贪心算法的实现与思考

摘 要

本文主要通过贪心算法的基本概念,来引出常见的Dijisktra算法,Prim算法，Kruskal算法三种贪心算法的含义，探讨并研究了该三种贪心算法的基本思路及利用Matlab软件实现三个算法过程,并通过三者其实现过程进行相应比较得出每种贪心算法的适应场景和明显的局限性。在实现算法的基础上对于生成的最短路径和最小生成树进行可视化处理使结果更加直观有效。

关键词：贪心算法；Dijisktra算法；Prim算法；Kruskal算法；最小生成树；最短路径；Matlab软件；

## 1引言

通常情况下，在现实生活中我们遇到的问题涉及到许多因素的影响，可能不能用一些简单的数学计算而得到最佳的方案，在此情况下贪心算法就是处理该问题的最佳处理方式，因为贪心算法能在最初解的情况下通过算法逐步逼近给定的目标，以便更快的得到更好的解。所以本文主要通过常见的Dijistra算法，Prim算法，Kruskal算法三种贪心算法的实现，了解贪心算法的具体应用场景，通过分析并探究该三种算法得出贪心算法的优缺点，和在某些场景下的局限性。

## 2、基本概念

### 2.1贪心算法的基本含义

贪心算法是一种能够得到某种度量意义下的 最优解的分级处理方法, 它总是做出在当前看来是最优的选择, 也就是说贪心策略并不是从整体上加以考虑,它所做出的选择只是在某种意义上的局部最优解算法。

### 2.1 Dijistra算法

Dijistra算法是一种典型的单源最短路径的贪心算法，用于计算图（数据结构）中一个顶点到其它各顶点的最短路径。其基本思路为设 G = ＜V，E，W＞ 是一个带权图，把图中顶点集合V分成两个集合，第一个集合S为已求出最短路径的顶点集合，初始时S中只有一个源点v0，即 S = {v0} ; 另外一个集合用 U 表示，该集合为未确定最短路径的顶点集合 U = V － S． 接下来每求得一条最短路径 ，就将从 U 中新找到的点 k 加入到集合 S 中，S = S∪{k}，同时U=U－{k} ，直到图中全部顶点都加入到 S 中，算法就结束了．上述过程是按最短路径长度 的递增次序依次把第二个集合中的顶点加入 S 中．在加入的过程中，总保持从源点 v0 到 S 中各 顶点的最短路径长度不大于从源点 v0 到 U 中任 何顶点的最短路径长度．

### 2.2Prim算法

Prim算法的本质为生成最小生成树的一种贪心算法，用于计算连通图的最小权值路径。基本思路为普里姆算法在找最小生成树时，将顶点分为两类，一类是在查找的过程中已经包含在生成树中的顶点（假设为 A 类），剩下的为另一类（假设为 B 类）。对于给定的连通网，起始状态全部顶点都归为 B 类。在找最小生成树时，选定任意一个顶点作为起始点，并将之从 B 类移至 A 类；然后找出 B 类中到 A 类中的顶点之间权值最小的顶点，将之从 B 类移至 A 类，如此重复，直到 B 类中没有顶点为止。所走过的顶点和边就是该连通图的最小生成树。

### 2.3Kruskal算法

Kruskal算法和Prim算法很类似都是为了在连通图中实现最小生成树的实现，但Kruskal算法的基本思路是选择无向加权连通图G中权值最小且不和已选择的边形成圈的边, 并将其添加到边集E中;否则 就选择下一条边。直到边集E中有n-1 条边为 止( 图G中有n个顶点) 。 在无向加权连通图G=( V,E) 中, V= {V1, V2, 。。。, Vn}是n个顶点构成的集合, E= {e 1, e2, 。。。, em}是m条边构成的集合, ω={ω1, ω 2, 。。。, ωm}是每条边对应的权值。构造出的最小 生成树记作T=G{e1, e2, 。。。, en-1}

