**Câu 1:(Sắp xếp ngoài)**

Phương pháp trộn Run-Thuật Toán(Ý tưởng)

Thuật toán sắp xếp tập tin bằng phương pháp trộn Run có thể tóm lược như sau:

* Input:f0 là tập tin cần sắp thứ tự
* Output:f0 là tập tin đã được sắp thứ tự
* Trong đó f0,f1,f2 có thể là các tập tin văn bản thường (text file) hoặc các tập tin nhị phân

Minh Họa:

**Bước 1 :**

* Giả sử các phần tử trên f0 là:

3 **6** 7 **9** 2 **5** 8 **4** 1

* Khởi tạo f1,f2 rỗng
* Thực hiện phân bố m = 1 phần tử từ f0 lần lượt vào f1,f2

f1: 3 7 2 8 1

f2: **6 9 5 4**

* Trộn f1,f2 vào f0

f0: 3 6 **7 9** 2 5  **4** **8** 1

**Bước 2 :**

* Phân bố m=2 \* m=2 phần tử từ f0 vào f1,f2
* f0:3 6 **7 9** 2 5 **4** **8** 1
* f1:3 6 2 5 1
* f2:**7 9 4** **8**
* Trộn f1,f2 thành f0:
* f0:3 6 **7 9** 2 5 **4 8** 1

**Bước 3 :**

* Tương tự bước 2,phân bố m = 2 \* m =4 phần tử từ f0 vào f1,f2
* f0:3 6 **7 9** 2 5 **4 8** 1
* f1:3 6 2 5 1
* f2:**7 9 4 8**
* Trộn f1,f2 thành f0:
* f0:2 3 4 5 6 7 8 9 **1**

**Bước 4 :**

* Phân bố m = 2 \* m =8 phần tử lần lượt từ f0 vào f1,f2
* f1: 2 3 4 5 6 7 8 9
* f2:**1**
* Trộn f1,f2 thành f0:
* f0:1 2 3 4 5 6 7 8 9

**Bước 5:**

* Lặp lại tương tự các bước trên cho tới khi chiều dài m của run cần phân bố lớn hơn chiều dài N của f0 thì dừng

**Câu 2:**

**a.**

**b**.#include <bits/stdc++.h>

using namespace std;

struct MinHeapNode {

char data; unsigned freq;

MinHeapNode \*left, \*right;

MinHeapNode(char data, unsigned freq)

{

left = right = NULL;

this->data = data;

this->freq = freq;

}

};

struct compare {

bool operator()(MinHeapNode\* l, MinHeapNode\* r)

{

return (l->freq > r->freq);

}

};

void printCodes(struct MinHeapNode\* root, string str)

{

if (!root)

return;

if (root->data != '$')

cout << root->data << ": " << str << "\n";

printCodes(root->left, str + "0");

printCodes(root->right, str + "1");

}

void HuffmanCodes(char data[], int freq[], int size)

{

struct MinHeapNode \*left, \*right, \*top;

priority\_queue<MinHeapNode\*, vector<MinHeapNode\*>, compare> minHeap;

for (int i = 0; i < size; ++i)

minHeap.push(new MinHeapNode(data[i], freq[i]));

while (minHeap.size() != 1) {

left = minHeap.top();

minHeap.pop();

right = minHeap.top();

minHeap.pop();

top = new MinHeapNode('$', left->freq + right->freq);

top->left = left;

top->right = right;

minHeap.push(top);

}

printCodes(minHeap.top(), "");

}

int main()

{ char arr[] = { 'a', 'b', 'c', 'd', 'e', 'f' };

int freq[] = { 5, 9, 12, 13, 16, 45 };

int size = sizeof(arr) / sizeof(arr[0]);

HuffmanCodes(arr, freq, size);

return 0;}

**c .**

Quá trình xây dựng cây Huffman gồm các bước sau:

**Bước 2.1**: Sau khi đếm tần suất xuất hiện các phần tử đầu vào. Chúng ta tạo danh sách các nút lá với trọng số là tần suất xuất hiện. Danh sách sẽ có 6 phần tử như bên dưới.

