Chapter 10
Trees

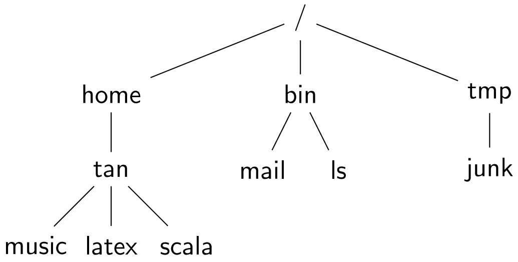
Discrete Structures for Computing on 27 May 2014

Huynh Tuong Nguyen, Tran Vinh Tan
Faculty of Computer Science and Engineering
University of Technology - VNUHCM

Contents

Introduction

- Very useful in computer science: search algorithm, game winning strategy, decision making, sorting, . . .
- Other disciplines: chemical compounds, family trees, organizational tree, . . .



Tree

Definition

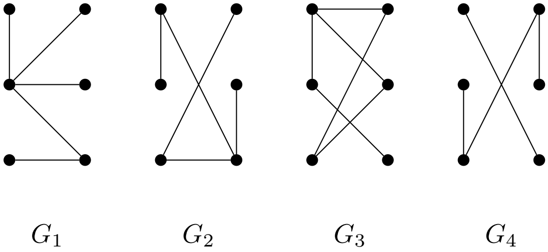
A tree (*cây*) is a connected undirected graph with no simple circuits. Consequently, a tree must be a simple graph.

Contents

Tree

Definition

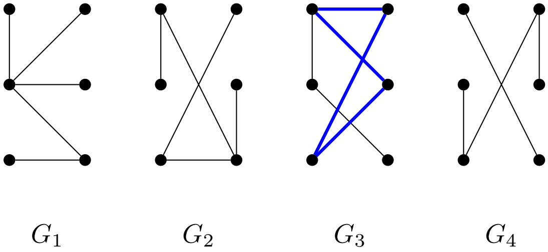
A tree (*cây*) is a connected undirected graph with no simple circuits. Consequently, a tree must be a simple graph.



Tree

Definition

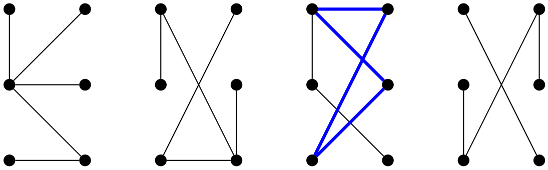
A tree (*cây*) is a connected undirected graph with no simple circuits. Consequently, a tree must be a simple graph.



Tree

Definition

A tree (*cây*) is a connected undirected graph with no simple circuits. Consequently, a tree must be a simple graph.



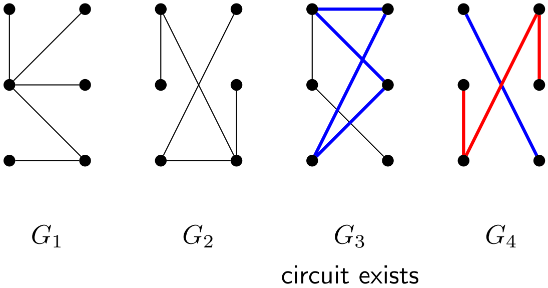
G_1 G_2 G_3 G_4

circuit exists

Tree

Definition

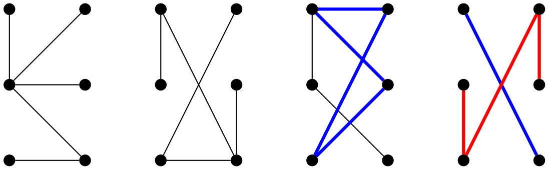
A tree (*cây*) is a connected undirected graph with no simple circuits. Consequently, a tree must be a simple graph.



Tree

Definition

A tree (*cây*) is a connected undirected graph with no simple circuits. Consequently, a tree must be a simple graph.



G_1 G_2 G_3 G_4

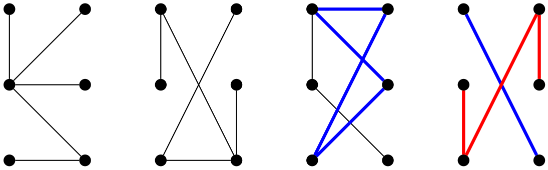
circuit exists

not connected

Tree

Definition

A tree (*cây*) is a connected undirected graph with no simple circuits. Consequently, a tree must be a simple graph.



G_1 G_2 G_3 G_4

circuit exists

not connected

Definition

Graphs containing no simple circuits that are not necessarily connected is forest (*rừng*), in which each connected component is a tree.

Rooted Trees

Definition

A rooted tree (*cây có gốc*) is a tree in which:

- One vertex has been designated as the root and
- Every edge is directed away from the root

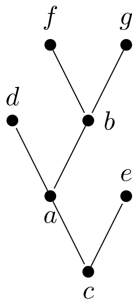
Contents

Rooted Trees

Definition

A rooted tree (*cây có gốc*) is a tree in which:

- One vertex has been designated as the root and
- Every edge is directed away from the root

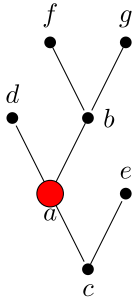


Rooted Trees

Definition

A rooted tree (*cây có gốc*) is a tree in which:

- One vertex has been designated as the root and
- Every edge is directed away from the root

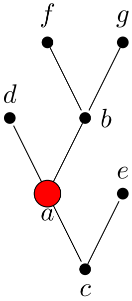


Rooted Trees

Definition

A rooted tree (*cây có gốc*) is a tree in which:

- One vertex has been designated as the root and
- Every edge is directed away from the root

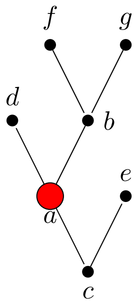


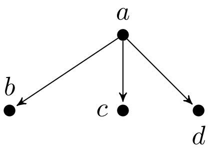
Rooted Trees

Definition

A rooted tree (*cây có gốc*) is a tree in which:

- One vertex has been designated as the root and
- Every edge is directed away from the root



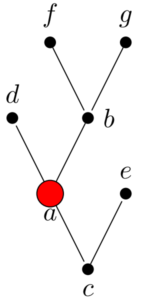


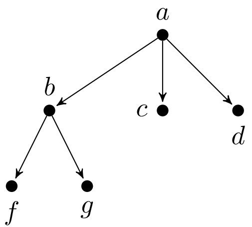
Rooted Trees

Definition

A rooted tree (*cây có gốc*) is a tree in which:

- One vertex has been designated as the root and
- Every edge is directed away from the root



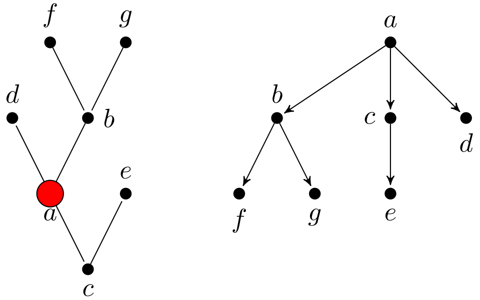


Rooted Trees

Definition

A rooted tree (*cây có gốc*) is a tree in which:

- One vertex has been designated as the root and
- Every edge is directed away from the root



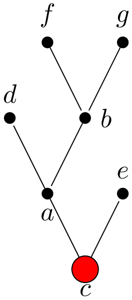
Contents

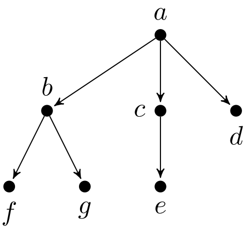
Rooted Trees

Definition

A rooted tree (*cây có gốc*) is a tree in which:

- One vertex has been designated as the root and
- Every edge is directed away from the root



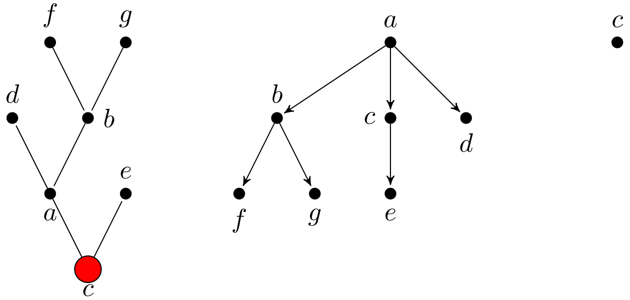


Rooted Trees

Definition

A rooted tree (*cây có gốc*) is a tree in which:

- One vertex has been designated as the root and
- Every edge is directed away from the root

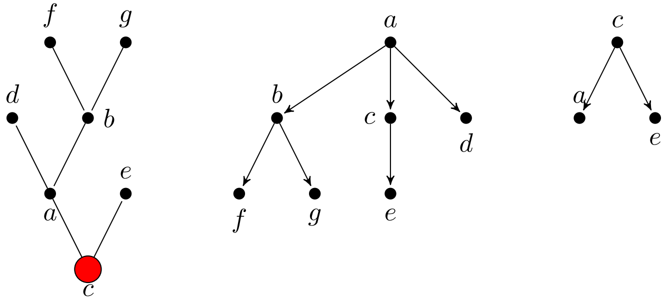


Rooted Trees

Definition

A rooted tree (*cây có gốc*) is a tree in which:

- One vertex has been designated as the root and
- Every edge is directed away from the root

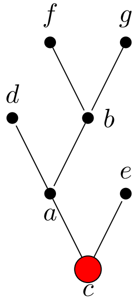


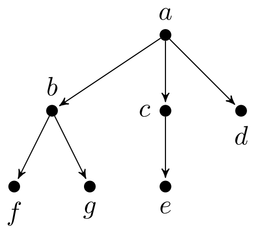
Rooted Trees

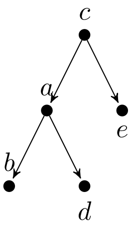
Definition

A rooted tree (*cây có gốc*) is a tree in which:

- One vertex has been designated as the root and
- Every edge is directed away from the root





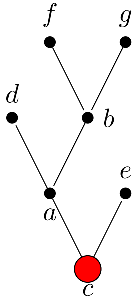


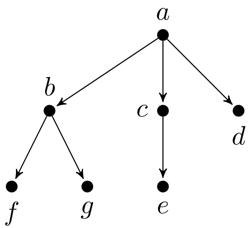
Rooted Trees

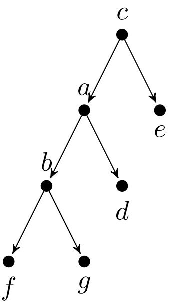
Definition

A rooted tree (*cây có gốc*) is a tree in which:

- One vertex has been designated as the root and
- Every edge is directed away from the root





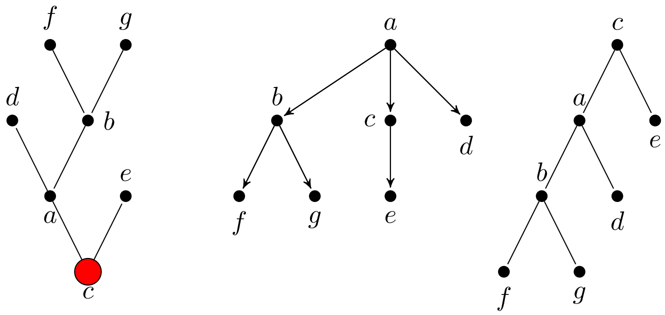


Rooted Trees

Definition

A rooted tree (*cây có gốc*) is a tree in which:

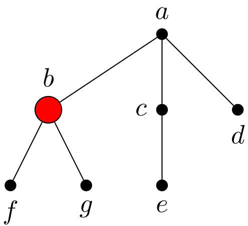
- One vertex has been designated as the root and
- Every edge is directed away from the root



Terminology

Definition

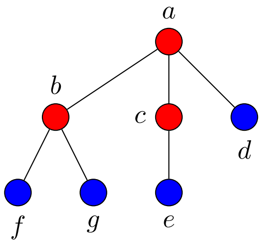
- parent (*cha*) of v is the unique u such that there is a directed edge from u to v
- when u is the parent of v, v is called a child (*con*) of u
- vertices with the same parent are called siblings (*anh em*)
- the ancestors (*tổ tiên*) of a vertex are the vertices in the path from the root to this vertex (excluding the vertex itself)
- descendants (*con cháu*) of a vertex v are those vertices that have v as an ancestor



