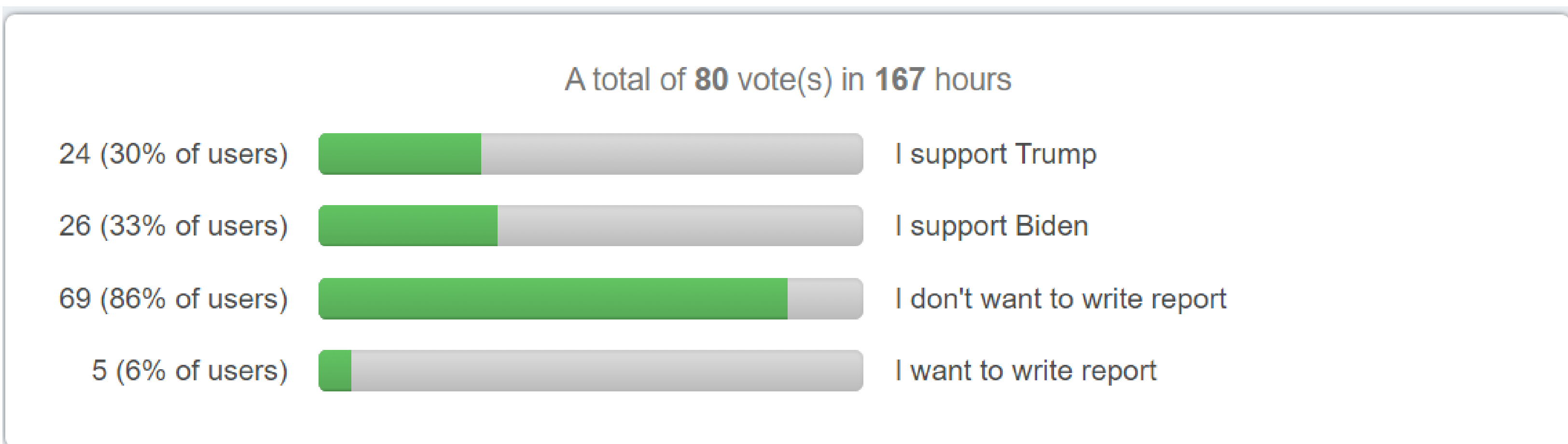
VE281 Midterm Distribution

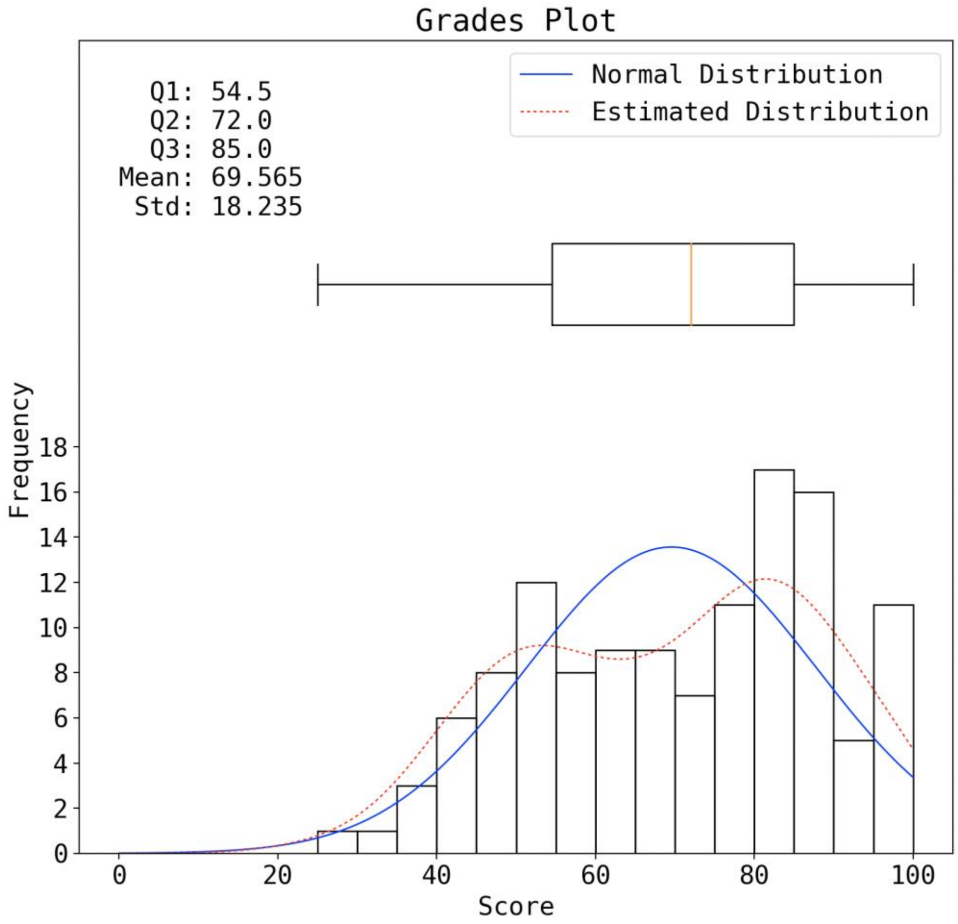
Biden Won Afterall

- How Similar to our Piazza Poll!
  - Trump was ahead but Biden had a cameback

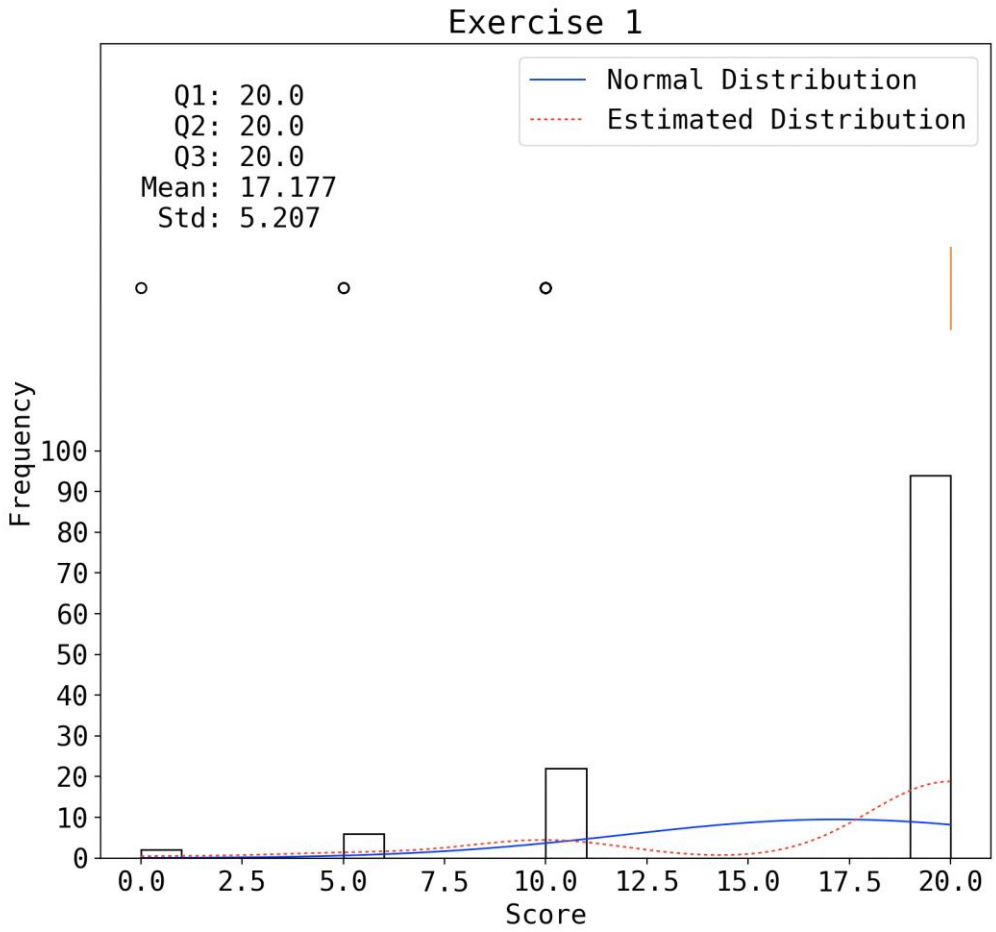
- Don’t give up too early!



