MAYANK SRIVASTAVA

9922103143

WEEK 6

1-)

#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;

int minProductSubset(vector<int>& arr) {

int n = arr.size();

int countNeg = 0;

int countZero = 0;

int product = 1;

for (int i = 0; i < n; ++i) {

if (arr[i] < 0) {

countNeg++;

product \*= arr[i];

} else if (arr[i] > 0) {

product \*= arr[i];

} else {

countZero++;

}

}

if (countNeg % 2 == 0 && countZero < n) {

return product;

}

return product;

}

int main() {

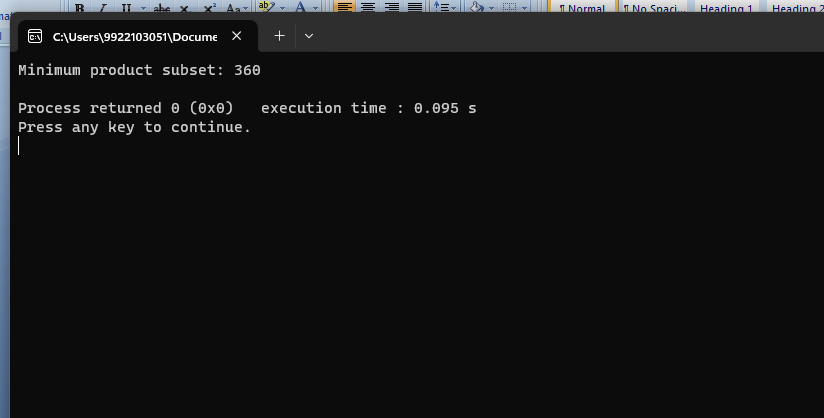
vector<int>arr = { 2, 3, 5, 4, 3 };

int result = minProductSubset( arr);

cout << "Minimum product subset: " << result << endl;

return 0;

}



2-)

#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;

struct Activity {

int start;

int end;

};

bool compareActivities(const Activity& a, const Activity& b) {

return a.end < b.end;

}

int maxActivities(vector<Activity>& activities) {

int n = activities.size();

if (n == 0) {

return 0;

}

sort(activities.begin(), activities.end(), compareActivities);

int count = 1;

int lastEnd = activities[0].end;

for (int i = 1; i < n; ++i) {

if (activities[i].start >= lastEnd) {

count++;

lastEnd = activities[i].end;

}

}

return count;

}

int main() {

int n;

cout << "Enter the number of activities: ";

cin >> n;

vector<Activity> activities(n);

cout << "Enter start and end times for each activity:\n";

for (int i = 0; i < n; ++i) {

cin >> activities[i].start >> activities[i].end;

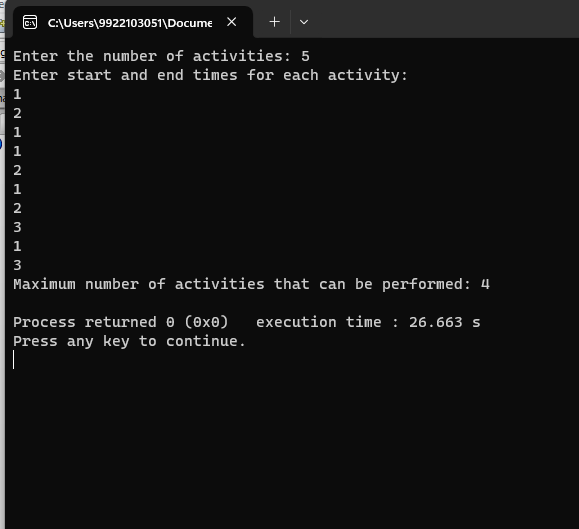
}

int result = maxActivities(activities);

cout << "Maximum number of activities that can be performed: " << result << endl;

return 0;

}



3-)

#include <iostream>

#include <queue>

#include <map>

using namespace std;

struct Node {

char symbol;

int frequency;

Node\* left;

Node\* right;

};

struct CompareNodes {

bool operator()(Node\* a, Node\* b) {

return a->frequency > b->frequency;

}

};

Node\* buildHuffmanTree(map<char, int>& freqMap) {

priority\_queue<Node\*, vector<Node\*>, CompareNodes> minHeap;

for (const auto& entry : freqMap) {

Node\* leaf = new Node;

leaf->symbol = entry.first;

leaf->frequency = entry.second;

leaf->left = nullptr;

leaf->right = nullptr;

minHeap.push(leaf);

}

while (minHeap.size() > 1) {

Node\* leftChild = minHeap.top();

minHeap.pop();

Node\* rightChild = minHeap.top();

minHeap.pop();

Node\* internalNode = new Node;

internalNode->symbol = '\0';

internalNode->frequency = leftChild->frequency + rightChild->frequency;

internalNode->left = leftChild;

internalNode->right = rightChild;

minHeap.push(internalNode);

