**最短路径算法研究**

制作人：海春瑞

讲解人：王越洋

资料提供者：陈昶声，车逸超，

孙铭浩，谢睿杰，张佳乐

我们小组研究的问题是最短路径算法研究及实现，我们找到的方法有以下几种：

1. **Floyd算法**
2. **BeIlmanFord（贝尔曼-福特）算法与SPFA算法**
3. **A算法**
4. **Dijkstra（迪杰斯特拉）算法**
5. **Johnson算法**

**(一)Floyd算法**

算法简介：Floyd算法，也称为插点法，是一种用于解决所有最短路径问题的算法。该算法同样使用动态规划的思想，通过逐步插入中间节点，来逐步求解每个节点之间的最短路径。

时间复杂度：O(n^3)。

空间复杂度：O(n^2)。

***适用情况：***

1：***适用于有权图中任意两点之间最短路径的问题***，这种问题通常可以用一个矩阵来表示线路之间的距离或代价。所以，Floyd算法对于求解全源最短路径问题非常有效。

2：其时间复杂度为O(n^3)，***适用于数据规模较小的情况***。

1. 特别地，***当图的边权值为正整数时，Floyd算法比较适用，***因为它不会产生负元的问题，也不需要进行额外的处理。
2. ***如果边权值为负数，则Floyd算法不再适用，***因为在这种情况下就会出现产生负环的情况，因此需要使用其他算法，如Bellman-Ford算法或者Dijkstra算法等来解决最短路径问题。其时间复杂度为O(n^3)，适用于数据规模较小的情况。

***优点：***

1）简单易懂：Floyd算法的核心思想是动态规划，容易理解。

2）正确性高：Floyd算法能够得到最优解，保证了算法的正确性。

3）可以处理有向图和负权边：Floyd算法能够处理有向图和负权边的情况，具有较高的通用性。

***缺点：***

1. 时间复杂度较高：Floyd算法的时间复杂度为O(n^3)，在图较大时计算量比较大。
2. 2.空间复杂度较高：Floyd算法需要使用二维数组存储任意两点之间的最短路径长度，占用空间较大。
3. 对于稀疏图不适用：如果图比较稀疏，使用Floyd算法会造成不必要的计算，影响算法效率。

***Floyd算法的使用选择：***

因此，在实际应用中需要根据具体情况来选择是否使用Floyd算法。如果图较大或者需要处理有向图、负权边等复杂情况，Floyd算法是一种很好的选择；如果图比较稀疏或者对时间复杂度和空间复杂度有较高要求，可以考虑使用其他最短路径算法来解决问题。

***下面是简单的C语言Floyd算法代码：***

#define N 100

#define INF 1000000

int dist[N][N]; //邻接矩阵存储图

int min(int a, int b) {

return a < b ? a : b;

}

void floyd(int n) {

for(int k=0; k<n; k++) {//遍历中间点

for(int i=0; i<n; i++) {//遍历出发点

for(int j=0; j<n; j++) {//遍历到达点

dist[i][j] = min(dist[i][j], dist[i][k] + dist[k][j]);//更新最短路径

}

}

}

}

Floyd算法的主要思想是动态规划。假设我们要求i到j的最短路径，可以枚举中间点k，假设通过k点可以使得i到j的路径更短，那么就更新i到j的最短路径。

具体实现上，使用邻接矩阵来存储图，dist[i][j]表示i到j的最短路径。在遍历中间点、出发点、到达点的过程中不断更新最短路径即可。

需要注意的是，在初始化时，如果有两点之间没有边相连，则可以将它们的最短路径初始化为一个较大的值（这里使用了INF来表示）。初始化完毕后，就可以调用Floyd算法进行求解了。

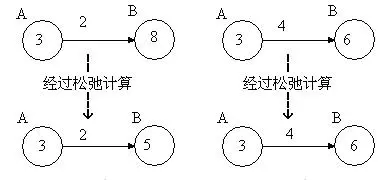
**(二) BeIlmanFord（贝尔曼-福特）算法与SPFA（Shortest Path Faster Algorithm）算法**

算法简介：Bellman-Ford算法的基本思想是将所有节点看作一个整体，进行n-1轮松弛操作，其中n为节点数。在每一轮操作中，对每一条边进行松弛操作，即检查是否可以通过当前路径可以获得更优的路径。如果在第n-1轮操作后仍然存在距离可以不断缩小的节点，那么说明存在负环。负环是指经过一条环路，可以一直减少路径长度，导致算法无法求出最短路径。

SPFA（Shortest Path Faster Algorithm）算法在Bellman-Ford算法的基础上进行了优化。它维护了一个队列，每次只对节点的邻接节点进行松弛操作，而不是像Bellman-Ford那样对所有边进行松弛操作。同时，如果某个节点的距离值在之前的操作中已经被修改过，那么把它加入队列就可以了，不需要每次都重新加入队列。这样可以大幅降低时间复杂度。

