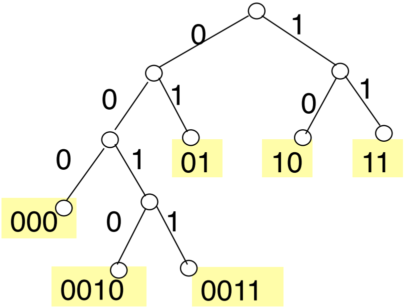
It is called **reverse Polish notation**. (*No parenthesis are needed!*)

Huffman Code Tree



E: 0
A: 10
S: 110
R: 1110
C: 1111

0101100: EASE

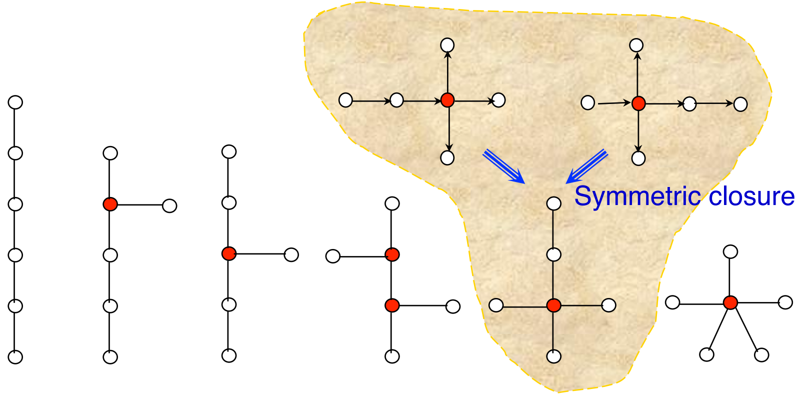


Undirected Tree

- An **undirected tree** is the symmetric closure of a tree.
- An undirected tree is represented by its graph, which has a single line without arrows connecting vertices a and b.
- The set $\{a, b\}$, where (a, b) and (b, a) are in T, is called an **undirected edge**, and a and b are called adjacent vertices.

Undirected Tree: Examples

Different undirected trees with six vertices:



Path and Cycle in a Tree

- Let $p : v_1, v_2, \cdots, v_n$ be path in a symmetric relation R, then p is **simple** if no two edges of p correspond to the same undirected edge.
- In above, if v_1 is equal to v_n , then p is a **simple cycle**.
- A symmetric relation R is **acyclic** if it contains no simple cycles.
- A symmetric relation R is **connected** if there is a path in R from any vertex to any other vertex.

Properties of an Undirected Tree

Let R be a symmetric relation on a set A. R is an undirected tree if and only if R is connected and acyclic.

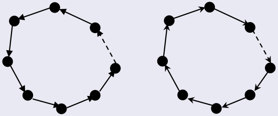
Properties of an Undirected Tree

Let R be a symmetric relation on a set A. R is an undirected tree if and only if R is connected and acyclic.

Proof.

$\Rightarrow$ Let R is the symmetric closure of some tree T.

- Suppose that R has a simple cycle $p : v_1, v_2, \cdots, v_n, v_1$. Then there is a figure of edges as the following in T. However, all possible orientation of the edges results in a cycle in T.



Note: this is impossible in a tree:

- Let v is the root of T. For any vertices u and w, there must be vu-path and vw-path in T, so, there are uv-path and vw-path in R. So, there is a uv-path in R.

□

Properties of an Undirected Tree (cont.)

Proof.

(cont.)

$\Leftarrow$ Suppose that R is a symmetric relation on a set A, and it is connected and acyclic

- Let v is any vertex in A. Since R is connected, there is a path from v to any other vertices, but not to v itself.
- Suppose that there are two paths from v to some w. There must be two vertices v', w', on both paths such that there are no common vertices on two different $v'w'$-path, since R is symmetric, one $v'w'$-path and the reverse of another $v'w'$-path form a cycle in R, contradiction.

$\square$

Unique Path

If T is an undirected tree, then for any vertices u, v, there is a unique simple uv-path in T.

Unique Path

If T is an undirected tree, then for any vertices u, v, there is a unique simple uv-path in T.

