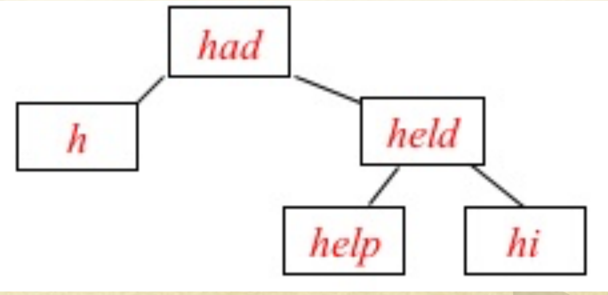
Data Structures

Giri Narasimhan
Office: ECS 254A
Phone: x-3748
giri@cs.fiu.edu

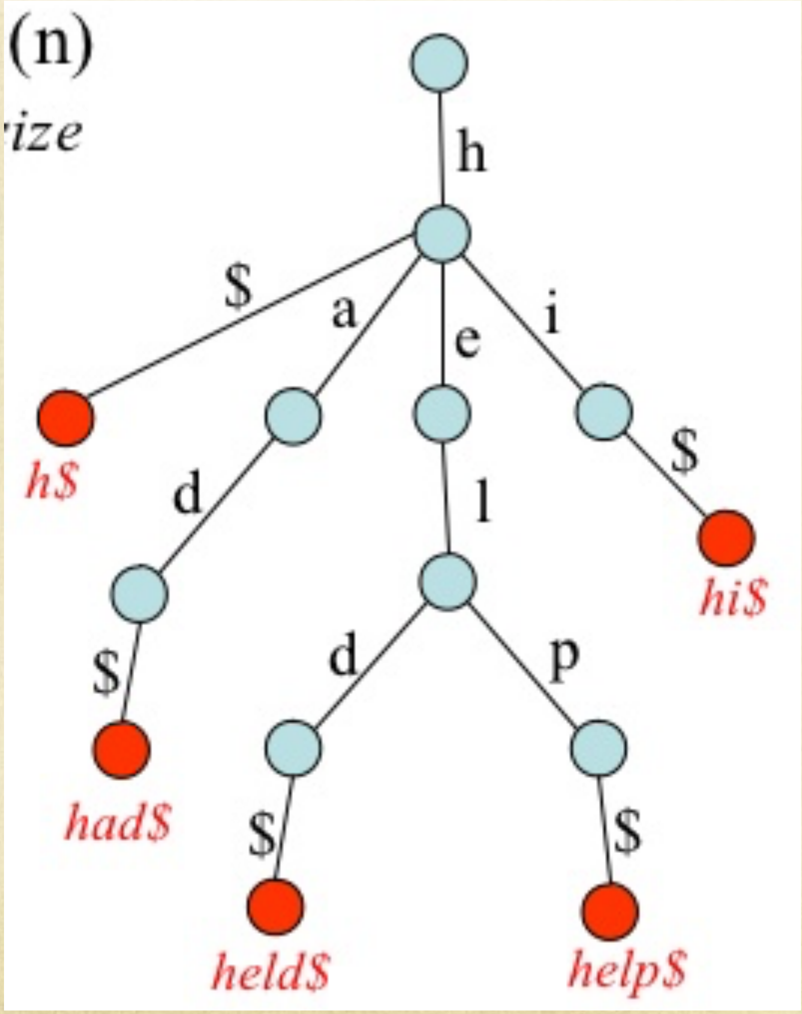
Suffix Arrays and Suffix Trees

String Data Structures

- If we have many strings (instead of int or double) to store, how can we facilitate search?
  - Store in a tree and use "compareTo" or "equals" methods
    - Time complexity is O(log N) where N is size of database
  - Better option: Hashing
    - Hashing takes O(s) time, where s is length of query string
    - Search can be done in O(s) time on the average
- If N is very large (dictionary)?
  - Hashing is much better than trees
  - But, no worst-case guarantees



