VÕ TIẾN
Thảo luận kiến thức CNTT trường BK về KHMT(CScience), KTMT(CEngineering)
https://www.facebook.com/groups/khmt.ktmt.cse.bku

Cấu Trúc Dữ Liệu Và Giải Thuật (DSA)

DSA3 - HK242

Lý Thuyết

Thảo luận kiến thức CNTT trường BK
về KHMT(CScience), KTMT(CEngineering)
https://www.facebook.com/groups/khmt.ktmt.cse.bku

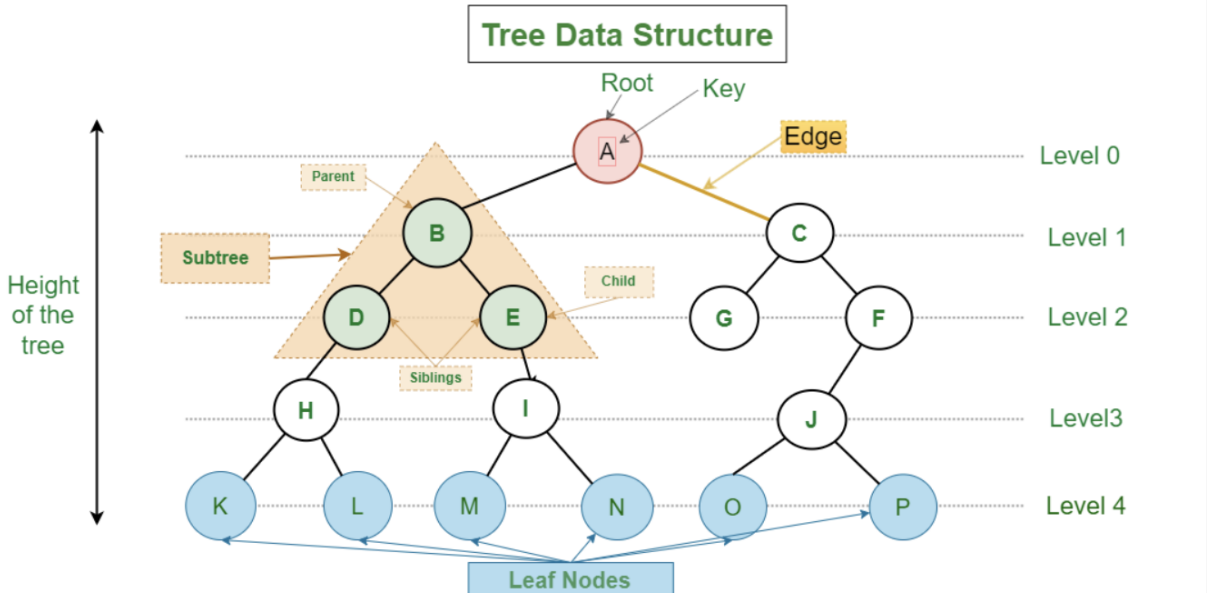
TP. HỒ CHÍ MINH, THÁNG 5/2025

Mục lục

1 Định nghĩa Tree

Cây (*Tree*) là tập hợp các phần tử được ngoại là nút (*Node*) và các đường có hướng được gọi là nhánh (*branches*).
Cây (*Tree*) là một đồ thị vô hướng liên thông không có mạch đơn giản. Do đó, một cây phải là một đồ thị đơn giản.

- Cây gốc (*root*): Nút trên cùng của cây được gọi là nút gốc. Đây là nút duy nhất không có nút cha.
- Nút lá (*leaf*): Nút không có nút con nào được gọi là nút lá.
- Nút nội (*internal*) là nút không phải là nút *root* và nút *leaf*.
- Nút con (*child*): Nút được kết nối với nút khác bằng cạnh được gọi là nút con. Nút mà nó được kết nối được gọi là nút cha (*parent*) nút cha có số lớn nút con lớn hơn 0.
- Tổ tiên (*ancestor*): Nút nằm trên đường đi từ nút gốc đến nút đã cho. Nút gốc là tổ tiên của tất cả các nút trong cây.
- Hậu duệ (*desancestor*): Nút có thể được truy cập bằng cách đi theo một đường đi gồm các cạnh từ nút đã cho. Nút đã cho là tổ tiên của chính nó.
- Anh chị em (*Siblings*): Hai nút có cùng nút cha được gọi là anh chị em.
- Đường đi (*path*): Một đường đi là một chuỗi các nút được kết nối bởi các cạnh. Đường đi từ nút gốc đến bất kỳ nút nào khác được gọi là đường đi từ gốc đến lá.
- Mức/Bật (*level*): Mức của một nút trong cây là số cạnh từ nút gốc đến nút. Nút gốc (*root*) ở mức 0 và các nút lá ở mức cao nhất.
- Chiều cao (*height*): Chiều cao của cây là mức cao nhất của bất kỳ nút nào trong cây.
- Cây con (*subtree*): Một cây con của cây là một tập hợp con của các nút trong cây tạo thành một cây riêng biệt. Cây con có gốc tại một nút N là tập hợp tất cả các nút là hậu duệ (*desancestor*) của N.
- Bậc của nút (*Degree_of_a_node*): Bậc của nút là số lượng nhánh (*branches*) liên kết với nó.
- Nhánh vào (*Indegree_branch*): Nhánh vào của một nút trong cây là số nút con trỏ tới nó.
- Nhánh ra (*Outdegree_branch*): Độ ra của một nút trong cây là số nút con mà nó trỏ tới.
- Sơ đồ tổ chức (*Organization_chart*): Sơ đồ tổ chức có thể được sử dụng để hiển thị thứ bậc của các nút trong cây.
- Danh sách ngoặc đơn (*parenthetical_listing*): Danh sách ngoặc đơn có thể được sử dụng để hiển thị mối quan hệ giữa các nút trong cây.
- Danh sách thụt lề (*indented_list*): Danh sách thụt lề có thể được sử dụng để hiển thị thứ bậc của các nút trong cây.
- Cây gốc có thứ tự (*ordered_rooted_tree*) : Cây gốc có thứ tự là cây gốc trong đó các con của mỗi đỉnh được sắp xếp. Điều này có nghĩa là có một thứ tự cụ thể trong đó các con của một đỉnh được truy cập khi duyệt cây



Hình 1: *Basic Tree Concepts*

- Cây gốc (*root*): Node A
- Nút lá (*leaf*): Node K, L, M, B, O, P, G
- Nút nội (*internal*) Node B, C, D, E, F, H, I, J
- Nút con (*child*) của Node *B*: Node D, E
- Nút cha (*Parent*) của Node *E*: Node B
- Tổ tiên (*ancestor*) của Node *J*: Node A, C, F
- Hậu duệ (*desancestor*) Của Node *C*: Node G, F, J, O, P
- Anh chị em (*Siblings*) Của Node *E*: Node D
- Đường đi (*path*) từ nút *A* đến nút *K*: Node A -> B -> D -> H -> K.
- Mức/Bật (*level*): level 3. node H, I, J
- Chiều cao (*height*): Height = max(level) + 1 = 5 .
- Cây con (*subtree*) của Node *A*: Tree node root B (subtree left) and Tree node root C (subtree right).
- Bậc của nút (*Degree_of_a_node*) của H: Bật của nút H là 3 nhánh DH, HK, HL
- Nhánh vào (*Indegree_branch*) của H: Nhánh DH
- Nhánh ra (*Outdegree_branch*) của H: Nhánh HK, HL
- Danh sách ngoặc đơn (*parenthetical_listing*): A(B(D(H(K, L)), E(null, I(M,N))), C(G, F(J(O, P))))

2 Binary Trees

Cây nhị phân $(Binary\_Trees)$ là một cấu trúc dữ liệu trong đó mỗi nút có tối đa hai con. Các con của một nút thường được gọi là con trái $(left)$ và con phải $(right)$.

- Cây nhị phân có N node
  - $min(height) = \lfloor log_2 N \rfloor + 1$ or $min(height) = \lceil log_2(N+1) \rceil$ Cây nhị phân hoàn chỉnh.
  - $max(height) = N$ cây đơn tuyến tính.
- Cây nhị phân có chiều cao là H
  - $min(Node) = H$ cây đơn tuyến tính.
  - $max(height) = 2^H - 1$ Cây nhị phân hoàn hảo.
- Cây nhị phân hoàn hảo $(Complete\_tree)$: Cây nhị phân hoàn hảo là cây nhị phân đầy đủ mà tất cả các nút lá đều ở cùng một mức $N = N_{max} = 2^n - 1$.
- Cây nhị phân gần hoàn chỉnh $(Nearly\_complete\_tree)$: Cây nhị phân gần hoàn chỉnh là cây nhị phân mà tất cả các cấp đều được lấp đầy ngoại trừ cấp cuối cùng, và các nút ở cấp cuối cùng nằm càng xa bên trái càng tốt $H = H_{min} = \lfloor log_2 N \rfloor + 1$
- Cây nhị phân đầy đủ $(Full\_binary\_tree)$: Cây nhị phân đầy đủ là cây nhị phân mà mỗi nút có hai con hoặc không có con nào.
- Cây cân bằng $(Balanced\_tree)$ là cây nhị phân trong đó sự khác biệt giữa chiều cao của các cây con trái và phải của bất kỳ nút nào là nhiều nhất 1.
  - Hệ số cân bằng $(balance\_factor)$ của một nút trong cây nhị phân là sự khác biệt giữa chiều cao của cây con trái và cây con phải của nút đó. $B = H_L - H_R$
  - $Balanced\_tree$ có $B \in \{0, -1, 1\}$
  - Cây con $(subtrees)$ cũng là cây cân bằng $(Balanced\_tree)$
- Cây đơn tuyến tính $(singly\_linked\_list)$ hay Degenerate$(DegenerateBinarytree)$_Binary_tree) là một cấu trúc dữ liệu tuyến tính trong đó mỗi nút chứa một giá trị và một con trỏ đến nút tiếp theo. Cây đơn tuyến tính không có nút đầu cuối hoặc nút cuối cùng.
- Hiện thực liên kết $(Linked\_implementation)$ của cây nhị phân là một cách lưu trữ cây nhị phân trong bộ nhớ bằng cách sử dụng danh sách liên kết. Trong phiên bản liên kết, mỗi nút trong cây chứa một con trỏ đến con trái, con phải và dữ liệu của nó. Nút gốc của cây là nút duy nhất không có cha mẹ.
- Hiện thực dựa trên mảng $(Array\_based\_implementation)$ Phù hợp với cây hoàn chỉnh $(Complete\_tree)$, cây gần hoàn chỉnh $(Nearly\_complete\_tree)$. Dựa trên mảng của cây nhị phân là một cách lưu trữ cây nhị phân trong bộ nhớ bằng cách sử dụng mảng. Trong triển khai dựa trên mảng, mỗi nút trong cây được lưu trữ trong một phần tử mảng và các phần tử mảng được sắp xếp theo cách phản ánh cấu trúc của cây.

