蛋白质网络最短路径的求解总结报告

第七组：杜昊雨、张子谦、刘岚、刘胤辅

项目背景

蛋白质互作网络（Protein-Protein Interaction Networks，PPI） 是由蛋白通过彼此之间的相互作用构成，来参与生物信号传递、基因表达调节、能量和物质代谢及细胞周期调控等生命过程的各个环节。系统分析大量蛋白在生物系统中的相互作用关系，对了解生物系统中蛋白质的工作原理，了解疾病等特殊生理状态下生物信号和能量物质代谢的反应机制，以及了解蛋白之间的功能联系都有重要意义。

在蛋白质互作网络中，两个蛋白质之间的最短距离可能代表着他们之间的生物学联系；因此通过该网络中所得到的信息可以对所得到的蛋白进行一些生物信息方向的分析。

项目的解决方案

通过对相关资料的查找，我们已经知道对于最短路径的求法存在多种方案。首先，我们可能通过交互输入的方式引入蛋白质相关数据，之后，我们尝试用矩阵构建一个蛋白质的网络。然后，同时通过多种算法求得网络中每一个蛋白质到其他蛋白质的最短路径。而我们使用到的算法主要有：

1. Floyd-Warshall Algorithm
2. Dijkstra algorithm
3. Bellman-Ford*algorithm

Dijkstra algorithm实现

主要的实现方式与细节可见代码，在这里只是简要的阐明该算法的实现思路（伪代码）。

首先构建3个矩阵：最短距离矩阵（array），邻居矩阵(neighbor)，是否完成矩阵(ifwork)。（动态分配内存）

最短距离矩阵是一个n*n（n是蛋白质个数）矩阵，在初始时由用户输入的数据进行构建，对于其中任意一个元素Mij，其在初始化时主要意义是说明蛋白质i和蛋白质j之间是否是邻居。如果两者不相邻，在初始化时用户会输入99.如果两者相邻，则会输入一个0~99的整数（即两相邻蛋白质之间的距离）。而经过计算之后，该矩阵中的所有元素Mij都将是蛋白质i和蛋白质j之间的最短距离。

依据最短距离矩阵在初始化时提供的信息，接下来构造邻居矩阵。其同样为n*n矩阵。其元素只能为0、1.如果矩阵元素Mij是1，说明蛋白质i和蛋白质j之间相邻。

而是否完成矩阵适用于检查某个蛋白质的所有计算是否完成，其为长度为n的一阶数组。如果元素Mi为1，说明蛋白质i到原点蛋白质之间的最短距离已经求出。由于计算都从第一个蛋白质开始，故M[0]=1，其余全为0。

之后，从第一个蛋白质开始进行计算。在第一次迭代中，我们搜索还没有完成计算的蛋白质，并且找出其中到原点距离最近的那个蛋白质。该蛋白质到原点的距离一定是最短路径，将该蛋白质ifwork项设置为1，跳转至该蛋白质。之后，对还没有完成计算的蛋白质我们进行路径开销的更新。即array[i][a]=min(array[i][a],array[source][a]+array[i][source]),source则指当前我们处于的元素。在更新之后，我们又能够找到当前到原点开销最短的那个蛋白质，于是再次发生跳转。再次更新。。。。。

经历了n次循环之后，就一定能够解决一个蛋白质到所有其他蛋白质的距离。又经过n次循环，就能够求解。

Bellman-Ford algorithm实现

规定每条边由起点、终点、权值组成。每张图包含节点总数，以及所有边的vector（图中每个点由0，1，...,节点总数-1 来表示）。

贝尔曼-福特算法的实现分三步。第一步是初始化图，代码中演示了控制台输入参数进行初始化或者代码中直接初始化的方式。（在本实现中我们规定了图的结构和其各项属性）第二步是松弛，持续对每一条边进行松弛操作。每次松弛操作实际上是对相邻节点的访问，第n次松弛操作保证了所有深度为n的路径最短。由于图的最短路径最长不会经过超过V-1条边，所以可知贝尔曼-福特算法所得为最短路径。第三步是检查负权环，因为负权环可以无限制的降低总花费，所以如果发现第次操作仍可降低花销，就一定存在负权环。

（虽然在蛋白质网络中一般不存在负边权）

Floyd-Warshall Algorithm实现

这个算法实现与Dijkstra 算法实现有很多相同之处。其最开始也是先根据输入得到最短距离矩阵，该矩阵初始化与Dijkstra 算法中初始化完全相同。但是，之后Floyd-Warshall算法直接采取遍历该数组的方式（通过一个三重循环），暴力求得最短路径。

