CNN
for
CV

AI for CV Group
2019

Week 5.
Preparation for Coding Test

Outline:

I. Things you have to know

A. Data Structure

B. Engineering Topics (For engineering-oriented guys)

II. Questions you have to remember

C. Classic questions usually asked in interviews

D. Advanced problems

III. Specific topics

E. Bit Operation

F. Dynamic Programming

G. Trie

I. Things You Have To Know

I. Things you have to know

A. Data structure:

C++ (STL)	Python (from collections)	Fundamental Struct.
unordered_map	dict	hash table/set
unordered_set	set	
map		red-black tree
set		
vector	list (not exactly list, but u can use list do all manipulations mentioned left.)	vector / continual memory space
queue		queue
deque		double ended queue
list		double ended list
stack		stack
priority_queue	heapq	heap

I. Things you have to know

A. Data structure:

Fundamental Data Structure	Time Complexity		
hash table	find/delete/insert: O(1)		
red black tree	find/delete/insert: O(logn)		
vector	find / delete: O(n)	insert: O(1)	
queue	find / delete: O(n)	insert: O(1)	
deque	find / delete: O(n)	insert: O(1)	
list	delete: O(n)/O(1)	insert: O(1)	find: O(n)
stack	find / delete: O(n)	insert: O(1)	
heap	insert / delete: O(logn)	find: O(n)	build: O(n)

I. Things you have to know

B. Engineering Topics (for engineering fields):

<table>
<tr><th>C++</th><th>Python</th></tr>
<tr><td>C++ 11/14/17 new features</td><td>if __name__ == “__main__”</td></tr>
<tr><td>smart pointer</td><td>list & tuple</td></tr>
<tr><td>auto</td><td>class</td></tr>
<tr><td>(pure) virtual function/table</td><td>use * & ** when passing param.</td></tr>
<tr><td>template</td><td>comprehension expression</td></tr>
<tr><td>const/static</td><td>usage of @</td></tr>
<tr><td colspan="2">segment</td></tr>
<tr><td colspan="2">try</td></tr>
<tr><td colspan="2">lambda function</td></tr>
</table>

II. Questions you have to remember

II. Questions you have to remember

C. Classic questions

C1. Quick Sort / Top k problem

Quick Sort (O(nlogn))

Kth largest element in an array

- sort (O(nlogn))
- heap (O(nlogk))
- semi-quick sort (O(n))

II. Questions you have to remember

C. Classic questions

C2. Binary Search

Basic Template

First-Last Element

Closest K Elements

II. Questions you have to remember

C. Classic questions

C3. Binary Tree Traversal

Pre/post/in-order traversal

Level order / zigzag order / right-view

Balanced / Binary search tree validation

II. Questions you have to remember

C. Classic questions

C4. Linked list

Reverse linked list (Iterative & recursive version)

Merge 2 linked lists

II. Questions you have to remember

C. Classic questions

C5. Sqrt / Pow

Binary Search / Newton

Recursive / Bit Operation (usually for matrix)

II. Questions you have to remember

C. Classic questions

C6. Points minimum distance in a matrix

Binary maze

Normal maze

II. Questions you have to remember

C. Classic questions

C7. Number Island I&II

DFS / BFS / Union Find

II. Questions you have to remember

C. Classic questions

C8. Other Regular Problems

2 Sum (O(nlogn) -> O(n))

Stock I - IV & House Robber

Permutation I & II

Combination Sum I - IV

Linked List Cycle I &II

Longest substring with same 0 & 1

Longest same subsequence

Calculator I-III

Palindrome series

……

II. Questions you have to remember

C. Classic questions

C9. CV specific problems!!

Range Sum (1d – immutable & mutable (indexed tree))
Range Sum (2d – immutable & mutable (indexed tree))

Sliding Window Maximum (1d [deque] & 2d [deque])

Convolution In N-Dimension (Usually 2 or 3)
[Convolution / Maximum / Kth Largest / Average]

II. Questions you have to remember

D. Advanced questions

D1. Classic Hard Problem

DP: Unique Path I&II / Wildcard & Regular Expression /
Word Break I&II

DFS: Course Schedule I&II / Android Unlock / Validate Parenthesis
Word Pattern I&II
Word Search I&II

BFS: Word Ladder I&II / Life Game

Longest series problems (in matrix, array, list, tree…)

II. Questions you have to remember

D. Classic questions

D2. Classic Notable Problems

N-Queens	[dfs]
Game 24	[dfs]
Minesweeper	[bfs]
Snake	[design]
Hannoi	[dp]
Sudoku I&II	[dfs]
KMP	

III. Specific Tops

III. Specific Topics

E. Bit Operations

Bit: High/low voltage

Data type: length of bits

Pros: High speed &
beautiful codes

Basic rules:

	1 □ 1	0 □ 0	1 □ 0
&（与）	1	0	0
|（或）	1	0	1
^（异或）	0	0	1
!（非）	!1 = 0 / !0 = 1		

III. Specific Topics

E. Bit Operations – e.g.

```
bool isOdd(int num) {
    // write your code here
    if (num % 2 == 1) {
        return true;
    }
    else {
        return false;
    }
}
```