Terminology

Definition

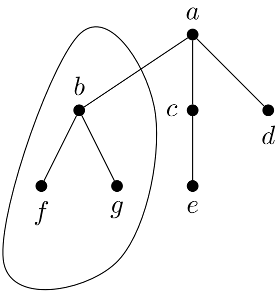
- a vertex of a tree is called a leaf (*lá*) if it has no children
- vertices that have children are called internal vertices (*đỉnh trong*)



Terminology

Definition

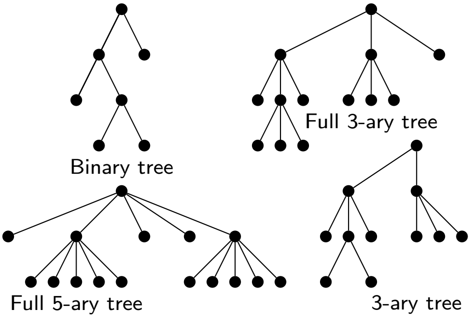
If a is a vertex in a tree, the subtree (*cây con*) with a as its root is the subgraph of the tree consisting of a and its descendants and all edges incident to these descendants.



m-ary tree

Definition

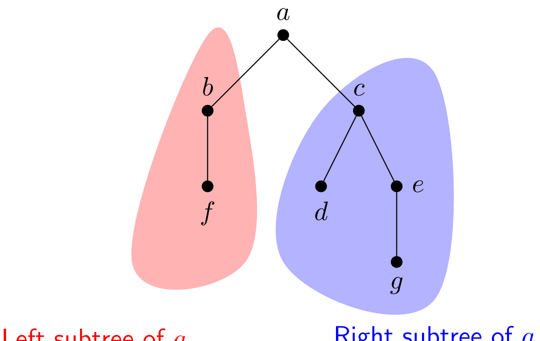
- m-ary tree (*cây m-phân*): at most m children on each internal vertex of a rooted tree.
- **full** m-ary tree (*cây m-phân đầy đủ*): every internal vertex has exactly m children.
- An m-ary tree with $m = 2$ is called a binary tree (*cây nhị phân*).



Ordered Rooted Trees

Definition

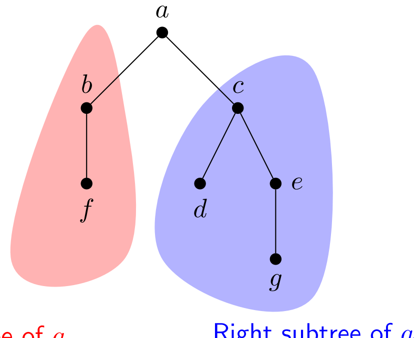
- An ordered rooted tree (*cây có gốc có thứ tự*) is a rooted tree where the children of each internal vertex are ordered (e.g. in order from left to right).



Ordered Rooted Trees

Definition

- An ordered rooted tree (*cây có gốc có thứ tự*) is a rooted tree where the children of each internal vertex are ordered (e.g. in order from left to right).
- In an ordered binary tree (*cây nhị phân có thứ tự*), if an internal vertex has two children, the first child is called the left child (*con bên trái*) and the second is called the right child (*con bên phải*).



Properties & Theorems

Theorem

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

Theorem

A full m-ary tree with i internal vertices contains $n = mi + 1$ vertices.

Properties & Theorems

Theorem

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

Theorem

A full m-ary tree with i internal vertices contains $n = mi + 1$ vertices.

(i) n *vertices has* $(n-1)/m$ *internal vertices and* $[(m-1)n+1]/m$ *leaves*

(ii) i *internal vertices has* $n = mi + 1$ *vertices and* $(m-1)i+1$ *leaves*

(iii) ℓ *leaves has* $n = (m\ell - 1)/(m-1)$ *vertices and* $(\ell-1)/(m-1)$ *internal vertices*

Example

Example (Chain Letter Game)

- Each person who receives the letter is asked to send it on to four other peoples.
- Some peoples do this, but others do not send any letters.
- How many people have seen the letter, including the first person, if no one receives more than one letter and if the chain letter ends after there have been 100 people who read it but did not send it out ?
- How many people sent out the letter?

Example

Example (Chain Letter Game)

- Each person who receives the letter is asked to send it on to four other peoples.
- Some peoples do this, but others do not send any letters.
- How many people have seen the letter, including the first person, if no one receives more than one letter and if the chain letter ends after there have been 100 people who read it but did not send it out ?
- How many people sent out the letter?

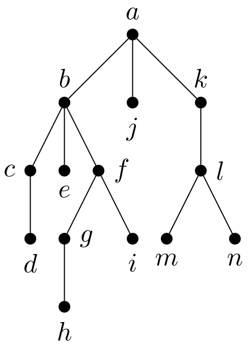
Solution

- *Using 4-ary tree with 100 leaves corresponding to 100 persons who did not send out the letter.*
- $\Longrightarrow n = (ml - 1)/(m - 1) = (4 \times 100 - 1)/(4 - 1) = 133$ *vertices and* $i = n - l = 133 - 100 = 33$ *internal vertices.*

Level and Height

Definition

- The level (*mức*) of a vertex v in a rooted tree is the length of the unique path from the root to this vertex.
- The level of the root is defined to be zero.
- The height (*độ cao*) of a rooted tree is the maximum of the levels of vertices (i.e. the length of the longest path from the root to any vertex).



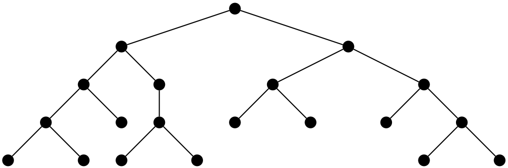
Example

- Level of root $a = 0$, $b, j, k = 1$ and $c, e, f, l = 2 \ldots$
- Because the largest level of any vertex is 4, this tree has height 4.

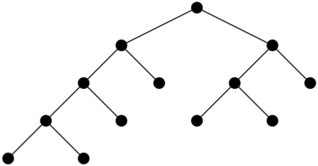
Balanced m-ary Trees

Definition

A rooted m-ary tree of height h is balanced (*cân đối*) if all leaves are at levels h or $h-1$.



T_1



T_2

Balanced m-ary Tree

Theorem

There are at most m^h leaves in an m-ary tree of height h.

Balanced m-ary Tree

Theorem

There are at most m^h leaves in an m-ary tree of height h.

It can be proved by using mathematical induction on the height.

Balanced m-ary Tree

Theorem

There are at most m^h leaves in an m-ary tree of height h.

It can be proved by using mathematical induction on the height.

Corollary

- *If an m-ary tree of height h has ℓ leaves, then $h \geq \lceil \log_m \ell \rceil$.*
- *If the m-ary tree is full and balanced, then $h = \lceil \log_m \ell \rceil$.*

Exercise

Exercise (Chess tournament)

Suppose 1000 people enter a chess tournament. Use a rooted tree model of the tournament to determine how many games must be played to determine a champion. If a player is eliminated after one loss and games are played until only one entrant has not lost. (Assume there are no ties)

Exercise (Isomorphic)

How many different isomers (*đồng phân*) do the following saturated hydrocarbons have ?

- C_3H_8
- C_5H_{12}
- C_6H_{14}

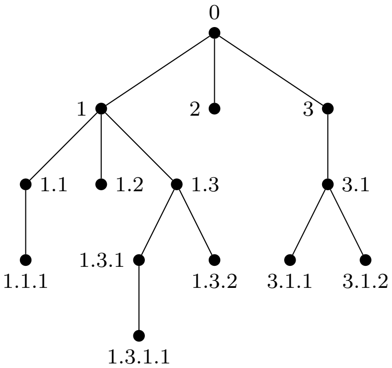
Question

Exercise

- How many vertices and how many leaves does a complete m-ary tree of height h have?
- Show that a full m-ary balanced tree (*cây m-phân hoàn hảo*) of height h has more than m^{h-1} leaves.
- How many edges are there in a forest of t trees containing a total of n vertices?

Labeling Ordered Rooted Trees

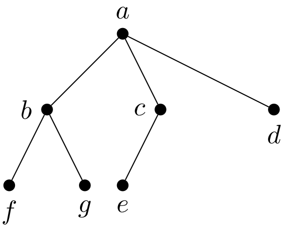
- Ordered rooted trees are often used to store information.
- Need a procedure for visiting each vertex of an ordered rooted tree to access data.
- Ordering and labeling the vertices is important to traverse them in any procedure
- Universal address system (*hệ địa chỉ phổ dụng*)
  $0 < 1 < 1.1 < 1.1.1 < 1.2 < 1.3 < \ldots < 2 < 3 < 3.1 < \ldots$



Traversal Algorithms (Thuật toán duyệt cây)

Preorder Traversal (duyệt tiền thứ tự)

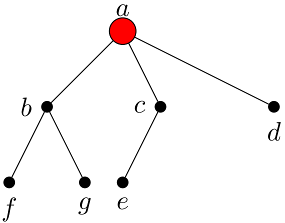
procedure *preorder*(T: ordered rooted tree)
    r := root of T
    print r
    for each child c of r from left to right
        $T(c)$:= subtree with c as its root
        preorder($T(c)$)



Traversal Algorithms (Thuật toán duyệt cây)

Preorder Traversal (duyệt tiền thứ tự)

procedure *preorder*(T: ordered rooted tree)
  $r :=$ root of T
  print r
  for each child c of r from left to right
    $T(c) :=$ subtree with c as its root
    preorder$(T(c))$

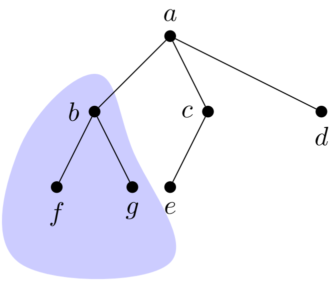


a

Traversal Algorithms (Thuật toán duyệt cây)

Preorder Traversal (duyệt tiền thứ tự)

procedure *preorder*(T: ordered rooted tree)
  r := root of T
  print r
  for each child c of r from left to right
    $T(c)$:= subtree with c as its root
    preorder($T(c)$)

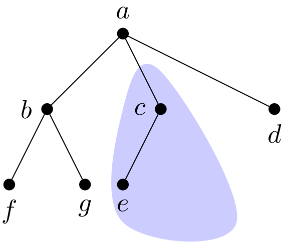


a

Traversal Algorithms (Thuật toán duyệt cây)

Preorder Traversal (duyệt tiền thứ tự)

procedure *preorder*(T: ordered rooted tree)
    r := root of T
    print r
    for each child c of r from left to right
        $T(c)$:= subtree with c as its root
        preorder$(T(c))$

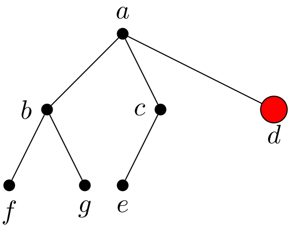


a

Traversal Algorithms (Thuật toán duyệt cây)

Preorder Traversal (duyệt tiền thứ tự)

procedure *preorder*(T: ordered rooted tree)
    r := root of T
    print r
    for each child c of r from left to right
        $T(c)$:= subtree with c as its root
        preorder$(T(c))$

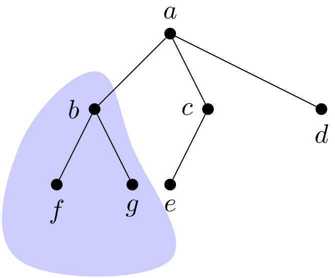


a d

Traversal Algorithms (Thuật toán duyệt cây)

Preorder Traversal (duyệt tiền thứ tự)

procedure *preorder*(T: ordered rooted tree)
    r := root of T
    print r
    for each child c of r from left to right
        $T(c)$:= subtree with c as its root
        preorder$(T(c))$

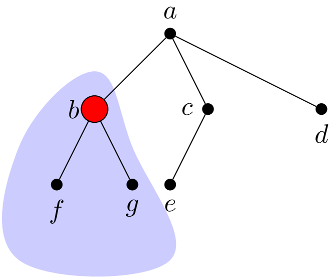


a   d

Traversal Algorithms (Thuật toán duyệt cây)

Preorder Traversal (duyệt tiền thứ tự)

procedure *preorder*(T: ordered rooted tree)
    r := root of T
    print r
    for each child c of r from left to right
        $T(c)$:= subtree with c as its root
        preorder$(T(c))$

