**Not All Questions** 

Abbreviation Using Backtracking

Medium

1. You are given a word.

2. You have to generate all abbrevations of that word.

Use recursion as suggested in question video

**Constraints**

1 <= length of string <= 32

**Format**

**Input**

A string representing a word

**Output**

Check the sample ouput and question video.

**Example**

**Sample Input**

pep

**Sample Output**

pep

pe1

p1p

p2

1ep

1e1

2p

3

#include<iostream>

#include<string>

#include<ctype.h>

using namespace std;

void solution(string str, string asf,int count, int pos)

{

// write your code here

//MY SOLUTION DON'T USE COUNT ANS POS

if(str.length() == 0) {

cout<<asf<<endl;

return;

}

char fc = str[0];

string ros = str.substr(1,str.length()-1);

solution (ros,asf+fc,count,pos);

if(asf.length() == 0 || isalpha(asf[asf.length()-1])) {

solution (ros,asf+"1",count,pos);

}else if(isdigit(asf[asf.length()-1])) {

int fi = asf[asf.length()-1] - '0';

fi++;

string nasf = asf.substr(0,asf.length()-1);

nasf = nasf+ to\_string(fi);

solution (ros,nasf,count,pos);

}

}

int main() {

string str;

cin >> str;

solution(str,"",0,0);

return 0;

}

N Queens - Branch And Bound

Hard

1. You are given a number n, the size of a chess board.

2. You are required to place n number of queens in the n \* n cells of board such that no queen can

kill another.

Note - Queens kill at distance in all 8 directions

3. Complete the body of printNQueens function - without changing signature - to calculate and

print all safe configurations of n-queens

Use sample input and output to get more idea.

Note -> The online judge can't force you to write the function recursively but that is what the spirit

of question is.

Write recursive and not iterative logic. The purpose of the question is to aid learning recursion, branch and bound technique and not test you.

**Constraints**

1 <= n <= 10

**Format**

**Input**

A number n

**Output**

Safe configurations of queens as suggested in sample output

**Example**

**Sample Input**

4

**Sample Output**

0-1, 1-3, 2-0, 3-2, .

0-2, 1-0, 2-3, 3-1, .

#include<iostream>

#include<vector>

#include<string>

using namespace std;

void solve(vector<vector<bool>> board ,int row,vector<bool> & coloum,vector<bool> &dia,

vector<bool> &revdia,string asf) {

if(row == board.size()){

cout<<asf<<"."<<endl;

return;

}

for(int col{}; col < board.size(); col++) {

if(coloum[col] == false && dia[row+col] == false&& revdia[row-col+board.size() -1] == false) {

board[row][col] = true;

coloum[col] = true;

dia[row+ col] = true;

revdia[row-col+board.size() -1] = true;

solve(board, row+1, coloum,dia,revdia,asf+to\_string(row)+"-"+to\_string(col)+", ");

board[row][col] = false;

coloum[col] = false;

dia[row+ col] = false;

revdia[row-col+board.size() -1] = false;

}

}

}

int main() {

int n{};

cin>>n;

vector<vector<bool>> board (n, vector <bool>(n));

//for blocking the positions for the queen

vector<bool> coloum(n,false);

vector<bool> dia(2\*n-1,false);// diagonal (down + right) , represents -> row + coloum

vector<bool> revdia(2\*n-1,false);//reverse digonal (down + left), represents -> row - coloum + (boardlength-1)

solve(board, 0, coloum,dia,revdia,"");

}

Max Score

Hard

1. You are given a list of words, a list of alphabets(might be repeating) and score of every alphabet

from a to z.

2. You have to find the maximum score of any valid set of words formed by using the given

alphabets.

3. A word can not be used more than one time.

4. Each alphabet can be used only once.

5. It is not necessary to use all the given alphabets.

Note -> Check out the question video and write the recursive code as it is intended without

changing signature. The judge can't force you but intends you to teach a concept.

**Constraints**

1 <= N <= 14

1 <= length of word <= 15

1 <= M <= 100

1 <= Value of score <= 10

**Format**

**Input**

A number N representing number of words

N space separated strings

A number M representing number of alphabets(might be repeating)

M space separated characters

26 numbers representing score of unique alphabets from a to z.

**Output**

Check the sample ouput and question video.

**Example**

**Sample Input**

4

dog cat dad good

9

a b c d d d g o o

1 0 9 5 0 0 3 0 0 0 0 0 0 0 2 0 0 0 0 0 0 0 0 0 0 0

**Sample Output**

23

#include <iostream>

#include <climits>

#include <cmath>

#include <unordered\_map>

#include <vector>

using namespace std;

//Pep coding solution is on leet code (in my submission)

void solve(int &max, vector<string> &words, unordered\_map<char, int> &freq,unordered\_map<char, int> asf,

unordered\_map<char, int> &scores) {

if( words.size() == 0) {

int thisScore{};

for(pair<char,int> p:asf){

thisScore += scores[p.first] \* p.second;

}

if(thisScore > max) {

max = thisScore;

}

return;

}

string wr = words[words.size()-1];

words.erase(words.begin()+words.size()-1);

solve(max,words,freq,asf,scores);//not adding

//for adding

for(char c: wr){

asf[c]++;

}

//for adding checking if adding doesn't exeeceds the limits

bool can {true};

for(char c: wr){

if(freq.find(c) == freq.end()||asf[c] > freq[c]){

can = false;

break;

}

}

if(can == true) {

solve(max,words,freq,asf,scores);

}

words.push\_back(wr);

return;

}

int main() {

int now{}; // number of words

cin>>now;

vector<string> words(now);

for (int i{}; i < now; i++) {

cin >> words[i];

}

// cout<<"check1"<<endl;

unordered\_map<char, int> freq;//this is the limit

int n{};

cin>>n;

char c{};

for (int i{}; i < n; i++) {

cin >> c;

freq[c]++;

}

// cout<<"checkwe"<<endl;

unordered\_map<char, int> scores;//scores of all individual characters

char sc{'a'};

int s{};

for (int i{}; i < 26; i++) {

cin >> s;

scores[sc] = s;

sc++;

}

// cout<<"check2"<<endl;

unordered\_map<char, int> asf;

int max = INT\_MIN;

// cout<<"check52"<<endl;

solve(max, words, freq, asf,scores);

// cout<<"chec234"<<endl;

cout << max << endl;

}

Josephus Problem

Easy

1. You are given two numbers N and K.

2. N represents the total number of soldiers standing in a circle having position marked from 0 to N-1.

3. A cruel king wants to execute them but in a different way.

4. He starts executing soldiers from 0th position and proceeds around the circle in clockwise direction.

5. In each step, k-1 soldiers are skipped and the k-th soldier is executed.

6. The elimination proceeds around the circle (which is becoming smaller and smaller as the executed soldiers are removed), until only the last soldier remains, who is given freedom.