## 3、程序设计

### 3.1 Dijistra算法的实现

#### 3.1.1问题陈述

设A、B、C、D、E、F表示一个乡的6个村庄,弧上的权值表两村之间的距离。现要在这6个村庄中选择一个村庄建一所医院,须求出医院建在那个村庄的距离最远的村庄的距离最短。这个问题是一个典型的最短路径问题示意图如下：



图 1村庄路径示意图

#### 3.1.2算法原理

①初始时，S 只包含源点，即 S = { v0 } ，v0 的 距离为 0． U 包含除 v0 外的其他顶点，即: U = { v1 ，v2 。。。，vn － 1 } ． 定义集合 U 中的顶点的距离: 若v0 与 U 中顶点 k 相邻，则 dist( k) = w( v0 ，k) 正常 有权值，若 v0 与 k 点不相邻，则 dist( k) = ∞ ．

②从 U 中选取一个点 k 加入 S 中，选择公式 dist( k) = min{ dist( k) | k∈U} ，把 k 加入 S 中 ( 该选定的距离就是 v0 到 k 的最短路径长度) ． 此时 S = S∪{ k} ，同时 U 集合中删除 k 点，即 U = U － { k} ．

③以 k 为新考虑的中间点，修改 U 中各顶点 的距离; 若从源点 v0 到顶点 u 的距离( 经过顶点 k) 比原来距离短，则修改顶点 u 的距离值，否则 u 的距离 值 不 变，修 改 公 式 dist ( u) = min { dist ( u) ，dist( k) + dist( k，u) }

④重复步骤②和③直到 S = V，算法停止．

#### 3.1.3算法步骤

( 1) S = { A}，此时最短路径: A→A: 0; U = { B，C，D，E，F} 。U 中点 的 距 离: Dist ( B) = 6，Dist ( C) = 3， Dist( D) = ∞ ，Dist( E) = ∞ ，Dist( F) = ∞ 。以 A 作为中间点在 U 中开始找 k 点，依据 dist( k) = min{ dist( k) | k∈U} ． U 中被选点 k: k←C。

( 2) S = { A，C} ，U = { B，D，E，F} ，此时最短 路径: A→A: 0 A→C: 3。修改新集合 U 中点距离: Dist( B) = min( 6，3 + 2) = 5，Dist( D) = min( ∞ ，3 + 3) = 6， Dist( E ) = min ( ∞ ，3 + 4 ) = 7，Dist ( F ) = ∞。以 C 作为中间点在 U 中开始找新的 k 点，依据同① k←B。

(3) S = { A，C，B} ，U = { D，E，F} ，此时最短 路径: A→A: 0 A→C: 3 A→C→B: 5． 修改新集合 U 中点距离: Dist( D) = 6 ,Dist ( E) = 7, Dist( F) = ∞ ． 以 B 作为中间点开始找新的 k 点，依据同① k←D．

(4) S = { A，C，B，D} ，U = { E，F} ，此时最短路径: A→A: 0 ,A→C: 3 ,A→C→B: 5, A→C→ D: 6。修改新集合 U 中点距离: Dist( E) = 7 ,Dist ( F) = min( ∞ ，6 + 3) = 9． 以 D 作为中间点开始找找新的 k 点，依据同 ① k←E．

( 5) S = { A，C，B，D，E}，U = { F} ，此时最短 路径: A→A: 0 ,A→C: 3 A→C→B: 5 A→C→ D: 6 A→C→E: 7。修改新集合 U 中点距离: Dist( F) = 9。以 E 作为中间点开始找找新的 k 点，此时只有一个点了k←F。

(6) S = { A，C，B，D，E，F} = V，U =空集，此时 最短路径: A→A: 0 A→C: 3 A→C→B: 5 A →C→D: 6 A→C→E: 7 A→C→D→F: 9，算法结束。