3 Tree Traversals

3.1 Depth-first traversal (Duyệt theo chiều sâu)

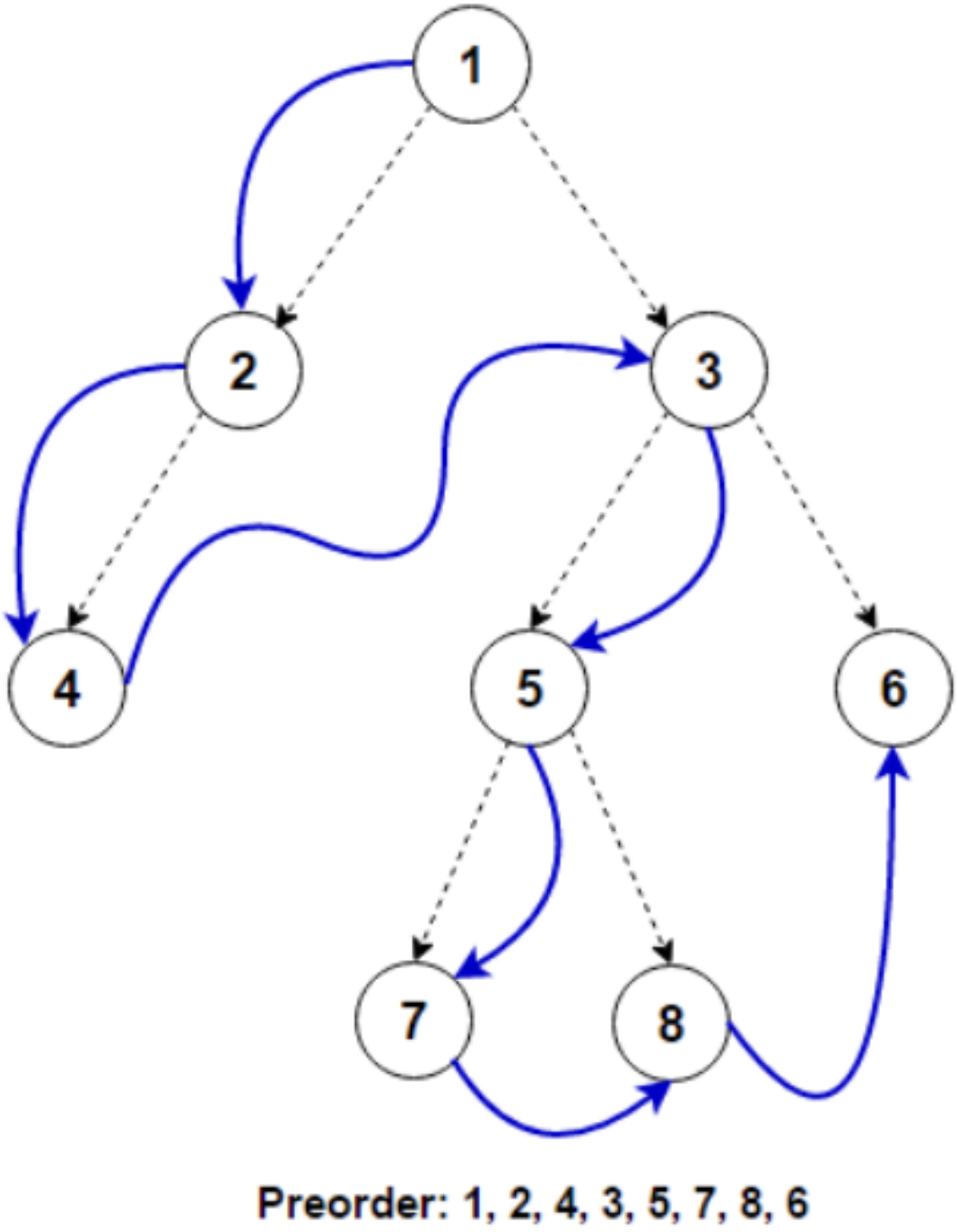
Duyệt chiều sâu (DFS) là một thuật toán đệ quy để duyệt hoặc tìm kiếm cấu trúc dữ liệu cây. Thuật toán bắt đầu từ nút gốc và khám phá đệ quy càng xa càng tốt xuống một nhánh trước khi quay trở lại và khám phá nhánh tiếp theo. xử lý tất cả các hậu duệ (*descendents*) của một đứa con trước chuyển sang con tiếp theo.

3.1.1 Preorder traversal (Duyệt tiền thứ tự)

Duyệt tiền thứ tự Duyệt nút cha trước rồi tới cây con bên trái tiếp theo đến con bên phải.

1. Nếu Cây là rỗng thì dừng *Return*
2. Thăm nút *root*
3. Duyệt tiền thứ tự cây con gốc trái $left$
4. Duyệt tiền thứ tự cây con gốc phải *right*

Sourcode 1



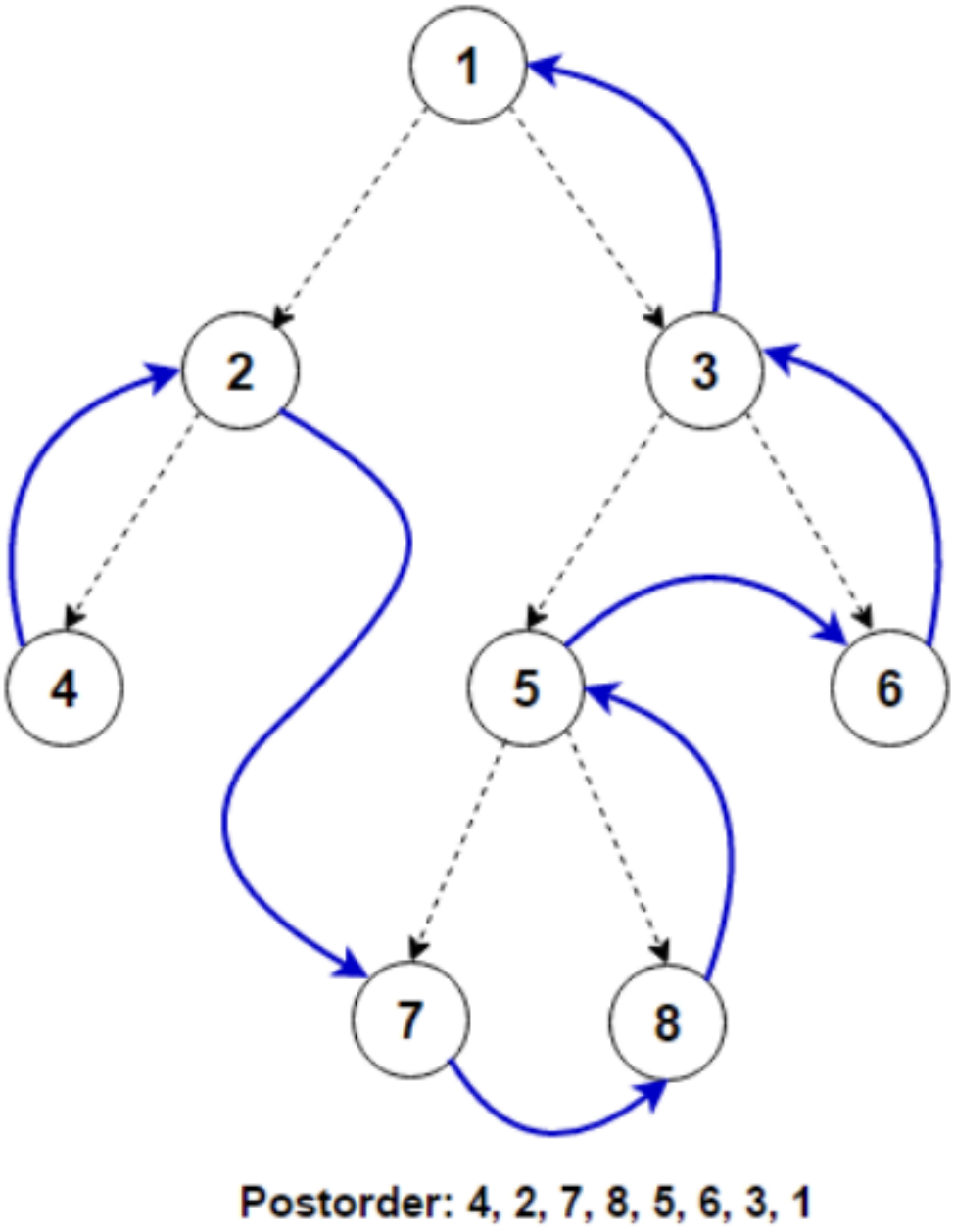
Hình 2: *Duyệt tiền thứ tự.*

3.1.2 Postorder Traversal (Duyệt hậu thứ tự)

Duyệt tiền thứ tự Duyệt nút con bên trái rồi con bên phải cuối cùng là nút cha.

