[图论](#图论)  
 [建图 邻接表](#建图-邻接表)  
 [最短路](#最短路)  
 [单源最短路 dijkstra](#单源最短路-dijkstra)  
 [负权单源最短路](#负权单源最短路)   
 [Bellman–Ford](#bellman--ford)  
 [SPFA](#spfa)  
 [多源最短路](#多源最短路)   
 [floyd](#floyd)  
 [多次 dijkstra](#多次-dijkstra)  
 [负权单源最短路 Johnson](#负权单源最短路-johnson)

# 图论

## 建图 邻接表

int n,m;// 点, 边数  
vector<int> a[N];  
  
for(int i=1;i<=m;++i)  
{  
 int x,y; cin >> x,y;  
 a[x].push\_back(y);  
 /\* 若是无向图，则双向边  
 a[y].push\_back(x);  
 \*/  
}

## 最短路

### 单源最短路 dijkstra

void dijkstra(int st,int d[])  
{  
 for(int i=1;i<=n;++i) d[i] = M;// 初始化  
 priority\_queue<edge> pq;  
 d[st] = 0;// 起始位置  
 bitset<N> vis;// 去重，防止重复访问降速  
 pq.push({st,d[st]});  
 while(pq.size())  
 {  
 int x = pq.top().y; pq.pop();  
 if(vis[x]) continue;  
 vis[x] = true;  
 for(auto &[y,v] : a[x])  
 {  
 if(d[y] > d[x] + v)  
 {  
 d[y] = d[x] + v;  
 pq.push({y,d[y]});  
 }  
 }  
 }  
}

### 负权单源最短路

#### Bellman–Ford

bool bellman\_ford(void)  
{  
 // 用于判负环，在 第 n 次任可松弛，证明有负环  
 bool op = false;  
 for(int i=1;i<=n;++i)// n 次枚举  
 {  
 op = false;  
 for(int x=1;x<=n;++x)// 共 m 条边  
 {  
 for(auto &[y,v] : a[x])  
 {  
 if(d[y] > d[x] + v)  
 {  
 d[y] = d[x] + v;  
 op = true;   
 }  
 }  
 }  
 }  
 return op;  
}

#### SPFA

但**最坏 ，且很容易这么卡**

bool spfa(int st,int d[])  
{  
 // 保证每个点只有松弛过才能用来松弛  
 queue<int> q;  
 bitset<N> inq;  
 q.push(st);  
 vector<int> cnt(n+1);  
 while(q.size())  
 {  
 int x = q.front(); q.pop(); inq[x] = false;  
 for(auto &[y,v] : a[x])  
 {  
 if(d[y] > d[x] + v)  
 {  
 // 一条边最多松弛 $n-1$ 次，否则负环  
 if(++ cnt[y] >= n) return true; // 负环  
 d[y] = d[x] + v;  
 if(!inq[y]) q.push(y), inq[y] = true;  
 }  
 }  
 }  
 return false;  
}

### 多源最短路

#### floyd

**使用邻接矩阵**

int n,m;  
int d[N][N];  
  
for(int i=1;i<=n;++i)  
{  
 for(int j=1;j<=n;++j) d[i][j] = (i == j ? 0 : M);  
}  
  
void floyd(void)  
{  
 for(int k=1;k<=n;++k)  
 {  
 for(int i=1;i<=n;++i)  
 {  
 for(int j=1;j<=n;++j)  
 {  
 d[i][j] = min(d[i][j],d[i][k]+d[k][j]);  
 }  
 }  
 }  
}

#### 多次 dijkstra

对一个点求多次 dijkstra 也可以

### 负权单源最短路 Johnson

**spfa判负环需要 次，因为加上虚拟零点后有 个点**

int d[N][N], h[N];// 最短路，势能
  
  
bool johnson(void)
  
{
  
 // 构造虚拟零点
  
 for(int i=1;i<=n;++i) a[0].push\_back({i,0});
  
 // spfa 求势能 h
  
 for(int i=1;i<=n;++i) h[i] = M;
  
 if(spfa(0,h)) return true;// 判负环
  
 // 用势能改造边为非负权
  
 for(int i=1;i<=n;++i)
  
 {
  
 for(auto &[j,v] : a[i]) v += h[i] - h[j];
  
 }
  
 // n 次单源最短路
  
 for(int i=1;i<=n;++i) dijkstra(i,d[i]);
  
 // 复原
  
 for(int i=1;i<=n;++i)
  
 {
  
 for(int j=1;j<=n;++j) d[i][j] -= h[i] - h[j];
  
 }
  
 return false;
  
}