7. You have to find the position of that lucky soldier.

Note -> Check out the question video and write the recursive code as it is intended without changing signature. The judge can't force you but intends you to teach a concept.

**Constraints**

1 <= N,K <= 200

**Format**

**Input**

2 numbers N and K

**Output**

Check the sample ouput and question video.

**Example**

**Sample Input**

4

2

**Sample Output**

0

#include<iostream>

#include<vector>

using namespace std;

// int solve (int index, int k,vector <int> &cir) {

// if(cir.size() == 1){

// return cir[0]+ 1;

// }

// // auto itr = cir.begin() + index;

// // cout<<index<<" "<<\*itr<<"[ ";

// cir.erase(cir.begin() + index);

// for(int a: cir) {

// cout<<a<<" ";

// }

// cout<<"]"<<endl;

// return solve( (index+k-1) % cir.size(),k, cir);

// }

int effSol(int n , int k) {

if( n== 1){

return 0;

}

int x = effSol(n-1, k);

int y = (x + k) % n;

return y;

}

int main() {

int n{};

int k{};

cin>>n>>k;

vector<int> cir(n);

for(int i{}; i< n;i++) {

cir[i] = i;

}

// int a = (k-1)%n;

// int ans = solve (a,k,cir);

int ans = effSol(n,k);

//here person starts from 0 if they starts from 1 just add 1 to final solution

cout<<ans<<endl;

}

Lexicographical Numbers

Medium

1. You are given a number.

2. You have to print all the numbers from 1 to n in lexicographical order.

Note -> Check out the question video and write the recursive code as it is intended without

changing signature. The judge can't force you but intends you to teach a concept.

**Constraints**

1 <= n <= 50000

**Format**

**Input**

A number

**Output**

Check the sample output and question video

**Example**

**Sample Input**

14

**Sample Output**

1

10

11

12

13

14

2

3

4

5

6

7

8

9

#include<iostream>

using namespace std;

void dfs (int i, int n){

if( i > n){

return ;

}

cout<<i<<endl;

for(int c{}; c<10 ;c++){

dfs((i\*10)+ c,n);

}

return ;

}

int main()

{

int n;

cin>>n;

//write your code here

for(int i{1} ; i< 10; i++) {

dfs(i, n);

}

}

Gold Mine - 2

Easy

1. You are given a number n, representing the number of rows.

2. You are given a number m, representing the number of columns.

3. You are given n\*m numbers, representing elements of 2d array a, which represents a gold mine.

4. You are allowed to take one step left, right, up or down from your current position.

5. You can't visit a cell with 0 gold and the same cell more than once.

6. Each cell has a value that is the amount of gold available in the cell.

7. You are required to identify the maximum amount of gold that can be dug out from the mine if

you start and stop collecting gold from any position in the grid that has some gold.

Note -> Check out the question video and write the recursive code as it is intended without

changing signature. The judge can't force you but intends you to teach a concept.

**Constraints**

1 <= n <= 10

1 <= m <= 10

0 <= e1, e2, .. n \* m elements <= 1000

**Format**

**Input**

A number n

A number m

e11

e12..

e12

e22..

m\*n numbers

**Output**

Maximum gold collected

**Example**

**Sample Input**

6

6

0 1 4 2 8 2

4 3 6 5 0 4

1 2 4 1 4 6

2 0 7 3 2 2

3 1 5 9 2 4

2 7 0 8 5 1

**Sample Output**

120

//6 out of 10 points on dados

#include <iostream>

#include <vector>

#include <climits>

using namespace std;

int digAndCollectedInThisComponent(int i, int j,vector<vector<int>> &goldMine,vector<vector<bool>> &visited){

if(i<0||j<0||i == goldMine.size() || j == goldMine[0].size()||visited[i][j] == true ||goldMine[i][j] == 0) { //visited and goldMine have same size

return 0;

}

int thisComp = goldMine[i][j];

visited[i][j] = true;

thisComp += digAndCollectedInThisComponent(i, j+1,goldMine,visited);//right

thisComp += digAndCollectedInThisComponent(i+1, j,goldMine,visited);//down

thisComp += digAndCollectedInThisComponent(i, j-1,goldMine,visited);//left

thisComp += digAndCollectedInThisComponent(i-1, j,goldMine,visited);//up

return thisComp;

}

int main(){

int n{};

int m{};

cin>> n>> m;

vector<vector<int>> goldMine (n,vector<int>(m));

for(int i{}; i < n; i++) {

for(int j{}; j < m ; j++) {

cin>>goldMine[i][j];

}

}

vector<vector<bool>> visited (n,vector<bool>(m,false));

int maxAns = INT\_MIN;

int thisComp {};

for(int i{}; i < n; i++) {

for(int j{}; j < m ; j++) {

if(visited[i][j] == false && goldMine[i][j]) {

thisComp = digAndCollectedInThisComponent(i, j,goldMine,visited);

}

if(thisComp > maxAns) {

maxAns = thisComp;

}

}

}

cout<<maxAns<<endl;

}

Solve Sudoku

Hard

1. You are give a partially filled 9\*9 2-D array(arr) which represents an incomplete sudoku state.

2. You are required to assign the digits from 1 to 9 to the empty cells following some rules.

Rule 1 -> Digits from 1-9 must occur exactly once in each row.

Rule 2 -> Digits from 1-9 must occur exactly once in each column.

Rule 3 -> Digits from 1-9 must occur exactly once in each 3x3 sub-array of the given 2D array.

Assumption -> The given Sudoku puzzle will have a single unique solution.

**Constraints**

0 <= arr[i][j] <= 9

**Format**

**Input**

9\*9 integers ranging from 1 to 9.

0 represents an empty cell.

**Output**

You have to print the solved sudoku.

