两种存图方式

1、邻接矩阵：

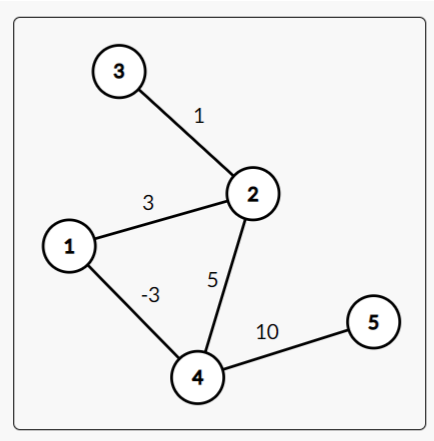
$edge[i][j]$ ： i 点与 j 点间的边权， 空间复杂度$O(n^2)$

2、邻接表：

$vector < pair < int, int >> adj[i]$ ： 保存从 i 出发的每条边的信息

也可以用链式前向星

空间复杂度 $O(n+m)$



如：这张无向图的邻接表可以表示为

adj[1]	**(2, 3)**	**(4, -3)**			
adj[2]	(1, 3)	(3, 1)	(4, 5)		
adj[3]	(2, 1)				
adj[4]	(1, -3)	(2, 5)	(5, 10)		
adj[5]	(4, 10)				

代码实现

```
vector<pair<int, int>> adj[N];
void add(int u, int v, int w) {
    adj[u].push_back({v, w});
}
```

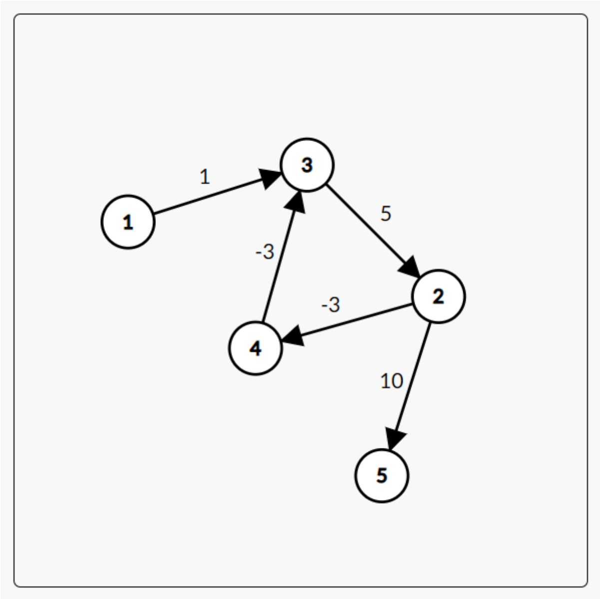
邻接矩阵的优缺点：

优点：访问更快

缺点：占用空间大

最短路存在条件

u,v 间存在最短路 $\iff$ 从 u 出发到 v 的所有路径中，不存在一个负环



单源最短路

源：起点。

指定一个起点， 求出到其他点的最短距离

bellman-ford算法

该算法的流程如下：

$d[u]$：从起点 st 出发， 到 u 的最短路

1 令

$$d[i] = \begin{cases} 0 & i = st \\ INF & else. \end{cases}$$

2 扫描每一条边，对于边 (u,v,w), 若 $d[v] > d[u] + w$ ，则令$d[v] = d[u] + w$

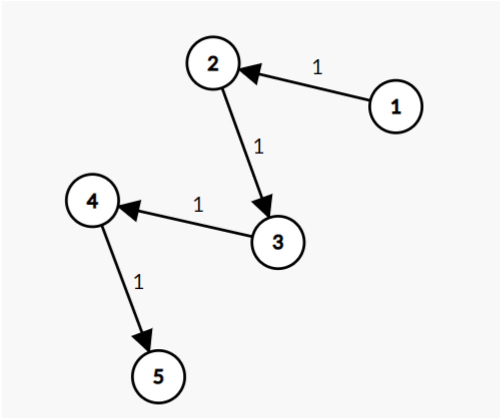
3 重复上述步骤 $n-1$ 次，则可得单源最短路

[bellman-ford算法演示]

对于一张 n 个点， m 条边的图而言

性质：任意一条最短路最多经过 n 个点

反证法：否则，必然有一点 p 经过两次，则第一次和第二次间的所有点构成一个负环



主代码

```
for (int i = 1; i < n; i++) {
    for (int u = 1; u <= n; u++) {
        for (int j = 0; j < adj[u].size(); j++) {
            int v = adj[u][j].first, w = adj[u][j].second;
            if (d[v] > d[u] + w) {
                d[v] = d[u] + w;
            }
        }
    }
}
```