Proof.

We know that T is connected, so, there is at least one uv-path in T. Suppose that there are two different uv-paths P, Q in T. Without loss of generality, there exists an edge $e = (x, y)$ satisfying $e \in P$, and x is nearer to u on P than y, but $e \notin Q$. Let $T^* = T - \{e\}$, then T^* contains Q. Note that xu-segment on $P + Q + vy$-segment on P is an xy-path in T^*. However, this path plus e is a cycle in T. Contradiction. □

No Edge Can Be Removed

Let T is an undirected tree, e is any edge in T, then $T - e$ is no longer connected.

No Edge Can Be Removed

Let T is an undirected tree, e is any edge in T, then $T - e$ is no longer connected.

Proof.

We have know that for any vertices v, w, there is a unique vw-path. Let $e = (x, y)$, then e is the unique path between x and y. So, there is no xy-path in $T - \{e\}$, which means that $T - \{e\}$ is no longer connected. $\square$

Adding One Edge Means Cycle

Let T be an undirected tree, u, v are two vertices not adjecent to each other, then $T + \{(u, v)\}$ must contain a cycle.

Adding One Edge Means Cycle

Let T be an undirected tree, u, v are two vertices not adjecent to each other, then $T + \{(u, v)\}$ must contain a cycle.

Proof.

Since T is a undirected tree, there must be a uv-path in T. Let it be P, then $P + \{(u, v)\}$ is a cycle in $T + \{(u, v)\}$. □

Adding One Edge Means Cycle

Let T be an undirected tree, u, v are two vertices not adjecent to each other, then $T + \{(u, v)\}$ must contain a cycle.

Proof.

Since T is a undirected tree, there must be a uv-path in T. Let it be P, then $P + \{(u, v)\}$ is a cycle in $T + \{(u, v)\}$. □

In fact, we can prove that there is only one cycle in $T + \{(u, v)\}$.

Number of Vertices and Edges

A tree with n vertices has $n - 1$ edges

Number of Vertices and Edges

A tree with n vertices has $n-1$ edges

Proof.

- There are at least $n-1$ edges to connect n vertices.
- Suppose that there are more than $n-1$ edges. So, the sum of in-degree of all vertices must be more than $n-1$. However, the in-degree of the root is zero, and in-degree of any of the other $n-1$ vertices is 1, which mean the sum is $n-1$. Contradiction.

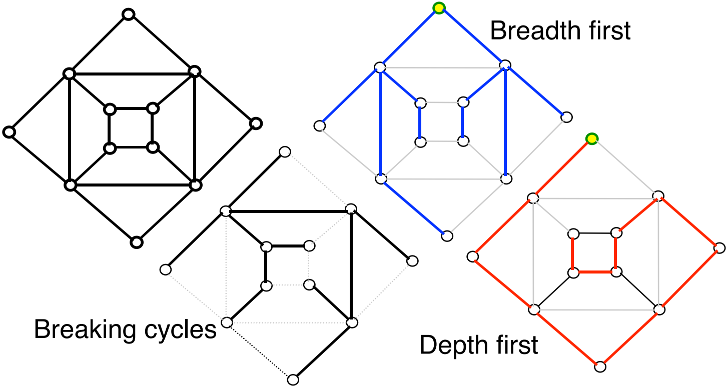
$\square$

Spanning Tree

- If R is a symmetric, connected relation on A, a tree T on A is a **spanning tree** of R if T is a tree with exactly the same vertices as R.
- An undirected spanning tree is the symmetric closure of a spanning tree.
- Note that an undirected spanning tree can always obtained by remove some edges from a symmetric, connected relation R.