**Example**

**Sample Input**

3 0 6 5 0 8 4 0 0

5 2 0 0 0 0 0 0 0

0 8 7 0 0 0 0 3 1

0 0 3 0 1 0 0 8 0

9 0 0 8 6 3 0 0 5

0 5 0 0 9 0 6 0 0

1 3 0 0 0 0 2 5 0

0 0 0 0 0 0 0 7 4

0 0 5 2 0 6 3 0 0

**Sample Output**

3 1 6 5 7 8 4 9 2

5 2 9 1 3 4 7 6 8

4 8 7 6 2 9 5 3 1

2 6 3 4 1 5 9 8 7

9 7 4 8 6 3 1 2 5

8 5 1 7 9 2 6 4 3

1 3 8 9 4 7 2 5 6

6 9 2 3 5 1 8 7 4

7 4 5 2 8 6 3 1 9

#include<iostream>

#include<vector>

using namespace std;

bool isvalid(int i, int j, int opt ,vector<vector<int>> &board) {

for(int col {}; col<9;col++) {

if(board[i][col] == opt){

return false;

}

}

for(int row {}; row<9;row++) {

if(board[row][j] == opt){

return false;

}

}

//now in the box

int sr = (i/3) \* 3;

// int er = (((i/3)+1) \* 3)-1;

int sc = (j/3) \* 3;

// int ec = (((j/3)+1) \* 3)-1;

for(int row = sr; row < sr+3;row++){

for(int col = sc; col < sc + 3;col++){

if(board[row][col] == opt){

return false;

}

}

}

return true;

}

void solve( vector<vector<int>> &board,int i , int j) {

if(i == 9) {

for(int i{}; i< 9;i++) {

for(int j{}; j< 9; j++) {

cout<<board[i][j]<<" ";

}

cout<<endl;

}

return ;

}

// cout<<i<<" "<<j<<endl;

int ni{};

int nj{};

if(j == 8){

ni = i+1;

nj = 0;

}else{

ni = i;

nj = j+1;

}

if(board[i][j] != 0) {

solve(board,ni,nj);

}else{

for(int opt{1}; opt <= 9;opt++){ //options

if(isvalid(i,j,opt, board)) {

board[i][j] = opt;

solve(board,ni,nj);

board[i][j] = 0;

}

}

}

}

int main() {

vector<vector<int>> board(9,vector<int>(9));

for(int i{}; i< 9;i++) {

for(int j{}; j< 9; j++) {

cin>> board[i][j];

}

}

solve(board , 0,0);

}

Crossword Puzzle

Easy

1. You are given a 10\*10 2-D array(arr) containing only '+' and '-' characters, which represents a

crossword puzzle.

2. You are also given n number of words which need to be filled into the crossword.

3. Cells containing '-' are to be filled with the given words.

Note -> Check out the question video and write the recursive code as it is intended without

changing signature. The judge can't force you but intends you to teach a concept.

**Constraints**

1 <= n <= 10

**Format**

**Input**

10 \* 10 grid of characters containing only '+' and '-'

A number n

str1

str2

...n strings

**Output**

Check the sample output and question video

**Example**

**Sample Input**

+-++++++++

+-++++++++

+-++++++++

+-----++++

+-+++-++++

+-+++-++++

+++++-++++

++------++

+++++-++++

+++++-++++

4

LONDON

DELHI

ICELAND

ANKARA

**Sample Output**

+L++++++++

+O++++++++

+N++++++++

+DELHI++++

+O+++C++++

+N+++E++++

+++++L++++

++ANKARA++

+++++N++++

+++++D++++

#include <iostream>

#include <string>

#include <vector>

using namespace std;

/\*

//my solution

//only one solution is discovered

//see previous submission in which i did with all four direction allowed

// a part of this question is on leet code (2018 problem number,medium)

bool placeWord(vector<vector<char>>& board ,int i,int j , string word,char d,

int n){ if(i <0||j<0||i>=board.size()||j>=board[0].size()){ if(word.length() ==

0){ return true; }else{ return false;

}

}

if(board[i][j] == '+' && word.size() == 0){

return true;

}

if(board[i][j] == '+'&&word.size() < n){

return false;

}

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

if(word.length()== 0){

return false;

}

char f = word[0];

board[i][j] = f;

string row = word.substr(1,word.length()-1);

bool gg{false};

if(d=='r'){

gg = placeWord(board,i,j+1,row,d,word.length());

}else if(d=='d'){

gg = placeWord(board,i+1,j,row,d,word.length());

}else if(d=='l'){

gg = placeWord(board,i,j-1,row,d,word.length());

}else if(d=='u'){

gg = placeWord(board,i-1,j,row,d,word.length());

}

if(gg){

return true;

}

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

return false;

}else{

if(board[i][j] != word[0]){

return false;

}else{

cout<<word<<endl;

string row = word.substr(1,word.length()-1);

bool gg{false};

if(d =='r'){

gg = placeWord(board,i,j+1,row,d,word.length());

}else if(d =='d'){

gg = placeWord(board,i+1,j,row,d,word.length());

}else if(d =='l'){

gg = placeWord(board,i,j-1,row,d,word.length());

}else if(d =='u'){

gg = placeWord(board,i-1,j,row,d,word.length());

}

if(gg){

return true;

}

}

}

return false;

}

bool placeWordInCrossword(vector<vector<char>>& board, string word) {

int n = board.size();

int m = board[0].size();

int l = word.length();

for(int i{};i< n; i++){

for(int j{}; j< m; j++) {

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

continue;

}

if(j-1 <0||board[i][j-1] == '+'){

if(placeWord(board,i,j,word,'r',l)){

return true;

}

}

if(i-1 <0||board[i-1][j] == '+'){

if(placeWord(board,i,j,word,'d',l)){

return true;

}

}

if(j+1 >=board[0].size()||board[i][j+1] == '+'){

if(placeWord(board,i,j,word,'l',l)){

return true;

}

}

if(i+1 >=board.size()||board[i+1][j] == '+'){

if(placeWord(board,i,j,word,'u',l)){

return true;

}

}

}

}

return false;

}

void solve(vector<vector<char>> &board,vector<string>& words,int i){

while(i<= words.size()){

placeWordInCrossword(board,words[i]);

i++;

}

for(auto v: board){

for(char c: v) {

cout<<c<<" ";

}

cout<<endl;

}

\*/

bool canPlaceHorizontally(vector<vector<char>> &board, string word, int i,

int j) {

// cout<<word<<endl;

if (j - 1 >= 0 && board[i][j - 1] != '+') {

return false;

