Q1 [Huffman Coding](https://practice.geeksforgeeks.org/problems/huffman-encoding/0)

#define CHAR\_VERTEX '\*'

class Solution {

class Model {

public:

char c;

int prob;

Model():c(0), prob(0) {}

Model(int character, int probability):c(character), prob(probability) {}

~Model() {}

bool operator==(Model& node) {

bool ret = (prob==node.prob && c==node.c) ? true : false;

return ret;

}

bool operator<(Model& node) {

bool ret = (prob < node.prob) ? true : false;

return ret;

}

bool operator>(Model& node) {

bool ret = (prob > node.prob) ? true : false;

return ret;

}

friend std::ostream& operator<<(std::ostream& os, const Model& node) {

os << "[" << node.c << "]";

os << "[" << node.prob << "]" << endl;

return os;

}

};

template<typename T>

class BinNode {

private:

public:

T data;

int height;

int npl;

BinNode<T>\* parent;

BinNode<T>\* lchild;

BinNode<T>\* rchild;

BinNode(): height(0), npl(1), parent(NULL), lchild(NULL), rchild(NULL) {}

BinNode(const T& e, BinNode<T>\* p=NULL, BinNode<T>\* lc=NULL, BinNode<T>\* rc=NULL, int h=0, int l=1):

data(e), parent(p), lchild(lc), rchild(rc), height(h), npl(l){

}

~BinNode() {

}

BinNode<T>\* insertLeftChild(const T& t);

BinNode<T>\* insertRightChild(const T& t);

bool operator==(const BinNode<T>& node);

bool operator<(const BinNode<T>& node);

bool operator>(const BinNode<T>& node);

BinNode<T>& operator=(const BinNode& node);

bool isRoot() {

return !parent ? true : false;

}

bool isLeftChild() {

if (!isRoot()) {

if (this == parent->lchild) return true;

return false;

}

else return false;

}

bool isRightChild() {

if (!isRoot()) {

if (this == parent->rchild) return true;

return false;

}

else return false;

}

bool isLeaf() {

return !hasChild();

}

bool hasParent() {

return parent ? true : false;

}

bool hasChild() {

return hasLeftChild() || hasRightChild();

}

bool hasBothChild();

bool hasLeftChild() { return lchild != NULL ? true : false; }

bool hasRightChild() { return rchild != NULL ? true : false; }

void traverse() {

std::queue<BinNode<T>\*> queue;

BinNode<T>\* node = this;

queue.push(node);

std::cout << "--- TREE TOP ------" << std::endl;

while (true) {

if (queue.empty()) break;

node = queue.front();

queue.pop();

cout << (\*node);

if (node->hasLeftChild()) queue.push(node->lchild);

if (node->hasRightChild()) queue.push(node->rchild);

}

std::cout << "\n--- TREE BOTTOM ------" << std::endl;

}

friend std::ostream& operator<<(std::ostream& os, const BinNode<T>& node) {

os << node.data;

return os;

}

};

public:

BinNode<Model>\* \_root;

std::map<char, string> codes;

// Encodes a URL to a shortened URL.

void grow(std::list<Model\*>& list) {

std::list<BinNode<Model>\*> subs; //synthetically created vertexes for growing Huffman tree

std::list<Model\*>::iterator it = list.begin();

std::list<BinNode<Model>\*>::iterator it\_subs;

Model\* l;

Model\* r;

Model\* m;

BinNode<Model>\* lchild;

BinNode<Model>\* rchild;

BinNode<Model>\* vertex;

while (it!=list.end()) {

lchild = NULL;

rchild = NULL;

vertex = NULL;

if (std::next(it, 1) == list.end()) {

//cout << "ending..." << endl;

if (subs.size() != 1) {

cout << "!!! error" << endl;

break;

}

l = NULL;

r = \*it;

rchild = new BinNode<Model>(\*r);

lchild = subs.back();

subs.pop\_back();

m = new Model(CHAR\_VERTEX, lchild->data.prob+r->prob);

vertex = new BinNode<Model>(\*m);

rchild->parent = vertex;

lchild->parent = vertex;

vertex->lchild = lchild;

vertex->rchild = rchild;

\_root = vertex;

break;

}

else {

l = \*it;

it++;

r = \*it;

m = new Model(CHAR\_VERTEX, (l->prob)+(r->prob));

for (it\_subs=subs.begin(); it\_subs!=subs.end();it\_subs++) {

if (\*l == (\*it\_subs)->data) { //a copy, not the same address

lchild = \*it\_subs;

subs.erase(it\_subs);

break;

}

}

for (it\_subs=subs.begin(); it\_subs!=subs.end();it\_subs++) {

if (\*r == (\*it\_subs)->data) {

rchild = \*it\_subs;

subs.erase(it\_subs);

break;