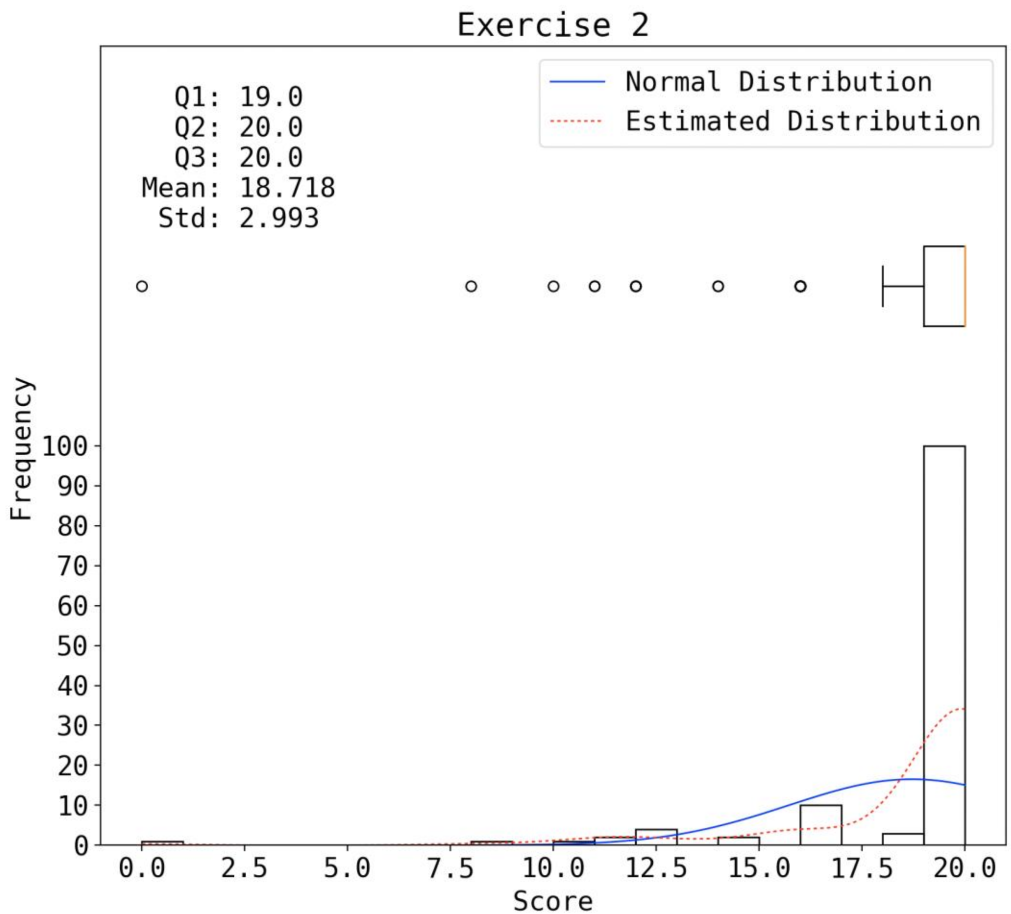
Overall Distribution



P1

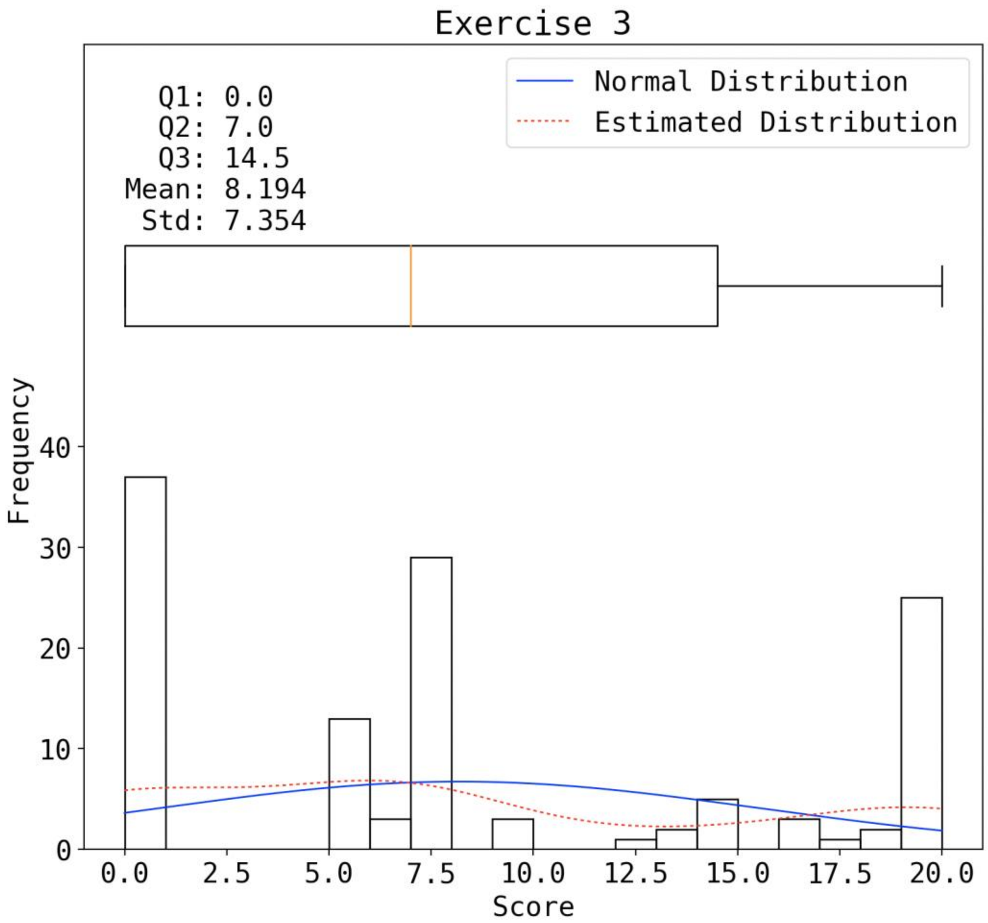

Exercise 1
Q1: 20.0
Q2: 20.0
Q3: 20.0
Mean: 17.177
Std: 5.207
Normal Distribution
Estimated Distribution
Frequency
100
90
80
70
60
50
40
30
20
10
0
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
Score

P2



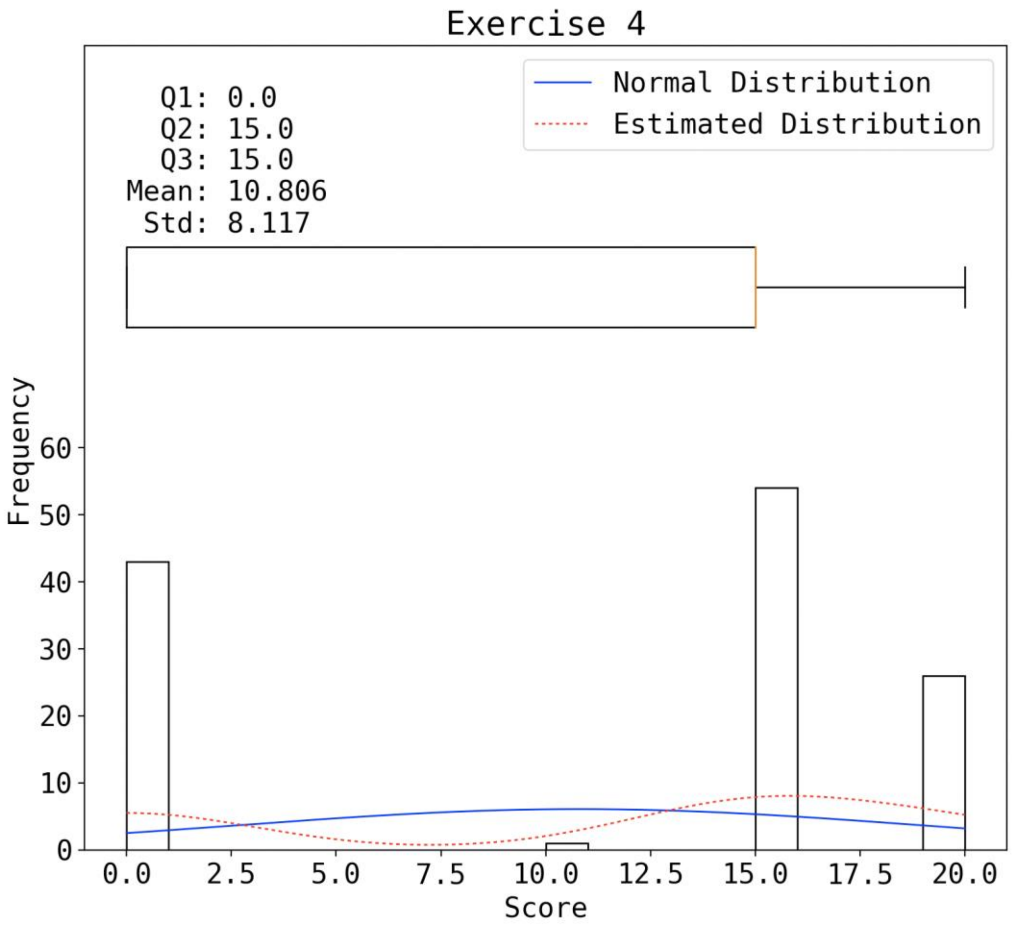
Exercise 2
Normal Distribution