1. Nếu Cây là rỗng thì dừng *Return*
2. Duyệt hậu thứ tự cây con gốc trái *left*
3. Duyệt hậu thứ tự cây con gốc phải *right*
4. Thăm nút *root*

Sourcode 1



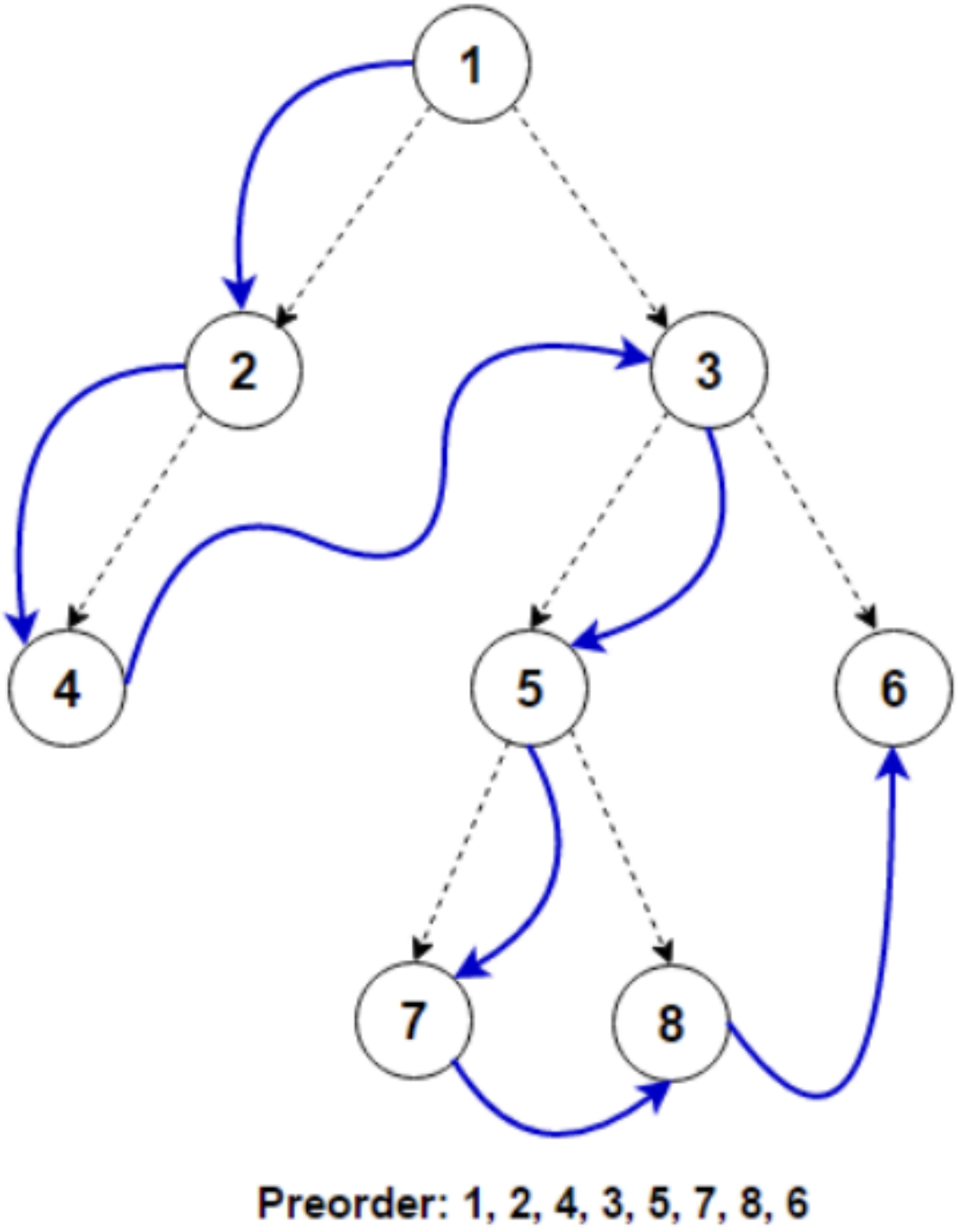
Hình 3: *Duyệt hậu thứ tự.*

3.1.3 Inorder Traversal (Duyệt trung thứ tự)

Duyệt tiền thứ tự Duyệt nút con bên trái rồi nút cha cuối cùng con bên phải.

1. Nếu Cây là rỗng thì dừng *Return*
2. Duyệt trung thứ tự cây con gốc trái *left*
3. Thăm nút *root*
4. Duyệt trung thứ tự cây con gốc phải *right*

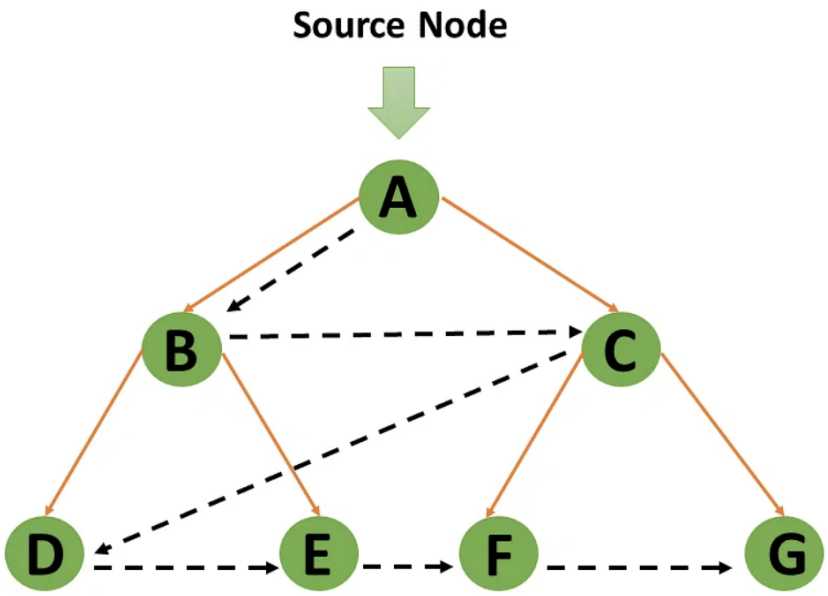
Sourcode 1



Hình 4: *Duyệt trung thứ tự.*

3.2 Breadth-First Traversals (Duyệt theo chiều rộng)

Duyệt theo chiều rộng (BFS) là một thuật toán để duyệt hoặc tìm kiếm cấu trúc dữ liệu cây. Thuật toán bắt đầu từ nút gốc và khám phá tất cả các nút ở cùng một cấp trước khi chuyển sang cấp tiếp theo.
Sourcode 1



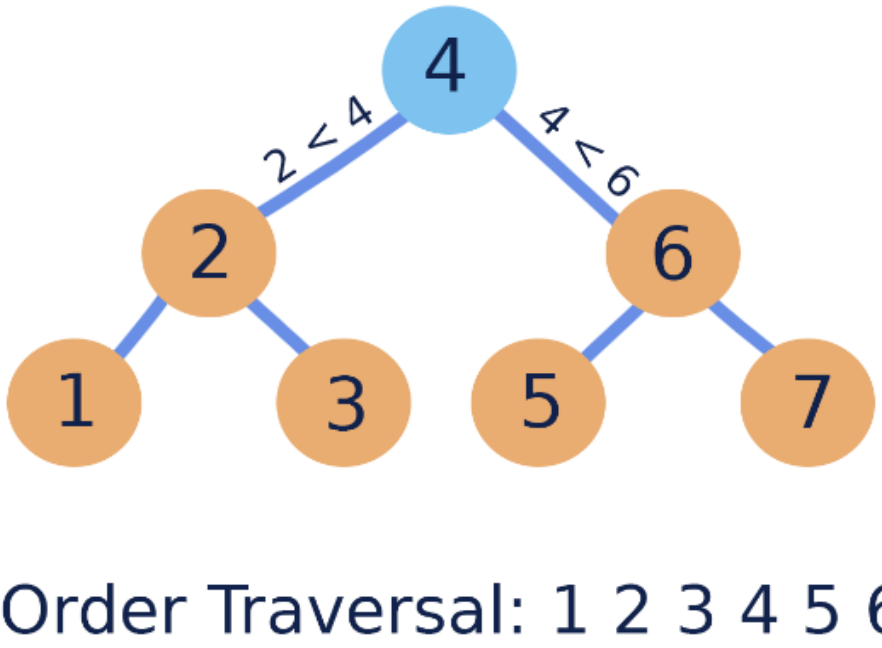
Hình 5: *Duyệt theo chiều rộng.*

3.3 Các dạng bài tập Phần lớn trong cuối kì TN

- Expression Trees cây biểu hiện biểu thức toán Prefix sang Infix and Postfix.
- Infix and Posrtfix tính kết quả
- PostOrder, InOrder, PreOrder cho 2 cái suy ra 1 cái còn lại
- PostOrder, InOrder, PreOrder cho 1 cái và 1 điều kiện nữa tìm ra 2 cái còn lại
- tìm PostOrder, InOrder, PreOrder.
- duyệt BFS

4 Binary Search Trees (Cây tìm kiếm nhị phân)

Cây nhị phân tìm kiếm (BST) là một cấu trúc dữ liệu cây trong đó mỗi nút có một giá trị và các nút con trái và phải. Các nút con trái *(left)* luôn có giá trị nhỏ hơn giá trị của nút gốc *(root)*, và các nút con phải *(right)*
Duyệt theo tiền thứ tự *(InOrder)* của cây tìm kiếm nhị phân tạo ra một danh sách có thứ tự.



