**论文编号：**

**关于图的最短路径算法分析与优化**

王加炜

（同济大学,软件学院,上海 200000）

**摘要**: 在计算机网络、交通运输工程、通信工程等各种应用中，常常需要计算图中从某个源点到其余各顶点的最短路径或各顶点之间的最短路径。在此基础上，研究人员提出了多种算法。目前常用的路径规划算法有Dijkstra算法、启发式算法(以A\*算法表示)、SPFA算法等等。这些算法在空间和时间复杂度、适用性和可靠性方面都有各自的优势。本文对上述几种经典的最短路径算法与改进给出全面的阐述，并提出笔者对于算法改进的个人想法。

**关键词: s**单源最短路径；Dijkstra算法；A\*算法；SPFA算法；

Analysis and optimization of the shortest path algorithm for Fig

*Wang JiaWei*

(1. Tongji University, ShangHai 200000, China)

**Abstract:** In various applications, such as computer network, transportation engineering, and communication engineering, it is often necessary to calculate the shortest path from one source point to the remaining vertices in the graph. Based on this result, the researchers propose multiple algorithms. The current commonly used path planning algorithms include Dijkstra algorithm, heuristic algorithm (expressed by A \* algorithm), SPFA algorithm, and so on. These algorithms have their advantages in spatial and temporal complexity, applicability, and reliability. This paper gives the comprehensive elaboration of the above several classic shortest path algorithm and improvement, and puts forward the author's personal idea for the algorithm improvement.

—————————————

# 1.引言

在现实生活,很多领域都需要解决最短路径问题。例如在组织生产中，将各工序衔接好，使生产任务完成得既快又好；在现有交通网络中，安排合理的交通路径，使得调运的物资数量多而费用小;在企业管理中，制定合理的管理计划 和设备购置计划，使收益大而费用小；这些问题，本质上来看都是最短路径问题。 求解给定网络图中某两点（一般常指始点和终点）之间的 最短路径问题，广泛应用于各个领域中，例如计算机网络路由 算法、机器人探路、交通路线导航、游戏设计等。

最短路径问题的目标是找到一对节点之间的最短路径。在一个图中，可能有多条路径可以从一个节点(称之为起始节点)到另一个节点(目标节点)，最短路径问题就是从中确定一条使这条路径上每条边的权重之和最小的路径。事实上，最短路径的相关算法的研究已经持续了较长时间，也取得了一定的突破。由于现代的信息技术的迅猛发展，寻找最短路径的算法不断被应用到实际工程问题的解决中，这些工程中的应用也不断促进着算法的完善和发展。

近数十年来，寻找最短路径问题已经发展成为一个计算机领域的热门课题。路径搜索方法主要是广度或深度优先搜索算法，也是最短路径算法的基础。荷兰科学家E. W. Dijkstra于1959年提出了赋权图中寻找图中某个起始节点到其他节点的最短路径的经典Dijkstra算法，之后的研究者不断提出对其的优化。现阶段常用的路径规划方法有:Dijkstra算法，启发式算法（以A\*算法为代表），SPFA算法,Bellman-Ford算法等等，这些算法在空间、时间复杂度、适用性和可靠性上各具特色。

笔者将对这几种基本的求最短路径算法给出基本的阐释，同时对于这些方法提出优化和改进方案。

# 2.数据结构——图的介绍

## 2.1基本概念

 图是由顶点集V和边集E组成，记作G=(V, E) ( G：Graphic图形,V：Vertex顶点,E：Edge 边)

     如果边是有方向的则称为有向图，如果边没有方向则称为无向图。本篇文章所介绍的所有算法都是基于无向图的。

对图中的边赋予具有一定意义的数值(路程、费用等等)的图称为带权图。本篇文章介绍的所有算法都基于加权图，即网络由两个集合组成：包含n个节点的节点集合V和包含m条边的边集合E，每条边连接两个节点（i，j）。对于给定的边，边的权重表示为。

