最短路

DIJ

记得进行初始化

```
dist = new llint[n + 1];    memset(dist, 0x3f, sizeof(llint) * (n + 1));
used = new bool[n + 1];     memset(used, 0, sizeof(bool) * (n + 1));
way_count = new llint[n + 1];   memset(way_count, 0, sizeof(llint) * (n + 1));
```

算法本体

```
struct node_dist {
        int p;
        llint dis;
        inline friend bool operator < (node_dist a, node_dist b) { return a.dis >
b.dis; }
        node_dist(int p, llint dis) { this->p = p; this->dis = dis; }
    };
    priority_queue<node_dist> heap;

void Dij(int start)
{
    dist[start] = 0;
    way_count[start] = 1;
    heap.push(node_dist(start, dist[start]));
    while (!heap.empty())
    {
        node_dist cur = heap.top(); heap.pop();
        if (used[cur.p]) continue;
        node& p = nod[cur.p];
        used[cur.p] = true;

        for (int i=0;i<=p.e.size();i++)
        {
            edge e = p.e[i];
            llint newdist = e.w + dist[e.u];
            //松弛同时维护最短路条数、记录upedg
            if (newdist < dist[e.v] && !used[e.v])
            {
                dist[e.v] = newdist;
                heap.push(node_dist(e.v, newdist));
                way_count[e.v] = way_count[e.u] % mod;
                nod[e.v].upedg = e;
            }
            else if (newdist == dist[e.v]) way_count[e.v] = (way_count[e.v] +
way_count[e.u])%mod;
        }
```

```
    }
    return dist;
}
```

路径构造

得到的路径是倒序的

```
int edges[MAXN];int count;
void get_path(int start,int end)
{
    count = 0;
    while(end != start)
    {
        edges[count++] = end;
        end = (nod[end].upedg).u;
    }
    edges[count++] = start;
}
```