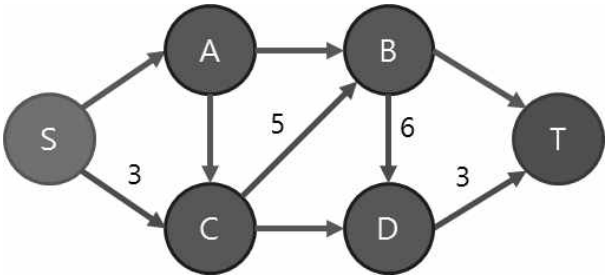
최소 비용 최대 유량(Minimum Cost Maximum Flow) (수정: 2019-08-14)

대회알고리즘
2016. 9. 12. 2:50

https://blog.naver.com/kks227/220810623254

개강하고서 너무 피곤하네요. 하지만 제가 정말정말 쓰고 싶던 글이니까 빨리 씁시다.
이번에 다룰 내용은 **MCMF**(min-cost max-flow)입니다.
이젠 간선에 용량뿐만 아니라 또다른 가중치, 비용(cost)이 존재합니다.
최대 유량을 흘리긴 흘리는데, 최소한의 비용을 들여서 흘리라는 겁니다.

비용은 간선마다 다르며, 어떤 간선의 비용이 d이고 해당 간선에 순방향으로 f만큼의 유량이 흐르고 있을 때 소비되는 비용은 d*f입니다.
전체 비용은 그래프 모든 간선의 비용의 합입니다.



이런 그래프에서 빨간색 간선으로 표현된 경로로 유량 1이 흐르고 있을 때,
해당 경로의 비용은 3+5+6+3 = 17이 됩니다.
이제 유량을 최대한으로 흘리되, 이 비용 합이 최소가 되게 하는 문제를 풀어야 합니다.

방법 자체는 생각보다는 간단합니다.
포드 풀커슨, 에드몬드 카프 알고리즘과 비슷하게 매번 경로를 찾아 유량을 흘리고, 더 이상의 경로가 없을 때까지 반복하는 건 마찬가지인데,
이때 매번 찾는 경로가 **최단 경로**입니다. 최단 경로라 함은, 소스에서 싱크로 가는 최소 비용의 거리.
즉, 간선에 있던 비용이 거리값이 되어서, 이 비용 값들을 기준으로 최단 경로를 찾는 겁니다.
매번 최단 경로를 찾고, 이 경로로 흐를 수 있는 최대 유량을 찾은 후 흘려보내면서, 결과(비용 합)에는 이번에 찾은 경로의 비용 합 * 유량을 더해주면 되겠습니다.

그런데 이 최단경로를 찾는 알고리즘은 무엇이어도 좋지만 최소한 **음의 가중치**가 있는 그래프에서도 잘 작동해야 합니다.
최대 유량 알고리즘에서 이미 유량이 흐르는 간선 (u, v)에 역방향 간선 (v, u)가 존재한다고 보고 그것도 경로에 포함시켰는데,
역방향 간선으로 유량을 흘리면 유량이 상쇄되어서 순방향으로 흐르던 유량이 사라지죠?
유량이 사라질 때는 그만큼 흐르던 비용 역시 차감되어야 합니다.
이것을 이렇게 구현할 수 있습니다. 순방향 간선의 비용이 d일 때, 역방향 간선의 비용을 -d로 주는 겁니다.
그렇다면 역방향 간선으로 유량을 흘려보내며 있던 유량을 상쇄시킬 때, 유량 1당 d씩의 비용도 다시 차감하여 없애버릴 수 있습니다.
또한 최단 경로 알고리즘도 이 역방향 간선의 음의 비용까지 잘 사용하여 동작할 겁니다.