III. Specific Topics

E. Bit Operations – e.g

```
bool isOdd(int num) {
     // write your code here
     return num % 2 == 1;
}
```

[After considered the *if* statement]

III. Specific Topics

E. Bit Operations – e.g

```
bool isOdd(int num) {
    // write your code here
    return num % 2 != 0;
}
```

[After considered num could be less than 0]

III. Specific Topics

E. Bit Operations – e.g

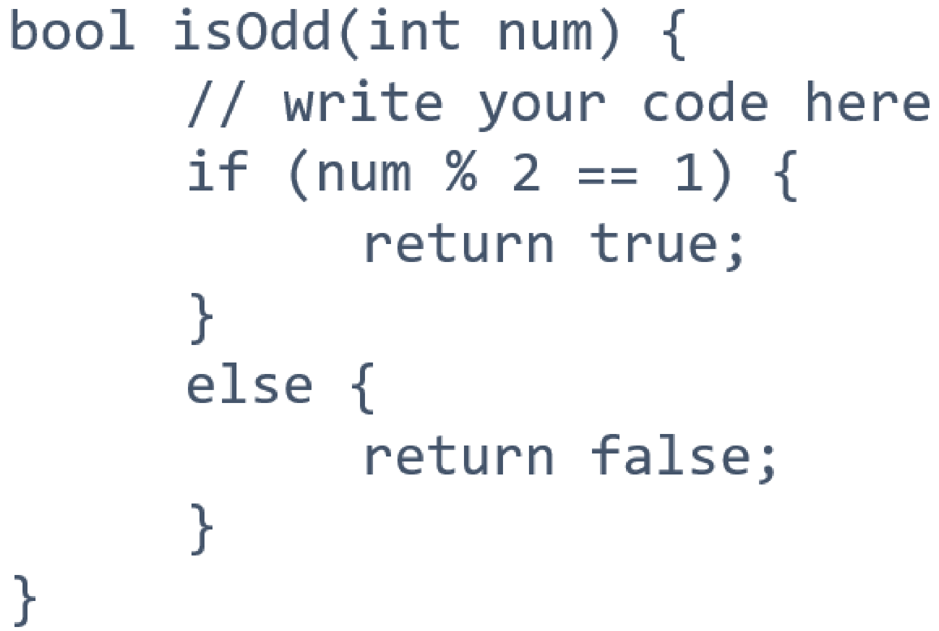
```
bool isOdd(int num) {
     // write your code here
     return num & 1;
}
```

[After considered last bit of an odd num is 1]

III. Specific Topics

E. Bit Operations – e.g.

```
bool isOdd(int num) {
        // write your code here
        return num & 1;
}
```



```
bool isOdd(int num) {
        // write your code here
        if (num % 2 == 1) {
                return true;
        }
        else {
                return false;
        }
}
```

III. Specific Topics

E. Bit Operations – Important Questions

Single number series: [Lintcode: 82, 83, 84, 824]

Hamming distance: [Lintcode: 835, 1217]

III. Specific Topics

F. Dynamic Programming

F1. Longest Common Subsequence: [Lintcode 77]

```
Input: str1: ABCFE   str2: ACEMNXOP
```

III. Specific Topics

F. Dynamic Programming

F1. Longest Common Subsequence: [Lintcode 77]

```
Input: str1: ABCFE   str2: ACEMNXOP
Output I: 3
Output II: ACE
```

III. Specific Topics

F. Dynamic Programming

F1. Longest Common Subsequence: [Lintcode 77]

```
Input: str1: ABCFE   str2: ACEMNXOP

dp[i, j]:
在str1位置为i, str2位置为j时, 最长的公共子序列长度

如何填充dp[i,j]:
若在[i,j]位置str1[i]=str2[j],
    则dp[i,j]为之前的最长长度(dp[i-1][j-1])+1
若str1[i]≠str2[j],
    则dp[i,j]为之前的最长长度(max(dp[i-1][j],dp[i][j-1]))
```

III. Specific Topics

F. Dynamic Programming

F1. Longest Common Subsequence: [Lintcode 77]

```
Input: str1: ABCFE   str2: ACEMNXOP
```

$$dp[i, j] = \begin{cases} 0, & i=0 \;||\; j=0 \\ dp[i-1,j-1] + 1, & i,j>0,\ xi=yi \\ \max\{dp[i,j-1],\ dp[i-1,j]\}, & i,j>0,\ xi!=yj \end{cases}$$

III. Specific Topics

F. Dynamic Programming

F1. Longest Common Subsequence: [Lintcode 77]

Most critical things:

I. Get the meaning of your dp matrix

II. Get the rule to fill in your dp matrix

III. Specific Topics

F. Dynamic Programming

F2. 硬币翻转（红绿灯问题）：

假设有一组硬币，材质非均匀，因而它们各自抛出后正面朝上的概率不同，假设为p1, p2, …, pn。问：当依次抛掷共n次后，出现k次正面朝上的概率是多少。

```
float getProb(vector<float> p, int k) {
    // write ur code here
}
```