}

if (j + word.length() < board[0].size() &&

board[i][j + word.length()] != '+') {

return false;

}

for (int jj{}; jj < word.length(); jj++) {

if (j + jj >= board[0].size()) {

return false;

}

if (board[i][j + jj] == '-' || board[i][j + jj] == word[jj]) {

continue;

} else {

return false;

}

}

return true;

}

bool canPlaceVertically(vector<vector<char>> &board, string word, int i,

int j) {

// cout<<word<<endl;

if (i - 1 >= 0 && board[i - 1][j] != '+') {

return false;

}

if (i + word.length() < board.size() &&

board[i + word.length()][j] != '+') {

return false;

}

for (int ii{}; ii < word.length(); ii++) {

if (i + ii >= board.size()) {

return false;

}

if (board[i + ii][j] == '-' || board[i + ii][j] == word[ii]) {

continue;

} else {

return false;

}

}

return true;

}

vector<bool> placeHorizontally(vector<vector<char>> &board, string word, int i,

int j) {

// cout<<word<<endl;

vector<bool> wePlaced(word.length(),

false); // you placed it or already it was there

for (int a{}; a < word.length(); a++) {

if (board[i][j + a] == '-') {

board[i][j + a] = word[a];

wePlaced[a] = true;

}

}

return wePlaced;

}

vector<bool> placeVertically(vector<vector<char>> &board, string word, int i,

int j) {

// cout<<word<<endl;

vector<bool> wePlaced(word.length(),

false); // you placed it or already it was there

for (int a{}; a < word.length(); a++) {

if (board[i + a][j] == '-') {

board[i + a][j] = word[a];

wePlaced[a] = true;

}

}

return wePlaced;

}

void unplaceWordHorizontally(vector<vector<char>> &board, string word, int i,

int j, vector<bool> wePlaced) {

// cout<<word<<endl;

for (int a{}; a < word.length(); a++) {

if (wePlaced[a]) {

board[i][j + a] = '-';

}

}

}

void unplaceWordVertically(vector<vector<char>> &board, string word, int i,

int j, vector<bool> wePlaced) {

// cout<<word<<endl;

for (int a{}; a < word.length(); a++) {

if (wePlaced[a]) {

board[i + a][j] = '-';

}

}

}

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

for (auto v : board) {

for (char c : v) {

cout << c;

}

cout << endl;

}

return;

}

void solve(vector<vector<char>> &board, vector<string> &words, int wi) {

if (wi == words.size()) {

printIt(board);

return;

}

string word = words[wi];

for (int i{}; i < board.size(); i++) {

for (int j{}; j < board[0].size(); j++) {

// cout<<i<<j<<endl;

if (board[i][j] == '-' || board[i][j] == word[0]) {

if (canPlaceHorizontally(board, word, i, j)) {

vector<bool> wePlaced = placeHorizontally(board, word, i, j);

solve(board, words, wi + 1);

unplaceWordHorizontally(board, word, i, j, wePlaced);

}

if (canPlaceVertically(board, word, i, j)) {

vector<bool> wePlaced = placeVertically(board, word, i, j);

solve(board, words, wi + 1);

unplaceWordVertically(board, word, i, j, wePlaced);

}

}

}

}

}

int main() {

vector<vector<char>> board(10, vector<char>(10));

for (int i{}; i < 10; i++) {

for (int j{}; j < 10; j++) {

cin >> board[i][j];

}

}

int numOfWords{};

cin >> numOfWords;

vector<string> words(numOfWords);

for (int i{}; i < numOfWords; i++) {

cin >> words[i];

}

solve(board, words, 0);

}

Cryptarithmetic

Easy

1. You are given three strings s1, s2 and s3.

2. First two are supposed to add and form third. s1 + s2 = s3

3. You have to map each individual character to a digit, so that the above equation holds true.

Note -> Check out the question video and write the recursive code as it is intended without

changing the signature. The judge can't force you but intends you to teach a concept.

**Constraints**

1 <= length of s1,s2,s3 <= 10

**Format**

**Input**

Three strings

s1

s2

s3

**Output**

Check the sample output and question video

**Example**

**Sample Input**

team

pep

toppr

**Sample Output**

a-3 e-9 m-4 o-1 p-2 r-6 t-0

a-3 e-9 m-5 o-1 p-2 r-7 t-0

a-3 e-9 m-6 o-1 p-2 r-8 t-0

a-4 e-9 m-2 o-1 p-3 r-5 t-0

a-4 e-9 m-5 o-1 p-3 r-8 t-0

a-5 e-9 m-2 o-1 p-4 r-6 t-0

a-5 e-9 m-3 o-1 p-4 r-7 t-0

a-6 e-9 m-2 o-1 p-5 r-7 t-0

a-6 e-9 m-3 o-1 p-5 r-8 t-0

a-7 e-9 m-2 o-1 p-6 r-8 t-0

//6.67 out of 10 points on dados

#include<iostream>

#include<unordered\_map>

#include<vector>

#include<algorithm>

#include<string>

using namespace std;

void baseCasePrintAsoluiton(unordered\_map<char,int> &cimap,string &s1,string &s2,string &s3) {

int n1 {};

int a = s1.length();

for(int i{}; i < a ; i++) {

n1 = n1\*10 +cimap[s1[i]];

// cout<<"------"<<n1<<" "<<cimap[s1[i]];

}

// cout<<"okay?"<<endl;

int n2 {};

for(int i{}; i <s2.length() ; i++) {

n2 = n2\*10 + cimap[s2[i]];

}

// cout<<"okay?"<<endl;

int n3 {};

for(int i{}; i < s3.length() ; i++) {

n3 = n3\*10 +cimap[s3[i]];

}

// cout<<"okay?"<<endl;

if(n1+n2 == n3 ){

// for(auto p: cimap){

// cout<<p.first<<" "<<p.second<<" ";

// }

// cout<<endl;

// cout<<n1<<" "<<n2<<" "<<n1+n2<<" and "<<n3<<endl;

for(char c = 'a'; c<='z';c++){

if(cimap.find(c) != cimap.end()){

cout<<c<<"-"<<cimap[c]<<" ";

}

}

cout<<endl;

}

// cout<<"okay?"<<endl;

// cout<<"okay?"<<endl;

return ;

}

void solution(string &unique, int inx, unordered\_map<char,int> &cimap,vector<bool> &usedNumber,string &s1,string &s2,string &s3) {

if( inx == unique.length()) {

baseCasePrintAsoluiton(cimap,s1,s2,s3);