따라서 최소한 벨만 포드 알고리즘 정도는 되어야 최단 경로를 제대로 찾을 수 있는데,
조금 변형시켜서 실행시간의 기댓값을 낮춘 **SPFA**(shortest path faster algorithm)라는 최단 경로 알고리즘이 존재하고, 지금 MCMF 문제를 풀 때 가장 많이 사용되는 알고리즘입니다.
https://en.wikipedia.org/wiki/Shortest_Path_Faster_Algorithm

```
procedure Shortest-Path-Faster-Algorithm(G, s)
 1    for each vertex v ≠ s in V(G)
 2        d(v) := ∞
 3    d(s) := 0
 4    offer s into Q
 5    while Q is not empty
 6        u := poll Q
 7        for each edge (u, v) in E(G)
 8            if d(u) + w(u, v) < d(v) then
 9                d(v) := d(u) + w(u, v)
10                if v is not in Q then
11                    offer v into Q
```

위키에 나와 있는 스도 코드를 보면 다익스트라를 절반쯤 섞어 놓은 것 같이 생겼는데, 우선순위 큐 대신 그냥 큐를 사용합니다.
읽어보시면 정말 쉽습니다. 저게 끝이고, 저걸 코드로 옮기는 것도 그리 어렵지 않습니다.

그나마 가장 난해한 부분이 정점 v가 큐 안에 있는지를 판단하는 조건문인데,
이건 inQueue 등의 bool 타입 배열을 마련해 조작하면 아주 쉽게 판단 가능합니다.

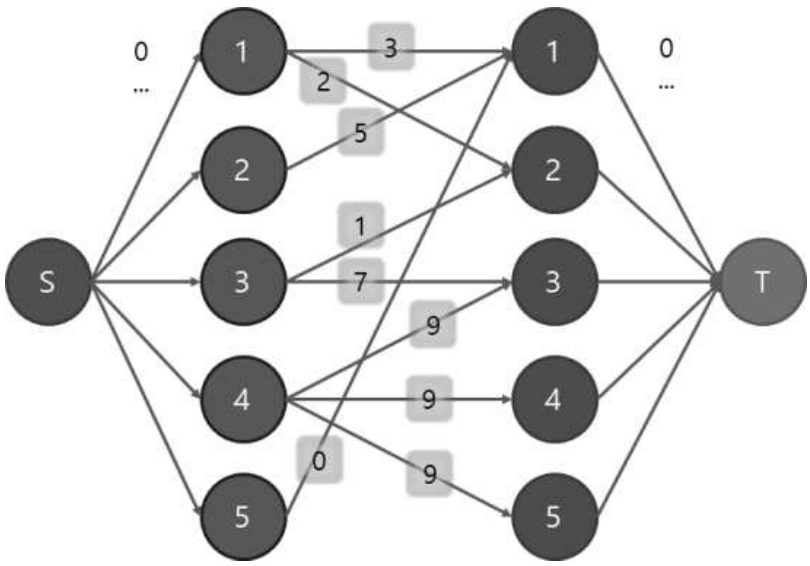
어쨌거나 기본은 벨만 포드 알고리즘이라 시간복잡도의 상한은 여전히 O(VE)이고 이걸 사용한 MCMF 코드의 시간복잡도도 O(VEf)지만,
아직까지는 MCMF라는 것만 추론해 내면 시간복잡도 때문에 못 풀 정도의 문제들은 거의 없고, 실제로 이 알고리즘도 저 시간복잡도로 보이는 기댓값보다는 빠르게 동작한다고 합니다.

https://www.acmicpc.net/problem/11408

BAEKJOON ONLINE JUDGE

11408번: 열혈강호 5
www.acmicpc.net

이 문제가 아주 대표적인 MCMF 문제입니다.
각 직원과 하는 일 쌍마다 서로 다른 비용값, 월급이 주어집니다.
이전 시리즈처럼 여전히 일은 최대한 많이 처리해야 합니다. 그때 월급을 되도록 최소한 지불하게 하라네요.
이분 매칭으로 모델링하여 문제를 풀 때 각 직원과 일을 매칭시켰고, 이 매칭에 비용이 들어간다고 보면 됩니다.



이분 매칭 그래프로 모델링해 보면 이와 같습니다.
이분 매칭 그래프 형태이니까 용량은 생략하고, 각 간선의 비용만 써보면 이렇게 됩니다.
소스나 싱크와 인접한 간선들은 모두 비용이 0입니다.

