Dijkstra算法及其分析

### ****迪杰斯特拉算法介绍****

迪杰斯特拉(Dijkstra)算法是典型[最短路径](https://so.csdn.net/so/search?q=%E6%9C%80%E7%9F%AD%E8%B7%AF%E5%BE%84&spm=1001.2101.3001.7020" \t "https://blog.csdn.net/diana_jiuri1314/article/details/_blank)算法，用于计算一个节点到其他节点的最短路径。  
它的主要特点是以起始点为中心向外层层扩展(广度优先搜索思想)，直到扩展到终点为止。

### 实现过程

将结点分成两个集合：已确定最短路长度的点集（记为S集合）的和未确定最短路长度的点集（记为T集合）。一开始所有的点都属于T集合。

初始化dis(s)=0dis(s)=0，其他点的dis均为正无穷大 +\infty。

然后重复这些操作：

1. 从T集合中，选取一个最短路长度最小的结点，移到S集合中。
2. 对那些刚刚被加入S集合的结点的所有出边执行松弛操作。

直到T集合为空，算法结束。

朴素Dijkstra算法：

时间复杂度：不使用任何数据结构进行维护，每次 2 操作执行完毕后，直接在 T 集合中暴力寻找最短路长度最小的结点。2 操作总时间复杂度为O(m) O(m)，1 操作总时间复杂度为O(n\*\*2) O(n^2)，全过程的时间复杂度为 O(n\*\*2+m)=O(n\*\*2)O(n^2 + m) = O(n^2)。

代码实现：Python版本

1. class Edge:
2. v = 0
3. w = 0
4. e = [[Edge() for i in range(maxn)] for j in range(maxn)]
5. dis = [63] \* maxn; vis = [] \* maxn
6. def dijkstra(n, s):
7. dis[s] = 0
8. for i in range(1, n + 1):
9. u = 0
10. mind = 0x3f3f3f3f
11. for j in range(1, n + 1):
12. if vis[j] == False and dis[v] < mind:
13. u = j
14. mind = dis[j]
15. vis[u] = True
16. for ed in e[u]:
17. v, w = ed.v, ed.w
18. if dis[v] > dis[u] + w:
19. dis[v] = dis[u] + w

堆优化Dijkstra算法：

时间复杂度：和二叉堆类似，但使用优先队列时，如果同一个点的最短路被更新多次，因为先前更新时插入的元素不能被删除，也不能被修改，只能留在优先队列中，故优先队列内的元素个数是O(m)的，时间复杂度为O(m\*logm) O(m \log m)。

代码实现：Python版本

1. def dijkstra(e,s):
2. '''
3. 输入：
4. e:邻接表
5. s:起点
6. 返回：
7. dis:从s到每个顶点的最短路长度
8. '''
9. dis = defaultdict(lambda:float("inf"))
10. dis[s] = 0
11. q = [(0,s)]
12. vis = set()
13. while q:
14. \_, u = heapq.heappop(q)
15. if u in vis: continue
16. vis.add(u)
17. for v,w in e[u]:
18. if dis[v] > dis[u] + w:
19. dis[v] = dis[u] + w
20. heapq.heappush(q,(dis[v],v))
21. return dis

Dijkstra算法实例应用：有N个村庄,村庄之间的道路有M条,连接类似,从a村到b村的距离为c

同时,有k个村庄有商店,编号和村庄的对应.

最后,有Q个询问,每次询问一个编号,这个编号的村庄最近的商店的距离是多少?

输出:输出每个询问的结果.

思路:可以想象每个商店是独立的个体,到达不同的村庄的距离就是和之前存的dist类似.这个时候有一个空结点0号结点,直接与这些点相连,到这些商店的距离都是0.

那么要求某个村庄最近的就是从0开始到这个村庄的最短路径,因为本身要求的就是村庄到商店的最短路,这个时候加上了一段距离为0的点开始,对这个询问没有影响,对距离也没有影响,所以反过来,求0结点到这个村庄的最短路距离就是求村庄到这些商店的最短路.

做法主要是在Dijkstra算法的基础之上,从0号结点开始到损耗也是0,每次加入小卖部的时候,从0号结点开始加入,损耗也设置成为0.

