上海交通大學

SHANGHAI JIAO TONG UNIVERSITY

《生物计算编程语言》项目报告

——蛋白质相互作用网络中的所有对最短路径问题

姓　　名：袁依帆,程子骏,胡思贤

学　　院：生物医学工程学院

指导老师：吴茂英

日　　期：2023.6.11

《生物计算编程语言》项目报告

——蛋白质相互作用网络中的所有对最短路径问题

小组成员：袁依帆，程子骏，胡思贤

【摘 要】本文研究了在蛋白质相互作用网络中解决全对最短路径问题的算法，并对其进行了实现和测试。蛋白质相互作用网络是生物信息学中重要的研究领域，对于理解蛋白质功能关系和通信途径具有重要意义。本文采用了三种经典的最短路径算法，包括 Floyd-Warshall、Dijkstra 和 SPFA，并比较了它们的性能和适用性。实验结果表明，Floyd-Warshall 算法适用于中小型网络，能够计算所有节点之间的最短路径。Dijkstra 算法适用于单源最短路径问题，能够快速计算从源节点到其他节点的最短路径。然而，Dijkstra 算法无法处理存在负权边和负权环的情况。相比之下，SPFA 算法是对 Bellman-Ford 算法的一种优化，能够处理带有负权边和负权环的网络，具有较好的准确性和适用性。文章还探讨了未来改进方向，包括真实数据应用、并行和分布式计算、数据整合和新算法优化等。这些算法的发展将为蛋白质相互作用网络分析提供更多潜力和机会，推动蛋白质研究、生物工程等相关领域的应用和发展。

【关键词】蛋白质相互作用网络（PPI）；全对最短路径；Floyd-Warshall 算法；Dijkstra 算法；SPFA 算法

1 背景及需求分析

1.1 背景

在生物学领域，蛋白质相互作用网络（PPI）是一种揭示生物分子之间复杂关系的重要工具。蛋白质是细胞生命活动的主要执行者，它们之间的相互作用形成网络，构成了生命的基本框架。在这个框架中，蛋白质之间的最短路径可以被理解为一种可能的通讯途径或功能关系，因为这种路径可能指示了蛋白质通过最少的相互作用步骤如何相互联系。此外，这些最短路径也可能揭示未知的生物过程或疾病机制。

研究蛋白质相互作用网络对于理解细胞过程、生物功能和疾病机制至关重要。在这些网络中，蛋白质之间的最短路径可以提供有关它们功能关系和通信途径的重要见解。这促使我们需要开发算法，以有效计算蛋白质相互作用网络中的全对最短路径。

图论是数学的一个分支，主要研究图的性质。在此背景下，图是由节点（在本案例中是蛋白质）和边（在本案例中是蛋白质之间的相互作用）组成的。这使得图论成为处理蛋白质相互作用网络问题的理想工具。

1.2 需求分析

在这个背景下，我们的需求是在蛋白质相互作用网络中寻找所有蛋白质对之间的最短路径。这需要我们借助于图论中的算法。在多种选择中，包括单源最短路径（例如 Dijkstra 算法）、多源最短路径（例如 Floyd-Warshall 算法）以及其他针对特定类型的图优化的算法（例如 SPFA 算法，这是 Bellman-Ford 算法的改进版）。

