CSE 100: TREE SEARCH AND HASH TABLES (1)

Goals for today

- Examine algorithms for tree search
- Explain the idea behind and advantages of hashing and hash tables
- Calculate collision probabilities in hash tables

New(ish)!

PA2: Implementing dictionaries and autocomplete!

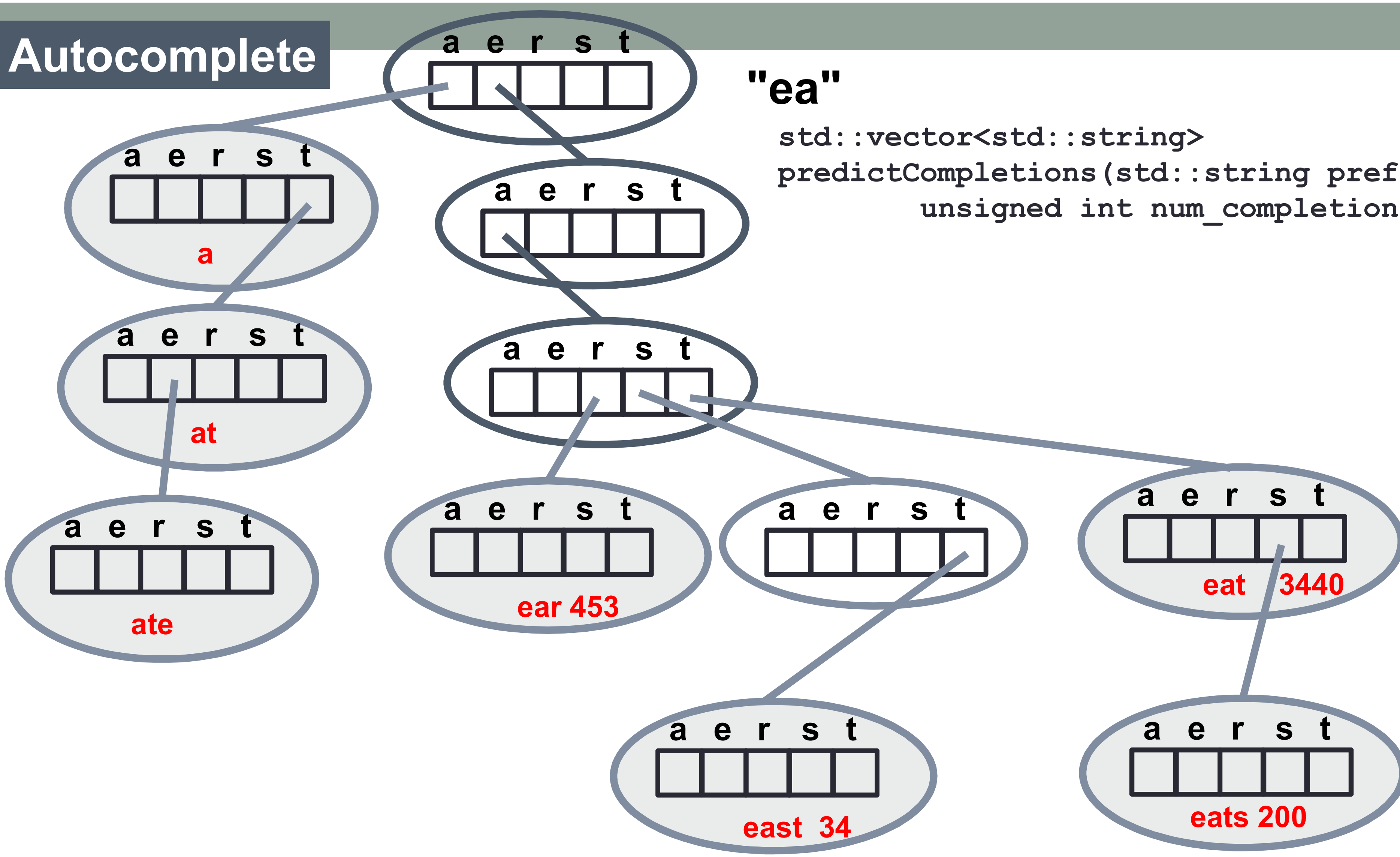
- Checkpoint: Implement string autocomplete
  - Autocomplete: Finds the top ‘n’ most frequently occurring words with a given prefix

“ap” “apple”
“ape”
“apology”
…

You will use Tries!!