Tries



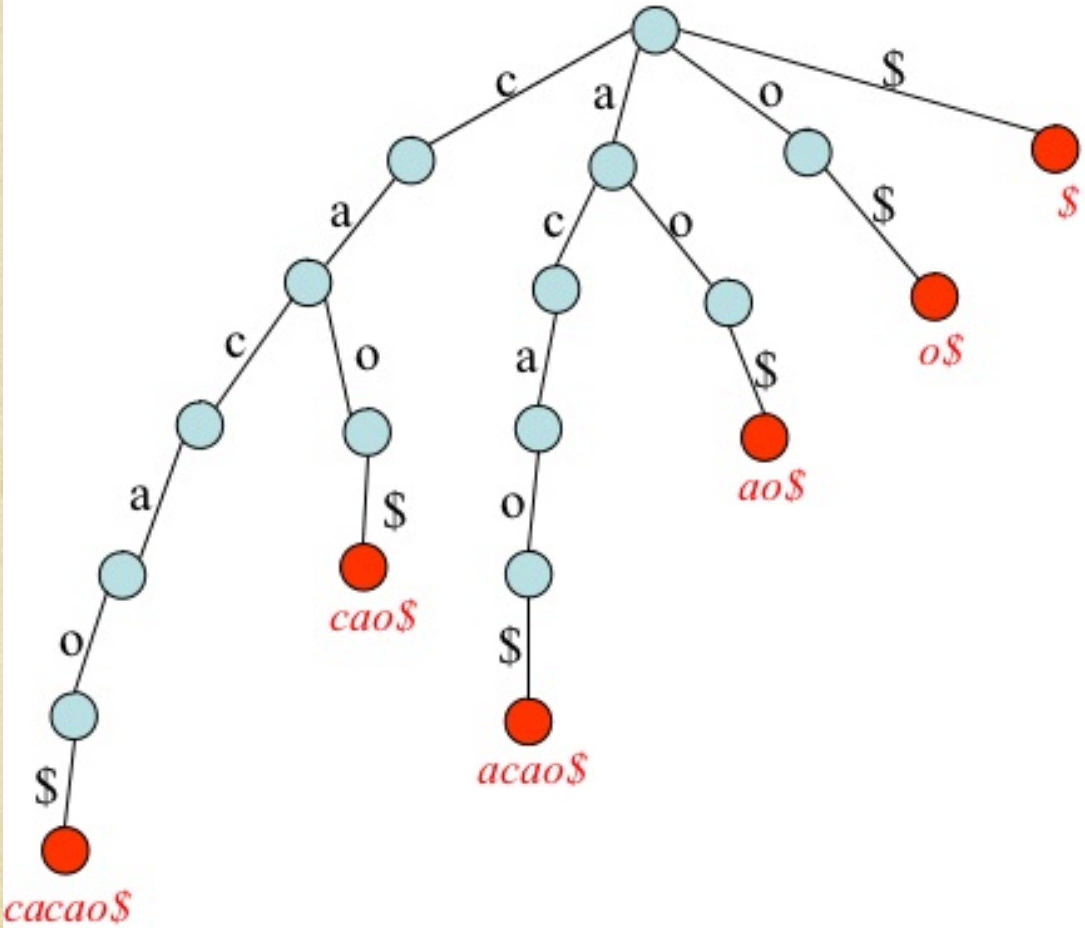
- Search can be done in O(s) time in the worst case
  - Assume that alphabet size is small
  - Otherwise, branching becomes expensive
- Space = $O(S_{total})$
- All terminal nodes are marked
- Not all terminal nodes are leaves

http://www.slideshare.net/zukun/advances-in-discrete-energy-minimisation-for-computer-vision

Other String searches

- What if we also want substring searches also?
  - Use a tree
    - Go left or right for next comparison?
    - It does not help (e.g., we have only 1 long string)
  - Use "contains"
    - Need to look at every string in database – expensive!
    - Concatenate all strings in database before search
      - What if we search many, many times
- What if we have a few long strings to store and many, many substring searches to make?
  - Suffix Arrays and Suffix Trees

Storing suffixes of “cacao$”



Suffixes
cacao$
acao$
cao$
ao$
o$
$

Sorted Suffixes
$
acao$
ao$
cacao$
cao$
o$

Substrings of "mississippi"