Estimated Distribution
Q1: 19.0
Q2: 20.0
Q3: 20.0
Mean: 18.718
Std: 2.993
Frequency
100
90
80
70
60
50
40
30
20
10
0
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
Score

P3



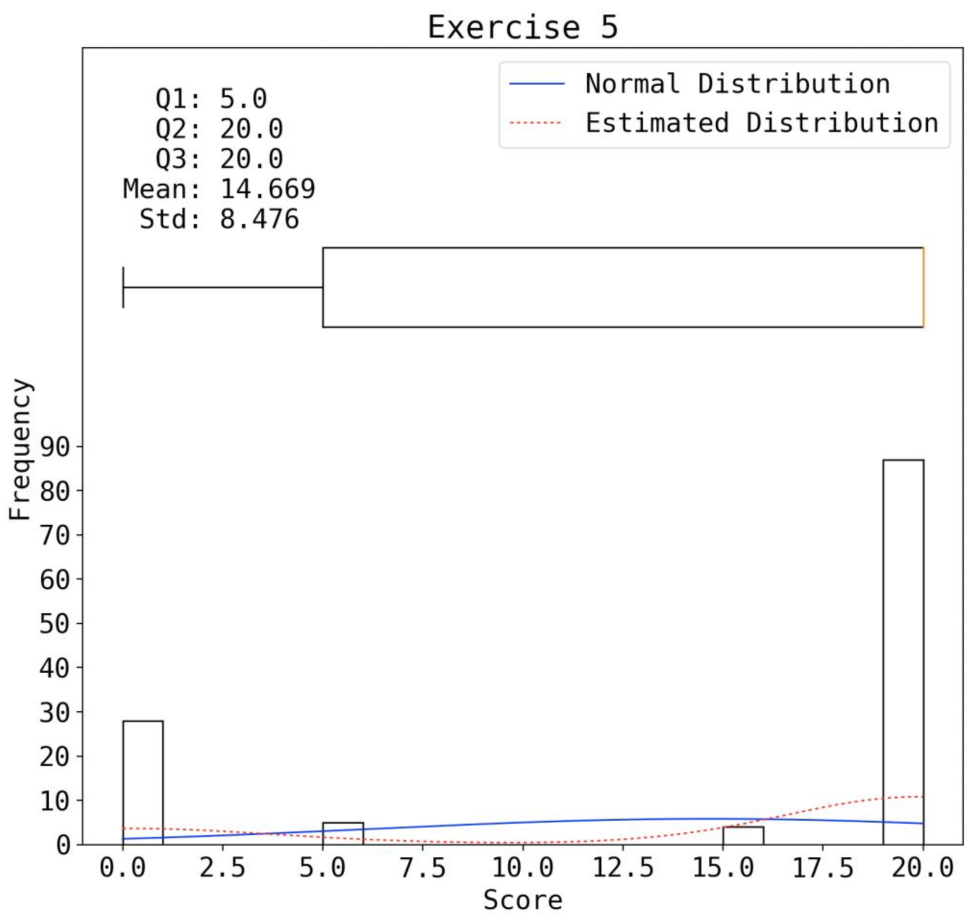
Exercise 3
Normal Distribution
Estimated Distribution
Q1: 0.0
Q2: 7.0
Q3: 14.5
Mean: 8.194
Std: 7.354
Frequency
0
10
20
30
40
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
Score

