Nafeez Shaikh
0821CS181125

Date / / Page no. 1

N-Queen's Program

```
#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\t",a[i][j]);
}}

int markedcol(int row){
   int i;
for(i=0;i<n;i++)
   if(a[row][i]=='Q') {return(i);
      break;
}}

int feasible(int row, int col){
   int i, tcol;
   for(i=0;i<n;i++){
      tcol = getmarkedcol(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 Gr[50][50], x[50];
int next colour (int k) {
int i, j;
x[k] = i;
for (i = 0; i < k; i++) {
if (Gr[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++)
Gr[i][j] = 0;

printf ("Enter value");
for (i = 0; i < e; i++)
{
scanf ("%d %d", &k, &l);
Gr[k][l] = 1;
Gr[l][k] = 1;
}
```

```
for (i=0; i<n; i++)
    next colour (i);

printf ("Colour of v");
for (i=0; i<n; i++)
    printf ("Vertex [%d] : %d \n", i+1, x[i]);

return 0;
}
```

Fibonacci series

Bottom up.

```
#include<iostream.h>

int fib (int N)
{
   int Fib [N+1], i;
    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);
}
    intmain ()
{
    int n;
    scanf ("%d", &n);
    printf ( fib(n));
    return 0;
}
```

Date: / / Page no: 8

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<<"\n Enter no";
  cin>>n;
   if (n==0)
       break;
     cout<<fact(n);
}
}
```

#Challenging Problem

Ternary 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; }
   else if (key > ar[m2]) {
        l = m2+1; }
   else {
        l = m1+1;
        l = m2-1; } }
```

```
    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;
        key = 50;
}
```

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;
                      j--) {
```

```
if (slot[j] == false)
{
    result[j] = i;
    slot[j] = true;
    break;  }}

for (int i=0; i<n; i++)
  { if (slot[i])
        cout << arr[result[i]].id;
  }

int main()
{
    Job arr = {{'a', 2, 100}, {'b', 1, 19}, {'c', 2, 27}};

    int n = size of (arr) / size of (arr[0]);
    cout << " Sequence of max. profit is";
      print (arr, n);

      return 0;
}
```

Optimal Merge.

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

static int minCom(int size, int files[])
{
    PriorityQueue<Integer> pq = new PriorityQueue<>();
for(int i=0; i<size; i++) {

    pq.add(files[i]);
}
  int count = 0;
  while (pq.size() >1) {

int temp = pq.poll() + pq.poll();
    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};
System.out.println("Optimal M.C" = + minCom(size, files));
}}
```

Hamiltonian Cycle.

```
#include <bits/stdc++.h>
using namespace std;
#define V 5

void print(int path[]);
bool isSafe(int v, bool graph[V][V], int path[], int pos)
{
    if (graph[ path[pos-1] ][v] == 0)
        return false;

    for (int i=0; i<pos; i++)
        if (path[i] == v)
            return false;
    return true;
}
bool hamCycleUtil(bool graph[V][V], int path[], int pos)
{
    if (pos == V)
    {
        if(graph[path[pos-1]][path[0]] == 1)
        return true;
        else
            return false;
    }

hamCycle()
  for(int v = 1; v < V; v++)
```

```
{ if (isSafe (v, graph, path, pos))
  {
    path[pos] = v;
    if (hamCycleUtil (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;

if (hamCycle (graph, path, 1) == false)
{
  cout << "Solu doesn't exist";
      return false; }
  print (path);
    return true;
}

void print ( int path[] )
{
    cout << " Soln exists".
  for (int i = 0; i < V; i++)
```

```
        cout << path[i] << path[0];
    }

int main()
{
    bool graph1[V][V] = {{0,1,0,1,0},
                         {1,0,1,1,1},
                         {0,1,0,0,1},
                         {0,1,1,1,0}};
    hamCycle(graph1);

    bool graph2[V][V] = {{0,1,0,1,0},
                         {1,0,1,1,1},
                         {1,1,0,1,0}};
    hamCycle(graph2);
    return 0;
}
```

Huffman Code

```
#include <iostream>
using namespace std;
#define MAX_TREE_HT 50

struct MinHeapNode
{
    char data;
    unsigned frequency;
    struct MinHeapNode *left, *right;
};

struct MinHeap
{
    unsigned size, capacity;
    char data;
    struct MinHeapNode **array;
};

struct MinHeapNode *newNode(char data, unsigned frequency)
{
struct MinHeapNode *temp = (struct MinHeapNode *) malloc(sizeof
                           (struct MinHeapNode));
    temp->left = temp->right = NULL;
    temp->data = data;
    temp->frequency = frequency;
    return temp;
}
```

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

    while (!isSizeOne(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, 6, 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<<"\n Enter no. of v\n";
  cin>>n;

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

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

cout<<"Enter starting v";
    cin>>v;
    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 a[10][10], visited[10], n;

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

for (i=0; i<=n; i++)
      for (j=0; j<n; j++)
        cin >> a[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] && a[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 currentpw = 0;
        int k = s;
        for (int i=0; i<vertex.size(); i++)
        {
            currentpw += graph[k][vertex[i]];
            k = vertex[i];
        }
        currentpw += graph[k][s];

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

    return min_path;
}
```

Date / / Page no. 25

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

    int s = 0;

    cout << travelSP(graph, s);

    return 0;
}
```