#include <iostream>

#include <set>

using namespace std;

int main ()

{

int myints[] = {1,2,3,4,5};

set<int> myset (myints,myints+5);

set<int>::iterator it;

cout << "myset contains:";

for ( it=myset.begin() ; it != myset.end(); it++ )

cout << " " << \*it;

cout << endl;

return 0;

}

#include <iostream>

#include <map>

using namespace std;

int main(){

multimap<int,int> a;

a.insert(pair<int,int> (1,2));

a.insert(pair<int,int> (1,2));

a.insert(pair<int,int> (1,4));

for (multimap<int,int>::iterator it= a.begin(); it != a.end(); ++it) {

cout << it->first << "\t" << it->second << endl ;

}

return 0;

}

#include <iostream>

#include <vector>

using namespace std;

int main()

{

// create a vector to store int

vector<int> vec;

int i;

// display the original size of vec

cout << "vector size = " << vec.size() << endl;

// push 5 values into the vector

for(i = 0; i < 5; i++){

vec.push\_back(i);

}

// display extended size of vec

cout << "extended vector size = " << vec.size() << endl;

// access 5 values from the vector

for(i = 0; i < 5; i++){

cout << "value of vec [" << i << "] = " << vec[i] << endl;

}

// use iterator to access the values

vector<int>::iterator v = vec.begin();

while( v != vec.end()) {

cout << "value of v = " << \*v << endl;

v++;

}

return 0;

}

------------------------------------------------\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

1. */\**
2. *\* C++ Program to implement Sieve of Eratosthenes*
3. *\*/*
5. #include <iostream>
6. const int len = 100;
8. int main()
9. {
10. int arr[100] = {0};
12. for (int i = 2; i < 100; i++)
13. {
14. for (int j = i \* i; j < 100; j+=i)
15. {
16. arr[j - 1] = 1;
17. }
18. }
19. for (int i = 1; i < 100; i++)
20. {
21. if (arr[i - 1] == 0)
22. std::cout << i << "**\t**";
23. }
24. }

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#include<iostream>

#include<conio.h>

#include<stdlib.h>

using namespace std;

int cost[10][10],i,j,k,n,qu[10],front,rare,v,visit[10],visited[10];

main()

{

int m;

cout <<"enterno of vertices";

cin >> n;

cout <<"ente no of edges";

cin >> m;

cout <<"**\n**EDGES **\n**";

for(k=1;k<=m;k++)

{

cin >>i>>j;

cost[i][j]=1;

}

cout <<"enter initial vertex";

cin >>v;

cout <<"Visitied vertices**\n**";

cout << v;

visited[v]=1;

k=1;

while(k<n)

{

for(j=1;j<=n;j++)

if(cost[v][j]!=0 && visited[j]!=1 && visit[j]!=1)

{

visit[j]=1;

qu[rare++]=j;

}

v=qu[front++];

cout<<v << " ";

k++;

visit[v]=0; visited[v]=1;

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#include <iostream>

#include <ctime>

using namespace std;

struct node

{

    int info;

    node \*next;

};

class Queue

{

    private:

        node \*front;

        node \*rear;

    public:

        Queue();

        ~Queue();

        bool isEmpty();

        void enqueue(int);

        int dequeue();

        void display();

};

void Queue::display()

{

    node \*p = new node;

    p = front;

    if(front == NULL)

    {

        cout<<"\nNothing to Display\n";

    }else{

        while(p!=NULL){

            cout<<endl<<p->info;

            p = p->next;

        }

    }

}

Queue::Queue()

{

    front = NULL;

    rear = NULL;

}

Queue::~Queue()

{

    delete front;

}

void Queue::enqueue(int data)

{

    node \*temp = new node();

    temp->info = data;

    temp->next = NULL;

    if(front == NULL){

        front = temp;

    }else{

        rear->next = temp;

    }

    rear = temp;

}

int Queue::dequeue() {

    node \*temp = new node();

    int value;

    if(front == NULL){

        cout<<"\nQueue is Emtpty\n";

    }else{

        temp = front;

        value = temp->info;

        front = front->next;

        delete temp;

    }

    return value;

