哈尔滨工业大学计算机科学与技术学院

实验报告

课程名称：数据结构与算法

课程类型：必修

实验名称：图型结构及其应用

实验项目：

图的搜索（遍历）算法是图型结构相关算法的基础，本实验要求编写程序演示图两种存储结构的建立和搜索（遍历）过程。

班级：1703005
学号：1170300511
姓名：易亚玲

一、实验目的:

熟悉图的邻接矩阵和邻接表的建立与遍历。

二、实验要求

实验要求：1.分别实现图的邻接矩阵和链接表存储结构的建立算法，分析和比较各建立算法的时间复杂度以及存储结构的空间占用情况；

2.实现图的邻接矩阵、邻接表两种存储结构的相互转换算法；

3.在上述两种存储结构上，分别实现图的深度优先搜索（递归和非递归）和广度优先搜索算法。并以适当的方式存储和显示相应的搜索结果（深度优先和广度优先生成森林（或生成树）、深度优先或广度优先序列和编号）；

4.分析搜索算法的时间复杂度；

5.以文件形式输入图的顶点和边，并显示相应的结果。要求顶点不少于10个，边不少于13个；

6．软件功能结构安排合理，界面友好，便于使用。

实验环境：Windows, Code::Blocks。

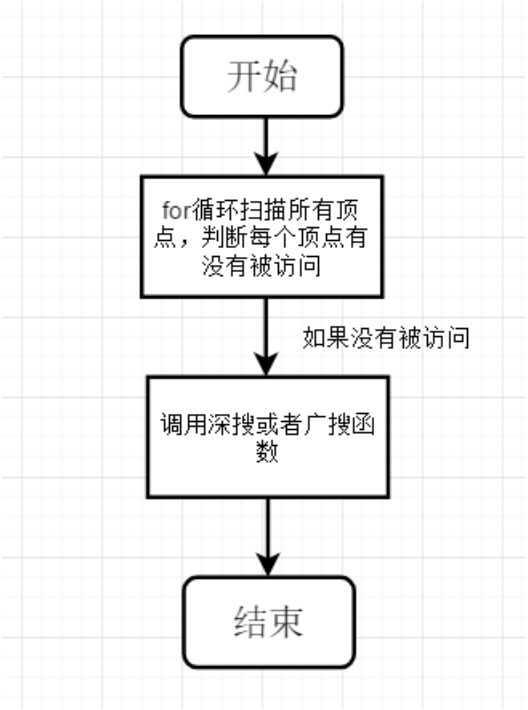
三、设计思想

★数据定义如下：

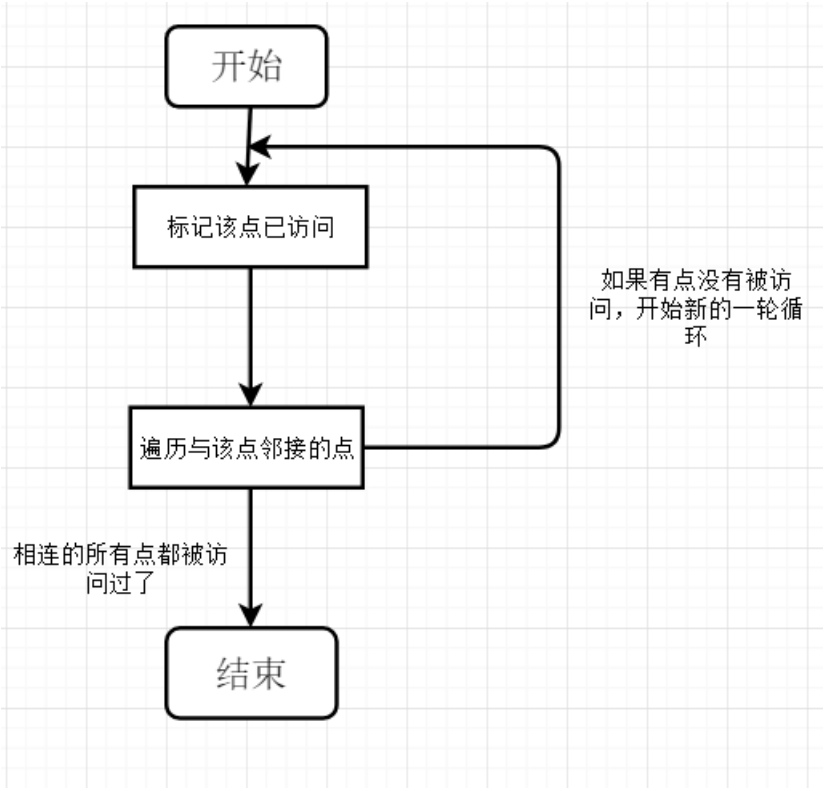
```
typedef int* oneint;  //int*
typedef oneint* twoint;//int** ,用于构建邻接矩阵
typedef struct yi{
  int n;
  struct yi* next;
}LINK;  //邻接表边表节点
typedef struct {
  char spot;
  int num;
  LINK* child;
}FORM;  //邻接表主表节点

int* visit;  //用于记录每一个顶点的访问编号
  int cnt; //用于计数访问
```

