哈尔滨工业大学计算学部

实验报告

课程名称：数据结构与算法

课程类型：专业基础（必修）

实验项目：图结构及其应用

实验题目：最短路径算法

实验日期：2022/11/2

班级：2103101

学号：2021112864

姓名：李煦

|  |  |  |
| --- | --- | --- |
| 设计成绩 | 报告成绩 | 指导老师 |
|  |  | 苗东菁 |

**一、实验目的**

掌握Dijkstra算法和Floyd-Warshall算法基本原理，设计和实现Dijkstra算法和Floyd-Warshall算法，求解最短路径问题。

**二、实验要求及实验环境**

**实验要求：**

1． 实现单源最短路径的Dijkstra算法，输出源点及其到其他顶点的最短路径长度和最短路径。

2． 实现全局最短路径的Floyd-Warshall算法。计算任意两个顶点间的最短距离矩阵和最短路径矩阵，并输出任意两个顶点间的最短路径长度和最短路径。

3． 利用Dijkstra或Floyd-Warshall算法解决单目标最短路径问题：找出图中每个顶点v 到某个指定顶点c 最短路径。

4． 利用Dijkstra或Floyd-Warshall算法解决单顶点对间最短路径问题：对于某对顶点u 和 v，找出u 到v 和v到u的一条最短路径。

5． 以文件形式输入图的顶点和边，并以适当的方式展示相应的结果。要求顶点不少于10个，边不少于13个。

6． （选做）实现Warshall算法，计算有向图的可达矩阵，理解可达矩阵的含义；

**实验环境：**

|  |
| --- |
| 操作系统：win11 32位  编程语言：c++  编译环境：clion  编码格式：UTF-8 |

**三、设计思想**（本程序中的用到的所有数据类型的定义，主程序的流程图及各程序模块之间的调用关系、核心算法的主要步骤）

1．数据类型与函数定义

邻接图的结构:

|  |
| --- |
| typedef int Vertex; //定义节点标记类型  typedef int Edge; //定义权重类型  typedef struct { //邻接矩阵  Vertex v[size]; //用于记录顶点  Edge edge[size][size]; //用于记录权值  int n,e; //表示点的个数以及边的个数  }MTgraph; |

主要函数定义：

|  |
| --- |
| void CreatMGraph(MTgraph\*G)//邻接矩阵表示，数组表示法 |
| void show(MTgraph g1)//展示邻接矩阵 |
| void Floyd(int\*\*D,MTgraph G,int \*\*P)//Floyd算法 |
| void ShowPath(int\*\*P,int\*\*D,MTgraph G,Vertex head,Vertex tail,int flag)//展示特定路径 |
| void ShowPath(int\*\*P,int\*\*D,MTgraph G,Vertex head,Vertex tail,int flag)//展示单源路径 |
| void Dijkstra(MTgraph C,int \*D,int \*P,int \*S)//Dijkstra算法的实现 |

2．物理设计

数据储存数据结构：



本实验中还用到了大量数组存储结构。

3．逻辑设计

主程序流程图：



核心算法的主要步骤如下：

1. Dijkstra算法

采用贪心策略每一次遍历距离单源点最近的且未访问过的点，遍历所有点，根据公式Dj是否大于等于Dw+E[w,j]来进行更新，迭代所有点从而得到单源点到其余点得最短路径以及最短路径长度。

1. Floyd算法

采用动态规划思想，从第一个顶点开始，求解其余顶点到另一个顶点经过该顶点时的最短距离有没有变化，如果变化则更新，遍历所有顶点即可求出任意顶点间的最短路径和最短路径长度。

1. 寻找最短路径算法

寻找i到j的最短路径，找到最短路径矩阵中[i][j]处点，记作w，进行递归寻找iw和wj之间的最短路径，若没有则返回输出若有则再次进行递归，递归结束所得结果即为最短路径。

**四、测试结果**

输入：data.txt(源点、终点，权值)

|  |
| --- |
| 10 16 1 2 1 1 3 3 1 5 1 1 9 5 9 1 5 1 10 6 2 3 4 3 4 6 4 8 7 5 4 12 5 6 2 6 7 3 7 8 4 8 9 7 8 4 7 10 9 9 |

