# CS300: Homework #4

Due on November 15, 2016 at 10:30am

*Prof. Sunghee Choi*

**20160051 Ohjun Kwon**

# Problem 1

**(a)** $x_i$가 가질 수 있는 $h$개의 값의 후보를 $a_1, a_2, \ldots, a_h$라고 하고, 이를 모은 집합을 $A = \{a_1, a_2, \ldots, a_h\}$라고 하고,

$$\sum_{i=1}^{n} m_i(x_i) + \sum_{i=2}^{n} \phi(x_{i-1}, x_i) \tag{1}$$

식 1을 최소로 만드는 $(x_1, x_2, \ldots, x_n)$을 $L(n)$이라고 하면, 우리가 구하고자 하는 정보를 $L(n)$이라고 나타낼 수 있을 것이다. 그렇다면 $L(n)$은 $L(n-1)$에 $f(x)$의 값을 최소로 만드는 하나의 $x_n \in A$를 찾아 $L(n-1)$에 연장시켜주는 방식으로 생각하면 될 것이다. 하지만, 이 방법에서는 $L(n)$이 $L(n-1)$에 하나의 숫자를 끝에 붙인 형태가 아닐 수도 있어 약간의 결점이 있다. 예를 들면, 우리가 구한 $L(n-1)$에서 $x_{n-1}$이 $q$라고 하면 이 $q$와 마지막 $x_n$의 관계($\phi$) 중에 가장 작은 수를 $A$에서 찾게 될 것인데, $\phi(q, r)$ $(\forall r \in A)$의 값이 모두 커 다른 수열을 통해서 $x_n$까지 도달하는 방향으로 찾는 것이 더 작은 $f(x)$를 찾게 될 수도 있다. 그런 상황에 대비하기 위해서 우리는 $L(n)$을 찾을 때 다른 경로를 통해서 찾을 수 있도록 잡아야 할 것이다. 식 1을 최소로 만드는 $(x_1, x_2, \ldots, x_{n-1}, x_n = k)$를 $L(n, k)$라고 하자. (단, $k \in A$). 이렇게 설정하면 최후에 $L(n)$은 $f(L(n, k))$를 최소화 하는 $L(n, k)$가 될 것이다. 이제 각 값을 찾는 방법에 대해서 설명하자면, $L(n, k)$를 찾기 위해서 전 단계의 결과를 참조하도록 하자.

$$\min_{q \in A} \{m_{n-1}(k) + \phi(q, k)\} \tag{2}$$

식 2을 만족하는 $q$를 찾으면 $L(n, k)$는 $L(n-1, q)$의 맨 끝에 $k$를 붙인 것이 된다. 이것을 $h$번 하면 $L(n)$을 구할

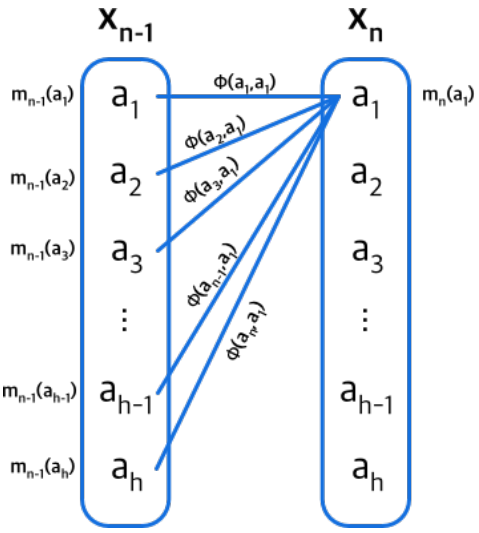


Figure 1: $L(n, a_1)$을 찾는 과정

수 있게 된다. 이는 $O(h^2)$의 시간이 걸리게 되고, 이를 $L(1)$일 때부터 $L(n)$일 때까지 계산은 총 $n$번 하므로 총 $O(nh^2)$의 시간이 걸리게 된다. 이를 의사코드로 나타내면 아래와 같다.
알고리즘 **L**의 시간복잡도를 줄 별로 계산해보면, 이중 for문내의 문장들은 $O(nh)$번 실행되게 될 것인데, 10번째 줄에서 최소를 찾을 때 $O(h)$의 시간이 걸린다. 그러므로 이중 for문을 탈출하기까지는 $O(nh^2)$의 시간이 걸리고, $L(n, k)$에서 $f(x)$의 최솟값을 이용하여 $L(n)$을 찾아내는 부분인 16번째 줄은 $O(h)$의 시간이 걸린다. 그러므로 총 $O(nh^2)$의 시간복잡도를 갖는다.

