DS Slips-Solutions(sem 4)

Bsc (computer science) (Savitribai Phule Pune University)

Slip - [1,2,3,6,18,23]

Q1) adjacency matrix(create & display matrix)

```
#include<stdio.h>
int main()
{
  int a[10][10],i,j,n;
  printf("\n Enter total no. of vertex: ");
  scanf("%d",&n);
  printf("\n**PRESS 1 FOR YES & 0 FOR NO**\n");
  for(i=0;i<n;i++)
  {
    for(j=0;j<n;j++)
   {
     a[i][j]=0;
     if(i!=j)
    {
     printf("\nIs there edge between %d & %d: ",i+1,j+1);
     scanf("%d",&a[i][j]);
      }
    }
  }
}
printf("\n Matrix is:\n");
for(i=0;i<n;i++)
 {
   for(j=0;j<n;j++)
  {
    printf("%d\t",a[i][j]);
  }
   printf("\n");
 }
}
```

Slip - [1,3,6,8,15,18,21,23,24]

Q.2) Prim's Minimum spanning tree algorithm.

Ans:

```
    #include<stdio.h>
    int cost[7][7]={ {0,5,3,999,999,999,999},
    {5,0,4,6,2,999,999},{3,4,0,5,999,6,999},
    {999,6,5,0,8,6,999},{999,2,999,8,0,3,5},
    {999,999,6,6,3,0,4}, {999,999,999,999,5,4,0}};
    int n=7;
```

```
void main()
{
   int a,b,u,v,i,j,e;
   int visited[10]={0},min,mincost=0;
   visited[0]=1;
   printf("\n");
   for(e=0;e<n;e++)
   {
      for(i=0,min=999;i<n;i++)
      for(j=0;j<n;j++)
      {
         if(cost[i][j]==0)
         cost[i][j]=999;
         if(cost[i][j]<min)
         if(visited[i]!=0)
         {
            min=cost[i][j];
            a=u=i;
            b=v=j;
         }
      }
     if(visited[u]==0 || visited[v]==0)
     {
        printf("\nedge %d:(%d %d)cost:
        %d",e+1,a+1,b+1,min);
        mincost+=min;
        visited[b]=1;
      }
    cost[a][b]=cost[b][a]=999;
  }
  printf("\nMinimun cost=%d",mincost);
}
```

Slip - [1,9,10,12,18,19,22]

Q.3) Preorder, inorder,postorder.

```
#include<studio.h>
#include<stdlib.h>
Typedef struct
{
  int info;
  struct node *left, *right;
}NODE;
```

```
NODE *create();
NODE *insert(NODE * ,int)
NODE *preorder(NODE *);
NODE *inorder(NODE *);
NODE *postorder(NODE *);

NODE * create()
{
   int i,n,x;
   NODE * root;
   root= NULL;
   printf("\n enter the no.of nodes");
   scanf("%d",&n);
   for(i=0,i<n,i++)
   {
       Printf("\n Enter the node: ");
       scanf("%d",&x);
       root=insert(root,x);
   }
return(root);
}
NODE *insert(NODE *T,int x)
{
  NODE *r;
  if(T==NULL)
  {
    r=(NODE *)malloc(sizeof(NODE));
    r->left=r->right=NULL;
    r->info=x;
    return(x);
  }
  else
  if(x<T->info)
  {
     T->left=insert(T->left,x);
     return(T);
  }
  if(x>T->info)
  {

     T->right=insert(T->right,x);
     return(T);
  }
```

```
   else
   printf("\n Duplicate value!!\n");
   return(T);
}

Void preorder(NODE *T)
{
    if(T!=NULL)
    {
        printf("%d\t",T->info);
        preorder(T->left);
        preorder(T->right);
    }
}

Void inorder(NODE *T)
{
    if(T!=NULL)
    {
        inorder(T->left);
        printf("%d\t",T->info);
        inorder(T->right);
    }
}

Void postorder(NODE *T)
{
    if(T!=NULL)
    {
        postorder(T->left);
        postorder(T->right);
        printf("%d\t",T->info);
    }
}

int main()
{
    NODE *root;
    root=create();
    preorder(root);
    inorder(root);
    postorder(root);
}
```

Slip - [3,7,11,14,16,23,25]