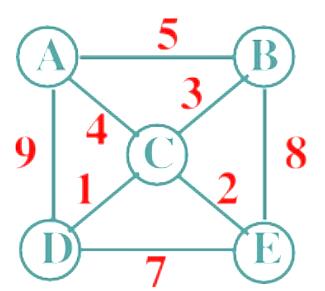


图2-1 交通路网抽象示意图

## 2.2存储结构

### 2.2.1邻接矩阵

邻接矩阵是表示图中顶点之间相邻关系的矩阵。设G=(V,E)是具有n个顶点的图，则G的邻接矩阵是具有如下定义的n阶方阵：

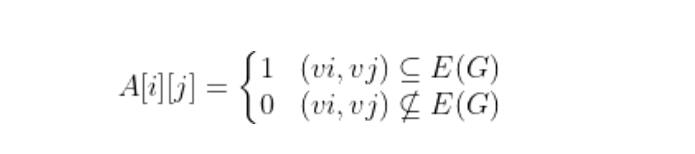


图2.21邻接矩阵存储方式

### 2.2.2邻接表

  邻接表是一种链式存储结构，类似于链表数组。

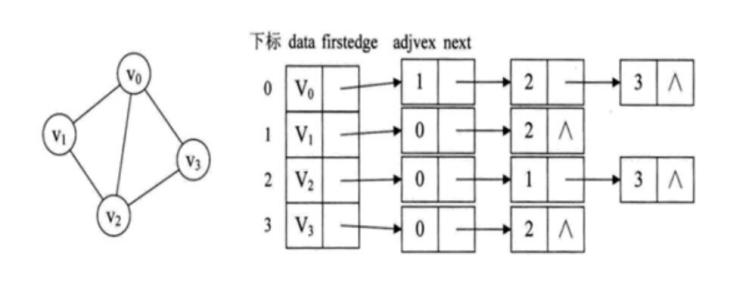


图2.2.2邻接表存储结构

## 2.3最短路径

定义路径P=<>为图中从起始节点到终止节点的经节点的一条路径。若节点各不相同，则记P为一条简单路径。若路径P的权值是图G中从到的所有简单路径中最小的，则称P为从到的最短路径。

下面以图2.3为例：一段交通路网有5个交通路口，标号1~5,1与2,3,4之间分别距离10m,3m,20m，2与3之间距离2m，2与4之间距离5m，3与5之间分别距离15m,4与5之间的距离是9m。

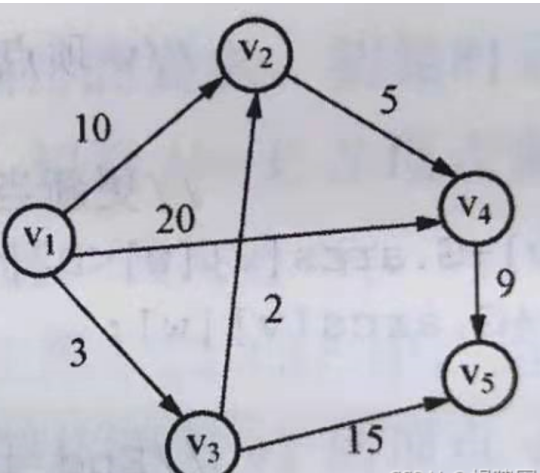


图2.3 带权图路径示意图实例

# **3. 最短路径算法**

## 3.1 路径搜索算法

路径搜索技术是各大交通路网中最短路径算法的基础，现在的路径搜索技术主要分为深度优先搜索算法（DFS）和广度优先搜索算法（BFS）。

### 3.1.1深度优先搜索算法

深度优先搜索算法与树结构的前序遍历相似。在交通路网最短路径问题中，其思想可以描述为：基于边（弧）的权值大小不断向前搜索，如果搜索不通就返回，重新选择路径向前搜索，直到找到要求的节点。

深度优先搜索是由不断探查和回溯构成的，先访问当前节点，然后在其所有临界顶点中找到未被访问过的顶点作为下一个探查的节点，向深处不断地搜索。若所有邻接顶点均已被访问过，则回退一步，将前一步所访问的节点作为当前节点。重复操作直至已经搜索完所有从起始节点可达的节点。

