DSA

Date : Note:

KHANG VIET BOOK

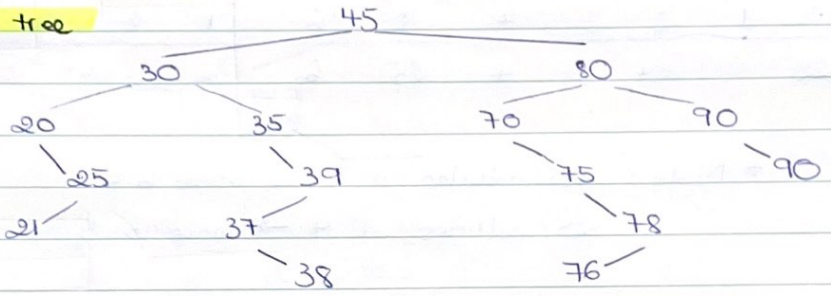
2019 Final

Binary tree

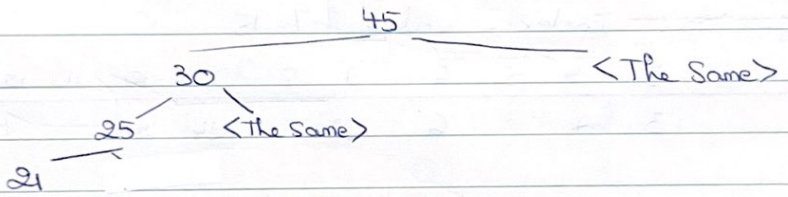
① Given a list of items: 45, 30, 80, 20, 35, 70, 90, 25, 21, 39, 37, 38, 75, 78, 76, 90

Trái nhỏ hơn, Phải to hơn

a) Insert all items one by one from left to right of the list into a binary search tree & draw the tree



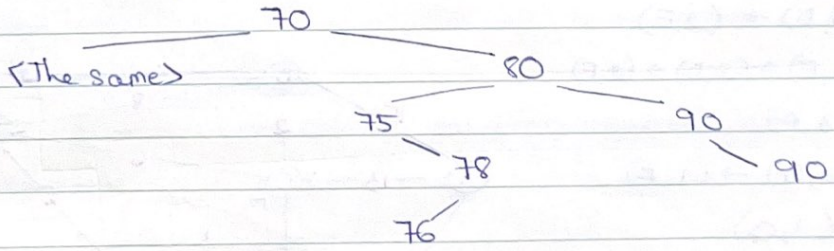
b) Delete item 20 from the tree and redraw the tree



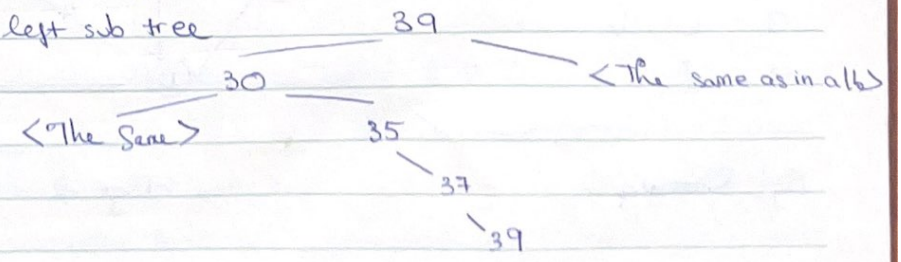
c) Delete item 45 from the tree and redraw the tree

* Using the tree from a) and b) is all acceptable

- Method 1: Using successor right sub tree → That is 70, smallest value in the right sub tree



- Method 2: Successor of left sub-tree → That is 39, the largest value in the left sub tree



Date: Note: KHANG VIET BOOK

② Hash table: Given a hash table of size 11

a) Assume that the linear probing algorithm is used to solve collision. Insert into the hash table the following items: 3, 4, 22, 24, 6, 25 and draw the hash table

