CS F211: DATA STRUCTURES & ALGORITHMS (2ND SEMESTER 2024-25) HASH MAPS, BINARY SEARCH TREES

Chittaranjan Hota, PhD
Senior Professor, Computer Sc.
BITS-Pilani Hyderabad Campus
hota[AT]hyderabad.bits-pilani.ac.in

MAP ABSTRACT DATA TYPE (ADT)

What is a Map?

A Map is an abstract data type (ADT)

- stores key-value (k, v) pairs
- can you have duplicate keys?

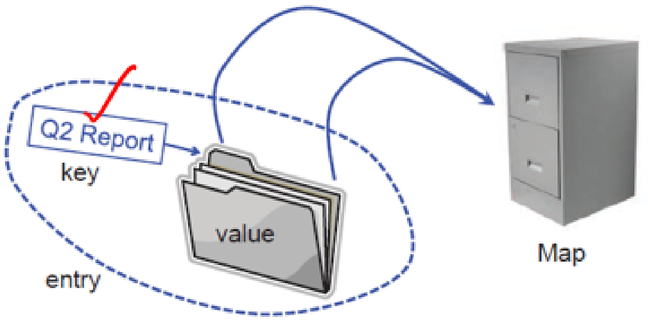
Let's say you want to implement a language dictionary.

Options: Vectors, LL, BST, Maps

Associative store or Associative container: Maps

```
#include <bits/stdc++.h>
using namespace std;

int main()
{
    priority_queue<int> pq;
    pq.push(90);
    pq.push(30);
    pq.push(90);
    cout << pq.top() << " ";
    pq.pop();
    cout << pq.top() << " ";
    pq.pop();
    return 0;
}
```

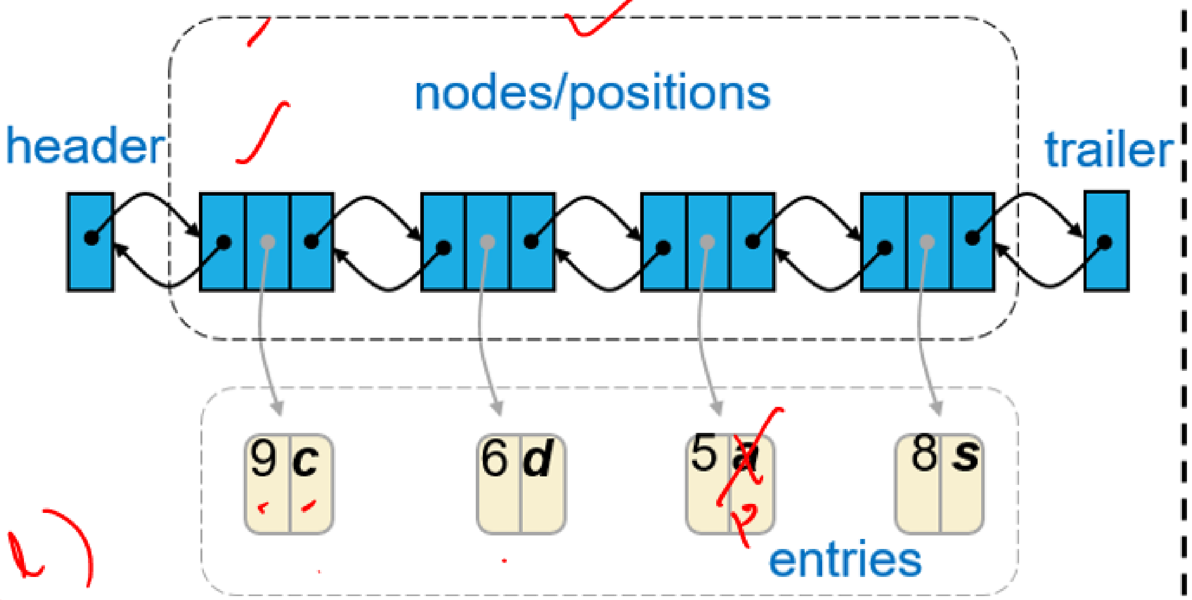
```
90 90
```



Does a priority queue allow duplicate keys?

Operation	*Output*	*Map*
empty()	**true**	$\emptyset$
put(5, A)	p_1 : [(5, A)]	
put(7, B)	p_2 : [(7, B)]	
put(2, C)	p_3 : [(2, C)]	
put(2, E)	p_3 : [(2, E)]	
find(7)	p_2 : [(7, B)]	
find(4)	end	
find(2)	p_3 : [(2, E)]	
size()	3	
erase(5)	–	
erase(p_3)	–	
find(2)	end	

IMPLEMENTATION OF MAPS



(using an unsorted list)

```
Algorithm put (k, v) {
  for each p in (S.begin(), S.end()) do
        if (p->key() == k) then
            p->keyValue(v);
            return p;
  p=S.insertBack ((k,v)) //no entry with key as 'k'
  n=n+1;
  return p;
}

Algorithm erase(k){
  for each p in (S.begin(), S.end()) do
     if (p→key() == k) then
        S.erase(p);
        n = n – 1;
}
```

STL MAPS IN C++

```cpp
#include <iostream>
#include <string>
#include <map>

using namespace std;
int main() {

    map < string, int> Students;

    Students.insert(std::pair<string, int>("Tushar", 91));

    Students.insert(std::pair<string, int>("Akshat", 94));

    Students.insert(std::pair<string, int>("Geet", 86));

    cout << "Map size is: " << Students.size() << endl;

    cout << endl << "Default map Order is: " << endl;

    for (map<string, int>::iterator it = Students.begin();
                    it != Students.end(); ++it) {

        cout << (*it).first << ": " << (*it).second << endl;
    }
}
```

Implementation: ?

```
Map size is: 3

Default map Order is:
Akshat: 94
Geet: 86
Tushar: 91
```

Search complexity: ?

Insertion/Deletion: ?

```cpp
#include <iostream>
#include <unordered_map>
using namespace std;

int main()
{

  unordered_map<string, int> mymap;

  // inserting values by using [] operator
  mymap["Rajesh"] = 10;
  mymap["Akash"] = 20;
  mymap["Shyamala"] = 30;
  mymap["Radhika"] = 30;
  mymap["Rohit"] = 30;
  mymap["Sachin"] = 30;

  // Traversing an unordered map
  for (auto x : mymap)
    cout << x.first << " " << x.second << endl
}
```

Implementation: ?

```
Sachin 30
Rohit 30
Shyamala 30
Radhika 30
Akash 20
Rajesh 10
```

Search complexity: ?

Insertion/Deletion: ?

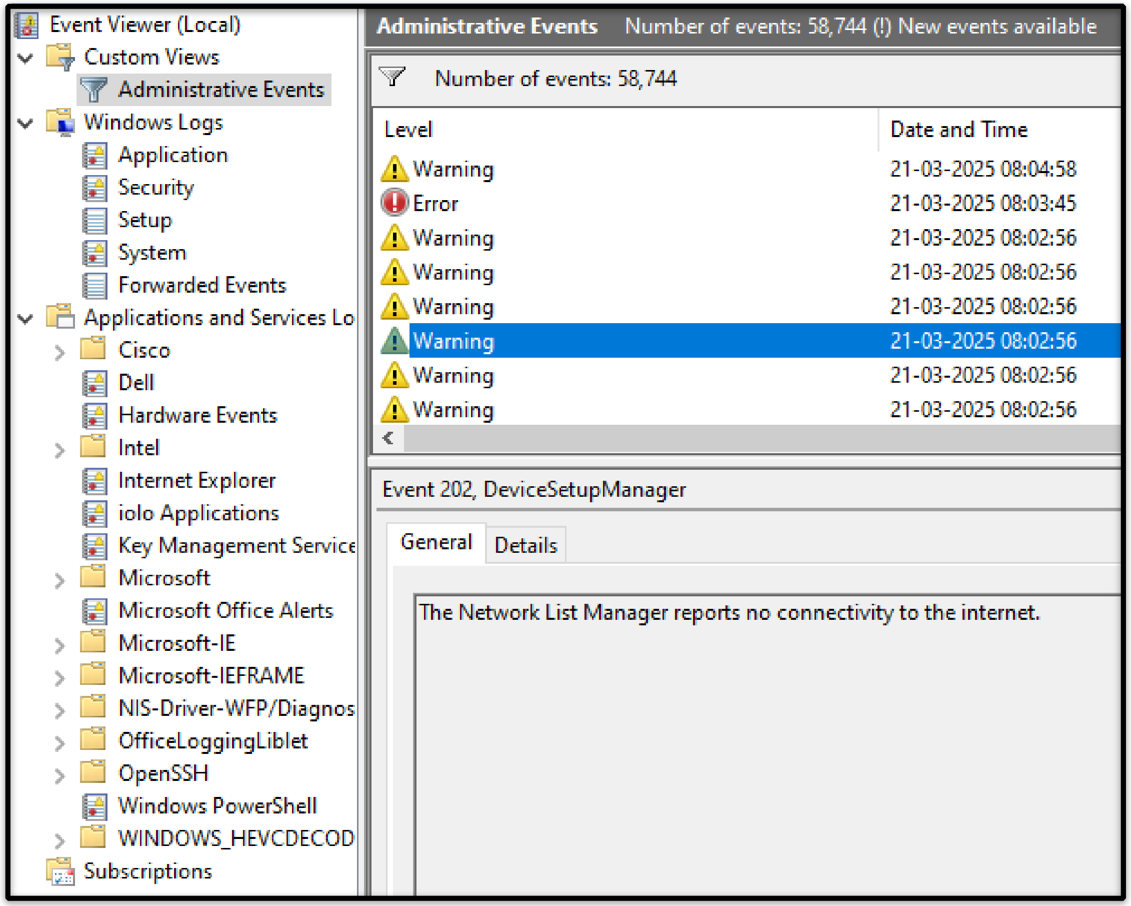
DICTIONARY ABSTRACT DATA TYPE (ADT)

-Like a map, a dictionary stores key-value pairs (k, v) called as entries.

-When a map insists that entries have unique keys, a dictionary allows for multiple entries to have the same key.

- Example: park: noun, and verb

Operation	Output	Dictionary
put(5, A)	p1:(5, A)	{(5, A)}
put(7, B)	p2:(7, B)	{(5, A), (7, B)}
put(2, C)	p3:(2, C)	{(5, A), (7, B), (2, C)}
put(8, D)	p4:(8, D)	{(5, A), (7, B), (2, C), (8, D)}
put(2, E)	p5:(2, E)	{(5, A), (7, B), (2, C), (8, D), (2, E)}
find(7)	p2:(7, B)	{(5, A), (7, B), (2, C), (8, D), (2, E)}
findAll(2)	{p3, p5}	{(5, A), (7, B), (2, C), (8, D), (2, E)}
erase(5)	—	{(7, B), (2, C), (8, D), (2, E)}



(Ex: A log-file or audit trail using unsorted sequence)

HASH TABLE (HASH MAP) DATA STRUCTURE

What is a Hash Table or Hash Map?

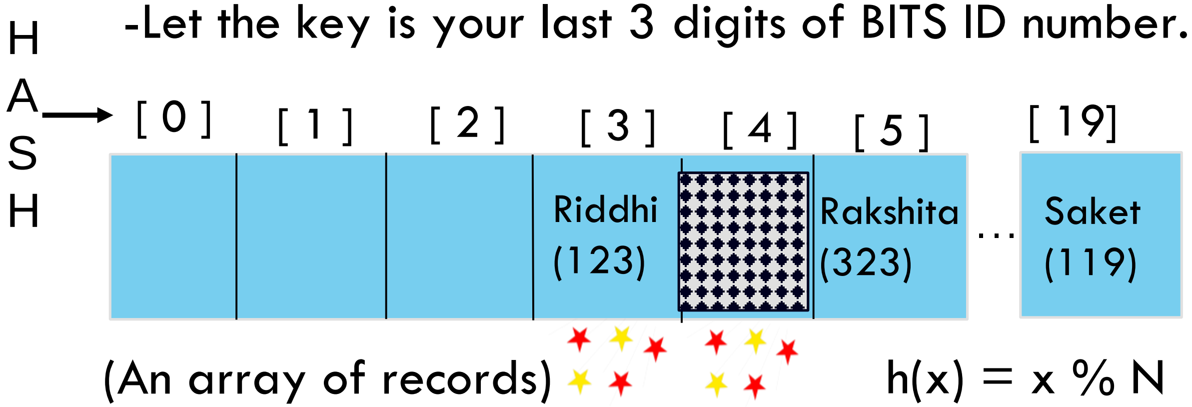
-A data structure used to implement the Map ADT

What are keys and values here?

-The simplest kind of hash table is an array of records.

-Let us consider an example that has 20 records.

-Let the key is your last 3 digits of BITS ID number.



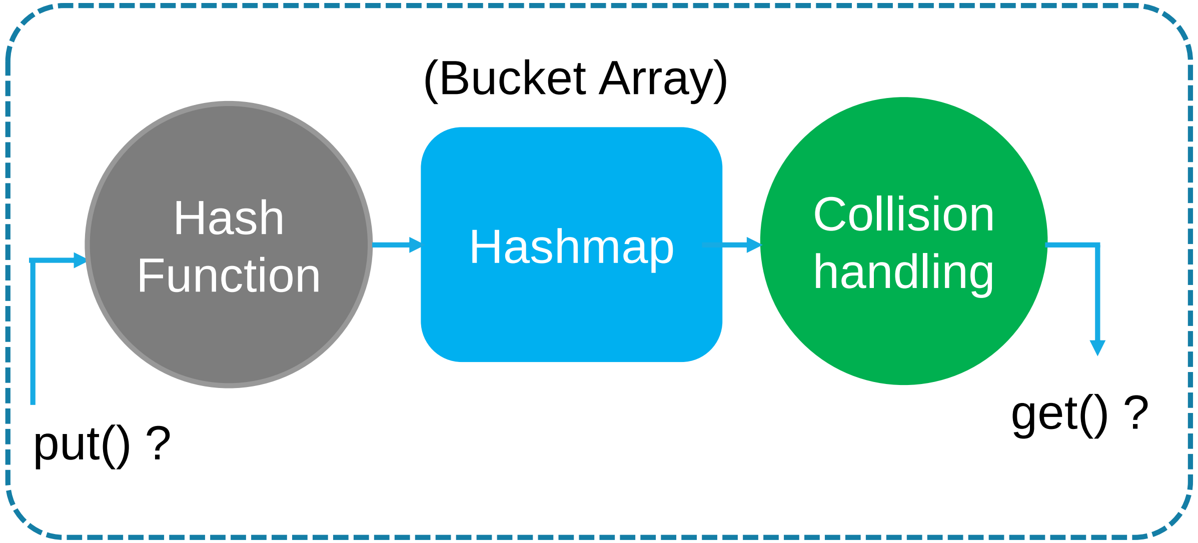
(An array of records)

$h(x) = x \% N$

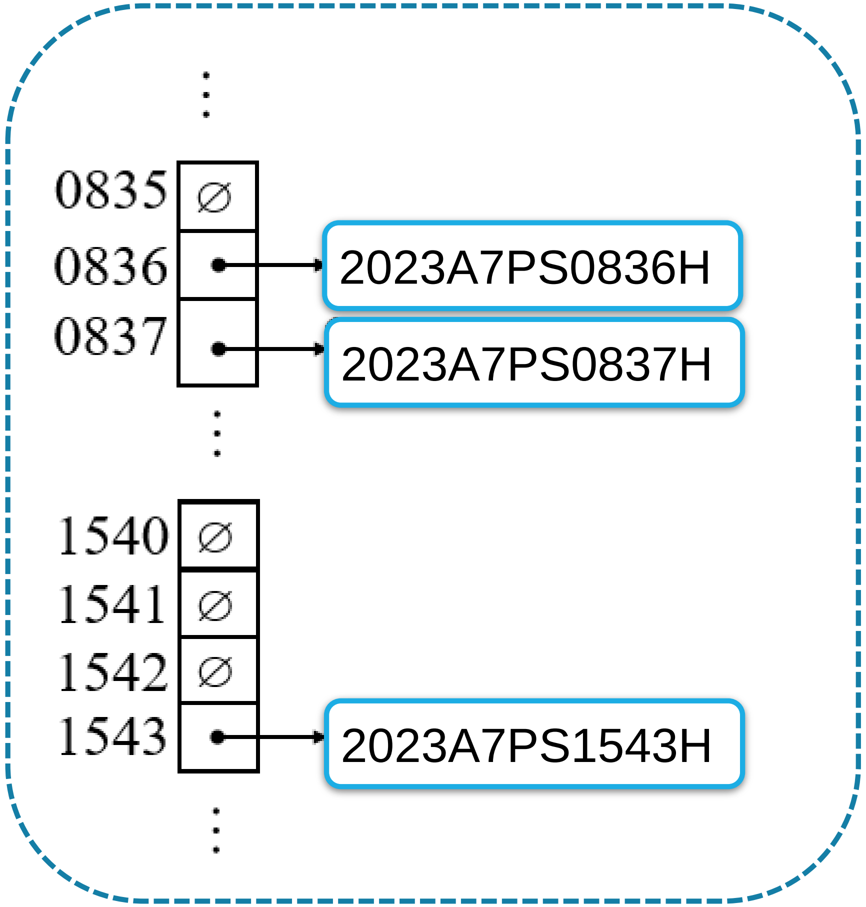
Rakshita with 323 will go where?

