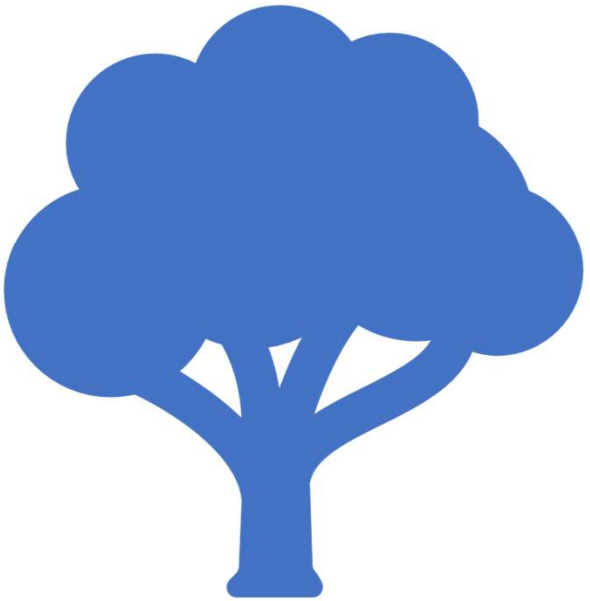
Advanced Data Structures and Algorithms

B-Tree | Quad Tree | Oct Tree



B-Tree

- Original B-Tree Proposed by R. Bayer and E. McCreigh in 1972.
- A B-Tree is a specialized multi-way tree designed especially for use on external disk.
- Improved versions of B-Trees were later proposed in 1982 by Huddleston and Mehlhorn, and by Maier and Salveter.
- B-tree variants are used mostly today as index structures in database applications.

Motivation for B-Tree

- Data is stored on disk in chunks (pages, blocks, allocation units) and the disk drive reads or writes a minimum of one page at a time.
- Index structures for large datasets cannot be stored in main memory
- Storing it on disk requires different approach to efficiency
- B-Tree nodes should correspond to a block of data
- Each node stores many data items and has many successors (stores addresses of successor blocks)

Motivation for B-Tree

- Assume that we use an AVL tree to store about 20 million records
- We end up with a very deep binary tree with lots of different disk accesses; $\log_2 20000000$ is about 24, so this takes about 0.2 seconds
- We know we can't improve on the $\log n$ lower bound on search for a binary tree
- But the solution is to use more branches and thus reduce the height of the tree!

As branching increases, depth decreases

Definition of B-Tree

A B-tree of order m is an m-way tree (i.e., a tree where each node may have up to m children) in which:

1. the number of keys in each non-leaf node is one less than the number of its children ($m - 1$) and these keys partition the keys in the children in the fashion of a search tree
2. all leaves are on the same level
3. all non-leaf nodes except the root have at least $\lceil m/2 \rceil$ children
4. the root is either a leaf node, or it has from two to m children
5. a leaf node contains no more than $m - 1$ keys

The number m should always be odd

Properties

If m is the order of the tree

- Every internal node has at most m children.
- Every internal node (except root) has at least $\lceil m / 2 \rceil$ children.
- The root has at least two children if it is not a leaf node.
- Every leaf has at most $m - 1$ keys
- An internal node with k children has $k - 1$ keys.
- All leaves appear in the same level

Total number of items in B-Tree

- The maximum number of items in a B-tree of order m and height h:

root	$m - 1$
level 1	$m(m - 1)$
level 2	$m^2(m - 1)$
. . .	
level h	$m^h(m - 1)$

- So, the total number of items is

$$(1 + m + m^2 + m^3 + \ldots + m^h)(m - 1) =$$

$$[(m^{h+1} - 1)/ (m - 1)] (m - 1) = \boldsymbol{m^{h+1} - 1}$$

- When $m = 5$ and $h = 2$ this gives $5^3 - 1 = 124$

Time Complexity

	Average Case	**Worst Case**
Search	$O(\log n)$	$O(\log n)$
Insertion	$O(\log n)$	$O(\log n)$
Deletion	$O(\log n)$	$O(\log n)$

Constructing a B-Tree

- Suppose we start with an empty B-tree and keys arrive in the following order:1 12 8 2 25 5 14 28 17 7 52 16 48 68 3 26 29 53 55 45
- We want to construct a B-tree of order 5
- The first four items go into the root:

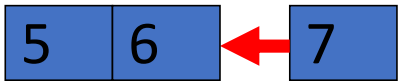
- To put the fifth item in the root would violate condition 5
- Therefore, when 25 arrives, pick the middle key to make a new root

Insertion in B-Tree

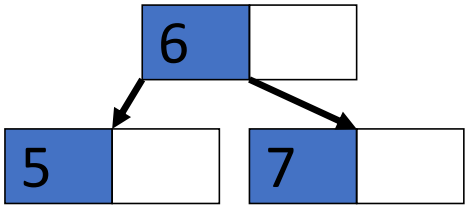
1. Attempt to insert the new key into a leaf
2. If this would result in that leaf becoming too big, split the leaf into two, promoting the middle key to the leaf's parent
3. If this would result in the parent becoming too big, split the parent into two, promoting the middle key
4. This strategy might have to be repeated all the way to the top
5. If necessary, the root is split in two and the middle key is promoted to a new root, making the tree one level higher

Two basic operations during insertion

- ***Split:***
  - When trying to add to a full node
  - Split node at median value



- ***Promote:***
  - Must insert root of split node higher up
  - May require a new split



Deletion in B-Tree

During insertion, the key always goes *into* a *leaf*. For deletion we wish to remove *from* a leaf. There are three possible ways we can do this:

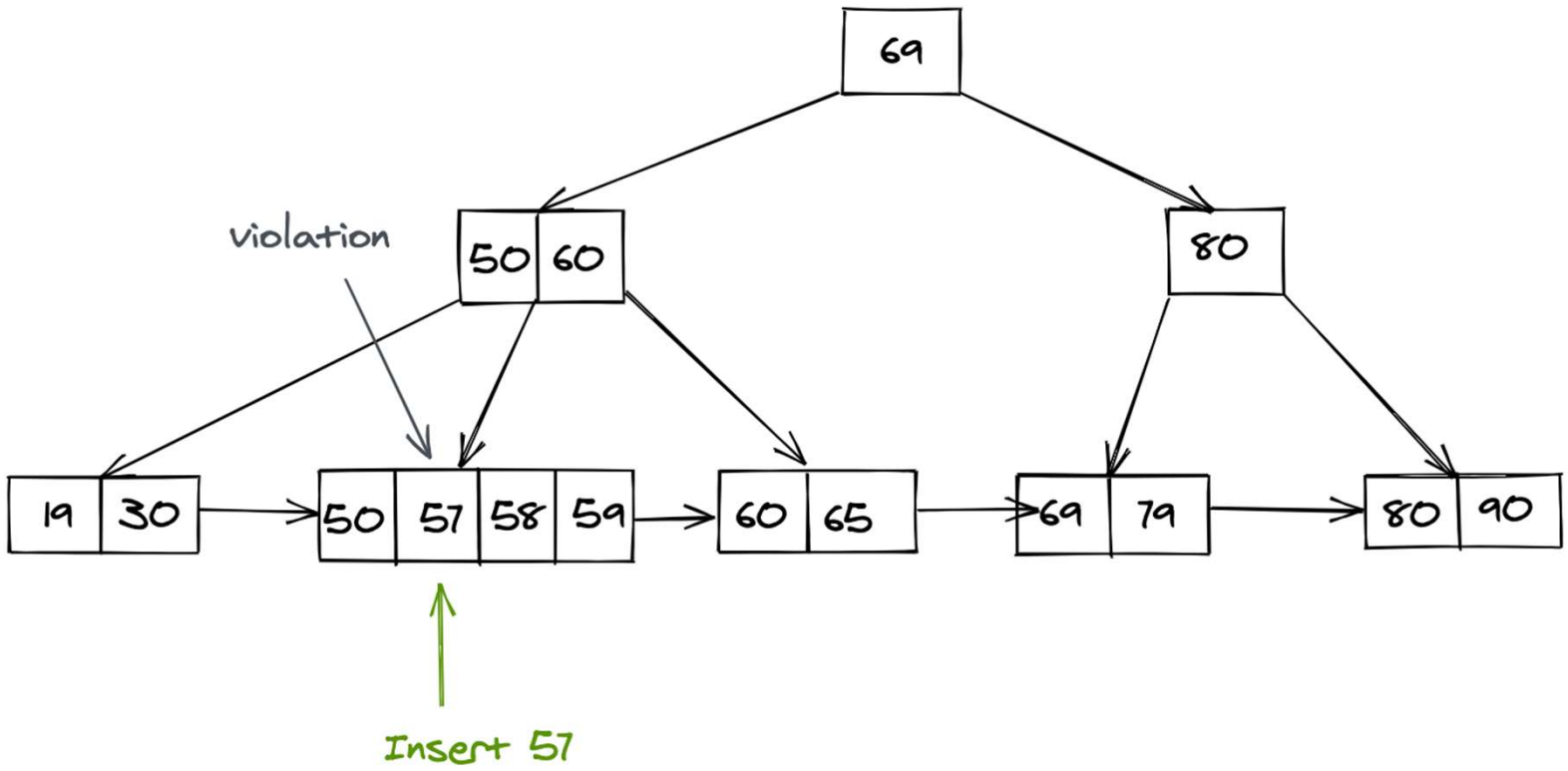
1. If **the key is already in a leaf node and** removing it doesn't cause that leaf node to have too few keys, then simply remove the key to be deleted.
2. If **the key is *not* in a leaf (internal node),** then it is guaranteed (by the nature of a B-tree) that its predecessor or successor will be in a leaf -- in this case we can delete the key and promote the predecessor or successor key to the non-leaf deleted key's position.

Deletion in B-Tree

If (1) or (2) **lead to a leaf node containing less than the minimum number of keys**, then we have to **look at the siblings** immediately adjacent to the leaf in question:

3. if one of them has more than the min. number of keys then we can promote one of its keys to the parent and take the parent key into our lacking leaf
4. if neither of them has more than the min. number of keys then the lacking leaf and one of its neighbours can be combined with their shared parent (the opposite of promoting a key) and the new leaf will have the correct number of keys; if this step leave the parent with too few keys then we repeat the process up to the root itself, if required

B+ Tree



Keys: 1 12 8 2 25 5 14 28 17 7 52 16
48 68 3 26 29 53 55 45 (Insert/Delete)

Deletion in B+ Tree

Case 1: Deleting a Key in a Leaf Node

- If the key to be deleted is in a leaf node, simply remove the key. No further adjustments may be needed if the node remains within the minimum occupancy limit.

Case 2: Underflow in Leaf Node

- If the deletion causes the number of keys in a leaf node to fall below the minimum occupancy, borrow keys from neighboring nodes or merge nodes to restore balance.

Case 3: Deleting a Key in an Internal Node

- If the key to be deleted is in an internal node, replace the key with its predecessor or successor from the child subtree.