// cout<<"back back back"<<endl;

return ;

}

// cout<<inx<<" "<<unique[inx]<<endl;

for(int i{0};i < 10; i++){

// cout<<inx<<" "<<unique[inx]<<"this is the i "<<i<<endl;

if(usedNumber[i] == false){

usedNumber[i] = true;

cimap[unique[inx]] = i;

solution(unique ,inx+1, cimap,usedNumber,s1,s2,s3);

usedNumber[i] = false;

cimap[unique[inx]] = -1;

}

}

}

int main() {

string s1 {};

string s2 {};

string s3 {};

cin>> s1>>s2>>s3;

string unique ="";

unordered\_map<char, int> cimap ;//charIntMap

for(int i{}; i< s1.length();i++){

if(cimap.find(s1[i]) == cimap.end()){

cimap[s1[i]] = -1;

unique += s1[i];

}

}

for(int i{}; i< s2.length();i++){

if(cimap.find(s2[i]) == cimap.end()){

cimap[s2[i]] = -1;

unique += s2[i];

}

}

for(int i{}; i< s3.length();i++){

if(cimap.find(s3[i]) == cimap.end()){

cimap[s3[i]] = -1;

unique += s3[i];

}

}

vector<bool> usedNumbers(10,false);

// cout<<unique<<" @@@@@@@@@@@@@@"<<endl;

solution( unique ,0, cimap,usedNumbers,s1,s2,s3);

return 0;

}

Friends Pairing - 2

Easy

1. You are given an integer n, which represents n friends numbered from 1 to n.

2. Each one can remain single or can pair up with some other friend.

3. You have to print all the configurations in which friends can remain single or can be paired up.

Note -> Check out the question video and write the recursive code as it is intended without

changing signature. The judge can't force you but intends you to teach a concept.

**Constraints**

1 <= n <= 10

**Format**

**Input**

A number n

**Output**

Check the sample ouput and question video.

**Example**

**Sample Input**

3

**Sample Output**

1.(1) (2) (3)

2.(1) (2,3)

3.(1,2) (3)

4.(1,3) (2)

#include <iostream>

#include <cmath>

#include <climits>

#include <unordered\_map>

#include <vector>

using namespace std;

int counter = 1;

void solution(int i, int n, vector<bool> used, string asf) {

if(i>n){

cout<<counter<<"."<<asf<<endl;

counter++;

return;

}

if(used[i] == true){

solution(i+1,n,used,asf);

return ;

}

used[i] = true;

solution(i+1,n,used,asf+"("+to\_string(i)+") ");

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

if(used[p] == false){

used[p] = true;

string as = "("+to\_string(i)+","+to\_string(p)+") ";

solution(i+1,n,used,asf+as);

used[p]= false;

}

}

used[i] = false;

return;

}

int main() {

int n{};

cin>> n;

vector<bool> used(n+1);

solution(1,n,used,"");

}

K-partitions

Easy

1. You are given two integers n and k, where n represents number of elements and k represents

number of subsets.

2. You have to partition n elements in k subsets and print all such configurations.

Note -> Check out the question video and write the recursive code as it is intended without

changing signature. The judge can't force you but intends you to teach a concept.

**Constraints**

1 <= n <= 10

1 <= k <= 10

**Format**

**Input**

A number n

A number k

**Output**

Check the sample ouput and question video.

**Example**

**Sample Input**

3

2

**Sample Output**

1. [1, 2] [3]

2. [1, 3] [2]

3. [1] [2, 3]

#include <iostream>

#include <cmath>

#include <climits>

#include <unordered\_map>

#include <vector>

using namespace std;

int counter {1};

void print(vector<vector<int>> &ans) {

cout<<counter<<". ";

counter++;

for(int i{}; i < ans.size();i++) {

cout<<"[";

for(int j {}; j<ans[i].size(); j++){

if(j == ans[i].size()-1){

cout<<ans[i][j];

}else{

cout<<ans[i][j]<<", ";

}

}

cout<<"] ";

}

cout<<endl;

return ;

}

void solution(int i, int n , int k , int rssf ,vector<vector<int>> &ans){

if(i > n) {

if( rssf == k){

print(ans);

}

return ;

}

// cout<<i<<"##44"<<endl;

// for(auto v: ans){

// for(int a: v){

// cout<<a<<" ";

// }

// cout<<endl;

// }

for(int a {}; a< ans.size(); a++){

if(ans[a].size() > 0){

ans[a].push\_back(i);

solution(i+1,n,k,rssf,ans);

ans[a].pop\_back();

}else{

ans[a].push\_back(i);

solution(i+1,n,k,rssf+1,ans);

ans[a].pop\_back();

break;

}

}

return ;

}

int main() {

int n{};

int k{};

cin>> n >>k;

vector<vector<int>> ans (k);

solution(1,n,k,0,ans);

}

All Palindromic Partitions

Medium

1. You are given a string of length n.

2. You have to partition the given string in such a way that every partition is a palindrome.

Note -> Check out the question video and write the recursive code as it is intended without

changing signature. The judge can't force you but intends you to teach a concept.

**Constraints**

1 <= length of string <= 15

**Format**

**Input**

A String of length n

**Output**

Check the sample ouput and question video.

**Example**

**Sample Input**

pep

**Sample Output**

(p) (e) (p)

(pep)

#include <iostream>

#include <string>

using namespace std;

bool isPalindrome(string &str) {

int a{0};

int b = str.length() - 1;

while (a < b) {

if (str[a] != str[b]) {

return false;

}

a++;

b--;

}

return true;

}

void solution(string str, string asf) {

if(str.length() == 0){

cout<<asf<<endl;

return ;

}

string prefix{""};

string ros{""};

for (int i{}; i < str.length(); i++) {

prefix = str.substr(0, i + 1);

ros = str.substr(i + 1, str.length() - (i + 1));

if (isPalindrome(prefix)) {

solution(ros, asf + "(" + prefix + ") ");

}

}

// MY SOLUTION little different

/\*

if(str.length() == 0){

string last = "";

int a = asf.length() - 1;

while(a >= 0 && asf[a] != ' '){

last = asf[a] + last;

a--;

}

asf = asf.substr(1,asf.length()-1);

if(isPalindrome(last)){

cout<<"(";

for(char c:asf){

if( c ==' '){

cout<<") (";

}else{

cout<<c;

}

}

cout<<") "<<endl;

}

return;

}

string last = "";

int a = asf.length() - 1;

while(a >= 0 && asf[a] != ' '){

last = asf[a] + last;

a--;

}

string ith ;

ith = str[0] ;

string ros = str.substr(1,str.length() -1);

if(last.length() == 0 || isPalindrome(last)){

solution(ros, asf +" " +ith );

}

if(last.length() >0) {

solution(ros, asf +ith );

}

\*/

}

int main() {

string str;

cin >> str;

solution(str, "");

return 0;

}

All Palindromic Permutations

Medium

1. You are given a string of length n.

2. You have to print all the palindromic permutations of the given string.

3. If no palindromic permutation exists for the given string, print "-1".

Note -> Check out the question video and write the recursive code as it is intended without

changing signature. The judge can't force you but intends you to teach a concept.