(Ex: Pigeonhole cabinet or Storage lockers at CSE dept)

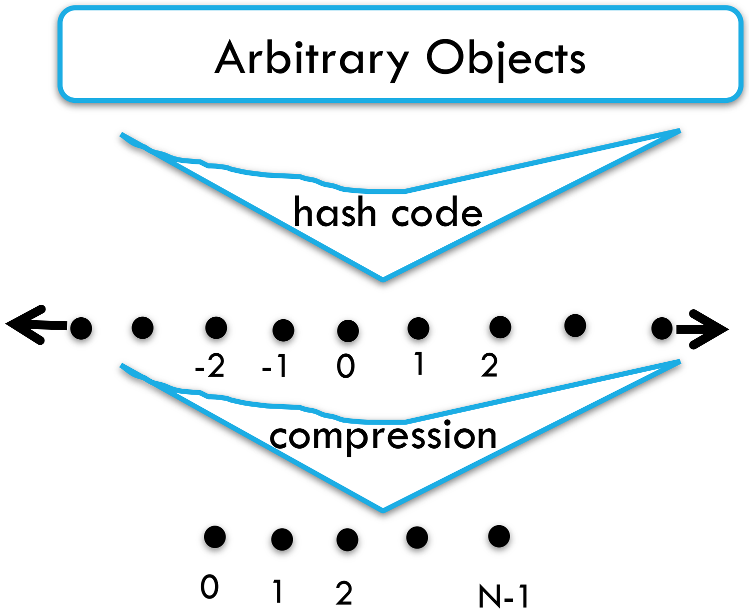
KEY COMPONENTS OF HASH MAP



Applications: Indexing in Oracle, and MySQL; Browser Cache (Chrome, Firefox); Grammarly (n-gram hashing); Google maps (Key: location ID, Values: neighbors, costs on links); Symbol Table (Lexical analyzers and Parsers in Compilers), Distributed Hash Tables (BitTorrent), etc…



HASH FUNCTIONS: CHARACTERISTICS



Common Techniques for Computing Hash codes:

- Integer cast, component sum, polynomial hash codes, cyclic shift hash codes. H("ab")=0x6162 = 24, 930 H("cat")=99('c')+97('a')+116('t')=312

Hash codes to indices in Hash table: H("hi")=(104('h')×31^1 +105('i')×31^0) %1000

- Division: $index = h(k) \bmod M$, where M is table size(Prime no.)
- Multiply-Add-Divide (MAD): $index = (a.h(k) + b) \bmod M$

hash<string> hashFunc;

size_t hashCode = hashFunc(text);

Hash Code of "Hello": 1595419238140062680

string text = "Hello";

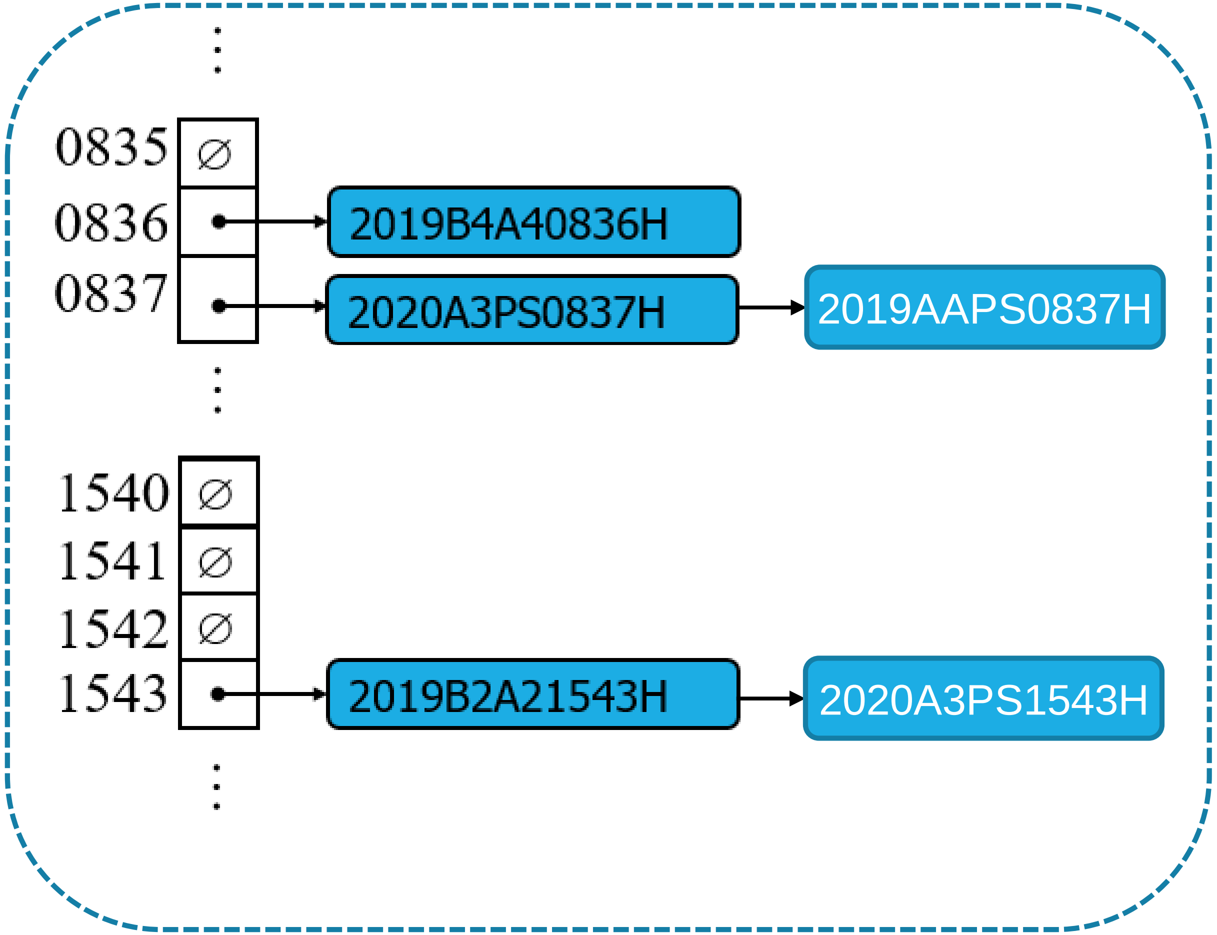
string hashValue = sha256(text); (Hash value of Hello)

185f8db32271fe25f561a6fc938b2e264306ec304eda518007d1764826381969

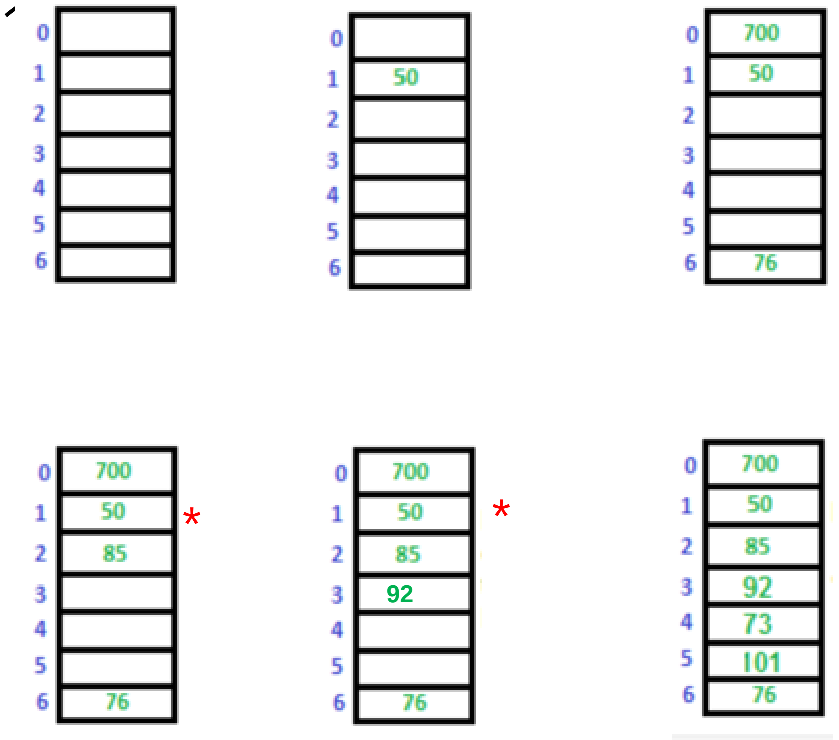
Easy to compute: It should be easy to compute. Uniform distribution: It should provide a uniform distribution across the hash table and should not result in clustering. Less collisions: Collisions occur when pairs of elements are mapped to the same hash value. These should be avoided.

perform a cyclic left shift by 5 bits, then XOR with the byte

COLLISION HANDLING TECHNIQUES

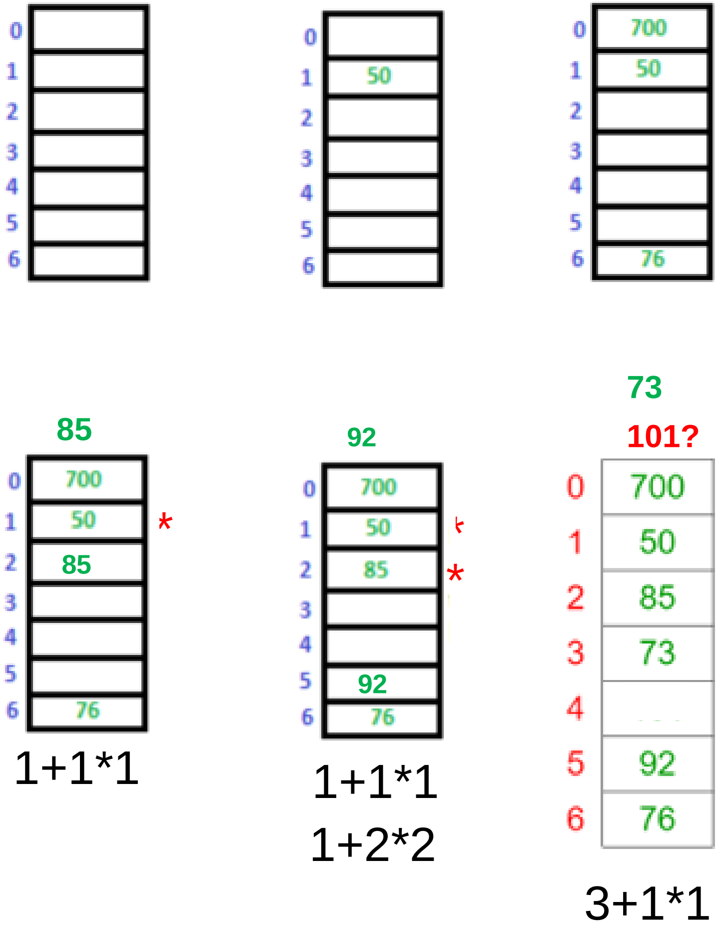


(Open Hashing: Separate Chaining)



(Example: h(x) = x mod 7)

(Closed Hashing: Linear Probing)

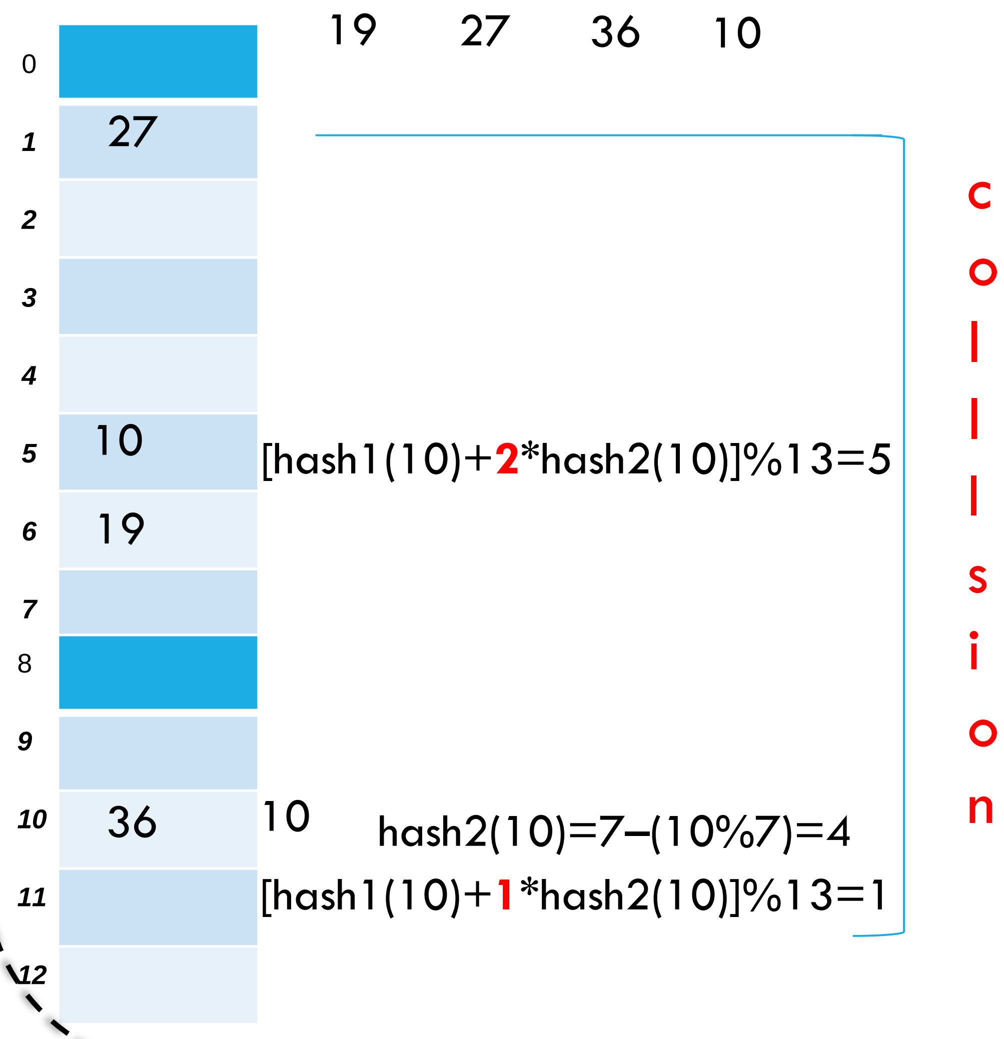


[Ex: h'(x) = x mod 7]

$$h(x) = (h'(x) + i^2) \bmod 7$$

(Closed Hashing: Quadratic Probing)

An Example: hash1(key) = key%13, hash2(key) = 7 – (key % 7)



(Double Hashing)

HOW IMPORTANT IS TABLE SIZE?

- Linear probing will always find an open spot if one exists (It might be a long search but we will find it).
- However, this is <u>not the case</u> with quadratic probing unless you take care in selecting the table size correctly.
- 98, 34, 50, 66, 82
- h’(x) = x mod 16

82 mod 16=2 ➡ does not get inserted as (2+4*4) mod 16 = 2 which is full.