Q.4) Floyd warshall's algorithm.

```
#include <stdio.h>
#define nV 4
#define INF 999
void printMatrix(int matrix[][nV]);
(int graph[][nV])
{
     int matrix[nV][nV], i, j, k;
     for (i = 0; i < nV; i++)
     for (j = 0; j < nV; j++)
     matrix[i][j] = graph[i][j];
     for (k = 0; k < nV; k++)
    {
      for (i = 0; i < nV; i++)
     {
        for (j = 0; j < nV; j++)
       {
          if (matrix[i][k] + matrix[k][j] < matrix[i][j])
          matrix[i][j] = matrix[i][k] + matrix[k][j];
       }
    }
 }
printMatrix(matrix);
}
void printMatrix(int matrix[][nV])
{
      for (int i = 0; i < nV; i++)
     {
       for (int j = 0; j < nV; j++)
      {
        if (matrix[i][j] == INF)
          printf("%4s", "INF");
        else
          printf("%4d", matrix[i][j]);
       }
       printf("\n");
     }
}
int main()
 {
     int graph[nV][nV] = {{0, 3, INF, 5},{2, 0, INF, 4},
```

```
                                {INF, 1, 0, INF},{INF, INF, 2,0}};
floydWarshall(graph);
}
```

Slip - [2,4,12]

Q.5) Topological sort.

```
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>
struct Stack
{
    int data;
    struct Stack* next;
};
struct Graph
 {
    int V;
    struct List* adj;
};
struct List
{
    int data;
    struct List* next;
 };
struct Stack* createStackNode(int data)
{
    struct Stack* newNode = (struct Stack*)malloc
    (sizeof(struct Stack));
    newNode->data = data;
    newNode->next = NULL;
    return newNode;
}
struct List* createListNode(int data)
{
    struct List* newNode = (struct List*)malloc
    (sizeof(struct List));
    newNode->data = data;
    newNode->next = NULL;
    L return newNode;
}
struct Graph* createGraph(int V)
{
     struct Graph* graph = (struct Graph*)malloc
```

```
    (sizeof(struct Graph));
    graph->V = V;
    graph->adj = (struct List*)malloc(V * sizeof(struct
    List));
    for (int i = 0; i < V; ++i)
    {
      graph->adj[i].next = NULL;
    }
  return graph;
}
void addEdge(struct Graph* graph, int v, int w)
{
     struct List* newNode = createListNode(w);
     newNode->next = graph->adj[v].next;
     graph->adj[v].next = newNode;
}
void topologicalSortUtil(struct Graph* graph, int v, bool visited[], struct Stack** stack)
{
     visited[v] = true;
     struct List* current = graph->adj[v].next;
     while (current != NULL)
    {
        int adjacentVertex = current->data;
        if (!visited[adjacentVertex])
       {
          topologicalSort
          Util(graph, adjacentVertex, visited, stack);
        }
    current = current->next;
    }
struct Stack* newNode = createStackNode(v); newNode->next = *stack;
*stack = newNode;
 void topologicalSort(struct Graph* graph)
{
     struct Stack* stack = NULL;
     bool* visited = (bool*)malloc(graph->V * sizeof
     (bool));
     for (int i = 0; i < graph->V; ++i)
     {
        visited[i] = false;
     }
    for (int i = 0; i < graph->V; ++i)
    {
```

```
        if (!visited[i])
        {
          topologicalSortUtil(graph, i, visited, &stack);
        }
     }
    while (stack != NULL)
    {
        printf("%d ", stack->data);
        struct Stack* temp = stack;
        stack = stack->next;
        free(temp);
    }
    free(visited);
    free(graph->adj);
    free(graph);
}
int main()
{
     struct Graph* g = createGraph(6);
     addEdge(g, 5, 2);
     addEdge(g, 5, 0);
     addEdge(g, 4, 0);
     addEdge(g, 4, 1);
     addEdge(g, 2, 3);
     addEdge(g, 3, 1);
     printf("Topological Sorting Order: ");
     topologicalSort(g); return 0;
}
```

Slip - [4,9,10,11,19,22]