Hash function: Index = value % 11

Key	0	1	2	3	4	5	6	7	8	9	10
Value	22		24	3	4	25	6				

* Note: 25 collides at 3 → move to 4
25 collides at 4 → move to 5

b) Change the hash table's size to 15, redraw it

Hash function: Index = value % 15

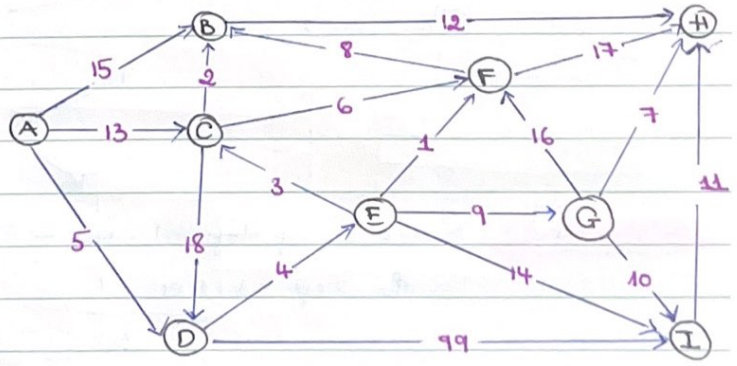
Key	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Value				3	4		6	22		24	25				

③ Graph – Elementary Algorithm

Given a graph G represented by the following list of successors for each pair (x, y): x is the weight of the edge; y is the terminal extremity of the trend

A: (5, D) → (2, E)
B: (4, E) → (2, D) → (2, F)
C: (3, F)
D: (2, G) → (1, F)
E: (1, G)
F: (3, H)
G: (3, H)
H: I

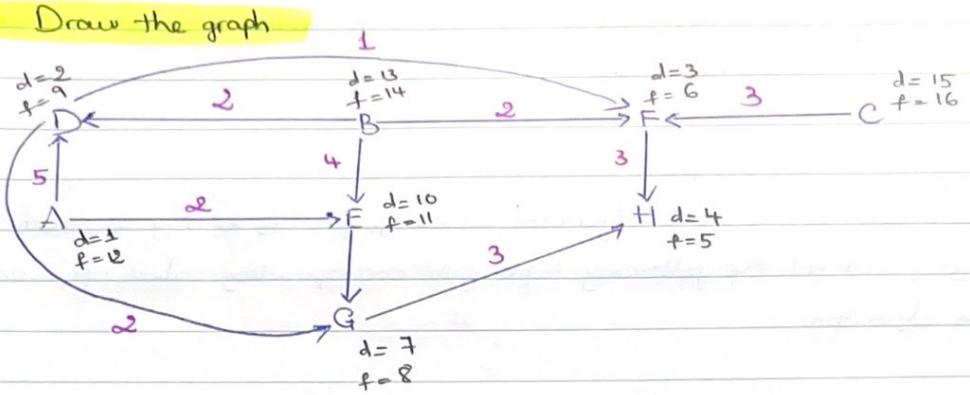
Fig 1. Successor of G



Figs 2. Graph Gs

Date : Note:

a) Draw the graph



b) Show an adjacency matrix of the graph

Nodes	A	B	C	D	E	F	G	H
A	0			5	2			
B		0		2	4	2		
C			0			3		
D				0		1	2	
E					0		1	
F						0		3
G							0	3
H								0

- Note : the others are ∞ (Do not have edges)

c) Topological sort the graph

(We perform Depth - First search on graph - question a)

→ Using pencil we have: { discovered time of each node
finishing time

→ Key point is that, each time you write down finishing time of each node, insert first that node to your answer list.

* Remember to follow alphebetical order (A, B, C, D, E, ...)

→ Answer list: C B A E D G F H

f=16 f=14 f=12 f=11 f=9 f=8 f=6 f=5

* By observation, it's simply that topological sort actually is the list of nodes, which is the result of DFS (Depth-First-search) algorithm, that has the finishing time from biggest to smallest value!

