Floyd算法及其应用分析

算法简介：Floyd算法是一种用于查找图中最短路径的动态规划算法。它通过迭代计算来优化最短路径，使用邻接矩阵表示图，并能够找到任意两个顶点之间的最短路径长度。该算法的核心思想是通过遍历所有顶点对和中间顶点，更新路径长度以得到最短路径。

时间复杂度：O(n^3)。

空间复杂度：O(n^2)。

适用情况：可以被运用在有向图，无向图，边权正负与算法正确性无关，但要求图上不存在负环。

优点：Floyd是一种全源最短路算法，算法支持负权边，使得它适用范围场景更加灵活。在解决规模较小的问题的时候更加灵活简单，便于实现。

缺点：Floyd的时间复杂度和空间复杂度（相较于邻接表）都较高，因而在处理大规模数据的时候并不太实用，对于稀疏图的效率低，因为边的数目过少，扩展时浪费的时间更多。

代码实现：

1. *//MAXN为最大的点的数量*
2. int f[MAXN][MAXN];*//f[i][j]代表节点i到节点j的最短路*
3. int main()
4. {
5. int n,m;*//n为图中点的数量，m为图中边的数量*
6. scanf("%d%d",&n,&m);
7. for(int i=1;i<=n;i++)
8. for(int j=1;j<=n;j++)
9. f[i][j]=1e9;*//初始每个节点对之间都是不可达，设为最大值*
10. for(int i=1;i<=n;i++)f[i][i]=0;*//自己到自己最短路为0*
11. for(int i=0;i<m;i++){
12. int x,y,w;*//分别是起点，终点和边权*
13. scanf("%d%d%d",&x,&y,&w);
14. f[x][y]=w;
15. f[y][x]=w;*//如果图为无向图*
16. }
17. for(int k=1;k<=n;k++)*//枚举转移节点*
18. for(int i=1;i<=n;i++)
19. for(int j=1;j<=n;j++){
20. f[i][j]=min(f[i][j],f[i][k]+f[k][j]);
21. *//枚举k作为i->j的转移节点计算*
22. }
23. for(int i=1;i<=n;i++){
24. for(int j=1;j<=n;j++){
25. printf("%d ",f[i][j]);
26. *//输出i到j的最短路*
27. }
28. puts("");
29. }
30. return 0;
31. }

代码思想：

我们记f[k][i][j]为在只允许通过1,2,..k的情况下，节点i到节点j的最短路径，显然我们有f[0][i][j]=原图给定的<i,j>之间的边权。想象转移过程:f[k][i][j]=min(f[k][i][j],f[k-1][i][k]+f[k-1][k][j])即我们通过枚举中间点k进行转移，从而使状态的第一维扩展，最终答案即为f[n][i][j]。我们可以注意到f[k][i][j]的转移与第一维无关，从而优化空间复杂度为O(n^2)。

简要证明转移与第一维无关：f[k][i][j]的转移仅仅与k-1有关，而在转移f[k][i][j]之前,f[k][i][j]存的值就是上一次f[k-1][i][j]的答案，所以运用滚动数组的方式，我们可以取消掉第一维来节省空间。

应用场景：

·给定一个正权无向图，寻找一个权值和最小的环：

由代码思想可得在第一个循环中我们规定了每次进行转移的最大的点的编号为k。在求最小环的思路中，我们同样可以认为环上的最大编号的点就是为k，在转移的过程中，记录枚举答案ans=f[i][j]+w[i][k]+w[k][j](注意到不是枚举f[i][k]+f[k][j]因为此时k还没进入到最短路更新中，先进行最小环寻找，不然最小环中可能会出现重点。)

代码实现:

1. int ans=1e9;
2. for(int k=1;k<=n;k++)*//枚举转移节点*
3. {
4. *//枚举小于k的点计算。*
5. for(int i=1;i<k;i++){
6. for(int j=i+1;j<k;j++){
7. ans=min(ans,f[i][j]+w[i][k]+w[k][j]);
8. }
9. }
10. for(int i=1;i<=n;i++)
11. {
12. for(int j=1;j<=n;j++)
13. {
14. f[i][j]=min(f[i][j],f[i][k]+f[k][j]);
15. }
16. }
17. }

