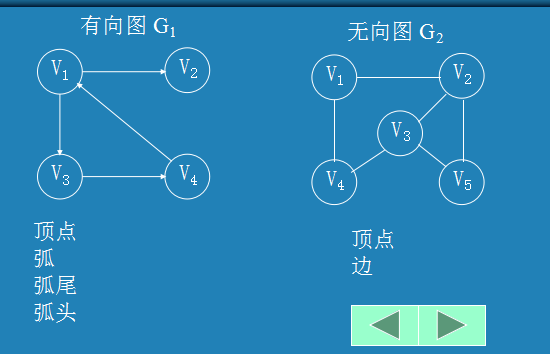
1. 图的定义和基本术语
2. 图的储存结构
   1. 邻接矩阵
   2. 邻接表
   3. 十字链表、邻接多重表、边集数组
3. 图的遍历
   1. 深度优先遍历
      1. 邻接矩阵代码AdMatrix\_DFS.c
      2. 邻接表代码AdList\_DFS.c
      3. 深度优先遍历思想：马踏棋算法knight.c
   2. 广度度优先遍历
      1. 邻接矩阵代码AdMatrix\_BFS.c
      2. 邻接表代码AdList\_BFS.c
4. 最小生成树
   1. 普里姆算法prim.c
   2. 克鲁斯卡尔算法kruskal.c
5. 最短路径
   1. 迪杰斯特拉算法dijkstra.c
   2. 佛洛依德算法floyd.c
6. 关键路径
   1. 拓扑排序算法ToplogicalSort.c
   2. 关键路径算法CriticalPath.c

### 关注：

1. 只关注入度/出度：邻接表(|只能考虑出度或者入度)，邻接矩阵(索引、编排方便|边数相对顶点较少则浪费)，
2. 同时关注入度/出度：十字链表(有向图的优化储存结构|删边难)
3. 关注边：邻接多重表

### 【图的定义和基本术语】



完全图：n个顶点有n(n-1)/2条边的无向图

有向完全图： n个顶点有n(n-1)条弧的有向图

稀疏图：有很少条边的图（如边数e < nlogn)

稠密图：非稀疏图

权： 与边或弧相关的数

网络： 带权的图

子图： G =(V,{E})和G1 = (V1,{E1})

若V1属于V, E1属于E 则G1是G的子图

邻接点：无向图中有边相连的两个顶点互为邻接点

顶点的度：无向图中和某顶点相连的邻接点数

入度：有向图中指向某顶点的弧的数目

出度：有向图中从某顶点出发的弧的数目

路径：两个顶点之间的顶点序列，该序列的每个顶点与其前驱是邻接点，每个顶点与其后继也是邻接点

回路（环）：第一顶点和最后顶点相同的路径

简单路径： 顶点不重复的路径

连通： 两个顶点之间有路径

连通图： 任意两个顶点之间有路径

连通分量： 无向图中的极大连通子图。

强连通图：任意两个顶点之间有双向路径

强连通分量：有向图中的极大强连通子图。

连通图的生成树：极小连通子图。

不唯一，但n个顶点的生成树

有且仅有n-1条边。

生成森林：

### 【邻接矩阵|数组表示法】

#define VRType int

#define InfoType int

#define VertexType char

#define MAX\_VERTEX\_NUM 20

#define MAX\_VALUE

typedef enum{DG, DN, AG, AN} GraphKind;

// 弧结点与矩阵的类型

typedef struct ArcCell{

VRType adj; //VRType为弧的类型。图--0,1;网--权值

InfoType \*info; //与弧相关的信息的指针,可省略

}ArcCell, AdjMatrix[MAX\_VERTEX\_NUM ][MAX\_VERTEX\_NUM ]

// 图的类型

typedef struct {

VetexType vexs[MAX\_VERTEX\_NUM ];//顶点数组,顶点向量

AdjMatrix arc;//矩阵

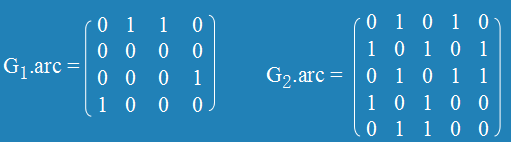
int vexnum, arcnum;//顶点数，边数

GraphKind kind; //图类型

}MGraph;