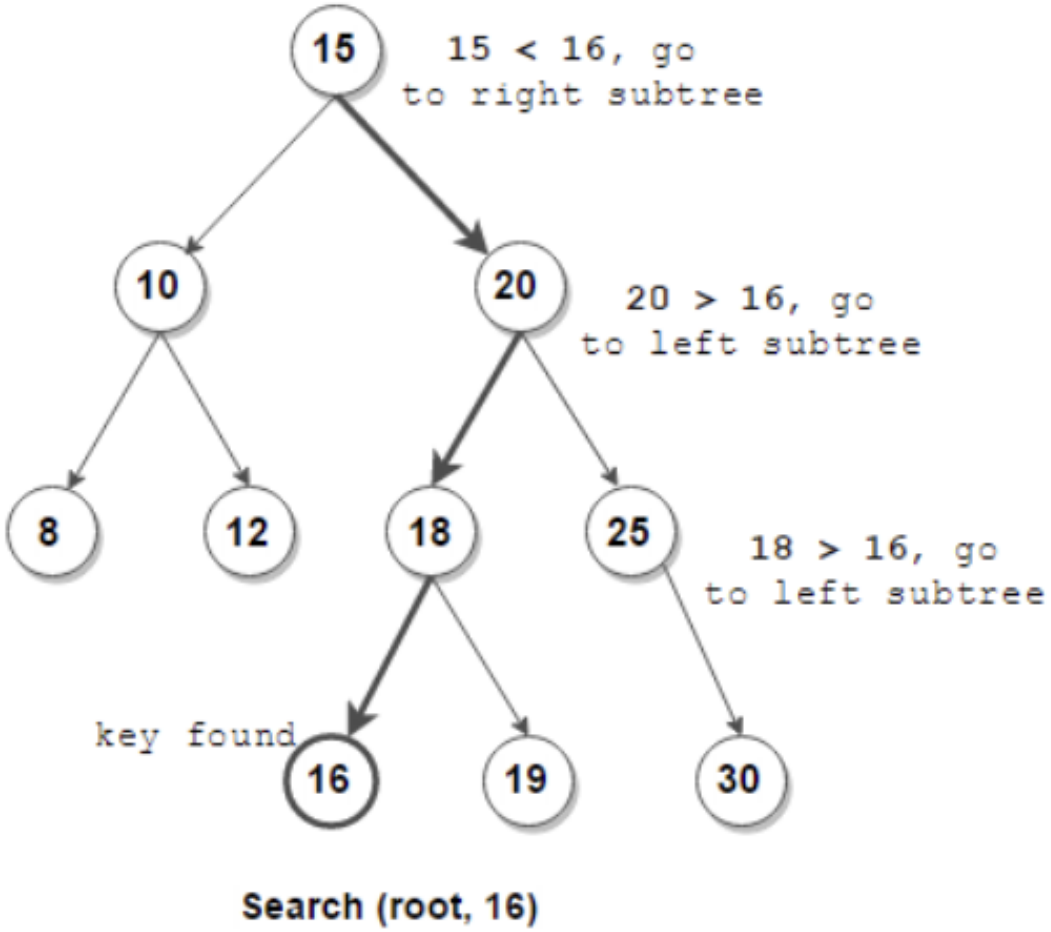
Hình 6: *Cây nhị phân.*

4.1 Binary Sreach

Binary Sreach làm theo bước tương tự như tìm kiếm nhị phân:

1. Nếu cây rỗng thì *return* không tìm thấy.
2. Nếu khóa có ở nút gốc, đã tìm thấy khóa.
3. Nếu khóa nhỏ hơn giá trị nút gốc, tìm kiếm ở cây con bên trái của cây.
4. Nếu khóa lớn hơn giá trị nút gốc, tìm kiếm ở cây con bên phải của cây.
5. Quá trình tìm kiếm được tiếp tục một cách đệ quy cho đến khi phần tử được tìm thấy hoặc tất cả các nút đã hết.

Sourcode



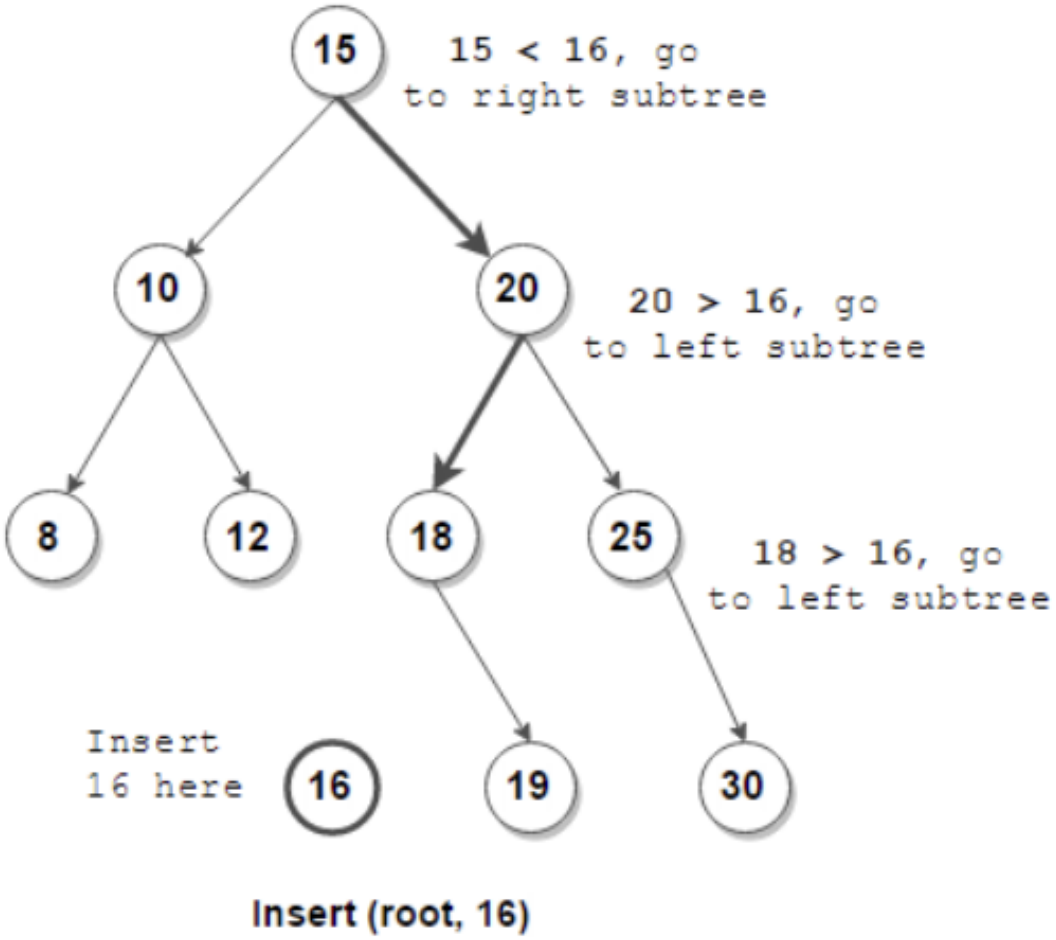
Hình 7: *Tìm kiếm nhị phân.*

4.2 Insertion Node

Các nút mới trong cây tìm kiếm nhị phân luôn được thêm vào ở vị trí lá. Insert làm theo bước:

1. Nếu nút null thì Chèn nút mới vào vị trí hiện tại.
2. Nếu giá trị của nút mới nhỏ hơn giá trị của nút gốc, di chuyển sang nút con trái của nút gốc.
3. Nếu giá trị của nút mới lớn hơn giá trị của nút gốc, di chuyển sang nút con phải của nút gốc.
4. Quá trình tìm kiếm được tiếp tục một cách đệ quy cho đến khi tìm thấy một nút null.