III. Specific Topics

F. Dynamic Programming

F2. 硬币翻转（红绿灯问题）：

假设有一组硬币，材质非均匀，因而它们各自抛出后正面朝上的概率不同，假设为p1，p2，…，pn。问：当依次抛掷共n次后，出现k次正面朝上的概率是多少。

dp[i,j]:

$$dp[i,j]=\begin{cases} ? \end{cases}$$

III. Specific Topics

F. Dynamic Programming

F2. 硬币翻转（红绿灯问题）：

假设有一组硬币，材质非均匀，因而它们各自抛出后正面朝上的概率不同，假设为p1，p2，…，pn。问：当依次抛掷共n次后，出现k次正面朝上的概率是多少。

dp[i,j]：抛掷i次后，出现j次正面的概率

$$dp[i,j]=\begin{cases} 0, & i=0 \\ dp[i-1][j-1]*p[i]+ & \text{ith是正面} \\ dp[i][j]*(1-p[i]) & \text{ith是背面} \end{cases}$$

III. Specific Topics

F. Dynamic Programming

Similar questions:

Leetcode: 931+get the path

Lintcode: 192+what’s ‘?’ and ‘*’ stands for?,
154+what’s ‘.’ and ‘*’ stands for?

III. Specific Topics

G. Trie

Lintcode: 132 (DFS + Trie)

442 (Trie)

Target: Speed up the finding process, especially for words sharing the same prefix