mississippi

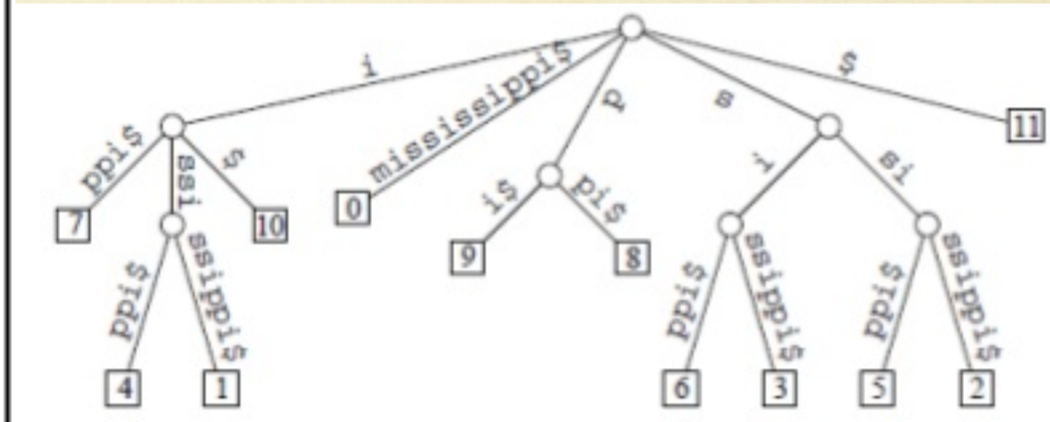
ε	11
i	10
ippi	7
issippi	4
ississippi	1
mississippi	0
pi	9
ppi	8
sippi	6
sissippi	3
ssippi	5
ssissippi	2

Obtain all suffixes of "mississippi"

- mississippi
- ississippi
- ssissippi
- sissippi
- issippi
- ssippi
- sippi
- ippi
- ppi
- pi
- i
- ε

Suffix Arrays vs Suffix Trees

ε	11
i	10
ippi	7
issippi	4
ississippi	1
mississippi	0
pi	9
ppi	8
sippi	6
sissippi	3
ssippi	5
ssissippi	2



http://image.slidesharecdn.com/advancesindiscreteenergyminimisationforcomputervision-121028223132-phpapp02/95/advances-in-discrete-energy-minimisation-for-computer-vision-30-638.jpg?cb=1351463578

More Details at:

- http://web.stanford.edu/class/cs97si/suffix-array.pdf

FIU Team

- FIU Football/Soccer/Basketball Team
- FIU Programming Team
  - Represent FIU at competitions
  - SCM Southeast Regional Programming Competition
    - Part of ACM ICPC
  - This year we have 12 team members & many trainees
  - Main focus of problems in competition
    - Data Structures & Efficient Algorithms
  - Train every Thursday from 3:30 – 4:45 PM
  - Also train on other days (varies with semester)
- Do you want to be an elite **Team Member**?

