**ADA PROGRAMS:**

**ADA 1: Quick sort ‘k’th smallest element**

#include <iostream>

#include <time.h>

using namespace std;

void quick(int[],int,int);

int partition(int[],int,int);

int main()

{

int a[10],n,i,s,e,k;

double time;

clock\_t start,end;

start=clock();

cout<<"Enter the size of an array: ";

cin>>n;

cout<<"Enter the elements of an array: ";

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

{

cin>>a[i];

}

cout<<"Enter start index: ";

cin>>s;

cout<<"Enter end index: ";

cin>>e;

cout<<"Enter k'th position: ";

cin>>k;

quick(a,s+1,e+1);

cout<<"Element at ‘k’th positon: "<<a[k-1];

time=(double)(end-start)/CLOCKS\_PER\_SEC;

cout<<"\nExecution time: "<<time<<"secs\n";

}

void quick(int a[],int low,int high)

{

int mid;

if(low<high)

{

mid=partition(a,low,high);

quick(a,low,mid-1);

quick(a,mid+1,high);

}

}

int partition(int a[],int low,int high)

{

int pivot,i,j,temp;

pivot=a[low];

i=low;

j=high;

while(i<=j)

{

while(a[i]<=pivot)

i++;

while(a[j]>pivot)

j--;

if(i<j)

{

temp=a[i];

a[i]=a[j];

a[j]=temp;

}

temp=a[low];

a[low]=a[j];

a[j]=temp;

return j;

}

}

OUTPUT:

Enter the size of an array: 6

Enter the elements of an array: 3 2 5 1 8 9

Enter start index: 1

Enter end index: 6

Enter ‘k’th position: 4

Element at ‘k’th positon: 5

**ADA 2: DFS/BFS connected components for undirected graph**

#include<iostream>

using namespace std;

class connected

{

int adjacent[10][10],visited[10],v,n;

public:

void input()

{

cout<<"enter the no. of components\n";

cin>>n;

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

{

visited[i]=0;

}

cout<<"enter the adjacent matrix of graph\n";

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

{

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

{

cin>>adjacent[i][j];

}

}

}

void DFS(int v)

{

cout<<v+1<<",";

visited[v]=1;

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

{

if(adjacent[v][i]==1 && visited[i]==0)

{

DFS(i);

}

}

}

void connected1()

{

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

{

if(visited[j]==0)

{

DFS(j);

cout<<"\n";

}

}

}

};

int main()

{

connected c;

c.input();

c.connected1();

return 0;

}

/\*

OUTPUT:

enter the no. of components

6

enter the adjacent matrix of graph

0 0 1 0 0 0

0 0 0 0 0 1

1 0 0 0 1 0

0 0 0 0 0 0

0 0 1 0 0 0

0 1 0 0 0 0

0,2,4,

1,5,

3,

\*/

**ADA** **3: DFS/BFS Clusters**

#include <bits/stdc++.h>

using namespace std;

int ctr = 0, flag = 0;

int rows, cols, arr[10][10];

bool isPossible(int row, int col)

{

if (row < rows && row >= 0 && col < cols && col >= 0)

return 1;

return 0;

}

void dfs(int row, int col)

{

int testrow[] = {row-1, row, row+1};

int testcol[] = {col-1, col, col+1};

for(int i = 0; i <= 2; i++)

{

for(int j = 0; j <= 2; j++)

{

if(isPossible(testrow[i], testcol[j]))

{

if(flag == 0)

{

ctr++;

flag = 1;

}

if(arr[testrow[i]][testcol[j]] == 1)

{

arr[testrow[i]][testcol[j]] = 0;

dfs(testrow[i], testcol[j]);

}

}

}

}

}

int main()

{

clock\_t start,end;

cout << "Enter no of rows and cols" << endl;

cin >> rows;

cin >> cols;

cout << "Enter elements of the array:" << endl;

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

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

cin >> arr[i][j];

start = clock();

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

{

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

{

if(arr[i][j] == 1)

{

flag = 0;

dfs(i, j);

flag = 1;

}

}

}

end = clock();

cout << "No of Islands: " << ctr << endl;

float exe = float(end-start)/CLOCKS\_PER\_SEC;

cout << "\nExecution time : "<<exe<<endl;

return 0;

}

/\*

OUTPUT:

Enter no of rows and cols

5

5

Enter elements of the array:

1 1 0 0 0

0 1 0 0 1

1 0 0 1 1

0 0 0 0 0

1 0 1 1 0

No of Islands: 4

Execution time : 8e-06

\*/

**ADA 4: N-Queens**

#include<bits/stdc++.h>

using namespace std;

int x[10];

bool flag=false;

bool place(int k,int i)

{

for(int j=1;j<=k-1;j++)

{

if((x[j]==i) || (abs(x[j]-i)==abs(j-k)))

{

return false;

}

}

return true;

}

void Nqueen(int k,int n)

{

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

{

if(place(k,i))

{

x[k]=i;

if(k==n)

{

for(int r=1;r<=n;r++)

{

cout<<x[r];

cout<<",";

}

flag=true;

cout<<"\n";

}

else

{

Nqueen(k+1,n);

}

}

}

}

int main()

{

int n;

cout<<"enter number of queens";

cin>>n;

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

{

x[i]=0;

}

Nqueen(1,n);

if(!flag)

{

cout<<"No solution";

}

}

/\*OUTPUT:

Enter number of queens: 4

2,4,1,3,

3,1,4,2

\*/

**ADA 5: Topological sorting Dependencies**

#include<bits/stdc++.h>

using namespace std;

int a[10][10],n;

void input()

{

int c=0,b=0;

cout<<"Enter no. of tasks \n";

cin>>n;

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

{

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

a[i][j]=0;

}

}

while(c!= -1 || b!=-1)

{

cout<<"Enter dependencies [b:c]........[-1,-1] to stop\n";

cin>>c; //c should be executed first

cin>>b;

if(c!= -1 || b!=-1)

{

a[c][b]=1;

}

}

}

void findorder()

{

int indegree[10],k,s[10],top=-1;

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

{

indegree[i]=0;

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

{

indegree[i] +=a[j][i];

}

}

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

{

if(indegree[i]==0)

{

s[++top]=i;

}

}

while(top!=-1 )

{

k=s[top--];

cout<<k<<",";

indegree[k]=-1;

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

{

if(a[k][r]==1)

{

indegree[r] -=1;

if(indegree[r]== 0)

{

s[++top]=r;

}

}

}

}

}

int main()

{

input();

findorder();

}

/\*

OUTPUT:

Enter no. of tasks:

4

Enter dependencies[c:b].....[-1,-1] to stop

0 1

Enter dependencies[c:b].....[-1,-1] to stop

3 2

Enter dependencies[c:b].....[-1,-1] to stop

-1 -1

3,2,0,1

\*/

**ADA 6: Back tracking Sum of Subsets**

#include <bits/stdc++.h>

using namespace std;

void displaySubset(int subSet[], int size)

{

cout<<"{";

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

{

cout << subSet[i] << ",";

}

cout<<"}";

cout << endl;

}

void subsetSum(int set[], int subSet[], int n, int subSize, int total, int nodeCount ,int sum)

{

if( total == sum)

{

displaySubset(subSet, subSize);

subsetSum(set,subSet,n,subSize-1,total-set[nodeCount],nodeCount+1,sum);

return;

}

else

{

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

{

subSet[subSize] = set[i];

subsetSum(set,subSet,n,subSize+1,total+set[i],i+1,sum);

}

}

}

void findSubset(int set[], int size, int sum)

{

int \*subSet = new int[size];

subsetSum(set, subSet, size, 0, 0, 0, sum);

delete[] subSet;

}

int main()

{

int weights[30],target,size;

cout<<"Enter no. of weights :";

cin>>size;

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

{

cout<<"Enter weight "<<i+1<<": ";

cin>>weights[i];

}

cout<<"Enter the target sum :";

cin>>target;

cout<<"\nSubSets { }"<<endl;

findSubset(weights, size, target);

return 0;

}

/\*

OUTPUT:

Enter no. of weights :5

Enter weight 1: 10

Enter weight 2: 5

Enter weight 3: 15

Enter weight 4: 7

Enter weight 5: 20

Enter the target sum :2

SubSets { }

Enter no. of weights :5

Enter weight 1: 10

Enter weight 2: 7

Enter weight 3: 5

Enter weight 4: 15

Enter weight 5: 20

Enter the target sum :20

SubSets { }

{5,15,}

{20,}

\*/

**ADA 7: Merge sort two array n obtain median**

#include<stdio.h>

#include<stdlib.h>

#include<math.h>

int median(int arr[], int n)

{

if (n%2 == 0)

return (arr[n/2] + arr[n/2-1])/2;

else

return arr[n/2];