虽然SPFA算法在某些情况下有很好的表现，但是它也有一些问题。它可能出现负环的问题，而且在某些情况下其时间复杂度可能会退化到O(n^2)，甚至更高，因而在实际应用中需要特别注意。

***松弛操作***

不断更新最短路径和前驱结点的操作。

如上图所示，松弛计算之前，点B的值是8，但是点A的值加上边上的权重2，得到5，比点B的值（8）小，所以，点B的值减小为5。这个过程的意义是，找到了一条通向B点更短的路线，且该路线是先经过点A，然后通过权重为2的边，到达点B。

当然，如果出现右边这种情况，则不会修改点B的值，因为3＋4>6。

对边集合 E 中任意边，以 w(u,v) 表示顶点 u 出发到顶点 v 的边的权值，以 d[v] 表示当前从起点 s 到顶点 v 的路径权值

若存在边 w(u,v)，使得：d[v]> d[u]+w(u,v)，则更新 d[v] 值：d[v]=d[u]+w(u,v)。所以松弛函数的作用，就是判断是否经过某个顶点，或者说经过某条边，可以缩短起点到终点的路径权值。

为什么将缩短距离的操作称之为“松弛”，不妨理解为，选择某种方式后，到达目的的总代价降低了。什么名字无关紧要，不必纠结。

适用情况：

1：**BellmanFord算法适用于一般图，包括带有负权边的图**。

BellmanFord算法的基本思想是对所有边进行 V-1 次松弛操作，其中 V 表示图中的顶点数。如果在第 V-1 次松弛操作后仍然存在长度更短的路径，说明图中存在负权环，因为对于任意一个环，经过它的长度可以不断减小，直到负无穷。因此，BellmanFord算法可以应用于带有负权边的图，但由于需要 V-1 次松弛操作，因此时间复杂度为 O(VE)。

2：**SPFA算法适用于稠密图或者存在正权边的图。**

SPFA算法（Shortest Path Faster Algorithm）的基本思想是利用队列进行顶点松弛操作，即从源点扩散更新到其他顶点时，如果当前顶点的最短路径估计值发生改变，则将其入队，直到队列为空。SPFA算法的优点在于它对于一般的图可以处理的比较快，不需要像BellmanFord算法那样对所有边进行松弛操作，但是由于它使用的是队列，当图的分支比较多时，队列的增长速度过快，可能会降低算法效率。因此，SPFA算法适用于稠密图或者存在正权边的图。

**贝尔曼-福德算法优点：**

1.可以处理负权边的图，而Dijkstra算法却不行。

2.在有些情况下比Dijkstra算法更快，如边权值随机分布的图。

**贝尔曼-福德算法的缺点**：

1.最坏情况下的时间复杂度为O(VE)，其中V是顶点数，E是边数，比Dijkstra算法慢。

2.可能会产生负环，导致无法求解最短路径。

***SPFA算法的优点：***

1.在处理稠密图（边数接近于V^2）时比Dijkstra算法更快。

2.可以处理负权边的图。

3.可以在任意一种图上运行，不同于Dijkstra算法要求非负权边。

***SPFA算法的缺点：***

1.在处理稀疏图时，由于其内部的队列操作，速度比Dijkstra算法慢。

2.在最坏情况下，可能会产生负环，导致无法求解最短路径。

**(三)A算法**

算法简介：A\*算法是一种启发式搜索算法，用于在图中找到两个节点之间的最短路径。该算法使用一个评估函数，结合边的权重和从起点到当前节点的距离，来计算该节点到终点的估计距离，然后优先访问估计距离小的节点。这种优先级队列的排序方式可以保证优先考虑最优解路径，减少搜索次数，提高算法效率。

