# CS202: 동적 계획법 보고서

20160385 오진영

## 1. Chaltteock

35 점짜리 풀이는 $O(n^2)$안에만 알아내면 되었으므로 naïve 하게 두 개의 떡의 모든 조합을 질문하면 되었지만, 100 점을 맞기 위해서는 O(nlogn) 안에 완료해야 하므로 위와 같은 방법으로는 어림 없었다. 그래서 처음 나온 아이디어가 up-down 게임에서처럼 이진 탐색을 이용하는 것이었는데, naïve 한 풀이에서 각 떡의 이웃한 떡을 찾기 위해 나머지 n-1 개의 떡에 대해 각각 한 번씩 확인했던 것을 이번에는 이진 탐색을 통해서 더 빠르고 효율적으로 찾는 것이다. '각 떡'을 pivot 이라 두고, 이웃한 떡이 있을 것으로 예상되는 수들의 집합을 반으로 나눈 후, 각 반의 집합에 대해서 pivot 을 포함한 것과 뺀 것의 덩어리 수를 물어 보면 된다. 이때, 만약 그 집합 안에 그 떡과 이웃한 떡이 있었더라면 pivot 을 뺐을 때 오히려 덩어리 수가 증가하게 된다. 그렇지 않은 경우, 다시 그 집합을 반으로 나눠 가면서 찾으면 된다.

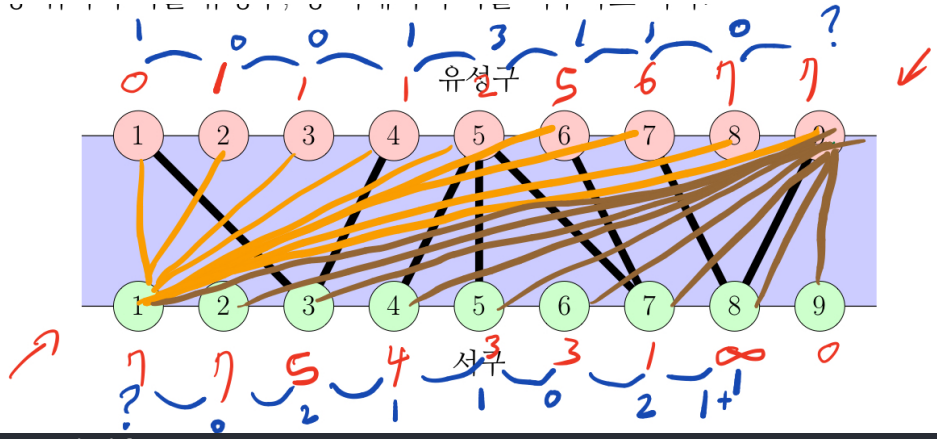
이렇게 한 결과 약 25000 번 정도가 필요했고, 문제의 조건을 만족시키기 위해 더 최적화가 필요했다. 위의 방법에서 가장 큰 오버헤드가 어디서 발생하는지 분석해 본 결과, 위의 알고리즘은 일반적인 이진 탐색이 한 쪽에 대해서만 조건을 검사해 그것이 아니라면 다른 쪽을 바로 취하는 것과 다르게 두 개의 이웃한 떡을 찾기 위해 양 쪽 모두 조건을 검사하기 때문에 오버헤드가 발생하는 것으로 보였다. 따라서, 이를 해결하기 위해서 한 쪽 끝을 찾고, 그 끝부터 아직 발견되지 않은 쪽 끝을 향해서 이웃한 떡을 하나씩 찾아 나가도록 하니 해결되었다.

맨 처음 시작 떡이 되는 것을 찾기 위해서는, 모든 떡의 집합에서 각각의 떡을 하나씩만 뺀 채로 pack 하여서 그 결과가 1 이라면 빠진 떡이 가장자리에 있다는 것이므로 그 떡을 사용하였다.

## 2. Bridges

아래 그림에서 보이는 것처럼, 유성구 쪽 모든 지점들에서는 서구의 1 번 지점으로 (주황 선), 서구 쪽 지점들에서는 유성구의 마지막 지점으로(갈색 선) sail 하였고, 그 과정에서 얻어진 값들을 붉은 색으로 나타내었다. 이때, 유성구의 왼쪽 지점들과 서구의 오른쪽 지점들에서는 값이 무한인 지점들(서구의 8 번 지점)이 나타나게 되는데, 우선 값이 무한인 지점들은 sail 한 것과 마찬 가지의 끝 목적지를 향해 난 다리 하나를 갖는다. 이와 더불어, 서구의 8 번 지점처럼 다른 부분으로도 난 다리를 가질 수 있는데, 이런 다리는 왼쪽 지점(유성구의 경우 오른쪽 지점)의 항해 결과값으로부터 얻어낼 수 있다.

나머지 지점들의 경우, 유성구의 경우 오른쪽 지점과 현재 지점 사이의(서구의 경우 왼쪽 지점과 현재 지점 사이의 값) sail 값의 차이가 현재 지점에 접해 있는 다리의 수가 된다. 이 다리의 수들은 파란색으로 표현하였다. 이렇게 하게 되면, 유성구의 맨 오른쪽과 서구의 맨 왼쪽 지점의 다리의 수를 측정하기 어려워 지는데, 이 점들은 값이 무한인 반대쪽 구 지점들의 수로부터 얻어낼 수 있다. 아래의 코드는 이 과정을 나타낸 코드다.




```
for(int i = 1; i < N; i++){
        if (youToLeftSeo[i] == -1){
            seo[1] += 1;
            you[i] = 1;
            if (youToLeftSeo[i+1] != -1){
                you[i] += youToLeftSeo[i+1];
            }
        } else if (youToLeftSeo[i+1] != -1){
            you[i] = youToLeftSeo[i+1] - youToLeftSeo[i];
        }
    }
```

각 지점에 나 있는 다리의 수를 구하고 나면 쉽다. 각 다리는 서로 교차할 수 없으므로, 유성구와 서구 양 쪽의 왼쪽 끝부터 시작해서 마치 합병 정렬에서 배열을 합치는 것과 유사하게 하나씩 다리를 그려 나가면 된다.

```
int sp = 1, yp = 1;
    while(sp <= N || yp <= N){
        while(you[yp] == 0) yp++;
        while(seo[sp] == 0) sp++;

        if(sp <= N && yp <= N) {
            answer.push_back(std::make_pair(yp,sp));
        }
        you[yp]--; seo[sp]--;
    }
```