P4



Exercise 4
Normal Distribution
Estimated Distribution
Q1: 0.0
Q2: 15.0
Q3: 15.0
Mean: 10.806
Std: 8.117
Frequency
0
10
20
30
40
50
60
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
Score

P5

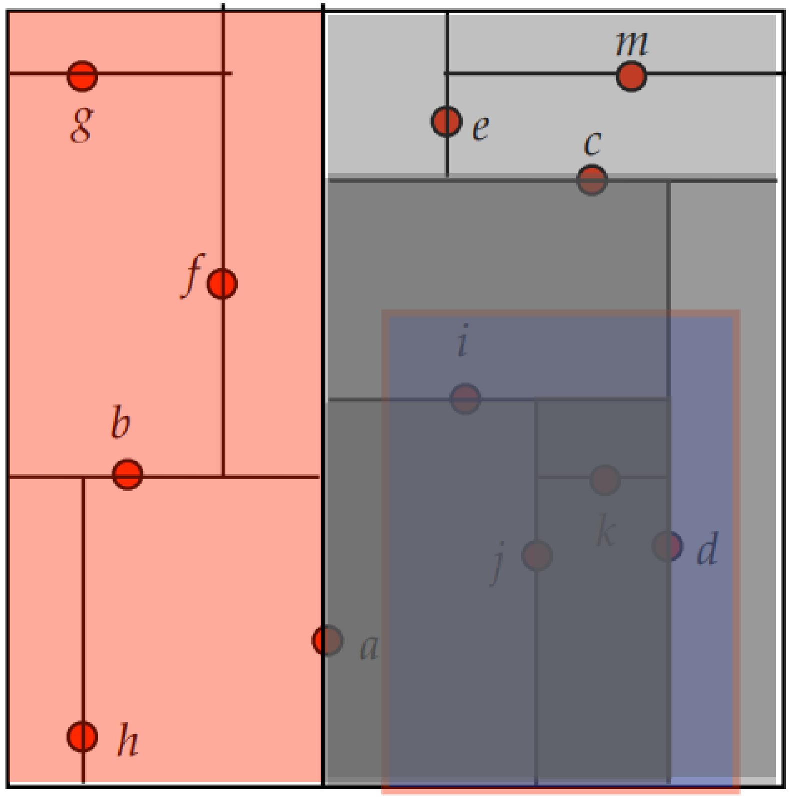

Exercise 5
Normal Distribution
Estimated Distribution
Q1: 5.0
Q2: 20.0
Q3: 20.0
Mean: 14.669
Std: 8.476
Frequency
0
10
20
30
40
50
60
70
80
90
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
Score

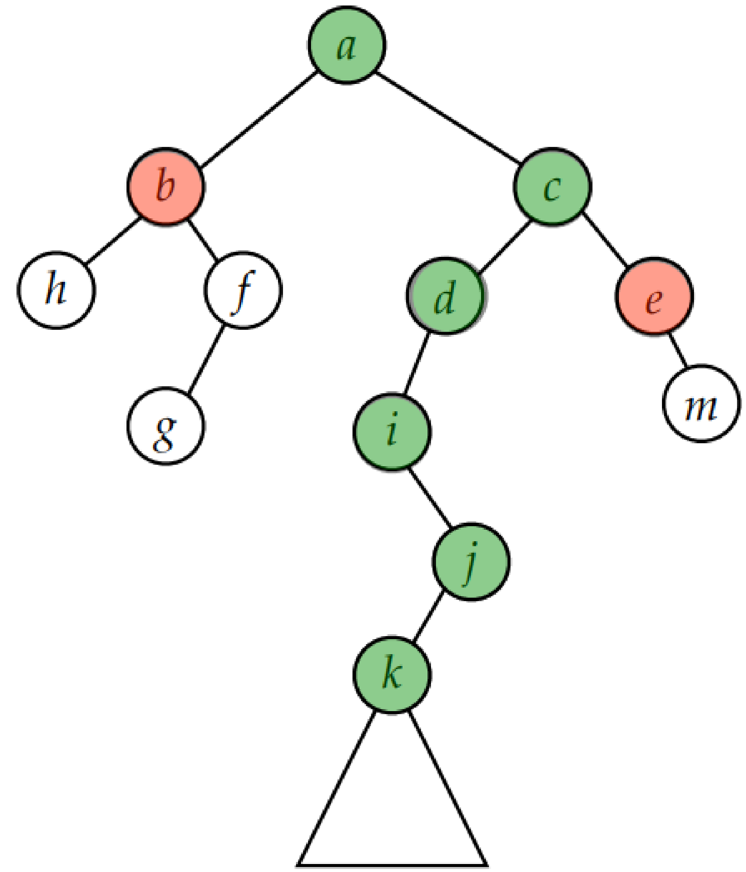
Regrading

- This Friday RC time
  - 8-10 PM
  - Location: JI Building, Room 326H

Ranged Search in K-d Tree

Range Searching Example





If query box doesn't overlap bounding box, stop recursion

If bounding box is a subset of query box, report all the points in current subtree

If bounding box overlaps query box, recurse left and right.