Q.6) Adjacency list.

```
#include<stdio.h>
#include<stdlib.h>
typedef struct node
{
    int vertex;
    struct node *next;
}NODE;

NODE *list[10];

int create(int a[10][10],int n)
{
```

```
   for(int i=0;i<n;i++)
   {
     for(j=0;j<n;j++)
     {
       a[i][j]=0;
       if(i!=j)
       {
         printf("\nIs there edge between %d & %d: ",          i+1,j+1);
         scanf("%d",&a[i][j]);
       }
     }
   }
}

void clist(int a[10][10],int n)
{
   NODE *temp, *newnode;
   for(int i=0;i<n;i++)
   {
     list[i]=NULL;
     for(int j=0;i<n;j++)
     {
       if(a[i][j]==1)
       {
          newnode=(NODE *)malloc(sizeof(NODE));
          newnode->vertex=j+1;
          newnode->next=NULL;
          if(list[i]==NULL)
            list[i]=temp=newnode;
         else
            temp->next=newnode;
            temp=newnode;
       }
     }
   }
}

void dlist (int n)
{
  NODE *temp;
  printf("\n the adjacency list is: ");
  for(int i=0;i<n;i++)
  {
```

```
      printf("v%d->",i+1)  ;
      temp=list[i];
      while(temp)
      {
        printf("v%d->",temp->vertex);
        temp=tenp->next;
      }
      printf("NULL\n");
  }
}

int main()
{
   int a[10][10],n;
   printf("\n Enter the no.of vertex: ");
   scanf("%d",&n);
   create(a,n);
   clist(a,n);
   dlist(n);
}
```

Slip - [5,7,8,16,17,24,25]

Q.7) Heapsort method.

```
#include<stdio.h>
void display(int a[],int n)
{
   int i;
   for(i=0;i<n;i++)
   printf("%d\t",a[i]);
}
void Heapify(int A[], int top, int last)
{
   int j,temp,key;
   key = A[top];
   j = 2*top+1;
   if((j<last) && (A[j]<A[j+1]))
   j=j+1;
   if((j<=last) && (key<A[j]))
  {
     temp = A[top];
    A[top] = A[j];
    A[j] = temp;
    Heapify(A,j,last);
```

```
  }
}
void BuildHeap(int A[],int n)
{
    int i;
    for(i=n/2-1; i>=0; i--)
    Heapify(A,i,n-1);
}
void HeapSort(int A[], int n)
{
    int i,temp,top=0,last;
    BuildHeap(A,n);
    printf("Initial heap=");
    display(A,n);
    for(last=n-1;last>=1;last--)
  {
      temp=A[top];
     A[top] = A[last];
     A[last] = temp;
     printf("\n after iteration %d :", n-last);
     display(A,n);
     Heapify(A,top,last-1);
  }
}
int main()
{
 intA[8]={26,5,77,1,61,11,59,15};
 HeapSort(A,8);
 printf("\nThe sorted elements are:");
 display(A,8);
 return 0;
}
```

Slip - [6,15,21]

Q.8) Dijkstra Shortest Path alogorithm.

```
      #include<stdio.h>
      int cost[6][6]={{0,7,9,999,999,14},
      {7,0,10,15,999,999},{9,10,0,4,999,2},
      {999,15,4,0,6,999},{999,999,999,6,0,9},
      {14,999,2,999,9,0}};
      void dj(int v,int n)
    {
      int dist[10],visited[10]={0};
```

```
    int i,j,u,w,count,min;
    visited[v]=1;
    for(i=0;i<n;i++)
   dist[i]=cost[v][i];
   count=2;
   while(count<n)
  {
      min=999;
      for(i=0;i<n;i++)
      if(visited[i]==0 && dist[i]<min)
     {
        min=dist[i];
        u=i;
      }
      visited[u]=1;
      for(w=0;w<n;w++)
      if(dist[u]+cost[u][w]<dist[w])
      dist[w]=dist[u]+cost[u][w];
     count++;
  }
   printf("\nShortest path from %d
   vertex are ",v+1);
  for(i=0;i<n;i++)
  printf("%d\t",dist[i]);
}
void main()
{
    dj(0,6);
}
```

Slip - [13,20]