}

}

if (lchild == NULL) lchild = new BinNode(\*l);

if (rchild == NULL) rchild = new BinNode(\*r);

vertex = new BinNode(\*m);

rchild->parent = vertex;

lchild->parent = vertex;

vertex->lchild = lchild;

vertex->rchild = rchild;

subs.push\_back(vertex);

for (std::list<Model\*>::iterator itt=it; it!=list.end(); itt++) {

if (\*m < \*\*itt || \*m == \*\*itt) {

list.insert(itt, m);

break;

}

else if (std::next(itt, 1) == list.end()) {

list.push\_back(m);

break;

}

}

it++;

}

}

}

void generate() {

std::queue<BinNode<Model>\*> queue;

BinNode<Model>\* node = \_root;

BinNode<Model>\* tmp;

queue.push(node);

std::string code;

while (true) {

if (queue.empty()) break;

node = queue.front();

queue.pop();

if (node->data.c == CHAR\_VERTEX) {

}

else {

if (node->isLeaf()) code.clear();

tmp = node;

while (tmp) {

if (tmp->isLeftChild()) code.insert(0, "0");

else if (tmp->isRightChild()) code.insert(0, "1");

tmp = tmp->parent;

}

if (node->data.c != CHAR\_VERTEX) codes[node->data.c] = code;

}

if (node->hasLeftChild()) queue.push(node->lchild);

if (node->hasRightChild()) queue.push(node->rchild);

}

}

string encode(string longUrl) {

std::map<char,int> probs;

std::list<Model\*> table;

std::string encoded;

for (string::iterator it=longUrl.begin(); it!=longUrl.end(); it++) {

probs[\*it]++;

}

for (std::map<char,int>::iterator it=probs.begin(); it!=probs.end(); it++) {

Model\* m = new Model(it->first, it->second);

table.push\_back(m);

}

struct comp {

bool operator() (Model\* m1, Model\* m2) {

return \*m1 < \*m2;

}

};

//cout << "AFTER SORT" << endl;

struct comp functor;

table.sort(functor);

//for (std::list<Model\*>::iterator it=table.begin(); it!=table.end(); it++) cout << \*\*it;

grow(table);

//cout << "AFTER GROW" << endl;

//\_root->traverse();

generate();

//cout << "AFTER GENERATE" << endl;

//for (std::map<char,string>::iterator it=codes.begin(); it!=codes.end(); it++)

// cout << it->first << " => " << it->second << endl;

for (string::iterator it=longUrl.begin(); it!=longUrl.end(); it++) {

char c = \*it;

string code = codes[c];

encoded.append(code);

}

return encoded;

}

// Decodes a shortened URL to its original URL.

string decode(string shortUrl) {

string decoded;

string str;

for (string::iterator it=shortUrl.begin(); it!=shortUrl.end(); it++) {

char c = \*it;

str += c;

for (std::map<char,string>::iterator it=codes.begin(); it!=codes.end(); it++) {

if (!str.compare(it->second)) {

char c = it->first;

decoded.insert(decoded.end(), c);

str.clear();

break;