Range Query PseudoCode

```
def RangeQuery(Q, T):
  if T == NULL: return empty_set()
  if BB(T) doesn't overlap Query: return 0
  if Query subset of BB(T): return AllNodesUnder(T)

  set = empty_set()
  if T.data in Query: set.union({T.data})

  set.union(RangeQuery(Q, T.left))
  set.union(RangeQuery(Q, T.right))

  return set
```

VE281

Data Structures and Algorithms

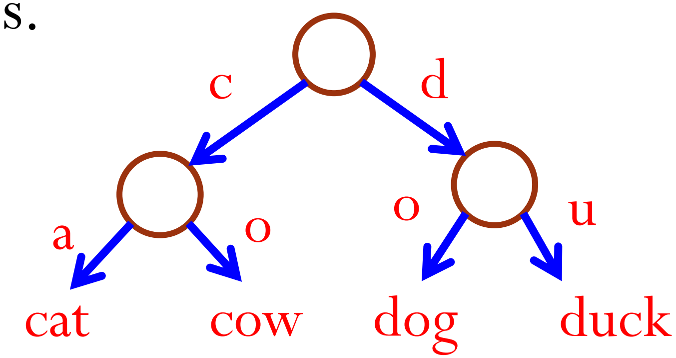
Tries

Learning Objectives:

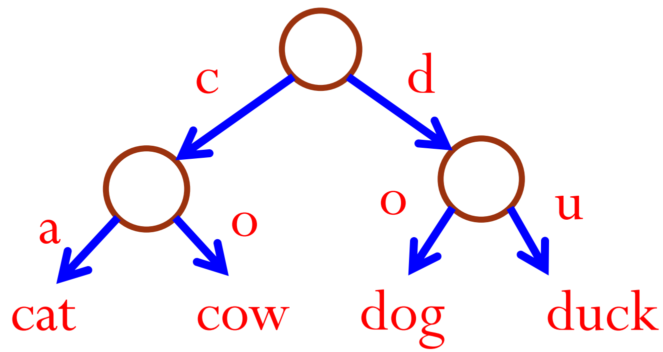
- Know what a trie is and understand its difference between binary search trees
- Know how to implement search, insertion, and removal for a trie

Trie

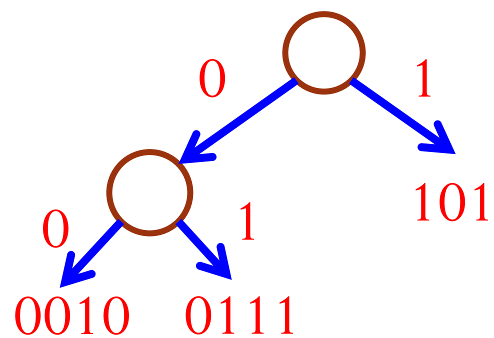
- The word "trie" comes from retrieval.
  - To distinguish with "tree", it is pronounced as "try".
- A trie is a tree that uses parts of the key, as opposed to the whole key, to perform search.
- Data records are only stored in **leaf** nodes. Internal nodes do not store records; they are "**branch**" points to direct the search process.



Trie

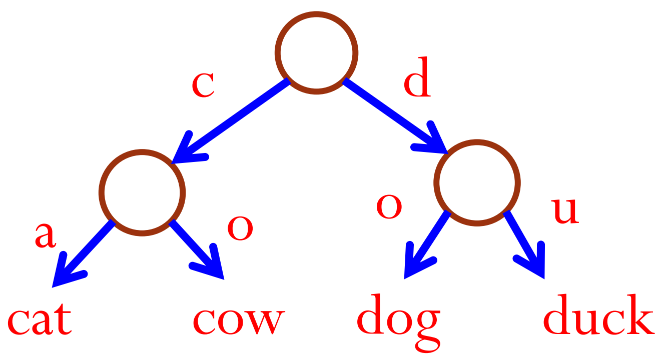


- Trie usually is used to store a set of strings from an alphabet.
  - The alphabet is in the general sense, not necessarily the English alphabet.
- For example, {0, 1} is an alphabet for binary codes {0010, 0111, 101}. We can store these three codes using a trie.



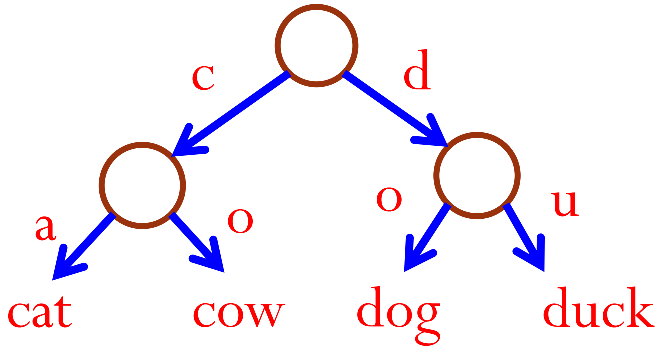
Trie

- Each edge of the trie is labeled with symbols from the alphabet.
- Labels of edges on the path from the root to any leaf in the trie forms a **prefix** of a string in that leaf.
  - Trie is also called **prefix-tree**.



Trie

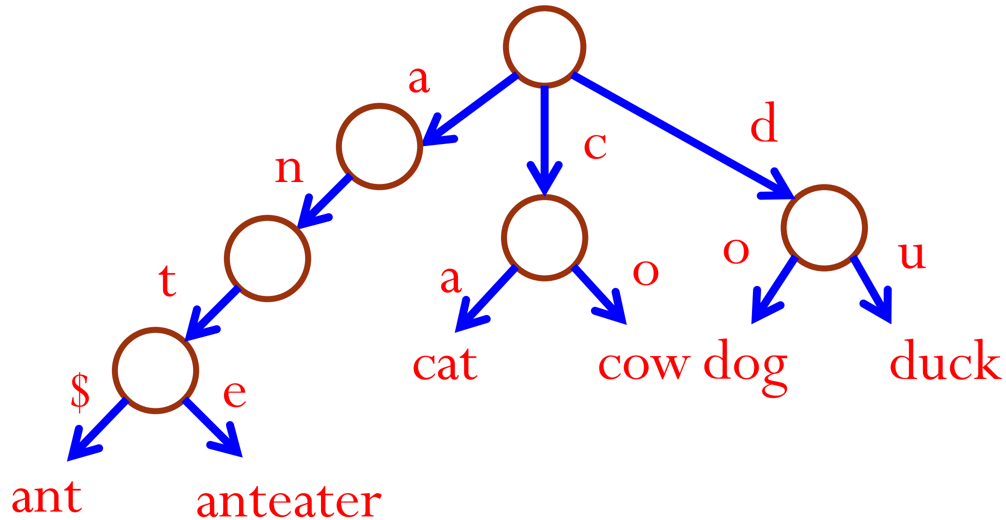
- The most significant symbol in a string determines the branch direction at the root.
- Each internal node is a "**branch**" point.
- As long as there is only one key in a branch, we do not need any further internal node below that branch; we can put the word directly as the leaf of that branch.