Deletion in B+ Tree

Case 4: Adjustment in Internal Nodes

- After deleting a key in an internal node, update the parent nodes to reflect any changes in the keys or children pointers.
- If the adjustment causes underflow in an internal node, borrow keys from neighboring nodes or merge nodes to maintain balance.

Case 5: Adjusting Parent Pointers

- If merging or borrowing occurs in an internal node, update the parent pointers to reflect any changes in the structure of the child nodes.

Case 6: Underflow in Root Node

- If the root node becomes empty after deletion (i.e., all keys are removed), update the root to be the merged or borrowed node, adjusting the tree height.

Deletion in B+ Tree

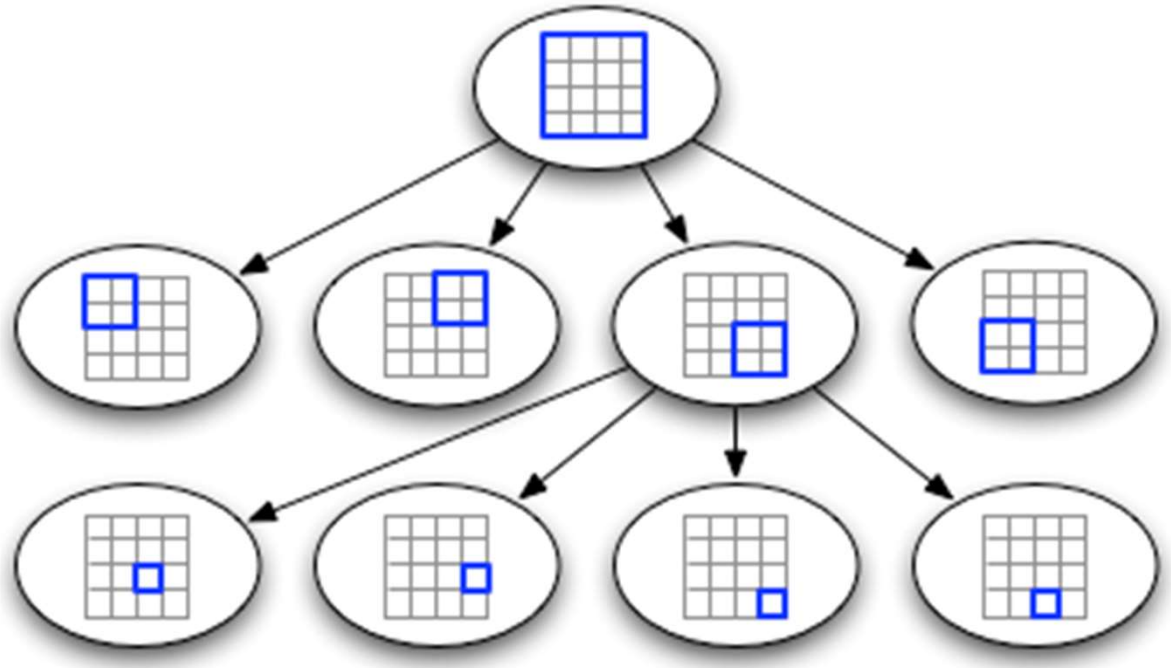
Case 7: Adjustment Propagation

- Ensure that any adjustments made in internal nodes due to deletion are propagated up the tree to the root to maintain balance.

Case 8: Final Adjustment

- Perform a final check to ensure that the B+ tree properties, such as minimum occupancy and order, are maintained after all adjustments.

QUAD TREE



Definition of Quad Tree

- Tree data structure that is used to represent two-dimensional space.
- The root node represents the entire space, and each child node represents a smaller portion of the space that is **divided into four quadrants**.
- This process of subdivision continues until each node represents a single point in space.
- Each node has **at most four children**.
- Quad Tree can also be constructed from the two-dimensional area.

Definition in relation to Point and Region Quad Trees

Types of Quad Trees

1. **Point Quad Tree**:
   - Designed to store points in a 2D space.
   - Each node in the tree represents a quadrant of the space, and points are stored in the appropriate quadrant.
   - Example: *A point quad tree representing cities on a map, with each node dividing the map into quadrants and storing cities in the appropriate quadrant.*