- Final submission: Autocomplete, Document Generation
  - Document Generation:
    - Given previous text, produce similar new text
    - Markov Text Generation

Autocomplete



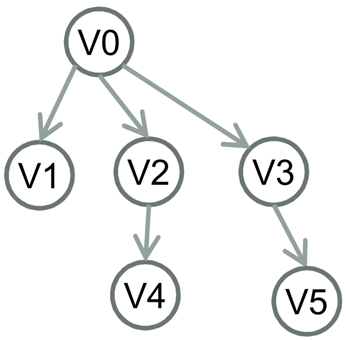
"ea"

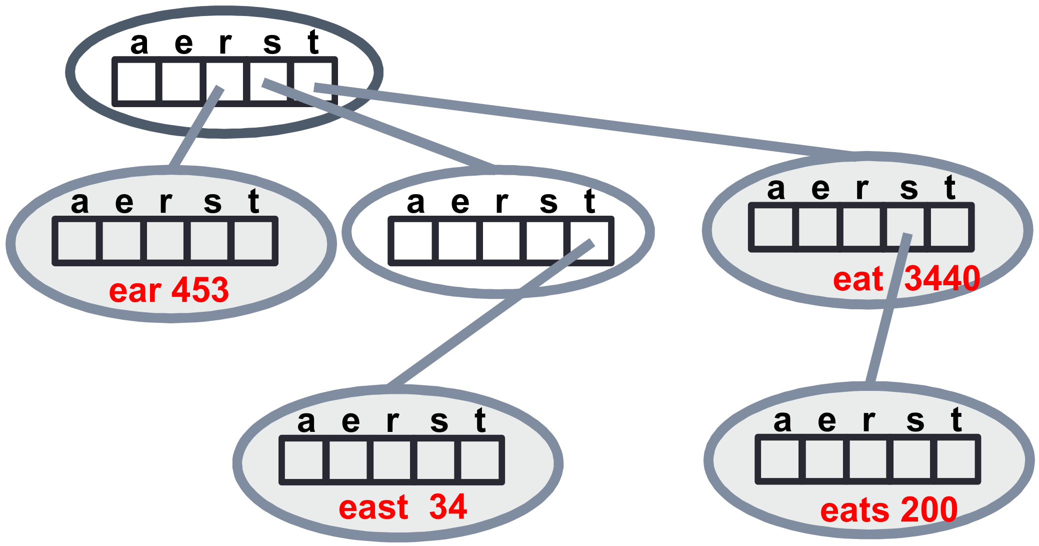
```
std::vector<std::string>
predictCompletions(std::string prefix,
         unsigned int num_completions);
```

Generic approach to graph (tree) search

Generic Goals:

- Find everything that can be explored
- Don't explore anything twice





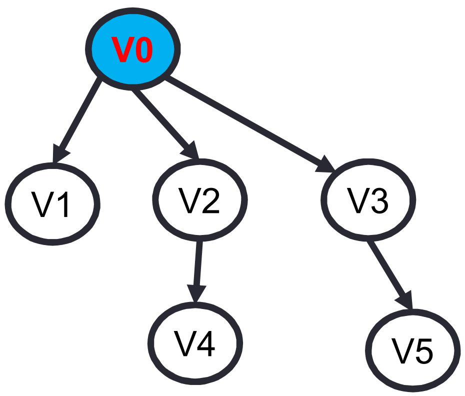
We will look at different graph (tree) search algorithms. Either can be used for PA2, at least at first

- Depth First Search
- Breadth First Search

Depth First Search for Tree Traversal

- Search as far down a single path as possible before backtracking

Write the order of nodes explored, starting at V0 and assuming smaller numbers are selected first