그런데 이 문제에서는 매번 찾는 경로마다 유량을 1씩만 흘리니까,

https://www.acmicpc.net/problem/11405

BAEKJOON ONLINE JUDGE

11405번: 책 구매하기
www.acmicpc.net

이 문제를 먼저 풀어봅시다.
이 문제에서는 최대 유량이 반드시 모든 사람의 A 값의 합이고, 항상 그만큼 흘리는 게 가능하다고 합니다.
j번 서점이 Bj개의 책을 갖고 시작하므로 소스에서 j번 서점 정점에 Bj만큼의 용량을 가진 간선을 연결하고,
i번 사람은 Ai개의 책을 주문하고 싶어하므로 i번 사람 정점에서 Ai만큼의 용량을 가진 간선을 싱크로 연결시키면 됩니다.
또한 i번 서점에서 j번 사람에게 책을 보내는 간선은 용량은 ∞로 두어도 좋지만, 대신 비용을 Cij로 설정해 주어야 합니다.
그 외의 모든 간선은 비용이 0입니다.

이 문제의 경우는 총 유량은 셀 필요가 없고(정해져 있으니까), 증가경로가 더 없을 때까지 루프를 반복하다가 나오면 됩니다.
만약 총 유량도 구해야 한다면, 그냥 포드 풀커슨 알고리즘 때처럼 구하면 됩니다.

cs

```cs
#include <cstdio>
#include <vector>
#include <queue>
#include <algorithm>
using namespace std;
const int MAX_N = 100; // 최대 N, M
const int MAX_V = 2*(MAX_N+1); // 최대 정점 개수
const int S = MAX_V-2; // 소스 정점 번호
const int E = MAX_V-1; // 싱크 정점 번호
const int INF = 1000000000;

int main(){
    // 정점 번호: 0~MAX_N: 서점, MAX_N~MAX_N*2: 사람
    int N, M;
    int c[MAX_V][MAX_V] = {0}; // 각 간선의 용량
    int d[MAX_V][MAX_V] = {0}; // 각 간선의 비용
    int f[MAX_V][MAX_V] = {0}; // 각 간선에 흐르는 중인 유량
    vector<int> adj[MAX_V]; // 각 정점의 인접 리스트

    scanf("%d %d", &N, &M);
    // 각 사람 정점과 싱크 정점 사이 간선 추가 (비용 0)
    for(int i=MAX_N; i<MAX_N+N; i++){
        scanf("%d", &c[i][E]);
        adj[E].push_back(i);
        adj[i].push_back(E);
    }
    // 소스 정점과 각 서점 정점 사이 간선 추가 (비용 0)
    for(int i=0; i<M; i++){
        scanf("%d", &c[S][i]);
        adj[S].push_back(i);
        adj[i].push_back(S);
    }
    // 서점과 사람 사이 간선 추가 (비용 C_ij)
```

```
34      for(int i=0; i<M; i++){
35          for(int j=MAX_N; j<MAX_N+N; j++){
36              scanf("%d", &d[i][j]);
37              d[j][i] = -d[i][j]; // 역방향 간선의 비용: 순방향의 -1배
38              c[i][j] = INF; // 순방향 간선만 용량이 1 이상
39              adj[i].push_back(j);
40              adj[j].push_back(i);
41          }
42      }
43
44      int result = 0; // 최소 비용
45      // MCMF 시작
46      while(1){
47          int prev[MAX_V], dist[MAX_V];
48          bool inQ[MAX_V] = {0}; // 해당 정점이 큐 안에 있는가?
49          queue<int> Q;
50          fill(prev, prev+MAX_V, -1);
51          fill(dist, dist+MAX_V, INF);
52          dist[S] = 0;
53          inQ[S] = true;
54          Q.push(S);
55
56          while(!Q.empty()){
57              int curr = Q.front();
58              Q.pop();
59              inQ[curr] = false; // 큐에서 꺼냄
60              for(int next: adj[curr]){
61                  // 최단 경로를 찾는 중이지만, 여전히 여유 용량 있어야 함
62                  if(c[curr][next]-f[curr][next] > 0 && dist[next] > dist[curr]+d[curr][next]){
63                      dist[next] = dist[curr] + d[curr][next];
64                      prev[next] = curr;
65                      // 큐에 들어있지 않다면 큐에 넣음
66                      if(!inQ[next]){
67                          Q.push(next);
68                          inQ[next] = true;
69                      }
70                  }
71              }
72          }
73          // 더 이상 경로가 없다면 루프 탈출
74          if(prev[E] == -1) break;
75
76          // 경로상에서 가장 residual이 작은 간선을 찾아 최대 흘릴 수 있는 flow 찾음
77          int flow = INF;
78          for(int i=E; i!=S; i=prev[i])
79              flow = min(flow, c[prev[i]][i] - f[prev[i]][i]);
80
81          // 경로상의 모든 간선에 flow만큼의 유량을 흘림
82          for(int i=E; i!=S; i=prev[i]){
83              result += flow * d[prev[i]][i]; // 총 비용이 각 간선 비용만큼 증가
84              f[prev[i]][i] += flow;
85              f[i][prev[i]] -= flow;
86          }
87      }
88      // 정답 출력
89      printf("%d\n", result);
90  }
```

