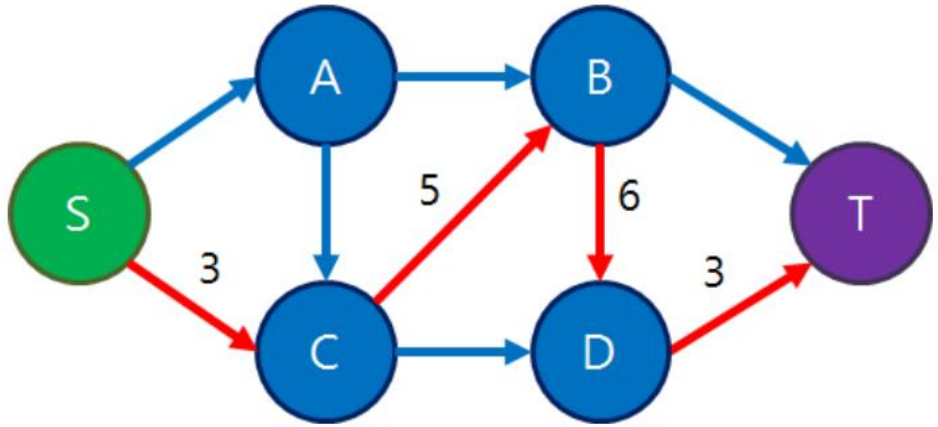
MCMF

minimum cost maximum flow

땅울림 알고리즘 중급반

MCMF란?

- 네트워크 플로우 그래프에서 간선에 가중치가 존재
- 간선 가중치(비용)의 합을 최소화 하며 최대의 유량을 S->T로 보냄
- 간선 비용이 d일 때 f만큼의 유량을 흐르면 총 비용은 d*f



네트워크 플로우 복습

- 경로 탐색(BFS)-애드몬드 카프 알고리즘
- 해당 경로로 유량이 얼만큼 흐를 수 있는가?
- 유량 흘려주기

MCMF

- 매번 최소비용의 경로를 찾아 유량을 흘려줌
- 경로탐색시 BFS가 아닌 가중치에 따른 최단경로 알고리즘 수행!
- 가장 비용이 작은 경로부터 순차적으로 유량을 흘려 주기 때문에 최소비용으로 최대유량을 보낼 수 있음
- 다익스트라?

SPFA(shortest path faster algorithm)

- 최단경로 알고리즘(벨만포드 업그레이드 버전)
- 음의가중치가 존재하여도 최단경로 탐색 가능

SPFA(shortest path faster algorithm)

- 모든 거리를 무한대로 초기화
- 큐를 사용하여 시작점부터 탐색(BFS와 유사)
- 큐에 현재 들어가 있는 정점은 push하지 않음
- 한 정점을 n번 방문하면 음의 사이클 발생으로 간주

* 시간복잡도 : O(V * E)이지만 평균적으로 O(V + E)수준으로 동작

11657_타임머신

- SPFA로 풀어보자!
- 음의 사이클이 발생하면 -1 출력
- 1번에서 출발 2~N번도시까지 가는 최단거리 출력

11657_타임머신

```
int n, m; // 정점, 간선 개수
ll dist[501]; // 해당 정점까지의 최단 거리를 저장
ll cycle[501]; // 정점을 몇번 방문했는지 저장
bool inQ[501]; // 해당 정점이 현재 큐에 들어있으면 True 아니면 False
vector<pair<int, int> > adj[501]; // 그래프
queue<int> q; // SPFA를 위한 큐
```

```
// 입력받고 그래프 생성
cin >> n >> m;
for (int i = 0; i < m; i++) {
    int frm, to, cst;
    cin >> frm >> to >> cst;
    adj[frm].push_back({ cst, to });
}
```

11657_타임머신

```cpp
// SPFA
for (int i = 1; i <= n; i++) // 전체 거리 무한대로 초기화
    dist[i] = inf;
// 1번부터 시작
dist[1] = 0;
q.push(1);
inQ[1] = true;
cycle[1]++;

while (!q.empty()) {
    int cur = q.front();
    q.pop();
    inQ[cur] = false; // 큐에서 꺼냈으므로 false
    for (int i = 0; i < adj[cur].size(); i++) {
        int nxt = adj[cur][i].second; // 다음 정점
        int nCst = adj[cur][i].first; // 다음 정점으로 가는 비용
        if (dist[nxt] > dist[cur] + nCst) { // 현재 정점에서 가는 비용이 더 적을 경우
            dist[nxt] = dist[cur] + nCst; // 다음 정점의 dist값 업데이트
            if (!inQ[nxt]) { // 큐안에 들어있지 않을 경우에만
                q.push(nxt); // 큐에 푸시
                inQ[nxt] = true;
                cycle[nxt]++;
                if (cycle[nxt] >= n) { // 음의 사이클 발생
                    cout << -1 << '\n';
                    return 0;
                }
            }
        }
    }
}
```