}

int max(int a,int b)

{

if(a>b)

return a;

else

return b;

}

int min(int a,int b)

{

if(b>a)

return a;

else

return b;

}

int getMedian(int ar1[], int ar2[], int n)

{

if (n <= 0)

return -1;

if (n == 1)

return (ar1[0] + ar2[0])/2;

if (n == 2)

return (max(ar1[0], ar2[0]) + min(ar1[1], ar2[1])) / 2;

int m1 = median(ar1, n);

int m2 = median(ar2, n);

if (m1 == m2)

return m1;

if (m1 < m2)

{

if (n % 2 == 0)

return getMedian(ar1 + n/2 - 1, ar2, n - n/2 +1);

return getMedian(ar1 + n/2, ar2, n - n/2);

}

if (n % 2 == 0)

return getMedian(ar2 + n/2 - 1, ar1, n - n/2 + 1);

return getMedian(ar2 + n/2, ar1, n - n/2);

}

int main()

{

int ar1[] = {1,12,15,26,38};

int ar2[] = {2,13,17,30,45};

int n1 = sizeof(ar1)/sizeof(ar1[0]);

int n2 = sizeof(ar2)/sizeof(ar2[0]);

if (n1 == n2)

printf("Median is %d", getMedian(ar1, ar2, n1));

else

printf("Doesn't work for arrays of unequal size");

return 0;

}

/\*OUTPUT:

Median is 16

\*/

**ADA 8: DFS/BFS shortest distance between source and destination (moving up, down, left, right only)**

#include <bits/stdc++.h>

using namespace std;

#define N 4

#define M 4

class QItem

{

public:

int row;

int col;

int dist;

QItem(int x, int y, int w)

: row(x), col(y), dist(w)

{

}

};

int minDistance(char grid[N][M])

{

QItem source(0, 0, 0);

bool visited[N][M];

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

{

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

{

if (grid[i][j] == '0')

visited[i][j] = true;

else

visited[i][j] = false;

if (grid[i][j] == 's')

{

source.row = i;

source.col = j;

}

}

}

queue<QItem> q;

q.push(source);

visited[source.row][source.col] = true;

while (!q.empty())

{

QItem p = q.front();

q.pop();

if (grid[p.row][p.col] == 'd')

return p.dist;

if (p.row - 1 >= 0 &&visited[p.row - 1][p.col] == false)

{

q.push(QItem(p.row - 1, p.col, p.dist + 1));

visited[p.row - 1][p.col] = true;

}

if (p.row + 1 < N &&visited[p.row + 1][p.col] == false)

{

q.push(QItem(p.row + 1, p.col, p.dist + 1));

visited[p.row + 1][p.col] = true;

}

if (p.col - 1 >= 0 &&visited[p.row][p.col - 1] == false)

{

q.push(QItem(p.row, p.col - 1, p.dist + 1));

visited[p.row][p.col - 1] = true;

}

if (p.col + 1 < M &&visited[p.row][p.col + 1] == false)

{

q.push(QItem(p.row, p.col + 1, p.dist + 1));

visited[p.row][p.col + 1] = true;

}

}

return -1;

}

int main()

{

char grid[N][M];

cout<<"Enter graph data in matrix form:\n";

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

{

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

{

cin>>grid[i][j];

}

}

cout <<"shortest distance between two cells:"<<minDistance(grid);

return 0;

}

/\*OUTPUT:

Enter graph data in matrix form:

0 \* 0 s

\* 0 \* \*

0 \* \* \*

d \* \* \*

shortest distance between two cells:6

Enter graph data in matrix form:

0 \* 0 s

\* 0 \* \*

0 \* \* \*

d 0 0 0

shortest distance between two cells: -1

\*/

**ADA 9:**

**9a:Search in sorted n rotated array**

#include<iostream>

#include<time.h>

using namespace std;

int findOffset(int \*arr, int low, int high, int size)

{

int mid;

if (low <= high)

{

mid = low + ((high - low) / 2);

if (arr[mid] > arr[mid+1])

return mid;

else if (arr[mid] > arr[size-1])

return findOffset(arr, mid+1, high, size);

else

return findOffset(arr, low, mid-1, size);

}

return -1;

}

int binarySearch(int \*arr, int low, int high, int item)

{

int mid;

if (low <= high)