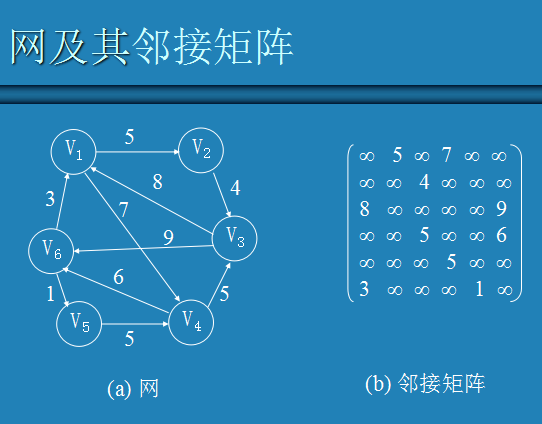
无向图、有向图、网均适用  
易求各顶点的度。

例如有向图G1和无向图G2的邻接矩阵为

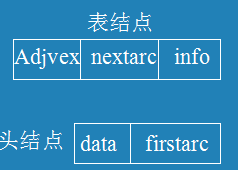


n2的存储量

无向图的邻接矩阵总是对称的--可以采用压缩存储



### 【邻接表】



#define VRType int

#define InfoType int

#define VertexType char

typedef enum{DG, DN, AG, AN} GraphKind;

#define MAX\_VERTEX\_NUM 20 　　　　//最大顶点数

typedef struct ArcNode{ 　　　　//边结点

int adjvex; 　　　　//邻接点的下标

struct ArcNode \*nextarc; //后继链指针

InfoType \*info; //与弧相关的信息的指针,可省略

}ArcNode;

typedef struct VNode{ 　　　　//顶点结点

VertexType data; //顶点数据

ArcNode \*firstarc; //边链头指针

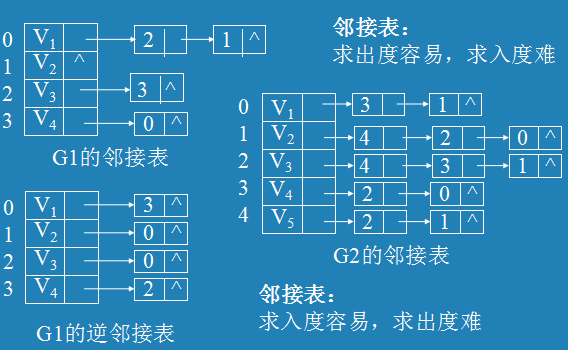
}VNode, AdjList[MAX\_VERTEX\_NUM];

typedef struct{

AdjList vertices; //邻接表

int vexnum,arcnum; //顶点数和边数

GraphType kind； 　　//图种类标志

}ALGraph;

### 【十字链表】

//顶点结点最大数量

int const MAX\_VERTEXNUM = 100;

//数据结构

typedef int InfoType;

typedef int Data;

//弧结点

typedef struct \_tagArcBox

{

int tailvex; //该弧的弧尾结点在图中的位置

int headvex; //该弧的弧头结点在图中的位置

struct \_tagArcBox\* hlink; //下一条与该弧有相同弧头结点的弧

struct \_tagArcBox\* tlink; //下一条与该弧有相同弧尾结点的弧

InfoType info; //弧的相关信息

}ArcBox;

//顶点结点

typedef struct \_tagArcNode

{

Data data; //数据

ArcBox\* firstin; //第一条以该节点为弧尾的弧

ArcBox\* firstout; //第一条以该结点为弧头的弧

}ArcNode;