Bước 2.2 và 2.3: Chọn 2 nút có trọng số thấp nhất, tạo nút gốc mới có trọng số bằng tổng 2 trọng số nút con. Sau đó gắn 2 nút con vào nút gốc và đẩy lại vào danh sách. Danh sách cần được biểu diễn đặc biệt để có thể lấy ra các nút trọng số nhỏ nhất một cách tối ưu nhất.

**Bước 2.4**: Lặp lại các bước 2.2 và 2.3.

**Bước 2.4**: Lặp lại các bước 2.2 và 2.3.

**Bước 2.5**: Chỉ còn lại 1 nút trong danh sách, nút này chính là cây Huffman

**Câu 3:**

+Vẽ hình minh họa:

h=(h+i) % Tablesize

I = 0 ->i=1->i=2->i=3

h=h+i

h=3+1=4

h=4+1=3 h=9+1=10%10=0

(9+2)%10=1

h=(9+3)%10=2

Code:(\*)

#include<bits/stdc++.h>

using namespace std;

int hashTableSize = 10;

int size = 0;

class Node{

public:

int key;

bool ischeck = true;

};

int hash(int x){

int index; index = x % hashTableSize;

return index;}

void insert(int x, Node a[]){

int index = hash(x);

if(size == hashTableSize){cout << "Bang bam da day khong the them phan tu " <<x <<" vao bang bam" << endl;}

else{ while(a[index].ischeck == false){index = (index + 1) % hashTableSize;cout <<"a ";}

a[index].key = x;a[index].ischeck = false;size++;}}

int main(){

Node a[hashTableSize];

insert(4371,a);insert(1323,a);

insert(6173,a); insert(4199,a);

insert(4344,a);insert(9679,a);

insert(1989,a);

for(int i = 0 ; i < 10 ; i++)

cout<<a[i].key<<"\t";}

**Câu 1** : Cho mảng số nguyên . Hãy sắp xếp chúng theo nguyên tắc: a đứng trước b nếu tổng các chứ số của a hỏ hơn b . Nếu hai số có tổng các chữ số bằng nhau, số nào nhỏ hơn sẽ đứng trước. Ví dụ số 4 và số 13 có tổng chữ số bằng nhau , nhưng 4 < 13 trong mảng kết quả VD : a={13,20,7,4} thì kết quả là : {20,4,13,7}

**Giải**

def sum\_of\_number(num):

sum = 0

for i in num:

sum += int(i)

return sum

a = [int(i) for i in input().split(",")]

print(a)

print(sum\_of\_number(str(a[0])))

# sort by sum of digits, if sum is same then sort by number

a.sort(key=lambda x: (sum\_of\_number(str(x)), x))

print(a)

**Câu 2** : Tìm mốt và trung bị của 1 mảng số sử dụng phương pháp sắp xếp đếm phân phối ( countign sort) . Minh họa các bước tìm mốt với mảng số A={1,4,1,2,7,1,2,5,3,6}**Chú ý:** mốt là số có tần suất lớn nhất trong mảng. Với 1 mảng các số được sắp xếp theo thứ tự tăng dần, trung vị là số chính giữa mảng nếu mảng có số lẻ phần tử, là trung bình của 2 số chính giữa mảng nếu mảng có số chẵn phần tử **Giải**

def counting\_sort(a):

max\_num = max(a)

count\_list = [0] \* (max\_num+1)

for i in a:

count\_list[i] += 1

for i in range(len(count\_list)):

for j in range(count\_list[i]):

print(i, end=" ")

a = [int(i) for i in input().split(",")]

counting\_sort(a)

**Bài 1(chương2)**Cho một vector chứa các số nguyên. Hãy đưa ra số lượng phần tử khác nhau trong vector đó.Ví dụ: Với inputVector = [1, 3, 3, 2], kết quả là 3.

