南京晓庄学院

《数据科学与大数据技术导论》

课程报告

班　级：25 数据科学与大数据技术（单招）

学　号：25131129

姓　名：周泓霖

课程报告：Dijkstra 算法与大数据应用

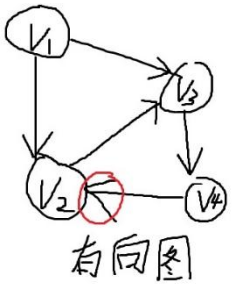
引言

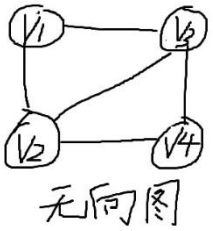
图论作为离散数学的重要分支，在计算机科学中具有广泛的应用，尤其在网络分析、路径规划、社交网络挖掘、交通调度等领域中发挥着关键作用。其中最短路算法，如 Dijkstra 算法、Bellman-Ford 算法和 Floyd-Warshall 算法，是图论中经典且基础的问题求解工具。其中，由艾兹格·迪杰斯特拉提出的 Dijkstra 算法，以其高效和稳定成为解决**非负权图单源最短路**问题的基石，为后续许多高级图算法奠定了重要基础。Dijkstra 算法的实现基于**贪心算法（Greedy Algorithm）**实现，并依赖于 **优先队列（Priority Queue）** 这一数据结构进行优化。

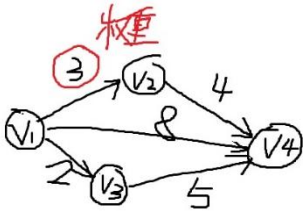
本文将系统介绍以 Dijkstra 算法 为代表的图论最短路算法的基本原理与典型实现，并重点探讨其在大数据环境下的应用场景与优化策略

图的概念及存储

1.图的概念





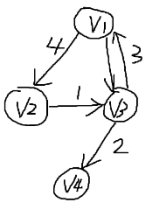


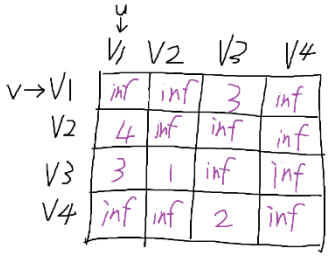
在计算机科学中，图（Graph） 是一种用于表示物体与物体之间关系的数据结构。一个图由两个集合构成：顶点（Vertex）V：用于存储更复杂的信息。边（Edge）E：代表关系。一条边连接两个顶点，可以是有方向的（有向图），也可以是无方向的（无向图）。边可以有权重（Weight），代表关系的成本或强度，在最短路算法中一般代表两个顶点之间的距离。

2.图的存储方式

在计算机中存储一个图，最常用的两种方法是邻接矩阵和邻接表。它们各有优劣，适用于不同的场景。

2.1 邻接矩阵

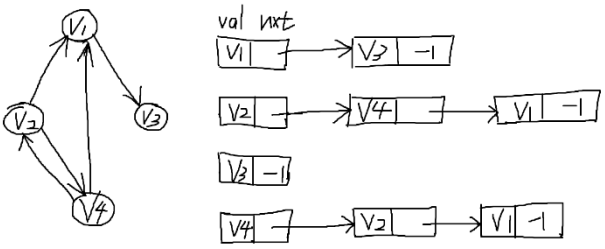




邻接矩阵使用一个 V x V 的二维数组来表示图，其中 V 是顶点数。对于顶点 u 和 v,g[u][v] 的值表示顶点u

到 v 边的权重（无权图用 1）。若不存在该边，则用一个特殊值 inf（无权图用 0）标记。适用场景：**稠密图**，即边数 E 接近顶点数 V 的平方时，空间利用率高。

2.2 邻接表



邻接表为每个顶点维护一个链表，链表中存储所有与该顶点直接相连的邻接顶点。对于每个顶点 u 都有一个链表来存储了所有由 u 出发的边所指向的顶点及其边的权重。适用场景：**稀疏图**，即边数远小于点数的平方时，这是最常见的情况。