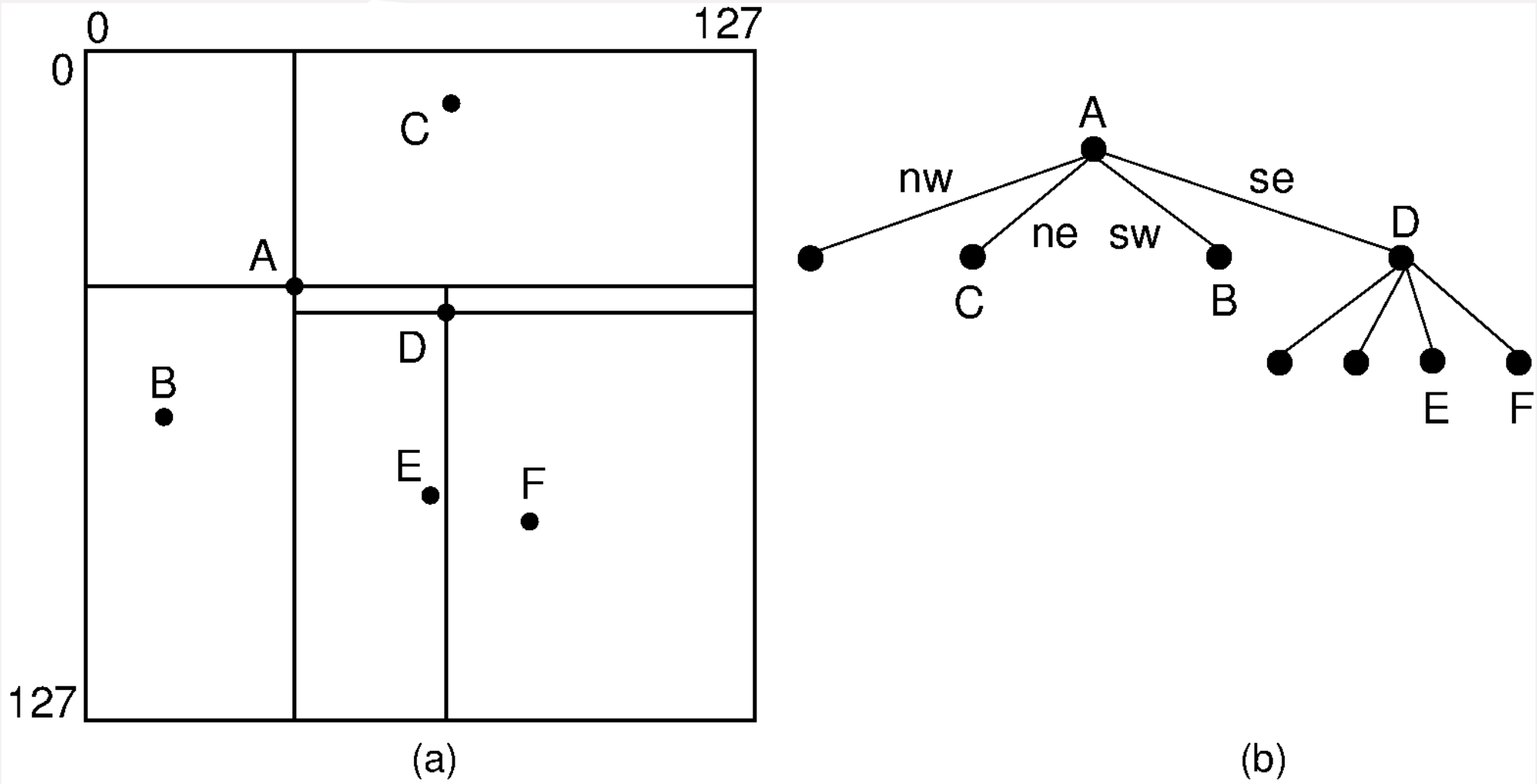
Types of Quad Tree

1. **Point Quad Tree**:

   When dealing with point quad trees and a point falls exactly on the boundary line between two quadrants, it can lead to ambiguity in determining the quadrant for that point. To resolve this situation, you can employ a common strategy known as the "**tie-breaking**" mechanism.

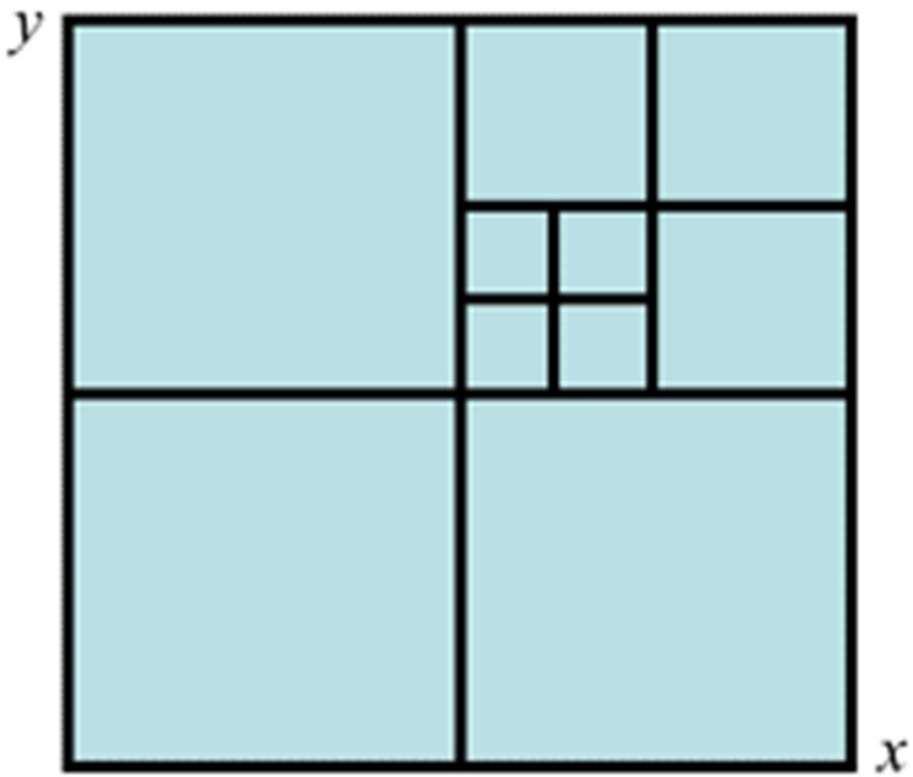
Types of Quad Tree

- **Tie Breaking Rule:**
  Choose a rule to break ties when a point lies exactly on the boundary. For instance, you can decide that if a point is on the boundary, it belongs to the quadrant to its right or the quadrant above it.

