CS300: Homework #5

Due on November 29, 2016 at 10:30am

Prof. Sunghee Choi

20160051 Ohjun Kwon

Problem 1

수업시간에 제시되었던 **BFS** 코드를 보자. 이러한 **BFS** 코드는 $Adj[i]$를 보면 알 수 있듯이 인접리스트를 이용하여

```
function BFS(G, s)
    Enqueue(Q, s)
    while Q ≠ ∅ do
        u ← Dequeue(Q)
        for each v ∈ Adj[u] do
            if d[v] = ∞ then
                d[v] ← d[u] + 1
                Enqueue(Q, v)
            end if
        end for
    end while
end function
```

그래프의 간선에 대한 정보를 알아내고 있다. 하지만, 우리의 경우에는 인접행렬을 이용하여 알고리즘을 다루고 싶은 경우이다. 이도 **BFS**와 비슷한 형식으로 작성하면 된다. $Adj[u]$가 들어가있는 부분만 바꾸어주면 된다. 쉽게 생각하면 인접리스트를 이용한 **BFS**의 시간복잡도인 $O(V+E)$에서 간선을 찾는데 걸리는 시간이 E가 아니라 V^2임에서 착안하여 $O(V^2)$이 됨을 대략적으로 알 수 있지만, 직접적으로 알고리즘을 작성하여 시간복잡도가 어떻게 되는지 알아보자. **BFS'**의 의사코드를 보면 $Adj[u]$가 있을 부분에 인접행렬의 한 행을 전부 검사하여

```
function BFS'(G, s)
    Enqueue(Q, s)
    while Q ≠ ∅ do
        u ← Dequeue(Q)
        for i ← 1 to |V| do
            if G[u, i] = 1 then
                if d[i] = ∞ then
                    d[i] ← d[u] + 1
                    Enqueue(Q, i)
                end if
            end if
        end for
    end while
end function
```

간선으로 연결되어있는 정점을 찾는 코드가 보인다. 한 정점에 대하여 총 $|V|$개의 정점에 대하여 검사를 하게 되므로 $O(V)$이고, 이를 각 정점에 대하여 모두 한 번씩은 연산하기 때문에 총 시간복잡도는 $O(V^2)$이 된다.

Problem 2

우리가 보이고 싶은 것은 그래프의 모든 cut에 대해 교차하는 유일한 light edge를 가질 때, 그 그래프는 유일한 최소신장트리를 가짐을 보이는 것이다. 이를 증명하기 위해 귀류법을 이용하자. 먼저 서로 다른 두 개의 MST M, S가 있다고 가정하고, 사실은 이 둘이 같음을 보이면 증명이 완료된다. 두 트리가 같다는 것은 M에 속한 임의의 간선이 S에 속하고 S에 속한 임의의 간선이 M에 속함을 보여 트리에 포함된 모든 간선이 같음을 보이는 방법으로 할 수 있다.
먼저, M에 속한 임의의 간선 (u,v)를 생각하자. 이 간선을 지운다면 최소신장트리인 M은 두 부분으로 **절단**될 것이다. 이 절단을 $(A, V-A)$라고 하고(이 절단에서 교차하는 간선은 (u,v)로 유일하므로 (u,v)는 M의 절단

Problem 2 continued on next page. . .

$(A, V-A)$에 교차하는 light edge이다), 동일한 정점 V로 이루어진 최소신장트리 S에서 이 절단에 대해 생각해보자. 트리를 $(A, V-A)$와 같이 절단할 때 교차하는 light edge는 가정에 의해 유일하기 때문에 간선 (u, v)는 트리 S에도 포함된다. 반대방향으로도 동일한 방법으로 (M과 S의 위치를 바꾸면) 증명될 수 있다. $\therefore$ M과 S는 동일한 트리이다.
이 명제의 역은 "유일한 최소신장트리를 가지는 그래프이면, 그래프의 모든 cut에 대해 교차하는 유일한 light edge를 가진다"이다. 이 명제에 대한 반례는 아래 그림 1과 같다. 그림 1 (a)와 같은 그래프가 반례가 된다.

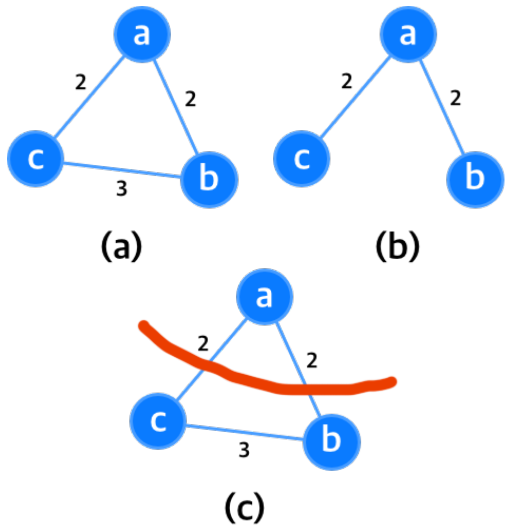

Figure 1: 반례

그림 1 (a)의 최소신장트리는 그림 1 (b)로 유일하다. 하지만 그림 1 (a)의 절단 중 하나인 그림 1 (c)와 교차하는 light edge는 (a, b), (a, c)로 유일하지 않다.