该算法首先枚举顶点k，其大小在1~n之间；

随后以顶点k为中介点，枚举所有顶点对i和j（i、j也大小在1~n之间）

如果dis[i][k]+ dis[k][j] <dis[i][j成立；

即赋值dis[i][j]= dis[i][k] + dis[k][j]

数据导入与实验结果

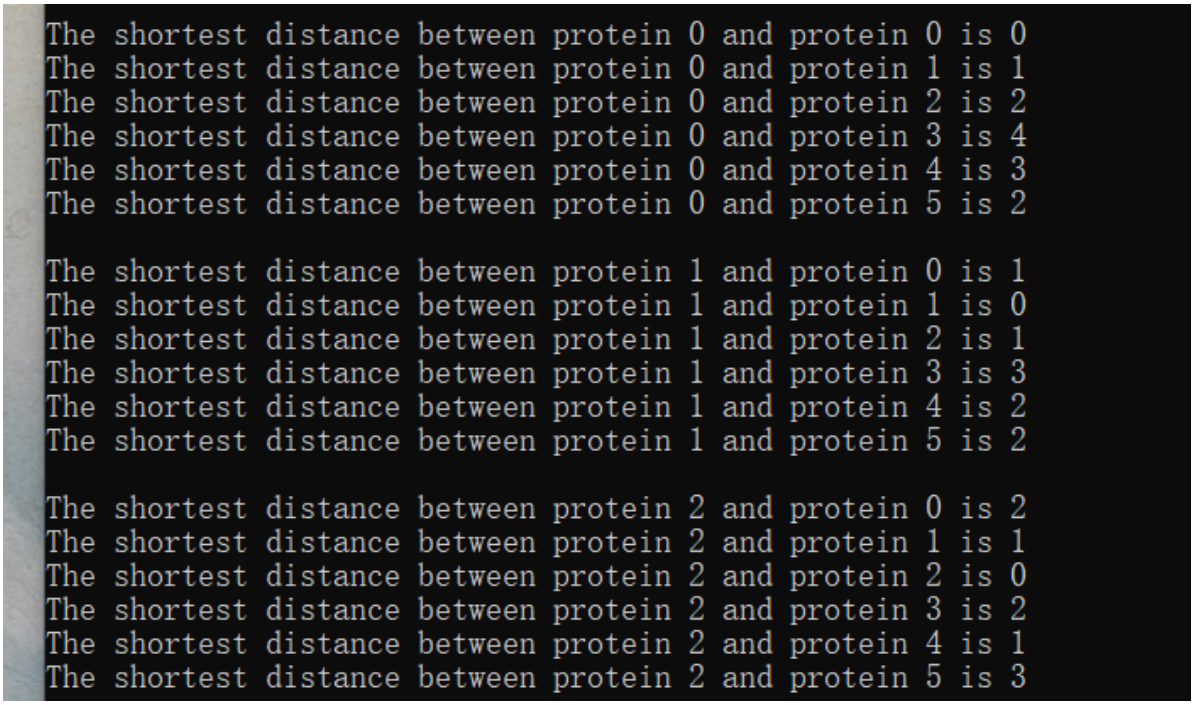
我们一共使用了3组数据（以最短距离矩阵形式给出，其中元素就是最初的可得信息）对我们的程序进行验证。其中第1、2组用于验证Dijkstra 算法和Floyd算法，因为两者输入相似性更高。