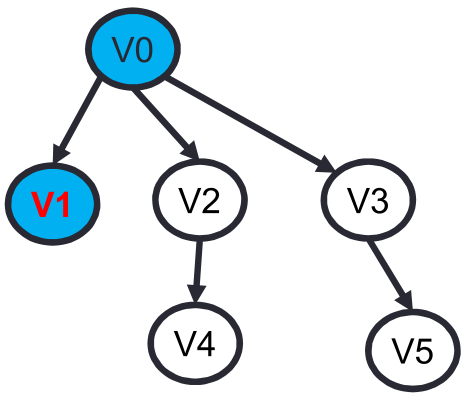


V0

Depth First Search for Tree Traversal

- Search as far down a single path as possible before backtracking

Write the order of nodes explored, starting at V0 and assuming smaller numbers are selected first

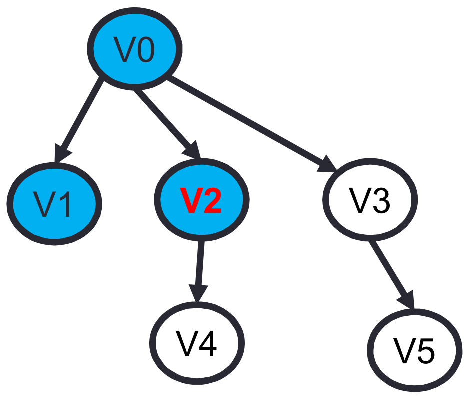


V0, V1

Depth First Search for Tree Traversal

- Search as far down a single path as possible before backtracking

Write the order of nodes explored, starting at V0 and assuming smaller numbers are selected first

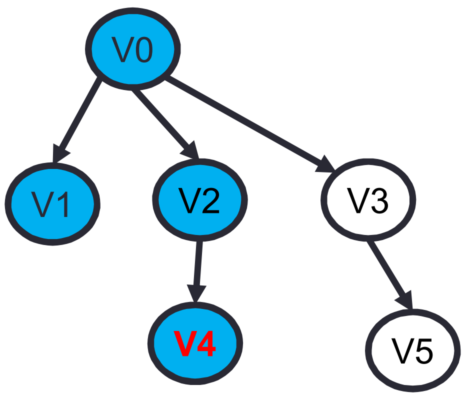


V0, V1, V2

Depth First Search for Tree Traversal

- Search as far down a single path as possible before backtracking

Write the order of nodes explored, starting at V0 and assuming smaller numbers are selected first

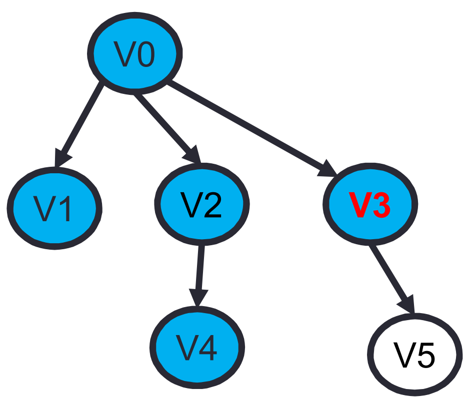


V0, V1, V2, V4

Depth First Search for Tree Traversal

- Search as far down a single path as possible before backtracking

Write the order of nodes explored, starting at V0 and assuming smaller numbers are selected first

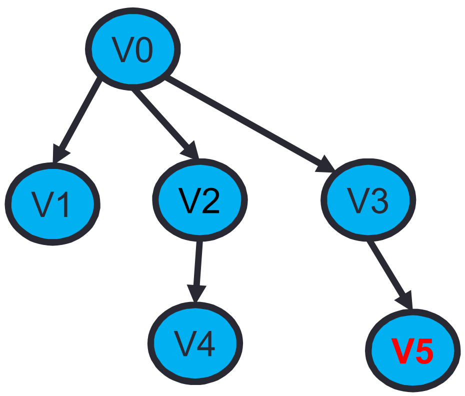


V0, V1, V2, V4, V3

Depth First Search for Tree Traversal

- Search as far down a single path as possible before backtracking

Write the order of nodes explored, starting at V0 and assuming smaller numbers are selected first