深度优先算法的步骤如下：

从图G中任选一顶点v作为初始出发点，首先访问出发点v，并将其标记为已访问过；然后依次搜索v的每个邻接点w，若w未曾访问过，则以w作为新的出发点，继续进行深度优先遍历，直到图中所有和v有路径相通的顶点都被访问到；若此时仍有顶点未被访问到（非连通图），则另选一个未访问过的顶点作为起点，重复上述过程，直到图中所有顶点都被访问到为止。

若采用邻接矩阵存储n个结点，e条边的图，遍历的时间复杂度为O(n+e),查找顶点所有的边的时间复杂度为O(n)，总的时间复杂度O()。

算法示意图如下图所示：

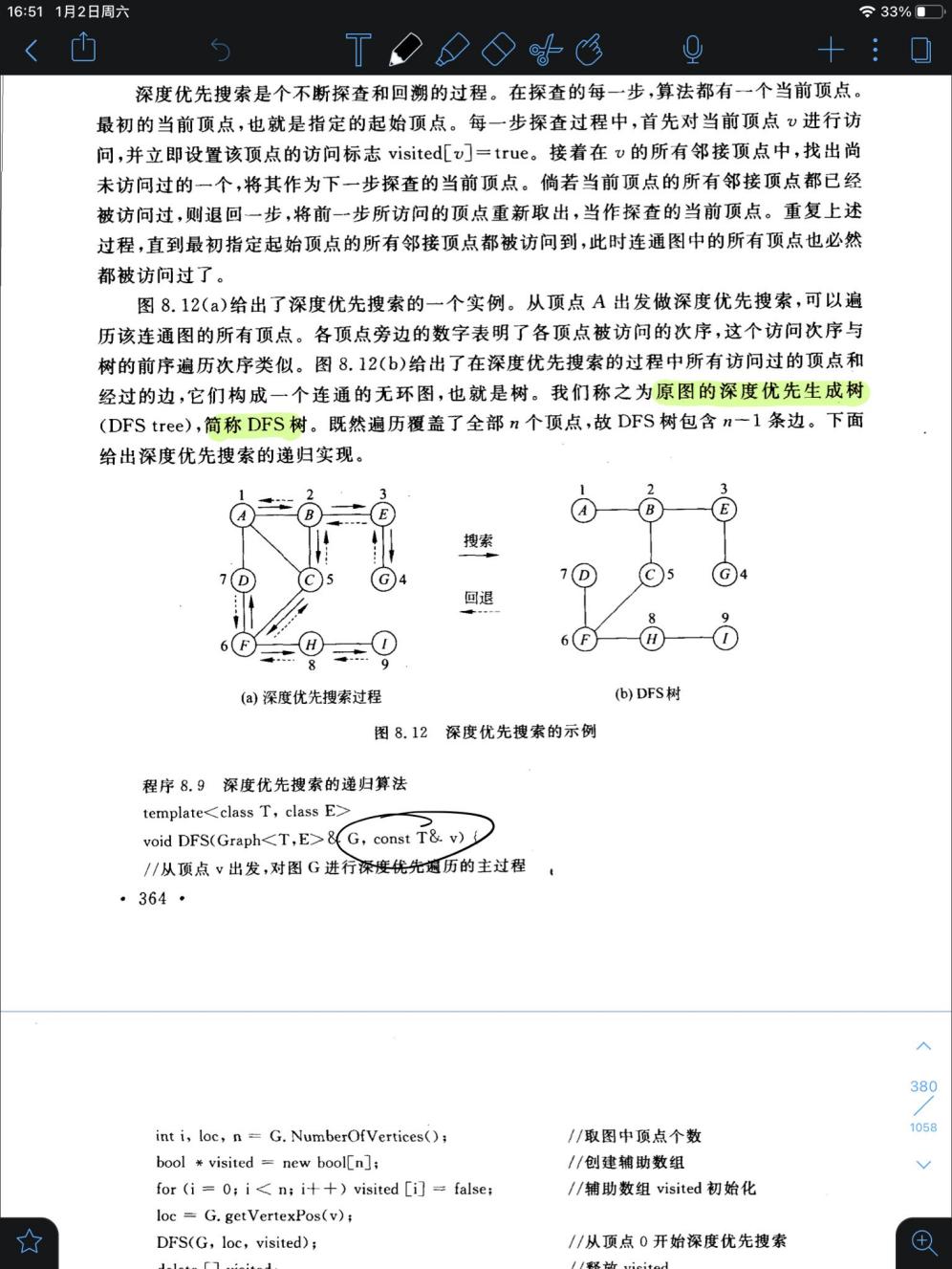


图3.1.1深度优先算法示例

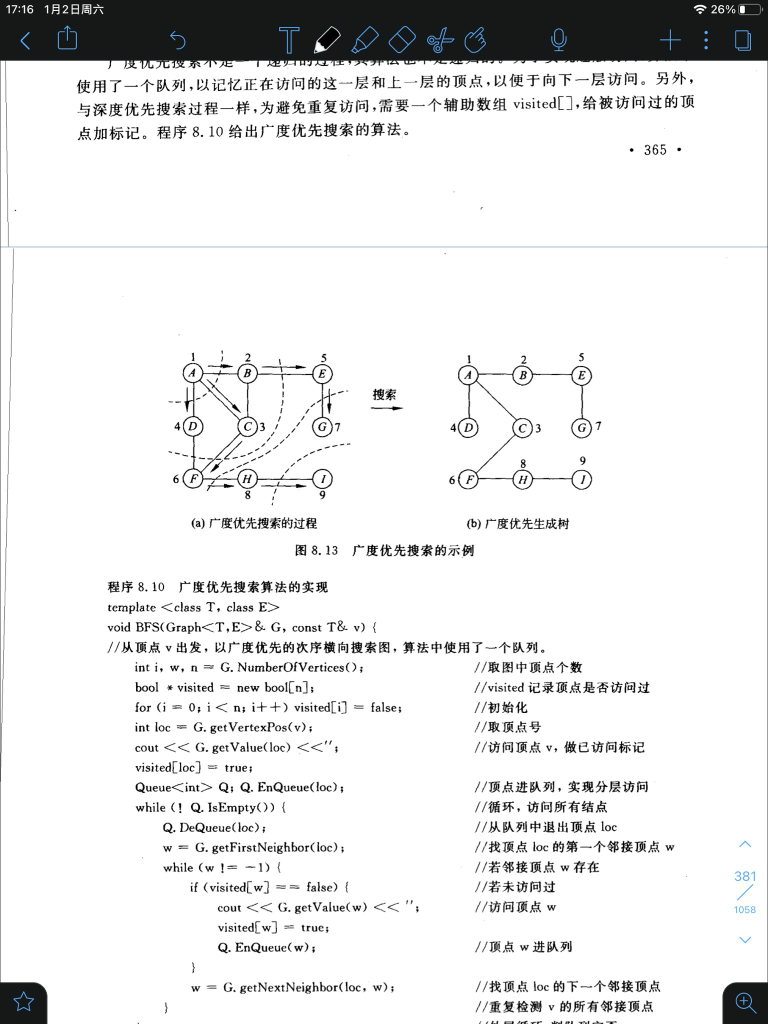
### 3.1.2广度优先搜索算法

广度优先搜索是一种按层次遍历的方法，基本思想是：从图G中任选一顶点Vi作为初始出发点，首先访问出发点Vi，接着依次访问Vi的所有未被访问过的邻接点Vi1，Vi2，...，Vit并均标记为已访问过，然后再按照Vi1，Vi2，...，Vit的次序，访问每一个顶点的所有未被访问过的邻接点并均标记为已访问过，依次类推，直到图中所有和Vi有路径相通的顶点都被访问到；若此时仍有顶点未被访问到（非连通图），则另选一个未访问过的顶点作为起点，重复上述过程，直到图中所有顶点都被访问到为止。

广度优先搜索算法没有探查和回溯的过程，该算法逐层地检查图中的所有节点以找寻结果，换句话说，它粗暴地遍历整个图，直到找到目标结果。为实现逐层访问，算法中设置了一个辅助队列用于存储正在访问的一层与上一层的节点便于访问下一层。

广度优先搜索的步骤如下:

1. 数据初始化。节点访问标志初始化为false。访问当前顶点v并加入辅助队列。
2. 若无尚未访问节点，退出。若有，继续执行。
3. 从队列中退出节点，置为当前节点。依次访问当前节点的所有邻接节点：若未被访问过，则置已访问标记为true，加入辅助队列。
4. 跳转到第二步循环执行上述过程。

图3.1.2 广度优先算法示例

为了实现逐层遍历，该算法经常需要使用辅助队列和辅助数组，空间复杂度为。

若采用邻接矩阵存储n个结点，e条边的图，对于每个被访问过的节点，广度优先搜索算法要检测权值矩阵D中的n个元素，总的时间复杂度O()。

我们可以发现，两种算法其实各有利弊。广度优先搜索需要存储图中所有节点，因此存储空间较大。但在搜索过程中不存在回溯过程，所以可以找到最短的路径。深度优先搜索由于可以回溯，所以找到可行路径的速度最快。

## 3.2 算法

### 3.2.1原算法

算法由荷兰科学家于1956年提出，目前仍是应用最为广泛的最短路径算法之一。算法正确得到结果的前提要求是图中的边权值都为正数。

该算法的基本思想是每次找到离原点（如1号节点）最近的一个顶点，然后以该顶点为中心进行扩展，最终得到源点到其余所有点的最短路径。

基本步骤：

1.设置标记数组book[]：将所有的顶点分为两部分，已知最短路径的顶点集合P和未知最短路径的顶点集合Q，很显然最开始集合P只有源点一个顶点。book[i]为1表示在集合P中；

2，设置最短路径数组dst[]并不断更新：初始状态下，dst[i]=edge[s][i](s为源点，edge为邻接矩阵),很显然此时dst[s]=0,book[s]=1.此时，在集合Q中可选择一个离源点s最近的顶点u加入到P中。并依据以u为新的中心点，对每一条边进行松弛操作（松弛是指由顶点s-->j的途中可以经过点u，并令dst[j]=min(dst[j],dst[u]+edge[u][j])）,并令book[u]=1;

3，在集合Q中再次选择一个离源点s最近的顶点v加入到P中。并依据v为新的中心点，对每一条边进弛操作（即dst[j]=min(dst[j],dst[v]+edge[v][j])）,并令book[v]=1;

4,重复3，直至集合Q为空.

|  |
| --- |
|  |

松弛操作伪代码

|  |
| --- |
| S;  dist[j]D[0][j]; j=1,2,...,n-1  do  {  dist[k]min{dist[i]},iV-S  SS;  dist[i]min{dist[i],dist[k] +D(k,i)} for every iV-S;  }while(S!=V) |

算法伪代码

最短路径确定之后，根据path[]数组回溯读取到各顶点之间的最短路径。比如0顶点到顶点4的最短路径形如：path[4]=xpath[x]=ypath[y]=0，即路径<0,y,x,4>。

分析算法可以得到该算法的空间与时间复杂度：对于一个有n个结点，m条边的加权图，算法的时间复杂度。

### 3.2.2 改进算法

改进算法采用了二叉堆的数据结构。二叉堆结构可以被视为一棵完全二叉树，而且其含义表明，完全 二叉树中所有非终端结点的值均不大于（ 或不小于） 其左、右子结点的值。除了用于堆排序之外，二叉堆最常见的应用是作为高效的优先级队列。该优先级队列是一种用来维护由一组元素构成集合S的数据结构，而且这一组元素中的每一个都有一个关键字key。

一般作用于优先级队列上的二叉堆的相应操作有：

Heapify（ S ） ：即首先将集合 S 调整成二叉堆，并 设定其根结点具有最小关键字；然后该操作从堆的 根结点开始，通过对当前结点的左右子树关键字的 比较，来调整相应结点在堆中的正确位置，即通常所 谓的“筛选”过程，而且此操作为维持堆性质的关键．

Heap-Insert（ x，S ） ：S∪｛x ｝←S，即将元素 x 插 入集合 S，并调用 Heapify 将其调整成二叉堆．该操 作是首先将堆加以扩展，即在树的最后一层加一片 叶子，然后遍历由新加的结点叶子到根的路径，以找 到放新元素的合适位置．