11657_타임머신

```
// 출력
for (int i = 2; i <= n; i++) {
    if (dist[i] == inf)
        cout << -1 << '\n';
    else
        cout << dist[i] << '\n';
}

return 0;
```

11408_열혈강호 5

- 직원N명 일M개
- Source -> 직원 -> 일 -> Sink의 mcmf구하기
- 간선의 가중치 : salary
- 간선의 용량 : 1

11408_열혈강호 5

```
int n, m; // 직원, 일 수
int total_cnt, total_sal; // 구하고 싶은 값
int S = 0, T = 801; // source : 0, sink : 801, 직원 : 1~400, 일 : 401~800
vector<pair<int, int> > adj[802]; // 그래프
int c[802][802], f[802][802], frm[802]; // 용량, 유량, 경로
int cycle[802], dist[802]; // 방문 횟수, 최소 거리
bool inQ[802]; // Q에 들어가 있는가
```

11408_열혈강호 5 [그래프 생성]

```
cin >> n >> m;

for (int i = 1; i <= n; i++) {
    int cnt;
    cin >> cnt;
    for (int j = 0; j < cnt; j++) {
        int num, sal;
        cin >> num >> sal;
        adj[i].push_back({ sal, num + 400 }); // [i번 직원]과 [일의 번호 + 400]을 연결
        adj[num + 400].push_back({ -sal, i }); // 반대 방향도 연결
        c[i][num + 400] = 1; // 용량은 1
    }
}
// Source와 직원을 연결
for (int i = 1; i <= n; i++) {
    adj[S].push_back({ 0, i });
    adj[i].push_back({ 0, S });
    c[S][i] = 1;
}
// 일과 Sink를 연결
for (int i = 1; i <= m; i++) {
    int w_num = i + 400;
    adj[w_num].push_back({ 0, T });
    adj[T].push_back({ 0, w_num });
    c[w_num][T] = 1;
}
```

- 반대방향은 가중치*-1

11408_열혈강호 5

```
// 최대유량 알고리즘
while (true) {
    spfa(); // 경로를 구함
    if (frm[T] == -1) // 경로가 없으면 break
        break;
    int flow = inf; // 현재 경로상에서 흐를 수 있는 유량은?
    for (int i = T; i != S; i = frm[i]) {
        flow = min(flow, c[frm[i]][i] - f[frm[i]][i]);
    }
    for (int i = T; i != S; i = frm[i]) {
        // 유량 흘려주기
        f[frm[i]][i] += flow;
        f[i][frm[i]] -= flow;
    }
    // 유량이 얼만큼 흘렀나
    total_cnt += flow;
    // 비용은 얼마나 발생했나? dist[T]는 0번부터 T번까지의 최단거리!
    total_sal += dist[T];
}
cout << total_cnt << '\n' << total_sal << '\n';
return 0;
```

11408_열혈강호 5_spfa

```
void spfa() {
    // 초기화
    memset(frm, -1, sizeof(frm));
    memset(inQ, false, sizeof(inQ));
    memset(cycle, 0, sizeof(cycle));
    for (int i = 1; i <= T; i++)
        dist[i] = inf;

    // Source부터 시작
    dist[S] = 0;
    queue<int> q;
    q.push(S);
    inQ[S] = true;
    cycle[S]++;
    while (!q.empty()) {
        int cur = q.front();
        q.pop();
        inQ[cur] = false;
        for (int i = 0; i < adj[cur].size(); i++) {
            int nxt = adj[cur][i].second;
            int ncst = adj[cur][i].first;

            if (c[cur][nxt] > f[cur][nxt] && dist[nxt] > dist[cur] + ncst) {
                dist[nxt] = dist[cur] + ncst;
                frm[nxt] = cur; // 경로 저장
                if (!inQ[nxt]) {
                    q.push(nxt);
                    inQ[nxt] = true;
                    cycle[nxt]++;
                    if (cycle[nxt] >= n) {
                        return;
                    }
                }
            }
        }
    }
}
```