·已知一个有向图中任意两点之间是否有连边，要求判断任意两点是否连通：

该问题即是求原图的传递闭包。

我们只需要按照Floyd的过程，逐个加入点判断一下。

只是此时的边的边权变为0/1，而取min操作变成了|(或)运算。

再进一步用 bitset 优化，复杂度可以到O(n^3/**ω**)的级别

代码实现：

1. bitset<MAXN> f[MAXN];
2. for (k = 1; k <= n; k++)
3. for (i = 1; i <= n; i++)
4. if (f[i][k]) f[i] = f[i] | f[k];

·求解恰好经过T条边的全源最短路

离散课中我们学习到了A^m中的节点A^m i,j即vi到vj有aij^m条长度为m的路。我们同样运用这个思想计算恰好经过k条边的最短路。

我们记a[i][j]为恰好经过x条路的两两最短路,b[i][j]为恰好经过y条路的两两最短路。

如果c[i][j]=min(c[i][j],a[i][k]+b[k][j]),c[i][j]中储存的就是经过中转点k的恰好经过x+y次的最短路。

在最开始我们拥有原图矩阵w[i][j]他代表恰好经过1次的最短路，而W^T即为最终答案。

如果T较大我们可以通过快速幂进行计算。时间复杂度为O(log(T)N^3)

至于如果只询问S到其他点的最短路，而非是全源最短路，开始的矩阵可以只需要压缩成一行即只包含第S行的矩阵。

代码实现：(使用c++实现)

1. int main()
2. {
3. int n, m;
4. cin >> n >> m;
5. vector<vector<vector<int>>> g(31, vector<vector<int>>(n + 1, vector<int>(n + 1, INF)));
6. *// vector<vector<vector<int>>> g(31, vector<vector<int>>(n + 1, vector<int>(n + 1, INF)));*
7. for (int i = 1; i <= m; i++)
8. {
9. int u, v, w;
10. cin >> u >> v >> w;
11. g[0][u][v] = w;
12. }
13. auto floyd = [=](vector<vector<int>> a, vector<vector<int>> b)
14. {
15. vector<vector<int>> dis(n + 1, vector<int>(n + 1, INF));
16. rep(k, 1, n) rep(i, 1, n) rep(j, 1, n) dis[i][j] = min(dis[i][j], a[i][k] + b[k][j]);
17. return dis;
18. };
19. for (int i = 1; i <= 30; i++)
20. g[i] = floyd(g[i - 1], g[i - 1]);
21. auto qpow = [&](int s, int k)
22. {
23. vector<int> ans(n + 1, INF);
24. ans[s] = 0;
25. int cnt = 0;
26. while (k)
27. {
28. if (k & 1)
29. {
30. vector<int> res(n + 1, INF);
31. for (int i = 1; i <= n; i++)
32. {
33. for (int j = 1; j <= n; j++)
34. {
35. res[j] = min(res[j], ans[i] + g[cnt][i][j]);
36. }
37. }
38. ans = res;
39. }
40. cnt++;
41. k >>= 1;
42. }
43. return ans;
44. };
45. int \_;
46. cin >> \_;
47. while (\_--)
48. {
49. int s, k;
50. cin >> s >> k;
51. vector<int> ans = qpow(s, k);
52. for (int i = 1; i <= n; i++)
53. {
54. if (ans[i] == INF)
55. cout << "-1 ";
56. else
57. cout << ans[i] << " ";
58. }
59. cout << "\n";
60. }
61. return 0;
62. }

文献来源：

[Floyd算法\_百度百科 (baidu.com)](https://baike.baidu.com/item/Floyd%E7%AE%97%E6%B3%95/291990?fromtitle=floyd&fromid=23665947&fr=aladdin)

[Floyd–Warshall algorithm - Wikipedia](https://en.wikipedia.org/wiki/Floyd%E2%80%93Warshall_algorithm)

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

[(11条消息) Floyd算法思路以及扩展应用\_floyd为什么压缩维度仍然是对的\_荼白777的博客-CSDN博客](https://blog.csdn.net/weixin_45724872/article/details/122811602)