Date : Note: KHANG VIET BOOK

* Note: The graph must be acyclic (no circle - no loop), to perform topological sort

(4) Graph - Shortest path algorithm

Run the Dijkstra's algorithm on the graph G_5 in Fig. 2 starting from node C, and fill the following table with corresponding values after each step of the algorithm

Selected nodes	A	B	C	D	E	F	G	H	I
Node	A	B	C	D	F	F	G	H	I
-	∞	∞	0	∞	∞	∞	∞	∞	∞
C		(2,C)		(18,C)		(6,C)		∞	
B				(18,C)		(6,C)		(14,B)	
F				(18,C)				(14,B)	
H				(18,C)	∞				∞
D					(22,D)		∞		(117,D)
E							(31,E)		(36,E)
G									(36,E)
I	∞								

(5) Rooted trees with unbounded branching

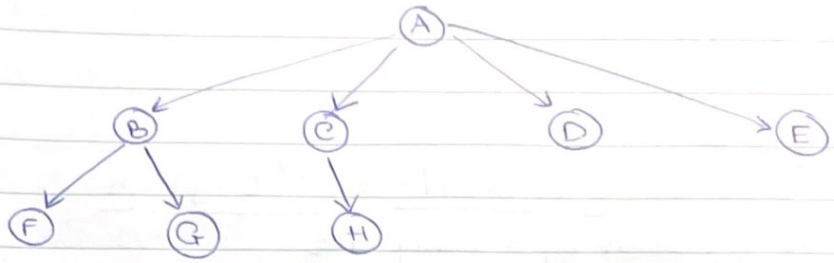
Given a tree whose node may have arbitrary numbers of children. There is a schema to represent that kind of tree name left-child, right sibling.

Each node x contains a parent pointer p, and two other pointers:

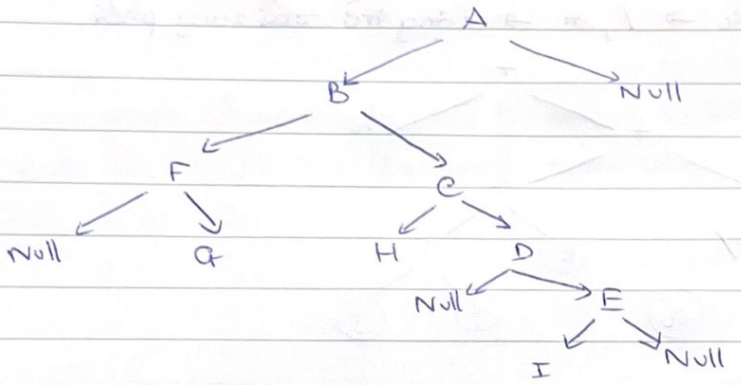
- left-child[x] points to the leftmost child of node x, and
- right-sibling[x] points to the sibling of x immediately to the right.

If node x has no children, the left-child[x] = NULL, and if node x is the rightmost child of its parent, then right-sibling[x] = NULL

Date : Note:



a) Draw a left-child, right-sibling representation of the tree T in sig. 3.



b) Write an O(n) non-recursive procedure that prints all the keys of an abitrary rooted tree with n nodes, where the tree is stored using the left-child, right-sibling representation

↳ Traversing tree without reursive requires at least
- 1 Stack or 1 Queue
- 1 Loop (Either For or While)

* Pseudo Code:

```
Traverse (node):
    Stack container = new Stack();
    Queue container = new Queue();
    while (!container.isEmpty()) {
        // Push left, move right
        container.push(node.left);
        print(node.value);
        node = node.right;
        // Print value also!
        if (node.left == node.right == null) {
            node = container.pop(); // OR container.dequeue()
        }
    } // end while loop
// End algorithm
```

OR

```
    // Push right, move left
    container.push(node.right);
    print(node.value);
    node = node.left;
```

=> Here's my higher-level traversal algorithm, print by layer using queue

@PeterTran @the-peter-page

Date: Note:

Week 7: Advanced Sorting

KHANG VIET BOOK

- **Shell Sort**: very efficient with medium-sized lists; depends on gap sequence, time complexity varies between O(N) & O(N^2)

41 15 82 | 5 65 19 | 32 43 8

↳ 5 15 8 32 43 19 41 65 82

↳ we have 3 subsets with 3 elements
↳ First sort
↳ Choose Shell's original sequence for interval increments: N/2, N/4, ...1