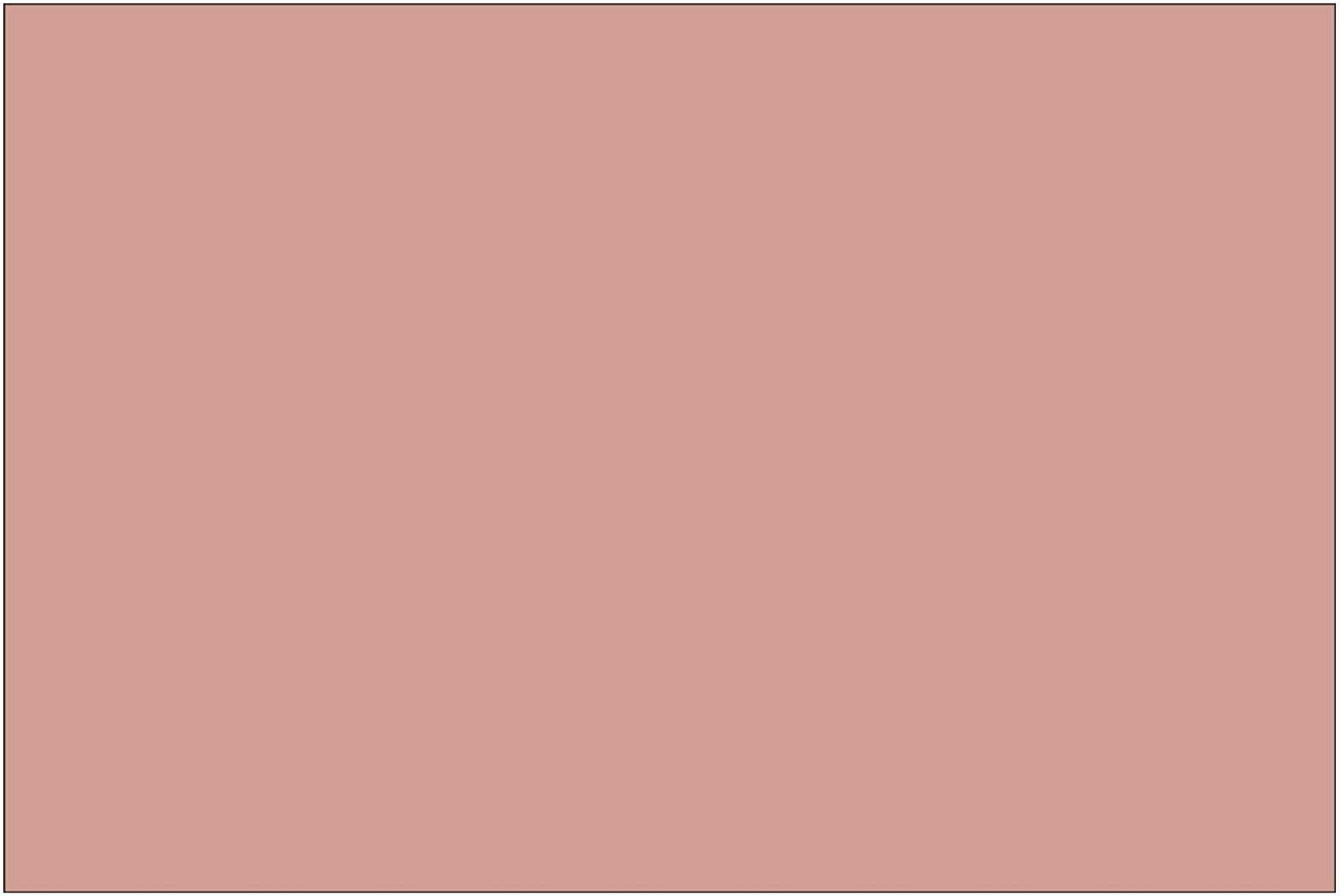

0
127
0
127
C
A
B
D
E
F
(a)
A
nw
ne
sw
se
C
B
D
E
F
(b)

Types of Quad Trees

2. **Region Quad Tree**:
   - Unlike a point quad tree, a region quad tree is used for storing regions or rectangles in a 2D space.
   - Each node in the tree represents a quadrant, and regions are stored based on their spatial location within these quadrants.
   - Example: *A region quad tree for a computer-generated image, where each node partitions the image, and regions (rectangles) are stored in the corresponding quadrants.*



Quadtree of subregions



Types of Quad Trees

3. **PR Quad Tree (Point-Region Quad Tree)**:
   - A hybrid quad tree that can store both points and regions.
   - The tree is divided into quadrants, and each quadrant can either contain a point or a region.
   - Example: *A PR quad tree representing a geographic area with cities (points) and parks (regions), where each quadrant can contain either a city or a park.*

Types of Quad Trees

4. **Hybrid Quad Tree**:
   - This type of quad tree can be a combination of different types, allowing it to adapt to various data types and applications.
   - For example, it might store points in some nodes and regions in others based on the characteristics of the data.
   - Example: *A hybrid quad tree handling a dataset with weather station measurements (points) and weather zones (regions), dynamically adapting its structure based on data distribution.*

Types of Quad Trees

5. **Adaptive Quad Tree**:
   - An adaptive quad tree adjusts its structure dynamically based on the distribution of data.
   - Nodes can split or merge as needed to maintain an efficient representation of the spatial information.
   - Example: *An adaptive quad tree dynamically adjusting to changes in a landscape over time, splitting nodes in densely populated areas and merging nodes in sparsely populated areas.*

Types of Quad Trees

1. **PR Octree (Point-Region Octree)**:
   - Similar to the PR Quad Tree but extended to three-dimensional space, the PR Octree can store points and regions in a 3D environment.
   - Example: *A PR octree in a 3D space representing a city with buildings (points) and zones (3D regions), adapting to the spatial distribution of both points and regions.*