第一组数据如下：（数据来自教材：计算机网络）

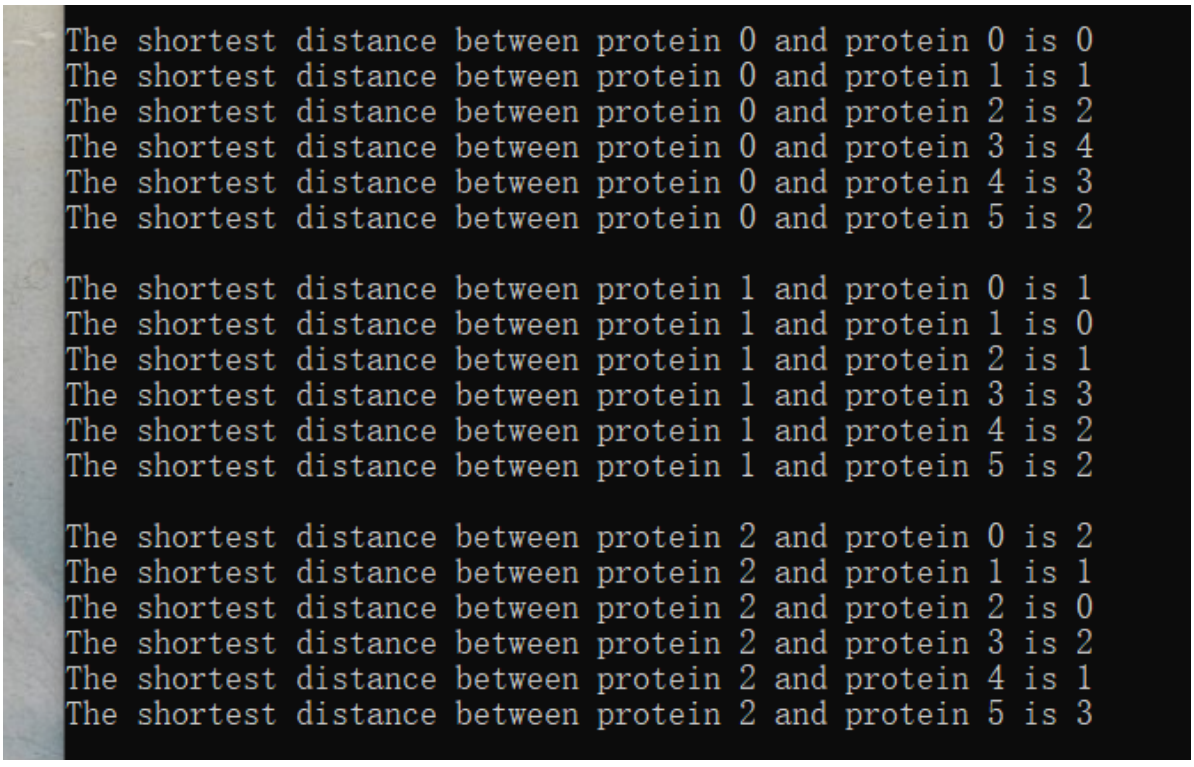
0 1 99 99 5 2
1 0 1 99 3 2
99 1 0 2 1 99
99 99 2 0 5 99
5 3 1 5 0 3
2 2 99 99 3 0