#include <bits/stdc++.h>

using namespace std;

int main(){ set<int> s;

int n,val;cin >> n;

for (int i = 0; i < n; i++){cin >> val;s.insert(val);}

cout << s.size();}

**Bài 2(chương2)** Anh Việt đang thống kê số liệu cho công ty, anh muốn giá trị nhỏ thứ hai ở trong mộtdãy số nguyên. Hãy giúp anh Việt tìm ra giá trị đó, nếu không có kết quả như yêu cầu thì trả về "NO". Ví dụ: Với arr = [1, 2, 3, 1, 1] thì kết quả sẽ là 2.

#include <bits/stdc++.h>

using namespace std;

int main(){

set<int> s;

int n,val;cin >> n;

for (int i = 0; i < n; i++){cin >> val;s.insert(val);}

if (s.size() <= 1){cout << "NO";}

else{set<int>::iterator i = s.begin();advance(i, 1);cout << \*i; }

return 0;}

**Bài 3(chương2)** Tính số lượng tối thiểu các ký tự cần thay đổi trong chuỗi str để chuỗi đó có k ký tựkhác nhau hoặc xuất ra điều đó là không thể. Chuỗi str chỉ bao gồm những chữ cái latinh viết thường và cũng chỉ có thể thay thành những chữ cái latinh viết thường.

#include <bits/stdc++.h>

using namespace std;

string diversity(string str, int k){

set<char> s;int i;

if (str.size() < k){return "impossible";}

for (i = 0; i < str.size(); i++){s.insert(str[i]);}

if (s.size() >= k){i = 0;}

else {i = k - s.size();}

return to\_string(i);}

int main(){ string str;int k;cin >> str >> k;

cout << diversity(str, k);}

**Bài 4(chương2)** An có một danh sách các sản phẩm trong siêu thị nhưng thật không may trong danhsách lại có những sản phẩm xuất hiện nhiều lần. Bạn hãy giúp An làm lại danh sách sao chomột sản phẩm chỉ xuất hiện một lần trong danh sách và các sản phẩm xuất hiện theo thứ tăng dần trong từ điển.

#include <bits/stdc++.h>

using namespace std;

vector<string> Solve(vector<string> &vstr){

vector<string> res;set<string> s;

for (auto &&i : vstr){s.insert(i);}

for (auto &&i : s){res.push\_back(i);}

return res;}

int main(){vector<string> test = {"abc", "ab", "abc"};

for (auto &&i : Solve(test)){cout << i << " ";}

return 0;}

**Câu 5(chương2)** Số siêu nguyên tố là số:

•Bản thân nó là số nguyên tố.

•Khi xóa đi lần lượt các chữ số sau cùng của nó thì số mới vẫn là số nguyên tố.Ví dụ 2393 là số siêu nguyên tố vì 2393, 239, 23, 2 là số nguyên tố.Cho một số n, hãy đưa số dãy số siêu nguyên tố nhỏ hơn hoặc bằng n đã được sắp xếp tăng

dần.Ví dụ:

• Với n = 30; thì superPrimeNumber(n) = [2, 3, 5, 7, 23, 29];

Vì các số 2, 3, 5, 7, 23 và 29 đều là số siêu nguyên tố và nhỏ hơn hoặc bằng 30.

#include <bits/stdc++.h> using namespace std;

bool isPrime(int n){

if (n < 2){return 0;}

for (int i = 2; i <= sqrt(n); i++){if (n % i == 0){return 0;}}

return 1;}

vector<int> superPrimeNumber(int n){

queue<int> q;

vector<int> res;

for (int i = 2; i <= n, i < 10; i++){if (isPrime(i)){q.push(i);}}

while (!q.empty()){for (int i = 1; i <= 9; i++){int k = q.front() \* 10 + i;if (k <= n && isPrime(k)){q.push(k);}}

res.push\_back(q.front());

q.pop();}

return res;

}

int main(){

for (auto &&i : superPrimeNumber(30)){cout << i << " ";}

return 0;}

**Bài 12(chương2)** Cho một mảng số nguyên và một số nguyên k. Tìm số lớn nhất trong từng mảng conliên tiếp kích thước k.Ví dụ, với mảng: arr[] = {1, 2, 3, 1, 4, 5, 2, 3, 6}, k = 3