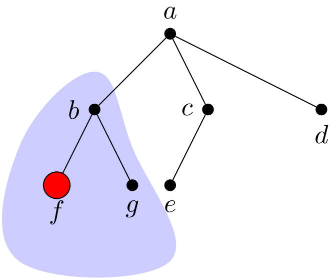


a b d

Traversal Algorithms (Thuật toán duyệt cây)

Preorder Traversal (duyệt tiền thứ tự)

procedure *preorder*(T: ordered rooted tree)
    r := root of T
    print r
    for each child c of r from left to right
        $T(c)$:= subtree with c as its root
        preorder$(T(c))$

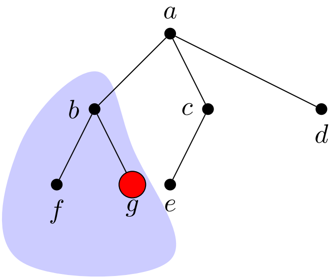


a b f d

Traversal Algorithms (Thuật toán duyệt cây)

Preorder Traversal (duyệt tiền thứ tự)

procedure *preorder*(T: ordered rooted tree)
  r := root of T
  print r
  for each child c of r from left to right
    $T(c)$:= subtree with c as its root
    preorder($T(c)$)

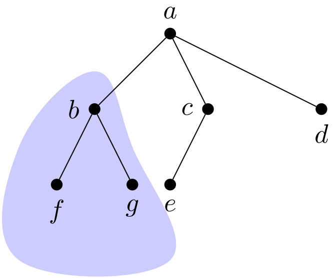


a b f g d

Traversal Algorithms (Thuật toán duyệt cây)

Preorder Traversal (duyệt tiền thứ tự)

procedure *preorder*(T: ordered rooted tree)
    r := root of T
    print r
    for each child c of r from left to right
        $T(c)$:= subtree with c as its root
        preorder($T(c)$)

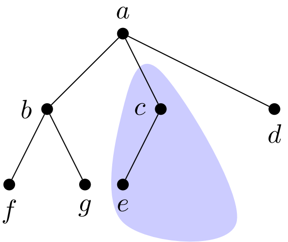


a b f g d

Traversal Algorithms (Thuật toán duyệt cây)

Preorder Traversal (duyệt tiền thứ tự)

procedure *preorder*(T: ordered rooted tree)
  r := root of T
  print r
  for each child c of r from left to right
    $T(c)$:= subtree with c as its root
    preorder$(T(c))$

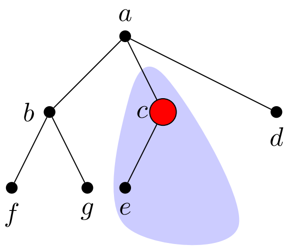


a b f g d

Traversal Algorithms (Thuật toán duyệt cây)

Preorder Traversal (duyệt tiền thứ tự)

procedure *preorder*(T: ordered rooted tree)
    r := root of T
    print r
    for each child c of r from left to right
        $T(c)$:= subtree with c as its root
        preorder($T(c)$)

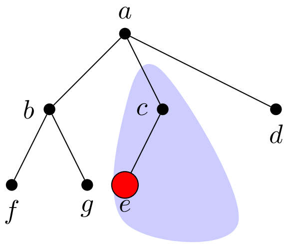


a b f g c d

Traversal Algorithms (Thuật toán duyệt cây)

Preorder Traversal (duyệt tiền thứ tự)

procedure *preorder*(T: ordered rooted tree)
    r := root of T
    print r
    for each child c of r from left to right
        $T(c)$:= subtree with c as its root
        preorder($T(c)$)

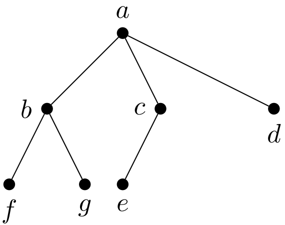


a b f g c e d

Traversal Algorithms

Inorder Traversal (Duyệt trung thứ tự)

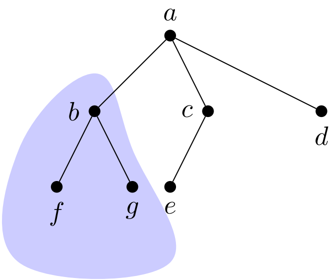
Suppose a tree T with root r. If T consists only of r, then r is inorder traversal of T. Otherwise, suppose r has subtrees T_1, T_2, $\ldots$, T_n from left to right, inorder traversal:
$T_1 \to r \to T_2 \to \ldots \to T_n$.



Traversal Algorithms

Inorder Traversal (Duyệt trung thứ tự)

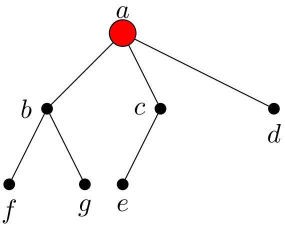
Suppose a tree T with root r. If T consists only of r, then r is inorder traversal of T. Otherwise, suppose r has subtrees T_1, T_2, $\ldots$, T_n from left to right, inorder traversal:
$T_1 \rightarrow r \rightarrow T_2 \rightarrow \ldots \rightarrow T_n$.



Traversal Algorithms

Inorder Traversal (Duyệt trung thứ tự)

Suppose a tree T with root r. If T consists only of r, then r is inorder traversal of T. Otherwise, suppose r has subtrees T_1, T_2, $\ldots$, T_n from left to right, inorder traversal:
$T_1 \rightarrow r \rightarrow T_2 \rightarrow \ldots \rightarrow T_n$.

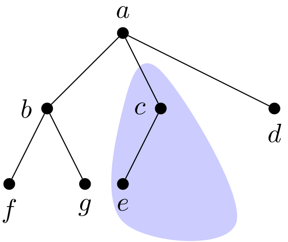


a

Traversal Algorithms

Inorder Traversal (Duyệt trung thứ tự)

Suppose a tree T with root r. If T consists only of r, then r is inorder traversal of T. Otherwise, suppose r has subtrees T_1, T_2, $\ldots$, T_n from left to right, inorder traversal:
$T_1 \to r \to T_2 \to \ldots \to T_n$.

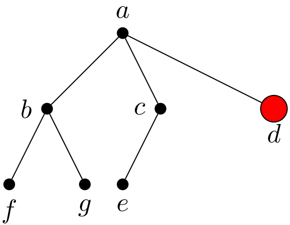


a

Traversal Algorithms

Inorder Traversal (Duyệt trung thứ tự)

Suppose a tree T with root r. If T consists only of r, then r is inorder traversal of T. Otherwise, suppose r has subtrees T_1, T_2, $\ldots$, T_n from left to right, inorder traversal: $T_1 \rightarrow r \rightarrow T_2 \rightarrow \ldots \rightarrow T_n$.

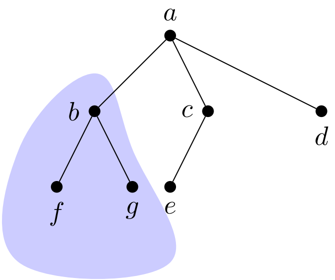


a d

Traversal Algorithms

Inorder Traversal (Duyệt trung thứ tự)

Suppose a tree T with root r. If T consists only of r, then r is inorder traversal of T. Otherwise, suppose r has subtrees T_1, T_2, $\ldots$, T_n from left to right, inorder traversal:
$T_1 \to r \to T_2 \to \ldots \to T_n$.

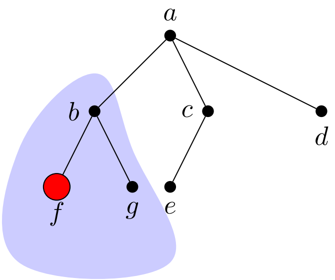


a d

Traversal Algorithms

Inorder Traversal (Duyệt trung thứ tự)

Suppose a tree T with root r. If T consists only of r, then r is inorder traversal of T. Otherwise, suppose r has subtrees T_1, T_2, $\ldots$, T_n from left to right, inorder traversal:
$T_1 \rightarrow r \rightarrow T_2 \rightarrow \ldots \rightarrow T_n$.

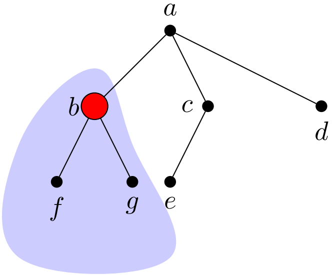


f a d

Traversal Algorithms

Inorder Traversal (Duyệt trung thứ tự)

Suppose a tree T with root r. If T consists only of r, then r is inorder traversal of T. Otherwise, suppose r has subtrees T_1, T_2, $\ldots$, T_n from left to right, inorder traversal: $T_1 \to r \to T_2 \to \ldots \to T_n$.

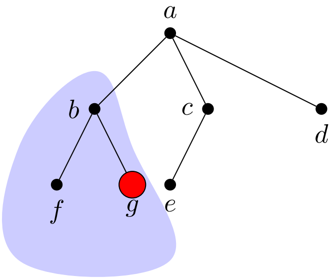


f b a d

Traversal Algorithms

Inorder Traversal (Duyệt trung thứ tự)

Suppose a tree T with root r. If T consists only of r, then r is inorder traversal of T. Otherwise, suppose r has subtrees T_1, T_2, ..., T_n from left to right, inorder traversal:
$T_1 \to r \to T_2 \to \ldots \to T_n$.

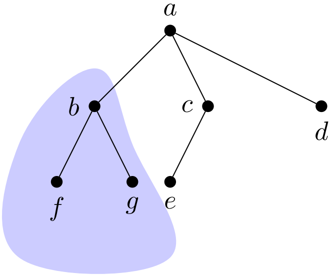


f b g a d

Traversal Algorithms

Inorder Traversal (Duyệt trung thứ tự)

Suppose a tree T with root r. If T consists only of r, then r is inorder traversal of T. Otherwise, suppose r has subtrees T_1, T_2, $\ldots$, T_n from left to right, inorder traversal:
$T_1 \to r \to T_2 \to \ldots \to T_n$.

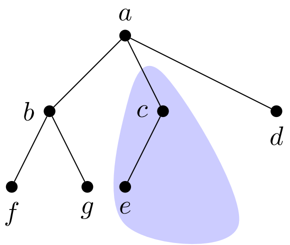


f b g a d

Traversal Algorithms

Inorder Traversal (Duyệt trung thứ tự)

Suppose a tree T with root r. If T consists only of r, then r is inorder traversal of T. Otherwise, suppose r has subtrees T_1, T_2, $\ldots$, T_n from left to right, inorder traversal:
$T_1 \rightarrow r \rightarrow T_2 \rightarrow \ldots \rightarrow T_n$.

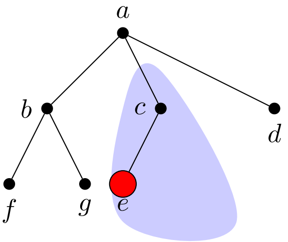


f b g a d

Traversal Algorithms

Inorder Traversal (Duyệt trung thứ tự)

Suppose a tree T with root r. If T consists only of r, then r is inorder traversal of T. Otherwise, suppose r has subtrees T_1, T_2, $\ldots$, T_n from left to right, inorder traversal:
$T_1 \rightarrow r \rightarrow T_2 \rightarrow \ldots \rightarrow T_n$.

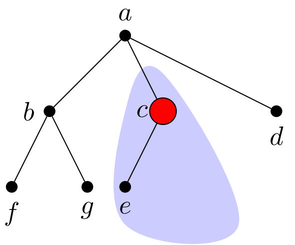


f b g a e d

Traversal Algorithms

Inorder Traversal (Duyệt trung thứ tự)

Suppose a tree T with root r. If T consists only of r, then r is inorder traversal of T. Otherwise, suppose r has subtrees T_1, T_2, $\ldots$, T_n from left to right, inorder traversal:
$T_1 \to r \to T_2 \to \ldots \to T_n$.