//十字链表存储结构的有向图

typedef struct \_tagOLGraph

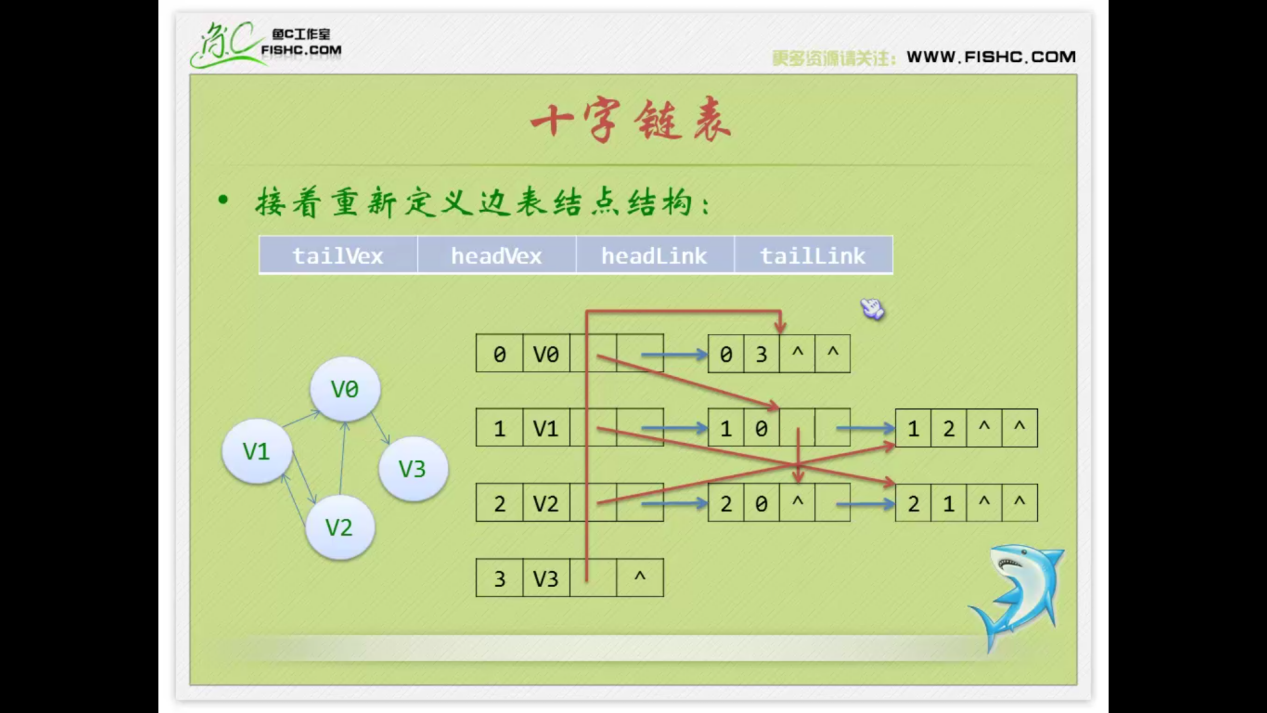
{

ArcNode vertex[MAX\_VERTEXNUM]; //顶点向量

int vexnum; //顶点数

int arcnum; //弧树

}OLGraph, \*POLGraph;



### 【十字链表】

#define MAX\_VERTEX\_NUM 20

//顶点数据类型

typedef int VertexType;

typedef int Status;

typedef enum {unvisited,isvisited}VisitIf;

//图的种类:无向图/网

typedef enum {UDG,UDN}GraphKind;

typedef struct EBox {

VisitIf mark; //访问标记

int ivex; //边顶点一

int jvex; //边顶点二

struct EBox \*ilink; //指向依附于顶点一的下一条边,类似于起点相同的一条弧

struct EBox \*jlink; //指向依附于顶点二的下一条边,类似与终点想同的一条弧

int weight; //边的权值

}EBox;

typedef struct VexBox {

VertexType data;