#include <bits/stdc++.h>

using namespace std;

int main(){

int a[1000],n, k;

cin >> n >> k;

for (int i = 0; i < n; i++){cin >> a[i];}

deque<int> dq;

for (int i = 0; i < k; i++)

{ while (!dq.empty() && a[i] >= a[dq.back()]){dq.pop\_back();}

dq.push\_back(i);}

for (int i = k; i < n; i++)

{ cout << a[dq.front()] << " ";

while (!dq.empty() && dq.front() <= i-k){dq.pop\_front();}

while (!dq.empty() && a[i] >= a[dq.back()]){dq.pop\_back();}

dq.push\_back(i);}

cout << a[dq.front()] << " ";

return 0;}

**Bài 14: (chương2)** Cho dãy số nguyên n phần tử a1, a2, ..., an, hãy thống kê tần suất xuất hiện của từng phần tử có trong dãy theo thứ tự từ bé đến lớn.

#include <bits/stdc++.h>

using namespace std;

int main(){ map<int, int> mp;int n, val;cin >> n;

for (int i = 0; i < n; i++){cin >> val;mp[val]++;}

for (auto &&i : mp){cout << i.first << " freq: " << i.second << "\n";}

return 0;}

**Bài 15: (chương2)** Cho dãy số nguyên n phần tử a1, a2, ..., an, hãy tìm những phần tử có tần suất xuất hiện nhiều nhất trong dãy. Ví dụ: a = {4 7 2 8 4 8 3 2}. Kết quả là các số {2 4 8} có tần suất lớn nhất là 2.

#include <bits/stdc++.h>

using namespace std;

int main()

{map<int, int> mp;

int n, val;cin >> n;

for (int i = 0; i < n; i++){cin >> val;mp[val]++;}

int max\_freq = INT\_MIN;

for (auto &&i : mp){if (i.second > max\_freq){max\_freq = i.second;}}

for (auto &&i : mp){if (i.second == max\_freq){cout << i.first << " ";}}

return 0;}

**Bài 18: (chương2)** Cho hai dãy số nguyên arr1 và arr2, hãy tính tổng những số xuất hiện trong cả hai dãy, lưu ý là mỗi số chỉ được tính một lần.

#include <bits/stdc++.h>

using namespace std;

int main()

{map<int, bool> check;

int n1, arr1,n2, arr2,sumOfCommon = 0;cin >> n1;

for (int i = 0; i < n1; i++){cin >> arr1;check[arr1] = true;}

cin >> n2;

for (int i = 0; i < n2; i++){cin >> arr2;if (check[arr2] == true){sumOfCommon += arr2;check[arr2] = false;}}

cout << "sumOfCommon(arr1, arr2) = " << sumOfCommon; return 0;}

**Bài 19: (chương2**) Cho một dãy số nguyên arr và một số nguyên dương

sum.Hãy kiểm tra xem dãy số có tồn tại hai số có tổng bằng sum

hay không.

#include <bits/stdc++.h>

using namespace std;

bool checkSum(vector<int> &arr, int sum)

{ map<int, int> cnt;

for (auto &&i : arr){cnt[i]++;}

for (auto &&i : arr){

if (i != sum - i && cnt[sum - i] >= 1){return true;}

if (i == sum - i && cnt[sum - i] >= 2){return true;}}

return false;

}

int main(){vector<int> a = {2, 2, 3, -1, 9, 8};

cout << checkSum(a, 4); return 0;}

**Bài 20: (chương2)** Cho một dãy gồm các số nguyên, hãy đưa ra số thỏa mãn là số lớn nhất trong những

số có tần số bé nhất.