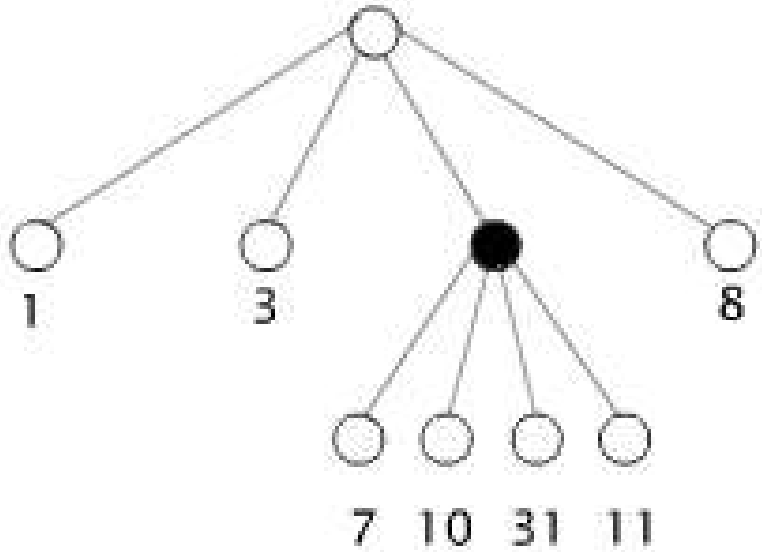
Construction of Quad Tree

Quad Tree can be constructed following the steps below:

1. **Divide** the current two-dimensional space into four quadrants/boxes.
2. If a quadrant/box contains one or more points in it, create a child node.
3. If a quadrant/box does not contain any points, do not create a child for it.
4. Recurse for each of the children.

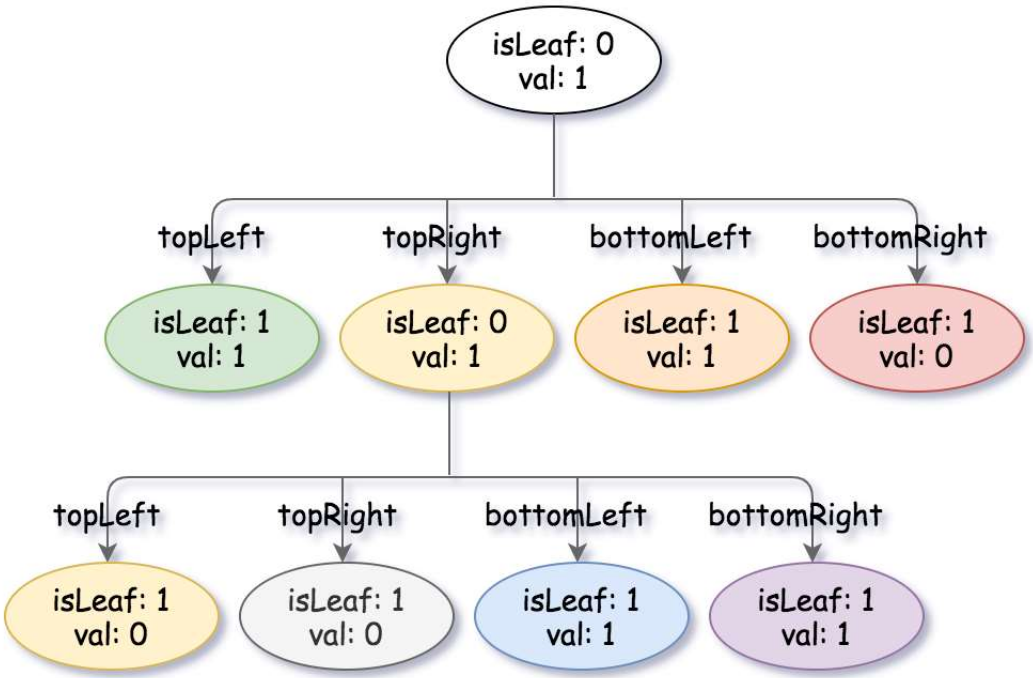
Source: https://www.geeksforgeeks.org/quad-tree/

Region Quad Tree Representation of $n \times n$ 2d matrix



Region Quad Data Structure Representation

1	1	1	1	0	0	0	0
1	1	1	1	0	0	0	0
1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1
1	1	1	1	0	0	0	0
1	1	1	1	0	0	0	0
1	1	1	1	0	0	0	0
1	1	1	1	0	0	0	0



Node Structure (Region Quad Tree)

```
class Node:
    def __init__(self, val, isLeaf, topLeft, topRight, bottomLeft, bottomRight):
        self.val = val
        self.isLeaf = isLeaf
        self.topLeft = topLeft
        self.topRight = topRight
        self.bottomLeft = bottomLeft
        self.bottomRight = bottomRight
```

Source: https://leetcode.com/

```python
class QuadTree:
    def construct(self, grid: List[List[int]]) -> 'Node':
        def dfs(a, b, c, d):
            zero = one = 0
            for i in range(a, c + 1):
                for j in range(b, d + 1):
                    if grid[i][j] == 0:
                        zero = 1
                    else:
                        one = 1
            isLeaf = zero + one == 1
            val = isLeaf and one
            if isLeaf:
                return Node(grid[a][b], True)
            topLeft = dfs(a, b, (a + c) // 2, (b + d) // 2)
            topRight = dfs(a, (b + d) // 2 + 1, (a + c) // 2, d)
            bottomLeft = dfs((a + c) // 2 + 1, b, c, (b + d) // 2)
            bottomRight = dfs((a + c) // 2 + 1, (b + d) // 2 + 1, c, d)
            return Node(val, isLeaf, topLeft, topRight, bottomLeft, bottomRight)

        return dfs(0, 0, len(grid) - 1, len(grid[0]) - 1)
```

Benefits of Quad Tree

- They are efficient for storing and retrieving spatial data.
- They can be used to quickly search for data within a given area.
- They can be used to compress images.