III. Specific Topics

G. Trie (a.k.a 前缀树/prefix tree)

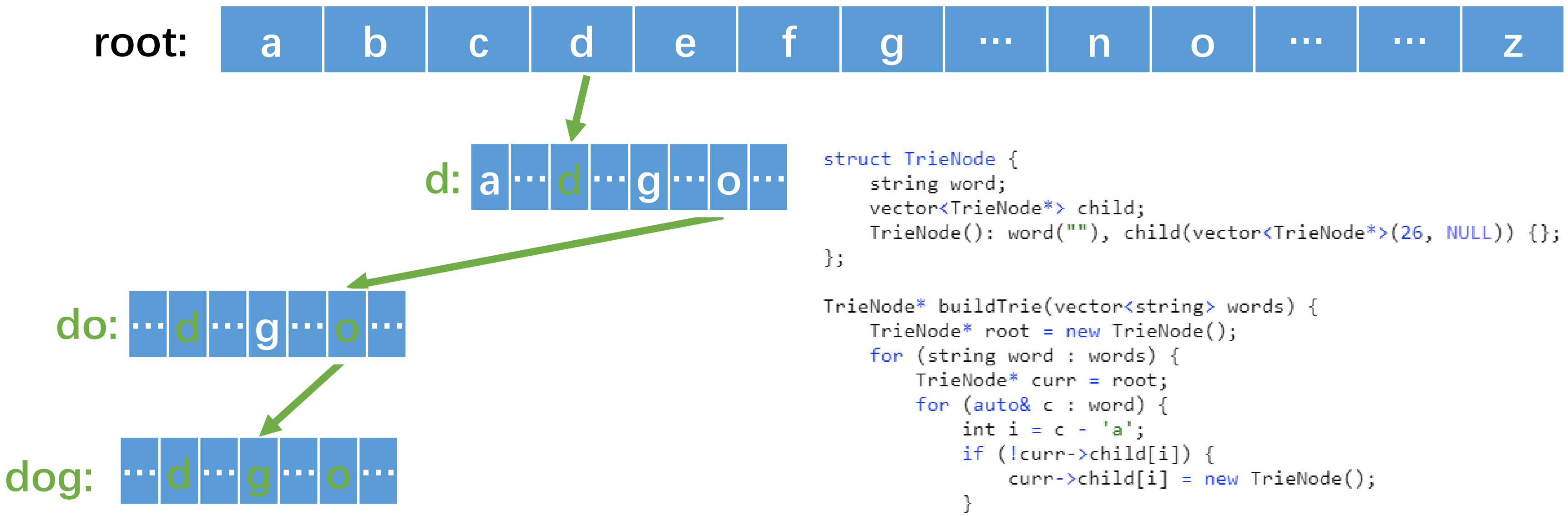
G1: 132

dog

dad

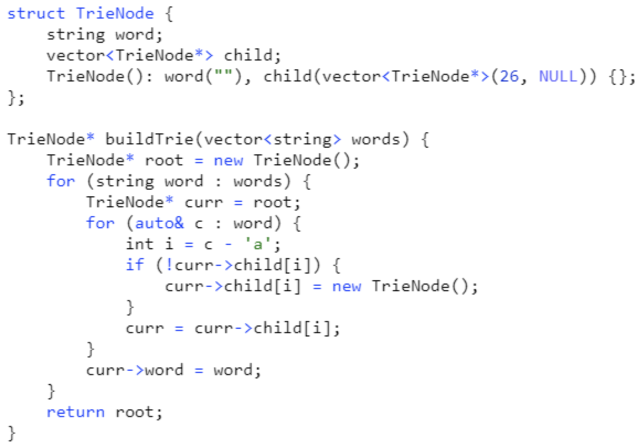
dgdg

can



```
struct TrieNode {
    string word;
    vector<TrieNode*> child;
    TrieNode(): word(""), child(vector<TrieNode*>(26, NULL)) {};
};

TrieNode* buildTrie(vector<string> words) {
    TrieNode* root = new TrieNode();
    for (string word : words) {
        TrieNode* curr = root;
        for (auto& c : word) {
            int i = c - 'a';
            if (!curr->child[i]) {
                curr->child[i] = new TrieNode();
            }
            curr = curr->child[i];
        }
        curr->word = word;
    }
    return root;
}
```



```
struct TrieNode {
    string word;
    vector<TrieNode*> child;
    TrieNode(): word(""), child(vector<TrieNode*>(26, NULL)) {};
};

TrieNode* buildTrie(vector<string> words) {
    TrieNode* root = new TrieNode();
    for (string word : words) {
        TrieNode* curr = root;
        for (auto& c : word) {
            int i = c - 'a';
            if (!curr->child[i]) {
                curr->child[i] = new TrieNode();
            }
            curr = curr->child[i];
        }
        curr->word = word;
    }
    return root;
}
```

III. Specific Topics

G. Trie (a.k.a 前缀树/prefix tree)

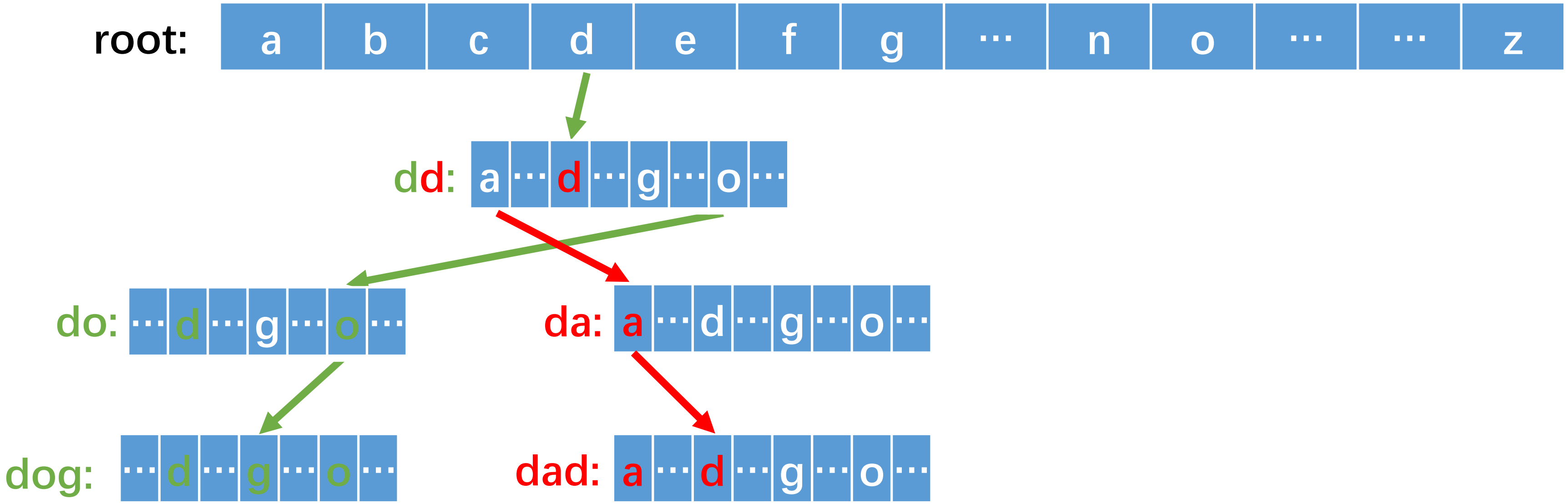
G1: 132

dog

dad

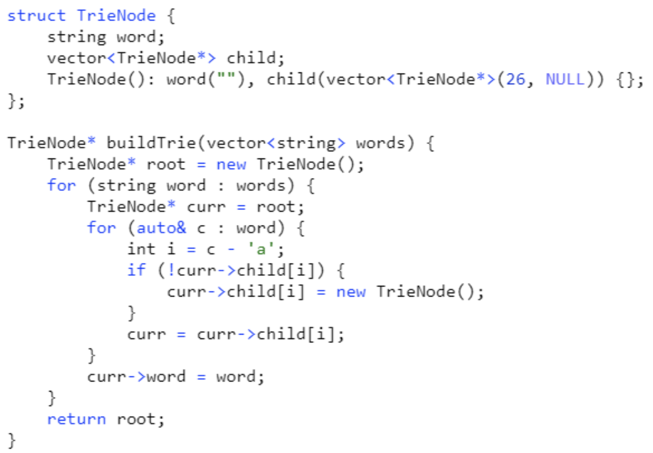
dgdg

can



III. Specific Topics

G. Trie (a.k.a 前缀树/prefix tree)



```
struct TrieNode {
    string word;
    vector<TrieNode*> child;
    TrieNode(): word(""), child(vector<TrieNode*>(26, NULL)) {};
};

TrieNode* buildTrie(vector<string> words) {
    TrieNode* root = new TrieNode();
    for (string word : words) {
        TrieNode* curr = root;
        for (auto& c : word) {
            int i = c - 'a';
            if (!curr->child[i]) {
                curr->child[i] = new TrieNode();
            }
            curr = curr->child[i];
        }
        curr->word = word;
    }
    return root;
}
```