输出：

|  |
| --- |
| Build successfully!  # 1 3 # 1 # # # 5 6  # # 4 # # # # # # #  # # # 6 # # # # # #  # # # # # # # 7 # #  # # # 12 # 2 # # # #  # # # # # # 3 # # #  # # # # # # # 4 # #  # # # 7 # # # # 7 #  5 # # # # # # # # #  # # # # # # # # 9 #  # 1 3 9 1 3 6 10 5 6  29 # 4 10 30 32 35 17 24 35  25 26 # 6 26 28 31 13 20 31  19 20 22 # 20 22 25 7 14 25  21 22 24 12 # 2 5 9 16 27  19 20 22 14 20 # 3 7 14 25  16 17 19 11 17 19 # 4 11 22  12 13 15 7 13 15 18 # 7 18  5 6 8 14 6 8 11 15 # 11  14 15 17 23 15 17 20 24 9 #  1 1 1 1 1 1 1 1 1 1  1 1 1 1 1 1 1 1 1 1  1 1 1 1 1 1 1 1 1 1  1 1 1 1 1 1 1 1 1 1  1 1 1 1 1 1 1 1 1 1  1 1 1 1 1 1 1 1 1 1  1 1 1 1 1 1 1 1 1 1  1 1 1 1 1 1 1 1 1 1  1 1 1 1 1 1 1 1 1 1  1 1 1 1 1 1 1 1 1 1  -1 1 1 3 1 5 6 7 1 1  9 -1 2 3 1 5 6 4 8 1  9 1 -1 3 1 5 6 4 8 1  9 1 1 -1 1 5 6 4 8 1  9 1 1 5 -1 5 6 7 8 1  9 1 1 8 1 -1 6 7 8 1  9 1 1 8 1 5 -1 7 8 1  9 1 1 8 1 5 6 -1 8 1  9 1 1 3 1 5 6 7 -1 1  9 1 1 3 1 5 6 7 10 -1  所有点到3号点的最短路径以及长度为：  1->3  The length is 3  2->3  The length is 4  4->8->9->1->3  The length is 22  5->6->7->8->9->1->3  The length is 24  6->7->8->9->1->3  The length is 22  7->8->9->1->3  The length is 19  8->9->1->3  The length is 15  9->1->3  The length is 8  10->9->1->3  The length is 17  9号点和2号点之间的路径及其长度为：  9->1->2  The length is 6  2->3->4->8->9  The length is 24 |

**五、经验体会与不足**

经验：理解了算法，体会了贪心策略和动态规划策略

不足：代码不够优雅

**五、源代码部分**

|  |
| --- |
| main.c |
| #include "Dijk.h"  int main() {  MTgraph G;  int point=3;  int nod1=9;  int nod2=2;  CreatMGraph(&G);  show(G);  int \*\*D=(int\*\*)malloc(sizeof(int\*)\*(G.n+1));  int \*\*S=(int\*\*)malloc(sizeof(int\*)\*(G.n+1));  int \*\*P=(int\*\*)malloc(sizeof(int\*)\*(G.n+1));  if(D==NULL||S==NULL||P==NULL)  {  printf("fail!");  return -1;  }  for(int i=1;i<=G.n;i++)  {  D[i]=(int\*) malloc(sizeof(int)\*(G.n+1));  S[i]=(int\*) malloc(sizeof(int)\*(G.n+1));  P[i]=(int\*) malloc(sizeof(int)\*(G.n+1));  if(D[i]==NULL||S[i]==NULL||P[i]==NULL)  {  printf("fail!");  return -1;  }  }  Dijkstra\_X(G,D,P,S);  showP(D,G.n);  showP(S,G.n);  showP(P,G.n);  printf("所有点到%d号点的最短路径以及长度为：\n",point);  for(int k=1;k<=G.n;k++)  {  ShowPath\_X(P[k],G,D[k],k,point);  }  printf("\n");  printf("%d号点和%d号点之间的路径及其长度为：\n",nod1,nod2);  ShowPath\_X(P[nod1],G,D[nod1],nod1,nod2);  ShowPath\_X(P[nod2],G,D[nod2],nod2,nod1);  return 0;  } |
| #include "floyd.h"  int main() {  MTgraph G;  int nod1=9;  int nod2=2;  int point=5;  int\*\*D=NULL;  int\*\*P=NULL;  CreatMGraph(&G);  D=(int\*\*) malloc(sizeof (int\*)\*(G.n+1));  P=(int\*\*) malloc(sizeof (int\*)\*(G.n+1));  for(int i=1;i<=G.n;i++)  {  D[i]=(int\*) malloc(sizeof(int)\*(G.n+1));  P[i]=(int\*) malloc(sizeof(int)\*(G.n+1));  }  Floyd(D,G,P);  show(G);  showP(D,G.n);  showP(P,G.n);  printf("所有点到%d号点的最短路径以及长度为：\n",point);  for(int k=G.n;k>=1;k--)  {  if(k==point)continue;  ShowPath(P,D,G,k,point,1);  if(D[k][point]!=NONE)  printf("\nThe length is %d\n",D[k][point]);  }  printf("\n\n");  printf("%d号点和%d号点之间的路径及其长度为：\n",nod1,nod2);  ShowNodes(P,D,G,nod1,nod2);  return 0;  } |