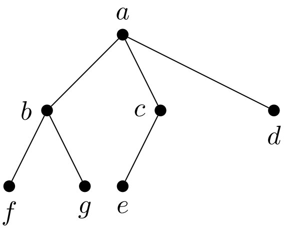


f b g a e c d

Traversal Algorithms

Postorder Traversal (Duyệt hậu thứ tự)

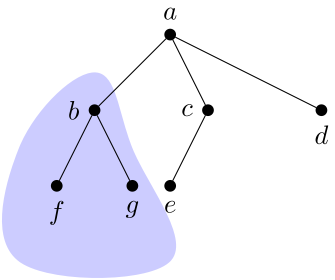
procedure *postorder*(T: ordered rooted tree)
  r := root of T
  for each child c of r from left to right
    $T(c)$:= subtree with c as its root
    postorder($T(c)$)
  print r



Traversal Algorithms

Postorder Traversal (Duyệt hậu thứ tự)

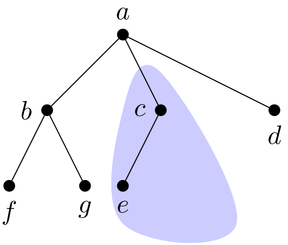
procedure *postorder*(T: ordered rooted tree)
    r := root of T
    for each child c of r from left to right
        $T(c)$:= subtree with c as its root
        postorder($T(c)$)
    print r



Traversal Algorithms

Postorder Traversal (Duyệt hậu thứ tự)

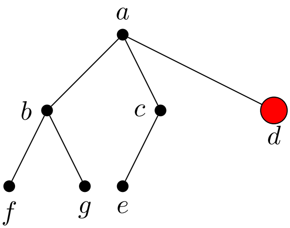
procedure *postorder*(T: ordered rooted tree)
    $r :=$ root of T
    for each child c of r from left to right
        $T(c) :=$ subtree with c as its root
        postorder$(T(c))$
    print r



Traversal Algorithms

Postorder Traversal (Duyệt hậu thứ tự)

procedure *postorder*(T: ordered rooted tree)
    r := root of T
    for each child c of r from left to right
        $T(c)$:= subtree with c as its root
        postorder($T(c)$)
    print r

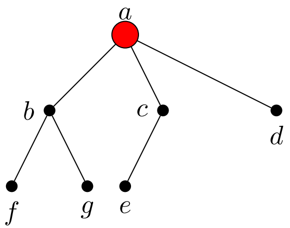


d

Traversal Algorithms

Postorder Traversal (Duyệt hậu thứ tự)

procedure *postorder*(T: ordered rooted tree)
  r := root of T
  for each child c of r from left to right
    $T(c)$:= subtree with c as its root
    postorder($T(c)$)
  print r

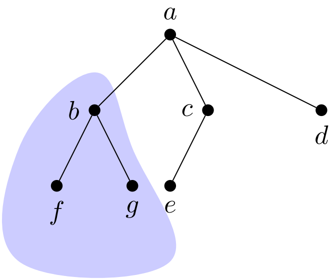


d a

Traversal Algorithms

Postorder Traversal (Duyệt hậu thứ tự)

procedure *postorder*(T: ordered rooted tree)
    r := root of T
    for each child c of r from left to right
        $T(c)$:= subtree with c as its root
        postorder($T(c)$)
    print r

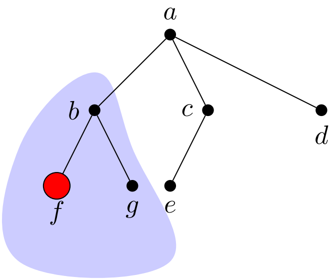


d a

Traversal Algorithms

Postorder Traversal (Duyệt hậu thứ tự)

procedure *postorder*(T: ordered rooted tree)
    r := root of T
    for each child c of r from left to right
        $T(c)$:= subtree with c as its root
        postorder($T(c)$)
    print r

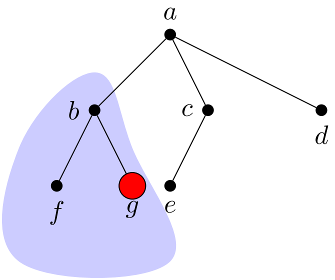


f d a

Traversal Algorithms

Postorder Traversal (Duyệt hậu thứ tự)

procedure *postorder*(T: ordered rooted tree)
    $r :=$ root of T
    for each child c of r from left to right
        $T(c) :=$ subtree with c as its root
        postorder$(T(c))$
    print r

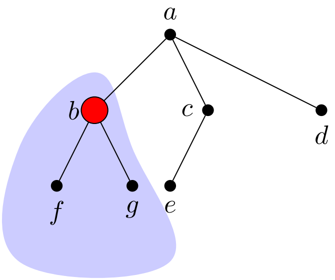


f g d a

Traversal Algorithms

Postorder Traversal (Duyệt hậu thứ tự)

procedure *postorder*(T: ordered rooted tree)
  r := root of T
  for each child c of r from left to right
    $T(c)$:= subtree with c as its root
    postorder($T(c)$)
  print r

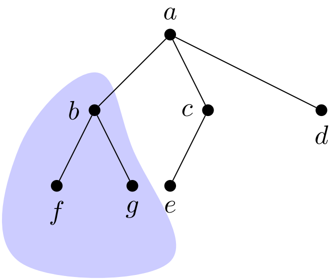


f g b d a

Traversal Algorithms

Postorder Traversal (Duyệt hậu thứ tự)

procedure *postorder*(T: ordered rooted tree)
    $r :=$ root of T
    for each child c of r from left to right
        $T(c) :=$ subtree with c as its root
        postorder$(T(c))$
    print r

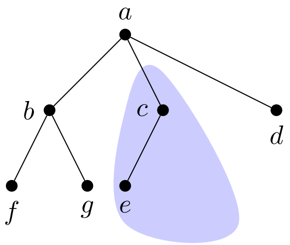


f g b d a

Traversal Algorithms

Postorder Traversal (Duyệt hậu thứ tự)

procedure *postorder*(T: ordered rooted tree)
  r := root of T
  for each child c of r from left to right
    $T(c)$:= subtree with c as its root
    postorder$(T(c))$
  print r

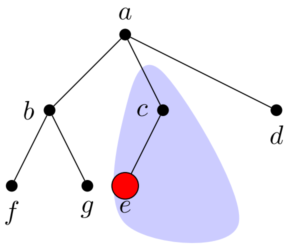


f g b d a

Traversal Algorithms

Postorder Traversal (Duyệt hậu thứ tự)

procedure *postorder*(T: ordered rooted tree)
    r := root of T
    for each child c of r from left to right
        $T(c)$:= subtree with c as its root
        postorder($T(c)$)
    print r

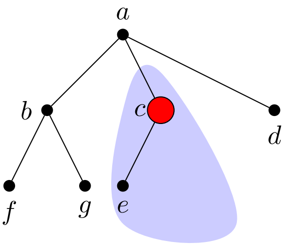


f g b e d a

Traversal Algorithms

Postorder Traversal (Duyệt hậu thứ tự)

procedure *postorder*(T: ordered rooted tree)
    r := root of T
    for each child c of r from left to right
        $T(c)$:= subtree with c as its root
        postorder($T(c)$)
    print r



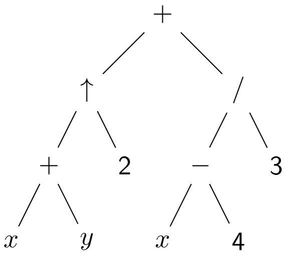
f g b e c d a

Infix, Prefix and Postfix Notations

- Infix (*trung tố*):
  $((x+y)\uparrow 2)+((x-4)/3)$

- Prefix (*tiền tố*):
  $+\ \uparrow\ +\ x\ y\ 2\ /\ -\ x\ 4\ 3$

- Postfix (*hậu tố*):
  $x\ y\ +\ 2\ \uparrow\ x\ 4\ -\ 3\ /\ +$



Exercise

Exercise

Find the ordered rooted tree representing

$$(\neg(p \wedge q) \vee (\neg q \wedge r)) \rightarrow (\neg p \vee \neg r)$$

Then use this rooted tree to find the prefix, postfix and infix forms of this expression

Exercise

Exercise

Find the ordered rooted tree representing

$$(\neg(p \wedge q) \vee (\neg q \wedge r)) \rightarrow (\neg p \vee \neg r)$$

Then use this rooted tree to find the prefix, postfix and infix forms of this expression

Solution

- *Constructing the rooted tree from the bottom up*
- *Preorder traversal creates prefix notation*
  $\rightarrow \vee \neg \wedge p\ q \vee \neg q\ r \vee \neg p\ r$
- *Postorder traversal creates postfix notation*
  $p\ q \wedge \neg \vee q \neg r \wedge p \neg r \vee \rightarrow$
- *Inorder traversal creates infix notation (with parentheses)*
  $p\ q \neg \vee q \neg \wedge r \rightarrow p \neg \vee r$

Exercise

Exercise

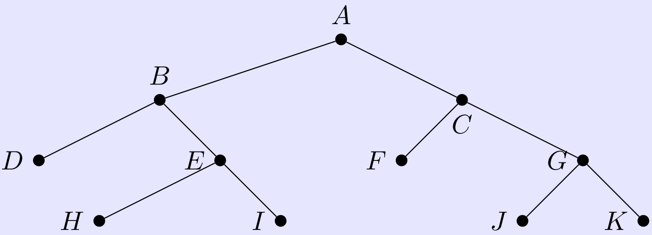
Find postorder traversal of a binary tree with inorder D B H E I A F C J G K and preorder A B D E H I C F G J K.

Exercise

Exercise

Find postorder traversal of a binary tree with inorder D B H E I A F C J G K and preorder A B D E H I C F G J K.

Solution



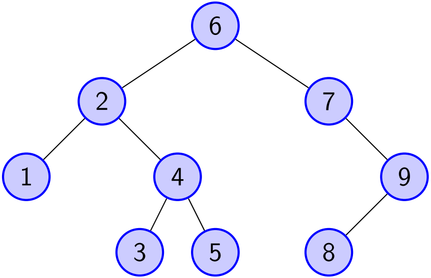
Post order: D H I E B F J K G C A.

Binary Search Trees

Definition

Binary search tree (*cây tìm kiếm nhị phân* - BST) is a binary tree in which the assigned key of a vertex is:

- larger than the keys of all vertices in its left subtree, and
- smaller than the keys of all vertices in its right subtree.



Adding and Locating an Item in BST

Example

Form a BST for the words *mathematics*, *physics*, *geography*, *zoology*, *meteorology*, *geology*, *psychology*, *chemistry* using alphabetical order.

Adding and Locating an Item in BST

Example

Form a BST for the words *mathematics*, *physics*, *geography*, *zoology*, *meteorology*, *geology*, *psychology*, *chemistry* using alphabetical order.

Adding and Locating an Item in BST

Example

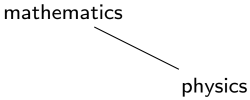
Form a BST for the words *mathematics*, *physics*, *geography*, *zoology*, *meteorology*, *geology*, *psychology*, *chemistry* using alphabetical order.

mathematics

Adding and Locating an Item in BST

Example

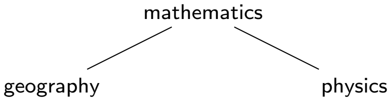
Form a BST for the words *mathematics*, *physics*, *geography*, *zoology*, *meteorology*, *geology*, *psychology*, *chemistry* using alphabetical order.



Adding and Locating an Item in BST

Example

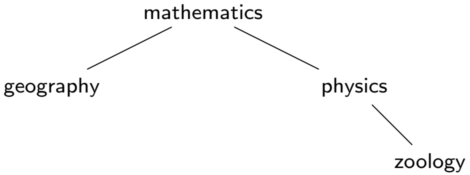
Form a BST for the words *mathematics*, *physics*, *geography*, *zoology*, *meteorology*, *geology*, *psychology*, *chemistry* using alphabetical order.



Adding and Locating an Item in BST

Example

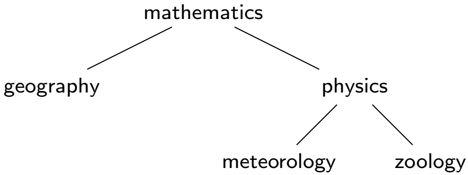
Form a BST for the words *mathematics*, *physics*, *geography*, *zoology*, *meteorology*, *geology*, *psychology*, *chemistry* using alphabetical order.



Adding and Locating an Item in BST

Example

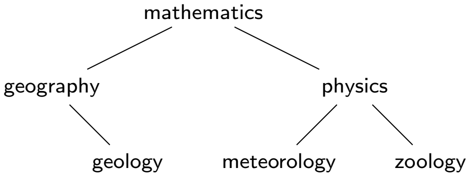
Form a BST for the words *mathematics*, *physics*, *geography*, *zoology*, *meteorology*, *geology*, *psychology*, *chemistry* using alphabetical order.