Trie

Implementation Issue

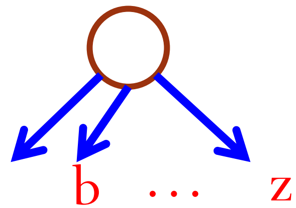
- Sometimes, a string in the set is exactly a **prefix** of another string.
  - For example, "ant" is a prefix of "anteater".
  - How can we make "ant" as a leaf in the trie?
- We add a symbol to the alphabet to indicate the end of a string. For example, use "$" to indicate the end.



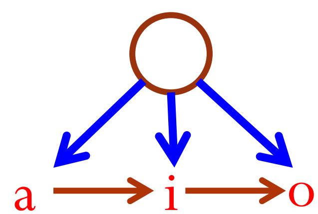
Trie

Implementation Issue

- We can keep an array of pointers in a node, which corresponds to **all** possible symbols in the alphabet.



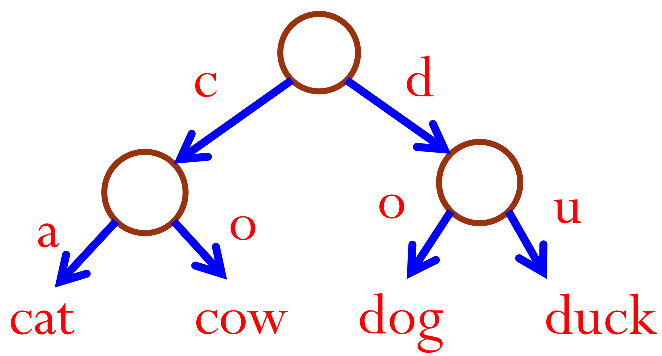
- However, most internal nodes have branches to only a small fraction of the possible symbols in the alphabet.
  - An alternate implementation is to store a linked list of pointers to the child nodes.



Trie

Search

- Follow the search path, starting from the root.
- When there is no branch, return false.
- When the search leads to a leaf, further compare with the key at the leaf.



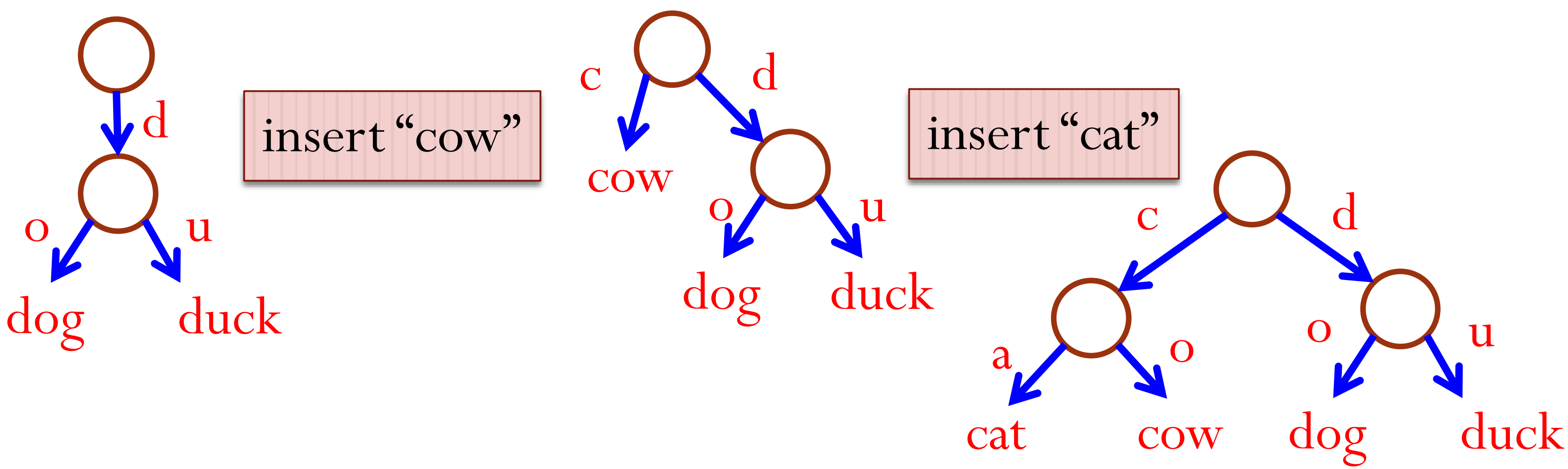
search “monkey”

search “cat”

Trie

Insertion

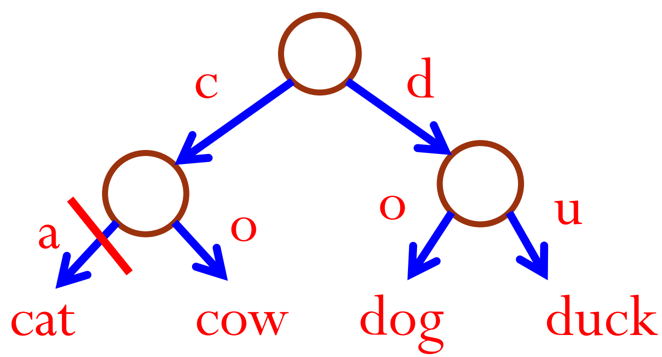
- Follow the search path, starting from the root.
- If a new branch is needed, add it.
- When the search leads to a leaf, a conflict occurs. We need to branch.
  - Use the next symbol in the key
  - The originally-unique word must be moved to lower level



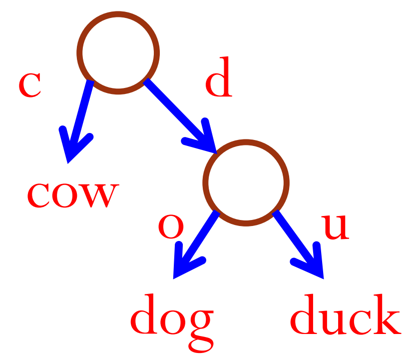
Trie

Removal

- The key to be removed is always at the leaf.
- After deleting the key, if the parent of that key now has only one child *C*, remove the parent node and move key *C* one level up.
  - If key *C* is the only child of its new parent, repeat the above procedure again.