Sourcode

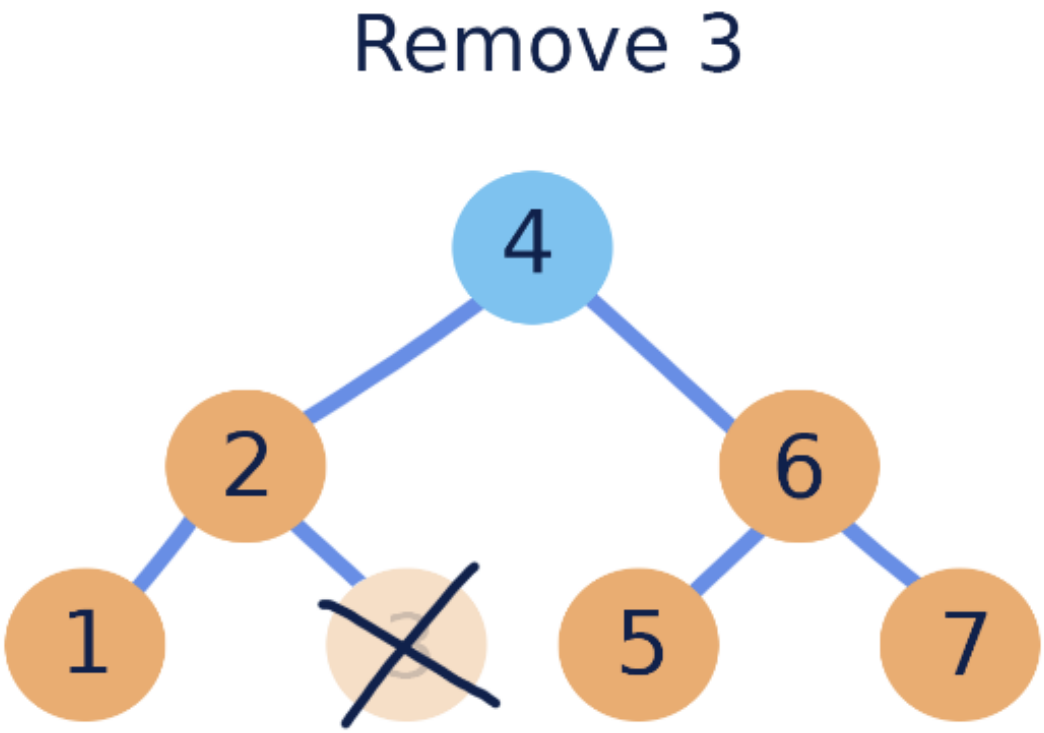


Hình 8: *Chèn node.*

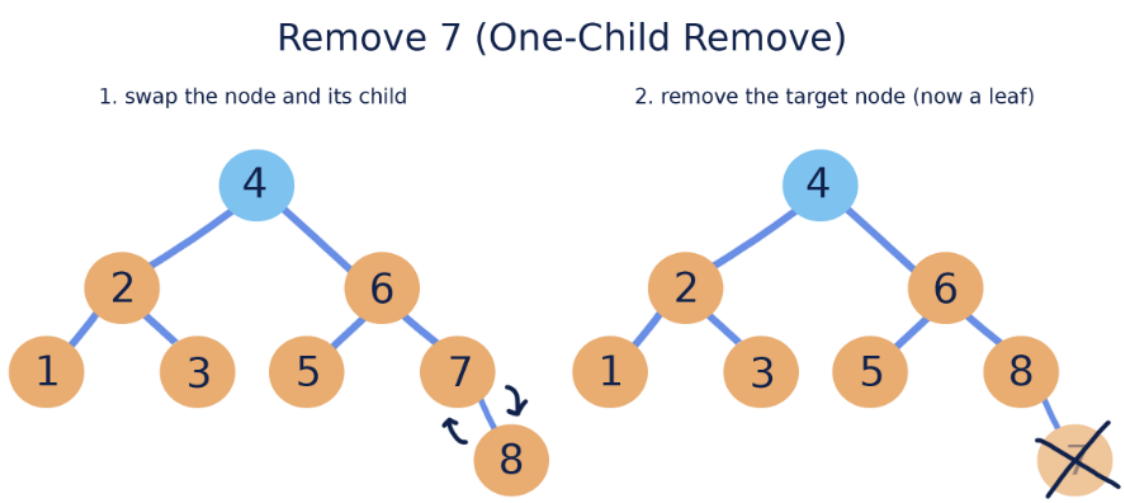
4.3 Delete Node

xóa làm theo bước:

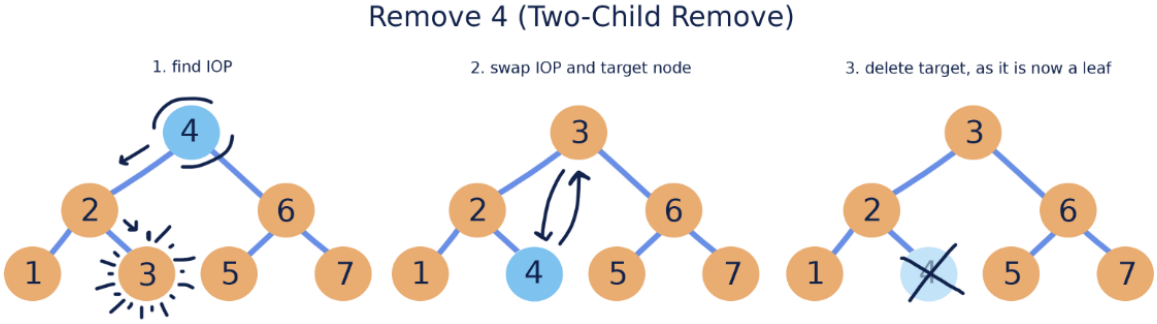
1. Tìm nút bạn muốn xóa.
2. TH1 : Nếu nút không có nút con, chỉ cần xóa nó.
3. TH2 : Nếu nút chỉ có một nút con, hãy thay thế nút bị xóa bằng nút con.
4. TH3 : Nếu nút có hai nút con, hãy thay thế nút bị xóa bằng nút con lớn nhất của cây con trái hoặc nút con nhỏ nhất của cây con bên phải



Hình 9: *xóa node trường hợp 1.*



Hình 10: *xóa node trường hợp 2.*



Hình 11: *xóa node trường hợp 3.*

5 AVL Tree

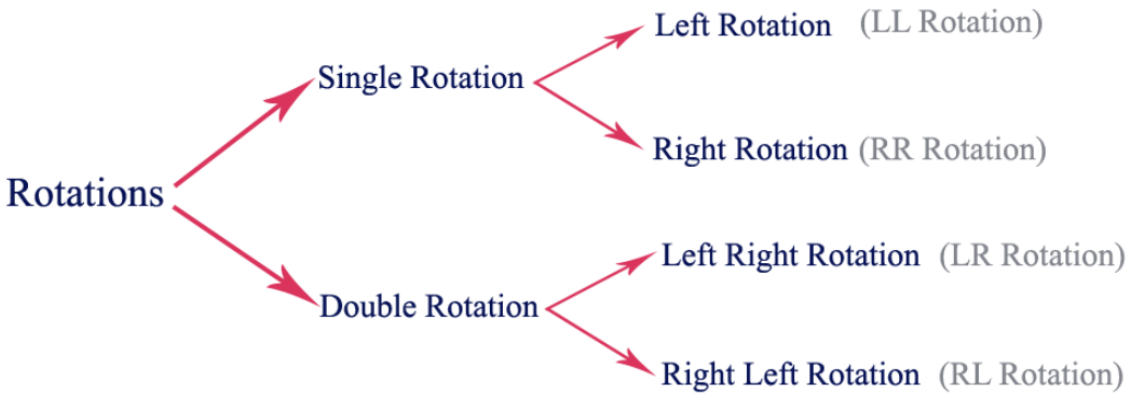
AVL Tree là cây tìm kiếm nhị phân (*Binary_Search_Trees*) và là cây cân bằng (*Balanced_tree*). Định nghĩa rõ hơn Cây AVL là một cây nhị phân tìm kiếm tự cân bằng trong đó độ cao của hai cây con trái và phải của bất kỳ nút nào không chênh lệch quá 1.

- left higher (LH): $H_L = H_R + 1$
- equal height (EH): $H_L = H_R$
- right higher (RH): $H_L = H_R - 1$

5.1 AVL Balance (cân bằng cây AVL)

Qua trình xử lí insert và delete sẽ ảnh hưởng tới sự cân bằng của cây AVL nến cần phải cân bằng lại.

- Left of Left: Cây và cây con đều left higher (LH)
- Right of right: Cây và cây con đều right higher (RH)
- Right of left: Cây left higher (LH), cây con bên trái right higher (RH)
- Left of right: Cây right higher (RH), cây con bên phải left higher (LH)

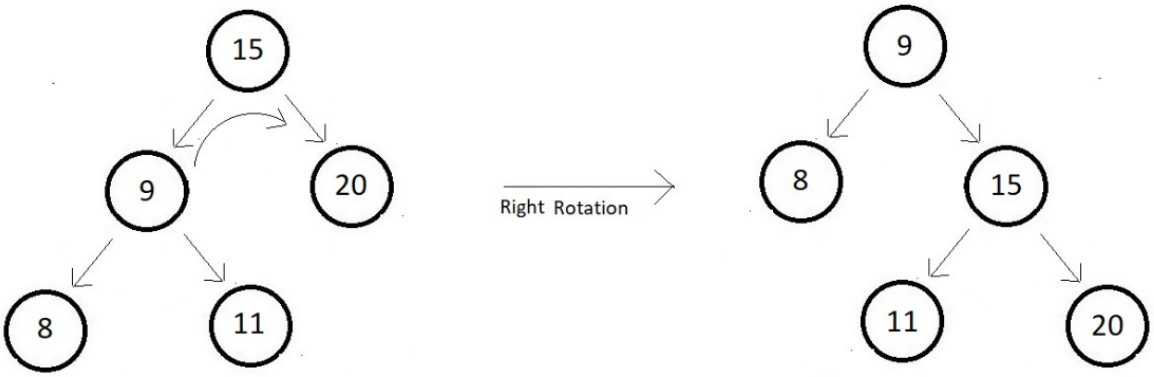


Hình 12: *Rotate.*

5.1.1 Rotate Right (quay phải)

1. nâng nút con bên trái lên làm nút cha rồi đưa nút cha xuống làm con bên phải của nút vừa đứa lên.
2. Cây con bên phải của nút vừa nâng lên thành con bên trái của nút cha vừa đưa xuống
3. Hệ số cân bằng của các nút bị ảnh hưởng được cập nhật1.