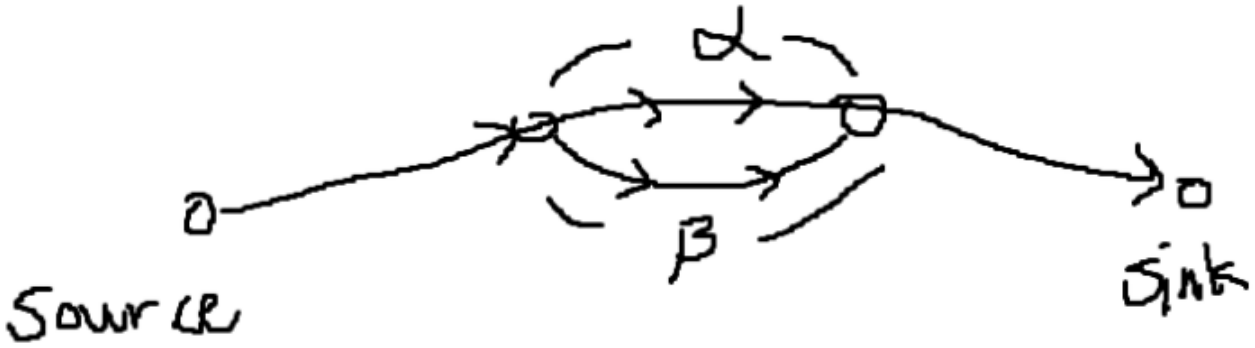
11408_열혈강호 5

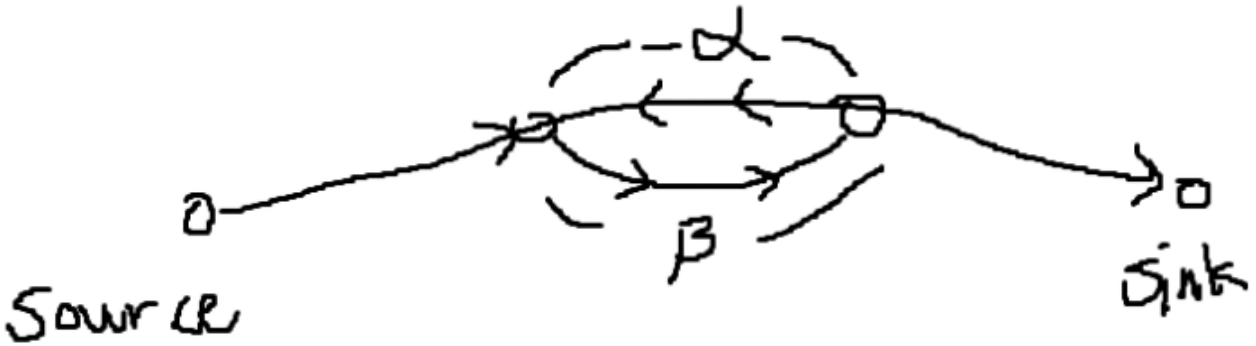
```
void spfa() {
    memset(frm, -1, sizeof(frm));
    memset(inQ, false, sizeof(inQ));
    //memset(cycle, 0, sizeof(cycle));
    for (int i = 1; i <= T; i++)
        dist[i] = inf;
    dist[S] = 0;
    queue<int> q;
    q.push(S);
    inQ[S] = true;
    //cycle[S]++;
    while (!q.empty()) {
        int cur = q.front();
        q.pop();
        inQ[cur] = false;
        for (int i = 0; i < adj[cur].size(); i++) {
            int nxt = adj[cur][i].second;
            int ncst = adj[cur][i].first;

            if (c[cur][nxt] > f[cur][nxt] && dist[nxt] > dist[cur] + ncst) {
                dist[nxt] = dist[cur] + ncst;
                frm[nxt] = cur;
                if (!inQ[nxt]) {
                    q.push(nxt);
                    inQ[nxt] = true;
                    /*cycle[nxt]++;
                    if (cycle[nxt] >= n) {
                        return;
                    }*/
                }
            }
        }
    }
}
```

- 음의 사이클은 발생하지 않음
- 초기에 음의사이클이 존재하지 않으면 음의 유량에 의해서도 음의 사이클이 생기지 않음

11408_열혈강호 5





여기에서 α가 최단경로에 포함 되었다고 하면,
β≥α가 성립해야 합니다.

그리고 나서 가중치와 방향을 모두 뒤집으면
아래처럼 -α로 바뀝니다.

만약 이후에 음수 사이클이 생기면
-α+β<0 즉 β<α가 되어 모순이 생깁니다.

```
                    if (cycle[nxt] >= n) {
                        return;
                    }*/
                }
            }
        }
    }
}
```