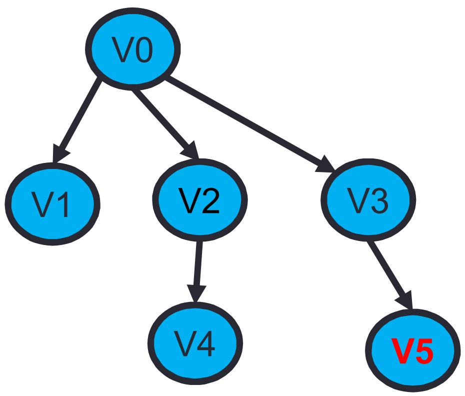
V0, V1, V2, V4, V3, V5

Depth First Search for Tree Traversal

- Search as far down a single path as possible before backtracking

Write the order of nodes explored, starting at V0 and assuming smaller numbers are selected first

How to keep track of where to search next?

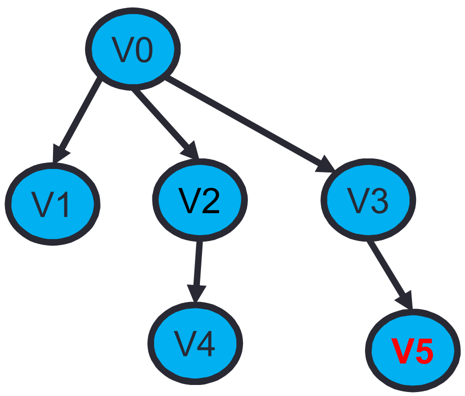


V0, V1, V2, V4, V3, V5

Depth First Search for Tree Traversal

- Search as far down a single path as possible before backtracking

Write the order of nodes explored, starting at V0 and assuming smaller numbers are selected first



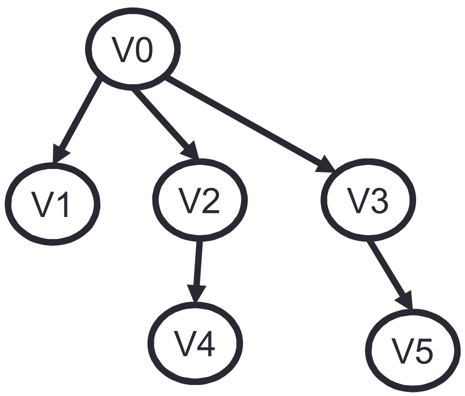
How to keep track of where to search next?
A Stack! (LIFO)

V0, V1, V2, V4, V3, V5

Depth First Search for Tree Traversal

- Search as far down a single path as possible before backtracking

Write the order of nodes explored, starting at V0 and assuming smaller numbers are selected first

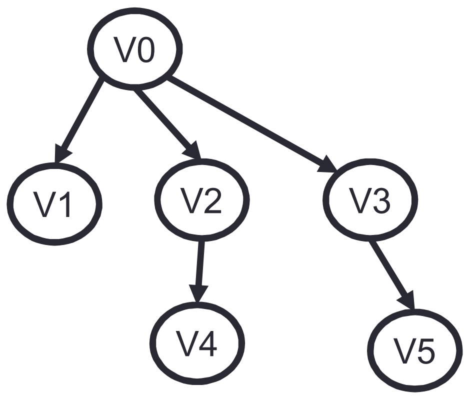



```
DFS(Start):
  Initialize stack
  Push Start onto the stack
  while stack is not empty:
    pop node curr from top of stack
    visit curr
    for each of curr's children, n
      push n onto the stack
// When we get here then we're done exploring
```

Recursive Depth First Search for Tree Traversal

- Search as far down a single path as possible before backtracking

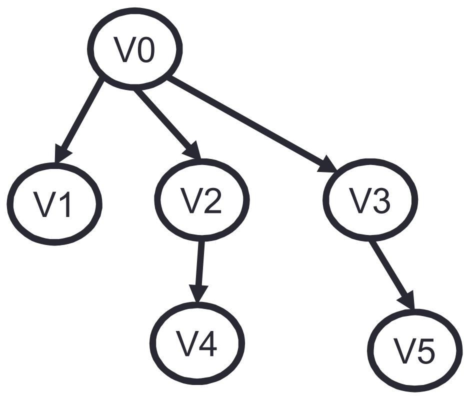
Write the order of nodes explored, starting at V0 and assuming smaller numbers are selected first




```
DFS(Start S):
  for each of S's neighbors, n:
     "visit" n
     DFS(n)
```

What happens when we switch from a stack to a queue?

