最大流

Maximum Flow

Ford-Fulkerson 方法

Concepts：

- 剩余容量 Residual Capacity：一条边的容量与流量之差，$c_f(u,v) = c(u,v) - f(u,v)$
- 残量网络 Residual Network：所有剩余容量大于 0 的边的生成子图
- 增广路 Augmenting Path：原图 G 中，一条从源点到汇点的由剩余容量都大于 0 的边构成的路径

Idea：不断寻找增广路直到找不到为止。

Edmonds-Karp

Idea：bfs 寻找增广路。

Complexity：$O(VE^2)$

ATT：链式前向星存储时，`edgeNum` 初始化为1；建图时建流为 `0` 的反向边。

Code：

```
int pre[N], minFlow[N];
int bfs(){
    queue<int> q;
    for(int i = 1; i <= n; i++){
        pre[i] = 0;
        minFlow[i] = INF;
    }
    q.push(src);
    while(!q.empty()){
        int cur = q.front(); q.pop();
        for(int i = head[cur]; i; i = edge[i].nxt){
            if(edge[i].flow && !pre[edge[i].to]){
                pre[edge[i].to] = i;
```

```
                minFlow[edge[i].to] = min(minFlow[cur], edge[i].flow);
                q.push(edge[i].to);
            }
        }
    }
    if(pre[dst] == 0)   return -1;
    return minFlow[dst];
}

int EK(){
    int flow = 0, maxflow = 0;
    while((flow = bfs()) != -1){
        int t = dst;
        while(t != src){
            edge[pre[t]].flow -= flow;
            edge[pre[t]^1].flow += flow;
            t = edge[pre[t]^1].to;
        }
        maxflow += flow;
    }
    return maxflow;
}
```

Dinic

Idea：bfs 将图分层，dfs 按分层图寻找增广路。

Optimization：当前弧优化。

Complexity：$O(V^2E)$

ATT：链式前向星存储时，`edgeNum` 初始化为1；建图时建流为 `0` 的反向边。

Code：

```
// s refers to source, t refers to destination
bool inq[N];
int dep[N];
bool bfs(){
    for(int i = 1; i <= n; i++)
        dep[i] = INF, inq[i] = 0;
    queue<int> q;
    q.push(s);
    inq[s] = 1;
```

```
    dep[s] = 0;
    while(!q.empty()){
        int cur = q.front(); q.pop();
        inq[cur] = 0;
        for(int i = head[cur]; i; i = edge[i].nxt){
            if(dep[edge[i].to] > dep[cur] + 1 && edge[i].flow){
                dep[edge[i].to] = dep[cur] + 1;
                if(!inq[edge[i].to]){
                    q.push(edge[i].to);
                    inq[edge[i].to] = 1;
                }
            }
        }
    }
    if(dep[t] != INF)   return 1;
    return 0;
}
int dfs(int x, int minFlow){
    int flow = 0;
    if(x == t)  return minFlow;
    for(int i = head[x]; i; i = edge[i].nxt){
        if(dep[edge[i].to] == dep[x] + 1 && edge[i].flow){
            flow = dfs(edge[i].to, min(minFlow, edge[i].flow));
            if(flow){
                edge[i].flow -= flow;
                edge[i^1].flow += flow;
                return flow;
            }
        }
    }
    return 0;
}
int Dinic(){
    int maxFlow = 0, flow = 0;
    while(bfs()){
        while(flow = dfs(s, INF))
            maxFlow += flow;
    }
    return maxFlow;
}
```

Code（当前弧优化）：

```
namespace FLOW{

    int n, s, t;
```

```
    struct Edge{
        int nxt, to;
        LL flow;
    }edge[M<<1];
    int head[N], edgeNum = 1;
    void addEdge(int from, int to, LL flow){
        edge[++edgeNum].nxt = head[from];
        edge[edgeNum].to = to;
        edge[edgeNum].flow = flow;
        head[from] = edgeNum;
    }
    void ae(int from, int to, LL flow){
        addEdge(from, to, flow), addEdge(to, from, 0);
    }

    bool inq[N];
    int dep[N], curArc[N];
    bool bfs(){
        for(int i = 1; i <= n; i++)
            dep[i] = 1e9, inq[i] = 0, curArc[i] = head[i];
        queue<int> q;
        q.push(s);
        inq[s] = 1;
        dep[s] = 0;
        while(!q.empty()){
            int cur = q.front(); q.pop();
            inq[cur] = 0;
            for(int i = head[cur]; i; i = edge[i].nxt){
                if(dep[edge[i].to] > dep[cur] + 1 && edge[i].flow){
                    dep[edge[i].to] = dep[cur] + 1;
                    if(!inq[edge[i].to]){
                        q.push(edge[i].to);
                        inq[edge[i].to] = 1;
                    }
                }
            }
        }
        if(dep[t] != 1e9)   return 1;
        return 0;
    }
    LL dfs(int x, LL minFlow){
        LL flow = 0;
        if(x == t)  return minFlow;
        for(int i = curArc[x]; i; i = edge[i].nxt){
            curArc[x] = i;
            if(dep[edge[i].to] == dep[x] + 1 && edge[i].flow){
                flow = dfs(edge[i].to, min(minFlow, edge[i].flow));
```

```
                if(flow){
                    edge[i].flow -= flow;
                    edge[i^1].flow += flow;
                    return flow;
                }
            }
        }
        return 0;
    }
    LL Dinic(){
        LL maxFlow = 0, flow = 0;
        while(bfs()){
            while(flow = dfs(s, INF))
                maxFlow += flow;
        }
        return maxFlow;
    }

    void init(){
        edgeNum = 1;
        for(int i = 1; i <= n; i++){
            head[i] = 0;
        }
    }
}
```

ISAP

预流推进 Push-Relable