}

return minHeap.top();

}

void generateHuffmanCodes(Node\* root, string code, map<char, string>& huffmanCodes) {

if (!root)

return;

if (root->symbol != '\0')

huffmanCodes[root->symbol] = code;

generateHuffmanCodes(root->left, code + "0", huffmanCodes);

generateHuffmanCodes(root->right, code + "1", huffmanCodes);

}

int main() {

map<char, int> freqMap = {

{'Q', 3},

{'p', 23},

{'T', 30},

{'a', 12},

{'d', 18}

};

Node\* root = buildHuffmanTree(freqMap);

map<char, string> huffmanCodes;

generateHuffmanCodes(root, "", huffmanCodes);

cout << "Huffman Codes:" << endl;

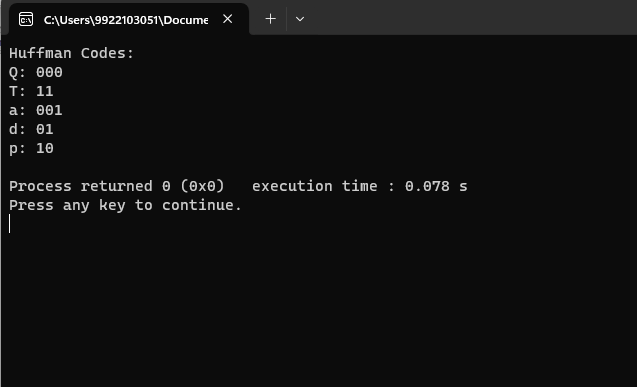
for (const auto& entry : huffmanCodes) {

cout << entry.first << ": " << entry.second << endl;

}

return 0;

}



4-)

#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;

struct Mobile {

string name;

int count;

int profit;

};

bool compareByRatio(const Mobile& a, const Mobile& b) {

return (a.profit \* b.count) > (b.profit \* a.count);

}

int maximizeProfit(vector<Mobile>& mobiles, int capacity) {

sort(mobiles.begin(), mobiles.end(), compareByRatio);

int totalProfit = 0;

for (const Mobile& mobile : mobiles) {

int itemsToAdd = min(capacity, mobile.count);

totalProfit += itemsToAdd \* mobile.profit;

capacity -= itemsToAdd;

if (capacity == 0)

break;

}

return totalProfit;

}

int main() {

vector<Mobile> mobiles = {

{"Redmi Note 9", 10, 20000},

{"Samsung Galaxy M12", 5, 10500},

{"OnePlus 5", 26, 89000},

{"Realme Narzo 20 Pro", 8, 4000},

{"Xiaomi Poco M3", 12, 23000}

};

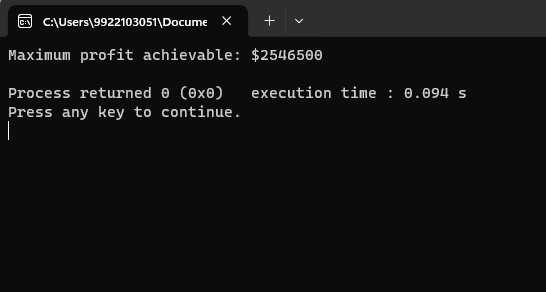
int shopCapacity = 40;

int maxProfit = maximizeProfit(mobiles, shopCapacity);

cout << "Maximum profit achievable: $" << maxProfit << endl;

return 0;

}



5-)

#include <iostream>

#include <vector>

#include <unordered\_map>

using namespace std;

unordered\_map<string, vector<string>> adjacencyList;

void addEdge(const string& district1, const string& district2) {

adjacencyList[district1].push\_back(district2);

adjacencyList[district2].push\_back(district1);

}

int graphColoring() {

unordered\_map<string, int> colorMap;

int numColors = 0;

for (const auto& entry : adjacencyList) {

const string& district = entry.first;

vector<bool> usedColors(numColors, false);

for (const string& neighbor : entry.second) {

if (colorMap.find(neighbor) != colorMap.end()) {

usedColors[colorMap[neighbor]] = true;

}

}

for (int color = 0; color < numColors; ++color) {

if (!usedColors[color]) {

colorMap[district] = color;

break;

}

}

if (colorMap.find(district) == colorMap.end()) {

colorMap[district] = numColors++;

}

}

return numColors;

}

int main() {

addEdge("Agra", "Aligarh");

addEdge("Agra", "Lucknow");

addEdge("etah","budaun");

addEdge("bareilly","moradabad");

addEdge("bijnor","muzzafarnagar");

int minColors = graphColoring();

cout << "Minimum number of colors needed: " << minColors << endl;

return 0;

}