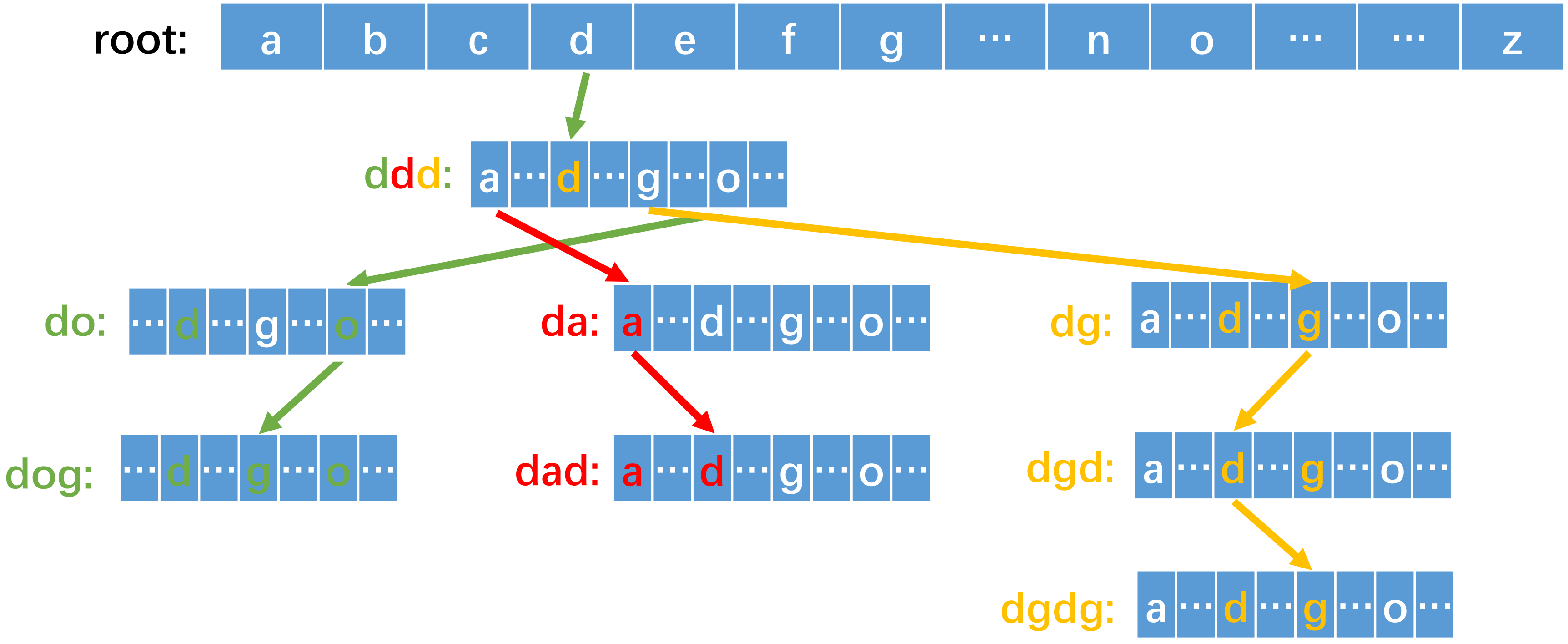
G1: 132

dog

dad

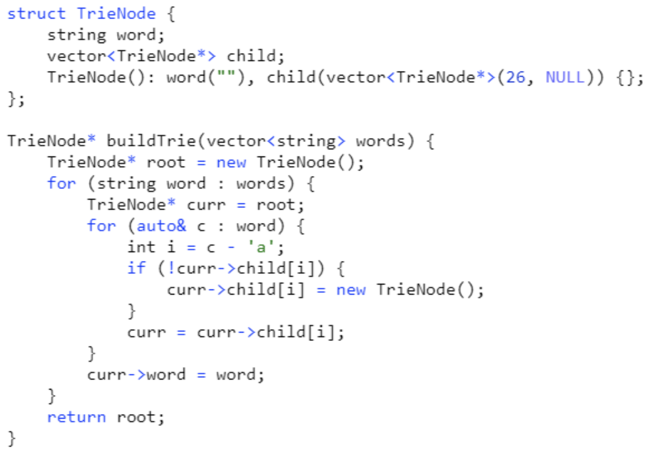
dgdg

can



III. Specific Topics

G. Trie (a.k.a 前缀树/prefix tree)



```
struct TrieNode {
    string word;
    vector<TrieNode*> child;
    TrieNode(): word(""), child(vector<TrieNode*>(26, NULL)) {};
};

TrieNode* buildTrie(vector<string> words) {
    TrieNode* root = new TrieNode();
    for (string word : words) {
        TrieNode* curr = root;
        for (auto& c : word) {
            int i = c - 'a';
            if (!curr->child[i]) {
                curr->child[i] = new TrieNode();
            }
            curr = curr->child[i];
        }
        curr->word = word;
    }
    return root;
}
```