http://academy.cis.fiu.edu/team/

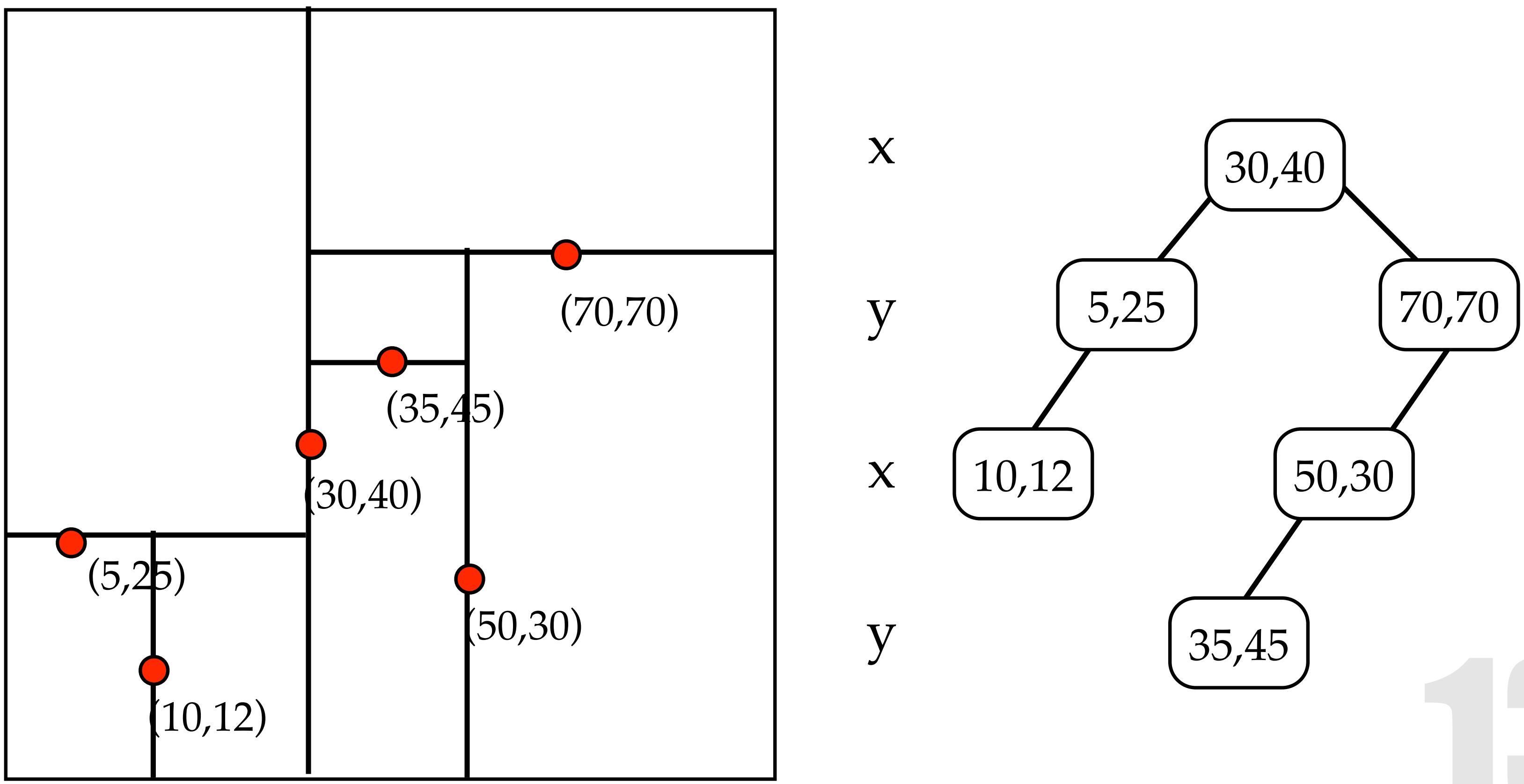
kD-Trees

Storing/Retrieving Points

- How do we solve geographical problems?
  - I am at 25.7617° N 80.1918° W
  - What is my nearest …
    - Post office, Burger King, Gas Station
  - Within a 5 mile radius, find all …
    - Post offices, Chinese restaurants, Uber Taxis
  - Where should I locate the next
    - Fire station, public school, …
- If all points are on a line, use SortedArray or AVL tree

Insert points into kD-tree

insert: (30,40), (5,25), (10,12), (70,70), (50,30), (35,45)



Can be easily generalized to higher dimensions

Insert Operation

```
insert(Point x, KDNode t, int cd) {
    if t == null    // empty tree
        t = new KDNode(x);
    else if (x == t.data)
        // error! duplicate
    else if (x[cd] < t.data[cd])
        t.left = insert(x, t.left, (cd+1) % DIM);
    else
        t.right = insert(x, t.right, (cd+1) % DIM);
    return t;
}
```

Nearest Neighbor

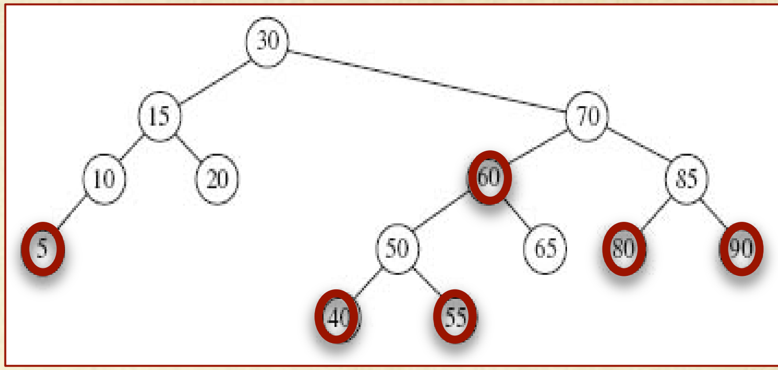
```
def NN(Point Q, kdTree T, int cd, Rect BB):

   // if this bounding box is too far, do nothing
   if T == NULL or distance(Q, BB) > best_dist: return

   // if this point is better than the best:
   dist = distance(Q, T.data)
   if dist < best_dist:
      best = T.data
      best_dist = dist
   // visit subtrees is most promising order:
   if Q[cd] < T.data[cd]:
      NN(Q, T.left, next_cd, BB.trimLeft(cd, t.data))
      NN(Q, T.right, next_cd, BB.trimRight(cd, t.data))
   else:
      NN(Q, T.right, next_cd, BB.trimRight(cd, t.data))
      NN(Q, T.left, next_cd, BB.trimLeft(cd, t.data))
```

Following Dave Mount's Notes (page 77)

Search time: O(log n)