}

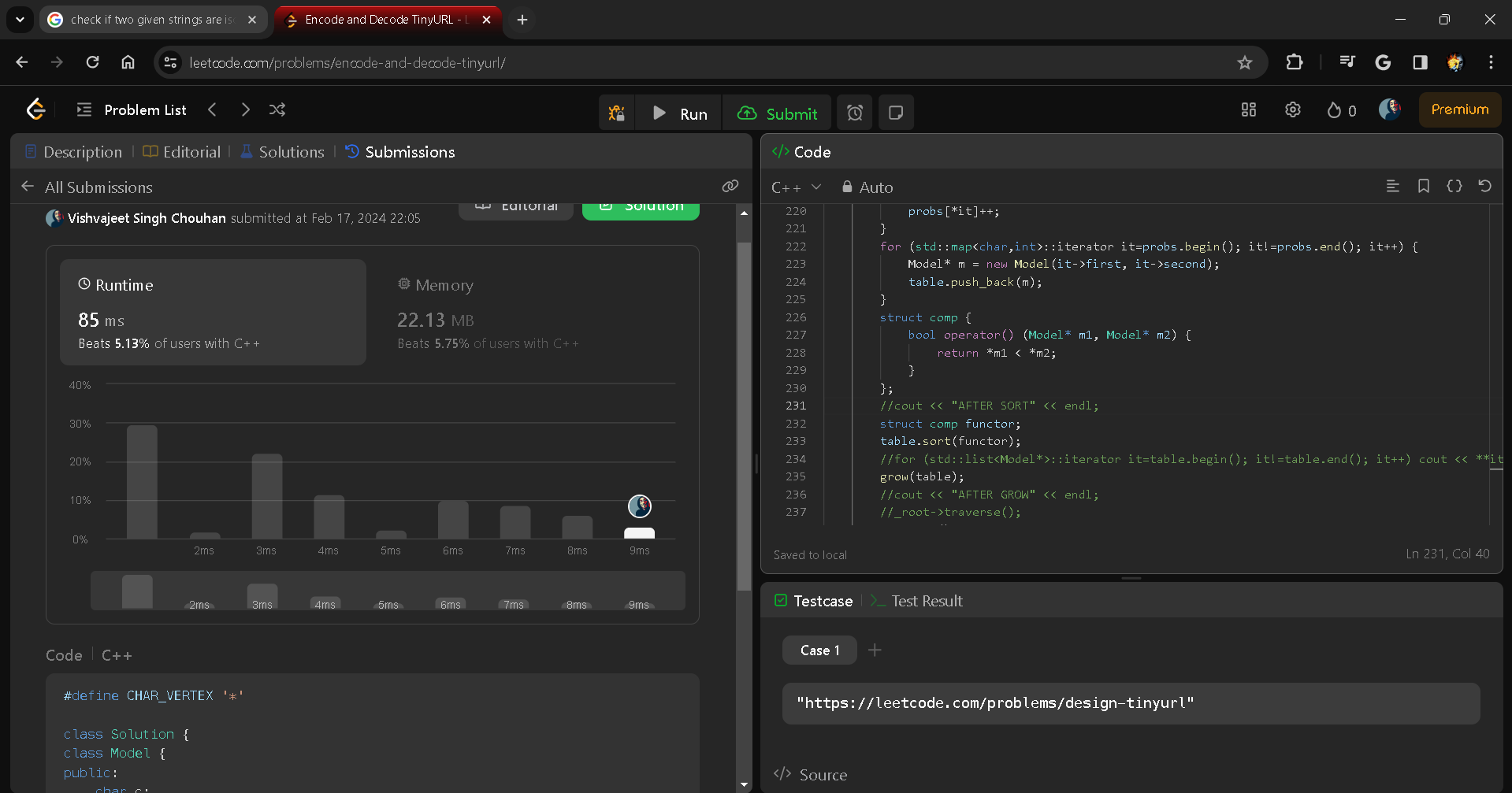
}

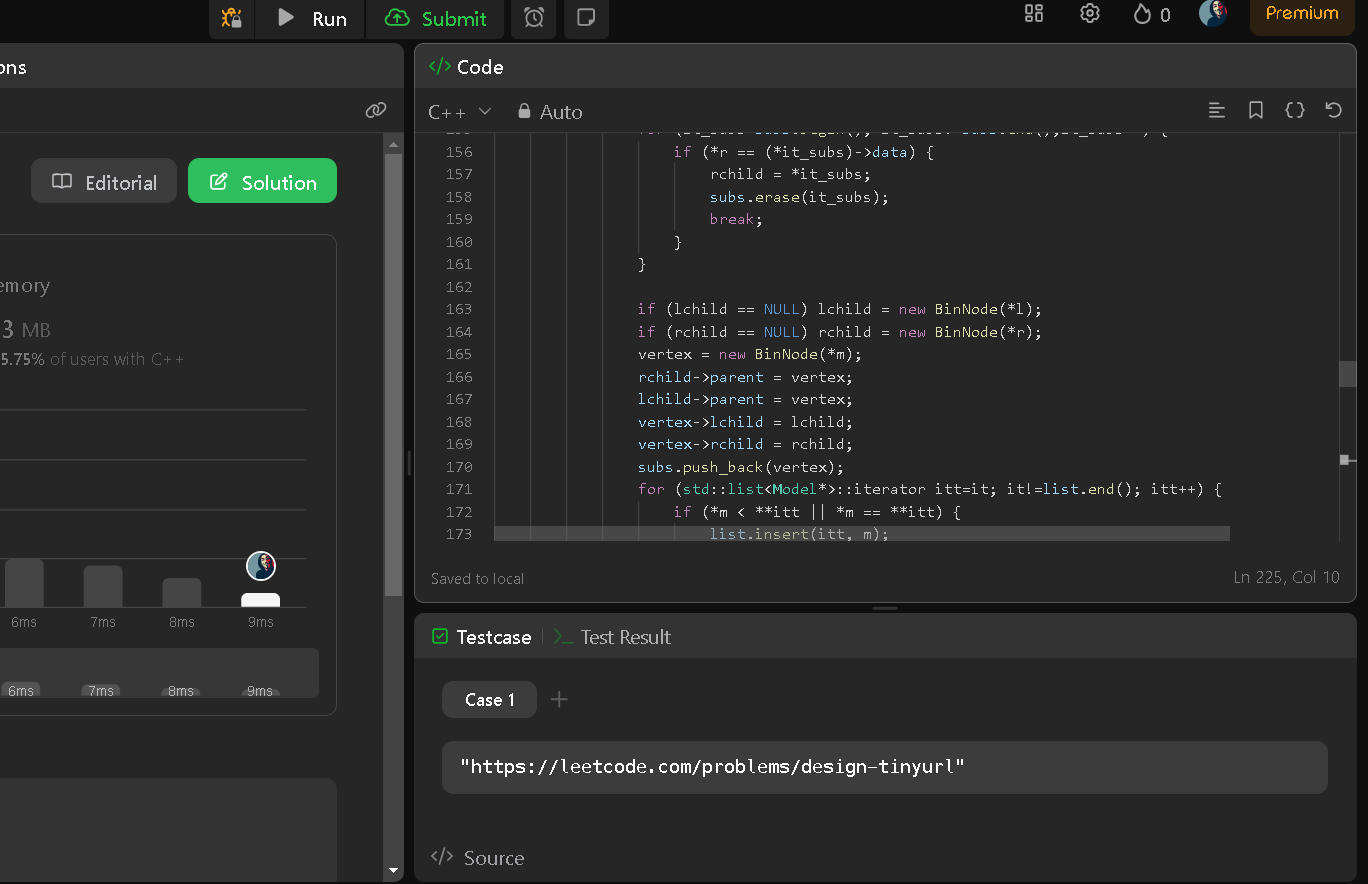
}

return decoded;

}

};





Q2 [Fractional Knapsack Problem](https://practice.geeksforgeeks.org/problems/fractional-knapsack/0)

int maximumUnits(std::vector<std::vector<int>>& arr, int k) {

int n = arr.size();

// Sort array according to the 2nd element using custom comparison function

std::sort(arr.begin(), arr.end(), [](const std::vector<int>& a, const std::vector<int>& b) {

return b[1] < a[1];

});

int i = 0, ans = 0;

while (k > 0 && i < n) {

// Check the limit of the truck & keeping boxes

int t = std::min(arr[i][0], k);

k -= t;

ans += t \* arr[i][1];

i++;

}

return ans;

}

A screenshot of a computer

Description automatically generated

Q3 [Greedy Algorithm to find Minimum number of Coins](https://practice.geeksforgeeks.org/problems/coin-piles/0)

int coinChange(vector<int>& coins, int amount) {

vector<int> s(amount + 1, amount + 1);

s[0] = 0;

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

for (int j = 0; j < coins.size(); j++) {

if (i >= coins[j]) {

s[i] = min(s[i - coins[j]] + 1, s[i]);