具体解法表如图：

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 步骤 | A(源) | B | C | D | E | F |
| 1 | （0,A） | Dist(B)=6 | Dist(C)=3 | Dist(D)=∞ | Dist(D)=∞ | Dist(D)=∞ |
| 2 |  | Dist(B)=5 | (3,A) | Dist(D)=6 | Dist(D)=∞ | Dist(D)=∞ |
| 3 |  | (5,C) |  | Dist(D)=6 | Dist(D)=∞ | Dist(D)=∞ |
| 4 |  |  |  | (6,C) | Dist(D)=∞ | Dist(D)=∞ |
|  |  |  |  |  | (7,C) | Dist(D)=∞ |
|  |  |  |  |  |  | (9,D) |
| 最短路径值 | 0 | 5 | 3 | 6 | 7 | 9 |

表格 1步骤示意表

由上表可知源点A到其余个点的最短路径为:

|  |
| --- |
| A→A: 0 |
| A→C: 3 |
| A→C→B: 5 |
| A→C→ D: 6 |
| A→C→E: 7 |
| A→C→D→F: 9 |

表格 2结果示意表

#### 3.1.4算法流程图



图 2 Dijistra算法流程图

#### 3.1.5算法及实例实现代码

1.Dijistra算法的Dijistra.m函数脚本如下：

dist：起点与终点之间的最短距离值

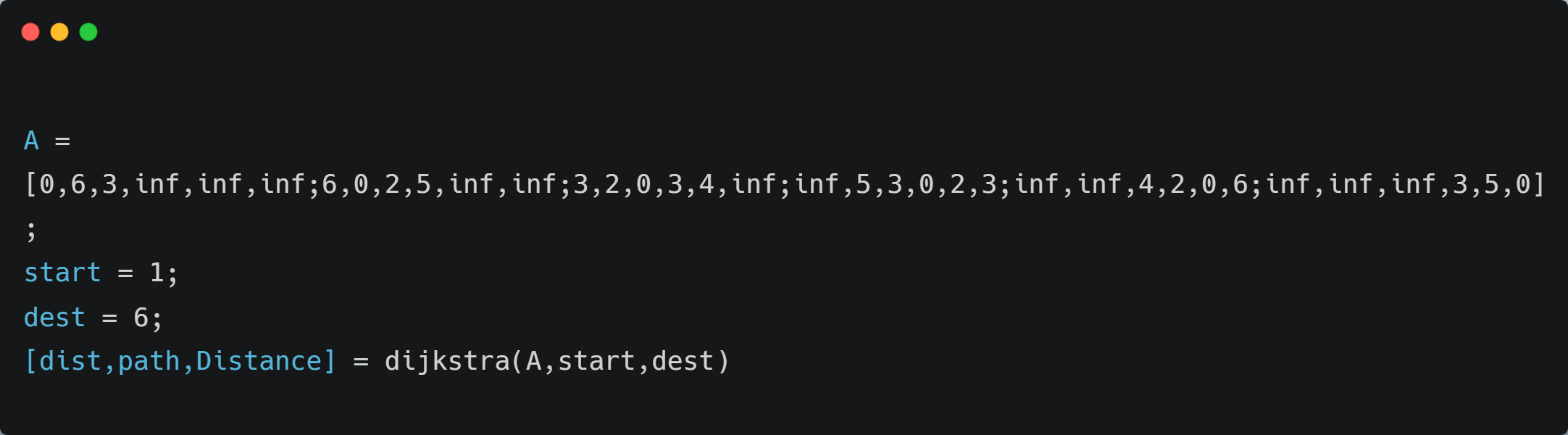
path：最短路径索引

Distance：最短路径下的距离值

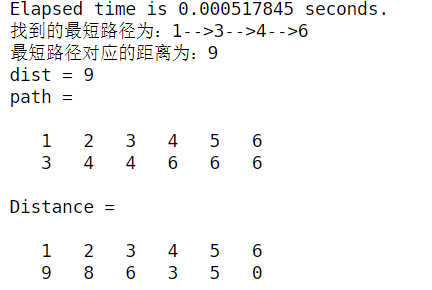


2.Dijistra算法的输入验证代码

将题中所给的带权图利用邻接矩阵来表示，再利用Matlab的矩阵运算输入到Dijistra算法当中得出最短路径。其中A-代表邻接矩阵；start-代表源点；dest-代表终点；