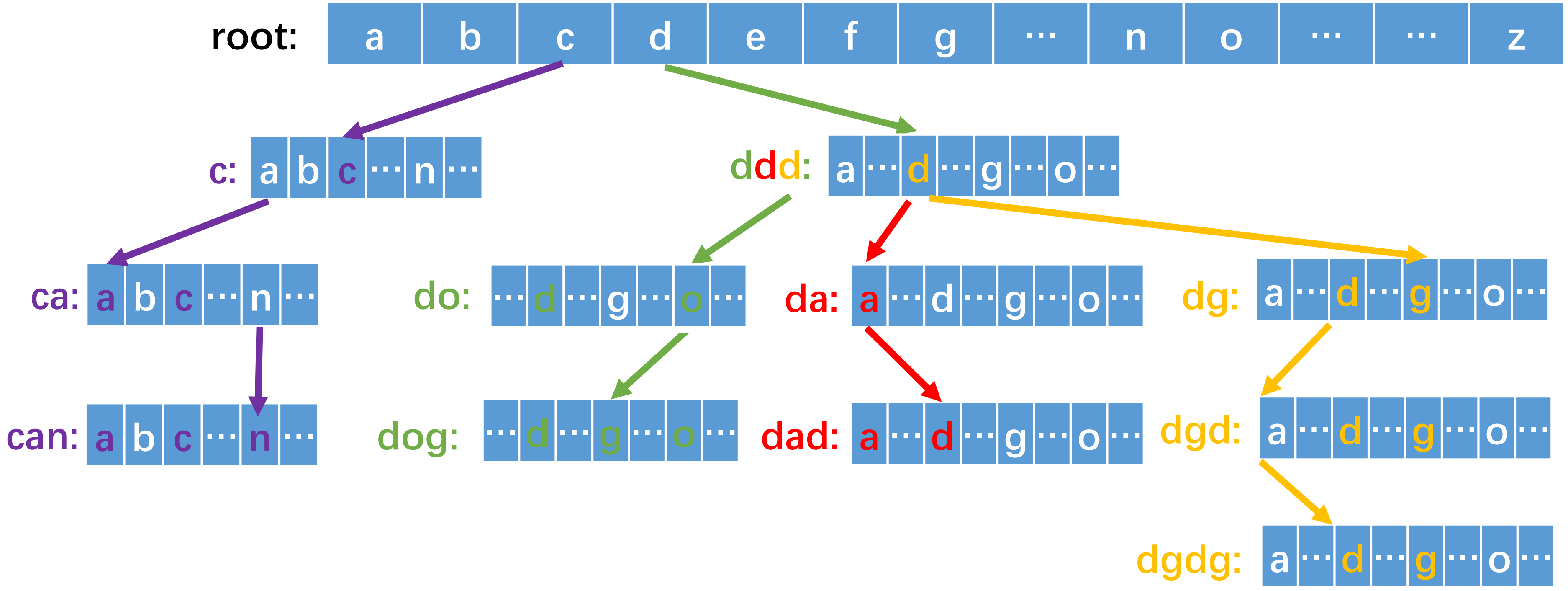
G1: 132

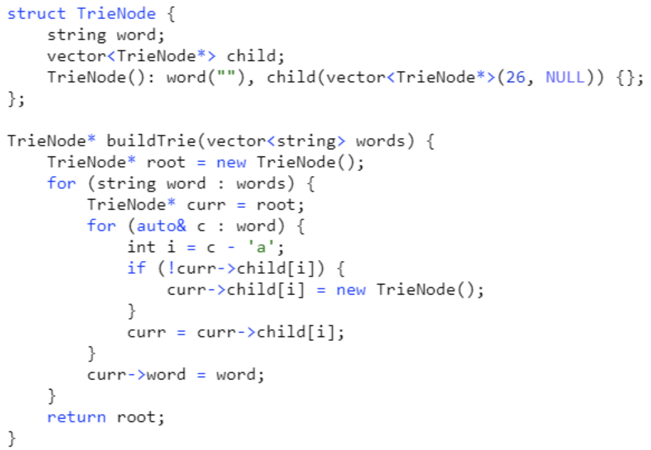
dog

dad

dgdg

can





```
struct TrieNode {
    string word;
    vector<TrieNode*> child;
    TrieNode(): word(""), child(vector<TrieNode*>(26, NULL)) {};
};

TrieNode* buildTrie(vector<string> words) {
    TrieNode* root = new TrieNode();
    for (string word : words) {
        TrieNode* curr = root;
        for (auto& c : word) {
            int i = c - 'a';
            if (!curr->child[i]) {
                curr->child[i] = new TrieNode();
            }
            curr = curr->child[i];
        }
        curr->word = word;
    }
    return root;
}
```