代码实现：

1. #include <iostream>
2. #include <cstring>
3. #include <algorithm>
4. #include <queue>
6. using namespace std;
7. typedef pair<int, int> PII;
9. const int N = 100010  , M = N\*3;
11. int n,m;
12. int h[N],e[M],ne[M],w[M],idx;
13. int dist[N];
14. bool st[N];
16. void add(int a,int b,int c){
17. e[idx] = b ,w[idx] = c , ne[idx] = h[a] , h[a] = idx++;
18. }
20. void dijkstra()  *// 求1号点到n号点的最短路距离*
21. {
22. memset(dist, 0x3f, sizeof dist);
23. dist[0] = 0;
24. priority\_queue<PII, vector<PII>, greater<PII>> heap;
25. heap.push({0, 0});
27. while (heap.size())
28. {
29. auto t = heap.top();
30. heap.pop();
32. int ver = t.second, distance = t.first;
34. if (st[ver]) continue;
35. st[ver] = true;
37. for (int i = h[ver]; i != -1; i = ne[i])
38. {
39. int j = e[i];
40. if (dist[j] > dist[ver] + w[i])
41. {
42. dist[j] = dist[ver] + w[i];
43. heap.push({dist[j], j});
44. }
45. }
46. }
47. }
49. int main()
50. {
51. cin>>n>>m;
52. memset(h,-1,sizeof h);
53. while(m--){
54. int a,b,c;
55. scanf("%d%d%d", &a, &b , &c);
56. add(a, b, c);
57. add(b, a, c);
58. }
59. scanf("%d", &m);
60. while(m--){
61. int x;
62. scanf("%d", &x);
63. add(0,x,0);
64. }
65. scanf("%d", &m);
66. dijkstra();
68. while (m -- ){
69. int x;
70. scanf("%d", &x);
71. printf("%d\n",dist[x]);
72. }

Dijkstra算法的局限性：

1. 不能处理带有负权边的图：Dijkstra算法要求所有边的权值都为正数，否则可能会出现错误的结果。如果存在负权边，则可能会导致算法无法找到正确的最短路径，或者陷入死循环。
2. 空间复杂度高：Dijkstra算法使用了一个距离数组来存储源节点到其他节点的最短距离，如果节点数很大，这个数组的空间复杂度会很高，导致算法运行缓慢。
3. 时间复杂度高：在稠密图中，Dijkstra算法的时间复杂度为O(N^2)，其中N是节点数，这可能会导致算法运行缓慢。在稀疏图中，使用堆优化可以将时间复杂度降至O(MlogN)，其中M是边数，但是在实际应用中，堆操作的常数因子可能会很大，导致算法的实际运行时间不尽如人意。
4. 不能处理含有环的图：Dijkstra算法不能处理含有环的图，否则可能会导致算法无法找到最短路径或者陷入死循环。如果需要处理含有环的图，可以使用Bellman-Ford算法或者SPFA算法。

综上所述，尽管Dijkstra算法在实际应用中有一些局限性，但是它仍然是一种非常有效的算法，可以在很多场景下得到广泛应用。如果要处理带有负权边的图，可以使用Bellman-Ford算法或者SPFA算法；如果要处理含有环的图，则需要使用Bellman-Ford算法或者SPFA算法。

资料来源：

[最短路 - OI Wiki (oi-wiki.org)](https://oi-wiki.org/graph/shortest-path/" \l "floyd-%E7%AE%97%E6%B3%95)

作者：柒龍猪  
链接：<https://www.zhihu.com/question/591705246/answer/2952776552>

Csdn博客：https://blog.csdn.net/diana\_jiuri1314/article/details/115676994?ops\_request\_misc=&request\_id=&biz\_id=102&utm\_term=dijkstra%E7%AE%97%E6%B3%95c&utm\_medium=distribute.pc\_search\_result.none-task-blog-2~all~sobaiduweb~default-2-115676994.142^v88^control,239^v2^insert\_chatgpt&spm=1018.2226.3001.4187

Csdn博客：dijkstra算法实例 acwing模板题