Hence, in order to guarantee that your quadratic probes will hit every single available spot eventually, your table size must meet these requirements: - a prime number, - never be > half full, - >1.3

0		
1		
2	98	
3	34	2+1*1
4		
5		
6	50	2+2*2
7		
8		
9		
10		
11	66	2+3*3
12		
13		
14		
15		

IMPLEMENTATION OF HASH MAP

```cpp
#include <iostream> #include <vector>
#include <list>  #include <string>
using namespace std;
class HashMap { //Hash Table Class
private:
    static const int TABLE_SIZE = 10;
    vector<list<pair<string, int>>> table;
    int hashFunction(const string &key) {
        hash<string> hashFunc;
        return hashFunc(key) % TABLE_SIZE;
    }
public:
    HashMap() {
        table.resize(TABLE_SIZE);
    }
    void insert(const string &key, int value) {
        int index = hashFunction(key);
        // Check if key already exists, update it
        for (auto &p : table[index]) {
            if (p.first == key) {
                p.second = value;
                return;
            }
        }
        // If key not found, insert new key-value
        table[index].emplace_back(key, value);
    }
void remove(const string &key) {
 int index = hashFunction(key);
 auto &chain = table[index];
 for(auto it=chain.begin(); it!=chain.end();++it){
    if (it->first == key) {
      chain.erase(it);
      return;
    }
 }
 cout <<  key << " not found.";
}
int search(const string &key) {
        int index = hashFunction(key);
        for (auto &p : table[index]) {
            if (p.first == key)
                return p.second;
        }
        throw runtime_error("Key not found");
    }
void display() {
        for (int i = 0; i < TABLE_SIZE; i++) {
            cout << "Index " << i << ": ";
            for (auto &p : table[i])
              cout <<  p.first << p.second;
            cout << endl;
        }
    }
};
```

```cpp
int main() {
  HashMap map;
  map.insert("Riddhi", 25);
  map.insert("Aryan", 30);
  map.insert("Mahadevan", 22);
  map.insert("Saket", 40);
  map.insert("Purva", 29);
  map.display();
  map.remove("Purva");
  map.display();
  return 0;
}
```

Lab11 Next week

```
Index 0:  [Mahadevan -> 22]
Index 1:  [Riddhi -> 25]
Index 2:  [Aryan -> 30]
Index 3:  [Purva -> 29]
Index 4:
Index 5:
Index 6:
Index 7:
Index 8:
Index 9:  [Saket -> 40]
Index 0:  [Mahadevan -> 22]
Index 1:  [Riddhi -> 25]
Index 2:  [Aryan -> 30]
Index 3:
Index 4:
Index 5:
Index 6:
Index 7:
Index 8:
Index 9:  [Saket -> 40]
```

CACHING IMPLEMENTED USING HASH TABLES

```
class FileManager {
private:
    int cap, timer;
    HashMap<int> map;
    vector<CacheItem*> cache;
    int freeSpace() {
      int idx = 0;
      for (int i = 0; i < cap; i++) {
        if (cache[i]->lastUsedTime == -1) {
           cout << "Free space found at " << i << '\n';
           return i;
        }
        if(cache[i]->lastUsedTime<cache[idx]->lastUsedTime) idx = i;
      }
      cout << "No free space found, removing " << cache[idx]->name
                                          << " to clear index " << idx << '\n';
      map.remove(cache[idx]->name);
      return idx;
    }
};
```

Lab11 Next week

```
Adding hello.txt ...
Free space found at 0
>> Hello World!

Adding bio.txt ...
Free space found at 1
>> My name is X

Adding lorem.txt ...
Free space found at 2
>> Lorem ipsum dolor sit amet, consectetur adipiscing elit

Fetching hello.txt ...
hello.txt found in cache at index 0
>> Hello World!

Adding sample.txt ...
No free space found, removing bio.txt to clear index 1
>> The quick brown fox jumps over the lazy dog

Fetching bio.txt ...
bio.txt not found in cache
No free space found, removing lorem.txt to clear index 2
>> My name is X
```

DNS LOOKUP USING HASH MAPS

- Each DNS server keeps track of its immediate children servers using a hashmap.

```
C:\Users\C R Hota>nslookup td.bits-hyderabad.ac.in
Server:  UnKnown
Address:  172.16.0.30

Non-authoritative answer:
Name:    td.bits-hyderabad.ac.in
Address:  182.75.45.29


C:\Users\C R Hota>nslookup www.google.com
Server:  UnKnown
Address:  172.16.0.30

Non-authoritative answer:
Name:    www.google.com
Addresses:  2404:6800:4009:815::2004
          172.217.174.68


C:\Users\C R Hota>
```

Lab11 Next week

```
Adding bits.pilani.hpc ...

Looking for .bits child server
Not found, creating .bits child server
Forwarding to .bits child server

Looking for .pilani child server
Not found, creating .pilani child server
Forwarding to .pilani child server

Looking for .hpc child server
Not found, creating .hpc child server
Forwarding to .hpc child server

Authoratative server reached, IP 192.222.115.111

Adding in.ka.bangalore.gov ...

Looking for .in child server
Not found, creating .in child server
Forwarding to .in child server
```

IMPL. OF LINEAR PROBING & PERFORMANCE

```
void insert(const string &key, int value) {
        int index = hashFunction(key);
        int originalIndex = index;
        bool found = false;
        while (table[index].isOccupied && !table[index].isDeleted) {
            if (table[index].key == key){// Key exists, update value
                table[index].value = value;
                return;
            }
            index = (index + 1) % TABLE_SIZE;
            if (index == originalIndex) { // Full table check
                cout << "Hash table is full! Cannot insert.\n";
                return;
            }
        }
```

```
        // Insert new key-value pair
        table[index].key = key;
        table[index].value = value;
        table[index].isOccupied = true;
        table[index].isDeleted = false;
}
```

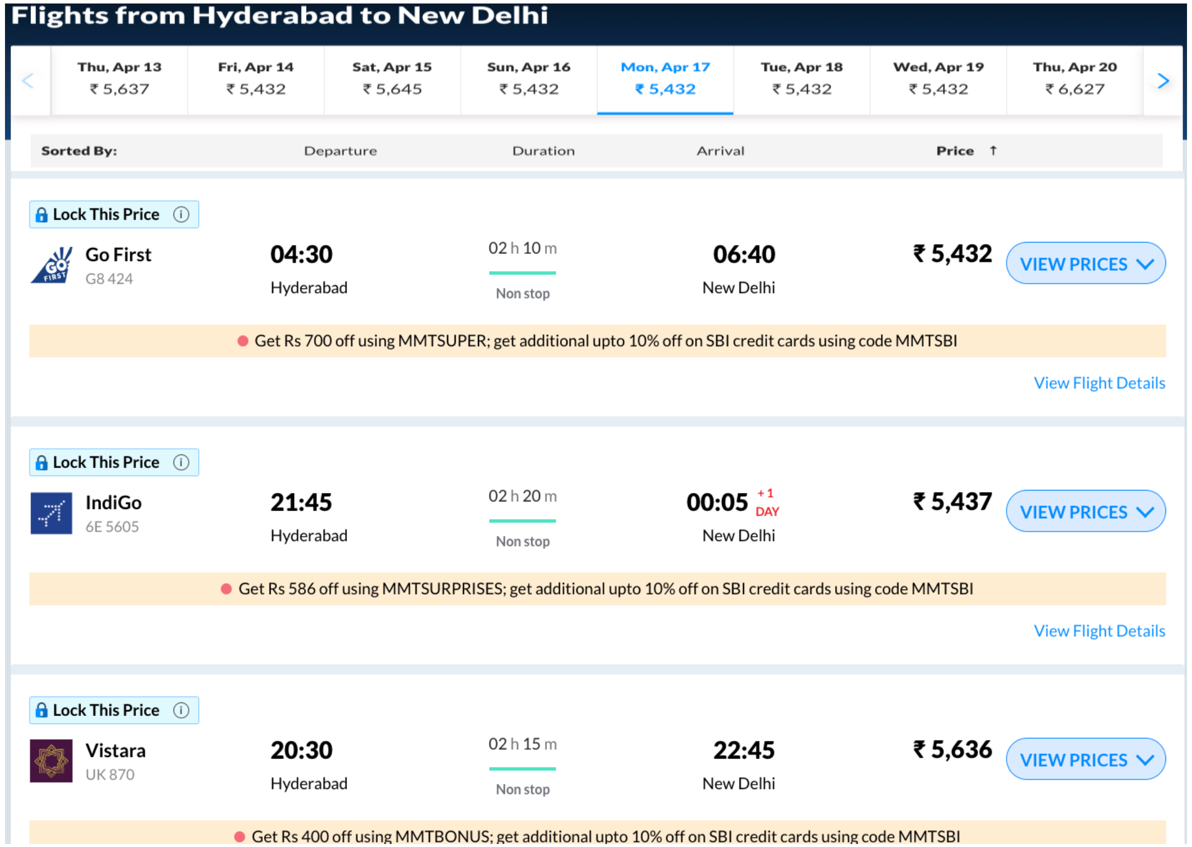
- In the worst case, searches, insertions and removals on a hash table take **?** time.
- The worst case occurs when all the keys inserted into the map collide.
- The load factor $\alpha = n/N$ affects the performance of a hash table. Default load factor is 0.75.

Rehashing

ORDERED MAPS: FLIGHT DATABASE

In some applications, looking up values based on associated keys is not enough.

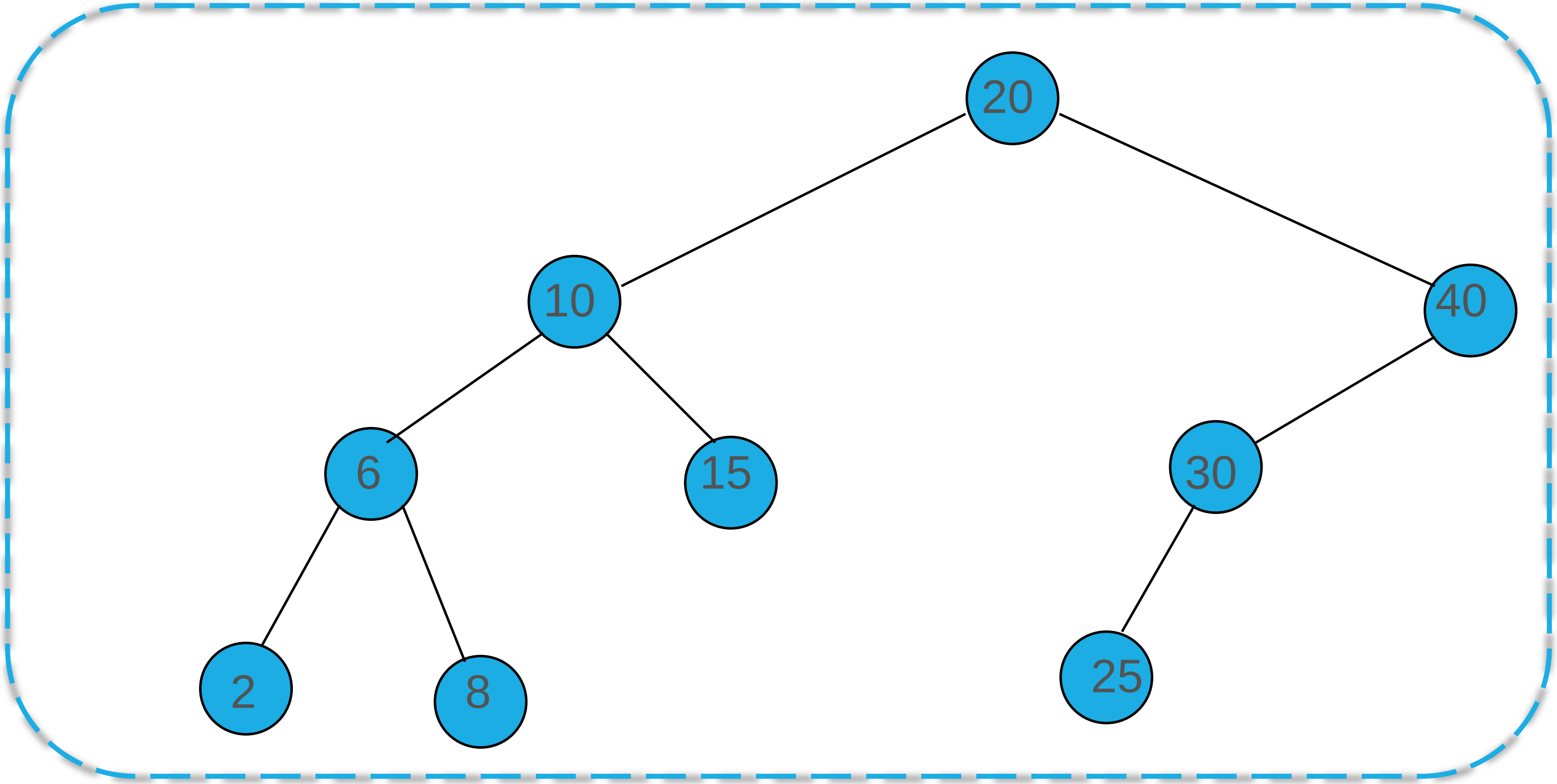
We often also want to keep the entries in a map sorted according to some total order and be able to look up keys and values based on this ordering. (ORDERED MAP).



Ex: Mid semester scores total display of CS F211.

Hashtables and lists are not the right ones, rather Skip lists and BSTs.

BINARY SEARCH TREE (BST) & ORDERED MAPS

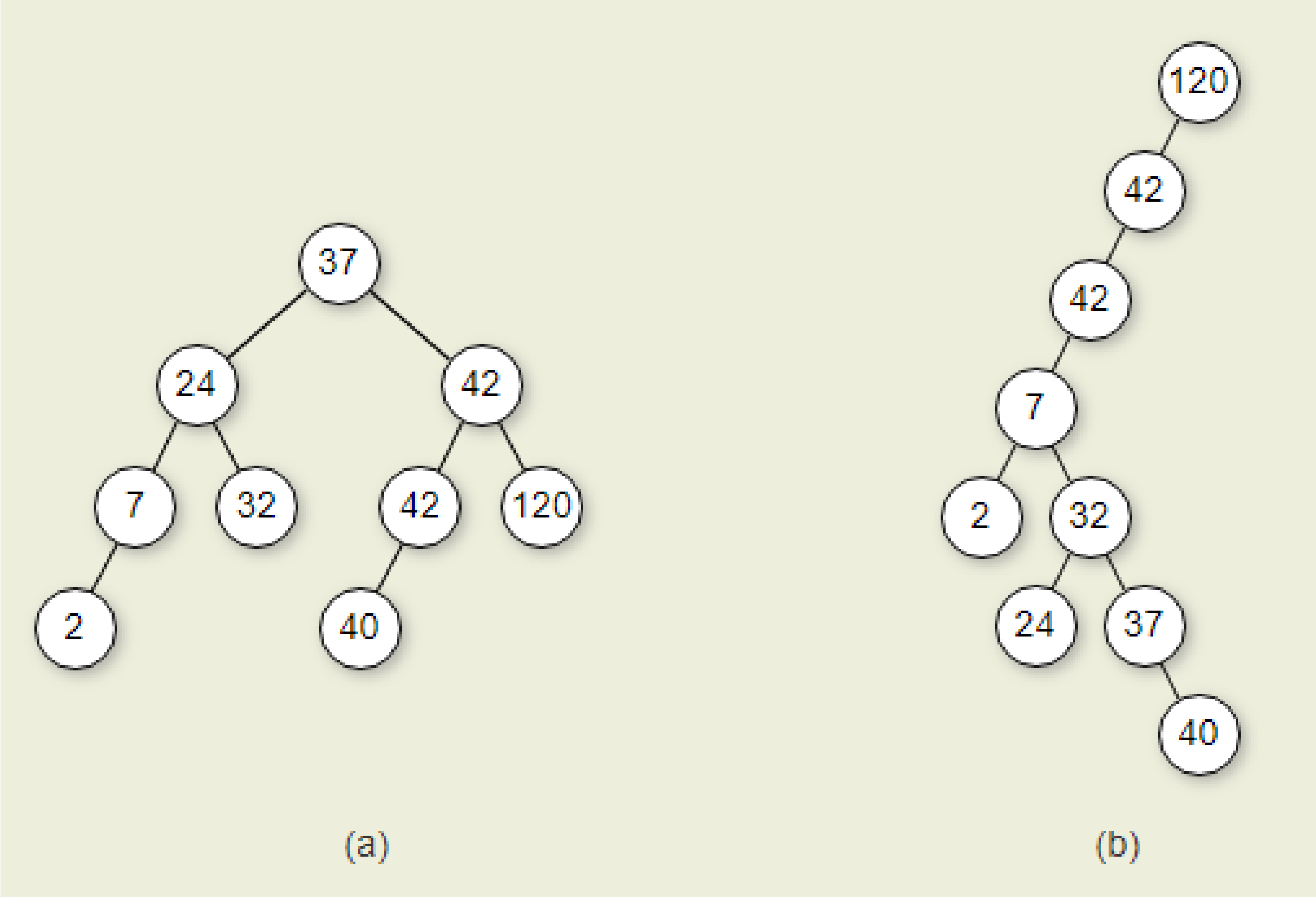


To get an ascending order, what is needed?