}

bool Queue::isEmpty()

{

    return (front == NULL);

}

class Graph {

    private:

        int n; // n represents the number of vertices in the graph

        int \*\*A; // The function of A is storing the edge between two vertices

    public:

        Graph(int size = 2);

        ~Graph();

        bool isConnected(int, int);

        void addEdge(int u, int v);

        void BFS(int );

};

Graph::Graph(int size)

{

    int i, j;

    if (size < 2) n = 2;

    else n = size;

    A = new int\*[n];

    for (i = 0; i < n; ++i)

        A[i] = new int[n];

    for (i = 0; i < n; ++i)

        for (j = 0; j < n; ++j)

            A[i][j] = 0;

}

Graph::~Graph()

{

    for (int i = 0; i < n; ++i)

    delete [] A[i];

    delete [] A;

}

bool Graph::isConnected(int u, int v) {

    return (A[u-1][v-1] == 1);

}

void Graph::addEdge(int u, int v) {

    A[u-1][v-1] = A[v-1][u-1] = 1;

}

void Graph::BFS(int s) {

    Queue Q;

    bool \*explored = new bool[n+1];//  it Keeps track of explored vertices

    for (int i = 1; i <= n; ++i)// Initailization of  all vertices as unexplored

    explored[i] = false;

    Q.enqueue(s);// Pushing of  initial vertex to the queue

    explored[s] = true; // marking it as explored

    cout << "Breadth first Search starting from vertex ";

    cout << s << " : " << endl;

    //Unless the queue is empty is to be performed

    while (!Q.isEmpty()) {

        // Pop the vertex from the queue

        int v = Q.dequeue();

        //display the explored vertices

        cout << v << " ";

        //From the explored vertex v try to explore all the

        //connected vertices

        for (int w = 1; w <= n; ++w)

            /\*Explores the vertex w if it is connected to v

            and and if it is unexplored\*/

            if (isConnected(v, w) && !explored[w]) {

                //adds the vertex w to the queue

                Q.enqueue(w);

                //marks the vertex w as visited

                explored[w] = true;

            }

    }

    cout << endl;

    delete [] explored;

}

int main() {

    // Creates a graph with 12 vertices

    Graph g(12);

    //Adds edges to the graph \*/

    g.addEdge(1, 2); g.addEdge(1, 3);

    g.addEdge(2, 4); g.addEdge(3, 4);

    g.addEdge(3, 6); g.addEdge(4 ,7);

    g.addEdge(5, 6); g.addEdge(5, 7);

    clock\_t t1;

    t1 = clock();

    //Explores all vertices findable from vertex 1

    g.BFS(1);

    float diff = (double)(clock() - t1)/CLOCKS\_PER\_SEC ;

    cout <<endl<< "The time taken for Breadth first search: "<< diff << endl;

}

void myFunction( int counter)  
{  
if(counter == 0)  
     return;  
else  
       {  
       cout <<counter<<endl;  
       myFunction(--counter);  
       return;  
       }  
}

void myFunction( int counter)  
{  
if(counter == 0)  
     return;  
else  
       {  
       cout<<"hello"<<counter<<endl;  
       myFunction(--counter);  
       cout<<counter<<endl;  
       return;  
       }  
}

int myFactorial( int integer)  
{  
if( integer == 1)  
     return 1;  
else  
       {  
       return (integer \* (myFactorial(integer-1)));  
       }  
}

int factorial(int number) {

if(number == 0) {

return 1;

}

return factorial\_i(number, 1);

}

int factorial\_i(int currentNumber, int sum) {

if(currentNumber == 1) {

return sum;

} else {

return factorial\_i(currentNumber - 1, sum\*currentNumber);

}

}

An example to print numbers counting down:

void print(int p)

{

if (p==0)

return;

cout<<p;

print(p-1);

return;

}

An example to print counting up:

void print(int p)

{

if (p==0)

return;

print(p-1);

cout<<p;

return;

}

An example to produce the fibonacci number for a given index in the series:

int Fibonacci(int n)

{

if (n==0)

return 0;

if (n==1)

return 1;

return( Fibonacci(n-2) + Fibonacci(n-1) );

}

A recursive function to determine if an input is prime:

bool isPrime(int p, int i=2)

{

if (i==p) return 1;//or better if (i\*i>p) return 1;

if (p%i == 0) return 0;

return isPrime (p, i+1);

}

// two versions of recursive solution to adding up numbers from 1 to any given number.