**(b)** $G$를 가중치가 적힌 유향그래프의 인접행렬이라고 하고, $path(i, k)$를 $v_i$로 끝나는 $(s_1, s_2, \ldots, s_k)$의 경로라고 두자. 그러면 $path(i, k)$는 $path(\cdot, k-1)$을 전부 검사하여 그곳에서 해당하는 $s_k$를 가진 간선이 있는지 여부를 확인하여 만약 있으면 그 경로에 끝에 이동 할 수 있는 곳을 붙여주면 된다. 그렇게 하면 $path(\cdot, k)$에는 원했던 답인 $s$를 거쳐 갈 수 있는 경로가 남는다. 의사코드 **Find**에는 그 중 정점의 번호가 가장 작은 곳에 도착하는

```
function L(n)
    L[1...n, 1...h] ▷ L(n,k) 메모이제이션 행렬
    M[1...h] ▷ f(x)를 저장해두는 행렬
    for i ← 1 to n do
        for j ← 1 to h do
            if i = 1 then
                L[i,j] ← [a_j]
                M[j] ← 0
            else
                q ← min_{1≤i≤h} {m_{n-1}(a_j) + φ(a_i, a_j)} 일 때 i
                L[i,j] ← L[i-1,q].append(a_j)
                M[j] ← M[j] + m_{n-1}(a_j) + φ(a_q, a_j)
            end if
        end for
    end for
    k ← min_{1≤i≤h} M[i] 일 때 i
    return L[n,k]
end function
```

경로를 출력하도록 하였다. $path(\cdot, k)$에 아무런 경로가 남지 않는다면 그 경로는 갈 수 없는 경로이므로 **No-Such-Path**를 출력하도록 하였다. (의사코드 **Find**를 Python 2로 구현한 실제 작동하는 코드를 첨부하였다.)

```
function FIND(G, s)
    path[0...n, 0...k]을 생성하고 전부 None으로 초기화 ▷ path(i,k)의 메모이제이션 행렬
    path[0,0] ← [0]
    for l ← 1 to k do
        for i ← 0 to n do
            if path[i, l-1] ≠ None then
                for j ← 0 to n do
                    if G[i,j] = s_l then
                        path[j,l] ← path[i, l-1]
                        path[j,l].append(j)
                    end if
                end for
            end if
        end for
    end for
    for i ← 0 to n do
        if path[i,k] ≠ None then
            return path[i,k]
        end if
    end for
    return "No-Such-Path"
end function
```

# Problem 2

$G$를 그래프의 인접리스트라고 하면, 인접한 정점끼리의 색이 겹치지 않도록 주변의 모든 정점의 색을 판단한 후 사용되지 않은 색을 칠하는 방식으로 하면 된다. 다만, 색을 칠할 때 최대한 사용했던 색을 사용하도록 하기 위해서 색에 번호를 부여한 후 작은 번호부터 차례로 체크하도록 하였다. 주변의 정점끼리만 비교해서 최소의 색을 사용하도록 하면 이가 전체의 해가 된다. 그래프 $G$를 주어진 조건에 맞게 색칠하는 알고리즘은 **Color**와 같이 짤 수 있다. (의사코드 **Color**를 Python 2로 구현한 예제는 뒤에 첨부되어 있다.)

```
1: function COLOR(G)
2:     A[0...|V|]을 생성하고 전부 0으로 초기화한다. ▷ A[i]: i번째 정점에 칠해진 색
3:     for i ← 0 to |V| do
4:         used[0...|V|+1]을 생성하고 전부 False로 초기화한다. ▷ used[i]: 색 i가 칠해진 여부
5:         used[0] ← True ▷ 색을 1부터 시작하게 하기 위해
6:         for j ∈ G[i] do
7:             used[A[j]] ← True
8:         end for
9:         for j ← 0 to |V|+1 do
10:            if not used[j] then
11:                A[i] ← j
12:                Break
13:            end if
14:        end for
15:    end for
16:    return A
17: end function
```

알고리즘 **Color**의 시간복잡도를 계산해보자. 첫 번째 for문 내부를 보면 for문이 하나 더 있는데 전부 펼쳐져 있기 때문에 for문 내부의 문장들은 $O(|V|)$번 실행되게 된다. 그러므로 전체적으로는 $O(V^2)$의 시간복잡도를 가지게 된다.

path.py

```
import copy

G = [[0, 2, 1, 0, 0],
     [0, 0, 2, 0, 3],
     [0, 0, 0, 5, 7],
     [0, 0, 0, 0, 2],
     [0, 0, 0, 0, 0]]


def find(G, s):
    path = []
    for i in range(0, len(G)):
        path.append([])
        for j in range(0, len(s) + 1):
            path[i].append(None)
    path[0][0] = [0]
    for k in range(1, len(s) + 1):
        for i in range(0, len(G)):
            if path[i][k - 1] is not None:
                for j in range(0, len(G)):
                    if G[i][j] == s[k - 1]:
                        path[j][k] = copy.copy(path[i][k - 1])
                        path[j][k].append(j)
    for i in range(len(G)):
        if path[i][len(s)] is not None:
            print path[i][len(s)]
            return
    print "No-Such-Path"
    return

find(G, (2, 2, 5, 2))

```

color.py

```python
G = [[1, 3],
     [0, 2, 3],
     [1, 4, 5],
     [0, 1, 4],
     [2, 3],
     [2]]


def color(G):
    A = [0 for x in range(len(G))]
    for i in range(len(G)):
        used = [False for x in range(len(G) + 2)]
        used[0] = True
        for j in G[i]:
            used[A[j]] = True
        for j in range(len(G) + 2):
            if not used[j]:
                A[i] = j
                break

    print A

color(G)

```