Spanning Tree: Examples

Different spanning tree are obtained from a symmetric, connected relation:



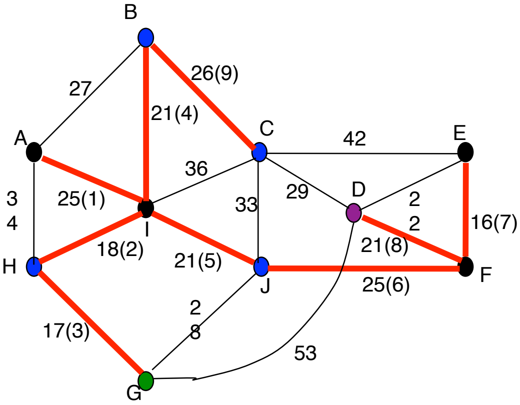
Generic Algorithm for MST Problem

Input: G: a connected, undirected graph
w: a function from E_G to the set of real number

Generic-MST(G,w)

```
A←0
while A does not form a spanning tree
    do find an edge (u,v) that is safe for A
    A←A∪{(u,v)}
return A
```

Output: a minimal spanning tree of G

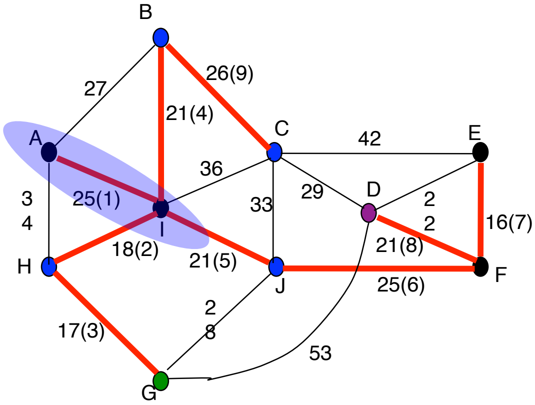


<u>Step 1</u>: $V = \{A\}, E = \{\}$

<u>Step 2</u>: Select the nearest neighbour of V, u, add the edge connecting u and some vertex in V into E

<u>Step 3</u>: Repeat step 2 until E contains $n-1$ edges

<u>End of Algorithm</u>



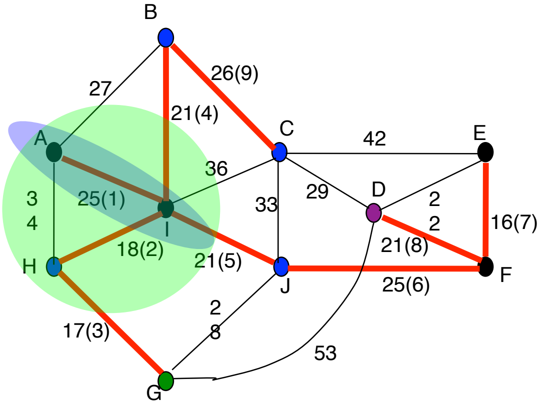
Step 1: $V = \{A\}, E = \{\}$

Step 2: Select the nearest neighbour of V, u, add the edge connecting u and some vertex in V into E

Step 3: Repeat step 2 until E contains $n-1$ edges

End of Algorithm

Prim's Algorithm for MST



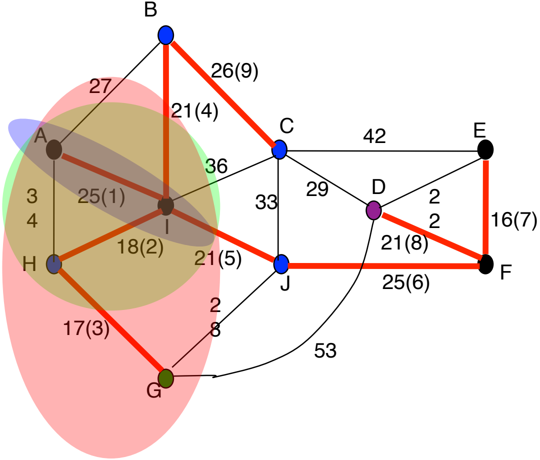
Step 1: $V = \{A\}, E = \{\}$

Step 2: Select the nearest neighbour of V, u, add the edge connecting u and some vertex in V into E

Step 3: Repeat step 2 until E contains $n-1$ edges

End of Algorithm

Prim's Algorithm for MST



Step 1: $V = \{A\}, E = \{\}$

Step 2: Select the nearest neighbour of V, u, add the edge connecting u and some vertex in V into E