Heap-Extract-Min（ S ） ：即抽取具有最小关键字 的元素，并调用 Heapify 将其调整成二叉堆．该操作 可通过对堆的 Heapify 操作来实现．其运行时间主 要花费在调整成二叉堆的操作上．

1.初始化操作：首先搜索与源结点StartPoint 关联的结点 Adj［StartPoint］，然后初始化结点列表 NodeList 中所有结点的权值cost［Nodei］，调用 Heap-Insert方法实现初始化Initialize操作,来初始化优先级队列Heap，同时创建堆Heap中结点与结点列表 NodeList中结点相互关联的索引Index.

2.抽取最短距离结点操作：即对优先级队列 Heap 的操作，通过调用 Heap-Extract-Min，选择结 点 Node［j］，使得 cost ［j］＝min｛cost［I］∈Node［i］ ∈Nodelist}。其中，Node［j］为当前求得的从 StartPoint 出发的最短路径终点．

3.松弛操作：对从Node［j］出发的结点Node［k］进行松弛操作，松弛操作是通过Decrease- Key方法来实现，即若 cost ［j］＋cost［j ,k］＜cost［k］，则修改cost［k］＝cost［j]＋cost［j ,k］；同时，将结点 Node［k］通过 Heap-Insert 加到优先级队列 Heap 中，并相应更新索引表 Index．而索引表中记 录的是结点列表 NodeList 与二叉堆中结点之间的 相对位置索引．

4.重复步骤2、3，直至Node［j]＝EndPoint．

5.结束.

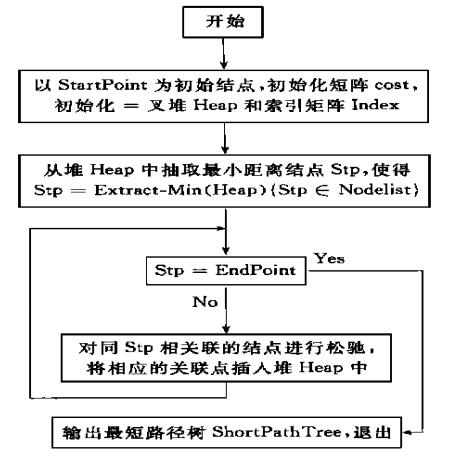


图3.2.2改进算法流程图

## 3.3 Bellman-Ford 算法

；

；

；

；

,），），

|  |
| --- |
|  |

图3.3.1 Bellman-Ford算法伪代码

## 3.4 Dijkstra算法和Bellman-Ford算法的比较

首先看Bellman－Ford算法在与顶点n 有关的计算中要知道n到其所有相邻顶点的权值以及从特定的源顶点S到这些顶点的路径的权值。每个顶点要有到网络中所有其它顶点的路径及其权值并且时时与其直接连接的相邻顶点交换这些信息。每个顶点都要用 Bellman－Ford算法来更新其权值以及到这些相邻顶点的权值。

另一方面考虑 Dijkstra算法。该算法需要每个顶点知道完整的网络拓扑信息。就是说每个顶点必须知道网络中所有链路的链路权值。 Dijkstra算法要求先构造最短通路树然后根据最短通路树形成路由表。构造最短通路树的过 程要用整个网络的拓扑信息即网络中所有链路 的费用。如果分布地实现路由算法Bell man－ Ford方法更合适。 Dijkstra 算法的优点是简单适合于在一个负载稳定拓扑变化不大的网络中运行。但是它的灵活性较差，无法对网络和拥塞和故障作出反应。Bellman－Ford 算法中每个节点的路由选择要依靠网络的当前状态信息来决定以设法适应网络流量、拓扑的变化。由于此算法中节点仅仅记录较好路径的变化所以若一条链路崩溃或者过载节点并没有办法了解到这些变化。因此此算法要频繁刷新每个节点的路由表以便适应不断变化的网络。

## 3.5 floyd算法

Floyd 算法是一种动态规划算法.对于G中包含的边eij，从任意节点 vi 到任意节点 vj 的最短路径 d[i，j]只存在两种可能：(1)直接从 vi 到 vj，此时 dij=eij；(2)从 vi 经过若干个节点到 vj.对于每个节点 vk，检查 dij>dik+dkj 是否成立，若成立，则令 dij=dik+dkj[6].由此得到状态转移方程:dij=min{dij,dik+dkj}.