流程图：

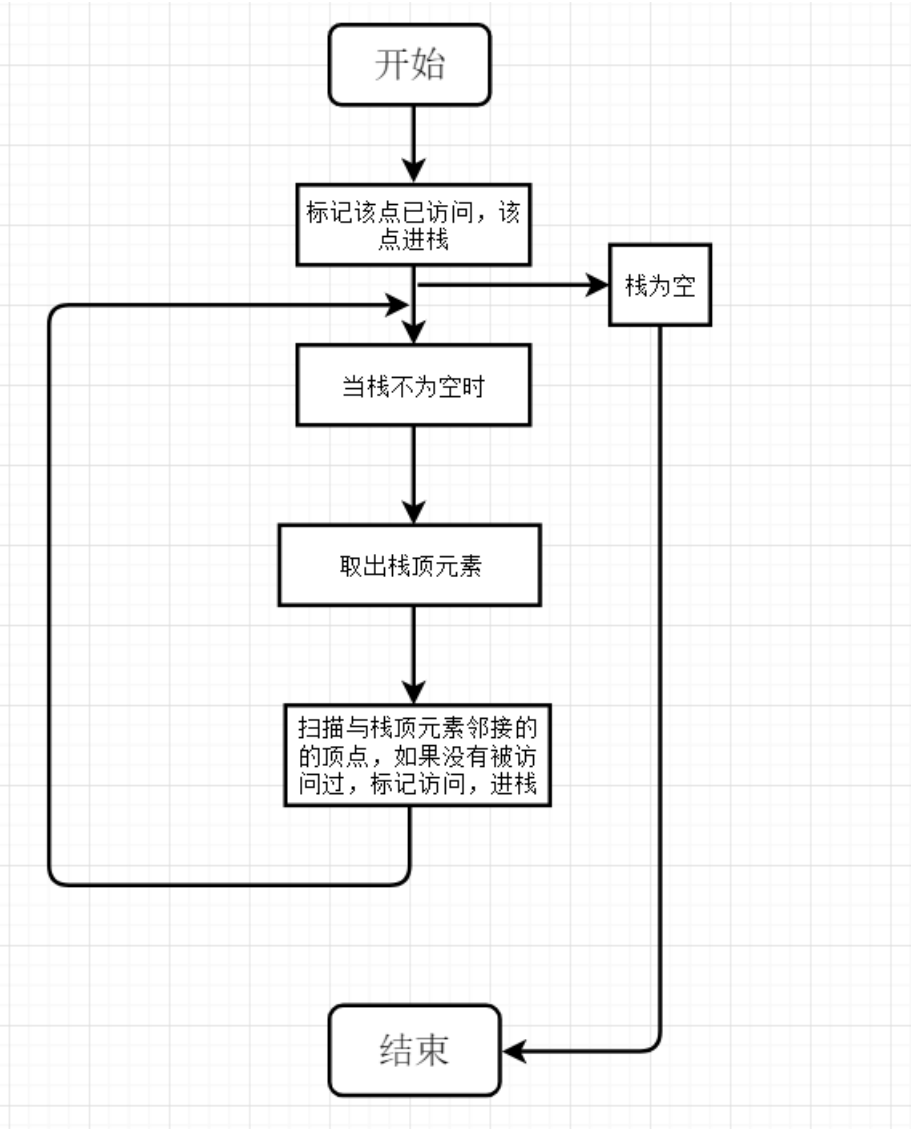
主体思想：



深搜递归：



深搜非递归:



广搜类似，只是用队列暂存顶点结构

四、测试结果如图

1.创建邻接矩阵:

2.创建邻接表

```
请输入一个数字选择如下功能:
0:退出
1:建立图的邻接矩阵存储
2:实现图的邻接表存储
3:将邻接矩阵转化为邻接表
4:将邻接表转化成邻接矩阵
5:深度优先搜索递归形式（邻接矩阵）
6:深度优先搜索非递归形式（邻接矩阵）
7:深度优先搜索递归形式（邻接表）
8:深度优先搜索非递归形式（邻接表）
9:广度优先搜索（邻接矩阵）
10:广度优先搜索（邻接表）

2
邻接表为:
a 11 8 7 4 2
b 9 8 5 3 1
c 12 9 8 5 2
d 10 8 6 1
e 7 3 2
f 9 4
g 10 5 1
h 9 4 3 2 1
i 10 8 6 3 2
j 11 9 7 4
k 10 1
l 3
m
请按任意键继续. . .
```

3.邻接矩阵转化为邻接表

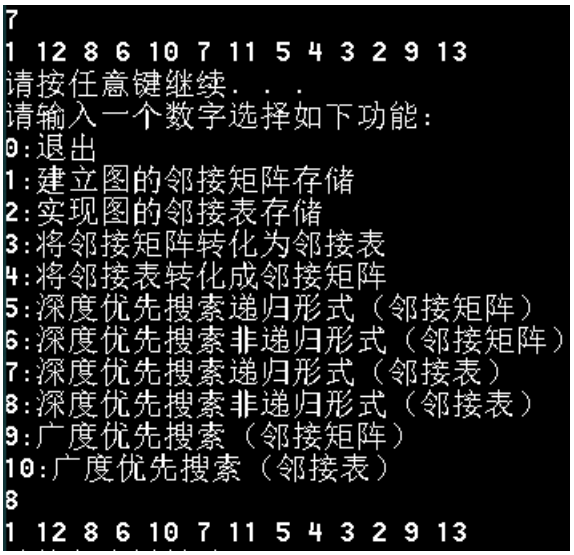
```
请输入一个数字选择如下功能:
0:退出
1:建立图的邻接矩阵存储
2:实现图的邻接表存储
3:将邻接矩阵转化为邻接表
4:将邻接表转化成邻接矩阵
5:深度优先搜索递归形式（邻接矩阵）
6:深度优先搜索非递归形式（邻接矩阵）
7:深度优先搜索递归形式（邻接表）
8:深度优先搜索非递归形式（邻接表）
9:广度优先搜索（邻接矩阵）
10:广度优先搜索（邻接表）
3
1  11 8 7 4 2
2  9 8 5 3 1
3  12 9 8 5 2
4  10 8 6 1
5  7 3 2
6  9 4
7  10 5 1
8  9 4 3 2 1
9  10 8 6 3 2
10  11 9 7 4
11  10 1
12  3
13
```

4.邻接表转化为邻接矩阵

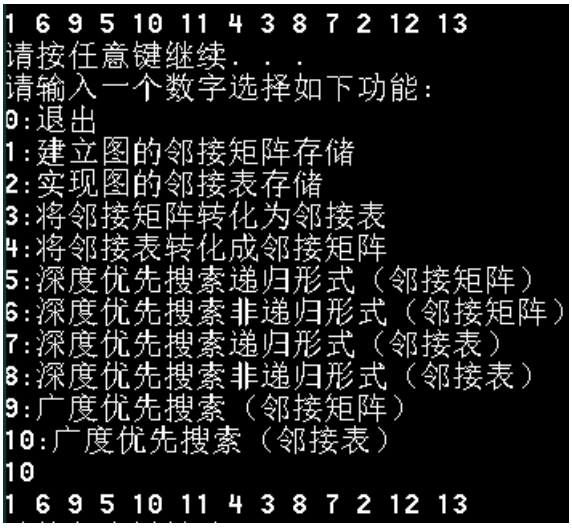
```
请输入一个数字选择如下功能:
0:退出
1:建立图的邻接矩阵存储
2:实现图的邻接表存储
3:将邻接矩阵转化为邻接表
4:将邻接表转化成邻接矩阵
5:深度优先搜索递归形式（邻接矩阵）
6:深度优先搜索非递归形式（邻接矩阵）
7:深度优先搜索递归形式（邻接表）
8:深度优先搜索非递归形式（邻接表）
9:广度优先搜索（邻接矩阵）
10:广度优先搜索（邻接表）
4
0 1 0 1 0 0 1 1 0 0 1 0 0
1 0 1 0 1 0 0 1 1 0 0 0 0
0 1 0 0 1 0 0 1 1 0 0 1 0
1 0 0 0 0 1 0 1 0 1 0 0 0
0 1 1 0 0 0 1 0 0 0 0 0 0
0 0 0 1 0 0 0 0 1 0 0 0 0
1 0 0 0 1 0 0 0 0 1 0 0 0
1 1 1 1 0 0 0 0 1 0 0 0 0
0 1 1 0 0 1 0 1 0 1 0 0 0
0 0 0 1 0 0 1 0 1 0 1 0 0
1 0 0 0 0 0 0 0 0 1 0 0 0
0 0 1 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0
请按任意键继续. . .
```