sourcode

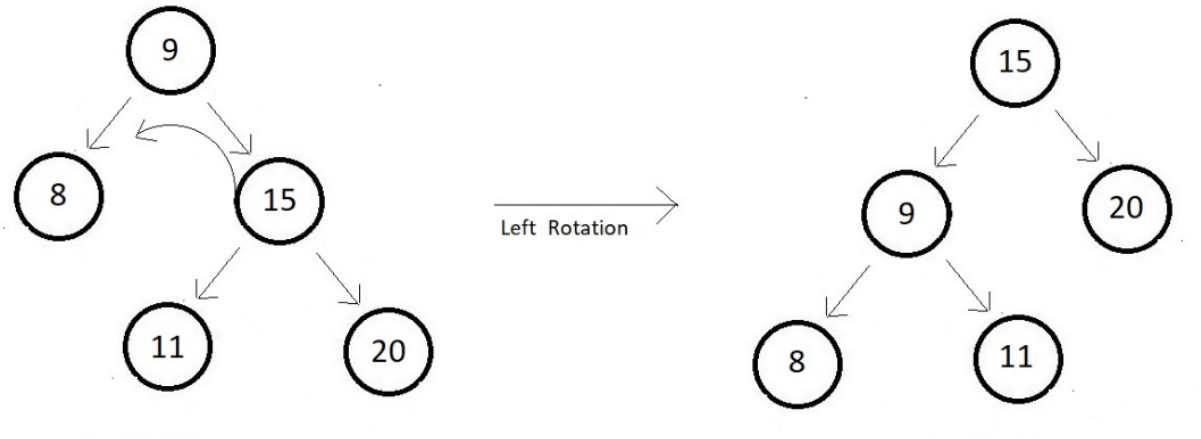


Hình 13: *quay phải.*

5.1.2 Rotate Left (quay trái)

1. nâng nút con bên phải lên làm nút cha rồi đưa nút cha xuống làm con bên trái của nút vừa đứa lên.
2. Cây con bên trái của nút vừa nâng lên thành con bên phải của nút cha vừa đưa xuống
3. Hệ số cân bằng của các nút bị ảnh hưởng được cập nhật.

sourcode

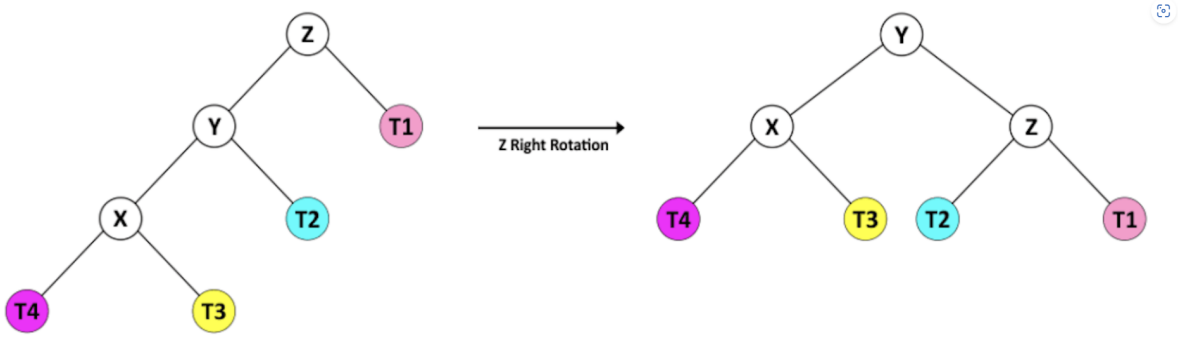


Hình 14: *quay trái.*

5.1.3 Left of Left

1. Quay phải nút Cha

sourcode

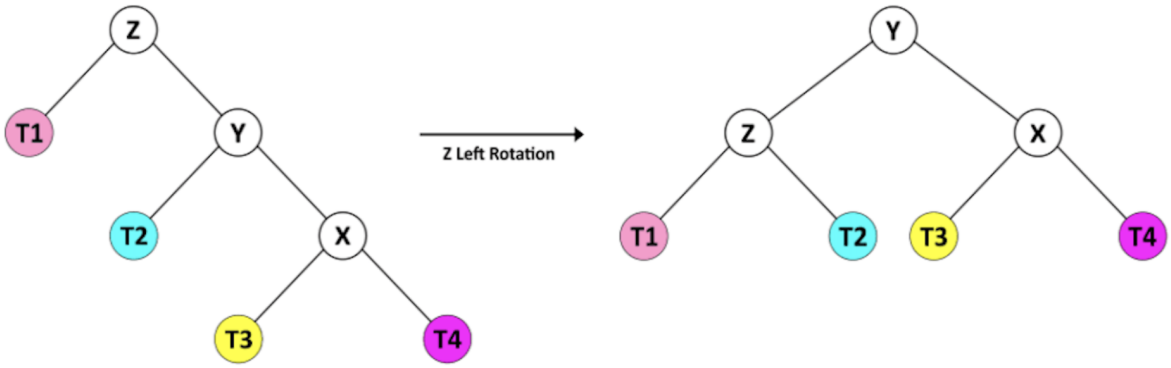


Hình 15: *Left of Left.*

5.1.4 Right of right

1. Quay trái nút Cha

sourcode

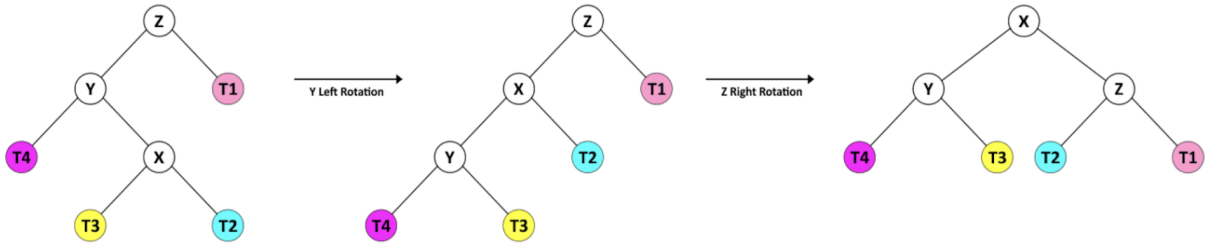


Hình 16: *Right of right.*

5.1.5 Right of left

1. Quay Trái cây con bên trái
2. Quay Phải nút gốc

sourcode

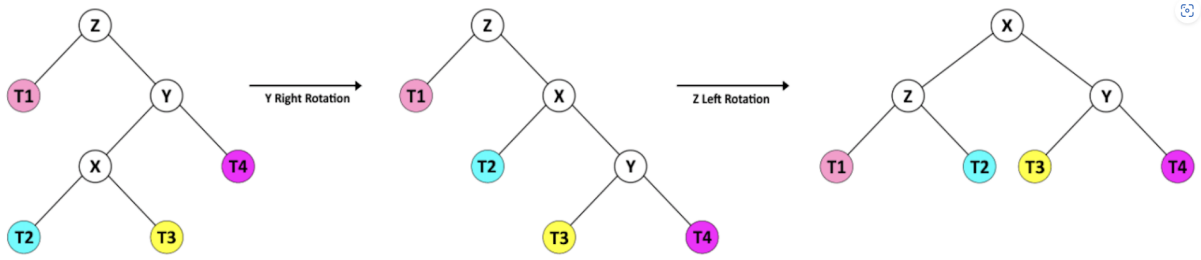


Hình 17: *Right of left.*

5.1.6 Left of Right

1. Quay Phải cây con bên Phải
2. Quay trái nút gốc

sourcode

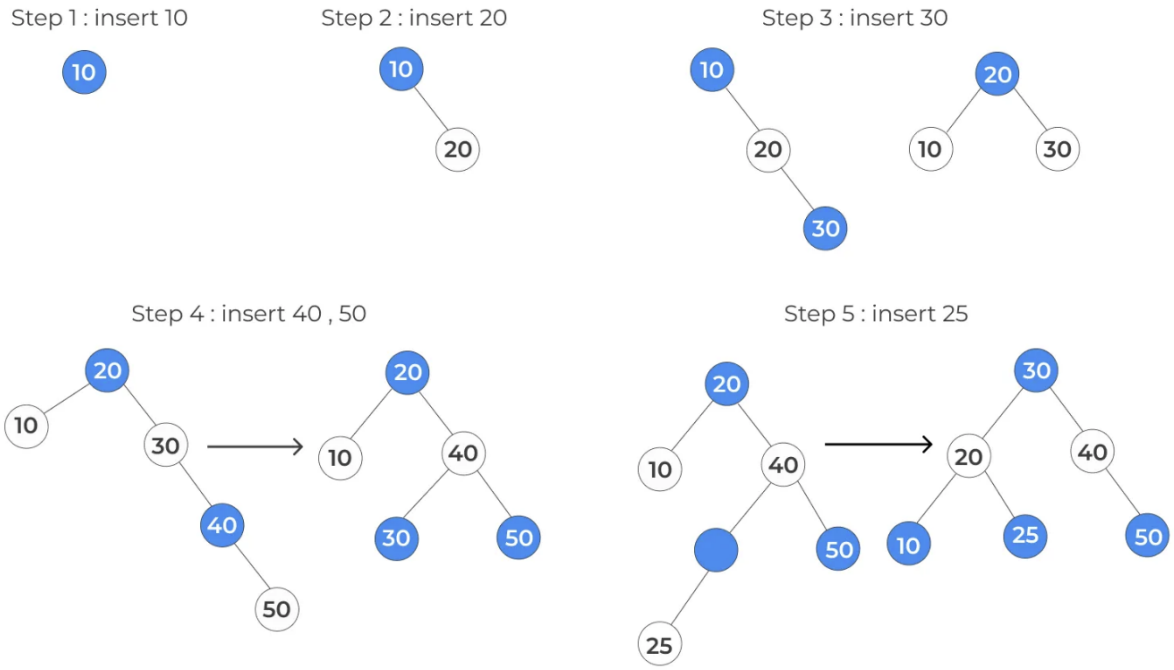


Hình 18: *Left of Right.*

5.2 AVL Insertion

1. insertion như cây tìm kiếm nhị phân
2. Cân bằng cây sẽ rời vào 4 TH left of left, left of right, right of left, right of right