Problem 3

원하는 식은 곱으로 표현되어 있기 때문에 우리가 가지고 있는 기술인 그래프 상에서의 최단 경로를 구하는 방법을 사용하지 못한다. 주어진 식 1을 덧셈 꼴로 변형하기 위해서 양변에 로그를 취해주자.

$$R[i_1, i_2] \cdot R[i_2, i_3] \cdot \ldots R[i_{k-1}, i_k] \cdot R[i_k, i_1] > 1 \tag{1}$$
$$\lg \{R[i_1, i_2] \cdot R[i_2, i_3] \ldots R[i_{k-1}, i_k] \cdot R[i_k, i_1]\} > \lg 1$$
$$\lg R[i_1, i_2] + \lg R[i_2, i_3] + \cdots + \lg R[i_k, i_1] > 0 \tag{2}$$

식 2가 얻어졌는데, 여기서 양변에 -1을 곱해주면 식 3과 같은 꼴이 만들어진다. 각 통화가 정점을 의미하고 환율이 간선을 의미하는 방향 그래프를 생각해보자. 각 $-\lg R[i_a, i_b]$을 두 노드 a와 b를 잇는 간선의 가중치라고 하면 부등식 1을 변형하여 얻은 동치인 식 3은 그래프에서 음의 가중치를 갖는 순환을 의미하게 되고, 주어진 문제는 그래프에서 음의 가중치를 갖는 순환을 찾는 문제로 환원된다.

$$-\lg R[i_1, i_2] + -\lg R[i_2, i_3] + \cdots + -\lg R[i_k, i_1] < 0 \tag{3}$$

벨만-포드 알고리즘을 이용하여 방향 그래프에서 음의 가중치를 갖는 순환을 가지는지 여부를 찾는 알고리즘을 우리는 수업시간에 작성하였으므로 자세한 설명은 생략하고, 벨만-포드 알고리즘을 이용하여 그래프가 음의 가중치를 갖는 순환을 갖는지 여부만 판별하는 것이 아니라 그 순환을 출력해야 하므로 수업시간에 작성했던 벨만-포드 알고리즘에 약간의 수정을 가하도록 하자. (python 2로 작성된 작동하는 알고리즘(prob3.py)은 별도의 페이지로 첨부되어있다. cf) 0-based index로 짜여진 소스이다.)
가장 먼저 **bellford**는 정상적인 벨만-포드 알고리즘의 알고리즘에서 음의 가중치를 갖는 순환 검출 부분에 삼각 부등식이 성립하지 않은 곳을 저장하도록 하였다. 모든 relaxation step이 끝난 이후에도 d 값이 변하는 이유는

Problem 3 continued on next page...

바로 음의 가중치를 갖는 순환이 있기 때문에 d 값이 계속 작아질 수 있는 것이다. 이렇게 확인된 정점에서 바로 직전에 방문했던 정점인 p를 따라가면 순환을 검출할 수 있게 된다. 하지만, 이 순서대로 출력하면 간선의 방향의 반대방향으로 출력되게 되므로 스택에 저장 후 하나씩 pop하면서 출력하면 음의 가중치를 갖는 경로를 검출할 수 있게 된다.
하지만, 벨만-포드 알고리즘은 전체를 탐색하는 것이 아니고 하나의 source에서 나아가는 경로를 탐색하는 알고리즘이므로 모든 정점에 대해 각각 음의 가중치를 갖는 경로를 찾을 수 있는지 검사하여야 한다. **bellford**의 시간복잡도는 정상적인 원래의 벨만-포드 알고리즘 뒤에 경로를 추적하는 루틴을 넣었으므로 $O(V^3 + V) = O(V^3)$인데 이를 각 정점마다 실행하게 되므로 **findCycle**의 시간복잡도는 총 $O(V^4)$가 된다.
bellford를 모든 정점에 대해서 실행시켜야 하는 이유는 그림 2 같은 그래프에서 0이나 1을 source로 선택하는 경우 모든 정점과의 d가 ∞이기 때문에 경로를 찾을 수 없기 때문이다. 약간의 편법을 이용하면 모든 정점에 대하여 확인해보지 않아도 알 수 있는데, ∞의 값 대신 매우 큰 자연수 값을 주게 되면 ∞의 역할도 하는 동시에 d 값이 감소할 수 있어 아무 정점이나 source로 선택하여 **bellford**를 실행하면 음의 가중치를 갖는 순환을 찾을 수 있다. 이렇게 되면 $O(V^3)$만에 찾을 수 있게 되는 것이다.

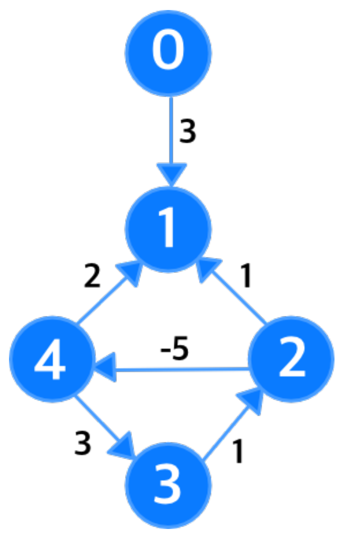

Figure 2: 예시

```python
inf = float('inf')


def findCycle(R):
    n = len(R)
    for i in range(n):
        if bellford(i, R):
            return
    print "No cycle"


def bellford(source, R):
    n = len(R)
    d = []
    p = []
    for i in range(n):
        d.append(inf)
        p.append(None)
    d[source] = 0

    for k in range(n):
        for i in range(n):
            for j in range(n):
                if d[j] > d[i] + R[i][j]:
                    d[j] = d[i] + R[i][j]
                    p[j] = i

    cyc = -1
    for i in range(n):
        for j in range(n):
            if d[j] > d[i] + R[i][j]:
                cyc = j

    if cyc == -1:
        return False
    else:
        ini = cyc
        path = []
        for i in range(n):
            path.append(cyc)
            cyc = p[cyc]
            if cyc == ini:
                break
        for i in reversed(range(len(path))):
            print path[i],
        return True

G = [[inf, inf, inf, inf],
     [2, inf, inf, -2],
     [2, -2, inf, inf],
     [inf, inf, 2, inf]]

findCycle(G)

```