利用bellman-ford算法判负环

```
for (int i = 1; i <= n; i++) {
    for (int u = 1; u <= n; u++) {
        for (int j = 0; j < adj[u].size(); j++) {
            int v = adj[u][j].first, w = adj[u][j].second;
            if (d[v] > d[u] + w) {
                if (i == n) return true; // 有负环
                d[v] = d[u] + w;
            }
        }
    }
}
return false;
```

时间复杂度 $O(nm)$

spfa

全名 shortest path fast algorithm。

算法流程：

1、建立一个队列，最初队列中只含有起点1。

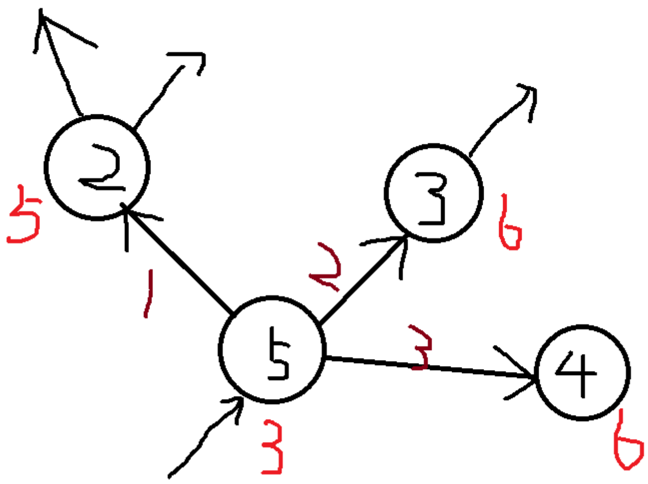
2、取出队头节点 u ，扫描他的所有出边 (u,v,w) ，若 $d[v]>d[u]+w$, 则 $d[v]=d[u]+w$ 。同时，若 v 不在队列，则把 v 入队。

3、重复上述步骤，直到队列为空

解释

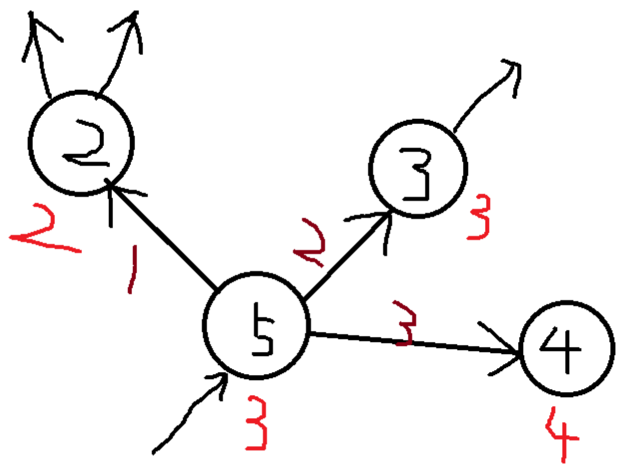
遍历从5号点出发的所有边

图1



(5,2)， (5,3)的访问有效， (5, 4)无效

图2



所有访问无效

spfa代码

```
int vis[N], cnt[N]; // cnt表示最短路径所含边数， cnt >= n时有负环
bool spfa(int st) { // 有负环， return true
    memset(cnt, 0, sizeof cnt); //
    for (int i = 1; i <= n; i++)
        d[i] = INF;
    d[st] = 0;
    queue<int> que;
    que.push(st);
    memset(vis, 0, sizeof vis);
    vis[st] = 1;
```

```
    while (que.size()) {
        int u = que.front(); que.pop();
        vis[u] = 0;
        for (auto [v, w] : adj[u]) { // 结构化绑定要求c++17标准
            if (d[v] > d[u] + w) {
                d[v] = d[u] + w;
                cnt[v] = cnt[u] + 1;
                if (cnt[v] >= n) return true;
                if (!vis[v]) {
                    que.push(v);
                    vis[v] = 1;
                }
            }
        }
    }
    return false;
}
```

时间复杂度 $O(km)$ 。在随机图上，k是一个较小的常数；但在特殊构造的图上，该算法会退化成 $O(nm)$ 。

dijkstra 算法

算法流程：

1、令

$$d[i] = \begin{cases} 0 & i = st \\ INF & else. \end{cases}$$

2、找出一个未被标记的， $d[u]$ **最小**的节点 u, 然后标记 u

3、扫描 u 的所有出边 (u, v, w)， 若 $d[v] > d[u] + w$, 则令 $d[v] = d[u] + w$

4、重复 2~3 两个步骤，直到所有节点都被标记

dijkstra算法演示

性质：在**非负权**图上，对于被标记的节点 u， $d[u]$ 不可能在后续操作中再被更改

朴素dij代码

```
for (int i = 1; i <= n; i++) {
    int u = 0;
    for (int j = 1; j <= n; j++) {
        if (!vis[j] && (u == 0 || d[j] < d[u])) u = j;
    }
    v[u] = 1;
    for (auto [v, w] : adj[u]) {
        d[v] = min(d[v], d[u] + w);
    }
}
```