Q.9) Kruskals algorithm.

```
#include<stdio.h>
typedef struct
{
 int src,dest,weight;
 }
edge;
edge graph[12]={1,2,5, 1,3,3, 2,3,4, 2,4,6, 2,5,2, 3,4,5, 3,6,6, 4,5,8, 4,6,6, 5,6,3, 5,7,5, 6,7,4};
int MSTvertices[10];
void sort(edge graph[],int nE)
{
  int i,pass;
```

```
  edge temp;
  for(pass=1; pass<=nE-1;pass++)
  for(i=0;i<nE-pass;i++)
if(graph[i].weight>graph[i+1].weight)
   {
     temp=graph[i];
     graph[i]=graph[i+1];
     graph[i+1]=temp;
   }
 }
 int find(int v,int nV)
 {
    for(int k=0;k<nV;k++)
     if(MSTvertices[k]==v)
      return 1;
    return 0;
  }
  int KruskalMST(int nV,int nE)
  {
    edge mst[14];
    int
i=1,j=2,k=1,count=1,mincost=0,first=0,second=0;
    sort(graph,nE);
    mst[0]=graph[0];
    MSTvertices[0]=graph[0].src;
    MSTvertices[1]=graph[0].dest;
    mincost=graph[0].weight;
    while(count<=nV-1)
    {
      first=find(graph[i].src,nV);
      second=find(graph[i].dest,nV);
      if(!(first&&second))
       {
         mst[k++]=graph[i];
         count++;
         mincost=mincost+graph[i].weight;
         if(first)
           MSTvertices[j++]=graph[i].dest;
         else
           MSTvertices[j++]=graph[i].src;
       }
       i++;
     }
```

```
      printf("The edges in the Minimum spanning tree are:");
      for(i=0;i<nV-1;i++)
printf("%d--%d==%d\n",mst[i].src,mst[i].dest,mst[i].weight);
       printf("Minimum cost of spanning tree:%d",mincost);
       }
       void main()
       {
         int nV=7,nE=12;
         KruskalMST(nV,nE);
       }
```

Slip - [4,13,20,21]

Q.10) Count Total nodes & total leaf nodes in tree.

```
#include<studio.h>
#include<stdlib.h>
Typedef struct
{
  int info;
  struct node *left, *right;
}NODE;

NODE *create();
NODE *insert(NODE * ,int)
NODE *countTN(NODE *);
NODE *countLN(NODE *);

NODE * create()
{
  int i,n,x;
  NODE * root;
  root= NULL;
  printf("\n enter the no.of nodes");
  scanf("%d",&n);
  for(i=0,i<n,i++)
   {
       Printf("\n Enter the node: ");
       scanf("%d",&x);
       root=insert(root,x);
   }
return(root);
}
NODE *insert(NODE *T,int x)
{
```

```
  NODE *r;
  if(T==NULL)
 {
  r=(NODE*)malloc(sizeof(NODE));
  r->left=r->right=NULL;
  r->info=x;
  return(x);
 }
 else
 if(x<T->info)
 {
    T->left=insert(T->left,x);
    return(T);
 }
 if(x>T->info)
 {
    T->right=insert(T->right,x);
    return(T);
 }
else
  Printf("\n Duplicate value!\n");
  return(T);
}

int countTN(NODE *T)
{
   if(T==NULL)
      return 0;
   else
return(1+countTN(T->left)+countTN(T->right));
}
int countLN(NODE *T)
{
  if(T==NULL)
     return 0;
  else
  if(T->left==NULL &&
     T->right==NULL)
  return 1;
  else
  return(countLN(T->left)+countLN
  (T->right));
}
```

```
int main()
{
  int key;
  NODE *root, *temp;
  root=create();
  printf("\n total no. of nodes:
  %d\n",countTN(root));
  printf("\n total no. of nodes:
  %d\n",countLN(root));
}
```

Slip - [5,7,14,16,17,25]