// the second example is tail recusion because once the total is found, the function returns and

// does not need to unravel previous recursive steps

int sum (int num)

{

if (num==0)

return 0;

return (sum(num-1)+(num));

}

int sum (int num, int total=0)

{

if (num<=0)

return total;

sum( num-1, sum );

}

1. // A Dynamic Programming based solution for 0-1 Knapsack problem
2. #include <iostream>
4. using namespace std;
6. // A utility function that returns maximum of two integers
7. int max(int a, int b)
8. {
9. return (a > b) ? a : b;
10. }
12. // Returns the maximum value that can be put in a knapsack of capacity W
13. int knapSack(int W, int wt[], int val[], int n)
14. {
15. int i, w;
16. int K[n + 1][W + 1];
18. // Build table K[][] in bottom up manner
19. for (i = 0; i <= n; i++)
20. {
21. for (w = 0; w <= W; w++)
22. {
23. if (i == 0 || w == 0)
24. K[i][w] = 0;
25. else if (wt[i - 1] <= w)
26. K[i][w]
27. = max(val[i - 1] + K[i - 1][w - wt[i - 1]], K[i - 1][w]);
28. else
29. K[i][w] = K[i - 1][w];
30. }
31. }
33. return K[n][W];
34. }
36. int main()
37. {
38. cout << "Enter the number of items in a Knapsack:";
39. int n, W;
40. cin >> n;
41. int val[n], wt[n];
42. for (int i = 0; i < n; i++)
43. {
44. cout << "Enter value and weight for item " << i << ":";
45. cin >> val[i];
46. cin >> wt[i];
47. }
49. // int val[] = { 60, 100, 120 };
50. // int wt[] = { 10, 20, 30 };
51. // int W = 50;
52. cout << "Enter the capacity of knapsack";
53. cin >> W;
54. cout << knapSack(W, wt, val, n);
56. return 0;
57. }

// C++ program to find number of ways to wear hats

#include<bits/stdc++.h>

#define MOD 1000000007

using namespace std;

// capList[i]'th vector contains the list of persons having a cap with id i

// id is between 1 to 100 so we declared an array of 101 vectors as indexing

// starts from 0.

vector<int> capList[101];

// dp[2^10][101] .. in dp[i][j], i denotes the mask i.e., it tells that

// how many and which persons are wearing cap. j denotes the first j caps

// used. So, dp[i][j] tells the number ways we assign j caps to mask i

// such that none of them wears the same cap

int dp[1025][101];

// This is used for base case, it has all the N bits set

// so, it tells whether all N persons are wearing a cap.

int allmask;

// Mask is the set of persons, i is the number of

// caps processed starting from first cap.

long long int countWaysUtil(int mask, int i)

{

// If all persons are wearing a cap so we

// are done and this is one way so return 1

if (mask == allmask) return 1;

// If not everyone is wearing a cap and also there are no more

// caps left to process, so there is no way, thus return 0;

if (i > 100) return 0;

// If we already have solved this subproblem, return the answer.

if (dp[mask][i] != -1) return dp[mask][i];

// Ways, when we don't include this cap in our arrangement

// or solution set.

long long int ways = countWaysUtil(mask, i+1);

// size is the total number of persons having cap with id i.

int size = capList[i].size();

// So, assign one by one ith cap to all the possible persons

// and recur for remaining caps.