Keys are: 20, 10, 6, 2, 8, 15, 40, 30 and 25

If node with key 8 has a left child with key 3, what would be the type?

ORDER OF INSERTION IS IMPORTANT

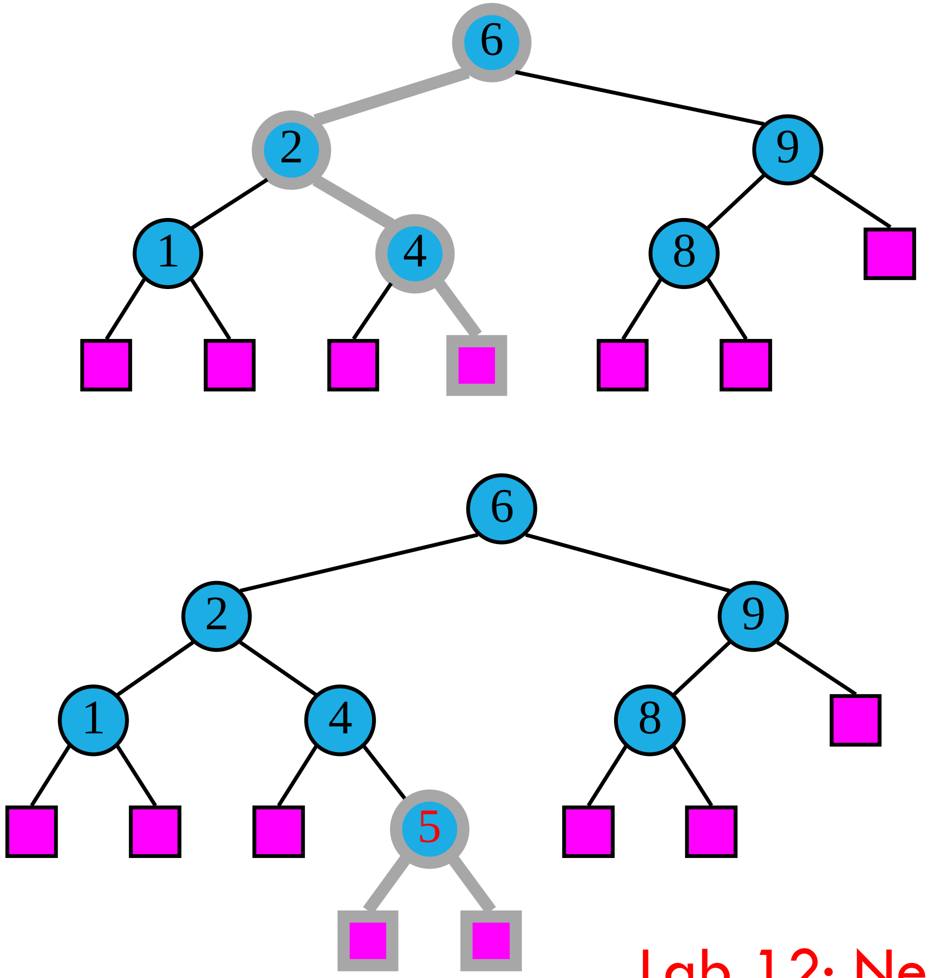


(a) if values are inserted in the order 37, 24, 42, 7, 2, 40, 42, 32, 120, (b) If the order is 120, 42, 42, 7, 2, 32, 37, 24, 40

```
template <typename E>
typename SearchTree<E>::TPos
SearchTree<E>::finder(const K& k, const TPos& v) {
    if (v.isExternal()) return v;  // Key not found
    if (k < (*v).key()) return finder(k, v.left());
    else if ((*v).key() < k) return finder(k, v.right());
    else return v;
}

template <typename E>
typename SearchTree<E>::Iterator
SearchTree<E>::find(const K& k) {
    TPos v = finder(k, root());
    if (v.isInternal()) {
        return Iterator(v);  // Found it
    }
    else return end();  // Didn't find it
}
```

Lab 12: Next to next week's lab…

INSERT INTO A BST




```
BST-Insert(T, z)
  y := NIL
  x := T.root
  while x ≠ NIL do
    y := x
    if z.key < x.key then
      x := x.left
    else
      x := x.right
    end if
  repeat
  z.parent := y
  if y = NIL then
    T.root := z
  else if z.key < y.key then
    y.left := z
  else
    y.right := z
  end if
```

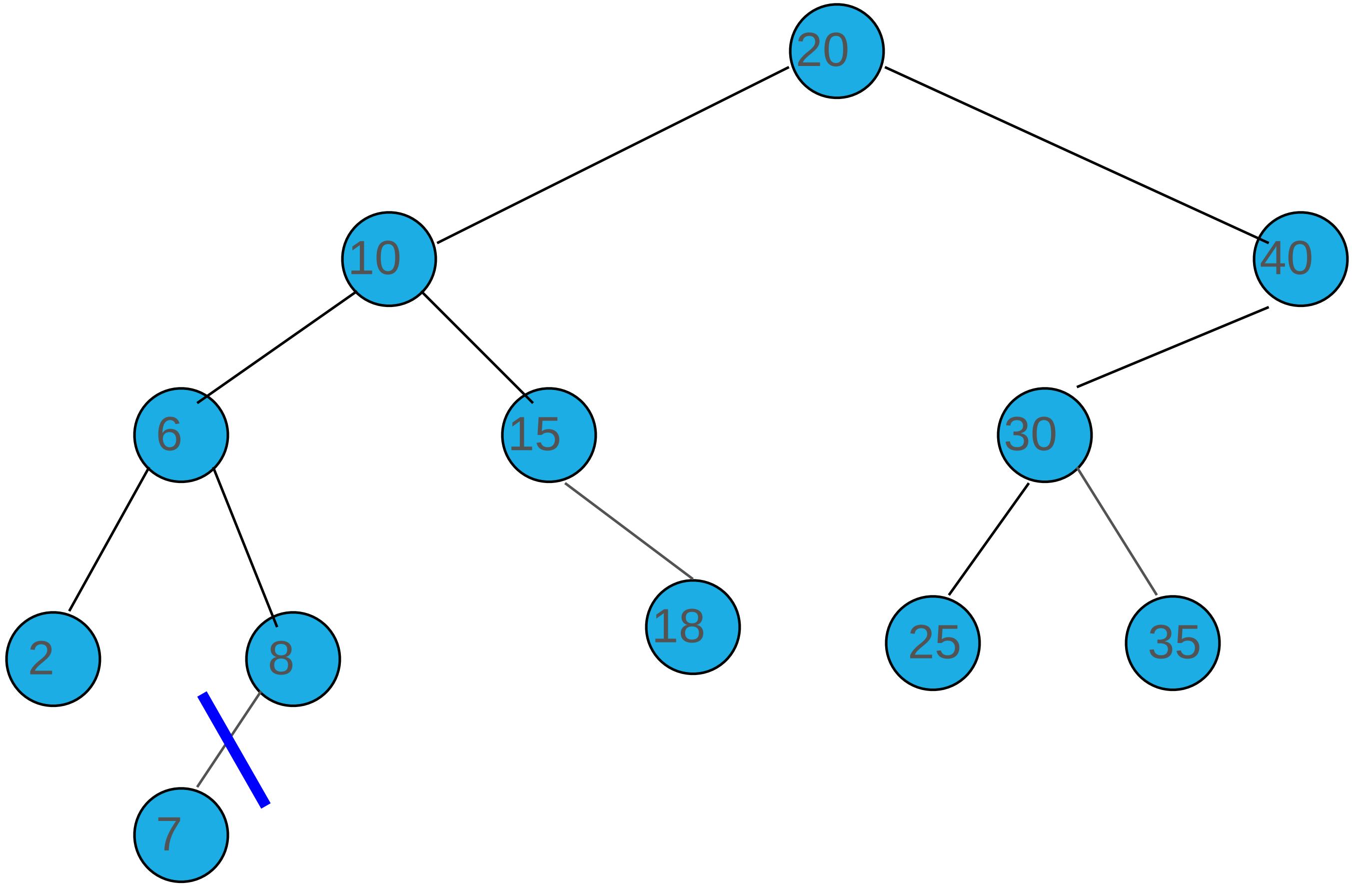
Lab 12: Next to next week's lab

Complexity?

DELETE AN ITEM FROM BST

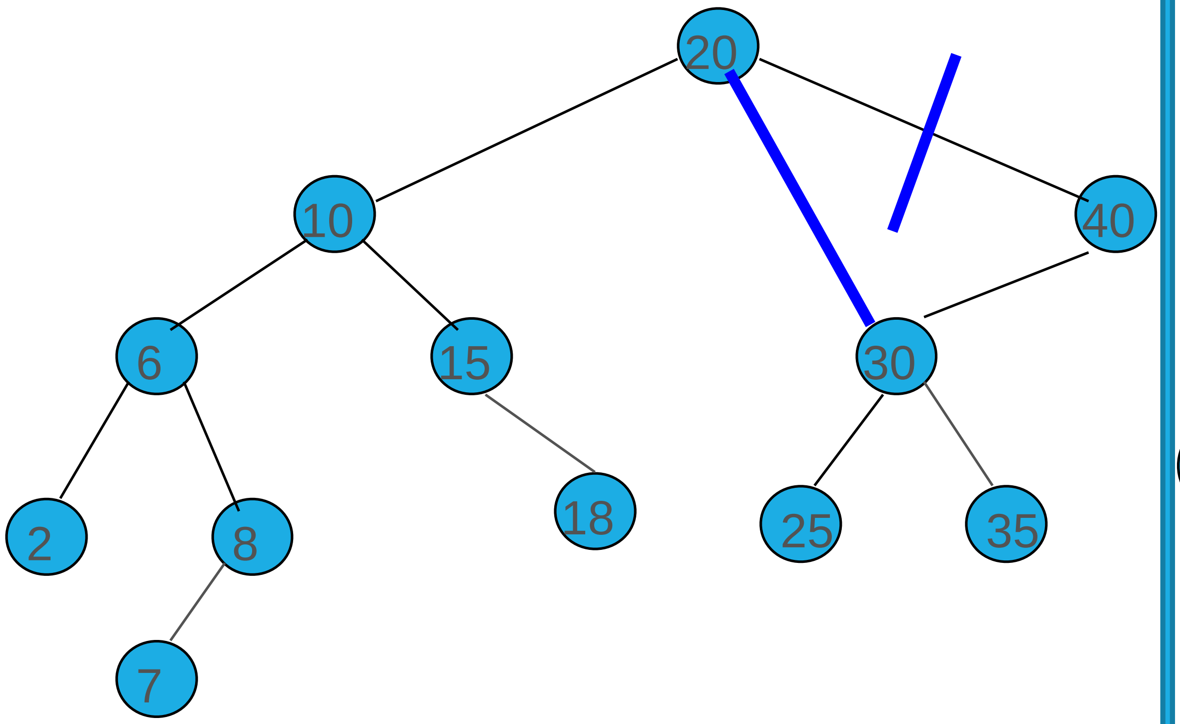
Three cases:

- Element is in a leaf (Case I)
- Element is in a degree 1 node (Case II)
- Element is in a degree 2 node (Case III)