5.深度搜索

```
5
1 12 8 6 10 7 11 5 4 3 2 9 13
请按任意键继续. . .
请输入一个数字选择如下功能:
0:退出
1:建立图的邻接矩阵存储
2:实现图的邻接表存储
3:将邻接矩阵转化为邻接表
4:将邻接表转化成邻接矩阵
5:深度优先搜索递归形式（邻接矩阵）
6:深度优先搜索非递归形式（邻接矩阵
7:深度优先搜索递归形式（邻接表）
8:深度优先搜索非递归形式（邻接表）
9:广度优先搜索（邻接矩阵）
10:广度优先搜索（邻接表）
6
1 12 8 6 10 7 11 5 4 3 2 9 13
```



6.广度搜索



五、系统不足与经验体会

通过书写程序，对图的搜索更加熟悉，更加了解深度和广度搜索的过程，有利于以后的运用。这个程序可能在可视化上还需要一些改进，还需要进一步优化。

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

/*该程序能够实现仅有邻接矩阵、仅有邻接表或者两者都有的输入方式，如果需要单独输入邻接表和邻接矩阵，需要将注释的内容（位主函数和代码末尾）重新恢复，并且稍微修改主函数*/
```

```
typedef int* oneint;  //int*
typedef oneint* twoint;//int** ,用于构建邻接矩阵
typedef struct yi{
  int n;
  struct yi* next;
}LINK;  //邻接表边表节点
typedef struct {
  char spot;
  int num;
  LINK* child;
}FORM;  //邻接表主表节点

int* visit;  //用于记录每一个顶点的访问编号
int cnt; //用于计数访问

/*辅助功能*/
void printmenu();

/*主功能*/
void search_main(FORM* adjform,twoint matrix,int M,int sel);  //搜索主函数
/*1*/twoint creat_adjmatrix(twoint matrix,int *M,int *edge);  //创建邻接矩阵
/*2*/FORM* creat_adjform(FORM* adjform,int *M,int *edge);  //创建邻接表
/*3*/void  matrix_to_form(twoint matrix,int M);  //邻接矩阵转化为邻接表
/*4*/void form_to_matrix(FORM* adjform,int M);  //邻接表转化为邻接矩阵
/*5*/void dfs_dg_matrix_main(twoint matrix,int M); //深搜递归主函数（邻接矩阵）
/*5*/void dfs_dg_matrix_child(twoint matrix,int k,int M); //深搜递归子函数（邻接矩阵
）
/*6*/void dfs_fdg_matrix_main(twoint matrix,int M); //深搜非递归主函数（邻接矩阵
）
/*6*/void dfs_fdg_matrix_child(twoint matrix,int k,int M); //深搜非递归子函数（邻接
矩阵）
/*7*/void dfs_dg_adjform_main(FORM* adjform,int M); //深搜递归主函数（邻接表）
/*7*/void dfs_dg_adjform_child(FORM* adjform,int k); //深搜递归子函数（邻接表）
/*8*/void dfs_fdg_adjform_main(FORM* adjform,int M); //深搜非递归主函数（邻接
表）
/*8*/void dfs_fdg_adjform_child(FORM* adjform,int k,int M); //深搜非递归子函数（
邻接表）
/*9*/void bfs_matrix_main(twoint matrix,int M);  //广搜主函数(邻接矩阵)
/*9*/void bfs_matrix_child(twoint matrix,int k,int M);  //广搜子函数（邻接表）
/*10*/void bfs_adjform_main(FORM* adjform,int M);   //广搜主函数（邻接表）
/*10*/void bfs_adjform_child(FORM* adjform,int k,int M); //广搜子函数（邻接表）
```