EBox \*firstarc; //指向第一条依附于该顶点的边

}VexBox;

typedef struct {

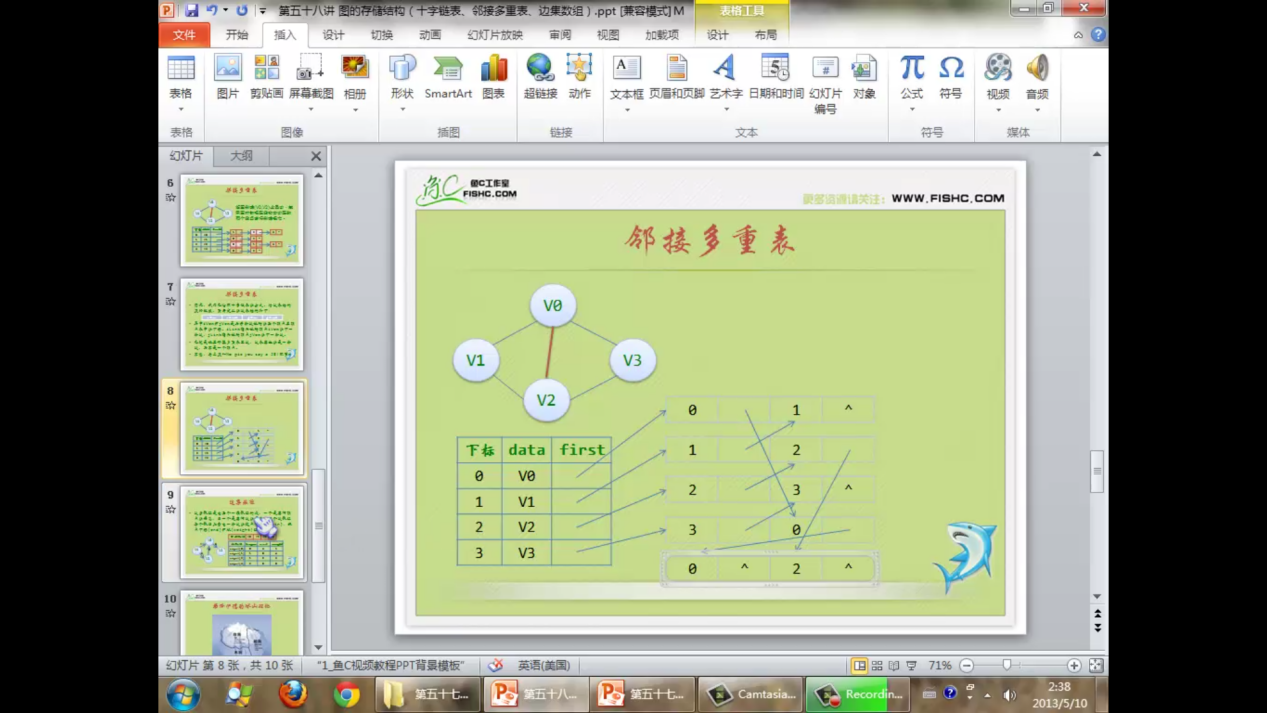
VexBox adjmulist[MAX\_VERTEX\_NUM];