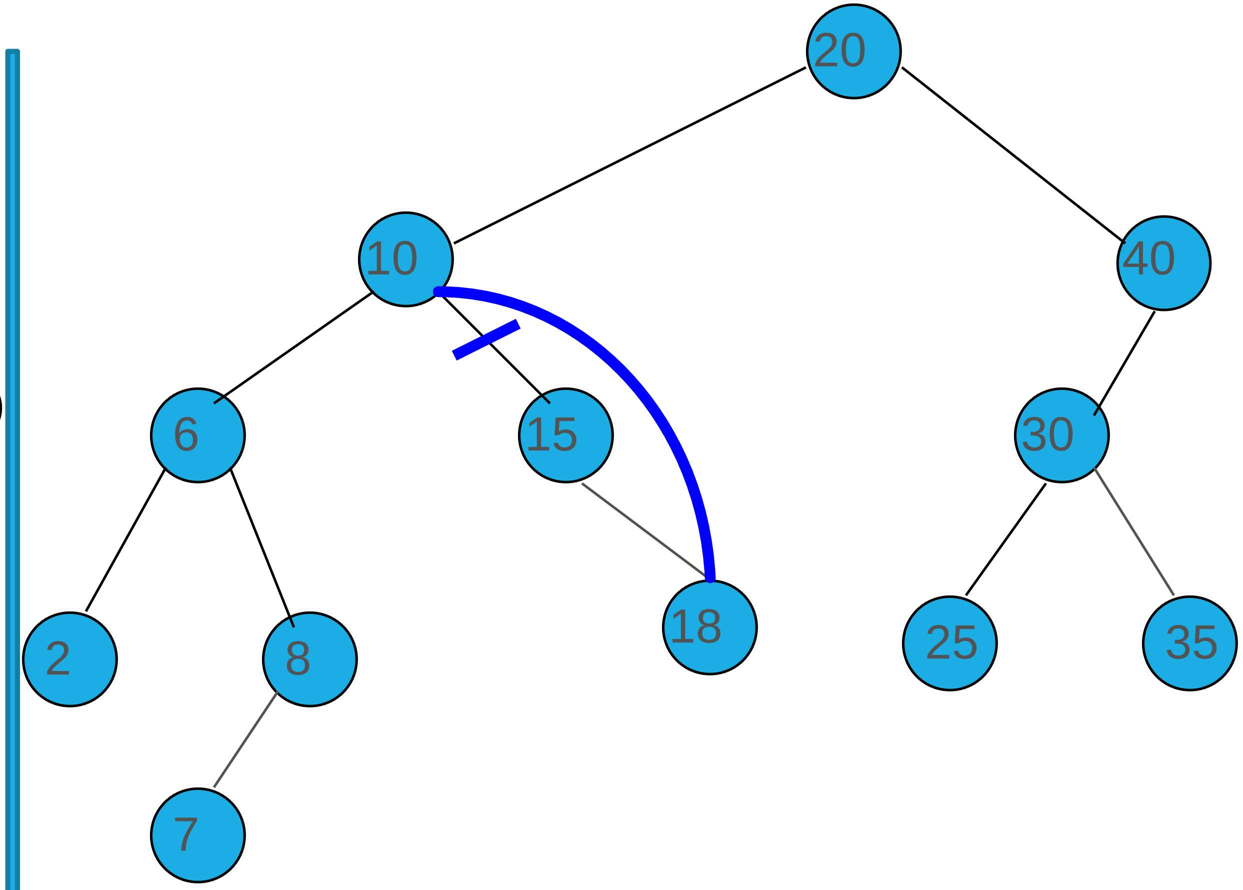


Case I: Erase key = 7

CASE II

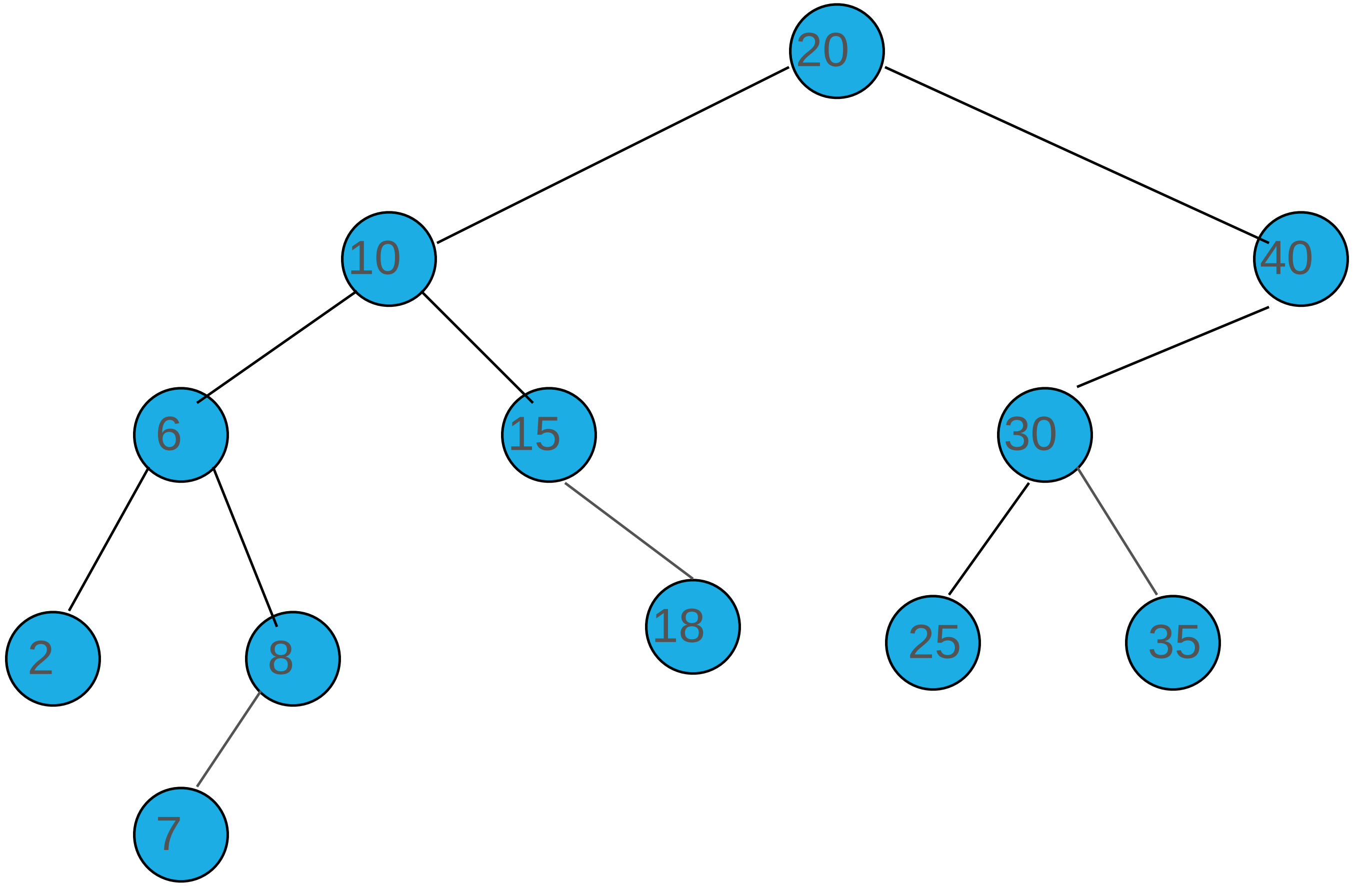


Erase key = 40



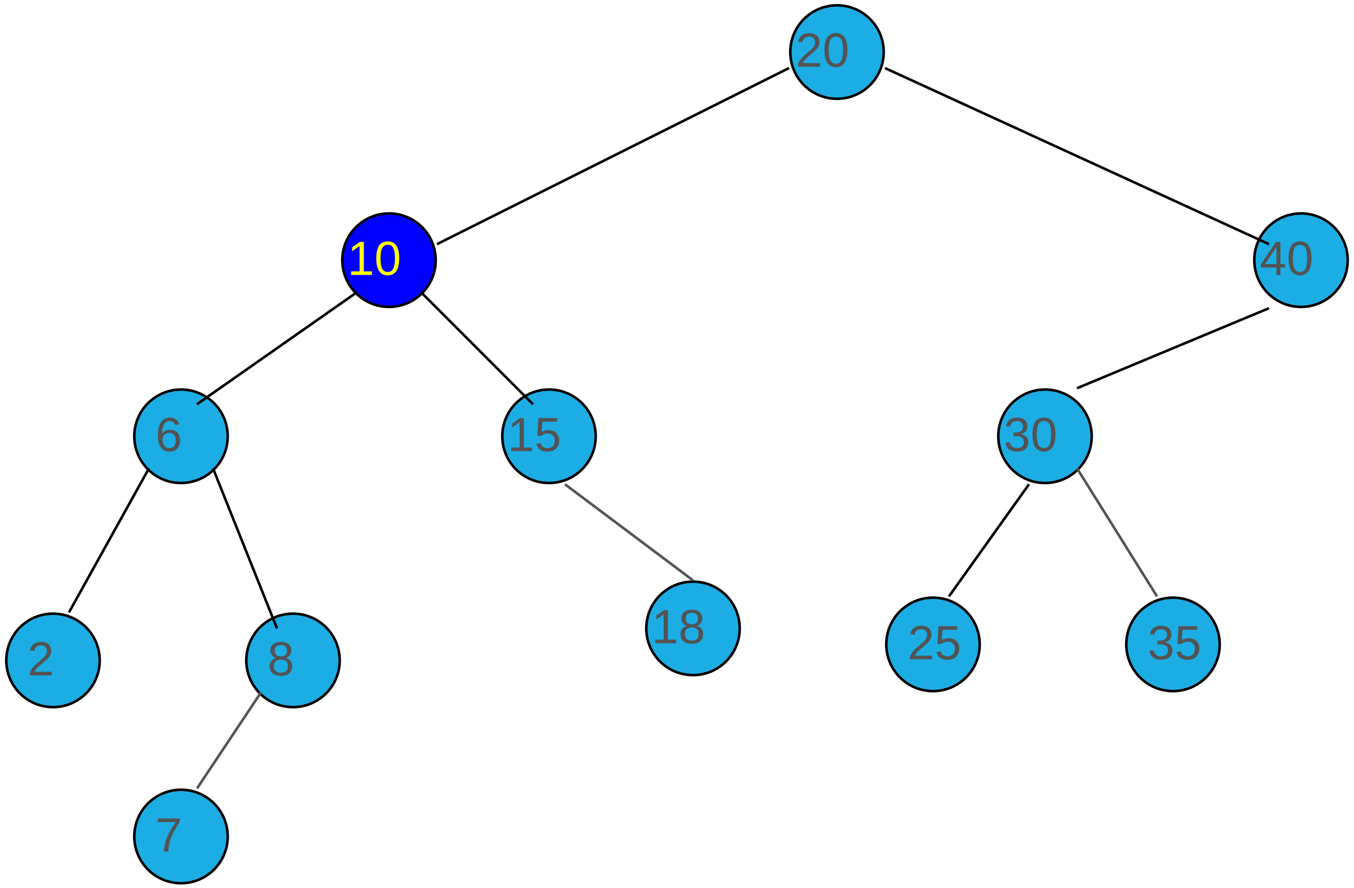
Erase key = 15

CASE III



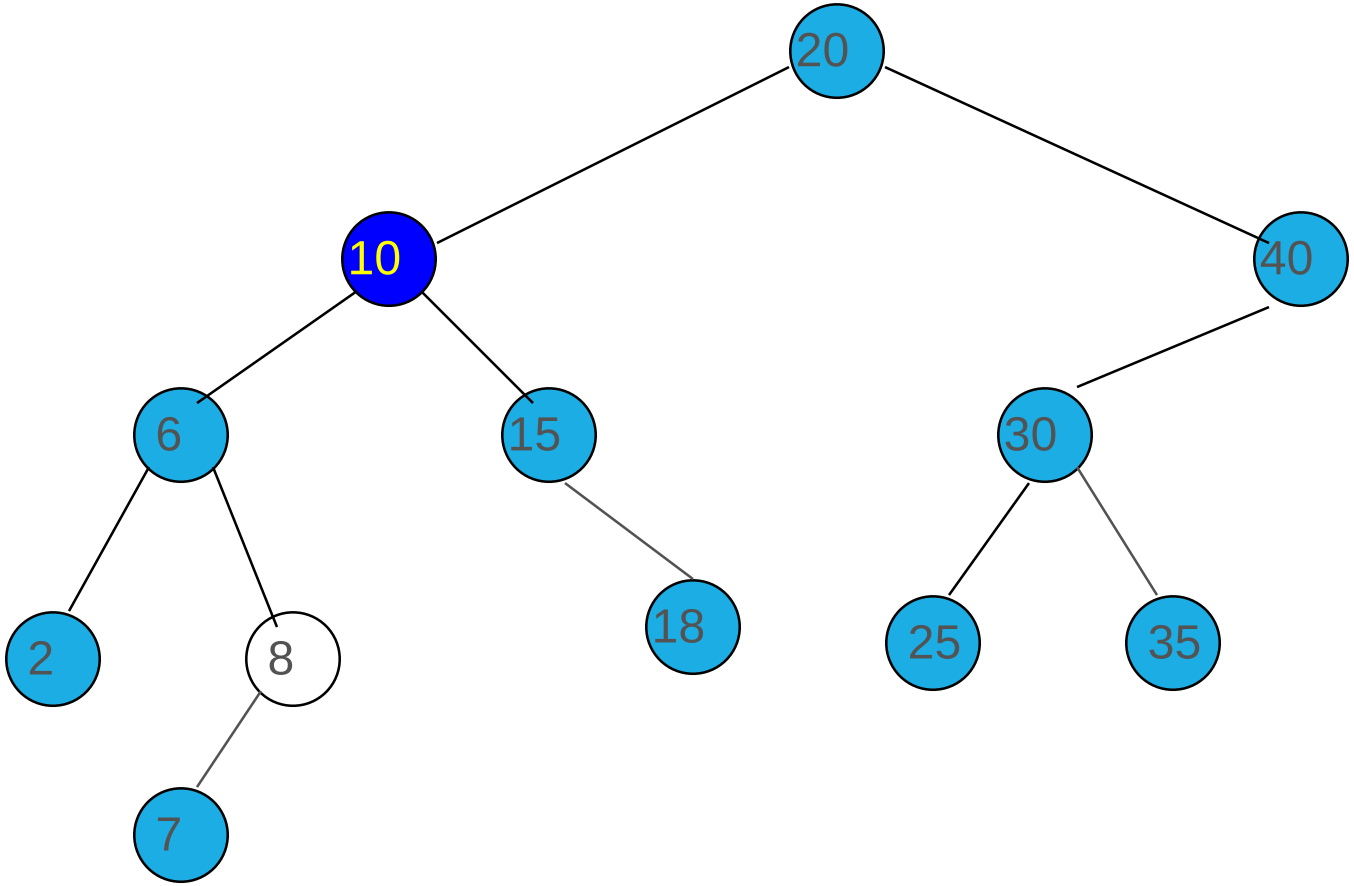
Erase from a degree 2 node. Erase key = 10

CASE III CONTINUED...



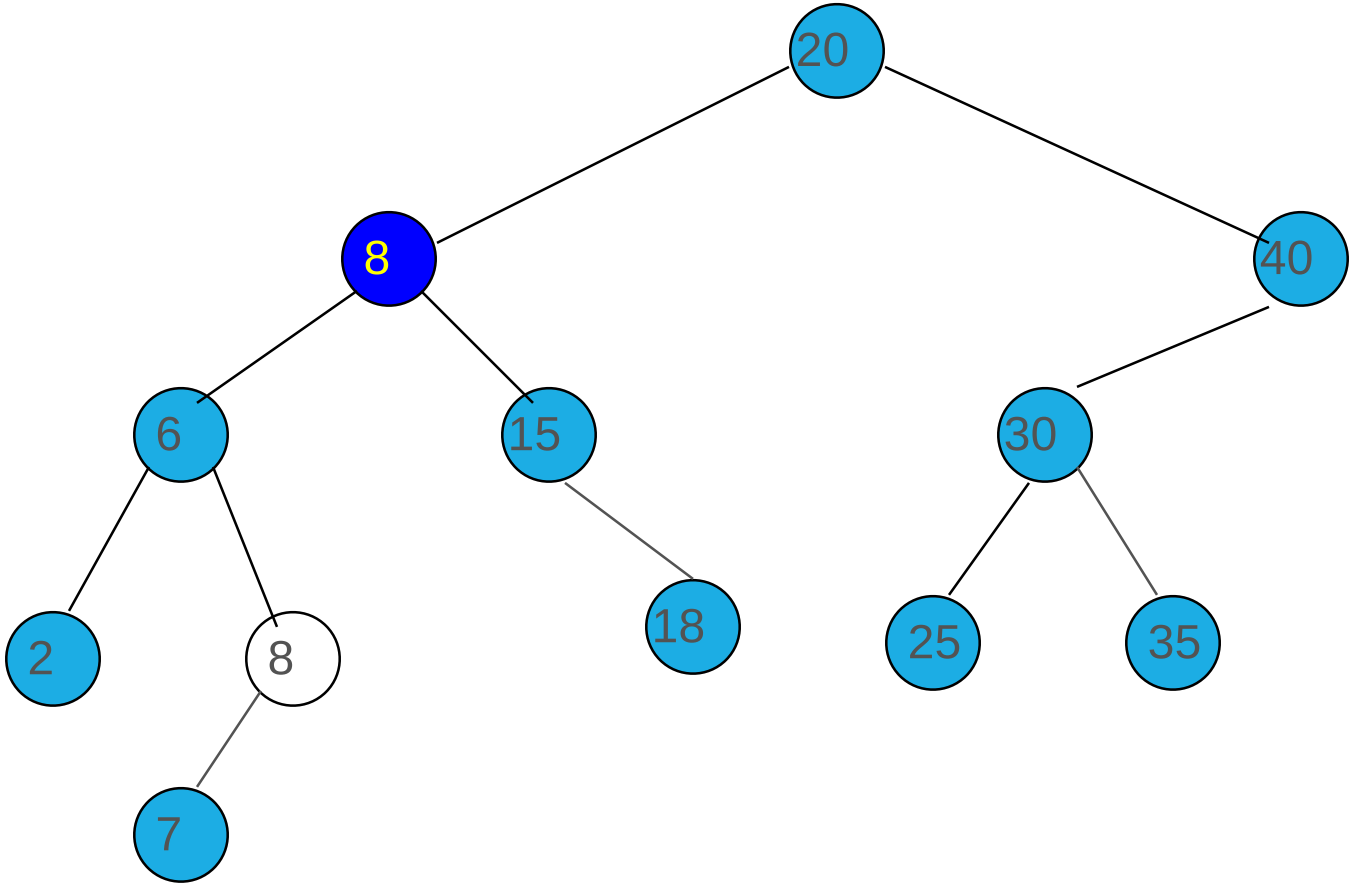
Replace with largest key in left subtree (or smallest in right subtree).

CASE III CONTINUED...