代码运行结果如下：



此时最小路径图为：



图 3最小路径示意图

### 3.2 Prim算法的实现

#### 3.2.1问题陈述

在 n 个带权路径城市之间(如图？所示)，可以建立 n(n-1)/2 条 路 径 ，这 些 路 径 构 成 一 个 无 向 完 全 图 。 在 这 些 可 能 的 路 径 中，选择 n-1 条路径构成一个通信网络，要求这个网络连通每个城市，并且路径总的权值最低，这就涉及到图的最小生成树 (minimum spanning tree，MST)问题[[1]](#footnote-1)。



图 4问题示意图

#### 3.2.2算法原理

Prim算法核心原理在于假设一个S是连通图G=(V,{E})上最小生成树中的边的集合。

（1）初始化U={u0}((u0∈V)，S={};

（2）对于任意的u∈U,v∈V-U所构成的边（u，v）∈E,寻找一条权值最小的边（u0,v0）,并将其加入到S集合当中，同时将v0写入到U；

（3）假如U=V，此时已经生成了n-1条边构成变的集合S，此时S即为需要生成的连通图G的最小生成树。但是U!=V时继续进行（2）操作直到U=V;

#### 3.2.3算法步骤

1．假如以顶点A为出发点，顶点B，D，G的权值分别为2，3，3,所以对于顶点A来说，到顶点B的权值最小，所以将顶点A加入到生成树

2．继续分析与顶点A和顶点B相邻的所有顶点（包含C,D,E）,其中权值最小的顶点D，将顶点D添加到生成树

3．紧接分析顶点A,B,D相邻的所有顶点中权值最小的顶点，发现为顶点G在添加到生成树中。

4．依次比较A，B，D，G相邻的所有顶点中的权值最小的点为C和E顶点加入到生成树

5．再通过比较权值将顶点F加入到生成树，此时如下图所示生成树即为最小生成树。



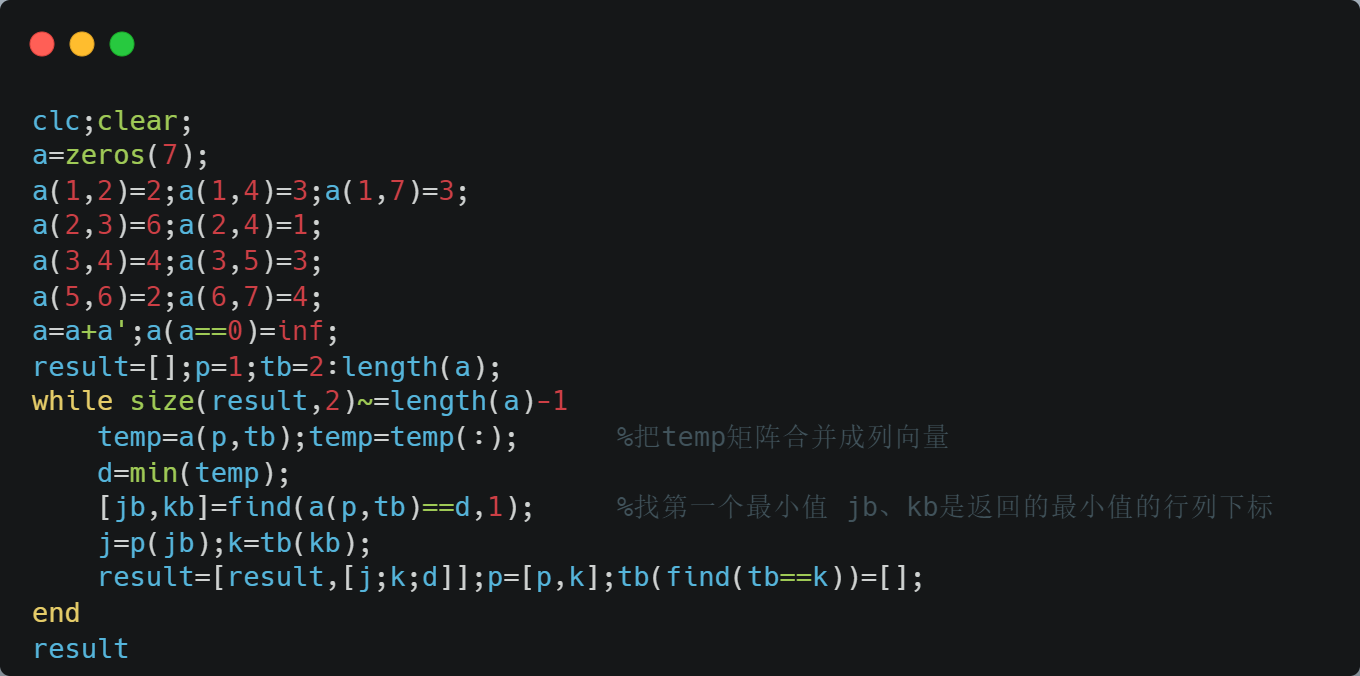
图 5结果示意图

#### 3.2.4算法流程图



图 6 Prim算法流程图

#### 3.2.5算法及实例实现代码



结果展示：