int vexnum; //图的顶点数

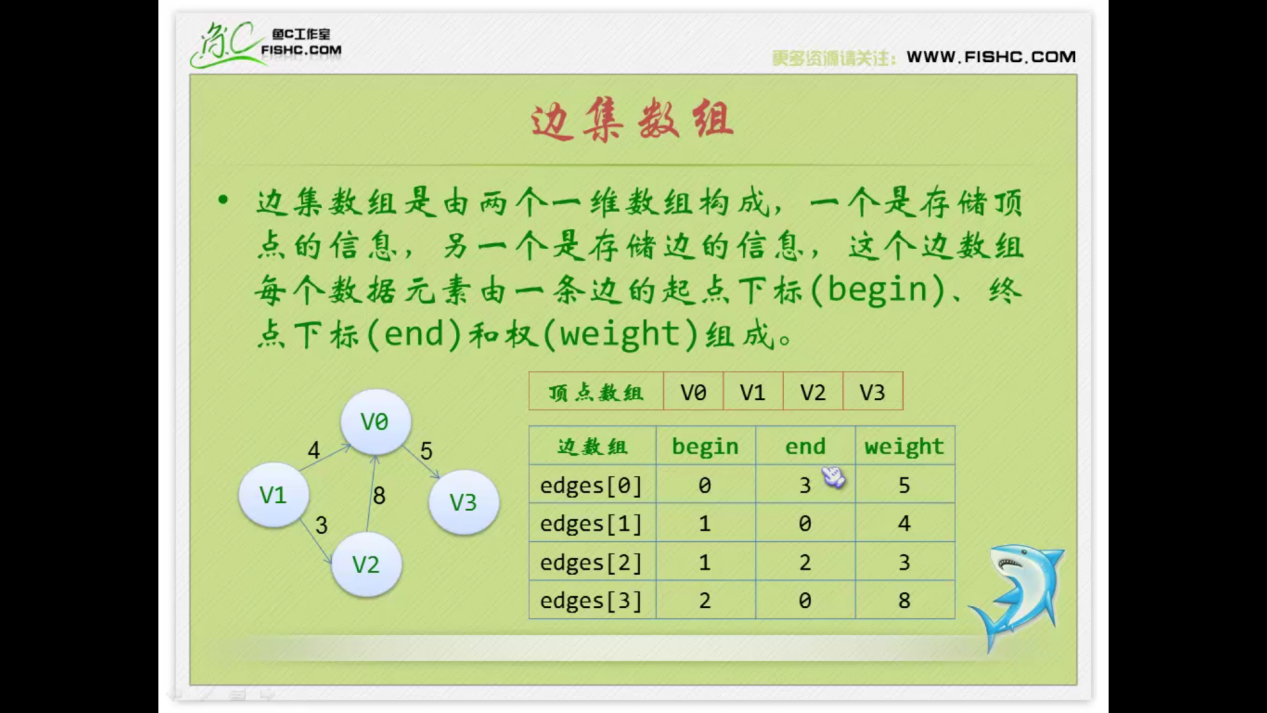
int arcnum; //图的边数

GraphKind kind; //图的种类

}AMLGraph;



【边集数组】



### 【深度遍历】

1. 邻接矩阵代码

#include <stdio.h>

#include <stdlib.h>

#include <stdbool.h>

#define VRType int

#define InfoType int

#define VertexType char

#define MAX\_VERTEX\_NUM 20

#define MAX\_VALUE 0

typedef enum{DG,DN,AG,AN}GraphKind;

typedef struct ArcCell{

VRType adj;

InfoType \*info;

}ArcCell,AdjMatrix[MAX\_VERTEX\_NUM][MAX\_VERTEX\_NUM];

typedef struct{

VertexType vexs[MAX\_VERTEX\_NUM];

AdjMatrix arc;

int vernum,arcnum;

GraphKind kind;

}MGraph;

void g\_create(MGraph \*graph)

{

int num;

int i,j,k;

char c;

printf("Please enter the num of vertex:\n");

scanf("%d",&graph->vernum);

getchar();

printf("Please enter vertex infomations:\n");

for(i=0;i<graph->vernum;i++)

{

scanf("%c",&graph->vexs[i]);

getchar();

}

for(i=0;i<graph->vernum;i++)

for(j=0;j<graph->vernum;j++)

graph->arc[i][j].adj = MAX\_VALUE;

graph->arcnum = 0;

for(i=0;i<graph->vernum;i++)

{

printf("Please enter vertex nextto the %c ,and end by #\n",graph->vexs[i]);

for(j=0;j<graph->vernum;j++)

{

scanf("%c",&c);

if(c=='#')

{

getchar();

break;

}

//scanf("%d",&num);

for(k=0;k<graph->vernum;k++)

{

if(graph->vexs[k]!=c)

continue;

//graph->arc[i][k].adj=num;

graph->arc[i][k].adj=1;

graph->arcnum++;

}

getchar();

}

}

graph->arcnum /=2;

printf("\n");

for(i=0;i<graph->vernum;i++)

{

for(j=0;j<graph->vernum;j++)

{

printf("%d ",graph->arc[i][j].adj);

}

printf("\n");

}

}

static void dfs\_graph(MGraph \*graph,bool visited[],const int i);

void g\_depth\_first\_search(MGraph \*graph)

{

bool visited[graph->vernum];

int i;

for(i=0;i<graph->vernum;i++)

visited[i]=false;

visited[0]=true;

dfs\_graph(graph,visited,0);

printf("\n");