#include <bits/stdc++.h>

using namespace std;

int largestElement(vector<int> &arr)

{

map<int, int> cnt;

int min\_freq = INT\_MAX;

int max\_value = INT\_MIN;

for (auto &&i : arr){cnt[i]++;}

for (auto &&i : cnt){if (i.second <= min\_freq){min\_freq = i.second;max\_value = i.first;}}

return max\_value;

}

int main()

{ vector<int> arr = {2,2,4,4,7,7,7};

cout << largestElement(arr);

return 0;}

**Câu 22: (chương2)** Cho danh sách các sản phẩm của 2 kho hàng A và B.

Do chiến lược kinh doanh bạnđược giao nhiệm nhập các sản phẩm

từ kho B vào kho A sao cho những sản phẩm nào đã có

trong kho A thì không nhập.

Ví dụ:

• Với A = ["Banana", "Banana", "Apple"], B = ["Orange", "Apple", "Banana",

"Watermelon"] thì mergeProducts(A, B) = [true, false, false, true].

#include <bits/stdc++.h>

using namespace std;

vector<bool> mergeProducts(vector<string> &A, vector<string> &B)

{

vector<bool> res;map<string, bool> check;

for (auto &&i : A){check[i] = true;}

for (auto &&i : B){if (check[i]){res.push\_back(false);}else{res.push\_back(true);}}

return res;

}

int main(){

vector<string> A = {"Banana", "Banana", "Apple"};

vector<string> B = {"Orange", "Apple", "Banana", "Watermelon"};

for (auto &&i : mergeProducts(A, B)){cout << boolalpha << i << " ";}

return 0;}

**Câu 5(chương 3)**

**+)dây chuyền**

#include<bits/stdc++.h>

using namespace std;

int hash(int x){

int index;index = x % 10;

return index;}

void insert(int x, vector<int> a[]){int index = hash(x);a[index].push\_back(x);}

int main(){

vector<int> a[10];insert(4371,a);

insert(1323,a);insert(6173,a);

insert(4199,a);insert(4344,a);

insert(9679,a);insert(1989,a);}

**+)thăm dò tuyến tính(câu3-ởđầu)**

**+)thăm dò bậc hai**

#include<bits/stdc++.h>

using namespace std;

int hashTableSize = 10;

int size = 0;

class Node{

public:

int key;

bool ischeck = true;};

int hash(int x){

int index;index = x % hashTableSize;

return index;}

void insert(int x, Node a[]){

int index = hash(x);

if(size == hashTableSize){ cout << "Bang bam da day khong the them phan tu " <<x <<" vao bang bam" << endl;}

else{ int i = 1;

while(a[index].ischeck == false){index = (index + i\*i) % hashTableSize;i++;cout <<"a ";}

a[index].key = x;a[index].ischeck = false;

size++;}}

int main(){

Node a[hashTableSize];

insert(4371,a);insert(1323,a);

insert(6173,a);insert(4199,a);

insert(4344,a);insert(9679,a);

insert(1989,a);

for(int i = 0 ; i < 10 ; i++)

cout<<a[i].key<<"\t";}

**Câu 6(chương 3)**

**+)dây chuyền**

#include<bits/stdc++.h>

using std::vector;

using std::list;

using std::string;

using std::cout;

using std::endl;

struct dict{

string key,value;

dict():key(""),value(""){};

dict(string Key, string Value):key(Key),value(Value){};

};

class HashChain\_std{

private:

int size,count;

vector<list<dict> > table;

int PreHashing(string key\_str);

int HashFunction(string key\_str);

public:

HashChain\_std(){};

HashChain\_std(int m):size(m),count(0){

table.resize(size); }

void Insert(dict data);

string Search(string key);

};

string HashChain\_std::Search(string key\_str){

int index = HashFunction(key\_str);

for (list<dict>::iterator itr = table[index].begin(); itr != table[index].end(); itr++) {

if ((\*itr).key == key\_str) {

return (\*itr).value;}

}

return "Tu can kiem khong co trong tu dien bang bam";}

void HashChain\_std::Insert(dict data){

int index = HashFunction(data.key);

table[index].push\_front(data);