{

mid = low + ((high - low) / 2);

if (arr[mid] == item)

return mid;

else if (arr[mid] > item)

return binarySearch(arr, low, mid-1, item);

else

return binarySearch(arr, mid+1, high, item);

}

return -1;

}

int main()

{

int i, size, offset, item, index, arr[100];

cout<<"Enter size of array: "<<endl;

cin>>size;

cout<<"Enter elements in array: "<<endl;

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

cin>>arr[i];

cout<<"Enter search item: "<<endl;

cin>>item;

offset = findOffset(arr, 0, size-1, size);

if (offset == -1)

{

index = binarySearch(arr, 0, size-1, item);

if(index != -1)

cout<<"Item found at index "<<index<< endl;

else

cout<<"Item not found"<<endl;

}

else

{

index = binarySearch(arr, 0, offset, item);

if (index != -1)

cout<<"Item found at index"<<index<<endl;

else

{

index = binarySearch(arr, offset+1, size-1, item);

if (index != -1)

cout<<"Item found at index: "<<index;

else

cout<<"Item not found. "<<endl;

}

}

return 0;

}

/\*

OUTPUT:

Enter size of array: 5

Enter elements in array: 9 10 6 7 8

Enter search item: 7

Item found at index: 3

Enter size of array: 4

Enter elements in array: 3 4 1 2

Enter search item: 5

Item not found

\*/

**9b: First, last occurrence and count**

#include<iostream>

using namespace std;

int binarySearch(int arr[], int l, int r, int x)

{

if (r < l)

return -1;

int mid = l + (r - l) / 2;

if (arr[mid] == x)

return mid;

if (arr[mid] > x)

return binarySearch(arr, l, mid - 1, x);

return binarySearch(arr, mid + 1, r, x);

}

int countOccurrences(int arr[], int n, int x)

{

int ind = binarySearch(arr, 0, n - 1, x);

if (ind == -1)

return 0;

int count = 1;

int left = ind - 1;

while (left >= 0 && arr[left] == x)

count++, left--;

int right = ind + 1;

while (right < n && arr[right] == x)

count++, right++;

return count;

}

void findFirstAndLast(int arr[], int n, int x)

{

int first = -1, last = -1;

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

{

if (x != arr[i])

continue;

if (first == -1)

first = i;

last = i;

}

if (first != -1)

cout << "First Occurrence = " << first << " Last Occurrence = " << last;

else

cout << "Not Found";

}

int main()

{

int i, n, offset, x, index, arr[100];

cout<<"Enter size of array: "<<endl;

cin>>n;

cout<<"Enter elements in array: "<<endl;

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

{

cin>>arr[i];

}

cout<<"Enter search item: "<<endl;

cin>>x;

findFirstAndLast(arr, n, x);

cout <<" count = "<<countOccurrences(arr, n, x)<<endl;

return 0;

}

/\*

OUTPUT:

Enter size of array:

5

Enter elements in array:

1 1 2 3 4

Enter search item:

5

First Occurrence = -1 Last Occurrence = -1 count = 0

|  |
| --- |
|  |
|  |  |
|  |  |
|  |  |

**ADA 10: Back tracking Rat in Maze**

#include<bits/stdc++.h>

#define size 4

using namespace std;

int solveMaze(int sR, int sC, int maze[size][size], int soln[size][size])

{

if ((sR == size - 1) && (sC == size - 1))

{

soln[sR][sC] = 1;

return 1;

}

else

{

soln[sR][sC] = 1;

if ((sC < size - 1) && maze[sR][sC + 1] == 1 && solveMaze(sR, sC + 1, maze, soln))

{

return 1;

}

if ((sR < size - 1) && maze[sR + 1][sC] == 1 && solveMaze(sR + 1, sC, maze, soln))

{

return 1;

}

soln[sR][sC] = 0;

return 0;

}

}

int main()

{

int maze[size][size];

cout<<"Enter the maze values"<<endl;

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

{

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

{

cin>>maze[i][j];

}

}

int soln[size][size] = {0};

int sR = 0;

int sC = 0;

if(solveMaze(sR, sC, maze, soln))

{

cout<<"Solution is: "<<endl;

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

{

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

{

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

}

cout <<endl;

}

}

else

{

cout<<"No Solution"<<endl;

}

return 0;

}

/\*

OUTPUT:

Enter the maze values

1 0 0 0

1 1 0 1

0 1 0 0

1 1 1 1

Solution is:

1 0 0 0

1 1 0 0

0 1 0 0

0 1 1 1

\*/