Adding and Locating an Item in BST

Example

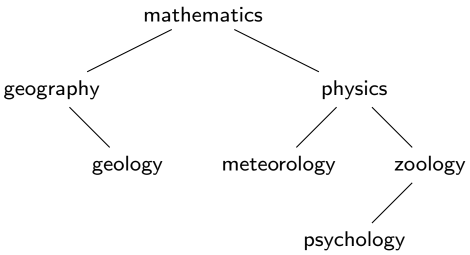
Form a BST for the words *mathematics*, *physics*, *geography*, *zoology*, *meteorology*, *geology*, *psychology*, *chemistry* using alphabetical order.



Adding and Locating an Item in BST

Example

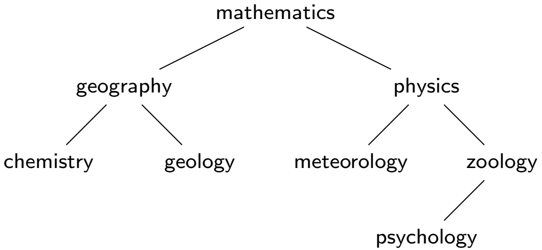
Form a BST for the words *mathematics*, *physics*, *geography*, *zoology*, *meteorology*, *geology*, *psychology*, *chemistry* using alphabetical order.



Adding and Locating an Item in BST

Example

Form a BST for the words *mathematics*, *physics*, *geography*, *zoology*, *meteorology*, *geology*, *psychology*, *chemistry* using alphabetical order.



Complexity in searching

$O(\log(n))$ vs. $O(n)$ in linear list

Decision Trees (Cây quyết định)

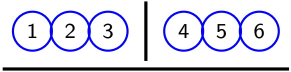
Example

There are seven coins, all with the same weight, and a counterfeit coin that weighs less than the others. How many weighings are necessary using a balance scale to determine which of the eight coins is the counterfeit one? Give an algorithm for finding this counterfeit coin.

Decision Trees (Cây quyết định)

Example

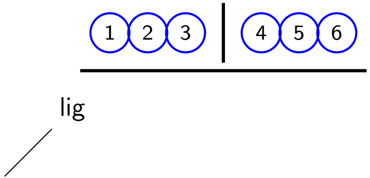
There are seven coins, all with the same weight, and a counterfeit coin that weighs less than the others. How many weighings are necessary using a balance scale to determine which of the eight coins is the counterfeit one? Give an algorithm for finding this counterfeit coin.



Decision Trees (Cây quyết định)

Example

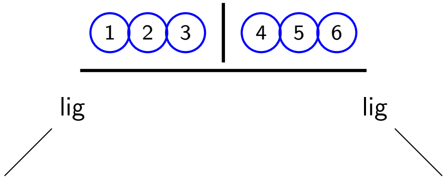
There are seven coins, all with the same weight, and a counterfeit coin that weighs less than the others. How many weighings are necessary using a balance scale to determine which of the eight coins is the counterfeit one? Give an algorithm for finding this counterfeit coin.



Decision Trees (Cây quyết định)

Example

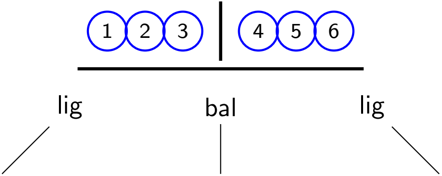
There are seven coins, all with the same weight, and a counterfeit coin that weighs less than the others. How many weighings are necessary using a balance scale to determine which of the eight coins is the counterfeit one? Give an algorithm for finding this counterfeit coin.



Decision Trees (Cây quyết định)

Example

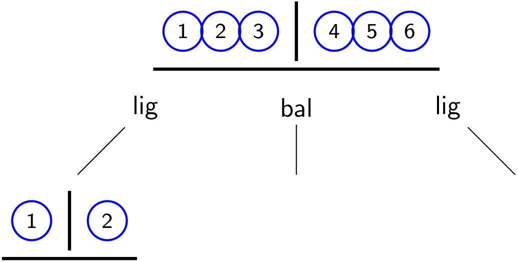
There are seven coins, all with the same weight, and a counterfeit coin that weighs less than the others. How many weighings are necessary using a balance scale to determine which of the eight coins is the counterfeit one? Give an algorithm for finding this counterfeit coin.



Decision Trees (Cây quyết định)

Example

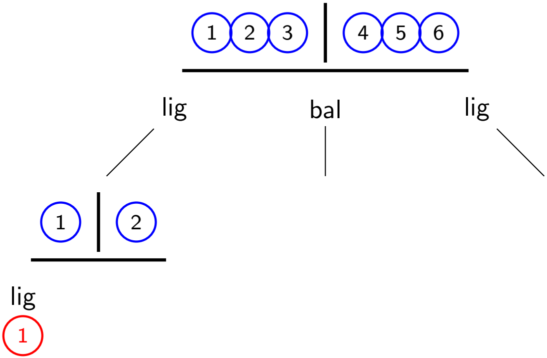
There are seven coins, all with the same weight, and a counterfeit coin that weighs less than the others. How many weighings are necessary using a balance scale to determine which of the eight coins is the counterfeit one? Give an algorithm for finding this counterfeit coin.



Decision Trees (Cây quyết định)

Example

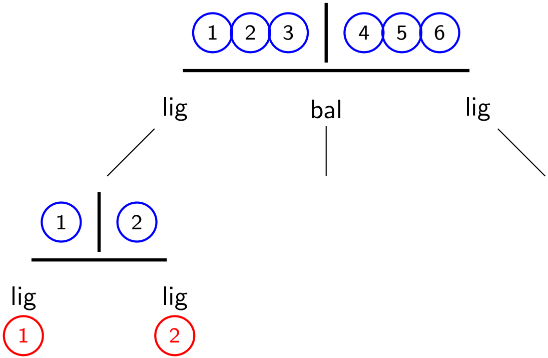
There are seven coins, all with the same weight, and a counterfeit coin that weighs less than the others. How many weighings are necessary using a balance scale to determine which of the eight coins is the counterfeit one? Give an algorithm for finding this counterfeit coin.



Decision Trees (Cây quyết định)

Example

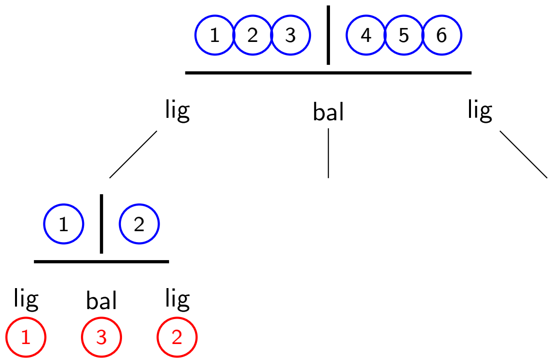
There are seven coins, all with the same weight, and a counterfeit coin that weighs less than the others. How many weighings are necessary using a balance scale to determine which of the eight coins is the counterfeit one? Give an algorithm for finding this counterfeit coin.



Decision Trees (Cây quyết định)

Example

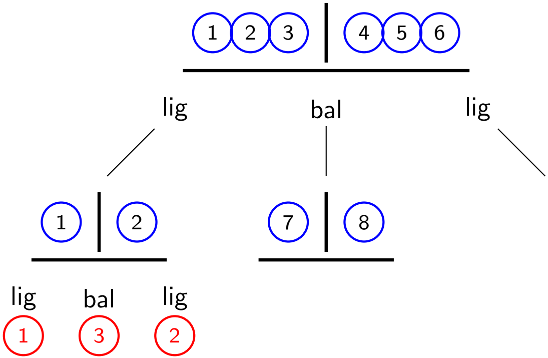
There are seven coins, all with the same weight, and a counterfeit coin that weighs less than the others. How many weighings are necessary using a balance scale to determine which of the eight coins is the counterfeit one? Give an algorithm for finding this counterfeit coin.



Decision Trees (Cây quyết định)

Example

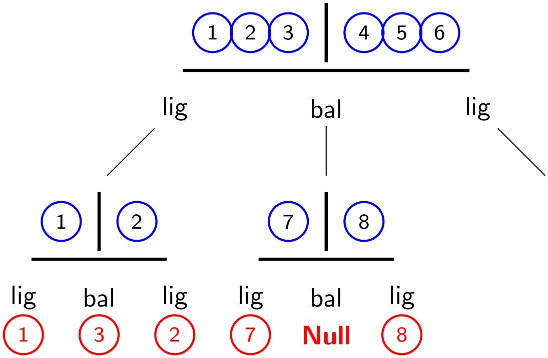
There are seven coins, all with the same weight, and a counterfeit coin that weighs less than the others. How many weighings are necessary using a balance scale to determine which of the eight coins is the counterfeit one? Give an algorithm for finding this counterfeit coin.



Decision Trees (Cây quyết định)

Example

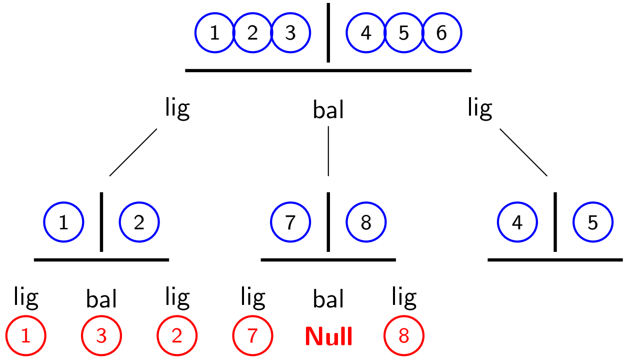
There are seven coins, all with the same weight, and a counterfeit coin that weighs less than the others. How many weighings are necessary using a balance scale to determine which of the eight coins is the counterfeit one? Give an algorithm for finding this counterfeit coin.



Decision Trees (Cây quyết định)

Example

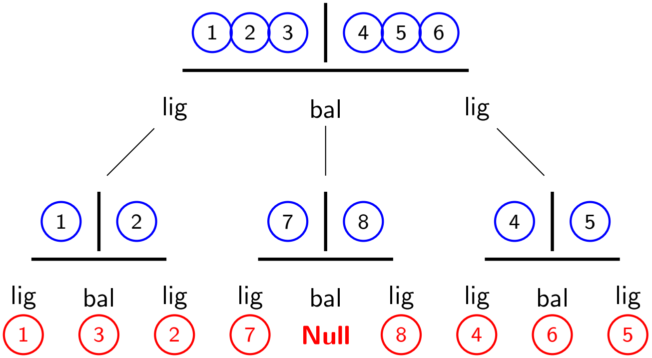
There are seven coins, all with the same weight, and a counterfeit coin that weighs less than the others. How many weighings are necessary using a balance scale to determine which of the eight coins is the counterfeit one? Give an algorithm for finding this counterfeit coin.



Decision Trees (Cây quyết định)

Example

There are seven coins, all with the same weight, and a counterfeit coin that weighs less than the others. How many weighings are necessary using a balance scale to determine which of the eight coins is the counterfeit one? Give an algorithm for finding this counterfeit coin.



Yet Another Application

Example

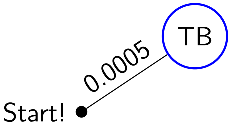
If we know that the probability that a person has tuberculosis (TB) is $p(\mathrm{TB}) = 0.0005$.
We also know $p(+|\mathrm{TB}) = 0.999$ and $p(-|\overline{\mathrm{TB}}) = 0.99$.
What is $p(\mathrm{TB}|+)$ and $p(\overline{\mathrm{TB}}|-)$?

Start! •

Yet Another Application

Example

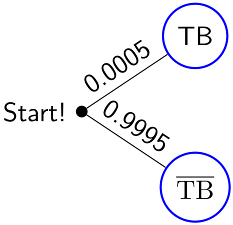
If we know that the probability that a person has tuberculosis (TB) is $p(\mathrm{TB}) = 0.0005$.
We also know $p(+|\mathrm{TB}) = 0.999$ and $p(-|\overline{\mathrm{TB}}) = 0.99$.
What is $p(\mathrm{TB}|+)$ and $p(\overline{\mathrm{TB}}|-)$?