**Constraints**

1 <= length of string <= 15

**Format**

**Input**

A String of length n

**Output**

Check the sample ouput and question video.

**Example**

**Sample Input**

aaabb

**Sample Output**

ababa

baaab

// 2 out of 10 on nados

#include <iostream>

#include <unordered\_map>

#include <string>

#include <vector>

using namespace std;

/\*

public static void generatepw(int cs, int ts, HashMap<Character, Integer> fmap, Character oddc, String asf) {

}

public static void main(String[] args) {

Scanner scn = new Scanner(System.in);

String str = scn.next();

HashMap<Character, Integer> fmap = new HashMap<>();

for (int i = 0; i < str.length(); i++) {

char ch = str.charAt(i);

fmap.put(ch, fmap.getOrDefault(ch, 0) + 1);

}

//write your code here

}

\*/

void generatepw (int cs, int ts ,unordered\_map<char,int> &fmap, char oddc , string asf){

//my algo is same as pepcoding but oreder of ans is different

if( ts == 0 ){

// cout<<"asf ->" <<asf<<endl;

string firstPart = asf;

for(int i{}; i < asf.length()/2; i++ ){

char temp = asf[i];

asf[i] = asf[asf.length() - 1 -i];

asf[asf.length() - 1 -i] = temp;

}

cout<<firstPart + oddc + asf<<endl;

return ;

}

for(pair<char,int> p : fmap){

if(p.second == 0){

continue;

}

fmap[p.first]--;

generatepw(0,ts-1,fmap,oddc,asf+p.first);

fmap[p.first]++;

}

return ;

}

int main() {

string s{};

cin >> s;

unordered\_map<char, int> fmap;

for(int i{}; i< s.length(); i++){

fmap[s[i]]++;

}

//write your code here

bool odd = false;

char oddc {};

for(pair<char,int> p :fmap){

if(p.second % 2 != 0 ){

if(odd == true){

cout<<-1<<endl;

return 0;

}else{

oddc = p.first;

odd = true;

}

}

fmap[p.first] = p.second/2;

}

// for(auto p: fmap){

// cout<<p.first<<" "<<p.second<<endl;

// }

generatepw(0,s.length()/2,fmap,oddc,"");

}

K Subsets With Equal Sum

Medium

1. You are given an array of n distinct integers.

2. You have to divide these n integers into k non-empty subsets such that sum of integers of every

subset is same.

3. If it is not possible to divide, then print "-1".

Note -> Check out the question video and write the recursive code as it is intended without

changing signature. The judge can't force you but intends you to teach a concept.

**Constraints**

1 <= n <= 20

1 <= arr[i] <= 100

1 <= k <= n

**Format**

**Input**

A number n

n distinct integers

A number k

**Output**

Check the sample ouput and question video.

**Example**

**Sample Input**

6

1

2

3

4

5

6

3

**Sample Output**

[1, 6] [2, 5] [3, 4]

#include <iostream>

#include <cmath>

#include <climits>

#include <unordered\_map>

#include <vector>

using namespace std;

/\*

import java.io.\*;

import java.util.\*;

public class Main {

public static void solution(int i, int n, int k, int rssf, ArrayList<ArrayList<Integer>> ans) {

//write your code here

}

public static void main(String[] args) {

Scanner scn = new Scanner(System.in);

int n = scn.nextInt();

int k = scn.nextInt();

ArrayList<ArrayList<Integer>> ans = new ArrayList<>();

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

ans.add(new ArrayList<>());

}

solution(1, n, k, 0, ans);

}

}

\*/

void solve(int i,int n,int k ,int noneg,vector<int>& nums,vector<int>& sob,vector<vector<int>> &ans) {

if(i==n){

if( noneg == k){

bool flag = true;

for(int a{1};a<sob.size(); a++){

if(sob[a] != sob[a-1]){

flag = false;

break;

}

}

if(flag){

for(int i{}; i < ans.size();i++) {

cout<<"[";

for(int j {}; j<ans[i].size(); j++){

if(j == ans[i].size()-1){

cout<<ans[i][j];

}else{

cout<<ans[i][j]<<", ";

}

}

cout<<"] ";

}

cout<<endl;

}

}

return ;

}

if(k - noneg > n-i+1){

return ;

}

for(int a{}; a < noneg;a++){

ans[a].push\_back(nums[i]);

sob[a] += nums[i];

solve(i+1,n,k,noneg,nums,sob,ans);

sob[a] -= nums[i];

ans[a].pop\_back();

}

if( noneg < sob.size()){

sob[noneg] += nums[i];

ans[noneg].push\_back(nums[i]);

solve(i+1,n,k,noneg+1,nums,sob,ans);

sob[noneg] -= nums[i];

ans[noneg].pop\_back();

}

return ;

}

int main() {

int n{};

cin>> n ;

vector<int> nums(n);

for(int i{}; i< n; i++){

cin>>nums[i];

}

int k{};

cin>>k;

int sum {};

for(int a : nums){

sum+=a;

}

if( k>n||sum%k != 0){

cout<<"-1"<<endl;

return 0;

}else if( k == 1){

cout<<"[";

for(int a:nums){

cout<<a<<", ";

}

cout<<"]"<<endl;

return 0;

}

vector<vector<int>> ans (k);

vector<int> sob(k);

solve(0,nums.size(),k,0,nums,sob,ans);

}

Pattern Matching

Medium

1. You are given a string and a pattern.

2. You've to check if the string is of the same structure as pattern without using any regular

expressions.