Write the order of nodes explored, starting at V0 and assuming smaller numbers are selected first




```
Mystery(Start):
  Initialize queue
  Push Start onto the queue
  while queue is not empty:
    pop node curr from front of queue
    visit curr
    for each of curr's children, n
      push n onto the queue
// When we get here then we're done exploring
```

PA2: Things to work out

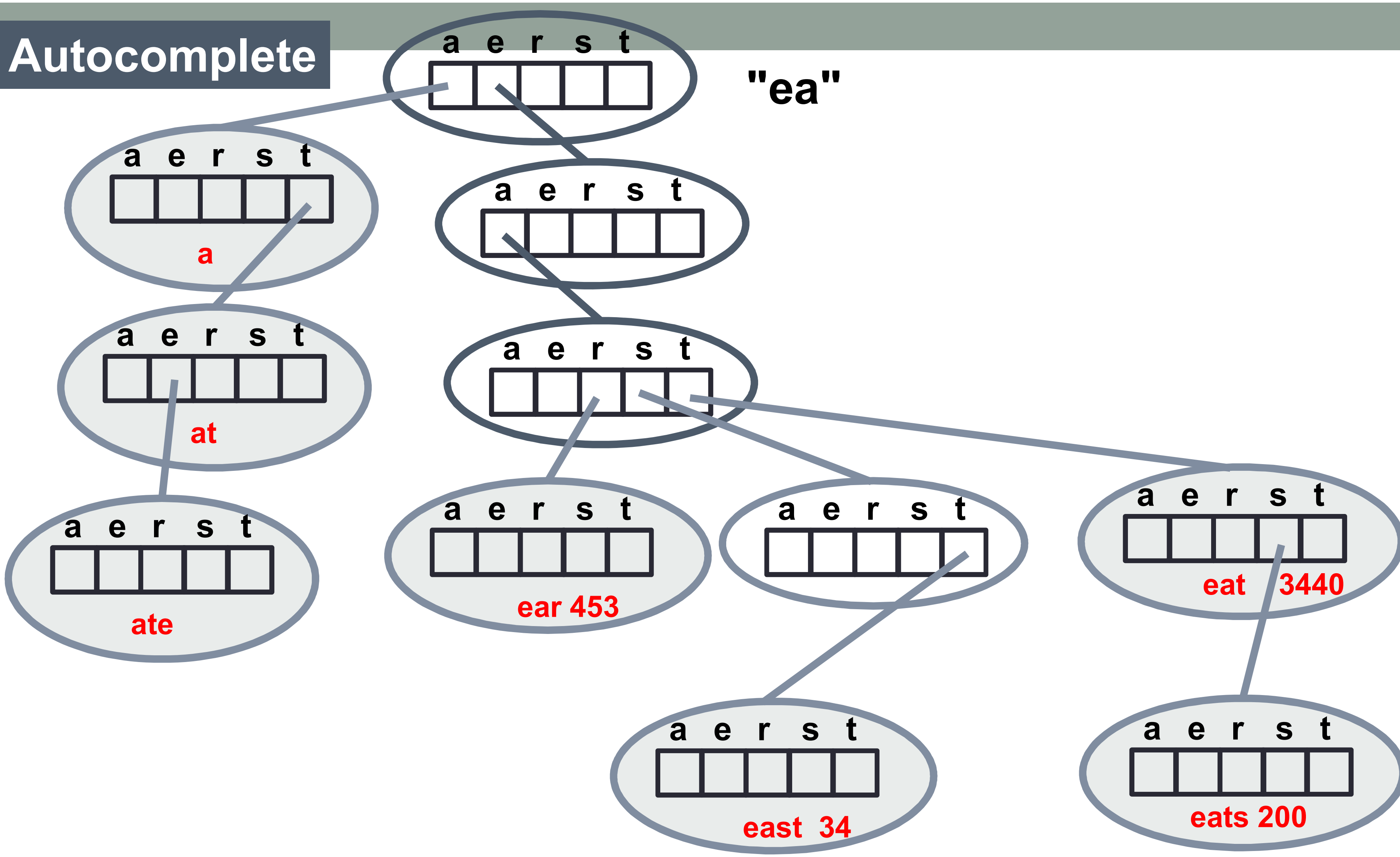
- Which search algorithm should be used for predict_completions on PA2?