Yet Another Application

Example

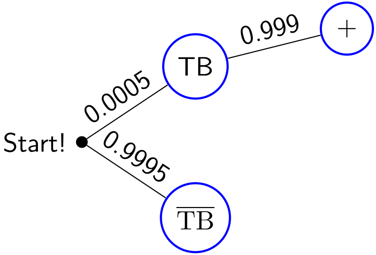
If we know that the probability that a person has tuberculosis (TB) is $p(\text{TB}) = 0.0005$.
We also know $p(+|\text{TB}) = 0.999$ and $p(-|\overline{\text{TB}}) = 0.99$.
What is $p(\text{TB}|+)$ and $p(\overline{\text{TB}}|-)$?



Yet Another Application

Example

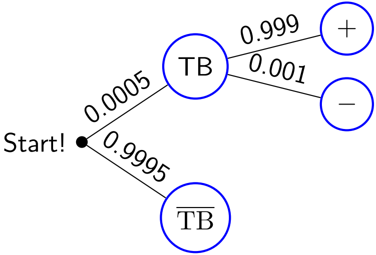
If we know that the probability that a person has tuberculosis (TB) is $p(\mathrm{TB}) = 0.0005$.
We also know $p(+|\mathrm{TB}) = 0.999$ and $p(-|\overline{\mathrm{TB}}) = 0.99$.
What is $p(\mathrm{TB}|+)$ and $p(\overline{\mathrm{TB}}|-)$?



Yet Another Application

Example

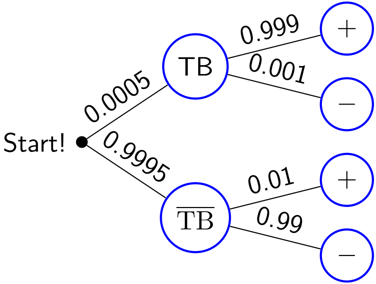
If we know that the probability that a person has tuberculosis (TB) is $p(\mathrm{TB}) = 0.0005$.
We also know $p(+|\mathrm{TB}) = 0.999$ and $p(-|\overline{\mathrm{TB}}) = 0.99$.
What is $p(\mathrm{TB}|+)$ and $p(\overline{\mathrm{TB}}|-)$?



Yet Another Application

Example

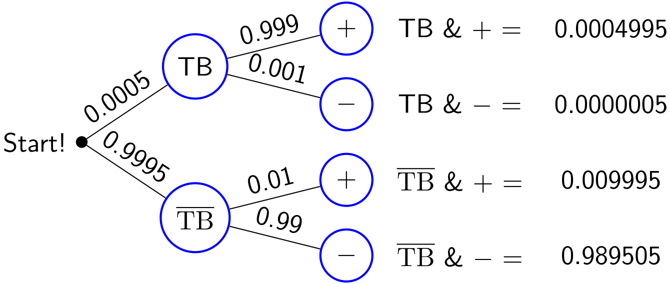
If we know that the probability that a person has tuberculosis (TB) is $p(\mathrm{TB}) = 0.0005$.
We also know $p(+|\mathrm{TB}) = 0.999$ and $p(-|\overline{\mathrm{TB}}) = 0.99$.
What is $p(\mathrm{TB}|+)$ and $p(\overline{\mathrm{TB}}|-)$?



Yet Another Application

Example

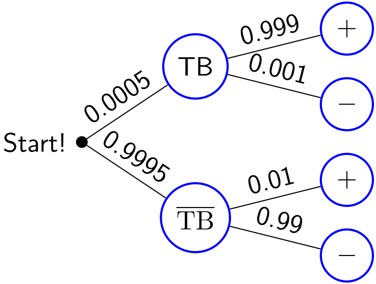
If we know that the probability that a person has tuberculosis (TB) is $p(\text{TB}) = 0.0005$.
We also know $p(+|\text{TB}) = 0.999$ and $p(-|\overline{\text{TB}}) = 0.99$.
What is $p(\text{TB}|+)$ and $p(\overline{\text{TB}}|-)$?



Yet Another Application

Example

If we know that the probability that a person has tuberculosis (TB) is $p(\mathrm{TB}) = 0.0005$.
We also know $p(+|\mathrm{TB}) = 0.999$ and $p(-|\overline{\mathrm{TB}}) = 0.99$.
What is $p(\mathrm{TB}|+)$ and $p(\overline{\mathrm{TB}}|-)$?



TB & $+ =$ 0.0004995

TB & $- =$ 0.0000005

$\overline{\mathrm{TB}}$ & $+ =$ 0.009995

$\overline{\mathrm{TB}}$ & $- =$ 0.989505

$$p(\mathrm{TB}|+) = \frac{p(\mathrm{TB}\cap +)}{p(+)} = \frac{0.0004995}{0.0004995 + 0.009995} \approx 0.0476$$

Problem

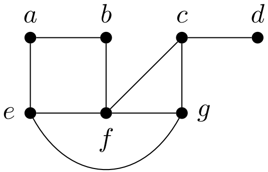
Definition

- A spanning tree (*cây khung*) in a graph G is a subgraph of G that is a tree which contains all vertices of G.

Problem

Definition

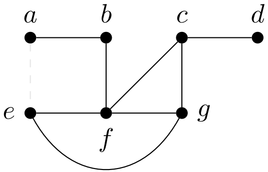
- A spanning tree (*cây khung*) in a graph G is a subgraph of G that is a tree which contains all vertices of G.



Problem

Definition

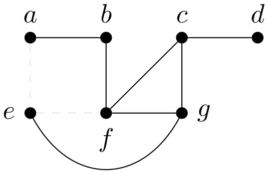
- A spanning tree (*cây khung*) in a graph G is a subgraph of G that is a tree which contains all vertices of G.



Problem

Definition

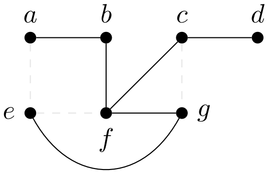
- A spanning tree (*cây khung*) in a graph G is a subgraph of G that is a tree which contains all vertices of G.



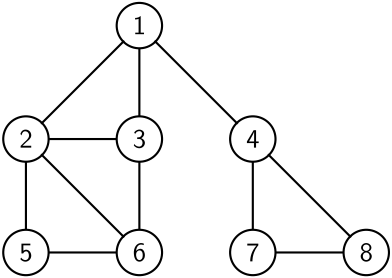
Problem

Definition

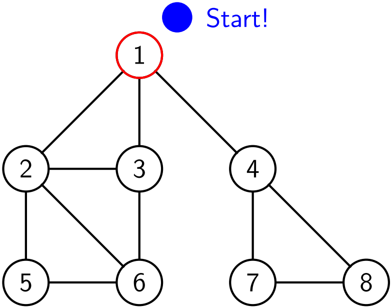
- A spanning tree (*cây khung*) in a graph G is a subgraph of G that is a tree which contains all vertices of G.



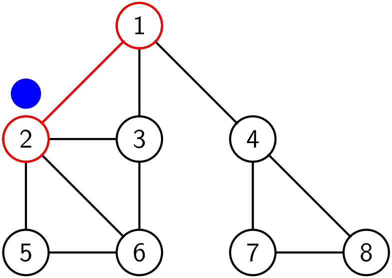
Depth-First Search (Tìm kiếm ưu tiên chiều sâu)



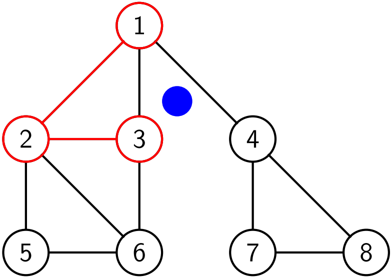
Depth-First Search (Tìm kiếm ưu tiên chiều sâu)



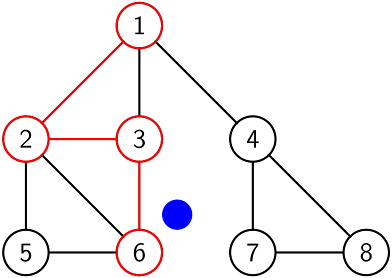
Depth-First Search (Tìm kiếm ưu tiên chiều sâu)



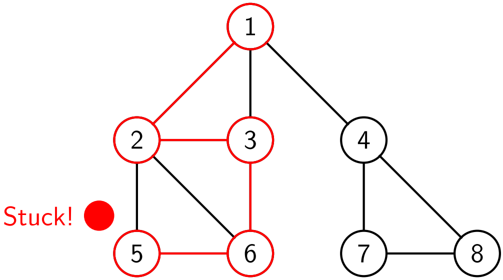
Depth-First Search (Tìm kiếm ưu tiên chiều sâu)



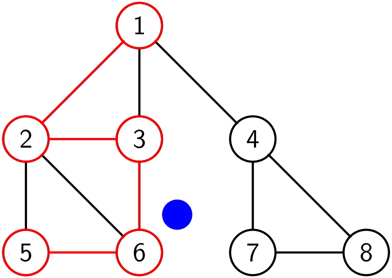
Depth-First Search (Tìm kiếm ưu tiên chiều sâu)



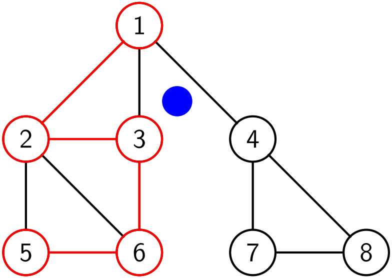
Depth-First Search (Tìm kiếm ưu tiên chiều sâu)



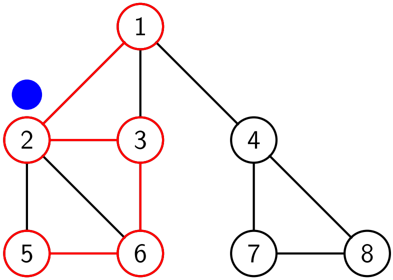
Depth-First Search (Tìm kiếm ưu tiên chiều sâu)



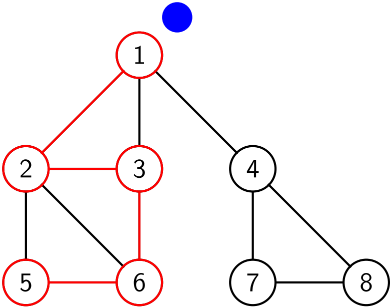
Depth-First Search (Tìm kiếm ưu tiên chiều sâu)



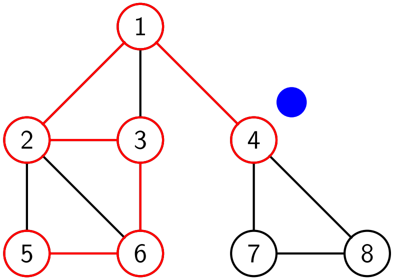
Depth-First Search (Tìm kiếm ưu tiên chiều sâu)



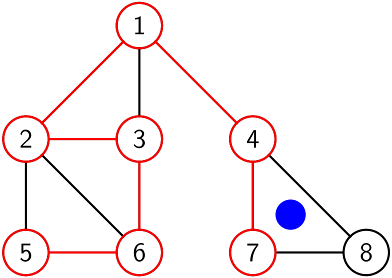
Depth-First Search (Tìm kiếm ưu tiên chiều sâu)



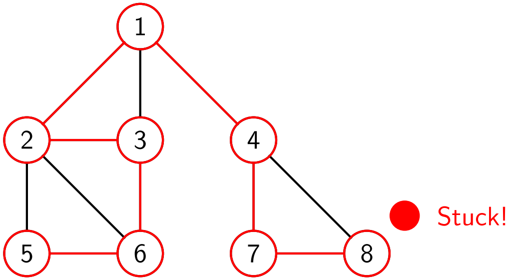
Depth-First Search (Tìm kiếm ưu tiên chiều sâu)



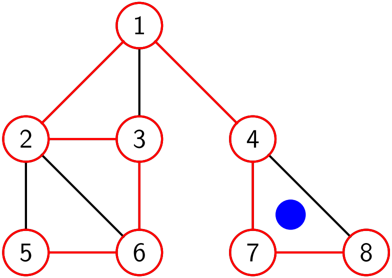
Depth-First Search (Tìm kiếm ưu tiên chiều sâu)



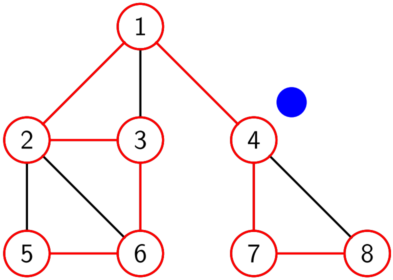
Depth-First Search (Tìm kiếm ưu tiên chiều sâu)