时间复杂度$O(n^2 + m)$

找最小值的过程可以用优先队列优化

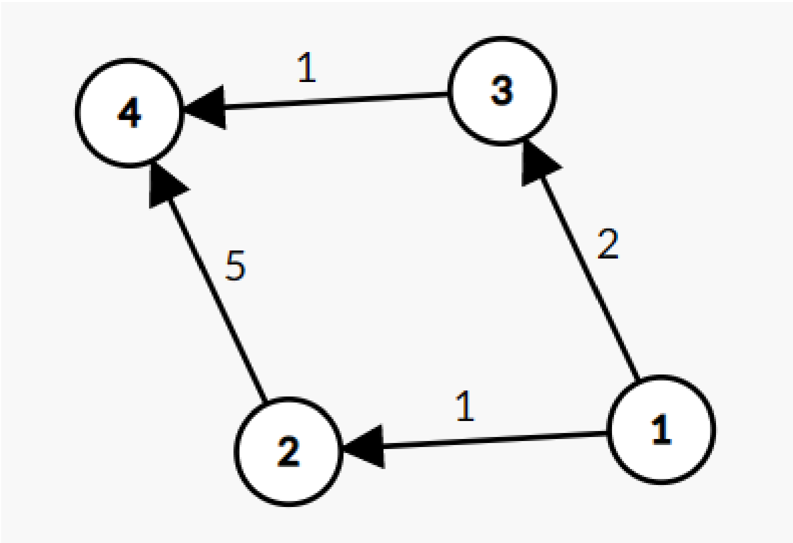
堆优化dij代码

```cpp
priority_queue<pair<int, int>, vector<pair<int, int>>, greater<pair<int, int>>> que; // 小根堆
que.push({0, st}); // 存pair对 (d[u], u) 。不能反过来，第一维作为排序依据

while (que.size()) {
    int u = que.top().second; que.pop();
    if (vis[u]) continue; // 如果被标记过了，则直接跳过
    vis[u] = 1;
    for (auto [v, w] : adj[u]) {
        if (d[v] > d[u] + w) {
            d[v] = d[u] + w;
            que.push({d[v], v});
        }
    }
}
```

时间复杂度$O(mlogm)$

注意：稠密图上朴素的dij算法更快



对于vis标记的解释

注意：dij只能用于非负权图！！！

任意两点间最短路

floyd算法

设 $d[k,u,v]$ 表示经过若干个编号不超过 k 的节点（不包括u,v）， 从 u 到 v 的最短路

可以列出 dp 方程

$$d[k,u,v] = min(d[k-1,u,v], d[k-1,u,k] + d[k-1,k,v])$$

对于该方程的解释是： u经过前k个点到v，有两种可能

1、u不经过第k个点，而是经过前k-1个到 v

2、u经过前k个点，且第k个点必经过

对于第二种可能，我们从路径中删去k， 则最短路分成u->k, 和 k->v两段，且中间经过的点必然 < k

另外可以发现，dp方程第一维是可以优化的 （Floyd的题点的个数一般为300~500, 不优化空间会炸掉）

floyd代码

```
for (int k = 1; k <= n; k++) {
    for (int u = 1; u <= n; u++) {
        for (int v = 1; v <= n; v++) {
            d[u][v] = min(d[u][v], d[u][k] + d[k][v]);
        }
    }
}
```

时间复杂度 $O(n^3)$

floyd计算传递闭包

传递性：对于定义在集合 S 上的二元关系 I, 若对于 $\forall a, b, c \in S$, 只要有 aIb 且 bIc ， 则有aIc。

例如：图论中的**可达**关系

```
for (int k = 1; k <= n; k++) {
    for (int u = 1; u <= n; u++) {
        for (int v = 1; v <= n; v++) {
            d[u][v] |= d[u][k] & d[k][v];
        }
    }
}
```

特殊的最短路

有向无环图 -> 拓扑排序 $O(n)$

边权为 1 的图 -> bfs $O(n)$

边权为 0 或 1 的图 -> 双端队列bfs $O(n)$

总结

做最短路题时，要根据数据范围选择合适的方法

对于一张 n 个点， m 条边的图而言（图论一般题 $n, m \le 1e5$ ）

floyd 复杂度为 $O(n^3)$ ， $n \le 500$ 可以考虑， $n > 500$ 一般不考虑

dijkstra 复杂度稳定在 $O(mlogm)$ ，几乎可以适用所有权值≥ 0 的图

spfa 复杂度 $O(km)$ ，按复杂度上界算会TLE， 但也有概率能过题。对于负权图，实在没办法的时候可以试试。

例题

acwing 342. 道路与航线 - AcWing题库)

工具

[Graph Editor (csacademy.com)](https://csacademy.com/app/graph_editor/)