}

static void dfs\_graph(MGraph \*graph,bool visited[],const int i)

{

int j;

printf("%c\t",graph->vexs[i]);

for(j=0;j<graph->vernum;j++)

{

if(graph->arc[i][j].adj != MAX\_VALUE && !visited[j])

{

visited[j]=true;

dfs\_graph(graph,visited,j);

}

}

}

int main()

{ MGraph graph;

g\_create(&graph);

g\_depth\_first\_search(&graph);

return 0;

}

1. 邻接表代码

#include <stdio.h>

#include <stdlib.h>

#include <stdbool.h>

#define VRType int

#define InfoType int

#define VertexType char

#define MAX\_VERTEX\_NUM 20

typedef enum{DG,DN,AG,AN}GraphKind;

typedef struct ArcNode

{

int adjvex;

struct ArcNode \*nextarc;

InfoType \*info;

}ArcNode;

typedef struct VNode

{

VertexType data;

ArcNode \*firstarc;

}VNode,AdjList[MAX\_VERTEX\_NUM];

typedef struct{

AdjList vertices;

int vexnum,arcnum;

GraphKind kind;

}ALGraph;

void InsertNode(ALGraph \*G,int i,int j)

{

printf("\ni=%d,j=%d\n`",i,j);

ArcNode \*ptrArcNode = (ArcNode\*)malloc(sizeof(ArcNode));

ptrArcNode->adjvex = j;

ptrArcNode->nextarc = G->vertices[i].firstarc;

G->vertices[i].firstarc = ptrArcNode;

printf("%c-%c\n",G->vertices[i].data,G->vertices[j].data);

ptrArcNode = (ArcNode\*)malloc(sizeof(ArcNode));

ptrArcNode->adjvex = i;

ptrArcNode->nextarc = G->vertices[j].firstarc;

G->vertices[j].firstarc = ptrArcNode;

}

void CreateALGraph(ALGraph \*G)

{

int i,j,k;

if(G==NULL) G = (ALGraph\*)malloc(sizeof(ALGraph));

printf("Please input the vexnum and arcnum:\n");

scanf("%d%d",&G->vexnum,&G->arcnum);

getchar();

printf("Please input the Vertices:\n");

for (i=0;i<G->vexnum;i++)

{

printf("Vertex %d:",i+1);

scanf("%c",&G->vertices[i].data);

G->vertices[i].firstarc=NULL;

getchar();

}

for(k=0;k<G->arcnum;k++)

{

printf("Please input (vi,vj) vertex:\n");

scanf("%d%d",&i,&j);

InsertNode(G,i,j);

}

}

void ShowALGraph(ALGraph \*G)

{

ArcNode \*ptrArcNode;

int i;

for(i=0;i<G->vexnum;i++)

{

printf("The Vertex %d is %c :",i+1,G->vertices[i].data);

ptrArcNode = G->vertices[i].firstarc;

while(1)

{

if(ptrArcNode==NULL)break;

printf("->%c",G->vertices[ptrArcNode->adjvex]);

if(ptrArcNode->nextarc==NULL)break;

ptrArcNode = ptrArcNode->nextarc;

}

printf("\n");

}

}

void DFS(ALGraph \*G,bool visited[],int i)

{

ArcNode \*ptrArcNode;

visited[i]=true;

printf("%c\t",G->vertices[i].data);

ptrArcNode = G->vertices[i].firstarc;

while(ptrArcNode)

{

if(!visited[ptrArcNode->adjvex])DFS(G,visited,ptrArcNode->adjvex);

ptrArcNode = ptrArcNode->nextarc;

}

}

void DFSTraverse(ALGraph \*G)

{

int i;

bool visited[MAX\_VERTEX\_NUM];

for(i=0;i<G->vexnum;i++)

visited[i]==false;

for(i=0;i<G->vexnum;i++)

{

if(!visited[i]) DFS(G,visited,i);

}

printf("\n");

}

int main()

{

ALGraph G;

CreateALGraph(&G);

ShowALGraph(&G);

DFSTraverse(&G);

return 0;

}