⇒ Efficiency: Range from $O(N^{3/2})$ down to $O(N^{7/6})$

```
public void sort (int arrayToSort []) {
    int n = arrayToSort.length;
    for (int gap = n/2; gap > 0; gap >= 2) {
        for (int i = gap; i < n; i++) {
            int key = arrayToSort [i];
            int j = i;
            while (j >= gap && arrayToSort [j-gap]
                > key) {
                arrayToSort [j] = arrayToSort [j-gap];
                j -= gap; }
            arrayToSort [j] = key; }}}
```

- **Partitioning Sort**: divide data into 2 groups

- Find an item (a)
  - in the left, pointed by leftPtr
  - bigger than pivot
- Find an item (b)
  - in the right, pointed by rightPtr
  - smaller than pivot
- Swap
- Repeat until 2 pointers meet

```
public int partitionIt (int left, int right, long pivot){
    int leftPtr = left - 1;
    int rightPtr = right + 1;
    while (true) {
    ↳ while (leftPtr < right &&
            theArray [++leftPtr] < pivot);
    ↳ while (rightPtr > left &&
            theArray [--rightPtr] > pivot);
    ↳ if (leftPtr >= rightPtr) ↳ break;
    ↳ else ↳ swap (leftPtr, rightPtr); }
    return leftPtr; }
```

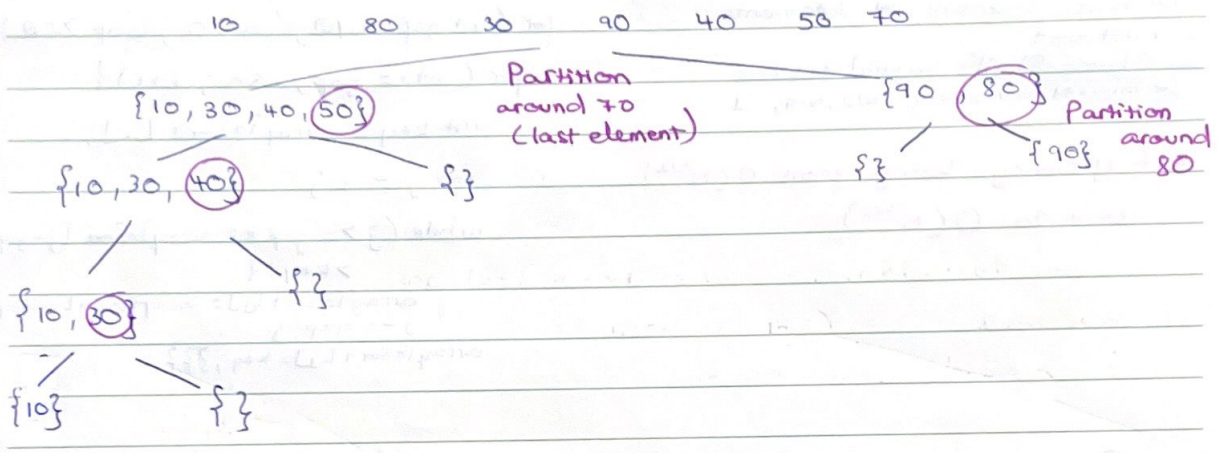
⇒ Efficiency: Two pointers start from 2 ends of array; Move towards each other; when they meet partition is complete → O(N)

Date: Note: KHANG VIET BOOK

- **Quick Sort**: most popular sorting algorithm, in most cases is the fastest, O(N*logN)