Q.11) Nodes At each level & count Node at each level & total levels in tree.

```
#include<stdio.h>
#include<stdlib.h>
typedef struct BSTnode
{
    int data;
    struct BSTnode *left,*right;
}BSTnode;

BSTnode *insert(BSTnode *T,int x)
{
    BSTnode *r;
    if(T==NULL).
   {
       r=(BSTnode*)malloc(sizeof(BSTnode));
       r->data=x;
      r->left=NULL;
      r->right=NULL;
      return(r);
   }
   if(x>T->data)
  {
      T->right=insert(T->right,x);
      return(T);
  }
  else
  if(x<T->data)
   {
       T->left=insert(T->left,x);
       return(T);
   }
```

```
    else
    return(T);
}
BSTnode *create()
{
    int n,x,i;
    BSTnode *root;
    root=NULL;
    printf("\n enter no. of nodes :");
    scanf("%d",&n);
    printf("\n Enter tree values :");
    for(i=0;i<n;i++)
    {
       scanf("%d",&x);
       root=insert(root,x);
    }
   return(root);
}

int NodesAtLevel(BSTnode *ptr, int level)
{
     if(ptr==NULL)
     return 0;
     if(level==0)return 1;
     return  NodesAtLevel(ptr->left,level-1)+
  NodesAtLevel(ptr->right,level-1);
}
int height(BSTnode *root)
{
     if (!root)
     return 0;
     else
     {
        int lheight = height(root-> left);
        int rheight = height(root ->
        right);
        if (lheight > rheight)
        return (lheight + 1);
        else return (rheight + 1);
     }
}
void Level(BSTnode* T, int level)
{
```

```
    if (T==NULL)
    return;
    if (level== 0)
   {
       printf("%d -> ", T->data);
   }
   else
  {
     Level(T->left, level - 1);
     Level(T->right, level - 1);
  }
}
void tree_level(BSTnode* root)
{
    if(root==NULL)
    return;
    int h = height(root);
    for(int i=0; i<h; i++)
   {
      printf("Level %d: ", i+1);
      Level(root, i);
      printf("\n");
   }
}
void main()
{
    BSTnode *root=NULL;
    int level;
    root=create();
    printf("Enter any level :: ");
    scanf("%d",&level);
    printf("\nNumber of nodes at
    level [ %d ] :: %d\n",level,Nodes
    AtLevel(root,level));
    tree_level(root);
    printf("\nTotal no. of levels:
            %d\n ",height(root));
}
```

Slip - [14,17]

Q.12) Hash Table(linear probing).

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

```
#include<stdlib.h>
#include<stdbool.h>
#define SIZE 20
struct DataItem
{
   int data;
   int key;
};
struct DataItem* hasharray[SIZE]
struct DataItem* dummyItem;
struct DataItem* item;
int hashcode(int key)
{
    return key % SIZE;
}

struct DataItem *search(int key)
{
   int hashIndex = hashcode(key);
   while(hasharray[hashIndex] !=
      NULL)
   {
      if(hasharray[hashIndex]->key
         == key)
      return hasharray[hashIndex];
      ++hashIndex;
      hashIndex %= SIZE;
   }
   return NULL;
}
void insert(int key,int data)
{
     struct DataItem *item =(struct
        DataItem*)
     malloc(sizeof(struct DataItem));
     item->data = data;
     item->key = key;
     int hashIndex = hashCode(key);
     while(hashArray[hashIndex] !=
        NULL && hasharray
     [hashIndex]->key !=key != -1)
     {
        ++hashIndex;
```

```
        hashIndex %= SIZE;
     }
     hashArray[hashIndex] = item;
}
struct DataItem* delete(struct DataItem* item)
{
    int key = item->key;
    int hashIndex = hashCode (key);
    while(hashArray[hashIndex] !=
    NULL)
   {
      if(hashArray[hashIndex]->
         key == key)
      {
         struct DataItem* temp =
            hasharray[hashIndex];
         hashArray[hashIndex] =
            dummyItem:
      }
      ++hashIndex;
      hashIndex %= SIZE;
   }
   return NULL;
}
void display()
{
   int i=0;
   for(i=0; i<SIZE; i++)
   {
      if(hashArray[i] != NULL)
         printf("(%d,%d",hashArray[i]
         ->key,hashArray[i]->data);
     else
         printf("~~");
   }
   printf("\n");
}
int main()
{
    dummyItem = (struct DataItem*)
       malloc(sizeof(structItem));
    dummyItem->data = -1;
    dummyItem->key = -1;
```

```
    insert(1, 20);
    insert(2, 70);
    insert(42, 80);
    insert(4, 25);
    insert(12, 44);
    insert(14, 32);
    insert(17, 11);
    insert(13, 78);
    insert(37, 97);
    item = search(37);
    if(item != NULL)
    {
        printf("Element found: %d\n",
        item->data);
    }  else
    {
        printf("Element not found\n");
    }
    delete(item);
    item = search(37);
    if(item != NULL)
   {
      printf("Element found: %d\n",
      item->data);
   }  else
   {
      printf("Element not found\n");
   }
}
```