|  |
| --- |
| Creategraph.c |
| //  // Created by Slater on 2022/10/30.  //  #include "Dijk.h"  void CreatMGraph(MTgraph\*G)//邻接矩阵表示，数组表示法  {  //printf("Adjacency matrix representation, array representation, array representation\n");  int w;//记录权值  int i,j;  FILE\* fp=fopen("D:\\SOPHOMORE\_1\\DSAFEcode\\Chapter4 Graph\\lab3\\Dijkstra\\graph.txt","r");//将图中结点信息保存在文件中  FILE\* fq=fopen("D:\\SOPHOMORE\_1\\DSAFEcode\\Chapter4 Graph\\lab3\\Dijkstra\\data.txt","r");  //printf("Please enter the number of nodes:");  fscanf(fq,"%d ",&G->n);//节点数目  //printf("Please enter the number of edges:");  fscanf(fq,"%d ",&G->e);//边的数目  for(i=0;i<G->n;i++)  {  fscanf(fp,"%d ",&(G->v[i]));//读入节点名称  }  fclose(fp);  for(i=1;i<=G->n;i++)  for(j=1;j<=G->n;j++)  G->edge[i][j]=NONE;  for(int k=1;k<=G->e;k++)  {  //printf("Please enter a node of the edge:");  fscanf(fq,"%d ",&i);//左顶点的位置信息  //printf("Please enter another node for the edge:");  fscanf(fq,"%d ",&j);//右顶点的位置信息  //printf("Please enter the weights:");  fscanf(fq,"%d ",&w);//记录权值  G->edge[i][j]=w;  // G->edge[j][i]=w;  }  fclose(fq);  printf("Build successfully!\n");  }  void show(MTgraph g1)//展示邻接矩阵  {  for(int i=1;i<=g1.n;i++)  {  for(int j=1;j<=g1.n;j++)  if(g1.edge[i][j]==NONE)  printf("%3c",'#');  else  printf("%3d",g1.edge[i][j]);  printf("\n");  }  printf("\n");  } |

|  |
| --- |
| Dijkstra.c |
| //  // Created by Slater on 2022/10/30.  //  #include"Dijk.h"  void Dijkstra(MTgraph C,int \*D,int \*P,int \*S)//Dijkstra算法的实现  {  int i,w;  int sum=0;  for(i=1;i<=C.n;i++)  {  D[i]=C.edge[1][i];//记录邻接矩阵  S[i]=0;  P[i]=1;  }  S[1]=1;  for(i=1;i<C.n;i++)  {  w=MinCost(D,S,C.n);  S[w]=1;  for(int v=2;v<=C.n;v++)  {  if(!S[v])  {  sum=D[w]+C.edge[w][v];  if(sum<D[v]){  D[v]=sum;  P[v]=w;  }  }  }  }  }  void Dijkstra\_X(MTgraph C,int \*\*D,int \*\*P,int \*\*S)  {  Vertex node=1;  while(node<=C.n)  {  int i,w;  int sum=0;  for(i=1;i<=C.n;i++)  {  D[node][i]=C.edge[node][i];  S[node][i]=0;  if(D[node][i]!=NONE)  P[node][i]=node;  else  P[node][i]=-1;  }  S[node][node]=1;  for(i=1;i<C.n;i++)  {  w=MinCost\_X(D,S,C.n,node);  if(w<=C.n)  {  S[node][w]=1;  for(int v=1;v<=C.n;v++)  {  if(v==node)  continue;  if(!S[node][v])  {  sum=D[node][w]+C.edge[w][v];  if(sum<D[node][v]){  D[node][v]=sum;  P[node][v]=w;  }  }  }  }  }  node++;  }  }  int MinCost(int \*D,int \*S,int n)//找到最小的路径长度对应的顶点  {  int temp=NONE;  int i;  int w=2;  for(i=2;i<=n;i++)  {  if(!S[i]&&D[i]<temp)  {  temp=D[i];  w=i;  }  }  return w;  }  int MinCost\_X(int \*\*D,int \*\*S,int n,Vertex node)//找到最小的路径长度对应的顶点  {  int temp=NONE;  int i;  int w=node+1;  for(i=1;i<=n;i++)  {  if(!S[node][i]&&D[node][i]<temp)  {  temp=D[node][i];  w=i;  }  }  return w;  } |