↳ Partition an array into 2 sub-arrays

- Then call itself recursively to quicksort each of these sub-arrays

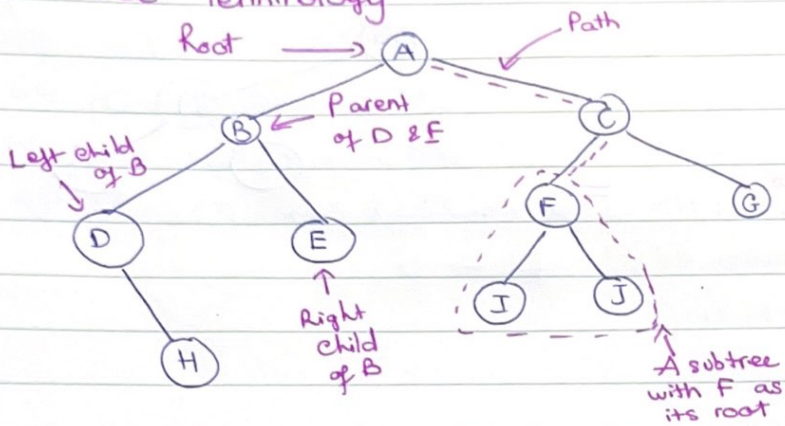


⇒ Efficiency: O(N*logN), a divide-and conquer algorithm

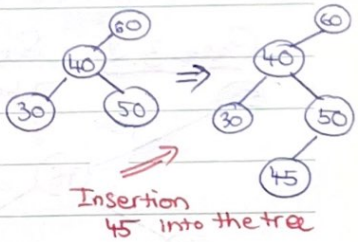
Date: Note:

Week 8 Binary Trees ♡

◦ Tree Terminology

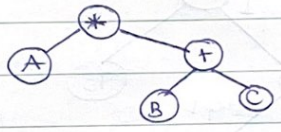


- Binary tree: every node have atmost 2 children
- Left chid : key < key of parent
- Right child: key >= key of parent



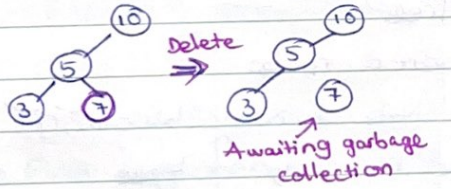
◦ Algebraic Expression

+ Infix: A*(B+C)
+ Prefix: *A+BC
+ Postfix: ABC+*

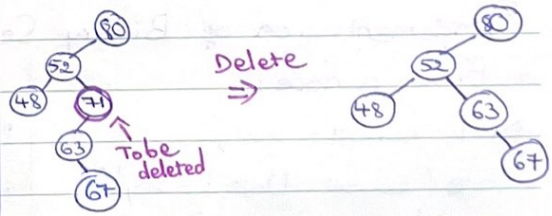


◦ Deleting a Node: 3 cases

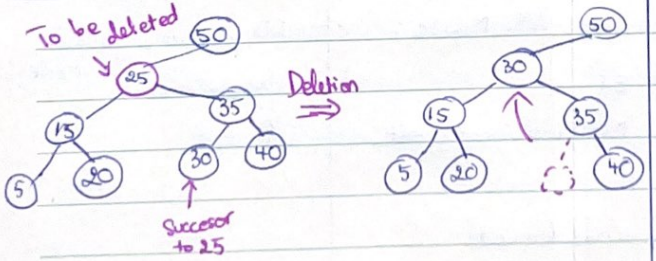
1. Node to be Deleted is a leaf (No children)