A. DFS

B. BFS

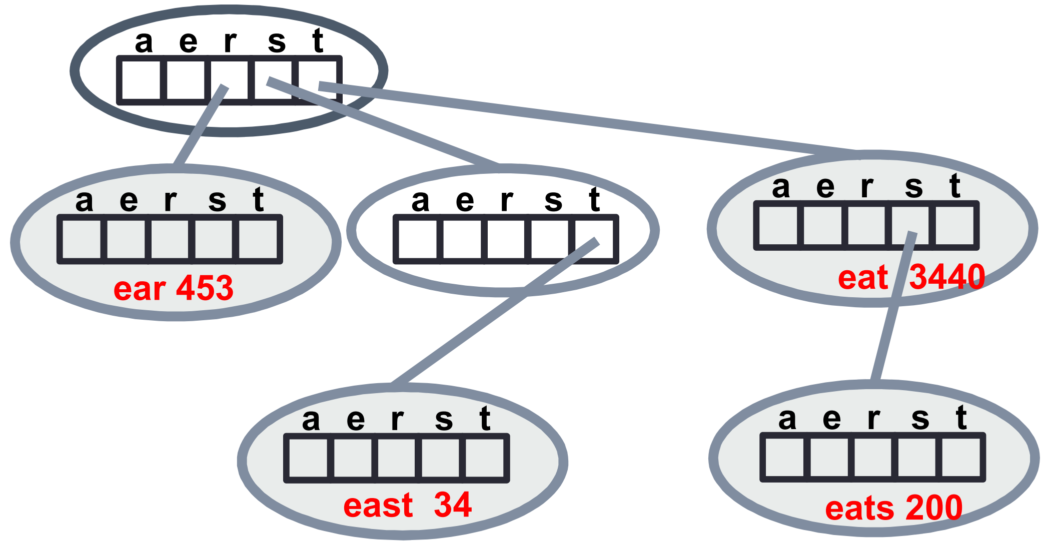
C. Either one, it doesn’t matter

Autocomplete



Search for PA2: things to work out

DFS(S, G, other_args):
for each of S's children, n:
figure out what to do with n
DFS(n, G, other_args)



vector<string> predictCompletions(string prefix) ;

Tree Search: Thought questions

- Which search algorithm will take longer for predict-completions (return the N most frequent completions)?

A. DFS
B. BFS
C. They will take the same

- If you ignore frequencies, which search algorithm will find words in increasing order of length in a MWT?

A. DFS
B. BFS
C. both
D. neither

- Which search algorithm will find words in increasing order of frequency in a MWT?

A. DFS
B. BFS
C. both
D. neither

Finding data fast

Imagine that you want to store integers between 0 and 1,000,000. You want to be able to find out whether an element is present in your set. You know you can do this with a BST with average running time of O(logN). But you decide to use an array with 1,000,000 Boolean values to store your data to try to make this faster. An entry will be true if the item is in your structure, and false otherwise.

What is the (Big-O) running time to "insert" an item into this proposed structure?