result第一行表示起点，第二行表示终点，第三行表示权值。



为了得到更加直观的最下生成树，如下图将生成树的生成过程：



图 7实验过程流程图

### 3.3 Kruskal算法的实现

#### 3.3.1问题陈述

Kruskal算法与Prim算法都是为了处理最小生成树问题，这里我们处理与上面的城市路径最短的问题。[[2]](#footnote-2)

#### 3.3.2算法原理

( 1) 初始化顶点集A={ }, 边集E= { }, 所有边的权值的和wT =0 。

(2) 将所有的边按从小到大的顺序排列, 记 为E′。

(3) 选择没有在边集E中、权值最小、不和边集E中的边构成圈, 且wT +w≤wT +w 的 边 ( 是连接顶点i和j的边,是的权值,是没有选择的边中的其它任何一边, 是边的权值) , 则将顶点i、j中没有在点集A 中的点添加到A中, 并将边 添加到边集E中; 否则不选择这条边。

(4) 重复第3 步直到顶点集有n个顶点, 边集中有n-1条边。

#### 3.3.3算法步骤

第一步：先将图中所有的边按照权值进行非降序排列

|  |  |  |  |
| --- | --- | --- | --- |
|  | 起点 | 终点 | 权值 |
| Edges[1] | B | D | 1 |
| Edges[2] | A | B | 2 |
| Edges[3] | E | F | 2 |
| Edges[4] | A | D | 3 |
| Edges[5] | A | G | 3 |
| Edges[6] | C | E | 3 |
| Edges[7] | D | C | 4 |
| Edges[8] | F | G | 4 |
| Edges[9] | B | C | 6 |

表格 3权值排序表

第二步：从图中所有的边中选择可以构成最小生成树的边。

1. 选择权值最小的边B-D：没有成环则添加入集合E
2. 选择权值最小A-B和E-F没有成环则添加入集合E
3. 选择边A-G,C-E没有成环加入集合E
4. 选择边D-C可以生成最小生成树