|  |
| --- |
| Floyd.c |
| //  // Created by Slater on 2022/10/30.  //  #include "floyd.h"  void Floyd(int\*\*D,MTgraph G,int \*\*P)//Floyd算法  {  int i,j,k,m;  for(i=1;i<=G.n;i++)  for(j=1;j<=G.n;j++)  {  D[i][j]=G.edge[i][j];  if(D[i][j]!=NONE)  P[i][j]=i;  else  P[i][j]=-1;  }//初始化所有需要用到的结构  for(k=4,m=1;m<=G.n;m++,k++)  {  if(k>G.n)k=k-G.n;  if(k<=0)k=k+G.n;  for (i = 1; i <= G.n; i++) {  if(k==i)continue;  for (j = 1; j <= G.n; j++)  {  if(k==j||i==j) continue;  if (D[i][k] + D[k][j] < D[i][j]) {  D[i][j] = D[i][k] + D[k][j];  P[i][j] = k;/\*当找到更优化的路径时更新距离并更新前驱节点\*/  }  }  }  }  } |

|  |
| --- |
| Show.c |
| //  // Created by Slater on 2022/10/31.  //  #include "Dijk.h"  void ShowPath(int\*P,MTgraph G,int\*D,int head)  {  int temp=2;  int count=1;  int p[size][size];  int i,j;  int length[size]={0};  while(count<=G.n) {  temp = count;  while (temp != -1) {  p[count][length[count]] = temp;  temp = P[temp];  length[count]++;  }  p[count][length[count]] = head;  count++;  if(count==head)count++;  }  for(i=1;i<=G.n;i++)  {  if(i==head)continue;  if(D[i]==NONE)  {  //printf("There is no path");  continue;  }  for(j=length[i]-1;j>0;j--)  {  printf("%d->",p[i][j]);  }  printf("%d\n",p[i][j]);  printf("The length is %d\n",D[i]);  }  }  void ShowPath\_X(int\*P,MTgraph G,int\*D,int head,int tail)  {  int temp=1;  int count=1;  int p[size][size];  int i,j;  int length[size]={0};  while(count<=G.n) {  temp = count;  while (temp != -1) {  p[count][length[count]] = temp;  temp = P[temp];  length[count]++;  }  p[count][length[count]] = head;  count++;  if(count==head)count++;  }  if(D[tail]==NONE)  {  //printf("There is no path");  return;  }  else  {  for(j=length[tail]-1;j>0;j--)  {  printf("%d->",p[tail][j]);  }  printf("%d\t",p[tail][j]);  printf("\nThe length is %d\n",D[tail]);  }  }  void ShowNode(int\*\*P,int\*\*D,MTgraph G,int nod1,int nod2)  {  ShowPath\_X(P[nod1],G,D[nod1],nod1,nod2);  ShowPath\_X(P[nod2],G,D[nod2],nod1,nod2);  }  void showP(int\*\*P,int n)  {  int i,j;  for(i=1;i<=n;i++)  {  for(j=1;j<=n;j++)  {  if(P[i][j]!=NONE)  printf("%3d",P[i][j]);  else  printf("%3c",'#');  }  printf("\n");  }  printf("\n");  } |
| //  // Created by Slater on 2022/10/31.  //  #include "floyd.h"  void showP(int\*\*P,int n)  {  int i,j;  for(i=1;i<=n;i++)  {  for(j=1;j<=n;j++)  {  if(P[i][j]!=NONE)  printf("%3d",P[i][j]);  else  printf("%3c",'#');  }  printf("\n");  }  printf("\n");  }  void ShowPath(int\*\*P,int\*\*D,MTgraph G,Vertex head,Vertex tail,int flag)  {  int temp;  int path[size]={0};  int length=0;  int i;  int \*p=(int\*)malloc(sizeof(int)\*(G.n+1));  for(i=1;i<=G.n;i++)  {  p[i]=P[head][i];  }  temp=p[tail];  if (temp==-1)  {  //printf("There is no path between %d->%d\n",head,tail);  return ;  }  while(temp!=head)  {  path[length]=temp;  temp=p[temp];  length++;  }  if(flag==1)  printf("%d->",head);  for(i=length-1;i>=0;i--)  {  printf("%d->",path[i]);  }  if(G.edge[path[0]][tail]!=NONE)  {  printf("%d\t",tail);  }  else  {  ShowPath(P,D,G,path[0],tail,0);  }  }  void ShowNodes(int\*\*P,int\*\*D,MTgraph G,int nod1,int nod2)  {  ShowPath(P,D,G,nod1,nod2,1);  if(D[nod1][nod2]!=NONE)  printf("\nThe length is %d\n",D[nod1][nod2]);  ShowPath(P,D,G,nod2,nod1,1);  if(D[nod2][nod1]!=NONE)  printf("\nThe length is %d\n",D[nod2][nod1]);  } |