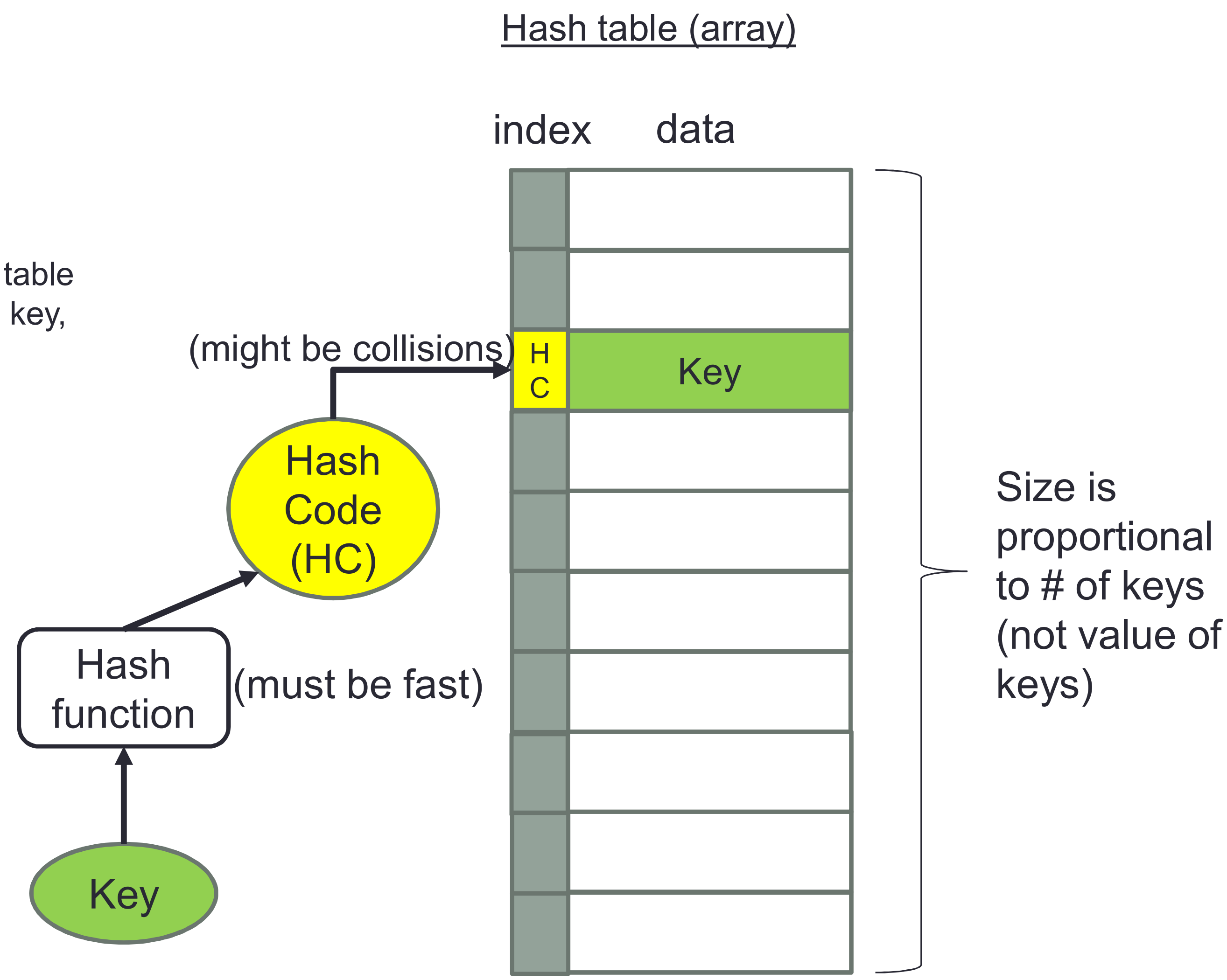
A. O(N)

B. O(logN)

C. O(1)

Hashing

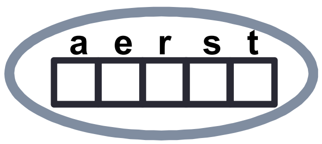
- Let's modify our array-based look up table
- Need a hash-function h(x): takes in a key, returns an index in the array
- gold standard: random hash function



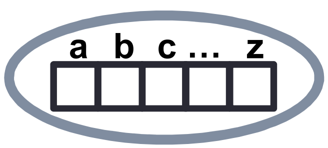
"Hashing" and MWTs

- The idea of hashing was at the heart of the MWT data structure we looked at. We mapped chars to ints to get their position in the array.

The hash function was:
H('a') = 0
H('e') = 1
H('r') = 2
H('s') = 3
H('t') = 4



What is the hash function you would use for your PA if you implement a MWT with a vector?
(ignore spaces for now)



H(c) = ________________________

This hash function has no collisions!