Floyd 算法的具体步骤如下：

(1)从图 G 任意一个节点 vk 开始，遍历图中所有边 eij，检查是否满足 dik+dkj<dij，如果成立，说明从节点 i 到节点 k 再到节点 j 的路径要短于直接从节点 i 到节点 j 的路径，故进行松弛操作，更新 dij=dik+dkj；

(2)重复步骤(1)，直至遍历完图中所有节点，dij 中记录的便是节点 i 到节点 j 的最短路径距离.

|  |
| --- |
| 1 let dist be a |V| × |V| array of minimum distances iniƟalized to ∞ (infinity)  V是顶点的集合， |V|表示顶点的个数，首先我们初始化最小距离矩阵dist，其中每一个元素都  是Inf.  2 for each vertex v  3 dist[v][v] ← 0  将dist矩阵的主对角线元素变为0.(相同的点的最短距离当然是0喽)  4 for each edge (u,v)  5 dist[u][v] ← w(u,v) // the weight of the edge (u,v)  如果u,v两个顶点之间有权重，则用权重更新最短距离矩阵.  （事实上，1‐5步就是在生成一个权重邻接矩阵）  6 for k from 1 to |V|  中间节点k从1‐ |V| 循环  7 for i from 1 to |V|  起始节点i从1‐ |V| 循环  8 for j from 1 to |V|  终点节点j从1‐ |V| 循环  9 if dist[i][j] > dist[i][k] + dist[k][j]  如果i,j两个节点间的最短距离大于i和k的最短距离+k和j的最短距离  10 dist[i][j] ← dist[i][k] + dist[k][j]  那么我们就令这两个较短的距离之和取代i,j两点之间的最短距离  11 end if  结束if循环 |

图3.5 floyd算法伪代码

## 3.6各种路径算法的个人想法

Dijkstra算法实际上是基于贪心策略。贪心策略的意思是始终以利益最大化为目标来考虑问题。Dijkstra 算法中计算两点之间最短路径时一共需要两次循环，因此时间复杂度为 O(N2)。Dijkstra 算法适用于稠密图，并且只能适用于边权都为正的图，不能解决负权图的最短路径问题。

Floyd算法实际上是一种动态规划算法。它适用于解决所有顶点之间的最短路径问题。不仅如此,Floyd算法允许图中出现带负权值的边,但不允许有包含带负权值的边组成的回路。Floyd算法的时间复杂度为O(N2)。

Bellman-Ford 算法可以解决负权图问题，并可以检查图中是否含有负权回路，适用于稀疏图和边的关系较为密切图。Bellman-Ford 算法最多需要执行 n-1 次循环，每次循环需要遍历图中所有边，因此时间复杂度为 O((n-1)\*m)=O(nm)，由于采用边集数组存储边的信息，Bellman-Ford算法的空间复杂度为O(m)。

总的来说，每个算法都有不同的适用条件。所以，在不同语境下应该具体地分析并比较各个算法的优与劣，并得出最优的算法。实际上，这几种算法之间也不是互相独立的，而是有一定联系的。通过这几种算法，可以给其他算法的发明与研究提供一定的借鉴经验。条条大路通罗马，我希望以后可以研究出更加简便，时间复杂度更低的关于最短路径的巧妙解法。



图3.5 最短路径算法比较分析图

# 参考文献

1. 赵卫绩,巩占宇,王雯,樊守芳.几种经典的最短路径算法比较分析[J].赤峰学院学报(自然科学版),2018.
2. 唐文武,施晓东,朱大奎.GIS中使用改进的Dijkstra算法实现最短路径的计算[J].中国图象图形学报,2000.

[3] 王秀珍,苑世宁.两种最短路由Dijkstra算法和Bellman-Ford算法之比较[J].黑龙江农垦师专学报,2002.