适用情况：

1.问题可以表示为图形或网格。

2.可以定义起点和终点。

3.可以定义节点之间的代价或距离。

4.可以定义一个估价函数，该函数可以评估从一个节点到终点的最小代价或距离。

优点：

1.最优性：A\*算法保证找到的路径是最短路径，即使在复杂的地图或网络中也能找到最优解。

2.完备性：如果存在路径，A\*算法能够找到一条路径。

3.启发式：A\*算法使用启发式函数来评估节点的价值，这样可以尽可能减少搜索的节点数，提高搜索效率。

4.可扩展性：A\*算法可以轻松地扩展到多种问题和应用中，例如机器人导航、游戏开发、路径规划等。

5.可定制性：A\*算法可以根据问题的特定需求进行修改和定制，例如更改估价函数或添加限制条件。

缺点：

1.启发式函数的设计需要一定的领域知识和经验，如果设计不好，可能会导致搜索效率低下。

2.如果启发式函数不是严格单调递增的，A\*算法可能会在搜索过程中反复遍历相同的节点，从而降低搜索效率。

3.对于具有高度分支因子的问题，A\*算法可能会导致搜索空间过大，导致计算时间和内存需求增加。

4.对于一些特殊情况，例如存在大量的障碍物或者没有可行解的情况，A\*算法可能会陷入死循环或无限搜索。

**(四)Dijkstra算法**

***算法简介：***

Dijkstra算法是一种用于解决带权重有向图或无向图的单源最短路径问题的贪心算法。该算法的基本思想是从源节点开始，依次对各个节点进行遍历，计算源节点到每个节点的最短路径，最终得到源节点到所有节点的最短路径。

***适用情况：***

1.Dijkstra算法适用于求解带权重的有向图或无向图中，单源最短路径问题，并且图中所有边的权重为非负数。

2.适用于求解单源最短路径问题，即给定一个起点，求它到图中所有其他节点的最短路径。

3.适用于稠密图和稀疏图，但是在稠密图中，时间复杂度会较高。

4.当边权值不同，而且权值关系比较复杂时，可以使用Dijkstra算法。

***优点：***

1.算法保证能够找到最短路径。对于带权重的有向图或无向图中，求解单源最短路径的问题，Dijkstra算法能够保证找到最短路径。

2.算法适用范围广。Dijkstra算法适用于带权重的有向图或无向图中，求解单源最短路径的问题，并且图中所有边的权重为非负数。

3.算法时间复杂度较低。通过使用堆优化，Dijkstra算法的时间复杂度可以降低到O(E + VlogV)，其中E是边数，V是节点数，使得Dijkstra算法可以高效地解决大规模的图论问题。

4.算法实现简单。Dijkstra算法的实现相对简单，容易理解和实现，因此被广泛应用于实际问题中。

***缺点：***

1.算法保证能够找到最短路径。对于带权重的有向图或无向图中，求解单源最短路径的问题，Dijkstra算法能够保证找到最短路径。

2.算法适用范围广。Dijkstra算法适用于带权重的有向图或无向图中，求解单源最短路径的问题，并且图中所有边的权重为非负数。

3.算法时间复杂度较低。通过使用堆优化，Dijkstra算法的时间复杂度可以降低到O(E + VlogV)，其中E是边数，V是节点数，使得Dijkstra算法可以高效地解决大规模的图论问题。

4.算法实现简单。Dijkstra算法的实现相对简单，容易理解和实现，因此被广泛应用于实际问题中。

**(五)Johnson算法**

***算法简介：***

Johnson算法则是一种用于解决带负权边的最短路径问题的算法。该算法首先对原图进行变换，使其不存在负权边，然后再利用Dijkstra算法求解每个源点到所有其他节点的最短路径。最后再根据变换前的结果得出最终结果。

***适用情况：***

1.带有负权边的稀疏图。在这种情况下，Dijkstra算法不能直接用于求解最短路径，因为它不能处理负权边。而Johnson算法通过将原图转换为一个不带负权边的图，再使用Dijkstra算法求解最短路径，从而解决了这个问题。

2.需要求解稀疏图中所有节点对之间的最短路径。在这种情况下，Floyd算法的时间复杂度为O(V^3)，而Johnson算法的时间复杂度为O(VE + V^2logV)，在稀疏图中效率更高。

***优点：***

1.解决了带有负权边的稀疏图的最短路径问题。Johnson算法通过将原图转换为一个不带负权边的图，再使用Dijkstra算法求解最短路径，从而解决了这个问题。

2.可以在稀疏图中高效地求解所有节点对之间的最短路径。在稀疏图中，Floyd算法的时间复杂度为O(V^3)，而Johnson算法的时间复杂度为O(VE + V^2logV)，在稀疏图中效率更高。

3.算法的实现相对简单，只需要对原图进行一次边权重的调整和一次Dijkstra算法的执行。

4.在实际应用中，由于负权边的存在，很多问题不能使用传统的最短路径算法求解，而Johnson算法提供了一种可行的解决方案。

***缺点：***

1.需要对原图进行一次边权重的调整，这个调整过程需要遍历整个图，因此在稠密图中，时间复杂度会比较高。

2.算法的实现相对复杂，需要使用堆等数据结构来实现Dijkstra算法，因此需要一定的编程技巧。

3.算法的空间复杂度比较高，需要额外的空间来存储转换后的图和每个节点的新权重值，因此在处理大规模图时，需要考虑内存的限制。

4.算法对负环的处理比较麻烦，需要使用贝尔曼-福德算法来判断是否存在负环，并且在存在负环的情况下，Johnson算法无法求解最短路径。