Red-Black Trees

Red-Black Trees



1. It is a binary search tree
2. Every node is colored red/black
3. Root is always black
4. Parent of red node is always black
5. Every path from root to a leaf has same number of black nodes and is called the Black Height of the tree

- **Consequences of rule 5:**
  - Height of tree < 2 log (N+1)
  - Search is O(log N)

Less strict than AVL trees. Thus, fewer rotations, but greater height

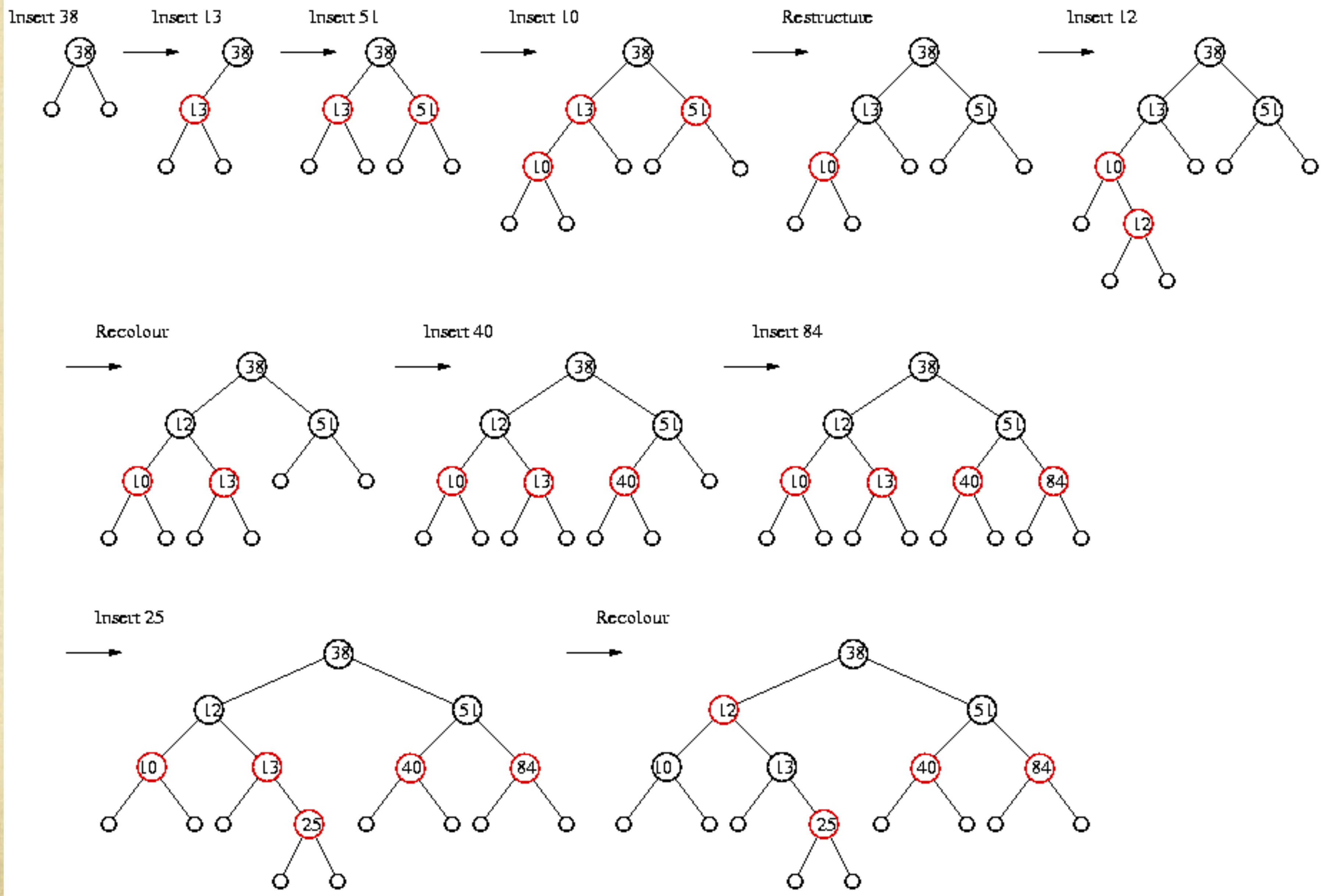
Insert Operation

- First apply BST insert and color new node as red (unless it is the root)
  - Rules 1, 2, 3 and 5 are fine.
  - Rule 4 may be violated and needs to be fixed
- Let's try the animation
  - https://www.cs.usfca.edu/~galles/visualization/RedBlack.html