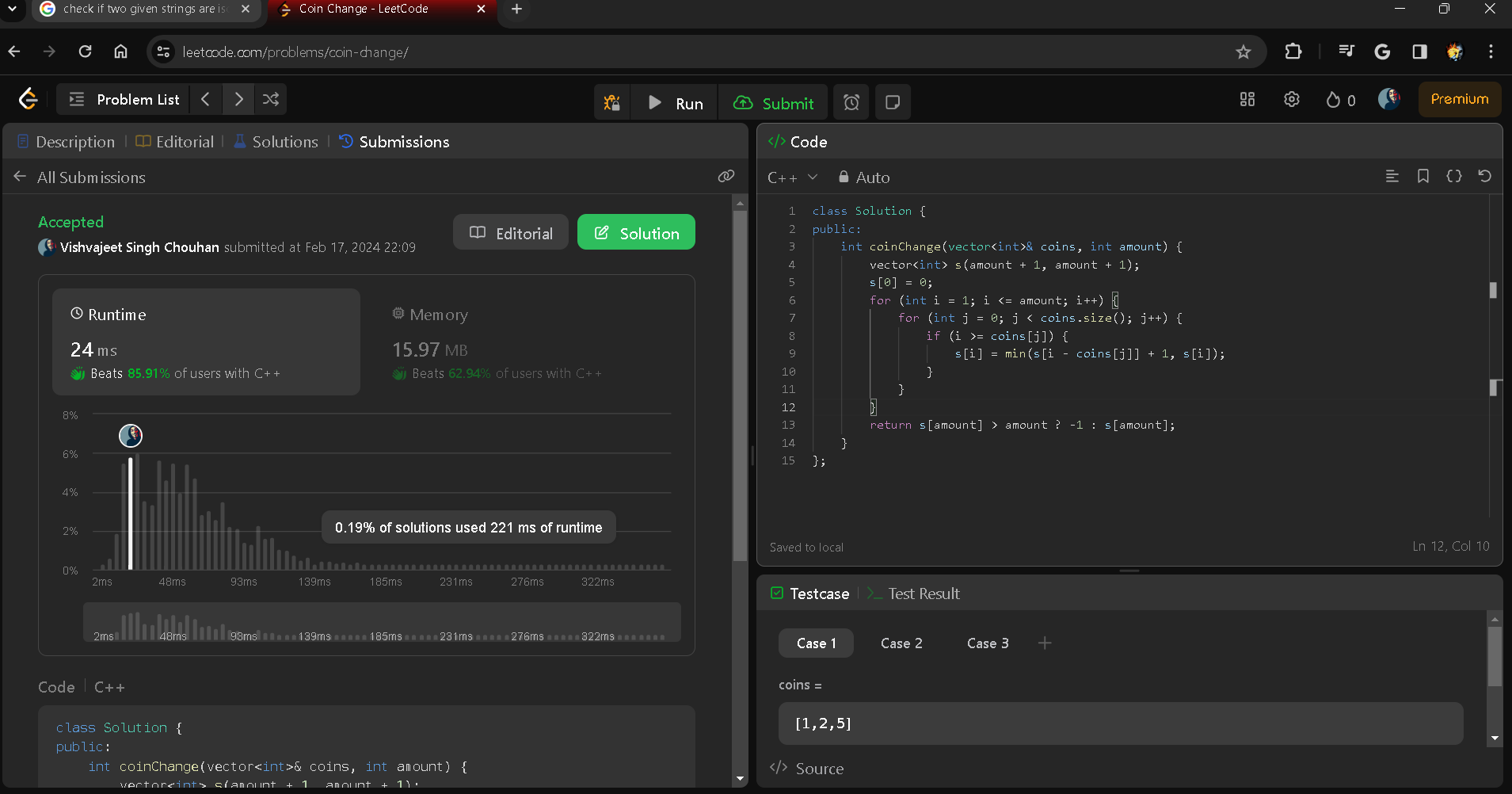
}

}

}

return s[amount] > amount ? -1 : s[amount];

}



Q4 [K Centers Problem](https://www.geeksforgeeks.org/k-centers-problem-set-1-greedy-approximate-algorithm/)

vector<vector<int>> kClosest(vector<vector<int>>& points, int k) {

vector<vector<int>> ans;

vector<pair<int, vector<int>>> v;

for(int i = 0; i < points.size(); i++){

int s = points[i][0] \* points[i][0] + points[i][1] \* points[i][1];

v.push\_back({s, {points[i][0],points[i][1]}});

}

sort(v.begin(), v.end());

int j = 0;