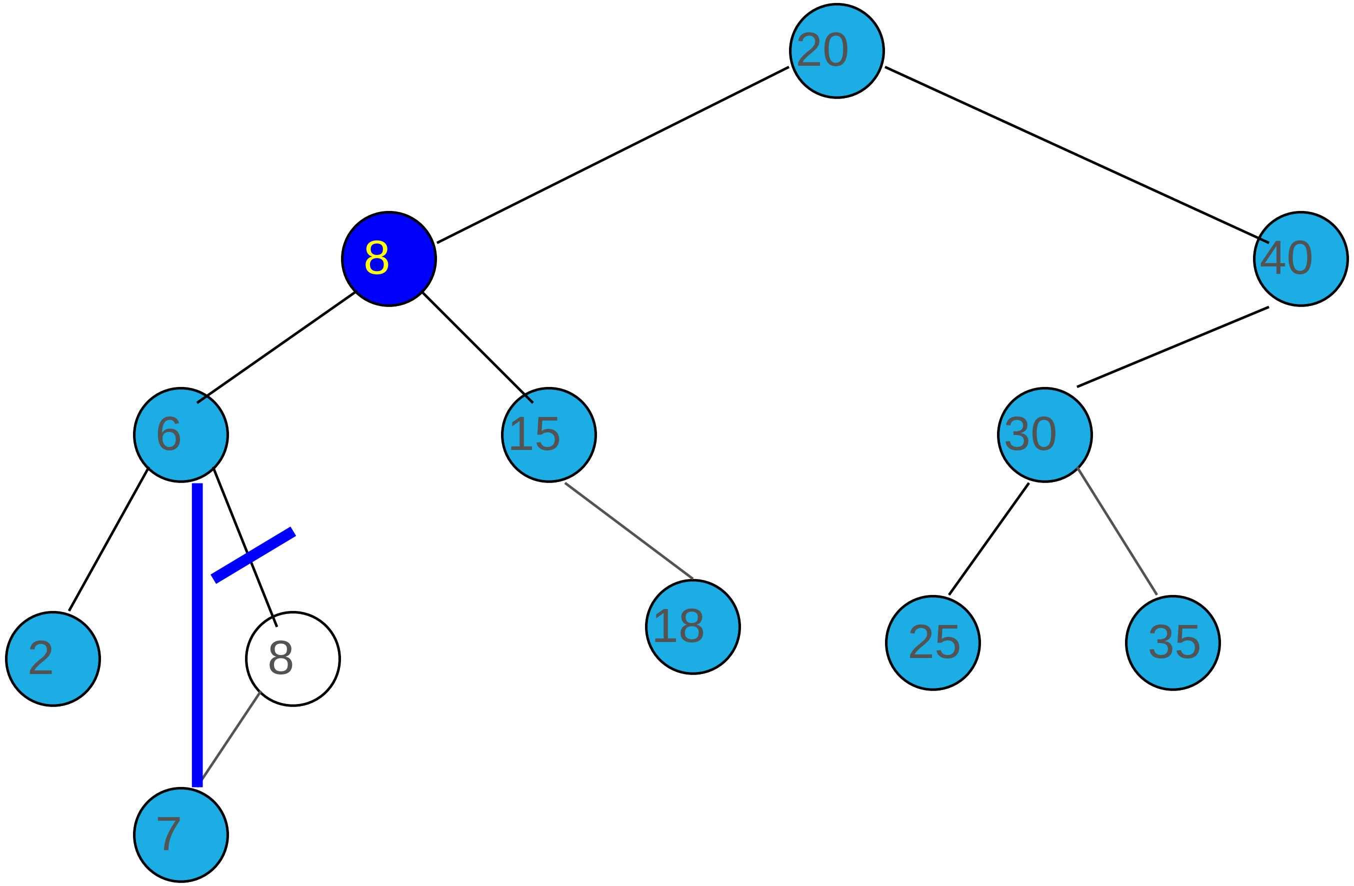
Replace with largest key in left subtree (or smallest in right subtree).

CASE III CONTINUED…



Replace with largest key in left subtree (or smallest in right subtree).

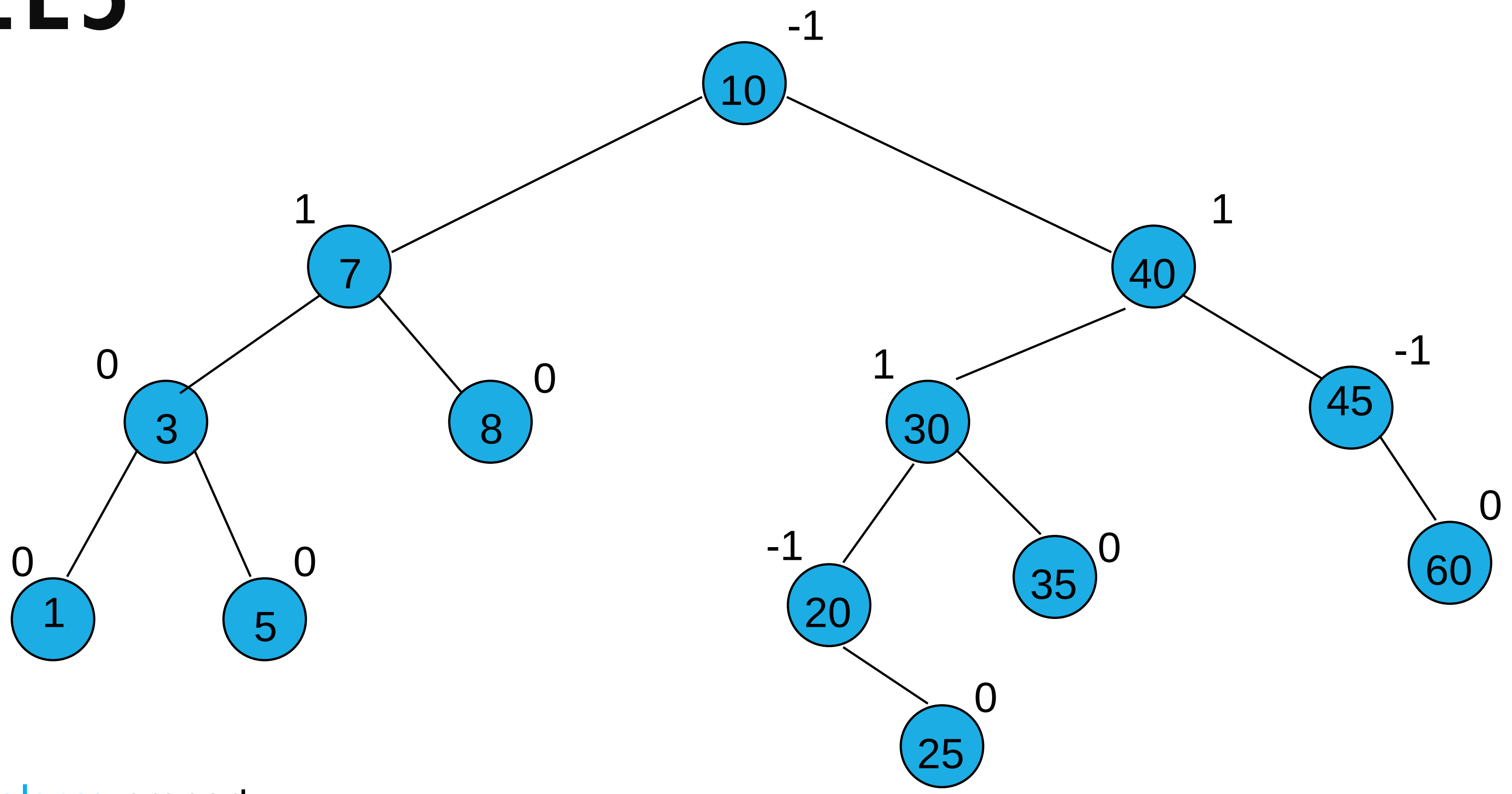
CASE III CONTINUED…



Largest key must be in a leaf or degree 1 node.

AVL TREES

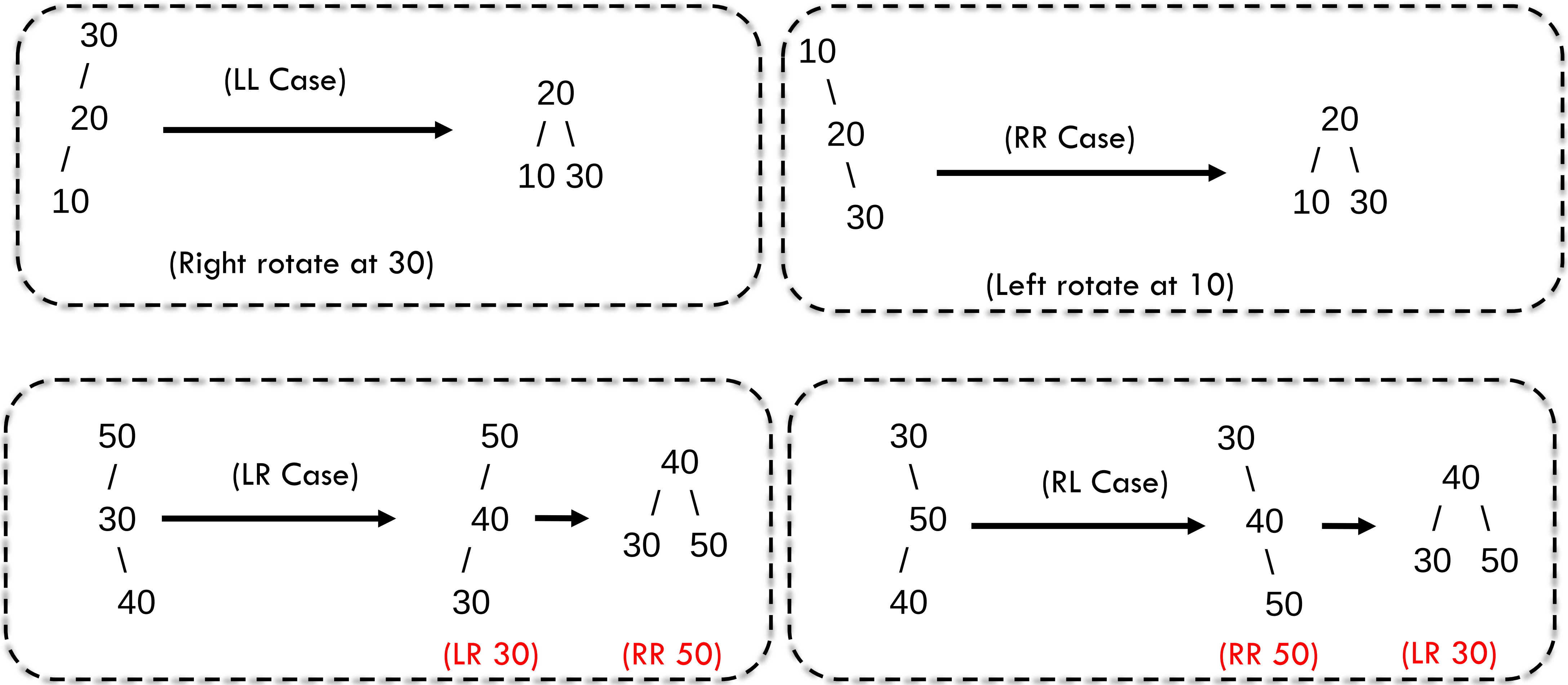
- Adelson-Velsky and Landis (Soviet mathematicians and computer scientists)



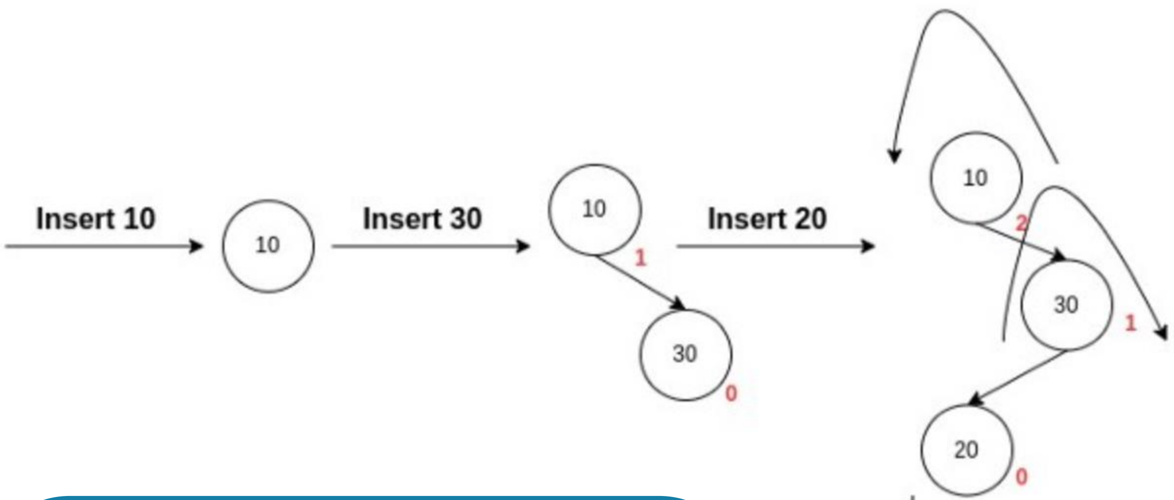
- BST with height-balance property.

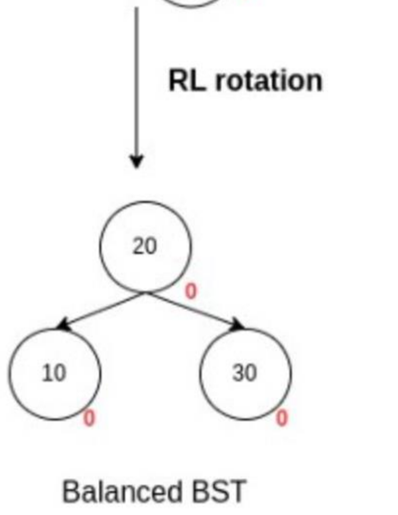
- What happens to the Balance Factors and how will you handle the imbalance?

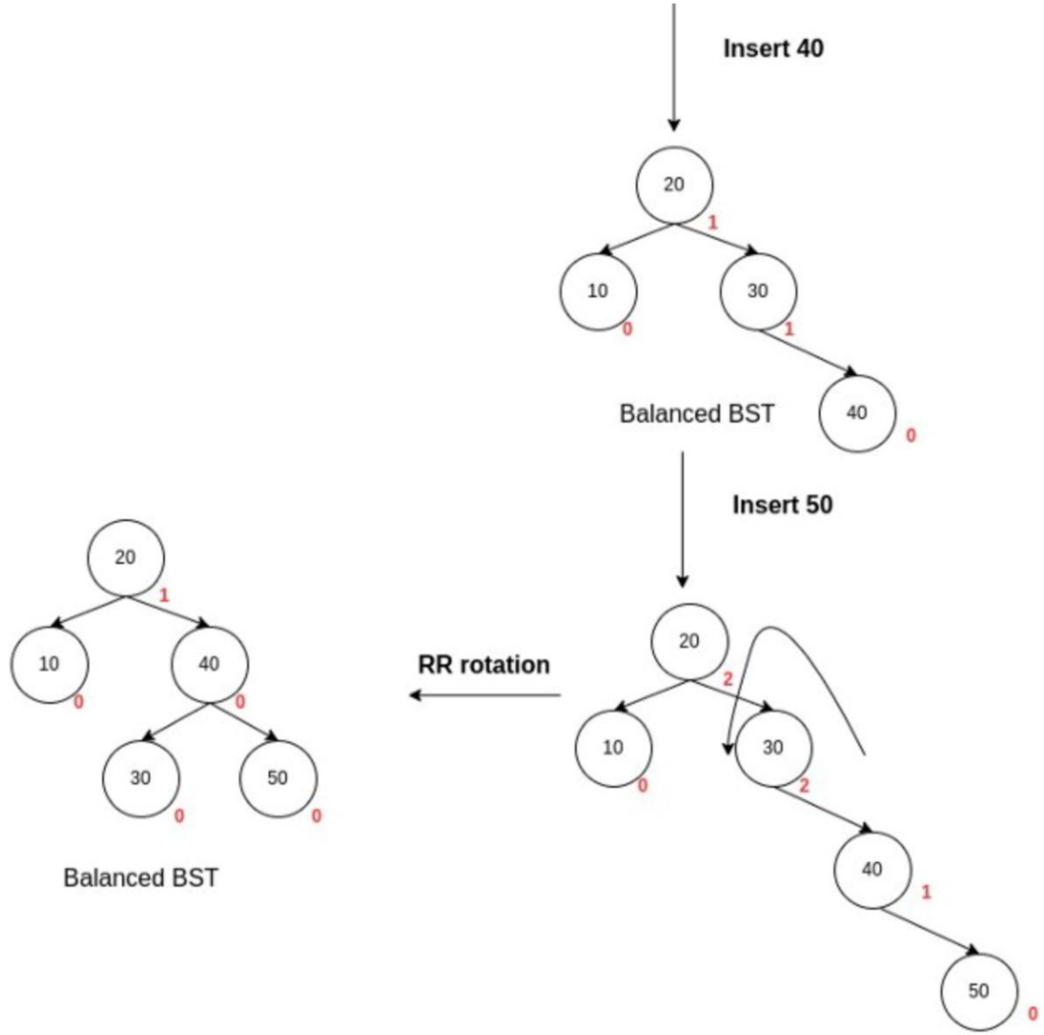
BALANCING THE TREE THROUGH ROTATIONS



ANOTHER EXAMPLE: ALL THE CASES







Algorithm:

1. Insert using BST insertion logic.
2. Check the B.F of each node.
3. Perform RR, RL, LL, or LR if needed.
4. Retrace till the root.