在选择算法时，我们需要考虑到 PPI 网络的特性，它们通常是无向的、非加权的，且含有大量的节点和边。因此，我们需要选择一个能够高效处理大规模

图，同时考虑到网络的特性的算法。

最后，我们还需要考虑如何以有意义的方式展示和解释结果，这将需要我们结合生物学的知识和理解，可能需要进一步的分析或可视化工具。

总的来说，我们的需求是找到一个适合处理蛋白质相互作用网络的最短路径问题的有效算法，并以一种可解释和有用的方式展示结果。

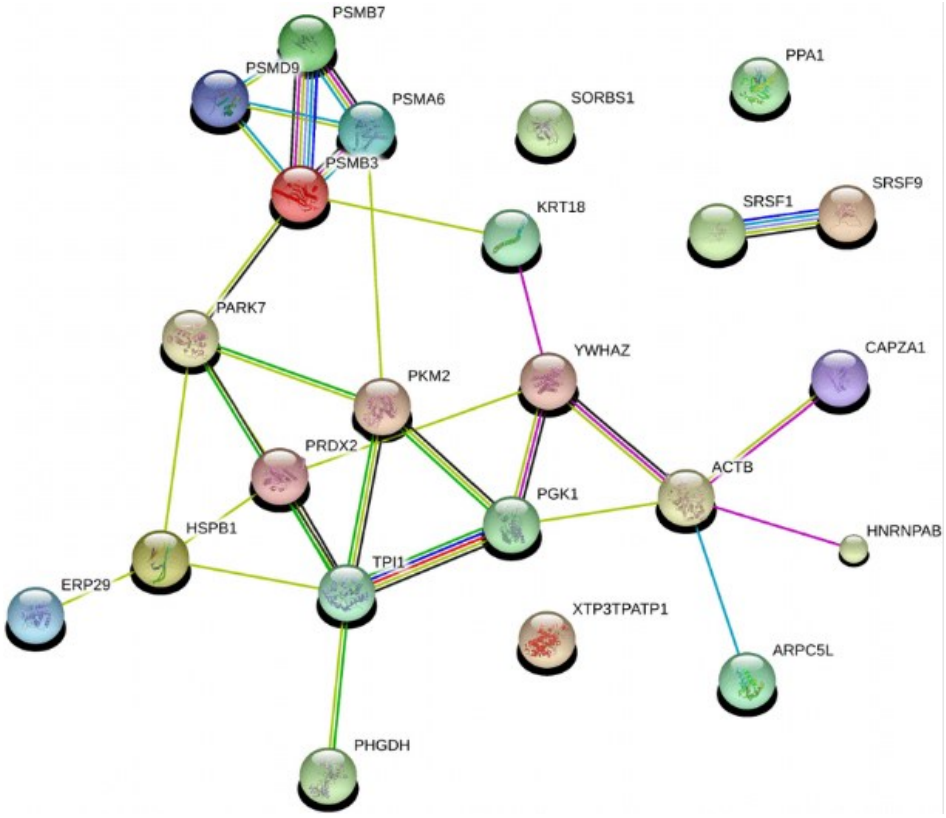

图 1. 蛋白质相互作用网络图（PPI）

2 算法介绍

为了解决蛋白质相互作用网络中的全对最短路径问题，我们可以利用各种算法。三种常用的算法是 Floyd-Warshall 算法、Dijkstra 算法和 SPFA 算法。

a) Floyd-Warshall 算法

Floyd-Warshall 算法是一种计算图中所有顶点对之间最短路径的动态规划算法。该算法假设图中没有负权环，但允许负权边。

Floyd-Warshall 算法的基本思想是逐渐改善路径的权值，使其成为最短路径。算法开始时，路径的权值为直接连接的边的权值。然后，对于每个顶点 k，算法尝试使用顶点 k 作为中间点，看看是否可以通过它改进所有顶点对 (i,j) 的路径。如果通过 k，从 i 到 j 的路径可以变得更短，那么就更新路径的权值。这个过程一直持续到考虑了所有的顶点，那么此时的权值就是最短路径的权值。

Floyd-Warshall 算法主要用于那些需要找出所有节点对之间最短路径的场景，尤其是在节点数量相对较少或者计算资源不是问题的情况下。例如，网络路由问题，社交网络分析，城市交通网络的设计等。