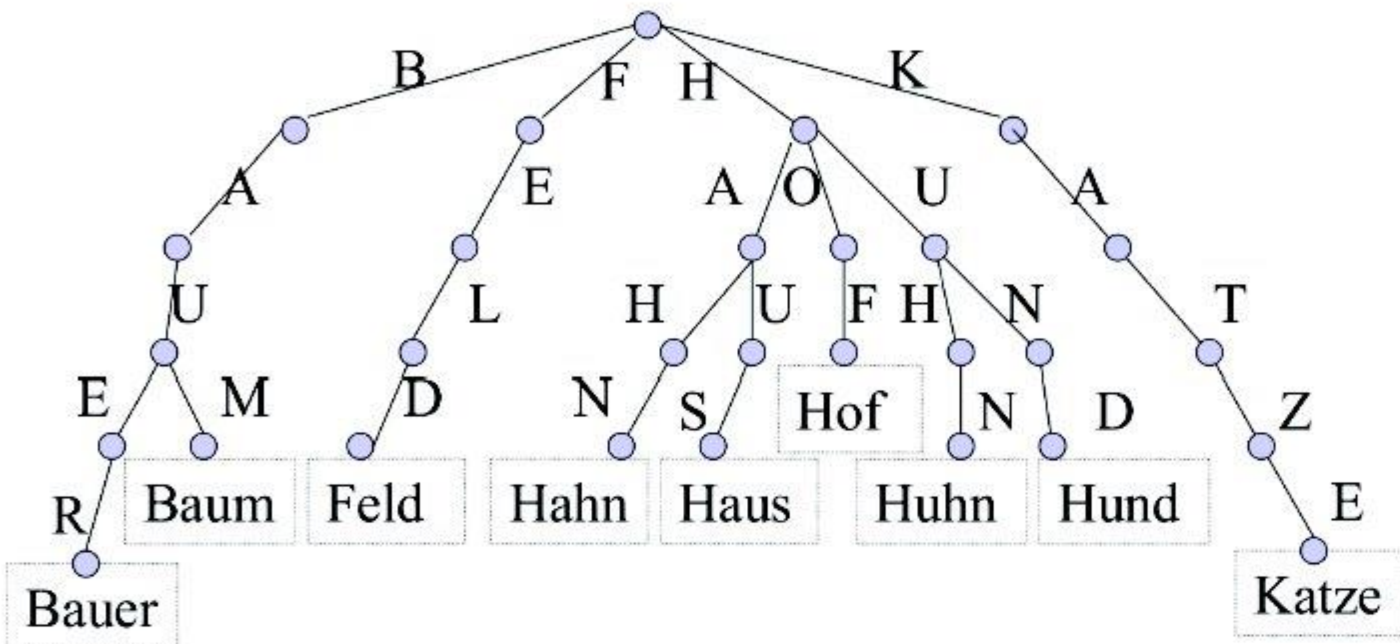
remove "cat"



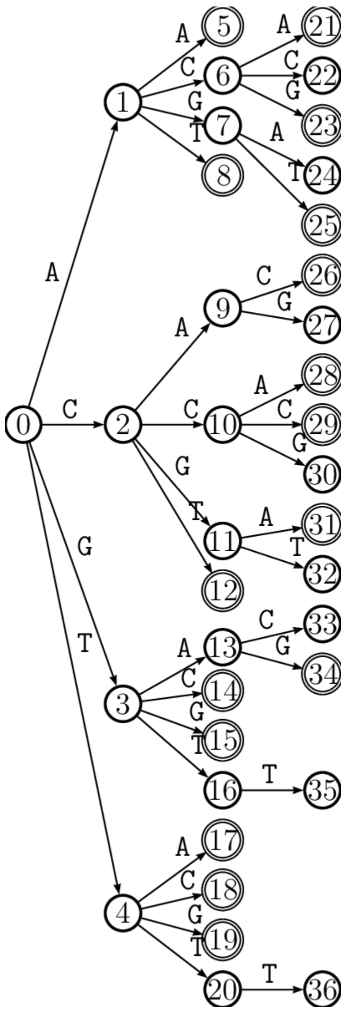
Time Complexity of Trie

- In the worst case, inserting or finding a key that consists of k symbols is $O(k)$.
  - This does not depend on the number of keys N.
  - Comparison: storing 32 integers in the range [0, 127] using a trie versus using a BST. What are heights in the **worst case**?
    - BST: 32; Trie: 7
- Sometimes we can access records even **faster**.
  - A key is stored at the depth which is enough to distinguish it with others.
  - For example, in the previous example, we can find the word "duck" with just "du".

Use Case – Spell Checking



Use Case – Human Genome



Example: Context Aware Seeds (My Research)

- Published in Algorithms in Molecular Biology 2020

Compute the Confidence Radii

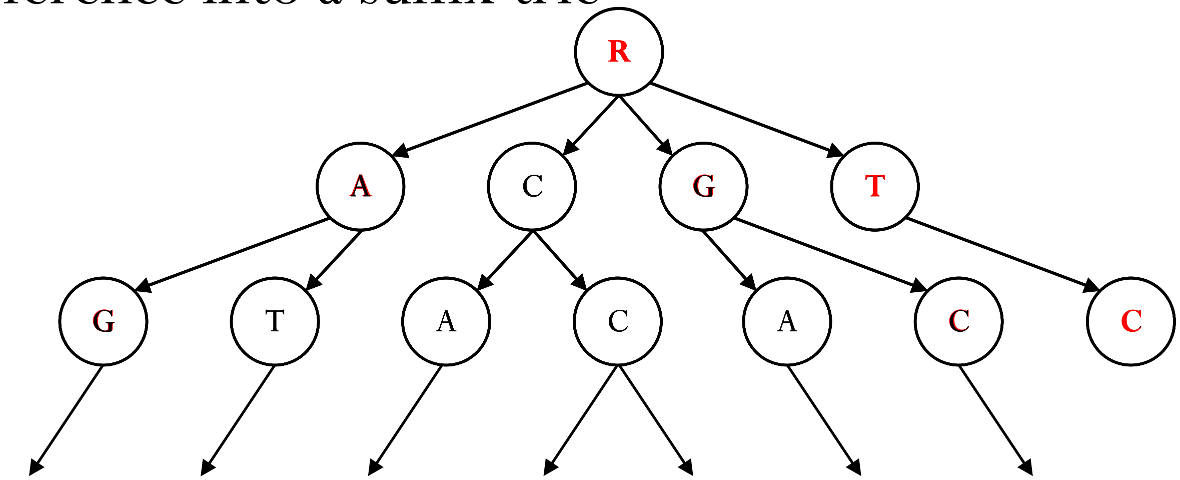
- **Confident radius**: **min** edit-distance (ED) to non-immediate neighbors