（5） 生成的最小生成树如下图：



图 8结果示意图

#### 3.3.4算法流程图



图 9 Kruskal算法流程图

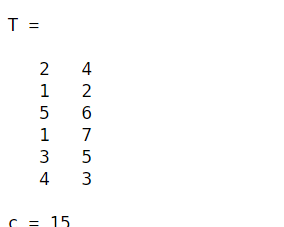
#### 3.3.5算法及实例实现代码

边权矩阵，每一列都表示一条边，从上到下分别为两个顶点以及它们边的权值

sortrows函数对某一列进行比较排序，所以我们先转置b矩阵，然后对第三列也就是权值进行排序

t数组用来标记选中的边，k用来计数，T矩阵用来存储选中的边，c计算最小生成树的总长度





为了得到更加直观的最下生成树，如下图将生成树的生成过程：



图 10实验过程示意图

### 3.4贪心算法的分析

#### 3.4.1 贪心算法普遍局限性

(1) 不能保证求得的最后解是最佳的。由于 贪心策略总是采用从局部看来是最优的选择, 因此 并不从整体上加以考虑。

(2) 贪心算法只能用来求某些最大或最小解 的问题。从前面的讨论中, 找零钱问题要求得到最 小数量采用贪心算法是可行的, 但是在另外一个求 解权值最小路径时采用贪心算法得到的结果并不 是最佳。

(3) 贪心算法只能确定某些问题的可行性范围。

#### 3.4.2 Prim算法与kruskal算法的对比

从策略上来说，Prim算法是直接查找，多次寻找邻边的权重最小值，而Kruskal是需要先对权重排序后查找的，prim:该算法的时间复杂度为O(n2)。与图中边数无关，该算法适合于稠密图。kruskal:需要对图的边进行访问，所以克鲁斯卡尔算法的时间复杂度只和边有关系，可以证明其时间复杂度为O，适合稀疏图。所以说，Kruskal在算法效率上是比Prim快的，因为Kruskal只需一次对权重的排序就能找到最小生成树，而Prim算法需要多次对邻边排序才能找到。

#### 3.4.3 Dijistra算法

dijkstra可以很快求出单源最短路径，并且拥有较稳定的时间复杂度，能很好地处理大多数最短路问题。然而，dijkstra的局限性体现在了负边和极端链状如果一个图中存在了负边，过一次负边路径长度会减小，dijkstra同样会出问题

## 4、结论

贪心算法是很常见的算法, 贪心策略是最接近 人的日常思维的一种解题策略, 虽然它不能保证求 得的最后解一定是最佳的, 但是它可以为某些问题 确定一个可行性范围, 因此贪心策略（Dijistra算法，Prim算法和Kruskal算法）在各级各类信息学竞赛, 尤其在对NPC类问题的求解中发挥着 越来越重要的作用。对于一个问题的最优解只能 用穷举法得到时, 用贪心算法是寻找问题最优解的较好算法。总之, 充分利用贪心算法的优势, 从局 部最优出发, 构造贪心策略比较容易,且简单易行。[[3]](#footnote-3)