在蛋白质相互作用网络中，Floyd-Warshall 算法可以用来计算所有蛋白质对之间的最短距离，为进一步的生物信息学分析提供基础数据。

b) Dijkstra 算法

Dijkstra 算法是一种寻找从单个源节点到图中所有其他节点的最短路径的贪婪算法。该算法假设所有的边权重都是非负的。

算法首先将源节点的最短路径长度设为 0，将所有其他节点的最短路径长度设为无穷大。然后，算法从源节点开始，每次选择一个最短路径长度最小的节点，更新这个节点的所有邻居的最短路径长度。算法继续选择下一个最短路径长度最小的节点，更新其邻居，如此反复，直到所有的节点都被选择。

Dijkstra 算法主要用于那些只需要找出单个源节点到所有其他节点的最短路径的场景，以及在所有边的权重都为正的场景下。例如，地图上的最短路径问题，如 GPS 导航系统，网络路由问题等。

在蛋白质相互作用网络中，如果我们只关心某个特定的蛋白质与其他蛋白质的关系，比如一个疾病相关的蛋白质，那么可以使用 Dijkstra 算法来找出这个蛋白质到所有其他蛋白质的最短路径。

c) SPFA（Shortest Path Faster Algorithm）算法

SPFA 算法是一种基于队列的最短路径查找算法，它是 Bellman-Ford 算法的优化版本。此算法假设边的权重可以是负数，但不允许存在负权重的环。整个算法包括了以下三部分：

① **初始化：**算法首先设定源节点的最短路径长度为 0，其他所有节点的最短路径长度设为无穷大。并将源节点加入到队列中。

② **循环处理：**只要队列非空，就从队列中取出第一个节点，然后遍历所有从该节点出发的边。如果通过当前节点到达邻居节点的路径长度小于已知的邻居节点的最短路径长度，那么更新邻居节点的最短路径长度，并将邻居节点加入到队列中。这个过程反复进行，直到队列为空。

③ **检查负权环：**如果在队列为空后，仍有节点的最短路径长度发生改变，那么图中可能存在负权环。

与 Bellman-Ford 算法相比，SPFA 算法不需要在每次迭代中检查所有的边，而是仅对那些在最近一次迭代中更新过最短路径长度的节点的邻居节点进行处

理。这使得 SPFA 算法在处理稀疏图时，尤其是无负权环的稀疏图时，运行效率更高。

SPFA 算法可以广泛应用于那些可能包含负权重边，但不包含负权环的图中，尤其是当图相对稀疏时。例如，路由寻优问题，交通网络优化等。

在蛋白质相互作用网络中，如果权重可能存在负值（例如，表示某种“阻抗”或“消耗”的情况），并且网络结构相对稀疏，那么使用 SPFA 算法来寻找最短路径可能是一个很好的选择。

解决蛋白质相互作用网络中的全对最短路径问题涉及到多种算法的应用，选择哪种算法主要取决于网络的特性和问题的具体需求。在蛋白质相互作用网络中，可以根据网络的密集程度、边的权重性质（加权或非加权以及是否存在负权边）以及研究目标来选择最合适的算法，从而有效地解决全对最短路径问题。

3 算法实现与测试

我们采用 python 对以上三个算法分别进行了实现与测试。

3.1 Floyd-Warshall 算法实现过程

Floyd-Warshall 算法是一种动态规划算法，主要用于解决所有顶点对之间的最短路径问题。这个算法在每一步中都会尝试使用新的中间顶点来更新已知的最短路径。其时间复杂度为 O(n^3)，且无法处理负环问题。我们在 Python 中实现了该算法。代码主要包括以下几个部分：

- 初始化阶段（__init__函数）：创建了一个邻接矩阵 self._side 来存储图的信息，并设置了一个比较函数 cmp 来比较两条路径的长度。
- 添加边阶段（push_oneway_side 和 push_side 函数）：使用这两个函数向图中添加单向边或双向边。每条边由两个顶点和一个权重组成。如果新添加的边的权重比现有的边的权重小，那么就用新的边来更新邻接矩阵。
- 计算最短路径阶段（calculate 函数）：在这个阶段，Floyd-Warshall 算法的核心思想被实现出来。算法会尝试使用每个顶点作为中间顶点，看看是否可以通过它来更新已知的最短路径。这个过程是通过三层嵌套循环实现的，因此这个算法的时间复杂度是 O(n^3)。注意这里的松弛操作，如果通过中间点 k 从 i 到 j 的路径长度更短，就更新答案矩阵 self._ans[i][j]。
- 获取最短路径长度阶段（get_ans 函数）：在完成所有的计算后，可以使用这个函数来获取任意两个顶点之间的最短路径长度。