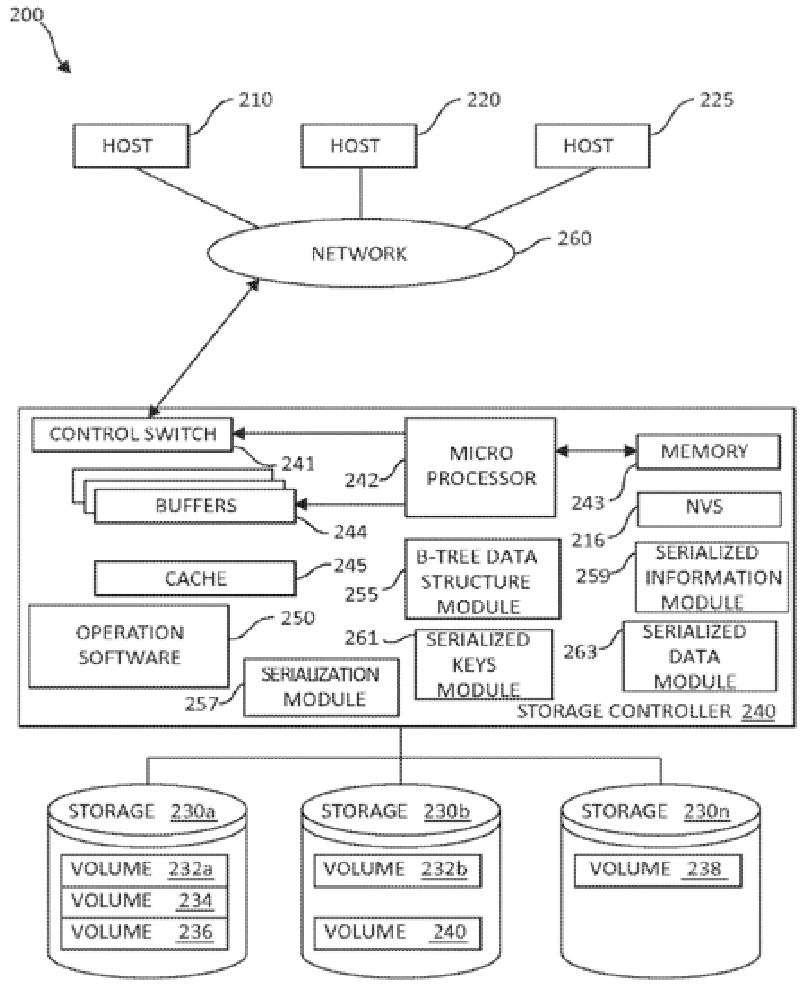
C++ IMPLEMENTATION OF AVL TREE

```
template <typename K, typename V>
void Tree<K, V>::remove(const K& key, Node*& spot) {
    if (spot == nullptr)
        return;

    if (spot->key < key)
        remove(key, spot->right);

    else if (spot->key > key)
        remove(key, spot->left);

    else if (spot->left != nullptr && spot->right != nullptr) {
        spot->value = findMin(spot->right)->value;
        remove(spot->value, spot->right);
    }

    else {
        Node* oldNode = spot;
        spot = (spot->left != nullptr) ? spot->right : spot->left;
        delete oldNode;
    }
    balance(spot);
}
```

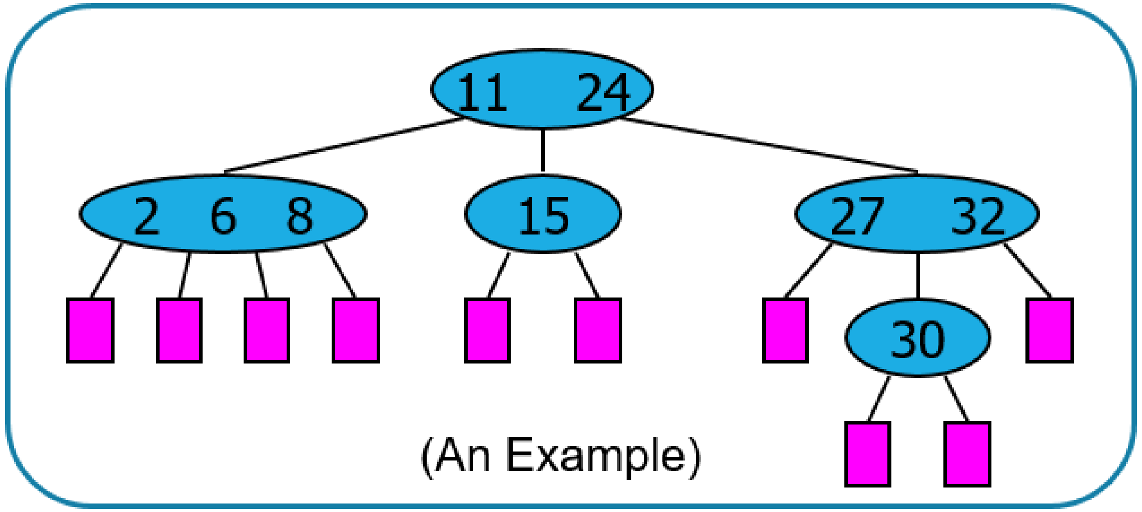
```
template <typename K, typename V>
void Tree<K, V>::balance(Node*& spot) {
    if (spot != nullptr) {
        //if violation is on left child
        if (height(spot->left) - height(spot->right) > ALOWED_IMBALANCE) {
            //LL violation
            if (height(spot->left->left) >= height(spot->left->right))
                rotateWithLeftChild(spot);
            else  //left-right violation
                doubleWithLeftChild(spot);
        }
        else {
            //if violation is on right child
            if (height(spot->right) - height(spot->left) > ALOWED_IMBALANCE) {
                //RR violation
                if (height(spot->right->right) >= height(spot->right->left))
                    rotateWithRightChild(spot);
                else  //right-left violation
                    doubleWithRightChild(spot);
            }
        }
        spot->height = max(height(spot->left), height(spot->right)) + 1;
    }
}
```

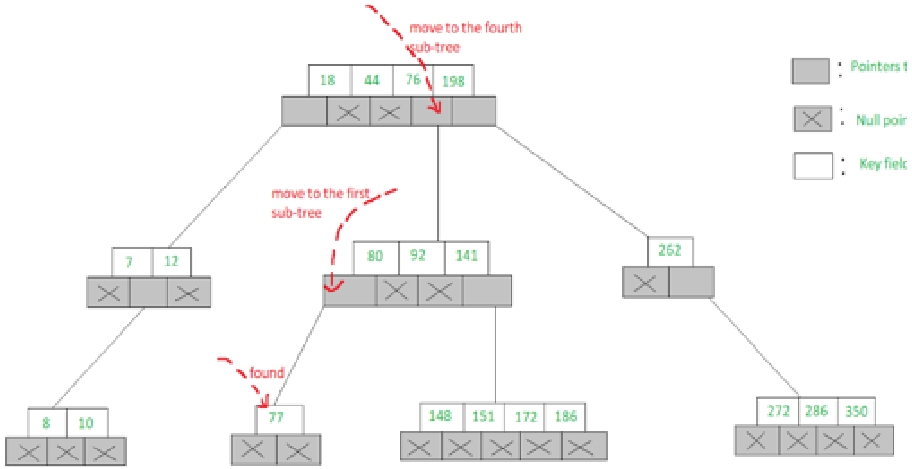
Lab 12: Next to next week…

MULTI-WAY SEARCH TREE: ORDERED MAPS



https://patents.google.com/patent/US9305040B2/en



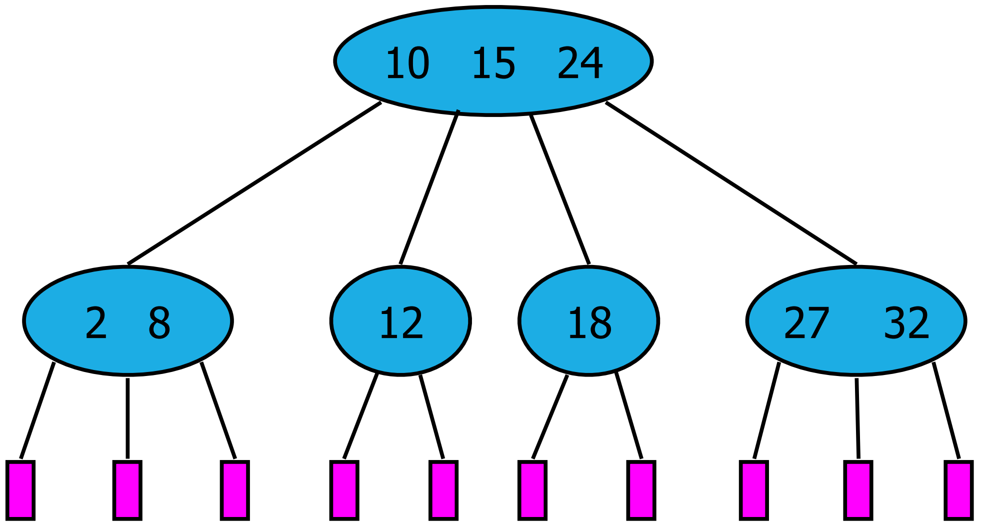


(2,4) TREES

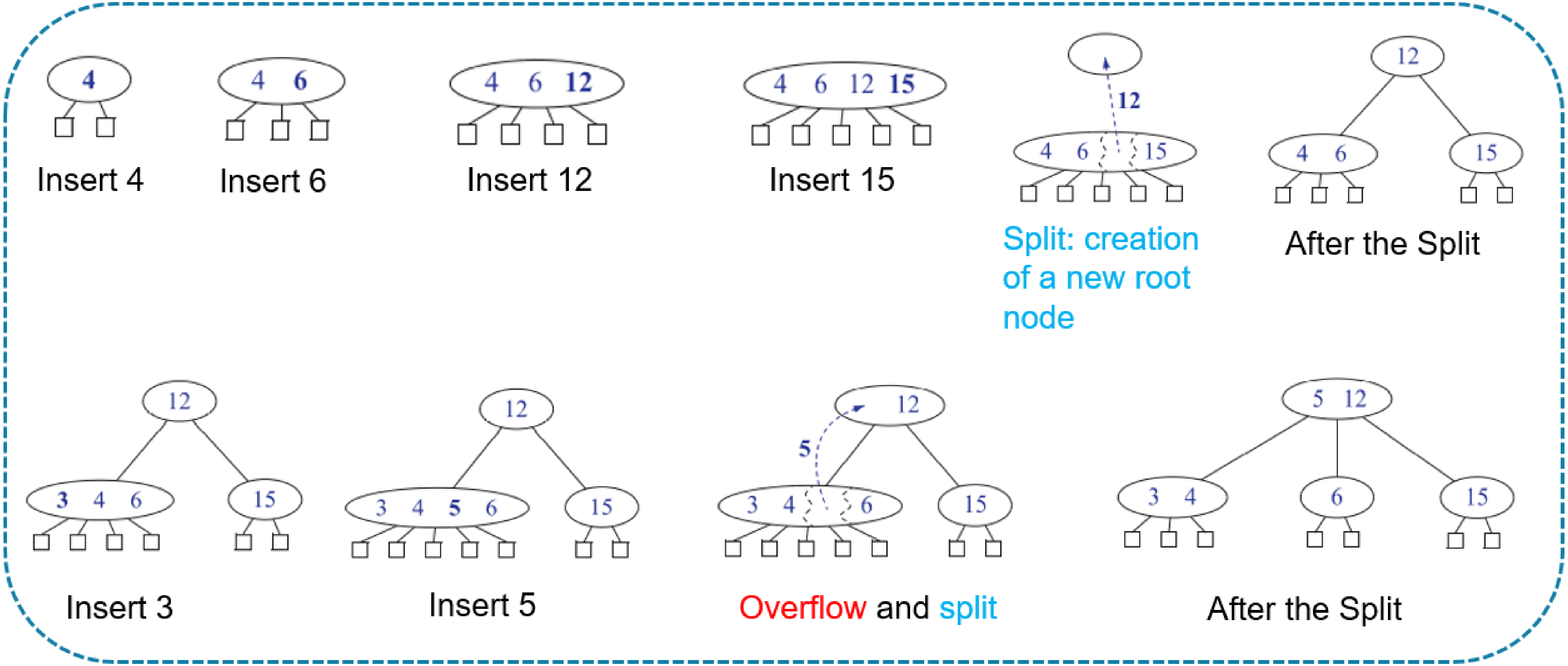
A (2,4) tree is a multi-way search tree with the following properties:

- Node-Size Property
- Depth Property

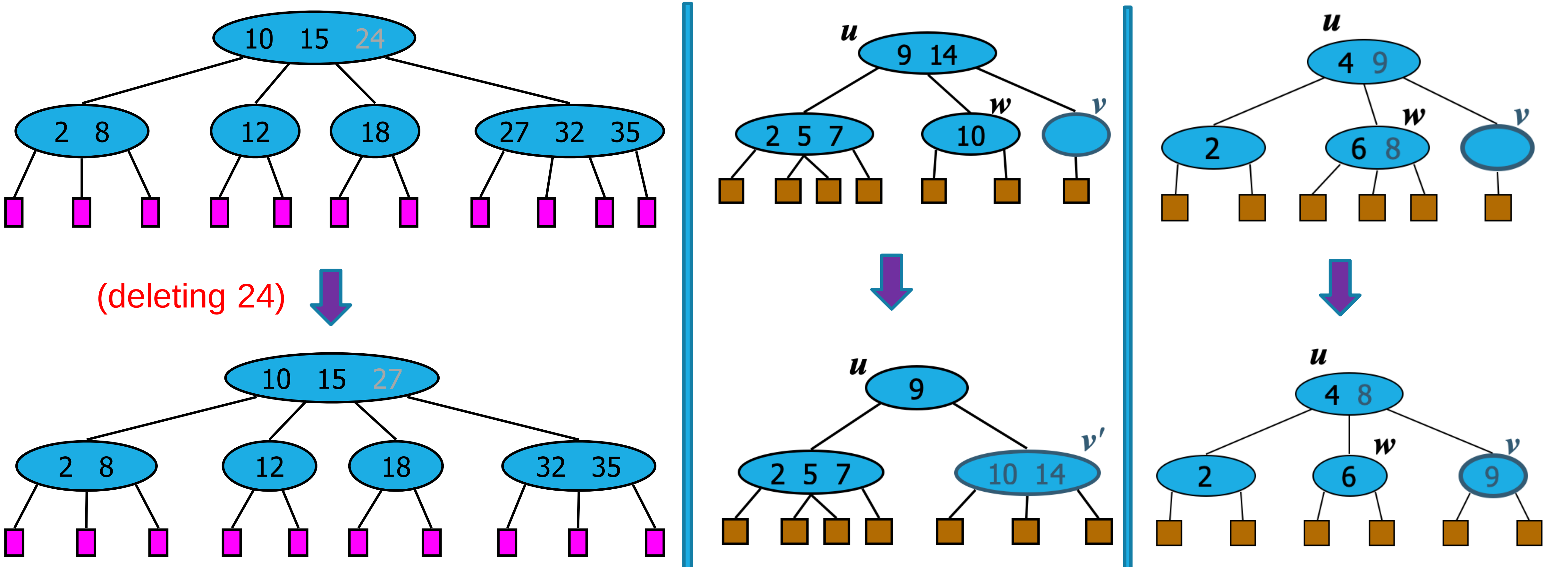
Theorem: A (2,4) tree storing ***n*** items has height ***O***(log ***n***)



INSERTION IN (2,4): OVERFLOW AND SPLIT



DELETION IN (2,4): UNDERFLOW AND FUSION/TRANSFER



(replace with what?)