}

int HashChain\_std::PreHashing(string key\_str){

int exp = 9,key\_int = 0,p = 1;

for (int i = (int)key\_str.size()-1; i >= 0; i--) {

key\_int += key\_str[i]\*p;

p \*= exp;}

return key\_int;}

int HashChain\_std::HashFunction(string key\_str){

return (PreHashing(key\_str) % this->size);}

int main() {

HashChain\_std hash(5);

hash.Insert(dict("dog","cho"));

hash.Insert(dict("hello","xin chao"));

cout <<hash.Search("hi") << ". " << endl;

return 0;}

**+)thăm dò tuyến tính**

#include<bits/stdc++.h>

#include<hashtable.h>

using namespace std;

int hashTableSize = 23;

typedef struct{

string tienganh; string tiengviet;}tudien;

int hash(string s){

int index,sum = 0;

for(int i = 0 ; i < s.size() ; i++) {sum+=(int)s[i];}

index = sum % hashTableSize;

return index;}

void insert(string s, string s1, tudien a[]){

int index = hash(s);

while(a[index].tienganh != ""){

index = (index + 1) % hashTableSize ;}

a[index].tienganh = s; a[index].tiengviet = s1;}

string search(string s, tudien a[]){

int index = hash(s) , i = 0;

while(a[index].tienganh != s && a[index].tienganh != ""){

index = (index + 1) % hashTableSize ;i++;}

if(a[index].tienganh == ""){return "Tu can kiem khong co trong bang bam";}

return a[index].tiengviet;}

int main(){

tudien a[23];

for(int i = 0 ; i < 23 ; i++) {a[i].tienganh = "";}

insert("hello","xin chao",a);

insert("good","tot",a);

insert("sad","buon",a);

cout<<search("hi",a);}

**+)thăm dò bậc hai**

#include<bits/stdc++.h>

#include<hashtable.h>

using namespace std;

int hashTableSize = 23;

typedef struct{

string tienganh; string tiengviet;}tudien;

int hash(string s){

int index,sum = 0;

for(int i = 0 ; i < s.size() ; i++) {sum+=(int)s[i];}

index = sum % hashTableSize;

return index;}

void insert(string s, string s1, tudien a[]){

int index = hash(s),i=1;

while(a[index].tienganh != ""){

index = (index + i\*i) % hashTableSize ;}

a[index].tienganh = s; a[index].tiengviet = s1;}

string search(string s, tudien a[]){

int index = hash(s) , i = 0;

while(a[index].tienganh != s && a[index].tienganh != ""){

index = (index + 1) % hashTableSize ;i++;}

if(a[index].tienganh == ""){return "Tu can kiem khong co trong bang bam";}

return a[index].tiengviet;}

int main(){

tudien a[23];

for(int i = 0 ; i < 23 ; i++) {a[i].tienganh = "";}

insert("hello","xin chao",a);

insert("good","tot",a);

insert("sad","buon",a);

cout<<search("hi",a);}

**Câu3(chương5)** Cho một mảng các xâu ký tự. Ví dụ: s[] = {"flower", "flow", "flight"}.

**a.vẽ minh họa**

**Câu4(chương5) == Câu2(ở đầu)**

**b.code**

class TrieNode:

def \_\_init\_\_(self,letter):

self.letter=letter

self.children={}

self.endword=False

class Trie:

def \_\_init\_\_(self):

self.root=TrieNode("\*")

def add\_word(self,word):

curr\_node=self.root

for letter in word :

if letter not in curr\_node.children:

curr\_node.children[letter]=TrieNode(letter)

curr\_node=curr\_node.children[letter]

curr\_node.endword=True

def tiento(self):

curr\_node=self.root

str=""

while len(curr\_node.children)==1 and curr\_node.endword==False:

for i in curr\_node.children:

curr\_node=curr\_node.children[i]

str+=curr\_node.letter

return str

def does\_word\_exits(self,word):

if word=="":

return False

curr\_node=self.root

for letter in word:

if letter not in curr\_node.children:

return False

curr\_node=curr\_node.children[letter]

return curr\_node.endword

trie=Trie()

words={"flower","flow","flight"}

for word in words:

trie.add\_word(word)

print(trie.tiento())