Note -> Check out the question video and write the recursive code as it is intended without

changing signature. The judge can't force you but intends you to teach a concept.

**Constraints**

1 <= length of str,ptr <= 20

**Format**

**Input**

A String str

A pattern ptr

**Output**

Check the sample ouput and question video.

**Example**

**Sample Input**

graphtreesgraph

pep

**Sample Output**

p -> graph, e -> trees, .

#include <iostream>

#include <string>

#include <cmath>

#include <climits>

#include <unordered\_map>

#include <vector>

using namespace std;

/\*

import java.io.\*;

import java.util.\*;

public class Main {

public static void solution(int i, int n, int k, int rssf, ArrayList<ArrayList<Integer>> ans) {

//write your code here

}

public static void main(String[] args) {

Scanner scn = new Scanner(System.in);

int n = scn.nextInt();

int k = scn.nextInt();

ArrayList<ArrayList<Integer>> ans = new ArrayList<>();

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

ans.add(new ArrayList<>());

}

solution(1, n, k, 0, ans);

}

}

\*/

void solution(string str,string &pattern, unordered\_map<char,string> &map,string op) {

if(op.length()==0) {

if(str.length() != 0){

return;

}else{

//extra work to match the order, otherwise just print the map

string unique{""};

unordered\_map<char,int> uniqueMap;

for(char c: pattern){

if(uniqueMap.find(c) != uniqueMap.end() ){

continue;

}

uniqueMap[c] = 1;

unique += c ;

}

for(char c: unique){

cout<<c<<" -> "<<map[c]<<", ";

}

cout<<"."<<endl;

return ;

}

return;

}

if( str.length() == 0){

return;

}

char thisC = op[0];

string rop = op.substr(1,op.length()-1);//rest of pattern

if(map.find(thisC) !=map.end() ){

string ths = map[thisC];

if(str.length() >= ths.length()){

string pre = str.substr(0,ths.length());

string ros = str.substr(ths.length(),str.length() - ths.length());

if(pre == ths){

solution(ros,pattern, map,rop);

return;

}else{

return;

}

}

}else{

for(int i{} ; i < str.length();i++) {

string pre = str.substr(0,i+1);

string ros = str.substr(i+1,str.length()-i-1);

// cout<<"pre "<<pre<<", "<<"ros "<<ros<<" "<<endl;

map[thisC] = pre;

solution(ros,pattern, map,rop);

map.erase(thisC);

}

}

return;

}

int main() {

string str{};

cin>> str;

string pattern {};

cin>> pattern;

unordered\_map<char,string> map;

solution (str,pattern,map,pattern);

return 0;

}

Word Break - I

Easy

1. You are given n space separated strings, which represents a dictionary of words.

2. You are given another string which represents a sentence.

3. You have to print all possible sentences from the string, such that words of the sentence are

present in dictionary.

Note -> Check out the question video and write the recursive code as it is intended without

changing signature. The judge can't force you but intends you to teach a concept.

**Constraints**

1 <= number of words <= 10

1 <= length of each word <= 15

1 <= length of sentence <= 1000

**Format**

**Input**

A number n

n strings representing words

a string representing a sentence

**Output**

Check the sample ouput and question video.

**Example**

**Sample Input**

11

i like pep coding pepper eating mango man go in pepcoding

ilikepeppereatingmangoinpepcoding

**Sample Output**

i like pepper eating man go in pep coding

i like pepper eating man go in pepcoding

i like pepper eating mango in pep coding

i like pepper eating mango in pepcoding

#include <iostream>

#include <cmath>

#include <climits>

#include <unordered\_map>

#include <vector>

#include <string>

using namespace std;

/\*

public static void main(String[] args) {

Scanner scn = new Scanner(System.in);

int n = scn.nextInt();

HashSet<String> dict = new HashSet<>();

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

dict.add(scn.next());

}

String sentence = scn.next();

wordBreak(sentence,"", dict);

}

public static void wordBreak(String str, String ans, HashSet<String> dict){

// write your code here

}

\*/

void wordBreak(string str, string ans, unordered\_map<string,bool> &dict) {

if(str.length() == 0){

cout<<ans<<endl;

return ;

}

string pre {};

string rest{};

for(int i{}; i<str.length() ; i++) {

pre = str.substr(0,i+1);

rest = str.substr(i+1,str.length() - i-1);

if(dict.find(pre) != dict.end() ){

// dict[pre] = false;// don't do that as one word may bw used several times

wordBreak(rest,ans+ pre+" ",dict);

}

}

}

int main() {

// i think only unique space seperated strings is passed but one word may bw used several times

unordered\_map<string,bool> dict; //there bool is of no use pepcoding used hashset

int n{};

cin>>n;

string in{};

while(n--){

cin >> in;

dict[in] = true;

}

string str{};

cin>> str;

wordBreak(str,"", dict);

return 0;

}

Remove Invalid Parenthesis

Hard

1. You are given a string, which represents an expression having only opening and closing parenthesis.

2. You have to remove minimum number of parenthesis to make the given expression valid.

3. If there are multiple answers, you have to print all of them.

Note -> Check out the question video and write the recursive code as it is intended without changing signature. The judge can't

force you but intends you to teach a concept.

**Constraints**

1 <= length of string <= 20

**Format**

**Input**

A string containing only opening and closing parenthesis

**Output**

Print all the Valid expressions.

Check the sample ouput and question video.

**Example**

**Sample Input**

()())()

**Sample Output**

(())()

()()()

#include<iostream>

#include<unordered\_map>

#include<vector>

#include<algorithm>

#include<string>

#include<stack>

using namespace std;

/\*

public static void solution(String str, int minRemoval, HashSet<String> ans) {

//write your code here

}

public static int getMin(String str){

//write your code here

return 0;

}

public static void main(String[] args) {

Scanner scn = new Scanner(System.in);

String str = scn.next();

solution(str, getMin(str),new HashSet<>());

}

\*/

int getMin(string str) {

//write your code here

stack<char> st;

for(char c: str){

if( c == '('){

st.push(c);

}else if(c == ')'){

if(st.size() > 0 && st.top() == '('){

st.pop();

}else{

st.push(c);

}

}

}

return st.size();

}

void solution(string str, int minRemoval,unordered\_map<string, bool> &ans) {

//write your code here

if( minRemoval == 0){

if(getMin(str) == 0){

if( ans.find(str) == ans.end() ){

cout<<str<<endl;

ans[str] = true;