```
int main()
{
  int sel = 0;
  int M = 0;
  int edge;
  twoint matrix = NULL;
  FORM* adjform = NULL;
  while(1)
  {
    printmenu();
    scanf("%d",&sel);
    switch(sel)
    {
      case 0: exit(0);
      case 1:matrix = creat_adjmatrix(matrix,&M,&edge);
          break;
      case 2:adjform = creat_adjform(adjform,&M,&edge);
          break;
      case 3:matrix_to_form(matrix,M);
          break;
      case 4:form_to_matrix(adjform,M);
          break;
      case 5:
          search_main(adjform,matrix,M,sel);
          //dfs_dg_matrix_main(matrix,M);
          break;
      case 6:
          search_main(adjform,matrix,M,sel);
          //dfs_fdg_matrix_main(matrix,M);
          break;
      case 7:
          search_main(adjform,matrix,M,sel);
          //dfs_dg_adjform_main(adjform,M);
          break;
      case 8:
          search_main(adjform,matrix,M,sel);
          //dfs_fdg_adjform_main(adjform,M);
          break;
      case 9:
          search_main(adjform,matrix,M,sel);
          //bfs_matrix_main(matrix,M);
          break;
      case 10:
          search_main(adjform,matrix,M,sel);
          //bfs_adjform_main(adjform,M);
          break;
```

```
        default:
              break;

      }
     system("pause");
   }
  return 0;
}


void printmenu()
{
   printf("请输入一个数字选择如下功能:\n");
   printf("0:退出\n");
   printf("1:建立图的邻接矩阵存储\n");
   printf("2:实现图的邻接表存储\n");
   printf("3:将邻接矩阵转化为邻接表\n");
   printf("4:将邻接表转化成邻接矩阵\n");
   printf("5:深度优先搜索递归形式（邻接矩阵）\n");
   printf("6:深度优先搜索非递归形式（邻接矩阵）\n");
   printf("7:深度优先搜索递归形式（邻接表）\n");
   printf("8:深度优先搜索非递归形式（邻接表）\n");
   printf("9:广度优先搜索（邻接矩阵）\n");
   printf("10:广度优先搜索（邻接表）\n");
}

twoint creat_adjmatrix(twoint matrix,int *M,int *edge)
{
   int row,side = 0;
   int i,j;
   int ch1,ch2;
   FILE *fp = NULL;
   fp = fopen("matrix.doc","r");
   fscanf(fp,"%d",M);
   row = *M;
   matrix =(twoint)calloc(sizeof(int *),row+1);
   for(i = 1;i <= row;i++)
     matrix[i] = (int *)calloc(sizeof(int),row+1);
   while(fscanf(fp,"%d %d",&ch1,&ch2) != EOF)
   {
      side ++;
      matrix[ch1][ch2] = 1;
      matrix[ch2][ch1] = 1;
   }
```

```
    *edge = side;
    printf("邻接矩阵为:\n");
    for(i = 1;i <= row;i++){
        for(j = 1;j <= row;j++)
            printf("%d ",matrix[i][j]);
        printf("\n");
    }
    fclose(fp);
    return matrix;
}

FORM* creat_adjform(FORM* adjform,int *M,int *edge)
{
  int row,i = 0,side = 0;
  int ch1,ch2;
  char ch;
  LINK* dot;
  FILE* fp;
  fp = fopen("form.doc","r");
  fscanf(fp,"%d",M);
  row = *M;
  adjform = (FORM*)malloc(sizeof(FORM)*(row+1));
  for(i = 1;i <= row;i++)
    adjform[i].child = NULL;
  for(i = 1;i <= row;i++)
  {
     fscanf(fp,"%c",&ch);
     adjform[i].spot = ch;
  }
  while(fscanf(fp,"%d %d",&ch1,&ch2) != EOF)
  {
    side ++;
    dot = (LINK*)malloc(sizeof(LINK));
    dot -> n = ch2;
    dot -> next = adjform[ch1].child;
    adjform[ch1].child = dot;
    dot = (LINK*)malloc(sizeof(LINK));
    dot -> n = ch1;
    dot -> next = adjform[ch2].child;
    adjform[ch2].child = dot;
  }
  *edge = side;
  printf("邻接表为:\n");
  for(i = 1;i <= row ;i++)
  {
    printf("%c ",adjform[i].spot);
```

