0827CS181120

Mohit Gite

A.D.A Assignment

N-Queens Problem

```
#include<stdio.h>
#include<math.h>

char a[10][10]; int n;
Void print()
{ int i,j;
  Printf("\n")
  for(i=0;i<n;i++)
  { for(j=0;j<n;j++)
    printf("%c",a[i][j]);
  }}
int markedcol(int row){
    int i;
  for(i=0;i<n;i++)
  if(a[row][i]='0')
  { return(i);
   break;
  }}
int feasible(int row, int col)
  { int i, tcol;
  for(i=0;i<n;i++)
  {
    tcol = markedcol(i);
    if(col==tcol || abs(row-i)=abs(col-tcol)
     return 0;
    }
    return 1;
  }
```

```
Void nqueen(int row)
{ int i,j;
if (row<n)
{ for(i=0; i<n; i++)
{ if (feasible(row,i))
{ a[row][i] = 'Q';
nqueen(row+1);
a[row][i] = '.'; }}}
else { print(); }}
int main(){
int i,j;
Printf("enter no. of queen");
scanf("%d",&n);
for(i=0; i<n; i++)
for(j=0; j<n; j++)
a[i][j] = '.';
nqueen(0);
return(0);
}
```

Graph Colouring

```
#include <stdio.h>
int G[50][50], x[50];
int nextColumn(int k)
{ int i, j;
  x[k] = 1;
  for (i=0; i<k; i++)
  { if (G[i][k] != 0 && x[k] == x[i])
      x[k] = x[i] + 1; }}

int main() {
  int n, e, i, j, k, l;
  printf("enter no. of v");
  scanf("%d", &n);
  printf("enter no of e");
  scanf("%d", &e);
  for(i=0; i<n; i++)
    for(j=0; j<n; j++)
      G[i][j] = 0;
  printf("enter Value");
  for(i=0; i<e; i++)
  { scanf("%d", &k, &l);
    G[k][l] = 1;
    G[l][k] = 1;
  }
for (i=0; i<n; i++)
  nextColumn(i);
  printf("Column of v")
    for (i=0; i<n; i++)
      printf("Vertex[%d], i+1, x[i])
  return 0;
}
```

fibonacci series

Bottom's up

```
#include <iostream.h>
int fib(int N)
{
    int fib[N+1];
    fib[0]=0;
    fib[1]=1;
    for(i=2; i<=N; i++)
        fib[i]=fib[i-1]+fib[i-2];
        return fib[N];
}
int main()
{   int n;
    Scanf("%d",&n);
    if(n<=1)
        Printf(n);
    else
        Printf(fib(n));
    Return 0;
}
```

Top Down

```
int fib(int n)
{   if(n<=1)
        return n;
    return fib(n-1)+fib(n-2);
}   int main()
{   int n;
    Scanf("%d",&n);
    Printf(fib(n));
    Return 0;
}
```

Factorial

```
#include <iostream>
using namespace std;
int result[1000] = {0};
int fact(int n) {
  if (n>=0) {
    result[0] = 1;
  for (int i = 1; i<=n; i++)
  { result[i] = i + result[i-1];
  }
  return result[n]; }}
int main()
{
  int n;
  while (1) {
  cout << " enter no";
  cin>> n;
  if (n==0)
    break;
   cout << fact(n);
  }}
```

Challenging Problem

Binary Search

```
#include <iostream>
Using namespace std;
int search (int l, int r, int Key, int ar[])
 {
    While (r >= l).
  { int m1 = l + (r-l)/3;
    int m2 = r - (r-l)/3;
    if (ar[m1] == Key ){
          return m1; }
    if (a[m2] == Key)
       { return m2; }
    if ( Key < ar[m1]){
          r = m1 - 1; }
    return -1;
  }
  int main().
 {
     int l, r, p, Key;
     int ar[] = {1,2,3,4,5,6,7,8,9,10};
      l=0;
      r= 9;
      Key = 5;
      p= search (l, r, Key, ar);
        Cout << p;
 }
```