Limitations of Quad Tree

- They can be complex to implement.
- They can be inefficient for storing data that is not evenly distributed.
- They can be difficult to update if the data changes frequently.

Uses of Quad Trees

- **Spatial Partitioning:**
  Quad trees are excellent for partitioning a two-dimensional space into regions or quadrants. This spatial partitioning allows for efficient organization and retrieval of data based on its spatial location.

- **Point Location and Retrieval:**
  Quad trees are often employed for quick point location and retrieval. They enable fast searches for points in a given region or quadrant of the space, making them ideal for applications like geographic information systems (GIS), computer graphics, and image processing.

Uses of Quad Trees

- **Collision Detection:**
  In computer graphics and physics simulations, quad trees can be used for efficient collision detection. By organizing spatial data hierarchically, it becomes easier to identify potential collisions between objects in a scene.
- **Nearest Neighbor Search:**
  Quad trees are useful for performing nearest neighbor searches efficiently. When you need to find the closest point or object to a given location, quad trees can significantly reduce the search space, speeding up the process.

Uses of Quad Trees

- **Image Compression:**
  In image processing, quad trees can be employed for image compression. By recursively subdividing regions of an image until a certain criterion is met (e.g., a minimum block size), quad trees can represent images in a more compact way, especially in scenarios with homogeneous regions.
- **Terrain Modeling:**
  Quad trees are valuable for representing and processing terrain data. In applications like computer games or geographical simulations, quad trees can be used to manage and optimize the rendering of terrain, focusing detailed rendering where needed.

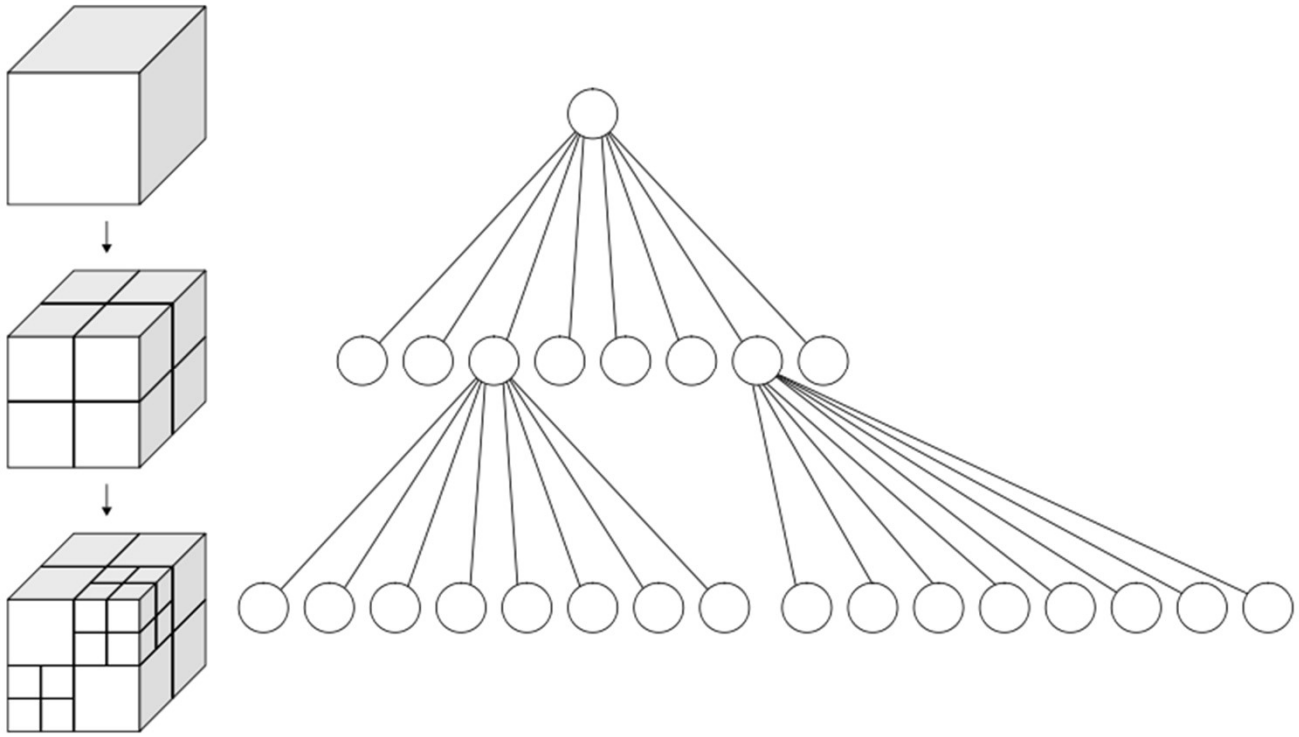
Uses of Quad Trees

- **Dynamic Spatial Indexing:**
  Quad trees can be adapted for dynamic scenarios where spatial data is constantly changing. By employing techniques like dynamic tree balancing, quad trees can efficiently handle insertions and deletions of data points without the need for frequent reconstruction.
- **Ray Tracing:**
  In computer graphics and rendering, quad trees are utilized for accelerating ray tracing algorithms. They help in quickly identifying relevant portions of a scene for ray-object intersection tests, improving rendering performance.