在真实的大数据场景下，如道路网络、社交网络、通信拓扑等，图结构几乎都是稀疏图。因此，对于 Dijkstra 最短路算法，我们应该优先选择使用邻接表来存储图。

贪心算法

贪心算法是一种在每一步选择中都采取当前状态下最优的选择，从而希望导致全局最优的算法。它的特点是只考虑眼前的局部最优解，而不从整体上加以考虑。

问题：找零钱

假设我们需要用面值为 [1, 5, 10, 20, 50] 的纸币，凑出总额为 66 元 的现金，并要求张数最少。

在解决找零钱问题时，算法会从最大面额开始，尽可能多地使用，然后处理更小的面额，依次进行，直到凑出总金额。这种方法坚信每一步的局部最优选择能最终导向全局最优解。

贪心算法的时间复杂度通常较低，因为它只进行一轮线性的选择操作。对于找零钱问题，其时间复杂度为 O(n)。

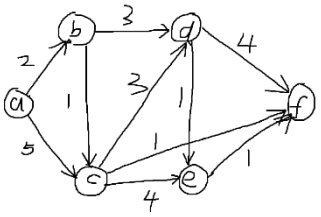
Dijkstra 算法概述

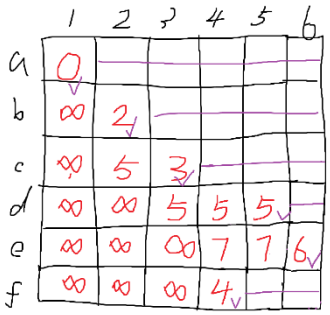
Dijkstra 算法是一种高效的最短路径搜索算法，用于在带权图中查找从一个源点到所有其他点的最短路径。它的特点是保证了在非负权图中能找到全局最优解。

Dijkstra 算法的核心思想是采用一种贪心策略，通过维护一个“最短距离”的集合，并逐步确定从源点到其余各顶点的最短路径。算法每次从**未确定**的节点中选取一个**距离源点最近**的节点，此时它的最短距离即被最终确定，然后通过这个节点的最短距离更新其所有

邻居节点的距离。这样，当所有节点都被处理过后，就能得到源点到所有节点的最短路径。

Dijkstra 算法主要包括三个步骤（假设 a 点为源点）：





第一步，找到距离 a 点最近且该未被访问过的节点

第二步，将该节点标记为访问过

第三步，通过该节点更新**与该节点相邻并且未被访问的节点**到 a 点的距离（取最小值）

Dijkstra 算法时间和空间复杂度

时间复杂度

Dijkstra 算法的时间复杂度主要包括两个部分：寻找最近的未确定节点和松弛邻居节点

```
for (int i = 1; i <= n; i++)
{
    //操作一：寻找最近的未确定节点
    int t = -1;
    for (int j = 1; j <= n; j++)
        if (!vis[j] && (t == -1 || dis[j] < dis[t]))
            t = j;
    vis[t] = true;
    //操作二：松弛邻居节点（用t号点更新邻居节点）
    for (int j = h[t]; j != -1; j = nxt[j])
    {
        int v = val[j]; //边的权重
        int p = ed[j];//邻居节点
        if (!vis[p])
            dis[p] = min(dis[p], dis[t] + v);
    }
}
```

操作一：**寻找最近的未确定节点**

这个操作需要遍历所有 n 个节点，检查它们是否已被处理并比较距离。这个操作在总共 n 次循环中，每次都需要 O(n)时间。因此，这一部分操作的总时间复杂度是 $O(n^2)$。

操作二：**松弛邻居节点**

这个操作是遍历当前节点 t 的所有出边。请注意，在整个算法运行期间，图中的每一条边都恰好被遍历一次（当这条边的起点被标记为“已确定”时）。如果图总共有 m 条边，那么所有松弛操作加起来的总时间是 O(m)。

空间复杂度

Dijkstra 算法的时间复杂度主要包括两个部分：图结构存储和算法运行辅助空间

图结构存储：代码使用邻接表来存储图。这需要存储所有的边和

```
int h[N], ed[M], nxt[M], w[M];//图的存储 O(N+M)
int dis[N];//距离数组O(N)
bool vis[N];//标记数组O(N)
```