2. Node has 1 child



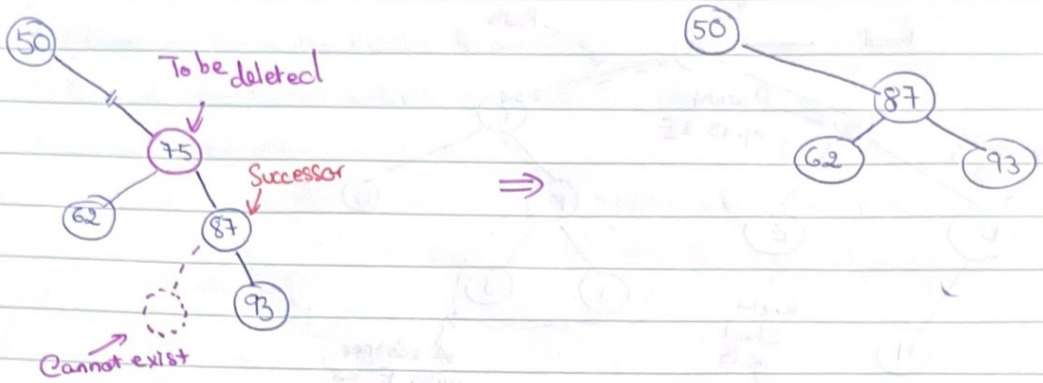
3. Node has 2 children



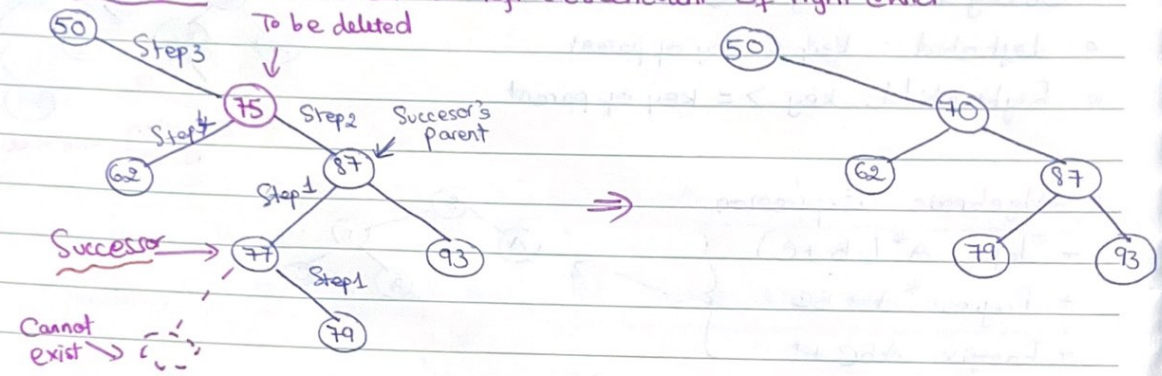
Finding Succesor:
The smallest of the set of nodes that are larger than given one.

Date: Note: KHANG VIET BOOK

Case 3.1 Successor is the right child of delNode



Case 3.2 : Successor is left descendant of right child



Implementation of Binary Search Tree

- Find a node:

```
Node current = root;
while (current.iData != key){
    if (key < current.iData)
        current = current.leftChild;
    else
        current = current.rightChild;
    if (current == null)
        return null;}
return current;
```

- Insert a node:

```
Node newNode = new Node();
newNode.iData = id;
newNode.dData = dd;
if (root == null)
    root = newNode;
else{
    Node current = root;
    Node parent;
    while (true){
        parent = current;
        if(id < current.iData){
            current = current.leftChild
            if (current == null){
                parent.leftChild = newNode;
                return;}
            else
                current = current.rightChild
            if(current == null){
                parent.rightChild = newNode;
                return;
            }
```

Date : Note:

KHANG VIET BOOK

- Traverse

```
private void inOrder(node localRoot){
    if (localRoot != null){
        inOrder(localRoot.leftChild);
        System.out.print(localRoot.iData + " ");
        inOrder(localRoot.rightChild);}}
```

- Find min

```
public Node minimum(){
    Node current, last;
    current = root;
    while (current != null){
        last = current;
        current = current.leftChild;}
    return last;}
```

Week 10: Hash Tables

Fast way to build & access tables (records)
provide nearly O(1) performance
Faster than trees

3 kinds of solving hashing: linear probing, quadratic & double hashing

- Algorithm: Given an input object,
  - Access this key value
  - Hash to a location in array
  - Store the object into this location (indexed by the hashed-to value)

* If object has a numeric key → use key as index, don't have to hash

1. Hashing

- To compress huge range of number into reasonable range
  - ↳ Modulo %: ArrayIndex = HugeNum % ArraySize ⇒ hash function
    - ↳ hash table (array to store data)

2. Collision Algorithms

- First Approach: Open Addressing
  - Search the array for empty cell, then insert new item there (ex: increase index by 1)
  - 3 schemes: linear probing, quadratic probing, double hashing
- Second Approach: Separate Chaining
  - + An item of array consists of linked list of words
  - + When collision occurs, new item is inserted in the list at that index