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

{

// if person capList[i][j] is already wearing a cap so continue as

// we cannot assign him this cap

if (mask & (1 << capList[i][j])) continue;

// Else assign him this cap and recur for remaining caps with

// new updated mask vector

else ways += countWaysUtil(mask | (1 << capList[i][j]), i+1);

ways %= MOD;

}

// Save the result and return it.

return dp[mask][i] = ways;

}

// Reads n lines from standard input for current test case

void countWays(int n)

{

//----------- READ INPUT --------------------------

string temp, str;

int x;

getline(cin, str); // to get rid of newline character

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

{

getline(cin, str);

stringstream ss(str);

// while there are words in the streamobject ss

while (ss >> temp)

{

stringstream s;

s << temp;

s >> x;

// add the ith person in the list of cap if with id x

capList[x].push\_back(i);

}

}

//----------------------------------------------------

// All mask is used to check of all persons

// are included or not, set all n bits as 1

allmask = (1 << n) - 1;

// Initialize all entries in dp as -1

memset(dp, -1, sizeof dp);

// Call recursive function count ways

cout << countWaysUtil(0, 1) << endl;

}

// Driver Program

int main()

{

int n; // number of persons in every test case

cin >> n;

countWays(n);

return 0;

}

/\* C program to find length of the shortest supersequence \*/

#include<stdio.h>

#include<string.h>

/\* Utility function to get max of 2 integers \*/

int max(int a, int b) { return (a > b)? a : b; }

/\* Returns length of LCS for X[0..m-1], Y[0..n-1] \*/

int lcs( char \*X, char \*Y, int m, int n);

// Function to find length of the shortest supersequence

// of X and Y.

int shortestSuperSequence(char \*X, char \*Y)

{

int m = strlen(X), n = strlen(Y);

int l = lcs(X, Y, m, n); // find lcs

// Result is sum of input string lengths - length of lcs

return (m + n - l);

}

/\* Returns length of LCS for X[0..m-1], Y[0..n-1] \*/

int lcs( char \*X, char \*Y, int m, int n)

{

int L[m+1][n+1];

int i, j;

/\* Following steps build L[m+1][n+1] in bottom up fashion.

Note that L[i][j] contains length of LCS of X[0..i-1]

and Y[0..j-1] \*/

for (i=0; i<=m; i++)

{

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

{

if (i == 0 || j == 0)

L[i][j] = 0;

else if (X[i-1] == Y[j-1])

L[i][j] = L[i-1][j-1] + 1;

else

L[i][j] = max(L[i-1][j], L[i][j-1]);

}

}

/\* L[m][n] contains length of LCS for X[0..n-1] and

Y[0..m-1] \*/

return L[m][n];

}

/\* Driver program to test above function \*/

int main()

{

char X[] = "AGGTAB";

char Y[] = "GXTXAYB";

printf("Length of the shortest supersequence is %d\n",

shortestSuperSequence(X, Y));

return 0;

}

#include<bits/stdc++.h>

#define n 3

using namespace std;

// Returns length of the longest path beginning with mat[i][j].

// This function mainly uses lookup table dp[n][n]

int findLongestFromACell(int i, int j, int mat[n][n], int dp[n][n])

{

// Base case

if (i<0 || i>=n || j<0 || j>=n)

return 0;

// If this subproblem is already solved

if (dp[i][j] != -1)

return dp[i][j];

// Since all numbers are unique and in range from 1 to n\*n,

// there is atmost one possible direction from any cell

if (j<n-1 && ((mat[i][j] +1) == mat[i][j+1]))

return dp[i][j] = 1 + findLongestFromACell(i,j+1,mat,dp);

if (j>0 && (mat[i][j] +1 == mat[i][j-1]))

return dp[i][j] = 1 + findLongestFromACell(i,j-1,mat,dp);

if (i>0 && (mat[i][j] +1 == mat[i-1][j]))

return dp[i][j] = 1 + findLongestFromACell(i-1,j,mat,dp);

if (i<n-1 && (mat[i][j] +1 == mat[i+1][j]))

return dp[i][j] = 1 + findLongestFromACell(i+1,j,mat,dp);

// If none of the adjacent fours is one greater

return dp[i][j] = 1;

}

// Returns length of the longest path beginning with any cell

int finLongestOverAll(int mat[n][n])

{

int result = 1; // Initialize result

// Create a lookup table and fill all entries in it as -1

int dp[n][n];

memset(dp, -1, sizeof dp);

// Compute longest path beginning from all cells

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

{

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

{

if (dp[i][j] == -1)

findLongestFromACell(i, j, mat, dp);

// Update result if needed

result = max(result, dp[i][j]);