指向下一条边的指针。存储整个图结构所需的空间与节点数 n 和边数 m 成正比，为 O(n + m)。这部分通常被视为输入数据本身占用的空间。

算法运行辅助空间：

1.距离数组：需要一个长度为 n 的数组（代码中的 dis）来记录每个节点当前的最近距离。空间为 O(n)。

2.标记数组：需要一个长度为 n 的布尔数组（代码中的 vis）来记录哪些节点的最短路径已被确定。空间为 O(n)。

最短路算法在大数据中的应用

随着大数据时代的到来，复杂网络关系的分析变得日益重要。最短路算法作为图论中的经典算法，能够在大规模网络数据中高效地寻找最优路径，因此在多个大数据应用场景中发挥着关键作用。

Dijkstra 算法由于其稳定的性能和可扩展的优化版本（如基于优先队列的优化），成为处理大数据中路径规划问题的核心工具之一。它特别适用于带权有向图或无向图中寻找单源最短路径，能够满足大数据环境下对实时性和准确性的高要求。

最短路算法在大数据中的应用主要体现在以下几个方面：

1. 网络路由与通信优化

在大规模通信网络（如互联网、物联网）中，Dijkstra 算法被广泛用于动态路由协议中。通过计算节点之间的最短路径，可以实现数据包的高效转发，降低网络延迟，提升通信质量。特别是在软件定义网络（SDN）中，结合大数据流量分析，系统能够实时调整路由策略，优化整体网络性能。

2. 交通与物流路径规划

在智慧交通和物流配送系统中，Dijkstra 算法用于计算最短或最快行驶路径。结合实时交通大数据（如路况、天气、历史拥堵数据），系统能够为车辆动态规划最优路线，显著提升运输效率，降低物流成本。例如，大型物流公司利用该算法进行每日数百万订单的路径优化。

3. 社交网络与推荐系统

在社交网络分析中，最短路径可用于衡量用户之间的“关系亲密度”。Dijkstra 算法帮助识别影响力传播的关键路径，优化信息推送和广告投放策略。此外，在基于图的推荐系

统中，通过计算用户与商品之间的最短关联路径，能够提升推荐的准确性和多样性。

优化

朴素 Dijkstra 算法在每一轮中都需要遍历所有节点来寻找距离起点最近的点，这在节点数量庞大时会导致较高的时间复杂度。为了提升算法效率，可以通过引入优先队列（最小堆）来优化这一查找过程，从而避免每次的线性扫描。

堆优化后的 Dijkstra 算法将查找最小距离节点的时间复杂度从 O(n) 降为 O(log n)，使得算法整体时间复杂度优化为 O(m log n)，特别适用于稀疏图或大规模网络中的最短路径计算。

```
int dijkstra()
{
    //创建小根堆（内部元素从小到大排列）
    typedef pair<int, int> PII;
    priority_queue<PII, vector<PII>, greater<PII>>q;
    //初始化
    memset(dis, 0x3f, sizeof dis);
    dis[1] = 0;
    q.push({0, 1});
    while (!q.empty())
    {
        //拿出堆顶节点（距离源点最近节点）O(logn)
        auto t = q.top();
        q.pop();
        int distance = t.first, p = t.second;
        if (vis[p]) continue;
        vis[p] = true;
        //松弛邻居节点
        for (int j = h[p]; j != -1; j = nxt[j])
        {
            int px = ed[j];
            int wx = w[j];
            if (dis[px] > distance + wx)
            {
                dis[px] = distance + wx;
                q.push({dis[px], px});
            }
        }
    }
    return dis[n];
}
```

总结

Dijkstra 算法作为一种高效的单源最短路径算法，具有稳定的性能和广泛的应用前景。在大数据与复杂网络环境下，通过合理的优化（如堆优化），Dijkstra 算法能够高效地处理大规模图中的路径规划任务，特别适用于对实时性要求较高的场景，如网络路由、智慧交通、社交网络分析及资源调度等领域。

[参考文献]

[1] 徐鹏. Dijkstra 算法在 GIS 路径规划中的优化与应用[D]. 南京: 南京师范大学, 2021.

[2] 李晓明. 电力通信网路由优化的 Dijkstra 改进算法研究[D]. 武汉: 华中科技大学, 2023.

[3] 王浩宇. 物流配送路径优化的多约束 Dijkstra 算法研究[D]. 广州: 华南理工大学, 2022.