Example: Open Addressing-linear probe

ex:

Date : Note: KHANG VIET BOOK

Linear Probing: $h_i(x) = (Hash(x) + i) \% HashTableSize$

ex: Keys: 7, 36, 18, 62, table size: 11

First: 7 mod 11 = 7

↳ $\frac{7}{11} = 0.64$ ⇒ Take whole part of 0.64 times Divisor

$0.64 = 0 \times 11 = 0$

Take Divisor subtract the answer

$7 - 0 = 7$ ↑ index

Second: 36 mod 11 = 3

$\frac{36}{11} = 3.27$, $3 \times 11 = 33 \Rightarrow 36 - 33 = 3$

Third: 18 mod 11 = 7 → 7+1 = 8

$\frac{18}{11} = 1.6 \rightarrow 1 \times 11 = 11 \rightarrow 18 - 11 = 7$ (collision at index 7 move to 8)

Fourth: 62 mod 11 = 7

$\frac{62}{11} = 5.6 \rightarrow 5 \times 11 = 55 \rightarrow 62 - 55 = 7$ (collision at index 8 move to 9)

index	0	1	2	3	4	5	6	7	8	9	10
value				36		5		7	18	62	

Quadratic Probing: $h_i(x) = (Hash(x) + i^2) \% HashTableSize$

- If $h_0 = (Hash(x) + 0) \% HashTableSize \rightarrow h_1$
- If $h_1 = (Hash(x) + 1) \% HashTableSize \rightarrow h_2$
- If $h_2 = (Hash(x) + 4) \% HashTableSize \rightarrow h_3$ (↑ $i = 2^2$)

ex: keys: 7, 36, 18, 62

- 7 → $h_0(7) = 7 \bmod 11 = 7$ (index)
- 36 → $h_0(36) = 36 \bmod 11 = 3$
- 18 → $h_0(18) = 18 \bmod 11 = 7$ (collision)
  ↳ $h_1(18) = ((18 + 1 * 1) \bmod 11) = 8$ (index)
- 62 → $h_0(62) =$ 62 mod 11 = 7 → $h_1 = 8 \rightarrow h_2 = ((62 + 2^2) \bmod 11) = 0$)

index	0	1	2	3	4	5	6	7	8	9	10
value	62			36				7	18		

do size table = 11 nên cộng thôi

Date : Note:

Double Hashing : $h_i(x) = (Hash(x) + i * Hash2(x)) \% HashTableSize$

- If $h_0 = (Hash(x) + 0) \% HashTableSize$ is full, try h_1
- If $h_1 = (Hash(x) + 1 * Hash2(x)) \%$ HashTableSize is full, try h_2
- If $h_2 = (Hash(x) + 2 * Hash3(x)) \%$ HashTableSize is full, try h_3 ...

Code for Double Hashing

```
public int hashFunc1 (int key) { return key % arraySize; }
// Hashing algorithms
public int hashFunc2 (int key) { return 5 - key % 5; }
// non-zero, less than array size, different from hF1
// array size must be relatively prime to 5, 4, 3 and 2
public void insert (int key, DataItem) // Insert a DataItem
// Assume table not full
{
    int hashVal = hashFunction1 (key)   // Hash the key
    int stepSize = hashFunction2 (key)  // get Step Size, until empty cell or -1

    while (harshArray [hashVal] != null &&
           hashArray [hashVal] . getKey () != -1)
    {   hashVal += stepSize; // add the step
        hashVal %= arraySize; // for wraparound
    }
    hashArray [hashVal] = item;     // insert item
} // end Insert()
```

Note : Double Hashing

+ Require table size to be a prime number : 2, 3, 5, 7, 11...
+ If not a prime number :
  - Size = 15, step = 5
  - The probe sequence = 0, 5, 10, 0, 5, 10, ...

