**Đề số 13:**

Câu1:Nêu ý tưởng về thuật toán sắp xếp (counting sort) và minh họa A={1,4,1,2,7,5,2)

Counting Sort là một thuật toán sắp xếp không dựa trên so sánh. Thay vì so sánh các phần tử, thuật toán sử dụng một mảng đếm để xác định số lần xuất hiện của từng giá trị, từ đó sắp xếp các phần tử.

Để đơn giản, giả sử dữ liệu trong khoảng 0 đến 9 ◦

Dữ liệu đầu vào: 1, 4, 1, 2, 7, 5, 2

Bước 1: Đếm số lần xuất hiện của từng phần tử trong mảng cần sắp

Index: 0 1 2 3 4 5 6 7 8 9

Count: 0 2 2 0 1 1 0 1 0 0

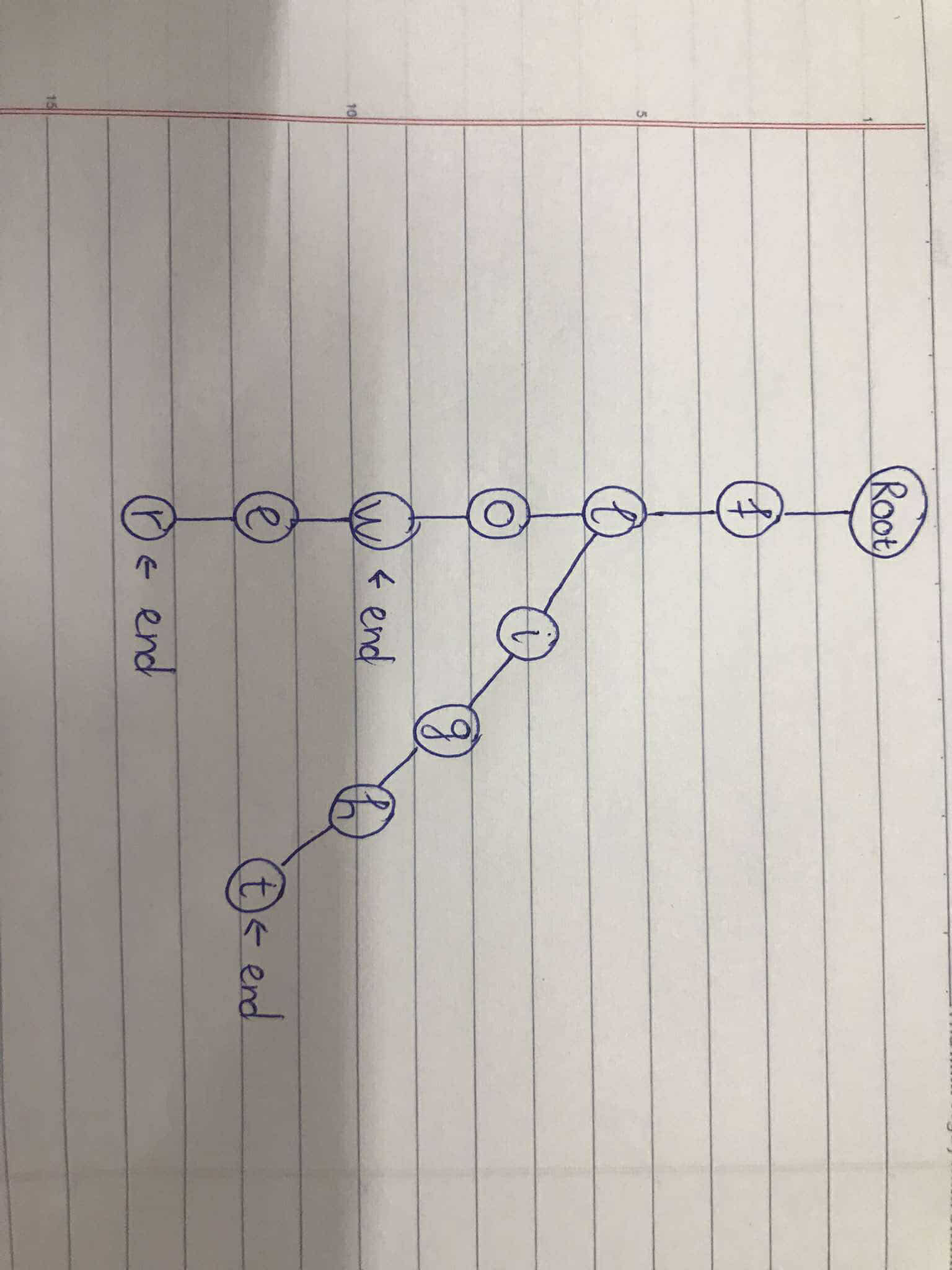
Bước 2: Sửa mảng count sao cho chỉ số của mỗi phần tử lưu tổng số lần đếm trước đó → vị trí của chúng trong mảng đầu ra