Job Scheduling

```
#include <iostream>
#include <algorithm>
Using namespace std;
Struct job {
  char id;
  int dead;
  int profit;
};
  bool comp(job a, job b)
  {
    return (a.profit > b.profit);
  }
  Void print(Job arr[], int n)
  {
    sort (arr, arr+n, comp);
      int result[n];
      bool slot[n];
  for (int i=0; i<n; i++)
    slot[i] = false;
  for (int i=0; i<n; i++)
    {
      for(int j=min(n,arr[i].dead-1; j>=0)
    {
  if (slot[j] == false)
    {  result[j] = i;
       slot[j] = true;
       break; } } }
int main() { Job arr = { {'a',2,100}, {'b',1,1}
                       , {'c',8,27} }
  int n = sizeof(arr) / size of(arr[0]);
  cout << " sequence of max. profit ; Print(arr
```

Optimal Merge

```
import java.util.Scanner
import java.util.PriorityQueue;
Public class Merge{

  Static int minCom(int size, int files[])
   {
      PriorityQueue<Integer> pq = new PriorityQue
                                  <>();
      for (int i=0; i<size; i++)
        { pq.add(files[i]); }
     int Count=0;
      while(pq.size()>1){
      int temp = pq.poll+ pq.pull();
        Count +=temp;
        pq.add(temp); }
       return count;
      }
      Public Static void main (String...args)
     {   int size=6;
         int files[] = new int[]{1,3,5,7,9,13};
     S.o.P("optimal M.C" =+ minCom(size, files
          }}
```

Hamilton Cycle

```
#include<bits/stdc++.h>
using namespace std;
#define V5
void print(int path[]);
bool isSafe(int v, bool graph[V][V] ==0)
    return false;
for(int i=0; i<pos; i++)
    if(path[i]==v)
        return false;
    return true;
}
bool hamCycle(bool graph[v][v], intpath[]
                                , int pos)
{
    if(pos==V)
    { if(graph[path[pos-1]][path[0]]==1]
        return true;
        else
            return false;
    }
    ham Cycle()
    for(int V=1; v<V.; v++)
{ if(issafe(v, graph, path, pos))
    { path[pos]=v;
        if(ham Cycle util(graph, path, pos+1)
                                == true)
            Return true;
            path[pos]=-1;}}
    return false; }
```

```
bool hamcycle (bool graph[v][v])
{
    int * path = new int[v];
    for (int i=0; i< v; i++)
        path[i] = -1;
        Path [0] =0;

Void print (int path[])
    Cout<< " Soln exists ";
    for (int i =0; i<V; i++)
        Cout << path[i] << path[0];
}
int main()
{
    bool graph[V][x] = { {0,1,0,1,0 }
                         { 1,0,1,1}
                        { 0,1,0,0, 1}
                        {0,1,1,1,0}};
                    hamCycle (graph);
    bool graph2 [v][v] = {{0,1,0,1,0}
                          {1,0,1,1}
                          { 1,1,0,1,0}};
    hamCycle (graph2);
    return 0;
}
```