```
    dot = adjform[i].child;
    while(dot != NULL)
    {
      printf("%d ",dot -> n);
      dot = dot -> next;
    }
    printf("\n");
  }
  fclose(fp);
  return adjform;
}

void matrix_to_form(twoint matrix,int M)
{
   int i,j;
   LINK* pt;
   FORM* adjform;
   adjform =(FORM *)malloc(sizeof(FORM)*(M+1));
   for(i = 1;i <= M;i++)
     adjform[i].child = NULL;
   for(i = 1;i <= M;i++)
   {
      for(j = i; j <= M;j++)
        if(matrix[i][j] == 1)
        {
           pt =(LINK*)malloc(sizeof(LINK));
           pt -> n = i;
           pt -> next = adjform[j].child;
           adjform[j].child = pt;

           pt =(LINK*)malloc(sizeof(LINK));
           pt -> n = j;
           pt -> next = adjform[i].child;
           adjform[i].child = pt;
        }
   }
   for(i = 1;i <= M;i++)
   {
      printf("%d  ",i);
      pt = adjform[i].child;
      while(pt != NULL)
      {
        printf("%d ",pt -> n);
        pt = pt -> next;
      }
      printf("\n");
```

```
    }
}

void form_to_matrix(FORM* adjform,int M)
{
  int i,j;
  twoint matrix;
  LINK* pt;
  matrix = (twoint)malloc(sizeof(oneint*)*(M+1));
  for(i = 1;i <= M;i++)
    matrix[i] = (int *)malloc(sizeof(int)*(M+1));
  for(i = 1;i <= M;i++)
    for(j = 1;j <= M;j++)
      matrix[i][j] = 0;
  for(i = 1;i <= M;i++)
  {
     pt = adjform[i].child;
     while(pt != NULL)
     {
       matrix[i][pt -> n] = 1;
       pt = pt -> next;
     }
  }
  for(i = 1;i <= M;i++)
  {
    for(j = 1;j <= M;j++)
      printf("%d ",matrix[i][j]);
    printf("\n");
  }
}


void dfs_dg_matrix_child(twoint matrix,int k,int M)
{
  int j;
  visit[k] = ++cnt;
  for(j = M;j > 0;j--)
  {
     if(matrix[k][j] == 1 && !visit[j])
       dfs_dg_matrix_child(matrix,j,M);
  }
}


void dfs_fdg_matrix_child(twoint matrix,int k,int M)
{
```

```
   int i,t,top = -1;
   int* st;
   st = (int*)malloc(sizeof(int)*(M+1));
   visit[k] = ++cnt;
   st[++top] = k;
   while(top != -1)
   {
     t = st[top--];
     if(!visit[t])
        visit[t] = ++cnt;
     for(i = 1;i <= M;i++)
        if(matrix[t][i] == 1 && !visit[i])
              st[++top] = i;
  }
}



void dfs_dg_adjform_child(FORM* adjform,int k)
{
   LINK* pt;
   visit[k] = ++cnt;
   pt = adjform[k].child;
   while(pt != NULL )
   {
      if(!visit[pt -> n])
        dfs_dg_adjform_child(adjform,pt -> n);
      else
       pt = pt -> next;
   }
}


void dfs_fdg_adjform_child(FORM* adjform,int k,int M)
{
  int* st;
  int top = -1;
  LINK* pt;
  st = (int*)malloc(sizeof(int)*(M+1));
  visit[k] = ++cnt;
  st[++top] = k;
  while(top != -1)
  {
    pt = adjform[st[top--]].child;
    while(pt != NULL)
    {
```

```
        if(!visit[pt -> n])
        {
           visit[pt -> n] = ++cnt;
           st[++top] = pt -> n;
           pt = adjform[pt -> n].child;
        }
        else
          pt = pt -> next;
     }
   }
}


void bfs_matrix_child(twoint matrix,int k,int M)
{
   int i,first,rear,pt;
   int* st;
   st = (int*)malloc(sizeof(int)*(M+1));
   visit[k] = ++cnt;
   first = rear = 0;
   st[rear++] = k;
   while(rear != first)
   {
      pt = st[first++];
      for(i = M;i > 0;i--)
         if(matrix[pt][i]==1&&!visit[i])
         {
            visit[i] = ++cnt;
            st[rear++] = i;
         }
   }
}

void bfs_adjform_child(FORM* adjform,int k,int M)
{
  int first,rear;
  int* st;
  LINK* pt;
  st = (int*)malloc(sizeof(int)*(M+1));
  visit[k] = ++cnt;
  first = rear = 0;
  st[rear++] = k;
  while(rear != first)
  {
     pt = adjform[st[first++]].child;
     while(pt != NULL)
```