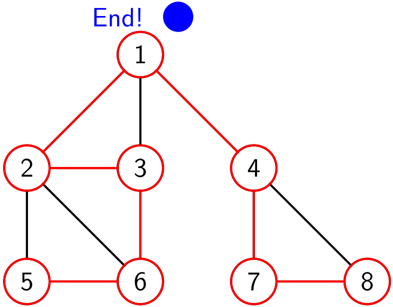
Depth-First Search (Tìm kiếm ưu tiên chiều sâu)



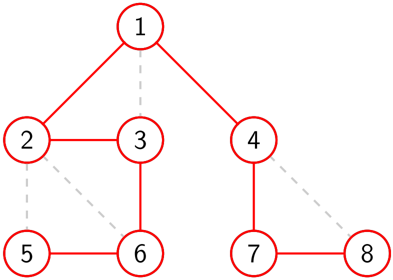
Depth-First Search (Tìm kiếm ưu tiên chiều sâu)



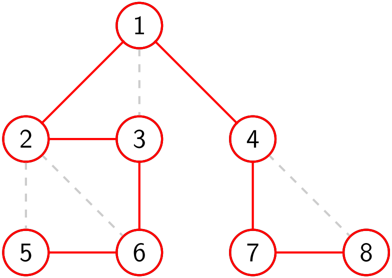
Depth-First Search (Tìm kiếm ưu tiên chiều sâu)



Depth-First Search (Tìm kiếm ưu tiên chiều sâu)



Depth-First Search (Tìm kiếm ưu tiên chiều sâu)



Property

- Go deeper as you can
- Backtrack (*quay lui*) to possible branch when you are stuck.
- $O(e)$ or $O(n^2)$

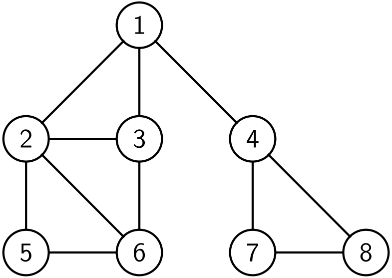
Depth-First Search

Algorithm

procedure *DFS* (G)
  $T :=$ tree consisting only vertex v_1
  visit(v_1)

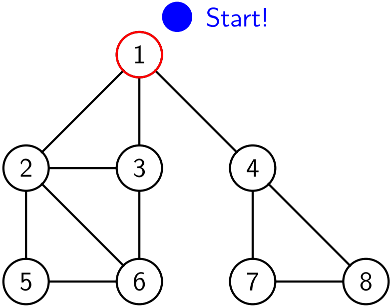
procedure *visit*$(v$: vertex of $G)$ /* recursive */
  for each vertex w adjacent to v and not in T
    add w and edge $\{v, w\}$ to T
    visit(w)

Breadth-First Search (Tìm kiếm ưu tiên chiều rộng)



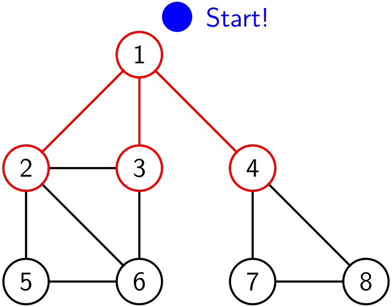
vertex	L
	$\emptyset$

Breadth-First Search (Tìm kiếm ưu tiên chiều rộng)



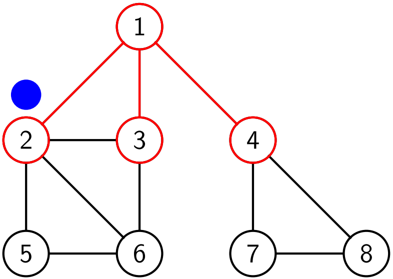
vertex	L
	$\emptyset$
1	

Breadth-First Search (Tìm kiếm ưu tiên chiều rộng)



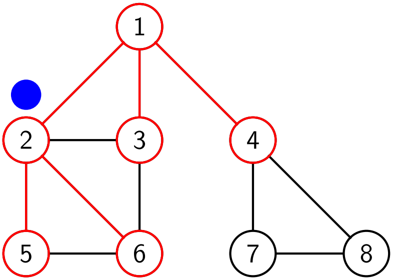
vertex	L
	$\emptyset$
1	2, 3, 4

Breadth-First Search (Tìm kiếm ưu tiên chiều rộng)



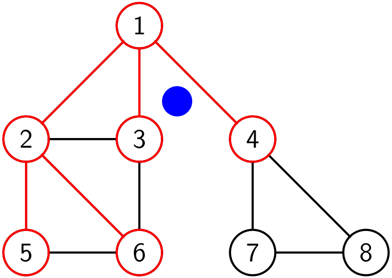
vertex	L
	$\emptyset$
1	2, 3, 4
2	3, 4

Breadth-First Search (Tìm kiếm ưu tiên chiều rộng)



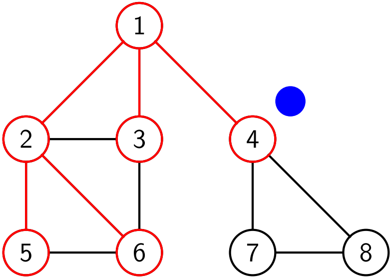
vertex	L
	$\emptyset$
1	2, 3, 4
2	3, 4, 5, 6

Breadth-First Search (Tìm kiếm ưu tiên chiều rộng)



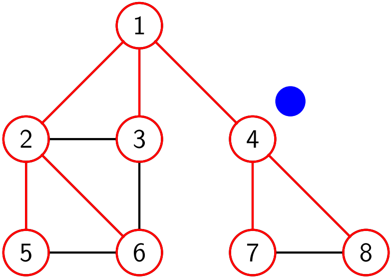
vertex	L
	$\emptyset$
1	2, 3, 4
2	3, 4, 5, 6
3	4, 5, 6

Breadth-First Search (Tìm kiếm ưu tiên chiều rộng)



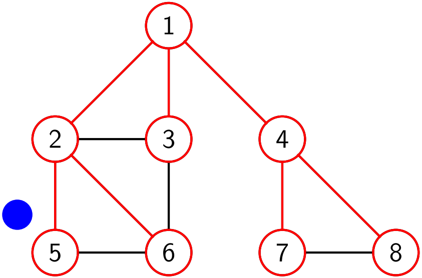
vertex	L
	$\emptyset$
1	2, 3, 4
2	3, 4, 5, 6
3	4, 5, 6
4	5, 6

Breadth-First Search (Tìm kiếm ưu tiên chiều rộng)



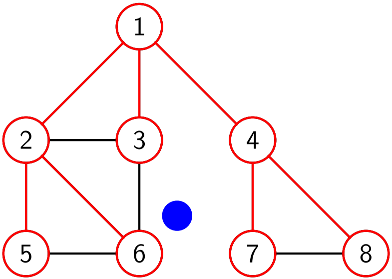
vertex	L
	$\emptyset$
1	2, 3, 4
2	3, 4, 5, 6
3	4, 5, 6
4	5, 6, 7, 8

Breadth-First Search (Tìm kiếm ưu tiên chiều rộng)



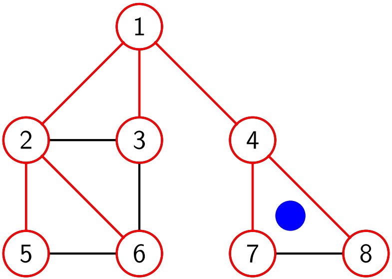
vertex	L
	$\emptyset$
1	2, 3, 4
2	3, 4, 5, 6
3	4, 5, 6
4	5, 6, 7, 8
5	6, 7, 8

Breadth-First Search (Tìm kiếm ưu tiên chiều rộng)



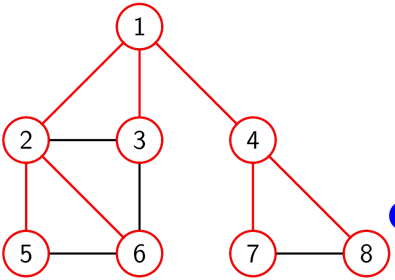
vertex	L
	$\emptyset$
1	2, 3, 4
2	3, 4, 5, 6
3	4, 5, 6
4	5, 6, 7, 8
5	6, 7, 8
6	7, 8

Breadth-First Search (Tìm kiếm ưu tiên chiều rộng)



vertex	L
	$\emptyset$
1	2, 3, 4
2	3, 4, 5, 6
3	4, 5, 6
4	5, 6, 7, 8
5	6, 7, 8
6	7, 8
7	8

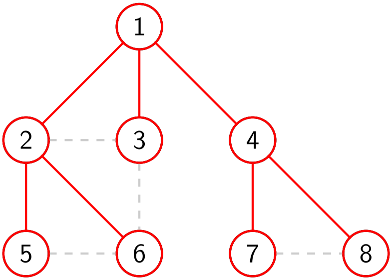
Breadth-First Search (Tìm kiếm ưu tiên chiều rộng)



End!

vertex	L
	$\emptyset$
1	2, 3, 4
2	3, 4, 5, 6
3	4, 5, 6
4	5, 6, 7, 8
5	6, 7, 8
6	7, 8
7	8
8	$\emptyset$

Breadth-First Search (Tìm kiếm ưu tiên chiều rộng)



vertex	L
	$\emptyset$
1	2, 3, 4
2	3, 4, 5, 6
3	4, 5, 6
4	5, 6, 7, 8
5	6, 7, 8
6	7, 8
7	8
8	$\emptyset$

Property

- $O(e)$ or $O(n^2)$

Breadth-First Search

Algorithm

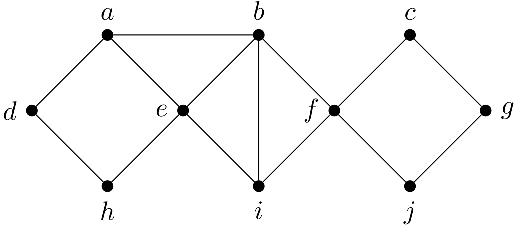
procedure *BFS* (G)

$T :=$ tree consisting only vertex v_1
$L :=$ empty list
put v_1 in the list L of unprocessed vertices
while L is not empty
remove the first vertex, v, from L
for each neighbor w of v
if w is not in L and not in T **then**
add w to the end of the list L
add w and edge $\{v, w\}$ to T

Exercise

Exercise

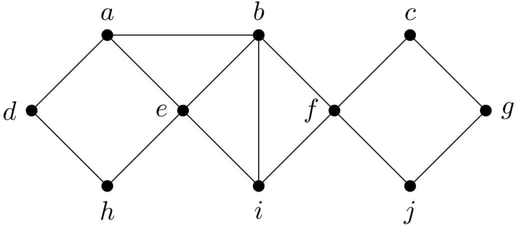
Find spanning tree in the following graphs.

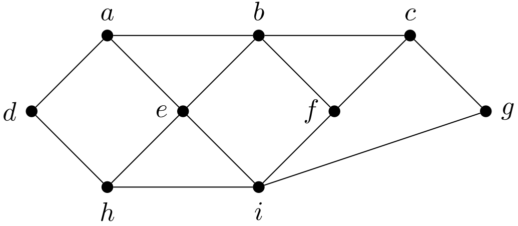


Exercise

Exercise

Find spanning tree in the following graphs.





Minimum Spanning Trees

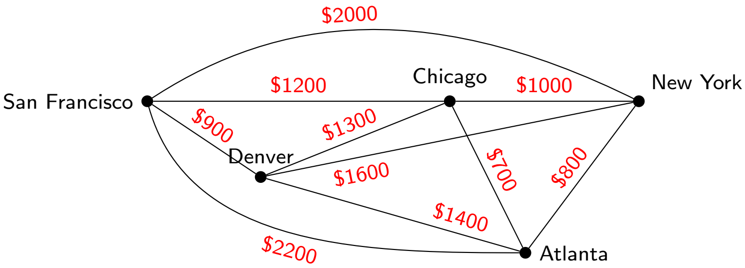
Definition

- A minimum spanning tree (*cây khung nhỏ nhất*) in a connected weighted graph is a spanning tree that has the smallest possible sum of weights of its edges.