sourcode

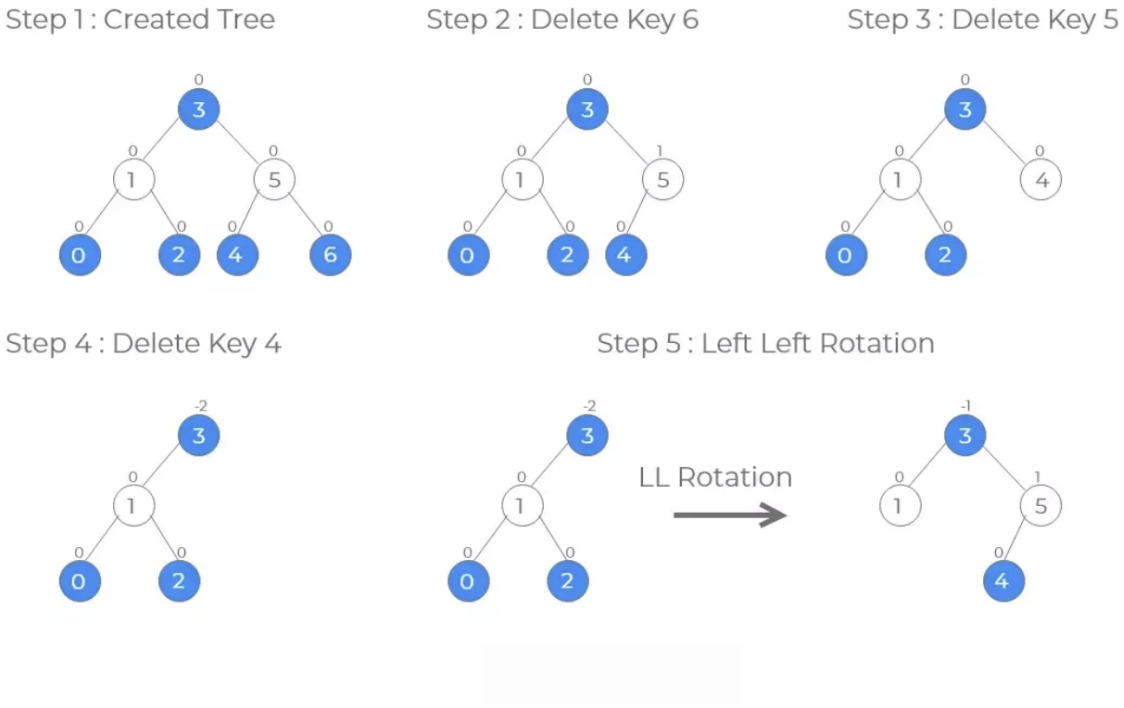


Hình 19: *AVL Insertion.*

5.3 AVL deletion

1. delete như cây tìm kiếm nhị phân
2. Cân bằng cây sẽ rời vào 4 TH left of left, left of right, right of left, right of right
3. Có một số trường hợp sẽ rơi vào cả 2 là left of left and right of left hay right of right and left of right thì thường chọn LL or RR cho dễ xử lí.

sourcode



Hình 20: *AVL deletion.*

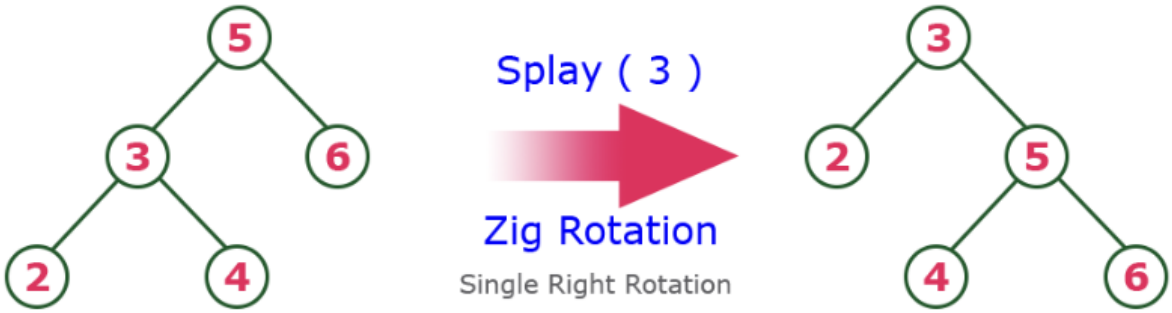
6 Splay Tree

Cây splay là một cây nhị phân tìm kiếm tự điều chỉnh (BST) giữ các nút được truy cập gần đây gần gốc cây. Điều này được thực hiện bằng cách thực hiện các vòng quay trên cây sau mỗi thao tác chèn, tìm kiếm hoặc xóa. Ưu điểm khi ta truy cập liên tục 1 phần tử độ phức tạp giảm xuống O(1)

6.1 Modification

6.1.1 Zig Rotation

Xoay Zig trong splay tree tương tự như quay phải (*Rotate_Right*) trong các phép quay AVL Tree. Trong xoay zig, mỗi nút di chuyển một vị trí sang phải từ vị trí của nó.
sourcode



Hình 21: *Xoay Zig.*

6.1.2 Zag Rotation

Xoay Zag trong splay tree tương tự như quay trái (*Rotate_Left*) trong các phép quay AVL Tree. Trong xoay zag, mỗi nút di chuyển một vị trí sang trái từ vị trí của nó.
sourcode

Hình 22: *Xoay Zag.*

6.1.3 Zig-Zig Rotation

Xoay Zig-Zig trong splay tree là một phép quay zig 2 lần. Trong vòng quay zig-zig, mỗi nút di chuyển hai vị trí sang phải so với vị trí hiện tại của nó.
sourcode

Hình 23: *Xoay Zig-Zig.*

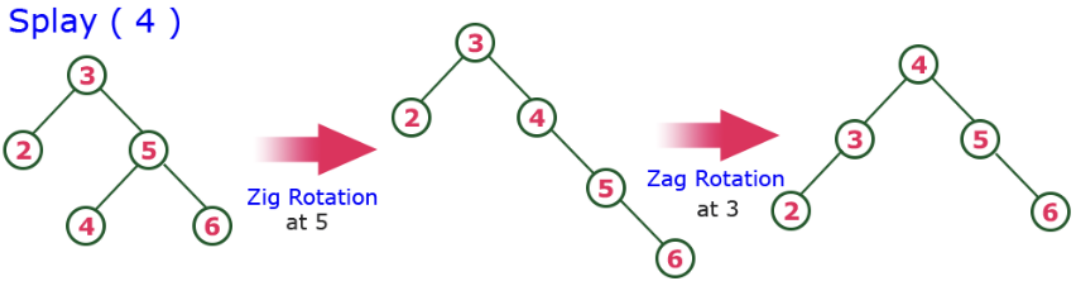
6.1.4 Zag-Zag Rotation

Xoay Zag-Zag trong splay tree là một phép quay zag 2 lần. Trong vòng quay zag-zag, mỗi nút di chuyển hai vị trí sang trái so với vị trí hiện tại của nó.
sourcode

Hình 24: *Xoay Zag-Zag.*

6.1.5 Zig-Zag Rotation

Xoay Zig-Zag trong splay tree là một chuỗi xoay zig theo sau là xoay zag. Trong xoay zig-zag, mỗi nút di chuyển một vị trí sang phải, sau đó là một vị trí sang trái từ vị trí hiện tại của nó
sourcode

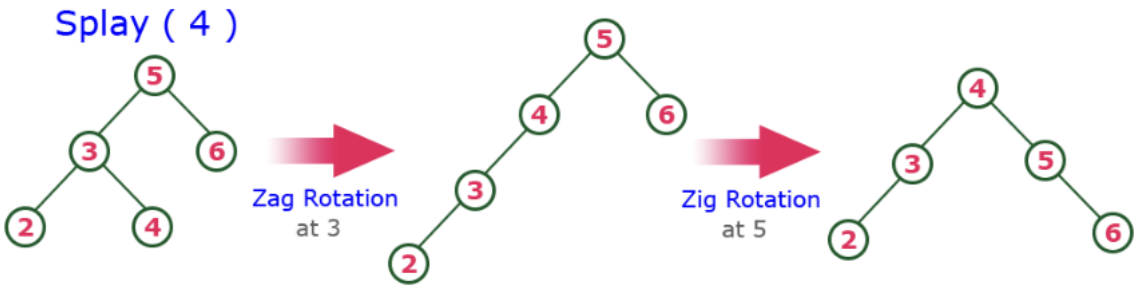


Hình 25: *Xoay Zig-Zag.*

6.1.6 Zag-Zig Rotation

Xoay Zag-Zig trong splay tree là một chuỗi xoay zag theo sau là xoay zig. Trong xoay zag-zig, mỗi nút di chuyển một vị trí sang trái, sau đó là một vị trí sang phải từ vị trí hiện tại của nó.

sourcode



Hình 26: *Xoay Zag-Zig.*

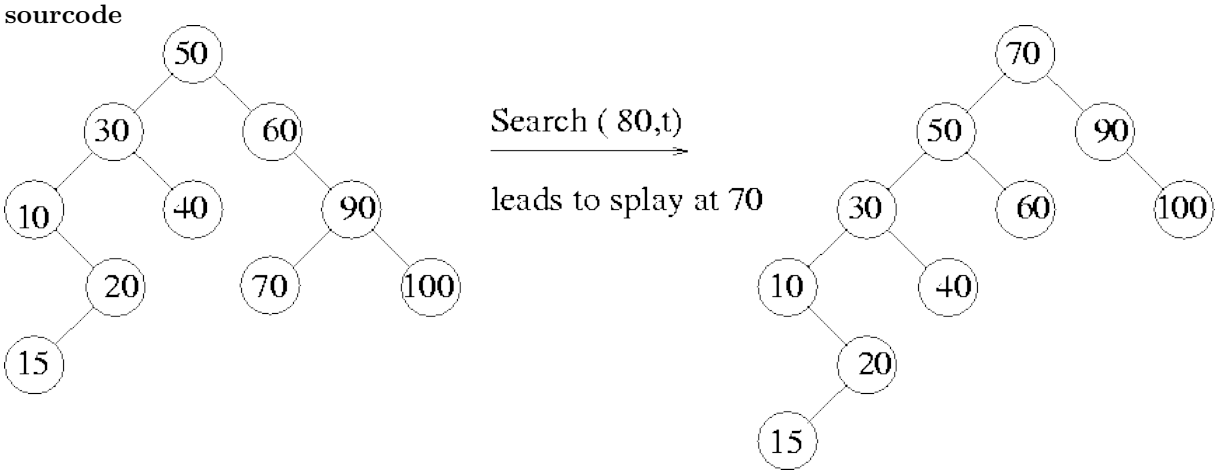
6.1.7 Splay

Tìm kiếm nút cần Splay và Splay nút đó lên thành nút gốc qua các thao tác zig, zag.