III. Specific Topics

G. Trie (a.k.a 前缀树/prefix tree)

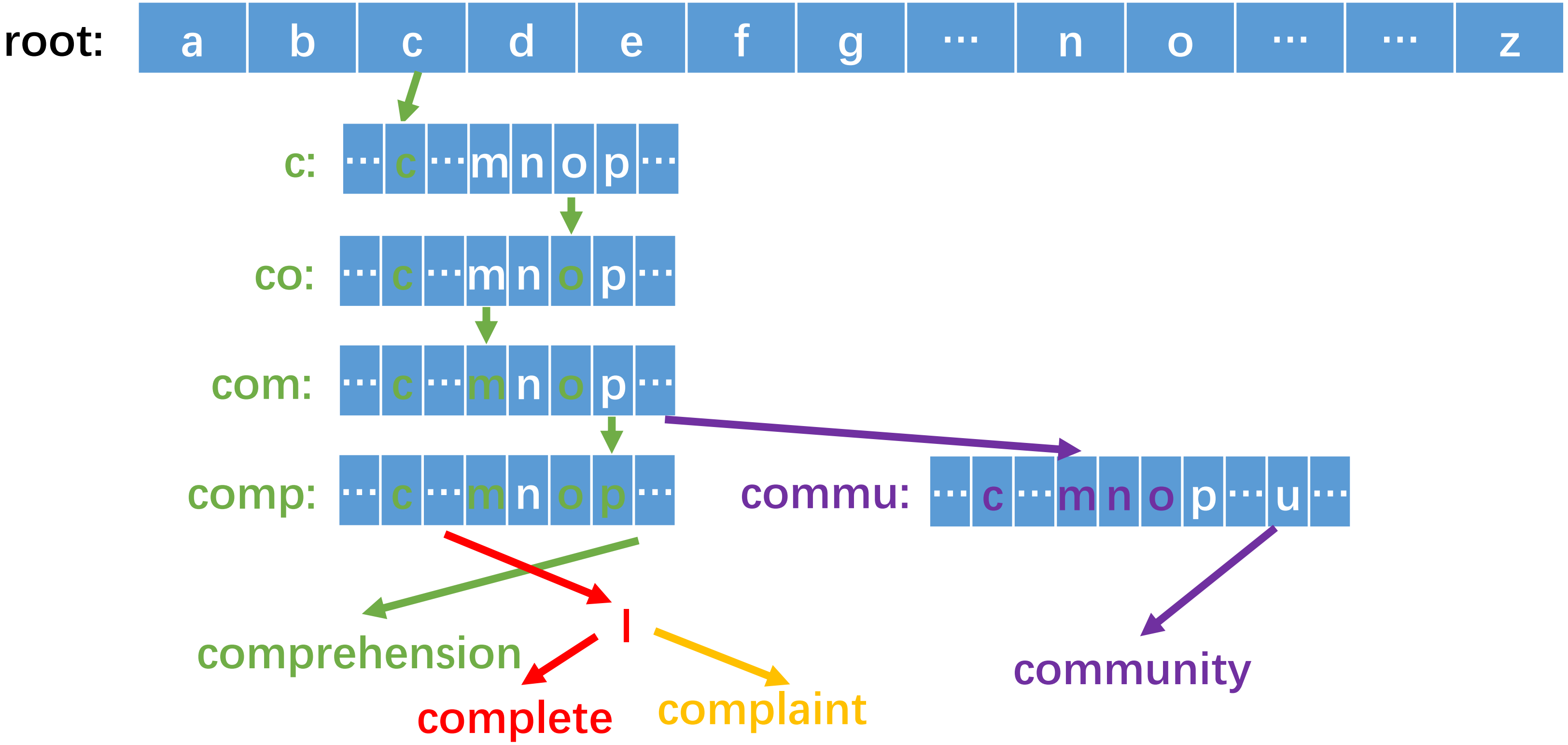
G1: 132

comprehension

complete

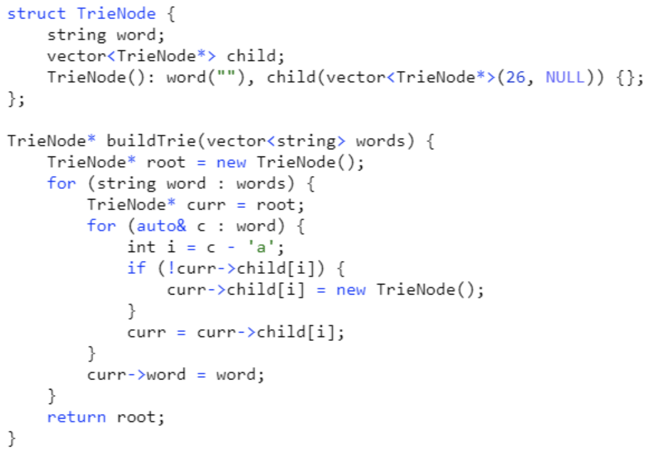
complaint

community



III. Specific Topics

G. Trie (a.k.a 前缀树/prefix tree)



```
struct TrieNode {
    string word;
    vector<TrieNode*> child;
    TrieNode(): word(""), child(vector<TrieNode*>(26, NULL)) {};
};

TrieNode* buildTrie(vector<string> words) {
    TrieNode* root = new TrieNode();
    for (string word : words) {
        TrieNode* curr = root;
        for (auto& c : word) {
            int i = c - 'a';
            if (!curr->child[i]) {