Colored by Color Scripter

코드가 길지만, c, f 배열 말고로 새로운 값인 d 배열(비용)이 추가되었다는 점과,
안에서 경로를 BFS로 찾지 않고 SPFA로 찾는다는 것, 추가적으로 비용 값을 구하고 있다는 점만 다릅니다.
47~72번 줄은 SPFA 알고리즘 스도 코드를 그대로 옮겼을 뿐입니다.
그러나 추가된 내용이 있는데, 사용할 수 있는 간선은 여유 용량(residual)이 0보다 커야 하므로, 그 조건을 체크합니다. 이 부분은 원래의 유량 알고리즘에서도 체크하던 부분.
또한 37번 줄에서 역방향의 가상 간선에는 음의 비용을 매겨주는 것도 중요합니다.
83번 줄에서는 이번에 찾은 경로의 모든 간선 비용과 이번에 새로 흘려보낸 유량 값을 곱해서 결과에 더해 줍니다.
물론, 더하고 있지만 감소할 수도 있죠. 유량을 상쇄시킬 때는.

이제 열혈강호 5 문제도 위 코드를 조금만 손보면 바로 풀 수 있습니다.

```cs
        for(int i=E; i!=S; i=prev[i]){
            cost += d[prev[i]][i];
            f[prev[i]][i]++;
            f[i][prev[i]]--;
        }
        total++;
```

MCMF 루프의 맨 마지막 부분, 76~86번 줄을 이렇게 바꾸면 됩니다.
아까도 언급했듯이 이 문제는 매번 찾는 경로마다 유량을 1만 흘릴 수 있기 때문에, 이미 flow가 1이라는 걸 알고 있기 때문.
이때 total은 총 유량 변수입니다. total을 증가시키는 걸 빼먹지 말도록 합시다.

토막상식인데, 최단 경로는 음의 싸이클이 없어야만 유한한 값으로 존재합니다.
그런데 원래 그래프에서 음의 싸이클이 없었어도, 유량을 흘려보내면서 음의 역방향 간선들이 드러나며 없던 음의 싸이클이 생성될 수도 있지 않을까요?

그러나 그런 경우는 없다고 합니다. https://www.acmicpc.net/board/view/1291
위 글에 들어가보시면 중간에 august14님이 아주 간결하게 그 이유를 설명해주셨습니다.
성지 순례 한번씩 하고 갑시다.

https://www.acmicpc.net/problem/11409

BAEKJOON
ONLINE JUDGE

11409번: 열혈강호 6
www.acmicpc.net

이번 문제는 이상합니다. 최소 비용이 아니라 최대 비용을 구하라네요.
그러나 이 문제, 생각보다 굉장히 쉽게 풀 수 있습니다.
모든 간선의 가중치에 -1을 곱한 뒤에 길이가 d인 최단 경로를 찾으면,
그건 원래의 그래프에서는 길이가 -d인 **최장 단순 경로**가 됩니다!!

```cs
            d[i][work] = -pay;
            d[work][i] = pay;
```

따라서 열혈강호 5 문제에서 d 배열의 값만 서로 부호를 바꾸어서 설정해주면 바로 문제가 풀립니다.
따라서 알고리즘을 잘 활용하면, 최소 비용뿐 아니라 최대 비용 문제도 풀 수 있게 됩니다.

https://www.acmicpc.net/problem/3640