Slip - [5,9,10,13,15,20]

Q.13) BFS.

```
#include<stdio.h>
#include<stdlib.h>
#define MAXSIZE 20
typedef struct
{
    int data[MAXSIZE];
    int front, rear;
}QUEUE;

void initq(QUEUE *pq)
{
```

```
    pq->front=pq->rear=-1;
}
void addq(QUEUE *pq,int n)
{
   pq->data[++pq->rear]=n;
}
int removeq(QUEUE *pq)
{
   return pq->data[++pq->front];
}
int isempty(QUEUE *pq)
{
   return(pq->front==pq->rear);
}

void create(int a[10][10],int n)
{
   printf("\n****TYPE 1 FOR YES & 0 FOR NO****\n");
   for(int i=0;i<n;i++)
  {
     for(int j=0;j<n;j++)
    {
       a[i][j]=0;
       if(i!=j)
      {
        printf("\nIs there any edge between %d & %d: ",
        i+1,j+1);
        scanf("%d",&a[i][j]);
      }
   }
  }
}
void bfs(int a[10][10],int n)
{
    int v=0;
    int visited[20]={0};
    QUEUE q;
    initq(&q);
    printf("\nThe Breadth First
    Traversal is:\n");
    visited[v]=1;
    addq(&q,v);
    while(!isempty(&q))
```

```
    {
      v=removeq(&q);
      printf("v%d\t",v+1);
      for(int w=0;w<n;w++)
      {
        if((a[v][w]==1)&&(visited[w]
         ==0))
        {
           addq(&q,w);
           visited[w]=1;
         }
       }
    }
}
int main()
{
   int a[10][10],n;
   printf("\nEnter the no. of vertex: ");
   scanf("%d",&n);
   create(a,n);
   bfs(a,n);
}
```

Slip - [2,11,19,22]

Q.14) DFS.

```
#include<stdio.h>
#include<stdlib.h>
typedef struct node
{
    int vertex;
    struct node *next;
}NODE;
void create(int a[20][20],int n)
{
    printf("\n***Enter 1 for Yes & 0 for No**\n");
    for(int i=0;i<n;i++)
    {
      for(int j=0;j<n;j++)
      {
        a[i][j]=0;
        if(i!=j)
       {
          printf("\nIs there edge between v%d & v%d: ", i+1,j+1);
```

```
            scanf("%d",&a[i][j]);
          }
       }
    }
}
void recdfs(int a[20][20],int n,int v)
{
      static int visited[20]={0};
      visited[v]=1;
      printf("v%d\t",v+1);
      for(int w=0;w<n;w++)
      {
         if((a[v][w]==1)&& (visited[w]
==0))
         recdfs(a,n,w);
      }
}
int main()
{
     int a[20][20],n;
     printf("\nEnter the no. of vertex:     ");
    scanf("%d",&n);
    create(a,n);
    printf("\nThe Depth First Search
       Traversal is: \n");
    recdfs(a,n,0);
}
```

Slip - [8,24,12]

Q.15) In degree/ Out degree.

```
#include<stdio.h>
#include<malloc.h>
void create(int a[10][10],int n)
{
    printf("\n****TYPE 1 FOR YES & 0 FOR NO****\n");
    for(int i=0;i<n;i++)
   {
       for(int j=0;j<n;j++)
      {
         a[i][j]=0;
         if(i!=j)
        {
```

```
            printf("\nIs there any edge between %d & %d: ",i+1,j+1);
            scanf("%d",&a[i][j]);
          }
        }
      }
}
void inout(int a[10][10],int n)
{
    int in=0,out=0;
    printf("\nVertex\tIndegree
                \tOutdegree\t");
    for(int i=0;i<n;i++)
   {
       in=out=0;
     for(int j=0;j<n;j++)
     {
        in=in+a[j][i];
        out=out+a[i][j];
      }

printf("\n%d\t%d\t%d\t",i+1,in,out);
   }
}
int main()
{
     int a[10][10],n;
     printf("\nEnter the no. of vertex: ");
     scanf("%d",&n);
     create(a,n);
     inout(a,n);
}
```