- Find all neighbors
- Find non-immediate neighbors
- Get the minimum ED

Suffix Trie

- Convert the reference into a suffix trie

Ref: GCCCAGATC



Dynamic Programming

- A recurrence relationship
- *a, b* can be neighbors only if *P(a), P(b)* are neighbors

a GCCCAGC
P(a) GCCCAG

a GCCCAGC
b GCCCACC

$c_{max} = 1$

a GCCCAGC
b GCCCACC
Neighbors

P(a) GCCCAG
P(b) GCCCAC
Neighbors

a GCCCAGC
b' GCCCACT
Not Neighbors

a GCCCAGC
b GCCCCCC
Not Neighbors

P(a) GCCCAG
P(b) GCCCCC
Not Neighbors

Find Neighbors

- Initialization: find all neighbors of the root node
- Iterate the suffix trie: top-to-bottom
  - Find neighbors for children nodes
- Invariant: all neighbors of current node is found

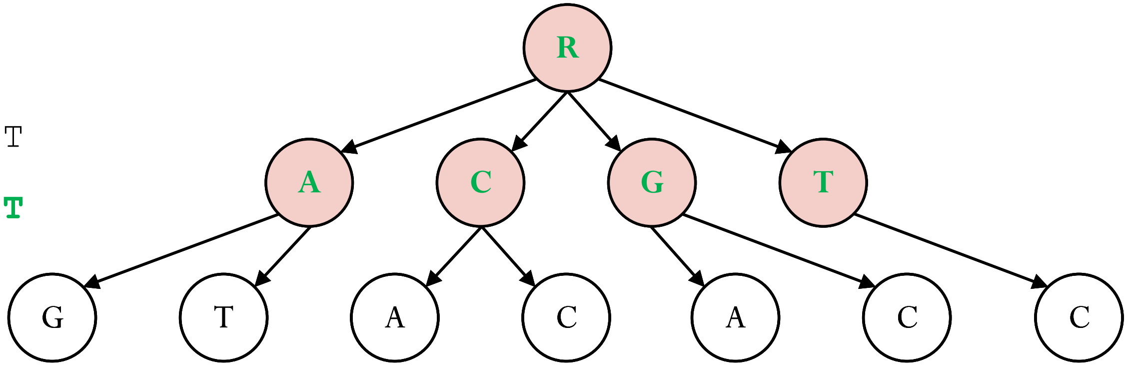
Initialize Root

$c_{max} = 1$

Ref: `GCCCAGATC`

Children of R : `R, A, C, G, T`

Neighbors of R : **`R, A, C, G, T`**



Iterate Trie—Build Neighbor List

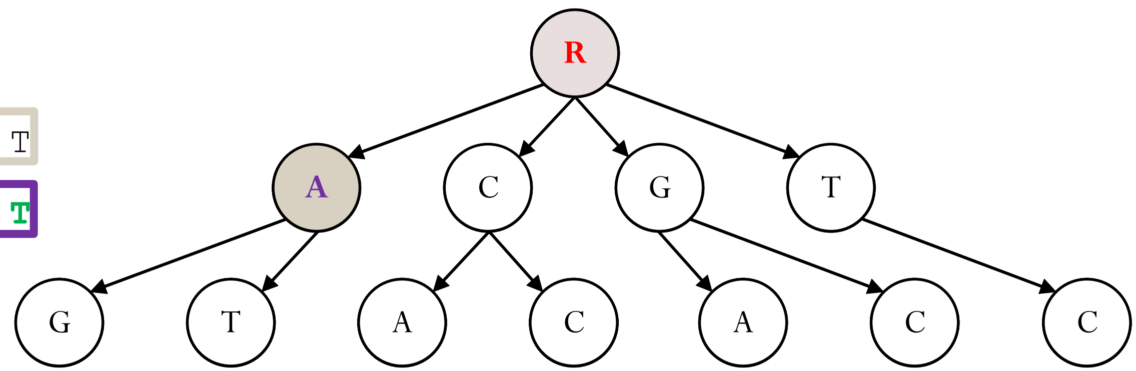
$c_{max} = 1$

Ref: `GCCCAGATC`

Children of R : R, A, C, G, T

Neighbors of R : **R, A, C, G, T**

Neighbors of A : Children of



x is a neighbor of **A** → $p(x)$ must be neighbor of $p($**A**$)$

$p(x)$ must be neighbor of **R**

Neighbors of **A** must be children of neighbors of **R**

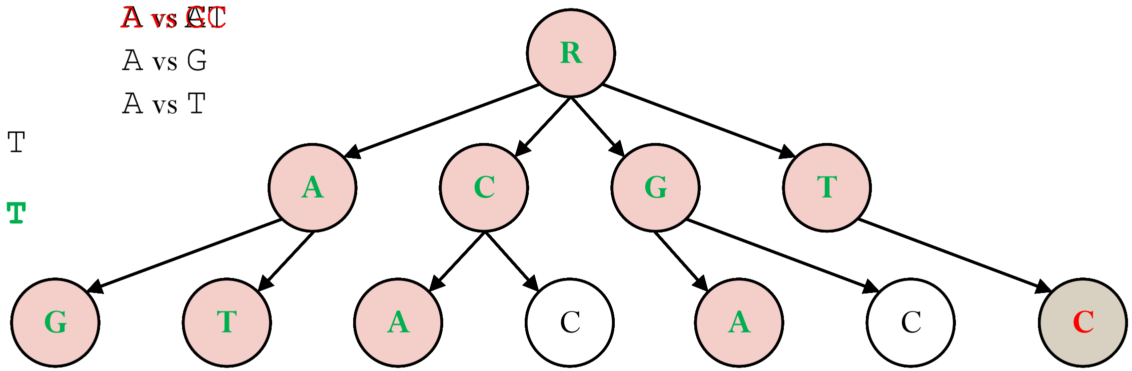
Neighbors of A

$c_{max} = 1$

A vs R
A vs A
A vs C
A vs G
A vs T

Children of R : A, C, G, T

Neighbors of R : $\boxed{R}$ A, C, G, T



Neighbors of A :

All neighbors of A is found!

Neighbors of A must be children of neighbors of R

Complexity: $O(\mathcal{M})$
$\mathcal{M}$: No. pairs of neighbors

Compute the **Confidence Radii**

- **Confident radius**: **min** edit-distance (ED) to non-immediate neighbors

- Find all neighbors
- **Find non-immediate neighbors**
  - **Remove immediate neighbors**
- Get the minimum ED

Does not increase complexity
Complexity: $O(\mathcal{M})$