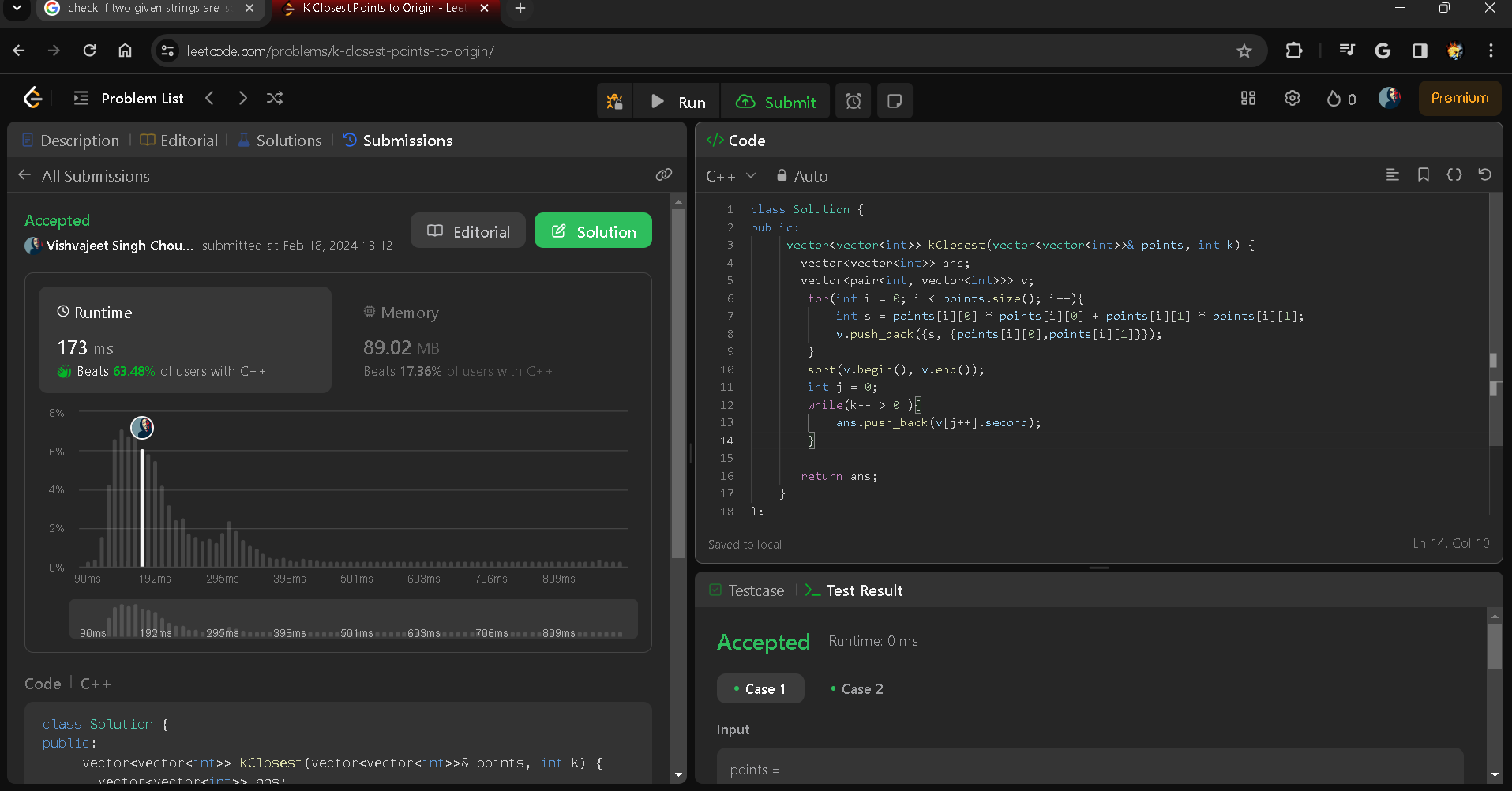
while(k-- > 0 ){

ans.push\_back(v[j++].second);

}

return ans;

}



Q5 [Word Break Problem using Backtracking](https://practice.geeksforgeeks.org/problems/word-break-part-2/0)

#include <vector>

#include <string>

#include <unordered\_set>

class Solution {

public:

vector<string> wordBreak(int n, vector<string>& dict, string s) {

unordered\_set<string> wordSet(dict.begin(), dict.end());

vector<string> result;

string current;

wordBreakHelper(s, 0, wordSet, current, result);

return result;

}

private:

void wordBreakHelper(const string& s, int start, const unordered\_set<string>& wordSet, string& current, vector<string>& result) {

if (start == s.length()) {

// We reached the end of the string, add the current word break to the result

result.push\_back(current);

return;

}

for (int end = start + 1; end <= s.length(); ++end) {

string word = s.substr(start, end - start);

if (wordSet.find(word) != wordSet.end()) {

// Word is in the dictionary, explore further

string previous = current.empty() ? word : (current + " " + word);

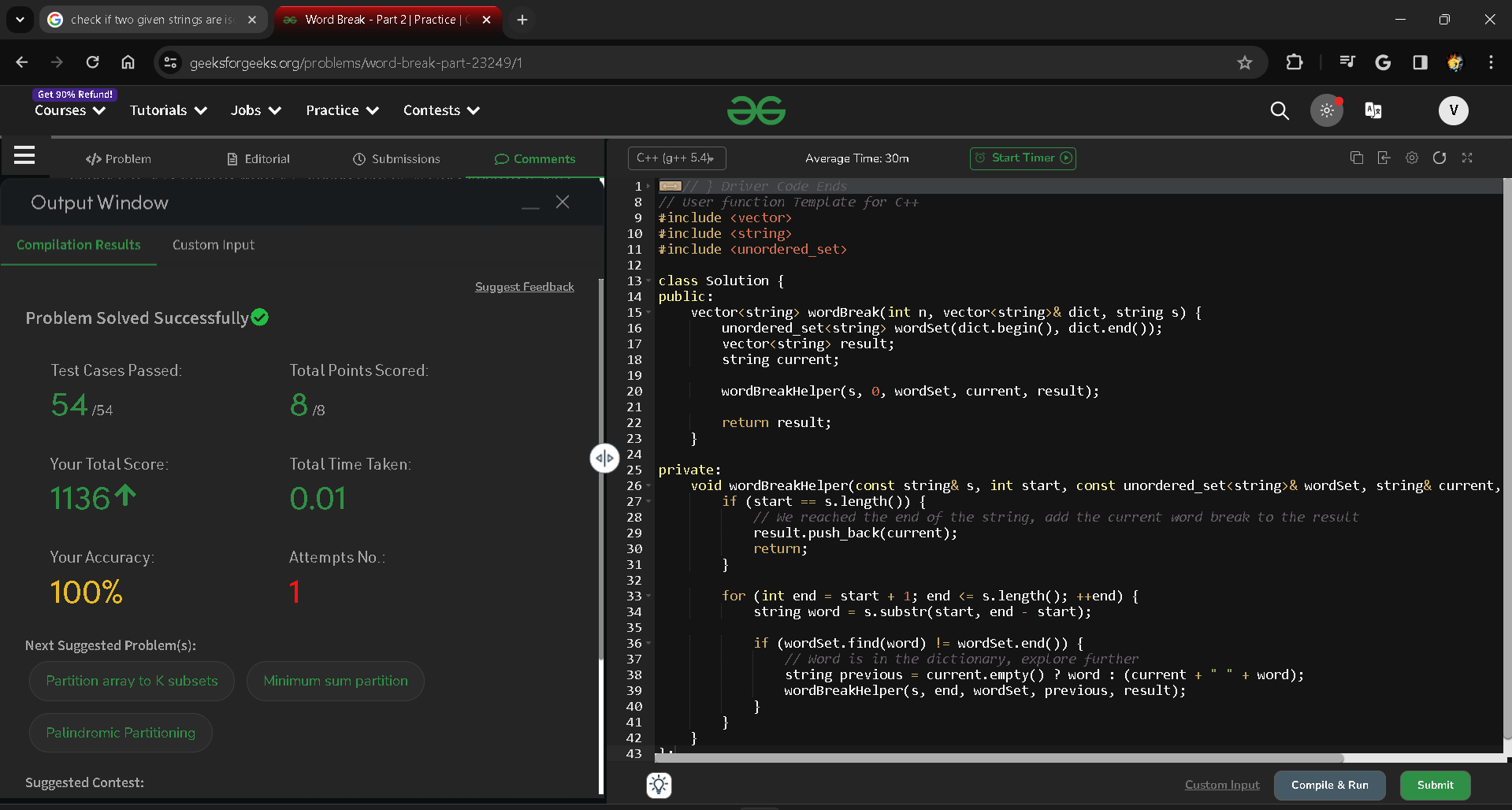
wordBreakHelper(s, end, wordSet, previous, result);