Minimum Spanning Trees

Definition

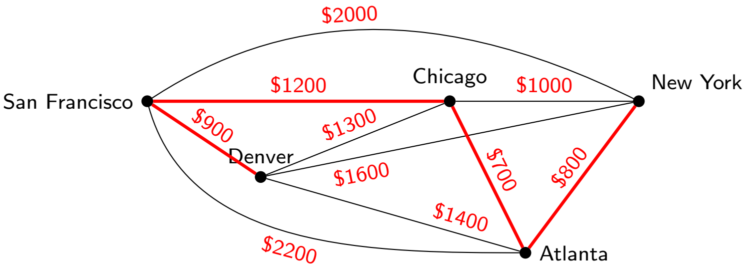
- A minimum spanning tree (*cây khung nhỏ nhất*) in a connected weighted graph is a spanning tree that has the smallest possible sum of weights of its edges.



Minimum Spanning Trees

Definition

- A minimum spanning tree (*cây khung nhỏ nhất*) in a connected weighted graph is a spanning tree that has the smallest possible sum of weights of its edges.



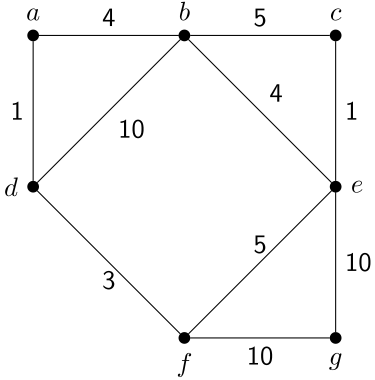
Prim's Algorithm (Nearest-Neighbor)

Prim's Algorithm (1957)

procedure $Prim(G)$
$T :=$ a minimum-weight edge
for $i := 1$ to $n - 2$
$e :=$ an edge of minimum weight incident to a vertex in T and not forming a simple circuit in T if added to T
$T := T$ with e added
return T

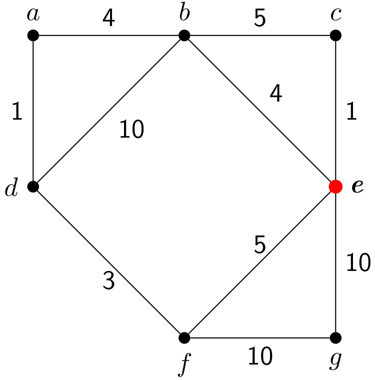
Prim's Algorithm (Nearest-Neighbor)

- Pick a vertex to start from
- Iteratively absorb smallest edge possible



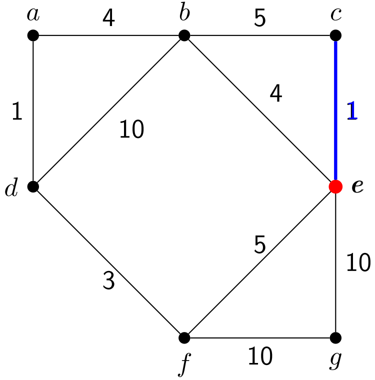
Prim's Algorithm (Nearest-Neighbor)

- Pick a vertex to start from
- Iteratively absorb smallest edge possible



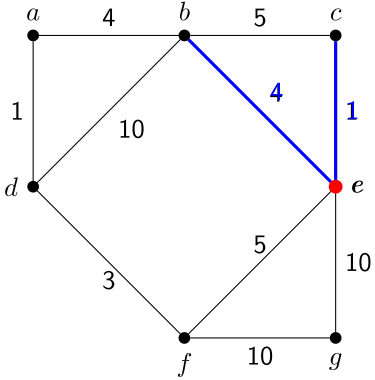
Prim's Algorithm (Nearest-Neighbor)

- Pick a vertex to start from
- Iteratively absorb smallest edge possible



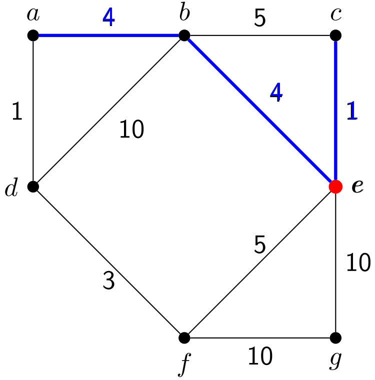
Prim's Algorithm (Nearest-Neighbor)

- Pick a vertex to start from
- Iteratively absorb smallest edge possible



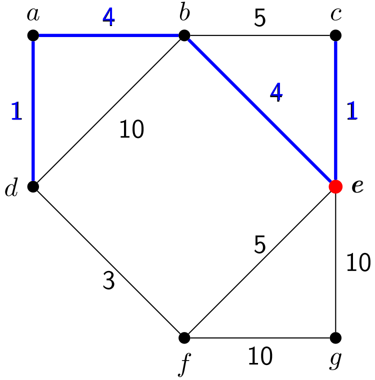
Prim's Algorithm (Nearest-Neighbor)

- Pick a vertex to start from
- Iteratively absorb smallest edge possible



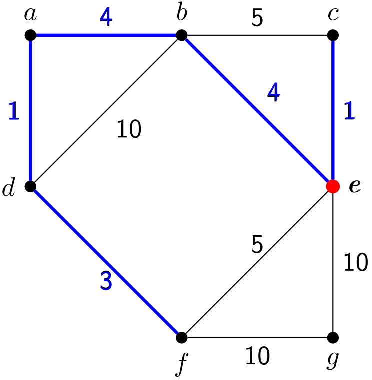
Prim's Algorithm (Nearest-Neighbor)

- Pick a vertex to start from
- Iteratively absorb smallest edge possible



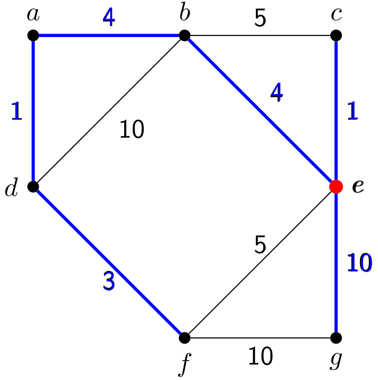
Prim's Algorithm (Nearest-Neighbor)

- Pick a vertex to start from
- Iteratively absorb smallest edge possible



Prim's Algorithm (Nearest-Neighbor)

- Pick a vertex to start from
- Iteratively absorb smallest edge possible



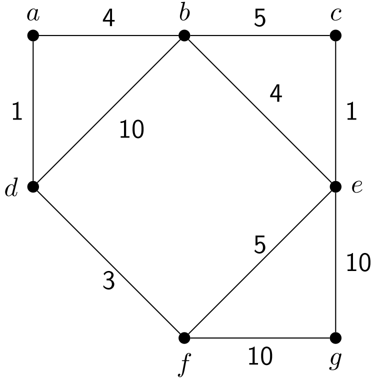
Kruskal's Algorithm (Lightest-Edge)

Kruskal's Algorithm (1958)

procedure *Kruskal*(G)
  $T :=$ empty graph
  for $i := 1$ **to** $n-1$
    $e :=$ any edge in G with smallest weight that does not form a simple circuit when added to T
    $T := T$ with e added
  return T

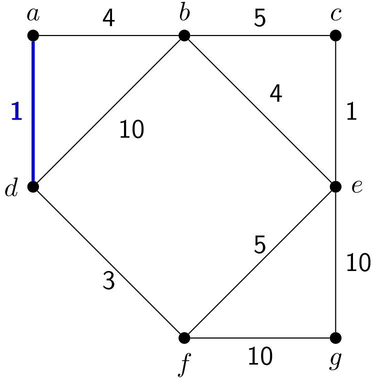
Kruskal's Algorithm (Lightest-Edge)

- Iteratively add smallest edge possible



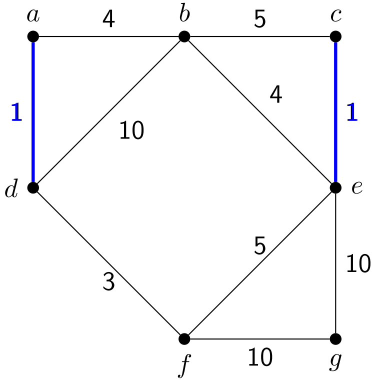
Kruskal's Algorithm (Lightest-Edge)

- Iteratively add smallest edge possible



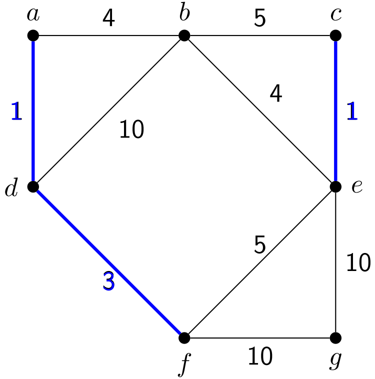
Kruskal's Algorithm (Lightest-Edge)

- Iteratively add smallest edge possible



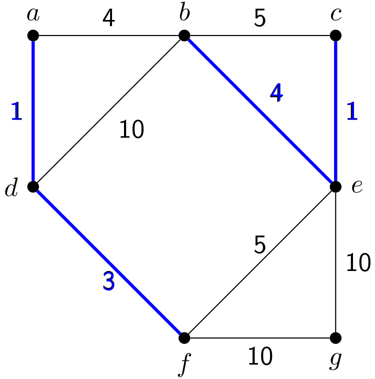
Kruskal's Algorithm (Lightest-Edge)

- Iteratively add smallest edge possible



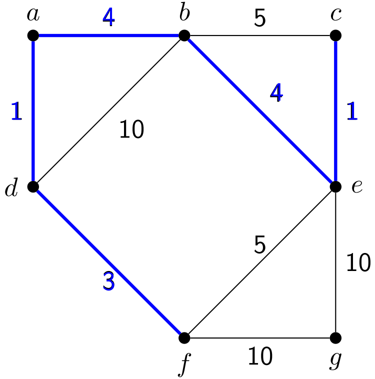
Kruskal's Algorithm (Lightest-Edge)

- Iteratively add smallest edge possible



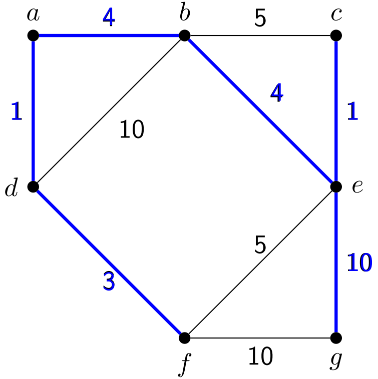
Kruskal's Algorithm (Lightest-Edge)

- Iteratively add smallest edge possible



Kruskal's Algorithm (Lightest-Edge)

- Iteratively add smallest edge possible



Exercise

Exercise

By using Prim's and Kruskal's algorithm, determine minimum spanning tree in the following graphs.

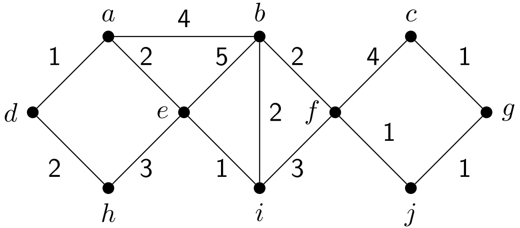
.

Exercise

Exercise

By using Prim's and Kruskal's algorithm, determine minimum spanning tree in the following graphs.

.

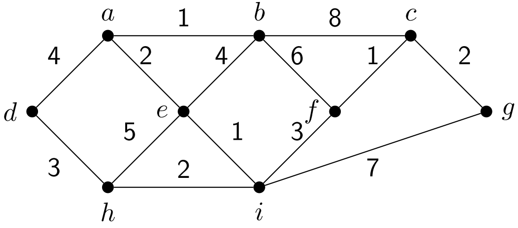


Exercise

Exercise

By using Prim's and Kruskal's algorithm, determine minimum spanning tree in the following graphs.

.

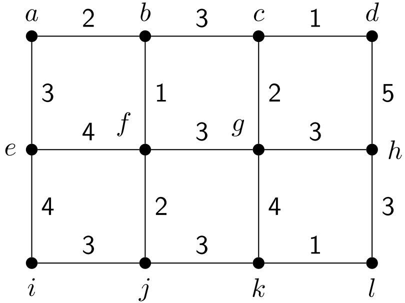


Exercise

Exercise

By using Prim's and Kruskal's algorithm, determine minimum spanning tree in the following graphs.

.



Exercise

Exercise

By using Prim's and Kruskal's algorithm, determine minimum spanning tree in the following graphs. (and maximum spanning tree (*cây khung cực đại*).