(Case-I: underflow and fusion with 2-node sibling)

(Case-II: underflow and transfer with 3/4-node sibling)

IMPL. OF 2-4 TREES

```cpp
    // Helper function to split a node into two nodes
    void splitNode(TreeNode* node) {
        int median = node->keys[1];

        TreeNode* leftNode = new TreeNode(node->keys[0]);
        TreeNode* rightNode = new TreeNode(node->keys[2]);

        if (!node->children.empty()) {
            leftNode->children.push_back(node->children[0]);
            leftNode->children.push_back(node->children[1]);
            rightNode->children.push_back(node->children[2]);
            rightNode->children.push_back(node->children[3]);
        }

        node->keys.clear();
        node->keys.push_back(median);
        node->children.clear();
```

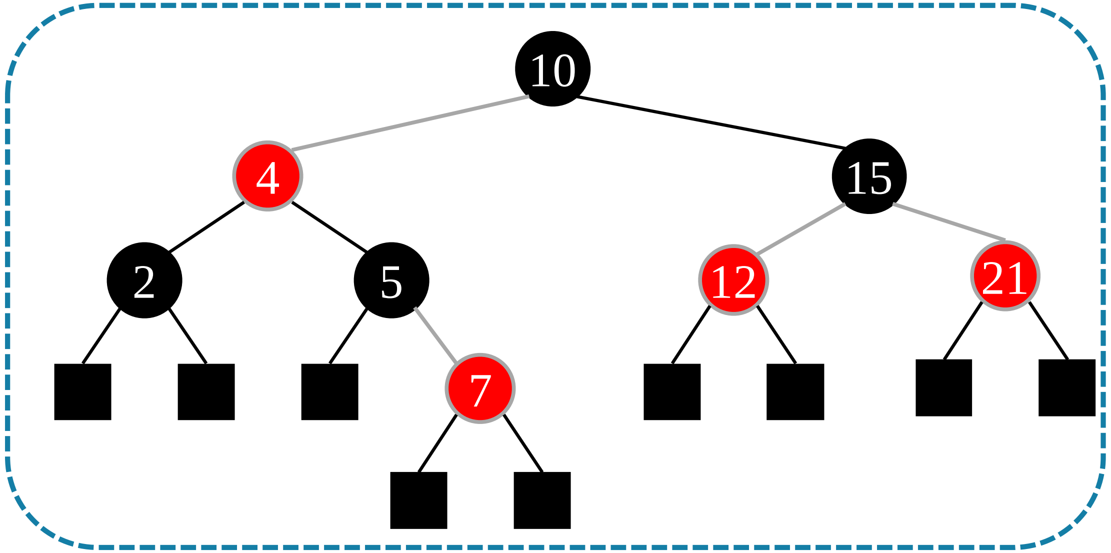
```cpp
int main() {
    Tree24 tree;
    vector<int> keys={5, 9, 1, 3, 7, 6, 10, 8, 2, 4};

    for (int key : keys) {
        tree.insert(key);
    }
```

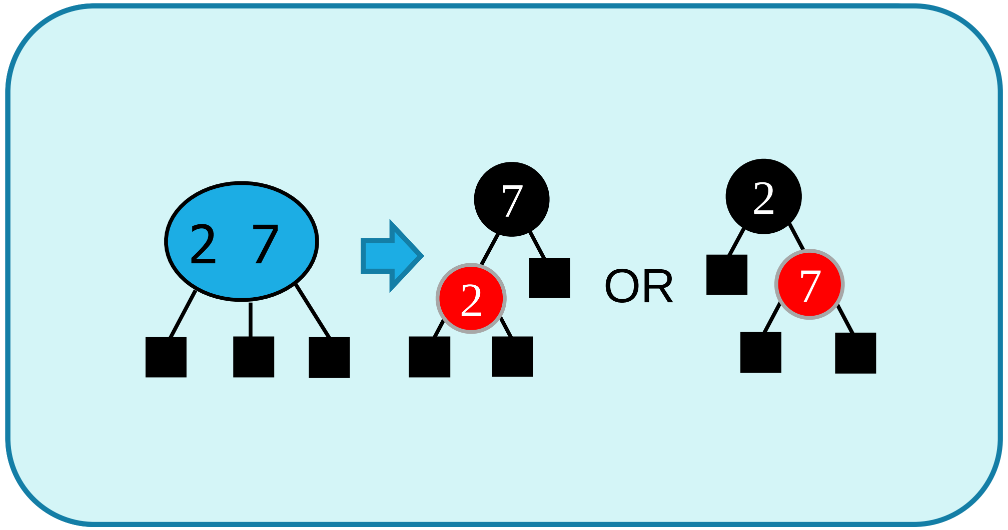
```
In-order traversal: 1 2 3 4 5 6 7 8
Pre-order traversal: 3 1 2 6 4 5 7 8
Post-order traversal: 1 2 4 5 7 8 6 3
```

```cpp
class Tree24 {
public:
    TreeNode* root;

    Tree24() {
        root = nullptr;
    }

    // Insert a key into the 2-4 tree
    void insert(int key) {
        if (root == nullptr) {
            root = new TreeNode(key);
        } else {
            insertKey(root, key);
        }
    }

    // Helper function to insert a key into a node
    void insertKey(TreeNode* node, int key) {
        int i = node->keys.size() - 1;
        if (node->children.empty()) {
            node->keys.push_back(key);
            sort(node->keys.begin(), node->keys.end());
        } else {
            while (i >= 0 && key < node->keys[i]) {
                i--;
            }
            insertKey(node->children[i + 1], key);
        }

        if (node->keys.size() == 4) {
            splitNode(node);
        }
    }
```

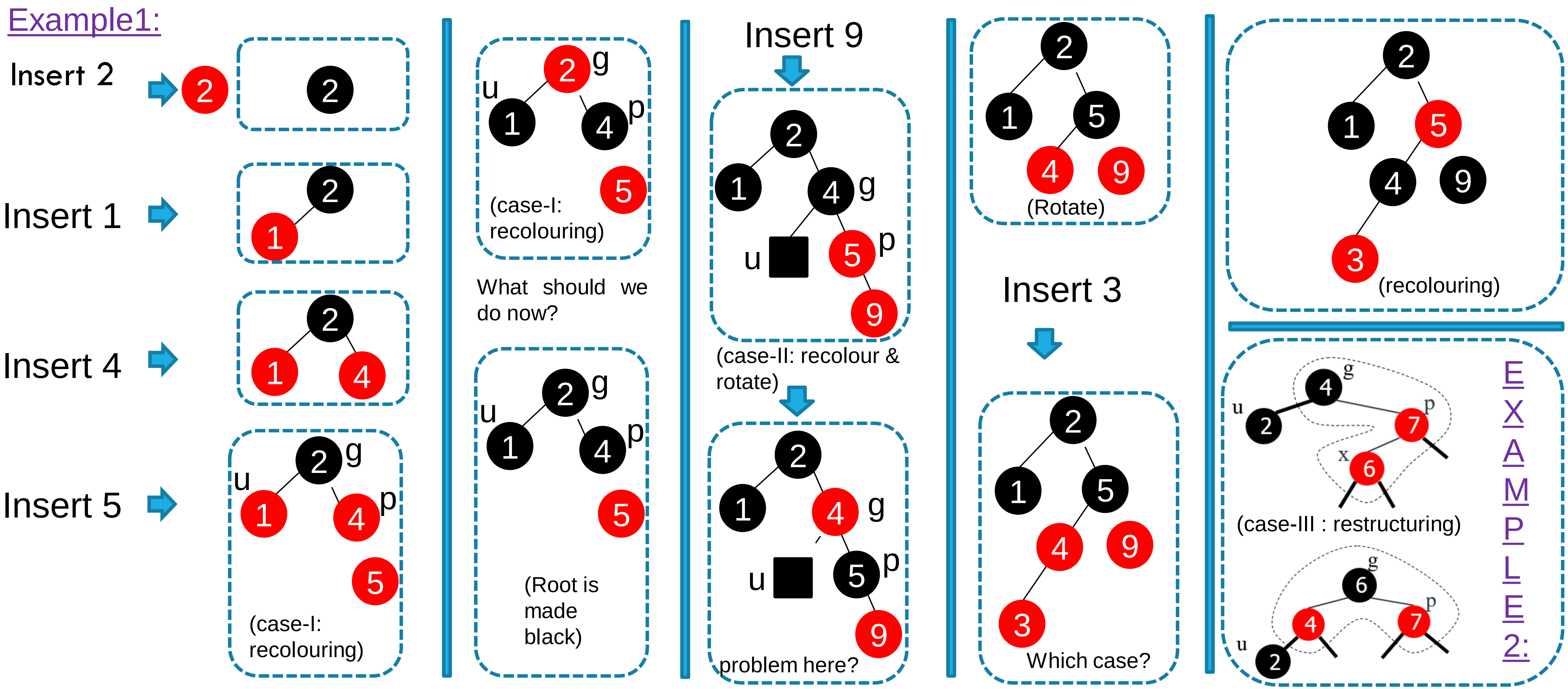
RED-BLACK TREE (RBT): AN ALTERNATIVE TO (2,4)

- Can we do away with complex Rotations (AVL tree) or Split/Fusion operations (2-4 trees)?
- Red-Black trees are self balancing Binary Search Trees (BSTs)
- Same logarithmic time performance O(logn) like AVL and (2,4)
- But, simpler implementation with a single node type that can be colored red or black
- Simulate (2-4) trees in a Binary tree (BST)





INSERTING INTO RBT: HANDLING DOUBLE RED

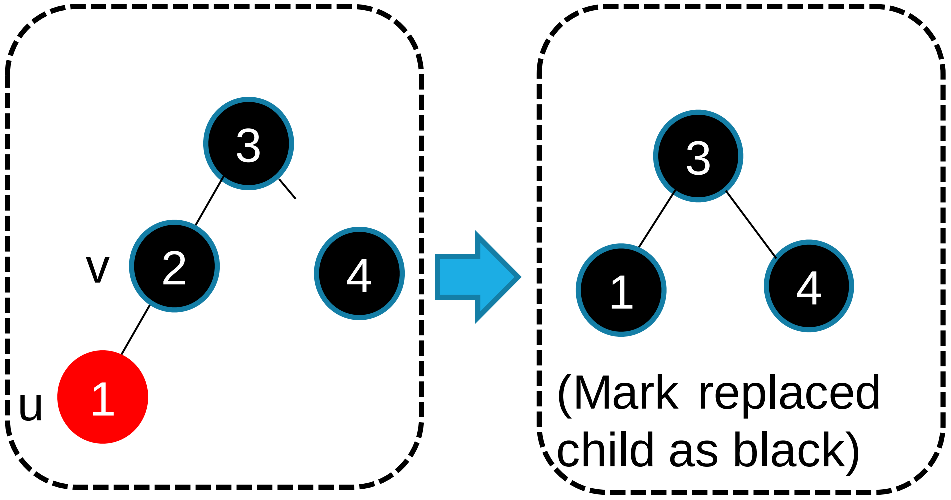


DELETION IN A RED BLACK TREE

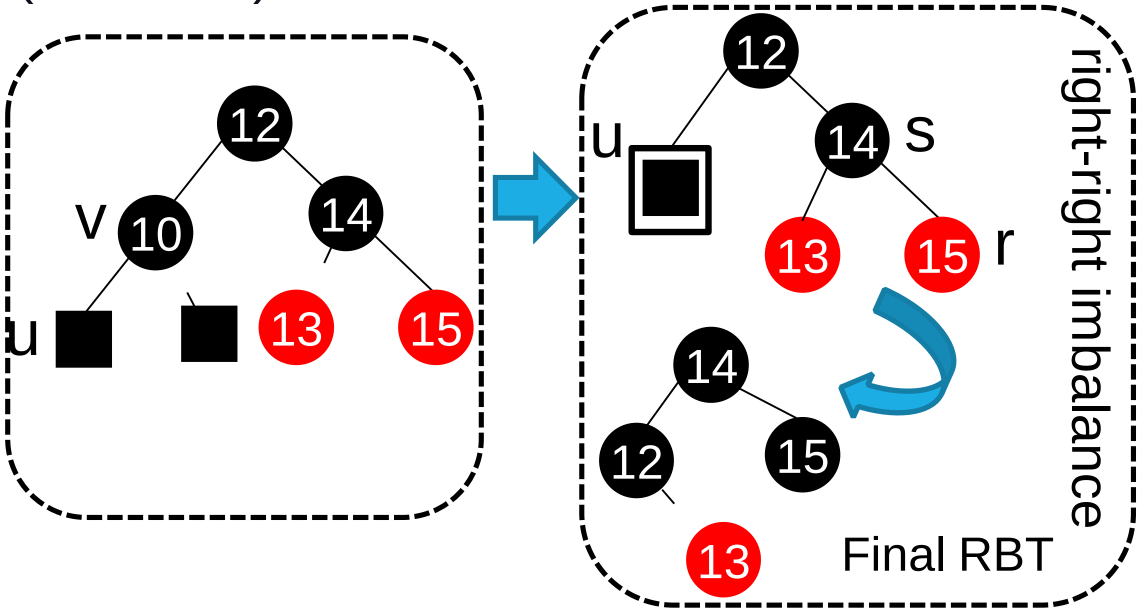
- Similar to BST, except that after deletion check if all the properties of RBT are satisfied.

- If not, perform recolor or rotation followed by recoloring to make it balanced.

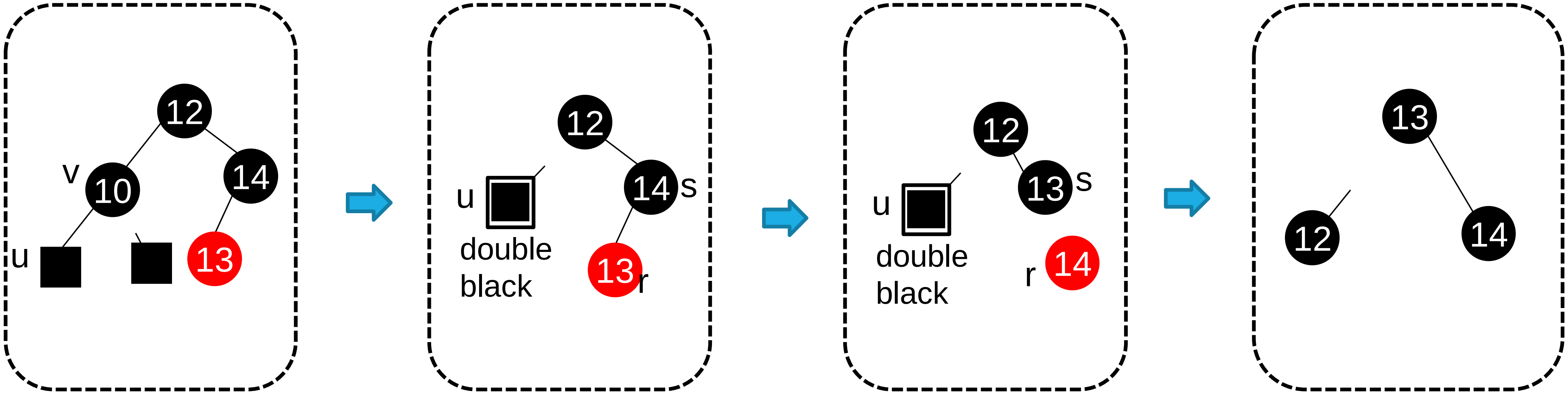
Case-I: If either v or u is red (delete 2)



Case-II: Double black: If both v and u are black (delete 10)



CASE-II CONTINUED...



delete 10

right left imbalance

right rotation & recolouring

left rotation & recolouring

C++ IMPLEMENTATION OF RBT

```
1 : Insert
2 : Find
3 : Erase
4 : Exit
Enter option : 1
Enter key and value : 10 Ramesh
Enter option : 1
Enter key and value : 20 Suresh
Enter option : 1
Enter key and value : 30 Radha
Enter option : 1
Enter key and value : 40 Shyam
Enter option : 2
Enter key to find : 30
Radha
Enter option : 3
Enter key to erase : 40
Enter option : 2
Enter key to find : 40
Key Not found
Enter option :
```

```cpp
void insert_case3(Node * inserted, Node * & root) {
  // parent and uncle are red, flip colors
  Node * my_uncle = uncle(inserted), * my_grandparent = grandparent(inserted);
  if(!my_uncle || !my_grandparent) return;
  // if there is no uncle or grandparent don't do anything
  if(is_red(my_uncle)) {
    inserted->parent->red = false;
    my_uncle->red = false;
    my_grandparent->red = true;
    insert_case1(my_grandparent, root);
  } else {
    insert_case4(inserted, root);
  }
}
```

```cpp
void delete_case2(Node * removed, Node * & root) {
  // node and brother are red, flip colors, make them black and parent red
  Node * my_brother = brother(removed);
  if(!my_brother) return;
  if(is_red(my_brother)) {
    removed->parent->red = true;
    my_brother->red = false;
    if(removed == removed->parent->left) rotate_with_right_child(removed->parent, root);
    else rotate_with_left_child(removed->parent, root);
  }
  delete_case3(removed, root);
}
```

THANK YOU!

Next class: Dividing and Conquer, …