- 清空阶段（clear 函数）：这个函数用来清空邻接矩阵和答案矩阵，准备进行新一轮的计算。

3.2 Dijkstra 算法

Dijkstra 算法是一种贪心算法，主要用于解决单源最短路径问题。这个算法会首先从源顶点开始，逐步选择距离源顶点最近的顶点，更新其它顶点的最短路径信息。其时间复杂度为 O(m*logn)，且无法处理负环问题。我们在 Python 中实现了该算法。总体流程与 Floyd-Warshall 算法类似。因此，我们将着重对代码的核心部分“计算最短路径阶段”进行阐述，其主要思想是反复利用图中的边信息来更新路径长度，直到找到所有节点的最短路径，具体流程如下

① **初始化：**这个函数首先初始化了结果列表（self._ans），用于存放从起始点到每个节点的最短路径长度，初始化为'None'。并且设置了一个标记列表（renewed），用于记录每个节点是否已经更新过。接着，设置了一个堆 hp，用于存放待处理的节点。堆中每个节点包含了节点编号和从源节点到该节点的当前最短路径长度。

② **起点处理：**函数将起点加入到结果列表，并将其放入到堆中。

③ **主循环：**主循环在堆变空之前持续执行。在每一轮中，算法首先从堆中取出当前最短路径最短的节点（即堆顶元素）。如果这个节点已经被更新过（即已经找到了从源节点到这个节点的最短路径），那么就跳过这个节点，处理下一个。否则，就遍历这个节点的所有邻居节点，如果通过这个节点可以得到一个更短的路径，就更新这个邻居节点的最短路径信息，并将这个邻居节点放入堆中。需要注意的是，这个过程可能会导致堆中出现同一个节点的多个副本，但是每个副本都有不同的路径长度。这并不会影响结果，因为最终我们只关心每个节点的最短路径。

此外，Dijkstra 算法还用到了 heap 最大最小堆数据结构，这是一个基于比较函数 cmp 的堆数据结构这个堆是基于完全二叉树的，采用列表来存储数据，堆的操作主要有插入元素，弹出堆顶元素，获取堆顶元素，检查堆的大小和是否为空。

这是一个通用的堆实现，可以通过传入不同的比较函数实现最大堆和最小堆。这是因为堆的性质（父节点大于或小于子节点）依赖于这个比较函数。例如，如果比较函数是小于操作符，那么堆就是最小堆，堆顶元素就是最小的元素；反之，如果比较函数是大于操作符，那么堆就是最大堆，堆顶元素就是最大的元素。

我们的算法实现的主要优势是每次操作的时间复杂度为 O(logn)，其中 n 是堆中的元素数量。这使得它在实现如 Dijkstra 算法等需要频繁插入和删除最大或最小元素的场景中非常有效。

3.3 SPFA 算法实现过程

在初始化阶段我们首先创建一个 SPFA 类的实例。在 SPFA 类的构造函数中，我们创建了两个邻接表 _side 和 _value，分别用来存储图中的边和对应的权重。这两个表都是列表的列表，其中每个子列表代表一个节点的所有邻接节点（_side）以及到这些节点的边的权重（_value）。我们还初始化了一个结果列表 _ans，用来存储从源节点到每个其他节点的最短路径长度，以及一个布尔型列表 in_queue，用来记录每个节点是否已经在队列中。这两个列表在算法开始时被全部填充为 'None' 和 False。此外，我们还定义了一个计数器列表 _cnt 和一个负环标志 _Negative_circle，分别用来记录每个节点的更新次数和检测是否存在负环。