Date: Note: KHANG VIET BOOK

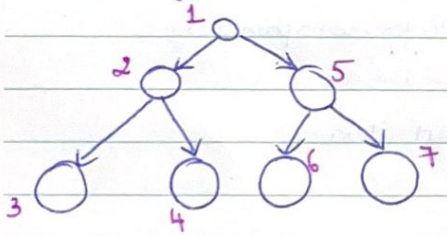
Separate Chaining: make each cell of hash table point to a linked list of records that have same hash function value

index	0	1	2	3	4	5	6
value	700	50 → 85 → 92		73 → 101			76

keys: 50, 700, 76, 85, 92, 73, 101

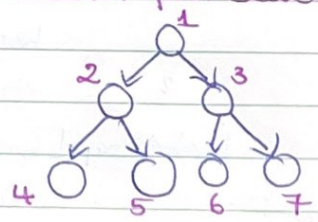
- 50 mod 7 = 1 (Shift + 6 (VINACAL Q+R: Quotient + Remainder)
- 700 mod 7 = 0
- 76 mod 7 = 6
- 85 mod 7 = 1
  - ↳ Collision at index 1
  - ↳ Create a link at index 1
- 92 mod 7 = 1
  - ↳ Collision at index 1
  - ↳ Create a link at index 1
- 73 mod 7 = 3
- 101 mod 7 = 3
  - ↳ Collision at index 3
  - ↳ Create link at index 3

Depth-first Search



- Traverse through left subtree(s) first, then traverse through the right subtree

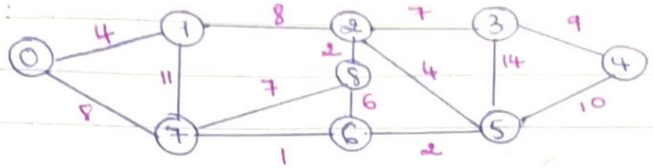
Breadth-first Search



- Traverse through one level of children nodes, then traverse through the level of grandchildren nodes...

Date: # Dijkstra's Algorithm

ex:



Adjacent matrix of the graph

0	4	0	0	0	0	0	8	0
4	0	8	0	0	0	0	11	0
0	8	0	7	0	4	0	0	2
0	0	7	0	9	14	0	0	0
0	0	0	9	0	10	0	0	0
0	0	4	14	10	0	2	0	0
0	0	0	0	0	2	0	1	6
8	11	0	0	0	0	1	0	7
0	0	2	0	0	0	6	7	0

Code:

```
import java.util.*; import java.lang.*; import java.io*;
class ShortestPath{
    static final int V = 9; // size from 0 → 8 = 9
    int minDistance (int dist[], Boolean sptSet[]){
        int min = Integer.MAX_VALUE, min_index = -1; // initiate min value
        for (int v = 0; v < V; v++)
            if (sptSet[v] == false && dist[v] <= min){
                min = dist[v]; min_index = v; }
        return min_index; }
    void printSolution (int dist[]){
        System.out.println (" Vertex \t\t " + dist[i]); }
        for (int i = 0; i < V; i++)
            System.out.println (i + " \t\t" + dist[i]); }
    void dijkstra (int graph[][], int src){
        int dist[] = new int[V];
        Boolean sptSet[] = new Boolean[V];
        for (int i = 0; i < V; i++){ dist[i] = Integer.MAX_VALUE;
                                     sptSet[i] = false; }
        dist[src] = 0;
        for (int count = 0; count < V-1; count++){
            int u = minDistance (dist, sptSet);
            sptSet[u] = true;
            for (int v = 0; v < V; v++)
                if (!sptSet[v] && graph[u][v] != 0 && dist[u] !=
                    Integer.MAX_Value && dist[dist[v] = dist[u] + graph[u][v])}
printSolution (dist); }
```

Date : Note:

KHANG VIET BOOK

```
// end dijkstra
// Driver method
public static void main(String[] args) {
    int graph[][] = new int[][] {{0,4,0,0,0,0,0,8,0}, ...
                                 ...{0,0,2,0,0,0,6,7,0}};
    ShortestPath t = new ShortestPath();
    t.dijkstra(graph, 0);}
}
```

Adjacent matrix of graph trang trc