```
      {
         if(!visit[pt -> n])
         {
            visit[pt -> n] = ++cnt;
            st[rear++] = pt -> n;
         }
         pt = pt -> next;
      }
   }
}

void search_main(FORM* adjform,twoint matrix,int M,int sel)
{
   int i;
   visit = (int*)malloc(sizeof(int)*(M+1));
   for(i = 1;i <= M;i++)
      visit[i] = 0;
   for(i = 1;i <= M;i++)
      if(!visit[i])
         switch(sel)
         {
            case 5:dfs_dg_matrix_child(matrix,i,M);
                   break;
            case 6:dfs_fdg_matrix_child(matrix,i,M);
                   break;
            case 7:dfs_dg_adjform_child(adjform,i);
                   break;
            case 8:dfs_fdg_adjform_child(adjform,i,M);
                   break;
            case 9:bfs_matrix_child(matrix,i,M);
                   break;
            case 10:bfs_adjform_child(adjform,i,M);
                   break;
         }
   for(i = 1;i <= M;i++)
      printf("%d ",visit[i]);
   printf("\n");
   cnt = 0;
   free(visit);
}

/*
void dfs_dg_matrix_main(twoint matrix,int M)
{
   int i;
   cnt = 0;
```

```
  visit = (int *)malloc(sizeof(int)*(M+1));
  for(i = 1;i <= M;i++)
    visit[i] = 0;
  for(i = 1;i <= M;i++)
    if(!visit[i])
      dfs_dg_matrix_child(matrix,i,M);
  for(i = 1;i <= M;i++)
    printf("%d ",visit[i]);
  printf("\n");
  cnt = 0;
  free(visit);
}

void dfs_fdg_matrix_main(twoint matrix,int M)
{
  int i;
  visit = (int *)malloc(sizeof(int)*(M+1));
  for(i = 1;i <= M;i++)
    visit[i] = 0;
  for(i = 1;i <= M;i++)
    if(!visit[i])
      dfs_fdg_matrix_child(matrix,i,M);
  for(i = 1;i <= M;i++)
    printf("%d ",visit[i]);
  printf("\n");
  cnt = 0;
  free(visit);
}

void dfs_dg_adjform_main(FORM* adjform,int M)
{
   int i;
   visit = (int *)malloc(sizeof(int)*(M+1));
   for(i = 1;i <= M;i++)
      visit[i] = 0;
   for(i = 1;i <= M;i++)
      if(!visit[i])
         dfs_dg_adjform_child(adjform,i);
   for(i = 1;i <= M;i++)
      printf("%d ",visit[i]);
   printf("\n");
   cnt = 0;
   free(visit);
}

void dfs_fdg_adjform_main(FORM* adjform,int M)
```

```
{
   int i;
   visit =(int*)malloc(sizeof(int)*(M+1));
   for(i = 1;i <= M;i++)
      visit[i] = 0;
   for(i = 1;i <= M;i++)
      if(!visit[i])
         dfs_fdg_adjform_child(adjform,i,M);
   for(i = 1;i <= M;i++)
      printf("%d ",visit[i]);
   printf("\n");
   cnt = 0;
   free(visit);
}

void bfs_matrix_main(twoint matrix,int M)
{
  int i;
  visit = (int*)malloc(sizeof(int)*(M+1));
  for(i = 1;i <= M;i++)
    visit[i] = 0;
  for(i = 1;i <= M;i++)
    if(!visit[i])
      bfs_matrix_child(matrix,i,M);
  for(i = 1;i <= M;i++)
    printf("%d ",visit[i]);
  printf("\n");
  cnt = 0;
  free(visit);
}

void bfs_adjform_main(FORM* adjform,int M)
{
   int i;
   visit = (int*)malloc(sizeof(int)*(M+1));
   for(i = 1;i <= M;i++)
      visit[i] = 0;
   for(i = 1;i <= M;i++)
      if(!visit[i])
         bfs_adjform_child(adjform,i,M);
   for(i = 1;i <= M;i++)
      printf("%d ",visit[i]);
   printf("\n");
   cnt = 0;
   free(visit);
}*/
```