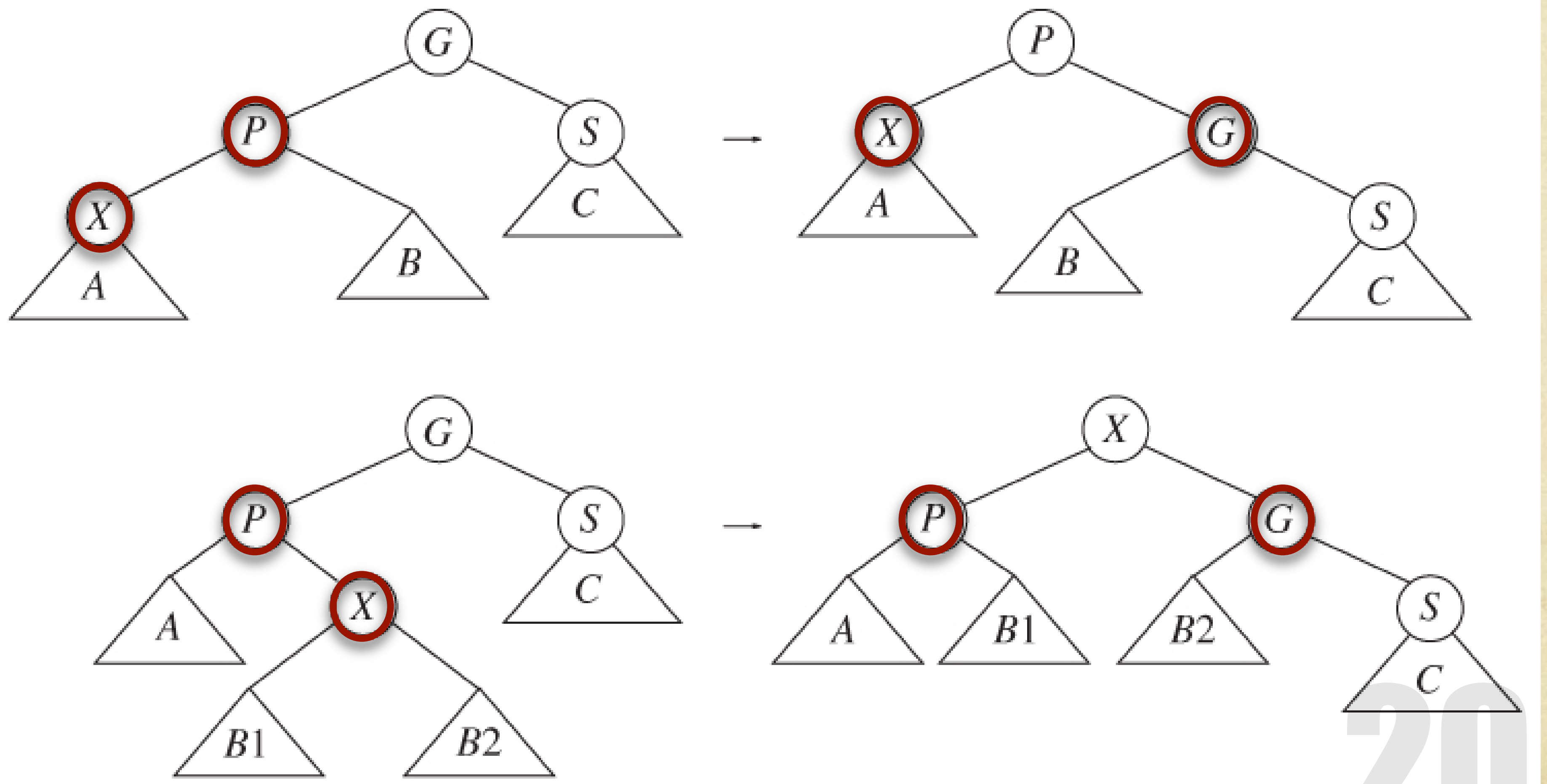
Red–Black Tree Example

http://www.csse.canterbury.ac.nz/research/RG/alg/rbtree.gif

Insertions: 38, 13, 51, 10, 12, 40, 84, 25



Insert: 2 cases



RBNode

```
private static class RBNode<AnyType> {
    AnyType               element;
    RBNode<AnyType>       left;
    RBNode<AnyType>       right;
    int                   color;

    // constructors
        ...
}
```

- 2 special nodes
  - nullNode
    - All null pointers point to this node
  - Root sentinel
    - Extra node whose right child is the real root