在添加边的步骤中，我们调用 push_side 函数来向图中添加边。该函数会调用 push_oneway_side 函数两次，一次为正向，一次为反向，从而实现双向边的添加。

计算最短路径的步骤在 calculate 函数中进行。首先，我们创建了一个空的队列 dots_queue，并将源节点添加进队列。同时，我们将源节点的最短路径长度设置为 0，并将其 in_queue 值设置为 True。

然后，我们开始一个循环，直到队列变为空或者发现一个负环为止。在每次循环中，我们首先从队列中取出一个节点 u，并将其 in_queue 值设置为 False。然后，我们遍历 u 的所有邻接节点，尝试通过“松弛操作”更新这些节点的最短路径长度。具体地，对于每个邻接节点 v 和对应的边的权重 w，如果 u 到 v 的路径长度加上 w 小于 v 当前的最短路径长度，那么我们就用这个更小的值更新 v 的最短路径长度，并将 v 添加到队列中（如果它还不在队列中的话）。

如果在更新节点 v 的最短路径长度时，我们发现 v 的更新次数已经达到或超过节点总数，那么我们就可以确定存在一个负环，于是将 _Negative_circle 设置为 True 并立即退出循环。

最后，我们使用 get_ans 函数来获取从源节点到其他任意节点的最短路径长度。如果存在负环，那么我们返回字符串 'Negative circle exists'；否则，我们返回对应的最短路径长度。

我们可以将整个过程整理如下：

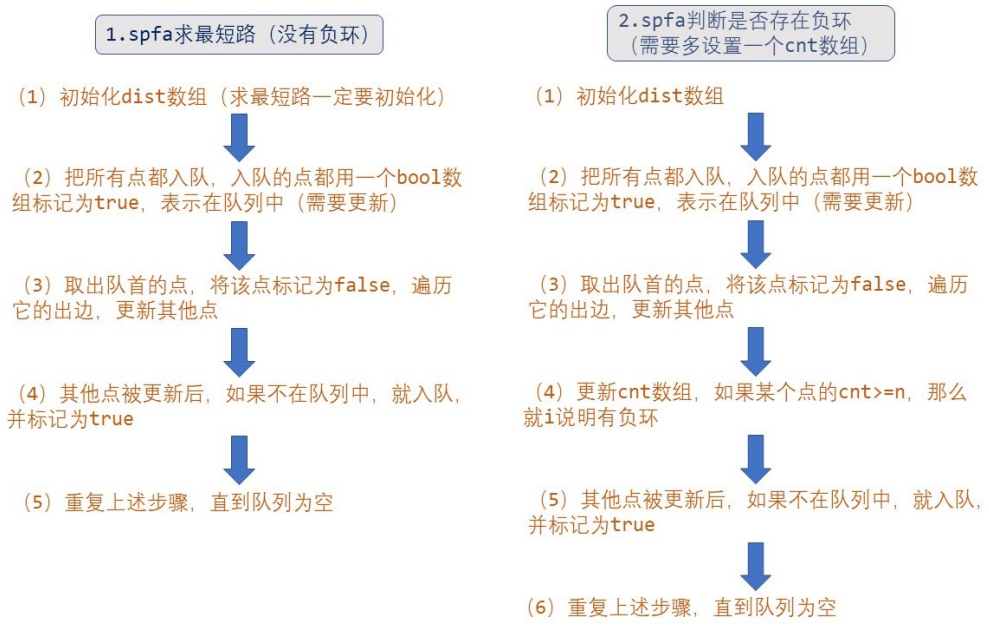

图 2. SPFA 算法流程图

3.4 算法测试

为了评估算法的性能，我们在模拟的蛋白质相互作用数据集（具有 4 个节点的无向图）上进行了实验。

3.4.1 Floyd-Warshall 算法和 Dijkstra 算法测试：

首先，针对 Floyd-Warshall 算法和 Dijkstra 算法，我们采用类似的测试方法。我们构建一个具有 4 个节点的无向图。边的权重分布如下：

节点 1 和节点 2 之间的权重为 9；节点 1 和节点 3 之间的权重为 1；
节点 1 和节点 4 之间的权重为 6；节点 2 和节点 3 之间的权重为 2；
节点 2 和节点 4 之间的权重为 7；节点 3 和节点 4 之间的权重为 5。