}

}

}

};



Q6 [Sudoku Solver](https://practice.geeksforgeeks.org/problems/solve-the-sudoku/0)

bool isSafe(int row, int col, vector<vector<char>>& board, int value){

int n = board.size();

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

// row check

if(board[row][i] == value) return false;

//col check

if(board[i][col] == value) return false;

//#\*3 matrix check

if(board[3\*(row/3)+(i/3)][3\*(col/3)+(i%3)]== value)

return false;

}

return true;

}

bool solve(vector<vector<char>>& board){

int n = board.size();

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

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

//check empty spaces

if(board[i][j] == '.'){

//filling value form 1 to 9

for(char value = '1'; value<= '9'; value++){

//check for safety

if(isSafe(i,j,board,value)){

board[i][j] = value;

// recursion call

bool aageKaSolution = solve(board);

if(aageKaSolution == true){

return true;

}

else{

//backtrack

board[i][j]='.';

}

}

}

return false;

}

}

}

//all cells are filled

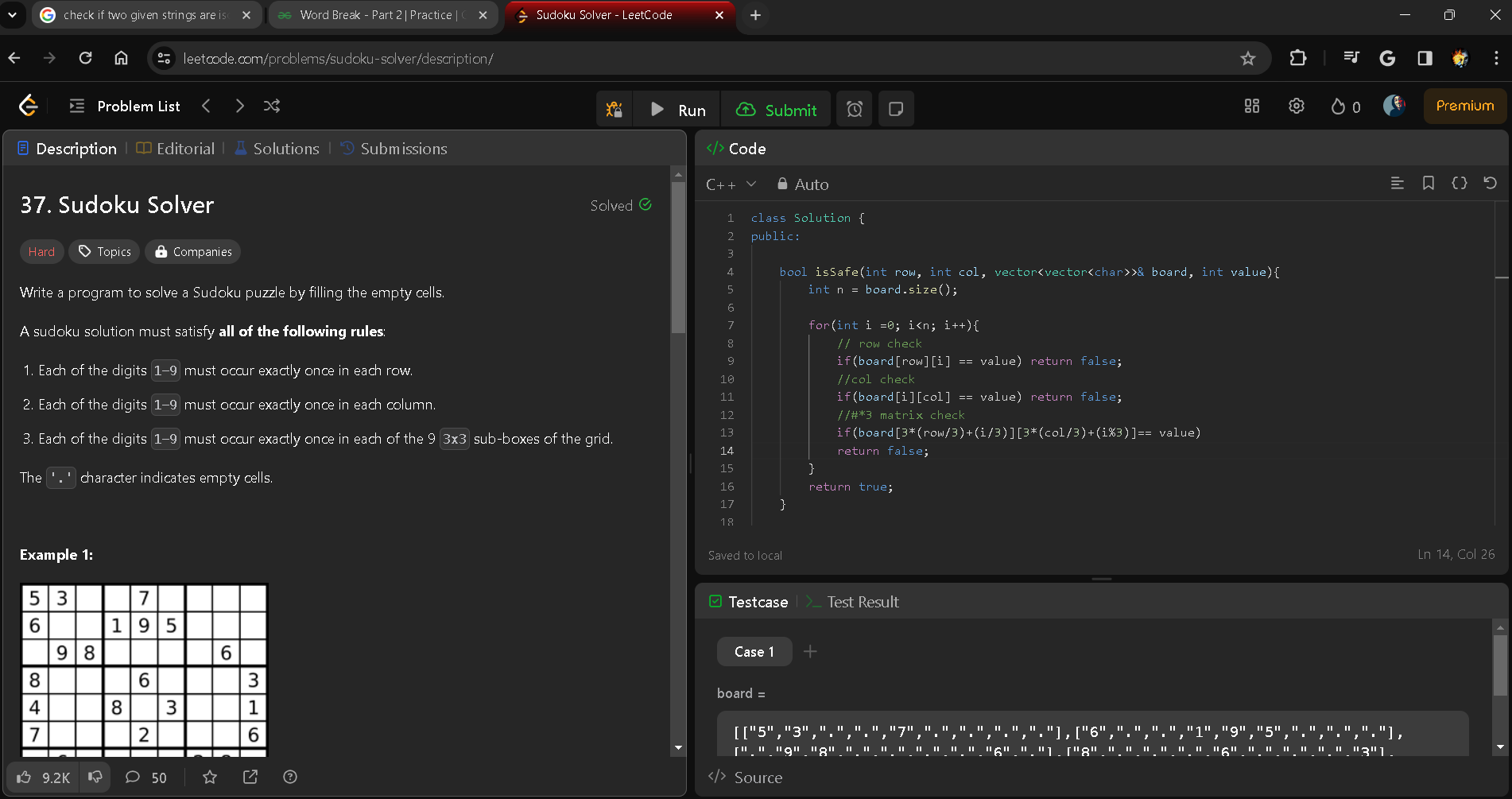
return true;

}

void solveSudoku(vector<vector<char>>& board) {

solve(board);

}



Q7 [The Knight’s tour problem](https://www.geeksforgeeks.org/the-knights-tour-problem-backtracking-1/)

bool checkValidGrid(vector<vector<int>>& grid) {

int n=grid.size();

if(grid[0][0]!=0) return false;

bool fl=true;

int x=0,y=0;

for(int i=0;i<n\*n;i++)

{

if( x+2<n && y+1<n && grid[x+2][y+1]==i+1 )

{

x=x+2;

y=y+1;

}

else if(x+1<n && y+2<n && grid[x+1][y+2]==i+1 )

{

x=x+1;

y=y+2;

}

else if(x+2<n && y-1>=0 && grid[x+2][y-1]==i+1 )

{

y=y-1;

x=x+2;

}

else if(x+1<n && y-2>=0 && grid[x+1][y-2]==i+1)

{

x=x+1;

y=y-2;

}

else if(x-2>=0 && y-1>=0 && grid[x-2][y-1]==i+1 )

{

x=x-2;

y=y-1;

}

else if(x-2>=0 && y+1<n && grid[x-2][y+1]==i +1)

{

y=y+1;

x=x-2;

}

else if(x-1>=0 && y-2>=0 && grid[x-1][y-2]==i+1 )

{

x=x-1;

y=y-2;

}

else if(x-1>=0 && y+2<n && grid[x-1][y+2]==i +1 )

{

x=x-1;

y=y+2;

}

else if(i==(n\*n-1))

{

break;

return true;

}

else

{

return false;

}

}return true; A screenshot of a computer

Description automatically generated }

Q8 [Evaluation of Postfix expression](https://practice.geeksforgeeks.org/problems/evaluation-of-postfix-expression/0)

boolean isOperator(String op){

return (op.equals("+") || op.equals("-") || op.equals("\*") || op.equals("/"));

}

public int evalRPN(String[] A) {

int len=A.length;

Stack<Integer> st = new Stack<>();

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

if(!st.isEmpty() && isOperator(A[i])){

int val2=st.pop();

int val1=st.pop();

int x=0;

if(A[i].equals("+")){

x = val1 + val2;

}

else if(A[i].equals("-")){

x = val1 - val2;

}

else if(A[i].equals("\*")){

x = val1 \* val2;

}

else if(A[i].equals("/")){

x = val1 / val2;

}

st.push(x);//push calculated ans back in stack

}

else{

st.push(Integer.parseInt(A[i]));

}

}

return st.peek();

}

A screenshot of a computer

Description automatically generated