Algorithm Cheat Sheet (Pageless)

Contents

1 QuickSelect

QuickSelect is an algorithm to find the **k-th smallest (or largest) element** in an unsorted array. It is related to QuickSort but only recurses on the side containing the k-th element.

- **Average time complexity:** $O(n)$
- **Worst-case time complexity:** $O(n^2)$
- **Space complexity:** $O(1)$ (in-place, recursive stack $O(\log n)$ on average)

1.1 Implementation

```
// Partition function: rearranges elements around a pivot
// After partitioning:
// - elements <= pivot are on the left
// - elements > pivot are on the right
// Returns the final index of the pivot
int partition(vector<int>& arr, int left, int right) {
    // Randomly pick a pivot index to reduce worst-case
    int pivot_idx = left + rand() % (right - left + 1);
    int pivot = arr[pivot_idx];

    // Move pivot to the end temporarily
    swap(arr[pivot_idx], arr[right]);

    int i = left;  // i points to the next position for swapping
        smaller elements
    for (int j = left; j < right; j++) {
        if (arr[j] <= pivot) {    // If element <= pivot, move it to
            left
            swap(arr[i], arr[j]);
            i++;
        }
    }
    // Place pivot in its correct sorted position
    swap(arr[i], arr[right]);
    return i; // return the index of the pivot
}

// QuickSelect: finds the k-th smallest element (1-indexed)
int quickSelect(vector<int>& arr, int left, int right, int k) {
    if (left == right) return arr[left]; // only one element

    // Partition the array and get pivot index
    int pivot_idx = partition(arr, left, right);
    int count = pivot_idx - left + 1; // number of elements <= pivot

    if (count == k) {
        return arr[pivot_idx]; // pivot is the k-th smallest element
    } else if (k < count) {
        // k-th element lies in left partition
        return quickSelect(arr, left, pivot_idx - 1, k);
    } else {
        // k-th element lies in right partition
        // adjust k because we discard left partition
        return quickSelect(arr, pivot_idx + 1, right, k - count);
    }
}
```

1.2 Practice Problems

- K-th Largest Element in an Array (LeetCode)

2 Trie (Prefix Tree)

Efficient data structure for string retrieval problems (prefixes, alphabets, etc).

2.1 Implementation

```
const int ALPHABET = 26; // number of lowercase letters

struct TrieNode {
    TrieNode *children[ALPHABET]; // pointers to child nodes
    int terminal;                 // number of words ending at this
        node
};

// Create and initialize a new Trie node
TrieNode *new_node() {
    TrieNode *node = new TrieNode;
    for(int i = 0; i < ALPHABET; ++i)
        node->children[i] = nullptr; // initialize all children to
            null
    node->terminal = 0;                // no word ends here yet
    return node;
}

// Insert a string into the trie
void insert(TrieNode *node, string s) {
    for (char c : s) {
        int idx = c - 'a';          // map char to index 0-25
        if (!node->children[idx])   // if child does not exist,
            create it
            node->children[idx] = new_node();
        node = node->children[idx]; // move to the child
    }
    node->terminal++;               // mark end of word
}

// Remove a string from the trie
void remove(TrieNode *node, string s) {
    for (char c : s) {
        int idx = c - 'a';
        if (!node->children[idx]) return; // word not found
        node = node->children[idx];
    }
    if (node->terminal > 0) node->terminal--; // unmark end of word
}

// Search for a string in the trie
bool search(TrieNode *node, string s) {
    for (char c : s) {
        int idx = c - 'a';
        if (!node->children[idx]) return false; // missing letter
        node = node->children[idx];
    }
    return node->terminal > 0; // true if word ends here
}
```

2.2 Practice Problems

- Codeforces 706D

3 Huffman Coding

4 Unit Testing

How to write unit tests for your code.

4.1 Pytest

Simple Python program:

```
# file: math_utils.py
def add(a, b):
    return a + b

def divide(a, b):
    if b == 0:
        raise ValueError("Cannot divide by zero")
    return a / b
```

```
# file: test_math_utils.py
import pytest
from math_utils import add, divide

def test_add():
    assert add(2, 3) == 5
    assert add(-1, 1) == 0
    assert add(0, 0) == 0

def test_divide():
    assert divide(6, 3) == 2
    assert divide(5, 2) == 2.5

    # Check that dividing by zero raises an exception
    with pytest.raises(ValueError):
        divide(1, 0)
```

How to run the tests:

```
pytest test_math_utils.py
```

4.2 Catch2

```
// file: math_utils.hpp
int add(int a, int b) {
    return a + b;
}

double divide(double a, double b) {
    if (b == 0) throw std::runtime_error("Cannot divide by zero");
    return a / b;
}
```