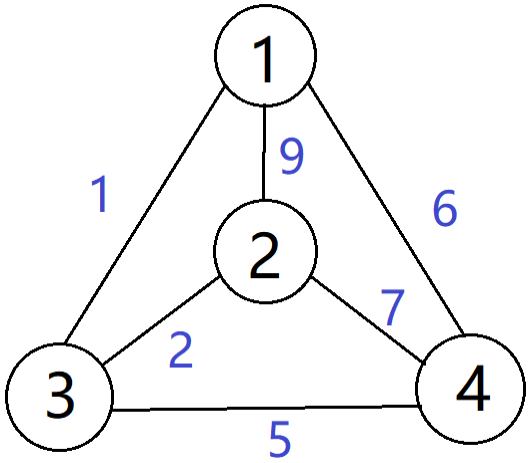

图 3. 4 节点无向图

然后，我们使用 push_side 方法将边和对应的权重添加到图中。注意，由于这是一个无向图，所以我们使用的是 push_side 方法而非 push_oneway_side 方法。接着，我们调用 calculate 方法计算从节点 1 到所有其他节点的最短路径。最后，我们使用 get_ans 方法获取从节点 1 到节点 2 的最短路径，并打印结果。实验结果显示，从节点 1 到节点 2 的最短路径长度为 3。

此实验采用模拟实验的方法演示了如何使用 Floyd-Warshall 算法和 Dijkstra 算法来解决最短路径问题。

3.4.2 Dijkstra 算法的负权边测试：

接下来，我们将测试 Dijkstra 对负环或负权边的能力。我们调整边的权重如下（节点 1 和节点 3 的权重调整为-1）：

*节点 1 和节点 2 之间的权重为 9；节点 1 和节点 3 之间的权重为**-1**；*
节点 1 和节点 4 之间的权重为 6；节点 2 和节点 3 之间的权重为 2；
节点 2 和节点 4 之间的权重为 7；节点 3 和节点 4 之间的权重为 5。

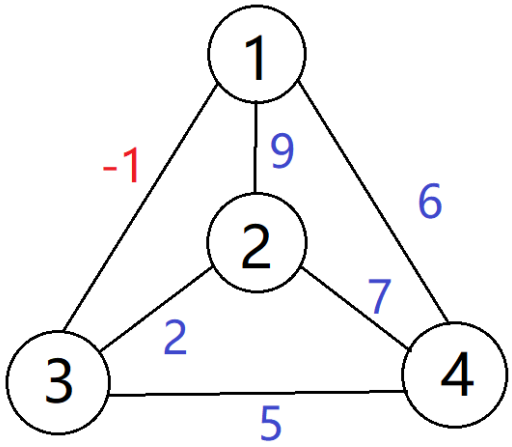

图 3.4 节点带负权环无向图

理论上：“ 1-3-1 ”一圈的和是-2，因此通过不断走负权 1-3-1 通道，值不断减小，实际答案是负无穷。而测试结果是 1，显然不符合。

因此我们可以得出结论，Dijkstra 算法无法正确处理负权边和负权环。因为 Dijkstra 算法的核心假设是，从起始点到任何节点的最短路径不会经过权值更大的边。也就是说，如果我们找到了一条从起始点到节点 A 的路径，且这条路径的权值比从起始点到任何节点 A 的邻接节点的路径的权值都要小，那么我们就认为找到了从起始点到节点 A 的最短路径。这条路径将不再被改变，节点 A 将被标记为已访问。然而，如果图中存在负权边或负权环，这个假设就不再成立。因为通过负权边或环，我们可能找到一条权值更小的路径，即使这条路径的部分边的权值更大。