Step 3: Repeat step 2 until E contains $n - 1$ edges

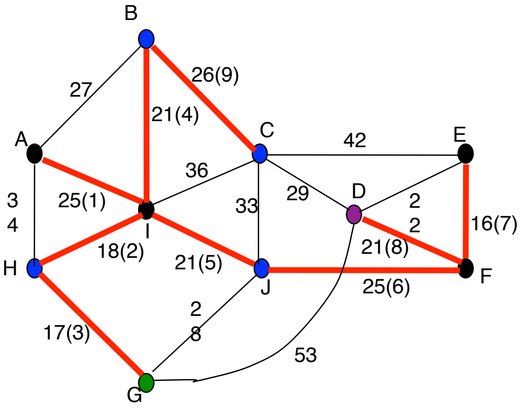
End of Algorithm

Correctness of Prim's Algorithm

Let T be the output of Prim's algorithm, and T contains edge $t_1 t_2 \cdots t_{n-1}$, as the order they are selected. $T_i = \{t_1, t_2, \cdots, t_i\}$ for $1 \leq i \leq n-1$, and $T_0 = \emptyset$. It can be proved that each T_i is contained in a MST:

- Assume that T_k is contained in a MST T', then $\{t_1, t_2, \cdots, t_i\} \subseteq T'$.
- If $t_{k+1} \notin T'$ then $T' \cup \{t_{k+1}\}$ contains a cycle, which cannot wholly be in T_k. (let the circle be $s_1 s_2 \cdots s_r t_{k+1}$.)
  Let s_l be the edge with smallest index l that is not in T_k. Exactly one of the vertices of s_l must be in T_k, which means that when t_{k+1} was chosen, s_l available as well. So, t_{k+1} has no larger weight than s_l. So, $(T' - \{s_l\}) \cup \{t_{k+1}\}$ is a MST containing T_{k+1}.

Kruskal's Algorithm for MST



Step 1: $E = \{\}$

Step 2: Select the edge with the least weight, and not making a cycle with members of E

Step 3: Repeat step 2 until E contains $n - 1$ edges

End of Algorithm

Proof of Kruskal Algorithm

Obviously, T is an undirected tree.

Suppose that T is not minimal. According to the ordering of adding edges in T, T contains edges $e_1, e_2, \cdots, e_{k-1}, e_k, \cdots, e_{n-1}$. Let T' is a minimal spanning tree which has most consecutive common edges from the beginning with T. And let e_k is the first edge not in T'. So, $T' + e_k$ contains a cycle, let $e_{k'}$ is on the cycle, but not in T, then $T^* = T' - \{e_{k'}\} \cup e_k$ is also a spanning tree, and we have $w(T^*) = w(T') - w(e_{k'}) + w(e_k)$. According to the criteria to select the edges, $w(e_{k'}) \geq w(e_k)$, $\therefore$ $w(T^*) \leq w(T')$, which means that T^* is also a minimal spanning tree, and with more common consecutive edges with T. Contradiction.

Kruskal Algorithm – Implementation

KRUSKAL(V, E, n)

<sort E by non-decreasing weight>; → $O(m\log m)$

for each $v \in V$ **do** $MAKESET(\{v\})$ **end for**; → $\Theta(n)$

$T = \{\}$;

while $|T| < n-1$ **do**

<check the next edge (x,y) in E>; → $\Theta(n)$

if $FIND(x) \neq FIND(y)$ **then**

$T = T \cup \{(x, y)\}$;

$UNION(x, y)$; → $O(m\log^* n)$

n-1 UNION's and 2m FIND's at most

end if

end while

return T;

The algorithm finishes in $O(m\log m)$ time.
Or, $O(n^2\log n)$; $O(n\log n)$ for planar graph.

Home Assignments

To be checked

- Ex.7.1: 18-22, 24,29, 32-34
- Ex.7.2: 7, 13,18, 25-27
- Ex.7.3: 10, 15, 19-22, 25, 33, 37-38
- Ex.7.4: 16-17, 19, 21, 26
- Ex.7.5: 6, 9, 11, 14, 18, 23

The End