                curr->child[i] = new TrieNode();
            }
            curr = curr->child[i];
        }
        curr->word = word;
    }
    return root;
}
```

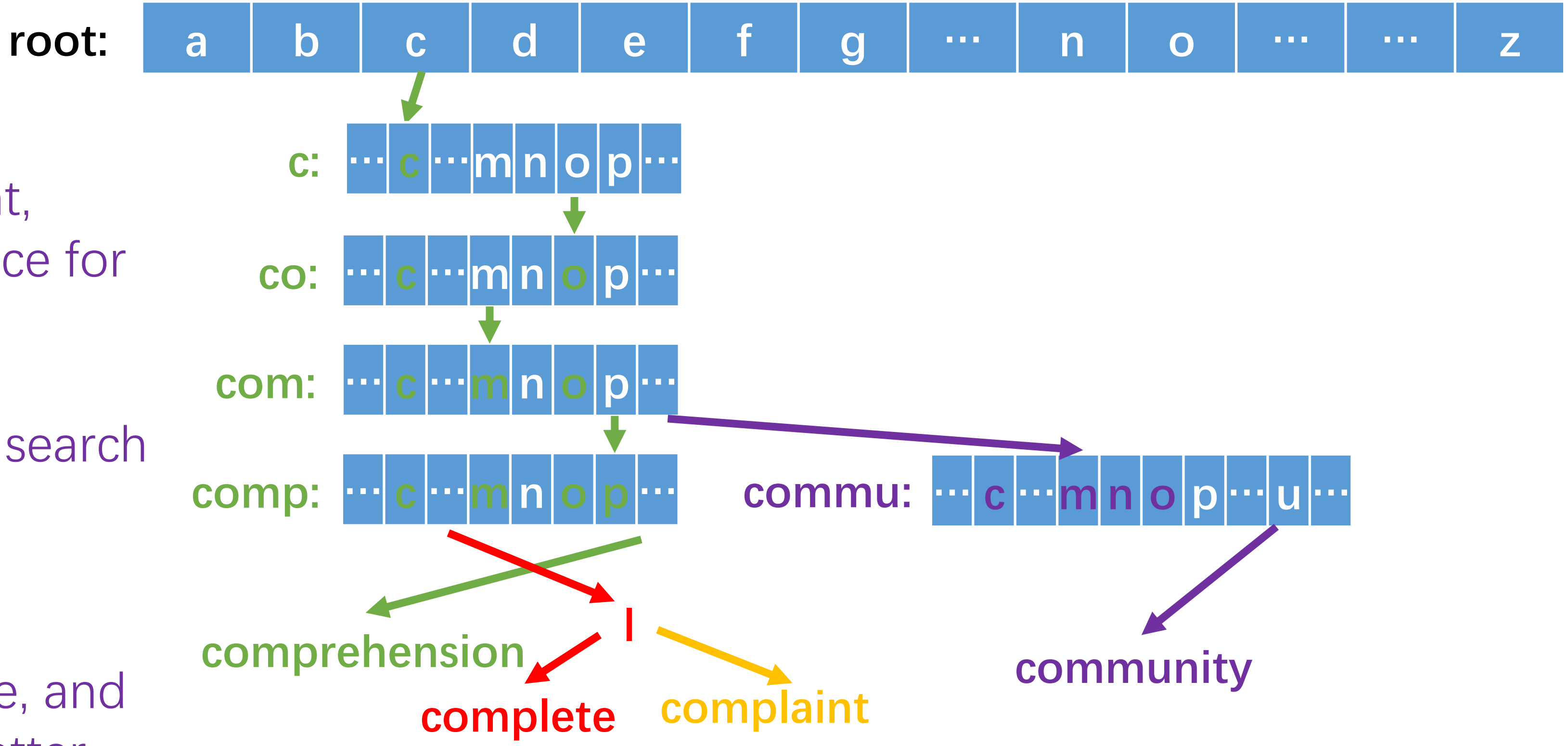
G1: 132

Reason of acceleration:

Though words are different, we only need to search once for their common prefix.

That means we needn’t to search those words as a whole one by one.

We do DFS. We search one, and search the next from the letter where the separated.


root:
a b c d e f g … n o … … z
c:
… c … m n o p …
co:
… c … m n o p …
com:
… c … m n o p …
comp:
… c … m n o p …
commu:
… c … m n o p … u …
comprehension
l
complete
complaint
community