Index: 0 1 2 3 4 5 6 7 8 9

Count: 0 2 4 4 5 6 6 7 7 7

Bước 3: Xuất từng phần tử trong chuỗi đầu vào tại ví trí là tổng số lần đếm trừ đi 1 và trừ số lần đếm đi 1. Xử lý dữ liệu đầu vào: 1, 4, 1, 2, 7, 5, 2. Vị trí của 1 là (2-1) = 1 thì đặt số 1 vào chỉ số 1 trong mảng đầu ra. Gặp số 1 tiếp theo thì số lần đếm lúc này là 1 và giảm đi thành 0 và đặt số 1 vào chỉ số 0

Câu 2: Xâu kí tự s=[]{“flower” , “flow”, ”flight”}



#include <iostream>

#include <unordered\_map>

#include <vector>

#include <string>

using namespace std;

// Định nghĩa một nút trong Trie

struct TrieNode {

unordered\_map<char, TrieNode\*> children; // Các con của nút

bool isEndOfWord; // Đánh dấu đây là điểm kết thúc của một từ

TrieNode() {

isEndOfWord = false;

}

};

class Trie {

private:

TrieNode\* root;

public:

Trie() {

root = new TrieNode(); // Khởi tạo gốc Trie

}

// Hàm chèn một từ vào Trie

void insert(const string& word) {

TrieNode\* current = root;

for (char ch : word) {

if (current->children.find(ch) == current->children.end()) {

current->children[ch] = new TrieNode();

}

current = current->children[ch];

}

current->isEndOfWord = true; // Đánh dấu kết thúc từ

}

// Hàm tìm tiền tố chung dài nhất

string findLongestCommonPrefix() {

string prefix = "";

TrieNode\* current = root;

// Duyệt xuống cây Trie đến khi không thể tiếp tục

while (current && current->children.size() == 1 && !current->isEndOfWord) {

auto it = current->children.begin(); // Lấy nút con duy nhất

prefix += it->first; // Thêm ký tự vào tiền tố

current = it->second; // Tiếp tục duyệt nút con

}

return prefix;

}

};

int main() {

// Test case

vector<string> words = {"flower", "flow", "flight"};

Trie trie;

// Chèn từng từ vào Trie

for (const string& word : words) {

trie.insert(word);

}

// Tìm tiền tố chung dài nhất

string lcp = trie.findLongestCommonPrefix();

// In kết quả

cout << "Tiền tố chung dài nhất: " << lcp << endl;

return 0;

}

Cau 3: Viết chương trình chèn dãy giá trị {4371,1323,6173,4199,4344,9679,1989} vào bảng băm thăm dò bậc 2 với hàm băm hash(x)=x%10 và vẽ hình minh họa chèn từng giá trị trên vào bảng băm thăm dò bậc 2

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | 4371 | 1323 | 6173 | 4199 | 4344 | 9679 | 1989 |
|  |  |  |  |  |  |  |  |  |
| 0 |  |  |  |  |  |  | 9679 | 9679 |
| 1 |  | 4371 | 4371 | 4371 | 4371 | 4371 | 4371 | 4371 |
| 2 |  |  |  |  |  |  |  |  |
| 3 |  |  | 1323 | 1323 | 1323 | 1323 | 1323 | 1323 |
| 4 |  |  |  | 6173 | 6173 | 6173 | 6173 | 6173 |
| 5 |  |  |  |  |  | 4344 | 4344 | 4344 |
| 6 |  |  |  |  |  |  |  |  |
| 7 |  |  |  |  |  |  |  |  |
| 8 |  |  |  |  |  |  |  | 1989 |
| 9 |  |  |  |  | 4199 | 4199 | 4199 | 4199 |

#include "bits/stdc++.h"

using namespace std;

const int TABLE\_SIZE = 10;

int hashFunction(int x){

return x%TABLE\_SIZE;

}

void insert(vector<int>& hashTable, int value){

int index = hashFunction(value);

int OGIndex = index;

int i = 1;

while(hashTable[index] != -1){

index = (OGIndex + i \* i) % TABLE\_SIZE;

i++;

if(index >= TABLE\_SIZE){

cout << "Bang bam day, ko the chen: " << value << endl;

return;

}

}

hashTable[index] = value;

cout << "Chen gia tri " << value << " vao vi tri " << index << endl;

}

void printHashTable(const vector<int>& hashTable) {

cout << "\nBang bam:\n";

for (int i = 0; i < TABLE\_SIZE; i++) {

if (hashTable[i] == -1) {

cout << "[" << i << "] (trong)\n";

} else {

cout << "[" << i << "] " << hashTable[i] << endl;

}

}

}

int main(){

vector<int> hashTable(TABLE\_SIZE, -1);

vector<int> values = {4371, 1323, 6173, 4199, 4344, 9679, 1989};

for (int value : values) {

insert(hashTable, value);

}

printHashTable(hashTable);

#### }