Uses of Quad Trees

- **Fractal Generation:**
  Quad trees are instrumental in generating fractal patterns. By recursively subdividing regions of interest, quad trees can be used to create intricate and detailed fractal structures.

Octree



Octree

- Each node in an octree represents a cubic region of space.
- Nodes in the octree can either be internal or leaf nodes.
- Internal nodes have eight children, each corresponding to a sub-octant, while leaf nodes contain actual data or objects.
- The octree is traversed based on the spatial location of the data.

Octree

- An octree is a hierarchical tree data structure that recursively subdivides three-dimensional space into octants.
- Each octant corresponds to one-eighth of the total space within a parent octant.
- This hierarchical structure allows for efficient spatial organization and retrieval of data.

Uses of Octree

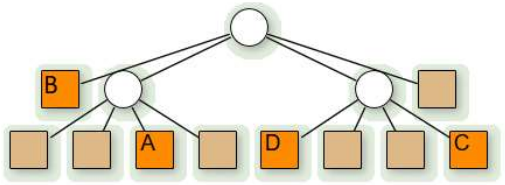
- Octrees find extensive use in 3D graphics and simulation applications.
- They are employed for tasks such as collision detection, ray tracing, spatial indexing of objects in a 3D scene, and
- Efficient representation of volumetric data like medical imaging or geological models.

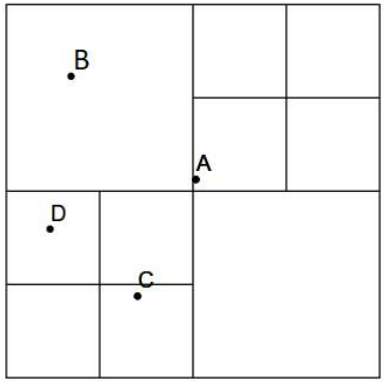
PR Quad Tree

Three types of nodes are used in PR quadtree:

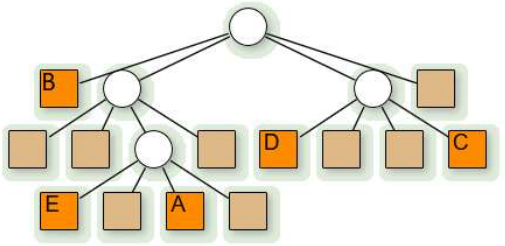
- **Point node**: Used to represent of a point. Is always a leaf node.
- **Empty node**: Used as a leaf node to represent that no point exists in the region it represent.
- **Region node**: This is always an internal node. It is used to represent a region.
- A region node always have 4 children nodes that can either be a point node or empty node.

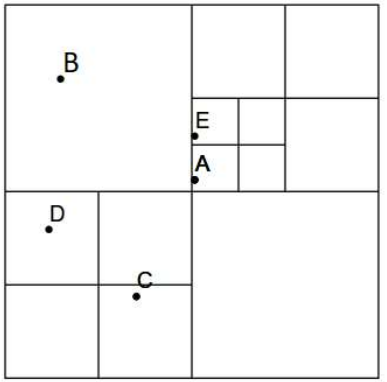
PR Quad Tree





Insert E





PR Quad Tree (Insertion in $O(\log n)$)

1. Start with root node as current node.
2. If the given point is not in boundary represented by current node, stop insertion with error.
3. Determine the appropriate child node to store the point.
4. If the child node is empty node, replace it with a point node representing the point. Stop insertion.
5. If the child node is a point node, replace it with a region node. Call insert for the point that just got replaced. Set current node as the newly formed region node.
6. If selected child node is a region node, set the child node as current node.
7. Goto step 2.

PR Quad Tree (Search in $O(\log n)$)

1. Start with root node as current node.
2. If the given point is not in boundary represented by current node, stop search with error.
3. Determine the appropriate child node to search the point.
4. If the child node is empty node, return FALSE.
5. If the child node is a point node and it matches the given point return TRUE, otherwise return FALSE.
6. If the child node is a region node, set current node as the child region node.
7. Goto step 2.

Practice Problems

Q1. Construct AVL Tree with the following keys:

1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15

Q2. From the resultant AVL tree, perform deletion (in order of the keys)

13, 6, 12, 10, 9, 14, 15

Q3. Construct B-Tree (m=5) from the following input keys:

89, 82, 50, 79, 45, 21, 9, 71, 18, 56, 25, 38, 76, 3, 46

Also, show deletion process highlighting all the cases.

Q4. Construct B+ Tree (m=5) with the above input data. Pick the keys to perform deletion process highlighting all the deletion cases.

Practice Problems

Q5. Given a 2D space of size 20x20, construct a quad tree with the following points.

(8, 1), (0, 1), (5, 9), (0, 6), (2, 0), (1, 4), (5, 4), (8, 2), (9, 3), (7, 0)

Q6. Construct a quad tree with the following input 2D space.

0	0	1	1	0	1	0	0
0	1	0	0	1	1	0	0
0	0	0	0	0	1	1	1
0	0	0	0	1	1	0	1
0	1	1	0	0	1	0	1
0	0	0	1	0	1	0	1
0	0	0	0	0	1	1	0
0	0	0	0	0	1	1	1

Practice Problems

Q5. Given a 2D space of size 20x20, construct a PR quad tree with the following points.

(8, 1), (0, 1), (5, 9), (0, 6), (2, 0), (1, 4), (5, 4), (8, 2), (9, 3), (7, 0)