6.2 Sreach Splay

1. Tìm kiếm như cây tìm kiếm nhị phân
2. Splay node vừa tìm lên

sourcode

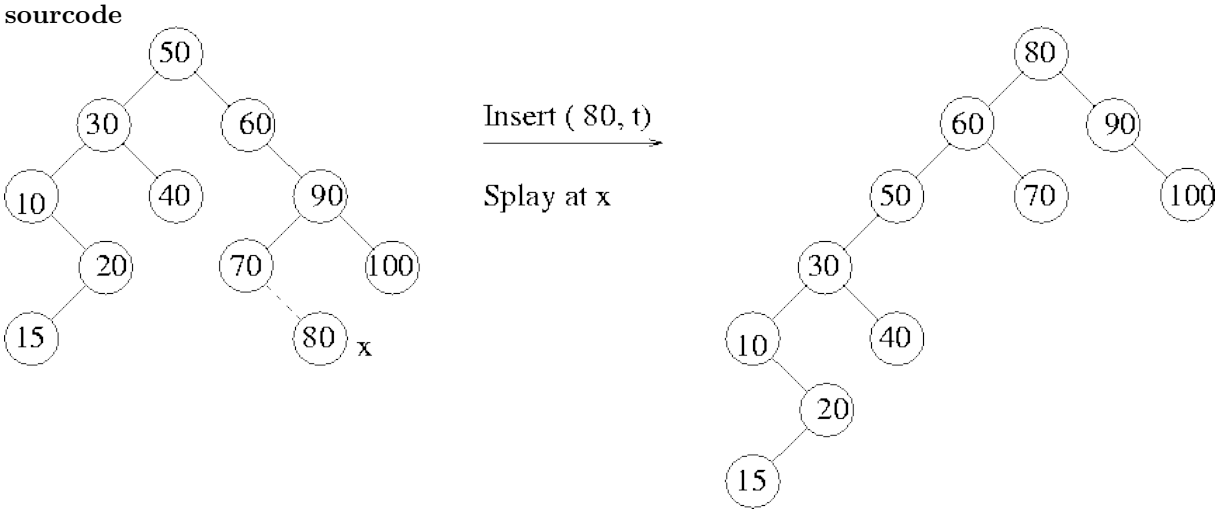


Hình 27: *tìm kiếm cây Splay.*

6.3 Insertion Splay

1. chèn như cây tìm kiếm nhị phân
2. Splay node vừa tìm lên

sourcode

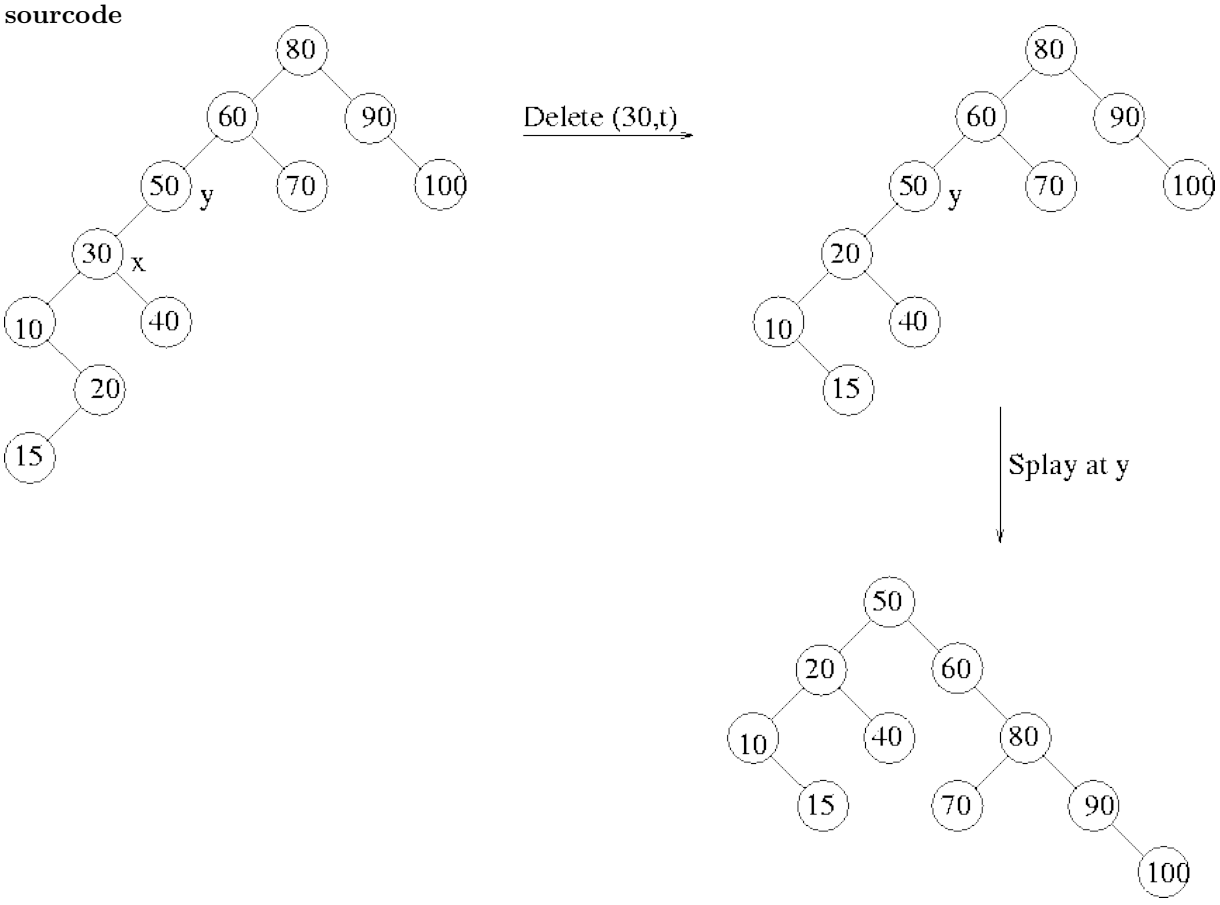


Hình 28: *chèn node Splay.*

6.4 Deletion Splay

1. tìm kiếm node cần xóa như cây tìm kiếm nhị phân
2. Splay node vừa tìm lên và xóa

sourcode



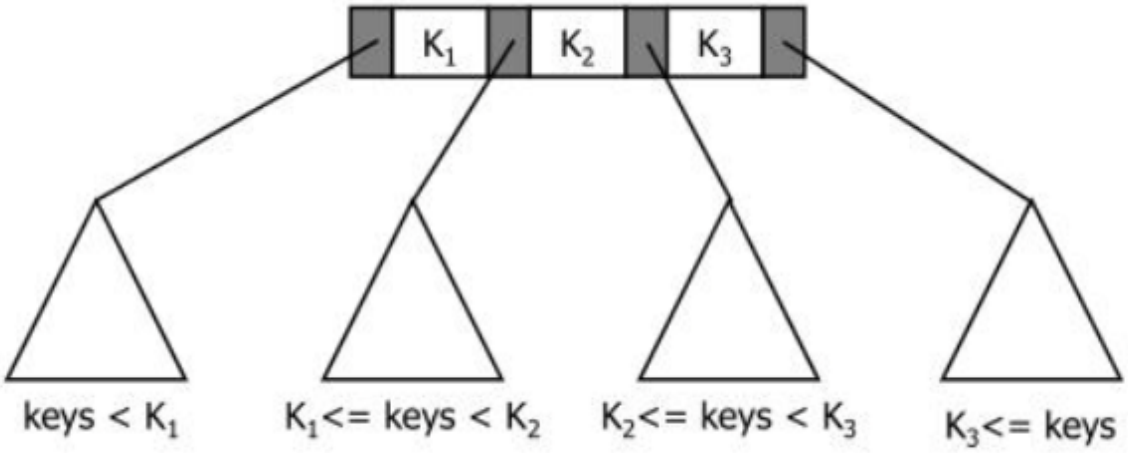
Hình 29: *xóa node Splay.*

7 B-Trees

7.1 Định nghĩa Multiway Trees

Cây Multiway Trees là cây có nhiều con có thể lớn hơn 2 của cây nhị phân.
M-Way Search Trees là một loại cây đa hướng trong đó mỗi nút có thể có tối đa m con. Số m được gọi là bậc của cây m-way. tìm kiếm giá trị của một nút được so sánh với giá trị của các con của nó để xác định thứ tự sắp xếp các nút. Cây m-way là một sự khái quát của cây tìm kiếm nhị phân, trong đó mỗi nút có thể có tối đa hai con.

- mỗi node có từ 0 đến m cây con
- Một node có $k < m$ cây con chứa k cây con và $k - 1$ dữ liệu.
- mọi data trong node của cảu cây con đều lớn hơn hoặc bằng với data bên trái node con đó là bé hơn data bên phải của node con đó
- data trong một node được xếp theo thứ tự
- tất cả các cây con đều là cây M-Way Search Trees.



Hình 30: *M-Way Search Tree.*