**参考文献**

1常友渠, 肖贵元和曾敏。 《贪心算法的探讨与研究》。 重庆电力高等专科学校学报, 期 03

2江波, 和张黎。 《基于Prim算法的最小生成树优化研究》。 计算机工程与设计 30 期

3王伟, 和孟思燕。 《Kruskal算法的研究与改进》。 重庆文理学院学报(自然科学版) 29,期

**附录**

主要代码源：

Dijistra算法：

Dijistra.m

|  |
| --- |
| function [dist,path,Distance] = dijkstra(A,start,dest)  % 测试数据 A =[0,12,inf,inf,inf,16,14;12,0,10,inf,inf,7,inf;inf,10,0,3,5,6,inf;inf,inf,3,0,4,inf,inf;inf,inf,5,4,0,2,8;16,7,6,inf,2,0,9;14,inf,inf,inf,8,9,0];  % 测试数据 start = 1;  % 测试数据 dest = 4;  % 计算程序运行时间  tic %开始计时    % 初始化操作  p = size(A,1); %计算顶点数目  S(1) = dest; %初始化集合S，已加入到路径中的顶点编号  U = 1:p; %初始化集合U，未加入到路径中的顶点编号  U(dest) = []; %删除终点编号  Distance = zeros(2,p); %初始化所有顶点到终点dest的距离  Distance(1,:) = 1:p; %重赋值第一行为各顶点编号  Distance(2,1:p) = A(dest,1:p); %重赋值第二行为邻接矩阵中各顶点到终点的距离  new\_Distance = Distance;  D = Distance; %初始化U中所有顶点到终点dest的距离  D(:,dest) = []; %删除U中终点编号到终点编号的距离  path = zeros(2,p); %初始化路径  path(1,:) = 1:p; %重赋值第一行为各顶点编号  path(2,Distance(2,:)~=inf) = dest; %距离值不为无穷大时，将两顶点相连    % 寻找最短路径  while ~isempty(U) %判断U中元素是否为空  index = find(D(2,:)==min(D(2,:)),1); %剩余顶点中距离最小值的索引  k = D(1,index); %发现剩余顶点中距离终点最近的顶点编号    %更新顶点  S = [S,k]; %将顶点k添加到S中  U(U==k) = []; %从U中删除顶点k    %计算距离  new\_Distance(2,:) = A(k,1:p)+Distance(2,k); %计算先通过结点k，再从k到达终点的所有点距离值  D = min(Distance,new\_Distance); %与原来的距离值比较，取最小值    %更新路径  path(2,D(2,:)~=Distance(2,:)) = k; %出现新的最小值，更改连接关系，连接到结点k上    %更新距离  Distance = D; %更新距离表为所有点到终点的最小值  D(:,S) = []; %删除已加入到S中的顶点  end  dist = Distance(2,start); %取出指定起点到终点的距离值  toc %计时结束    % 输出结果  fprintf('找到的最短路径为：');  while start ~= dest %到达终点时结束  fprintf('%d-->',start); %打印当前点编号  next = path(2,start); %与当前点相连的下一顶点  start = next; %更新当前点  end  fprintf('%d\n',dest);  fprintf('最短路径对应的距离为：%d\n',dist);  end |

Prim.m

|  |
| --- |
| clc;clear ;  a=zeros( 7 );  a(1,2)=2;a(1,4)=3;a(1,7)=3;  a(2,3)=6;a(2,4)=1;  a(3,4)=4;a(3,5 )=3;  a(5,6)=2;a(6,7 )=4;  a=a+a' ;a( a==0)=inf ;  result=[];p=1; tb=2: length(a);  while size( result,2 )~=length(a)-1  temp=a(p, tb);temp=temp( : ); %把 temp矩阵合并成列向量  d=min( temp);  [ jb,kb]=find(a(p,tb)==d,1);  %找第一个最小值jb、 kb是返回的最小值的行列下标  j=p( jb);k=tb(kb);  result=[result,[j;k;d]];p=[p,k];tb( find(tb==k))=[ ];  end  result |

Kruskal.m

|  |
| --- |
| %边权矩阵，每一列都表示一条边，从上到下分别为两个顶点以及它们边的权值  b = [2 1 5 1 1 3 4 6 2;  4 2 6 4 7 5 3 7 3;  1 2 2 3 3 3 4 4 6];  %sortrows函数对某一列进行比较排序，所以我们先转置b矩阵，然后对第三列也就是权值进行排序  [B,i]=sortrows(b',3);  %再将其转置回来  B=B';  %m为边的条数，n为点的个数  m=size(b,2);n=7;  %t数组用来标记选中的边，k用来计数，T矩阵用来存储选中的边，c计算最小生成树的总长度  t=1:n;k=0;T=[];c=0;  for i=1:m  if t(B(1,i))~=t(B(2,i))  k=k+1;T(k,1:2)=B(1:2,i),c=c+B(3,i);  tmin=min(t(B(1,i)),t(B(2,i)));  tmax=max(t(B(1,i)),t(B(2,i)));  for j=1:n  if t(j)==tmax  t(j)=tmin;  end  end  end  if k==n-1  break;  end  end  T,c, |

1. [↑](#footnote-ref-1)
2. 王和孟, 《Kruskal算法的研究与改进》。 [↑](#footnote-ref-2)
3. [↑](#footnote-ref-3)