3.4.3 SPFA 算法测试：

我们根据以上两种边的权重情况，对 SPFA 算法在正常情况和存在负环的情况下分别进行测试：

① **对于没有负权边或环的正常情况：**SPFA 算法能够成功计算从起始点到任意节点的最短路径。例如在第一个测试中，从点 1 到点 2 的最短路径权值为 3。

② **对于存在负权环的情况：**SPFA 算法能够成功检测出负权环的存在。例如在第二个测试中，由于存在一个权值之和为-2 的负权环 1-3-1，所以从点 1 到点 2 的最短路径实际上是负无穷。SPFA 算法通过检测每个节点的更新次数，当更新次数达到节点数量时，就能判断出存在负权环，并输出'Negative circle exists'。

4 项目总结

在蛋白质相互作用网络中，全对最短路径问题对于理解蛋白质之间的功能关系和通信途径具有重要作用。包括 Floyd-Warshall、Dijkstra 和 Bellman-Ford 在内的算法为计算这些网络中的最短路径提供了有效的解决方案。

通过我们的实验，我们观察到算法的选择取决于网络的具体特征以及在准确性和计算效率之间的权衡。Floyd-Warshall 算法适用于中小型网络，而 Dijkstra 算法适用于单源最短路径计算。Bellman-Ford 算法处理带有负权边的网络。

本项目的核心是探索最短路径问题，并实现和测试了 Dijkstra、Floyd 和 SPFA 等三种经典的最短路径算法。这些算法在理解和解决诸如蛋白质相互作用网络等生物信息学问题中有着重要的应用。

Floyd 算法：Floyd 算法能够解决所有节点间最短路径的问题。由于蛋白质相互作用网络通常涉及大量蛋白质，需要全面理解蛋白质之间的通信路径，Floyd 算法在此场景中非常适用。但是，由于其时间复杂度为 O(n^3)，在处理大规模蛋白质网络时可能会遇到效率问题。

Dijkstra 算法：Dijkstra 算法是一个单源最短路径算法，适用于求解有向图和无向图的单源最短路径问题。然而，该算法不能处理存在负权边或负权环的图。因此，Dijkstra 算法在蛋白质相互作用网络中，适用于确定一个特定蛋白质与网络中其他蛋白质的最短通信路径。

SPFA 算法：SPFA 算法是 Bellman-Ford 算法的一种优化，它可以处理存在负权边和负权环的图。这使得 SPFA 算法能够处理更复杂的蛋白质相互作用网络，例如，网络中可能存在反向激活或抑制的相互作用路径，这在其他算法中可能会导致问题。

在性能方面，Dijkstra 算法和 SPFA 算法通常在稀疏图上表现得更好，而 Floyd 算法在稠密图上表现得更好。对于蛋白质相互作用网络，具体选择哪种算法主要取决于网络的稠密程度、是否存在负权边，以及是否需要求解所有蛋白质间的最短路径等因素。

5 展望与改进

在这个项目中，我们已经成功地使用了几种算法，包括 Floyd-Warshall、Dijkstra 和 SPFA 来解决蛋白质相互作用网络中的全对最短路径问题。然而，这个领域还有很多改进和拓展的空间，未来的工作可能会集中在以下几个方向：

a) **真实数据应用：**在我们的实验中，所使用的都是模拟的蛋白质作用网络。在未来的工作中，我们希望使用真实的人类蛋白质相互作用数据进行研究，这将使我们的工作更加贴近实际，也能为理解真实生物系统中的蛋白质相互作用提供更多帮助。

b) **并行和分布式计算：**随着生物信息学数据规模的不断增长，提高算法的可扩展性成为了一个重要的课题。开发并行和分布式版本的算法，能够有效利用现代计算资源，极大提高处理大规模蛋白质相互作用网络的能力。

c) **数据整合：**蛋白质相互作用网络中的边并非仅仅由蛋白质间的物理接触决定，还包括蛋白质间的功能关系，这些信息往往来源于不同类型的生物学实验。整合这些额外的生物数据，将有助于我们更准确地计算最短路径，也能提供更多关于蛋白质相互作用的信息。

d) **新算法和优化：**尽管我们已经实现和测试了多种算法，但仍有许多其他的算法和优化方法未被尝试。例如，对于具有复杂拓扑结构或异质边权的网络，可能需要设计新的算法或改进现有的算法来更高效地处理。