}

}

return result;

}

// Driver program

int main()

{

int mat[n][n] = {{1, 2, 9},

{5, 3, 8},

{4, 6, 7}};

cout << "Length of the longest path is "

<< finLongestOverAll(mat);

return 0;

}

// A C program to find minimum possible time by the car chassis to complete

#include <stdio.h>

#define NUM\_LINE 2

#define NUM\_STATION 4

// Utility function to find minimum of two numbers

int min(int a, int b) { return a < b ? a : b; }

int carAssembly(int a[][NUM\_STATION], int t[][NUM\_STATION], int \*e, int \*x)

{

int T1[NUM\_STATION], T2[NUM\_STATION], i;

T1[0] = e[0] + a[0][0]; // time taken to leave first station in line 1

T2[0] = e[1] + a[1][0]; // time taken to leave first station in line 2

// Fill tables T1[] and T2[] using the above given recursive relations

for (i = 1; i < NUM\_STATION; ++i)

{

T1[i] = min(T1[i-1] + a[0][i], T2[i-1] + t[1][i] + a[0][i]);

T2[i] = min(T2[i-1] + a[1][i], T1[i-1] + t[0][i] + a[1][i]);

}

// Consider exit times and retutn minimum

return min(T1[NUM\_STATION-1] + x[0], T2[NUM\_STATION-1] + x[1]);

}

int main()

{

int a[][NUM\_STATION] = {{4, 5, 3, 2},

{2, 10, 1, 4}};

int t[][NUM\_STATION] = {{0, 7, 4, 5},

{0, 9, 2, 8}};

int e[] = {10, 12}, x[] = {18, 7};

printf("%d", carAssembly(a, t, e, x));

return 0;

}

// A Dynamic Programming based program to check whether a string C is

// an interleaving of two other strings A and B.

#include <iostream>

#include <string.h>

using namespace std;

// The main function that returns true if C is

// an interleaving of A and B, otherwise false.

bool isInterleaved(char\* A, char\* B, char\* C)

{

// Find lengths of the two strings

int M = strlen(A), N = strlen(B);

// Let us create a 2D table to store solutions of

// subproblems. C[i][j] will be true if C[0..i+j-1]

// is an interleaving of A[0..i-1] and B[0..j-1].

bool IL[M+1][N+1];

memset(IL, 0, sizeof(IL)); // Initialize all values as false.

// C can be an interleaving of A and B only of sum

// of lengths of A & B is equal to length of C.

if ((M+N) != strlen(C))

return false;

// Process all characters of A and B

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

{

for (int j=0; j<=N; ++j)

{

// two empty strings have an empty string

// as interleaving

if (i==0 && j==0)

IL[i][j] = true;

// A is empty

else if (i==0 && B[j-1]==C[j-1])

IL[i][j] = IL[i][j-1];

// B is empty

else if (j==0 && A[i-1]==C[i-1])

IL[i][j] = IL[i-1][j];

// Current character of C matches with current character of A,

// but doesn't match with current character of B

else if(A[i-1]==C[i+j-1] && B[j-1]!=C[i+j-1])

IL[i][j] = IL[i-1][j];

// Current character of C matches with current character of B,

// but doesn't match with current character of A

else if (A[i-1]!=C[i+j-1] && B[j-1]==C[i+j-1])

IL[i][j] = IL[i][j-1];

// Current character of C matches with that of both A and B

else if (A[i-1]==C[i+j-1] && B[j-1]==C[i+j-1])

IL[i][j]=(IL[i-1][j] || IL[i][j-1]) ;

}

}

return IL[M][N];

}

// A function to run test cases

void test(char \*A, char \*B, char \*C)

{

if (isInterleaved(A, B, C))

cout << C <<" is interleaved of " << A <<" and " << B << endl;

else

cout << C <<" is not interleaved of " << A <<" and " << B << endl;

}

// Driver program to test above functions

int main()

{

test("XXY", "XXZ", "XXZXXXY");

test("XY" ,"WZ" ,"WZXY");

test ("XY", "X", "XXY");

test ("YX", "X", "XXY");

test ("XXY", "XXZ", "XXXXZY");

return 0;

}