}

}

return ;

}

int ii{};

while (str[ii] == ')'){

ii++;

}

string st = str.substr(ii,str.length()-ii);

minRemoval -=ii;

if (minRemoval <= 0){

return;

}

ii = str.length()-1;

while(ii>=0&&st[ii] == '('){

ii--;

}

minRemoval -= str.length()-(ii+1);

if (minRemoval <= 0){

return;

}

str = st.substr(0,ii+1);

string left {};

string right{};

for(int i{}; i< str.length();i++ ){

left = str.substr(0,i);

right = str.substr(i+1,str.length() -i-1);

solution(left +right,minRemoval-1,ans);

}

}

int main(){

string str{};

cin>> str;

unordered\_map<string ,bool> ans; //there bool is of no use pepcoding used hashset

// int a = getMin(str);

// cout<<a<<endl;

solution(str,getMin(str),ans);

}

Largest Number Possible After At Most K Swaps

Easy

1. You are given a string which represents digits of a number.

2. You have to create the maximum number by performing at-most k swap operations on its digits.

Note -> Check out the question video and write the recursive code as it is intended without

changing signature. The judge can't force you but intends you to teach a concept.

**Constraints**

1 <= length of S <= 30

1 <= K <= 10

**Format**

**Input**

A string S and a number K

**Output**

A number

**Example**

**Sample Input**

1234567

4

**Sample Output**

7654321

#include<iostream>

#include<unordered\_map>

#include<vector>

#include<algorithm>

#include<string>

using namespace std;

/\*

static String max;

public static void findMaximum(String str, int k) {

//write your code here

}

public static void main(String[] args) {

Scanner scn = new Scanner(System.in);

String str = scn.next();

int k = scn.nextInt();

max = str;

findMaximum(str, k);

System.out.println(max);

}

\*/

string max {};

int maxi{};

void findMaximum(string str, int k) {

if(k == 0){

int toCheck = stoi(str);

if( toCheck > maxi){

cout<<"str -> "<<str<<" earlier max "<<maxi<<endl;

maxi = toCheck;

::max = str;

}

return ;

}

for(int i{}; i< str.length();i++) {

for(int j{i+1}; j< str.length();j++) {

if(str[j]<=str[i]){

continue;

}

char t = str[i];

str [i] = str[j];

str[j] = t;

int toCheck = stoi(str);

if( toCheck > maxi){

// cout<<"str -> "<<str<<" earlier max "<<maxi<<endl;

maxi = toCheck;

::max = str;

}

findMaximum(str,k-1);

str[j] = str[i];

str[i] = t;

}

}

}

int main() {

string str{};

cin>> str;

int k {};

cin >> k;

::max = str;

maxi = stoi(str);

findMaximum(str, k);

cout<<::max<<endl;

}

Tug Of War

Easy

1. You are given an array of n integers.

2. You have to divide these n integers into 2 subsets such that the difference of sum of two subsets

is as minimum as possible.

3. If n is even, both set will contain exactly n/2 elements. If is odd, one set will contain (n-1)/2 and

other set will contain (n+1)/2 elements.

3. If it is not possible to divide, then print "-1".

Note -> Check out the question video and write the recursive code as it is intended without

changing signature. The judge can't force you but intends you to teach a concept.

**Constraints**

1 <= n <= 20

1 <= arr[i] <= 100

**Format**

**Input**

A number n

n integers

**Output**

Check the sample ouput and question video.

**Example**

**Sample Input**

6

1

2

3

4

5

6

**Sample Output**

[1, 3, 6] [2, 4, 5]

#include <iostream>

#include <cmath>

#include <climits>

#include <cstdlib>

#include <unordered\_map>

#include <vector>

using namespace std;

//my solution using k partition

void solve(int &minDiff,int i,int n ,vector<int>& nums,vector<int>& sob,vector<vector<int>> &ans,vector<vector<int>> &finalans) {

if(i==n){

if(abs(sob[0]-sob[1]) <= minDiff){

// cout<<"check"<<endl;

// for(auto v: ans){

// for(int a: v){

// cout<<a<<" ";

// }

// cout<<endl;

// }

finalans = ans;

minDiff = abs(sob[0]-sob[1]);

}

return ;

}

// for(auto v: ans){

// for(int a: v){

// cout<<a<<" ";

// }

// cout<<endl;

// }

if(ans[0].size() < (nums.size()+1)/2){

ans[0].push\_back(nums[i]);

sob[0] += nums[i];

solve(minDiff,i+1,n,nums,sob,ans,finalans);

sob[0] -= nums[i];

ans[0].pop\_back();

}

if(ans[1].size() < (nums.size()+1)/2){

ans[1].push\_back(nums[i]);

sob[1] += nums[i];

solve(minDiff,i+1,n,nums,sob,ans,finalans);

sob[1] -= nums[i];

ans[1].pop\_back();

}

// for(int a{}; a < ans.size();a++){//it seems we are looping in the nums but we are looping in the bags

// if( a < noneg){

// ans[a].push\_back(nums[i]);

// sob[a] += nums[i];

// solve(minDiff,i+1,n,noneg,nums,sob,ans,finalans);

// sob[a] -= nums[i];

// ans[a].pop\_back();

// }else if( a == noneg){

// sob[noneg] += nums[i];

// ans[noneg].push\_back(nums[i]);

// solve(minDiff,i+1,n,noneg+1,nums,sob,ans,finalans);

// sob[noneg] -= nums[i];

// ans[noneg].pop\_back();

// break;

// }

// }

return ;

}

int main() {

int n{};

cin>> n ;

vector<int> nums(n);

for(int i{}; i< n; i++){

cin>>nums[i];

}

if( 2 > n){

cout<<"-1"<<endl;

return 0;

}

vector<vector<int>> ans (2);

vector<vector<int>> finalans (2);

vector<int> sob(2);

int minDiff = INT\_MAX;

solve(minDiff,0,nums.size(),nums,sob,ans,finalans);

// cout<<"&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&"<<endl;

for(auto v: ans){

for(int a: v){

cout<<a<<" ";

}

}

for(int i{}; i < finalans.size();i++) {

cout<<"[";

for(int j {}; j < finalans[i].size(); j++){

if(j == finalans[i].size()-1){

cout<<finalans[i][j];

}else{

cout<<finalans[i][j]<<", ";

}

}

cout<<"] ";

}

cout<<endl;

}