其结果如下：

Dijkstra 算法：



Floyd算法：



可知这两个算法都正确（已经经过验算）。

而对于Bellman-Ford algorithm，由于其内置算法是通过向量实现，故我们使用了单独的数据组。如下所示：

边序号 起点 终点 长度

0 0 2 1

1 0 1 3

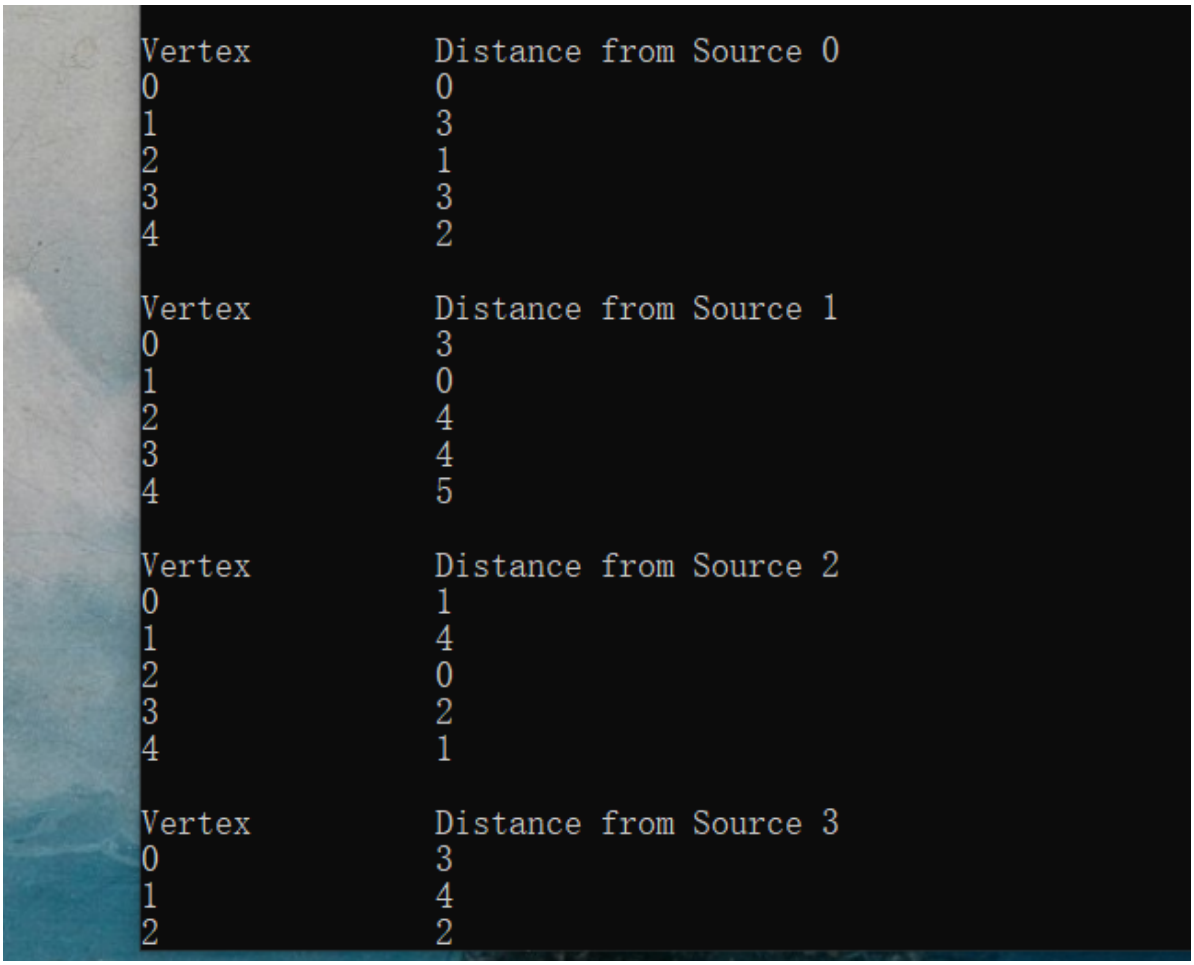
2 1 3 4

3 2 3 2

4 2 4 1

5 3 4 3

（部分）结果如下图所示：



经过验算发现该结果正确。

结论

从程序的运行情况我们可以看到，其实三种算法的正确性是毋庸置疑的。因此，接下来主要从理论上对算法进行分析。

首先，是对其时间、空间复杂度的分析。首先可以知道时间复杂度较高的算法为Floyd-Warshall Algorithm和 Bellman-Ford algorithm，因为前者使用了暴力的求解方式使用了三重循环，使得时间复杂度为O(n^3)；而后者的时间度则为O(V*E)，V为元素个数，E为边个数：在我们的数组实现中，其时间复杂度为O(n^2)，而其实可以达到O (E log V)。而对于空间复杂度方面，Floyd-Warshall Algorithm除了储存了最短距离矩阵之外，就几乎没有其他的内存消耗；而Bellman-Ford algorithm和Dijkstra algorithm则储存了大量辅助信息（距离向量、邻居矩阵、ifwork矩阵等）进行计算，其空间复杂明显更高。

而从应对发生错误、变化等方面进行分析，可以知道通过Floyd-Warshall Algorithm和Dijkstra algorithm实现的程序在遇到错误或变化时能够减少错误的影响力，因为在这两者进行时每个节点是独立、饱和的进行计算，存在修复错误的可能，同时错误不会扩散。

而理论分析可知Bellman-Ford algorithm中如果出现错误，错误将会扩散至所有节点；此外该算法还存在着无穷计数问题：该问题是指如果该网络达到平衡后又出现距离增加等问题时，Bellman-Ford algorithm必须花费多个循环迭代才能修正所有距离向量的问题，尤其是当变化突出时，花费的时间是难以计数的，这一问题进一步增加了Bellman-Ford algorithm的时间复杂度，也降低了其在遭遇网络改变

时的应对力。

最后，我们还从实现的难易度对三种算法进行分析。从原理上和实现上分析，可以知道Floyd-Warshall Algorithm的实现是最简单易懂的，之后是Dijkstra algorithm，而Bellman-Ford algorithm由于存在着节点间的交互等使得实现较为困难。

改进方向

首先是关于核心算法的实现方面。

在实现Dijkstra algorithm时，我们采用了更简单实现的数组实现方法，然而这一决定使得我们的实现时间复杂度未能达到最优；其实我们可以通过加入队列进一步提高该算法的性能。

其次是关于输入、输出方面。

在输入时对于部分算法，我们提供了两种输入方案，分别是手动输入和文件解析。但是手动输入十分繁琐，而文件解析的实现则是较为粗糙的，其需要固定格式的文件，而该文件的自动产生实际上我们尚未实现。故在输入方面其实可以有提高之处。

而在输出时我们是输出所有计算出的结果（即所有蛋白质互相的最短距离），当元素较多时显得很杂乱；经过我们的思考，我们发现其实可以通过由用户指定输出元素的方法，仅仅输出用户指定两元素之间的最短距离。

最后是在算法的重复性方面。我们程序中存在着一些很相似但是又有些许不同的代码，由于封装的实现存在困难，我们没能够将其封装成函数进行处理，使得代码重复度较高，这也是可以改进的地方