总结来说，研究蛋白质相互作用网络中的全对最短路径问题能够提供有关蛋白质功能关系和通信途径的重要见解。选择的算法提供了有效的解决方案，进一步的研究可以推动其可扩展性、准确性和适用性在更复杂的网络情境中的发展。通过改进算法和整合更多的生物信息，我们能够深入了解蛋白质相互作用网络的复杂性，为疾病研究、药物设计和生物工程等领域提供更多的应用价值。

6 项目感想

在本项目中，我们研究并实现了几种不同的最短路径算法，并在蛋白质相互作用网络上进行了实验和比较。项目的过程让我们体验到了计算机科学理论和实

践的结合，以及它们如何应用到生物信息学这样的跨学科领域。

我们通过理解和实现 Dijkstra、Floyd 和 SPFA 等算法，深化了对图论和最短路径问题的理解。同时，我们也了解到在现实世界的问题中，理论和实践可能会有所差异。例如，算法的选择需要考虑图的具体特性，如是否存在负权边、是否需要求解所有节点间的最短路径等。而在处理大规模数据时，如蛋白质相互作用网络，还需要考虑算法的计算效率和可扩展性。

在实验过程中，我们遇到了一些挑战，如数据结构的选择、算法的优化等。但是，正是这些挑战推动我们思考并解决问题，不断地学习和进步。通过比较不同算法的结果，我们更深入地理解了每种算法的优点和局限，这对我们今后选择和使用算法非常有帮助。

最后，项目让我们看到了计算机科学在生物信息学等领域的巨大潜力。对于蛋白质相互作用网络这样的复杂系统，计算机算法能提供有效的解决方案，帮助我们理解蛋白质之间的关系和通信途径。这使我们对未来的研究充满期待，同时也激发了我们继续探索和学习的热情。

总的来说，这个项目是一次富有挑战和收获的经历，我们对此感到非常满意和自豪。

7 参考文献

[1] Stark C, Breitkreutz B-J, Reguly T, Boucher L, Breitkreutz A, Tyers M. BioGRID: a general repository for interaction datasets. *Nucleic acids research.* 2006;34(suppl 1):D535–D539.

[2] Wilson N. Human Protein Reference Database. *Nature Reviews Molecular Cell Biology.* 2004;5(1):4–4.

[3] Hougardy S. The Floyd–Warshall algorithm on graphs with negative cycles[J]. Information Processing Letters, 2010, 110(8-9): 279-281.

[4] Weisstein E W. Floyd-warshall algorithm[J]. https://mathworld. wolfram. com/, 2008.

[5] Shu-Xi W. The improved dijkstra's shortest path algorithm and its application[J]. Procedia Engineering, 2012, 29: 1186-1190.

[6] Crauser A, Mehlhorn K, Meyer U, et al. A parallelization of Dijkstra's shortest path algorithm[C]//Mathematical Foundations of Computer Science 1998: 23rd International Symposium, MFCS'98 Brno, Czech Republic, August 24–28, 1998 Proceedings 23. Springer Berlin Heidelberg, 1998: 722-731.

[7] 段凡丁. 关于最短路径的 SPFA 快速算法[J]. 西南交通大学学报, 1994, 29(2): 207-212.

[8] 夏正冬, 卜天明, 张居阳. SPFA 算法的分析及改进[J]. 计算机科学, 2014, 41(6): 180-184.