Huffmann Code

```
#include<iostream>
Using namespace std;
#define MAX_TREE_HT50

Struct MinHeapNode
{
    char data;
    Unsigned freqency;
    Struct MinHeapnode, *left, *right;
};
Struct MinHeap Node * new Node (char data,
                                Unsigned freqency)

{ Struct MinHeapnode *temp =(Struct MinHeapNode
                             * malloc (sizeof
                             (Struct MinHeapNode))
    temp->left = temp->right = NULL;

Struct MinHeap *Create MinHeap (Unsigned
                                capacity.)
{ struct MinHeap * minHeap = (struct min Heap
                              malloc (sizeof(
                              struct Min He

    min Heap->size =0;
    minHeap -> Capacity = Capacity;
    min Heap -> array =
    return min Heap;
}
```

```
struct MinHeap *createMinHeap(unsigned capacity)
{
struct MinHeap *minHeap = (struct MinHeap *)
                          malloc(sizeof(struct MinHeap));
minHeap->size = 0;
minHeap->capacity = capacity;
minHeap->array = (struct MinHeapNode **)
                 malloc(minHeap->capacity));
return minHeap;
}

void swapMinHeapNode(struct MinHeapNode
     **a, struct **b)
{ struct MinHeapNode *t = *a;
  *a = *b;
  *b = *t;
}

void minHeapify(struct MinHeap *minHeap, int idx)
{
  int smallest = idx;
  int l = 2*idx+1;
  int r = 2*idx+2;

if (left < minHeap->size && minHeap->array[left]->
      frequency < minHeap->array[smallest]->freq
  smallest left;
else if
      smallest right;
```

```
if (smallest != idx)
{
    swapMinHeapNode (&minHeap->array[smallest],
                     minHeap->array[idx]);
    minHeapify (minHeap, smallest);
}
}

void insertMinHeap (* minHeap, * minHeapNode)
{
    ++minHeap->size;
    int i = minHeap->size - 1;

while (i && minHeapNode->frequency < minHeap->
       array [(i-1)/2]->frequency)
    {
        minHeap->array[i] = MinHeapNode;
    }

void print (int a[], int n)
{
    for (i=0; i<n; i++)
        cout << arr[i];
}

int isLeaf (struct MinHeapNode * root)

return !(root->left) && !(root->right);
```

```
struct MinHeapNode * buildHT (char data[], char
                              frequency[], int size)
{
    struct MinHeapNode *left, *right, *top;

    while (1 in size and (minHeap))
    {
        left = extractMin(minHeap);
        right = extractMin(minHeap);

        top->left = left;
        top->right = right;
    }

void HuffmanCodes (char data[], int frequency[],
                   int size)
{
    struct MinHeapNode * root = buildHT(data, frequency, size);

    int arr[MAX_TREE_HT], top = 0;

    printCodes(root, arr, top);
}

int main()
{
    char arr[] = {'A', 'B', 'C', 'D'};
    int frequency[] = {5, 9, 1, 3};

    int size = sizeof(arr) / sizeof(arr[0]);
    HuffmanCodes(arr, frequency, size);
}
```

Breadth first traversal

```
#include <iostream>
using namespace std;

int a[20][20], q[20], visited[20], n, i, j, f=0, r=-1;

void bfs(int v) {
  for(i=1; i<=n; i++)
  if (a[v][i] && !visited[i])
      q[++r]=i;
  if (f<=r) {
      visited[q[f]] = 1;
      bfs(q[f++]);     }}

void main()
{
    int v;
    cout<<"enter no of v";
    cin>>n;

    for(i=1; i<=n; i++)
    {
        q[i] = 0;
        visited[i] = 0;
    }
cout<<"enter graph data in matrix";
```

```
for(i=1; i<=n; i++)
{ for(j=1, j<=n; j++){
        cin>>a[i][j] ;
}}

cout<<" enter starting v";
  cin>>k;
  bfs(v);

cout<<" nodes acceptable are";
 for(i=1; i<=n; i++){
      if (visited[i])
        cout<<i;
    else
      cout<<" Not possible";
      break;
  }
}
```

Depth first search

```
#include <iostream>

void DFS (int);
int G[10][10], visited[10], n;

void main()
{
  int i, j;
  cout << "Enter no. of vertex";
  cin >> n;
  cout << "Enter matrix of graph";

  for (i=0; i<=n; i++)
      for (j=0; j<n; j++)
          cin >> G[i][j];

  for (i=0; i<n; i++)
          visited[i] = 0;
  DFS(0); }

void DFS (int i)
{ int j;
  cout << i;
      visited[i] = 1;
for (j=0; j<n; j++)
      if (!visited[j] && G[i][j]==1)
          DFS(j); }
```

Travelling Salesman

```
#include <bits/stdc++.h>
using namespace std;
#define V 4

int travelSP(int graph[][V], int s)
{
    vector<int> vertex;
    for (int i = 0; i < V; i++)
        if (i != s)
            vertex.push_back(i);
    int min_path = INT_MAX;
    do {
        int current_pw = 0;
        int k = s;
        for (int i = 0; i < vertex.size(); i++)
        {
            current_pw += graph[k][vertex[i]];
            k = vertex[i];
        }
        current_pw += graph[k][s];

        min_path = min(min_path, current_pw);
    } while (next_p(vertex.begin(), vertex.end()));

    return min_path;
}
```

```
int main()
{
    int graph[V][V] = { { 0, 10, 15, 20 },
                        { 10, 0, 35, 25 },
                        { 20, 25, 30, 0 } };

    int s = 0;

    cout << "Prim's MST is \n";
    primMST(graph, s);

    return 0;
}
```