

Day7 기출 풀이

선린인터넷고등학교 소프트웨어과

30610 나정휘

<https://JusticeHui.github.io>

문제 목록

- BOJ10713 기차 여행 (JOI15 #1)
- BOJ7573 고기잡이 (KOI13 지역 중등 #2)
- BOJ10714 케이크 자르기 2 (JOI15 #2)
- BOJ10165 버스 노선 (KOI14 고등 #2)
- BOJ2518 회전 테이블 (KOI12 고등 #3)
- BOJ18262 Milk Pumping (USACO19 Dec G1)
- BOJ18318 Springboards (USACO20 Jan G3)
- BOJ11750 Wall Clocks (ICPC Japan Regional 15 D)

기차 여행

- i 번 철도를 k 번 이용할 때 비용 : $\min(A_i * k, B_i * k + C_i)$
- 각 철도를 몇 번 이용하는지 알면 문제를 풀 수 있다!

기차 여행

- 필요한 연산
 - 임의의 구간 $[L, R]$ 에 x 를 더한다.
 - 더하는 쿼리가 모두 끝난 뒤, 임의의 지점 p 의 값을 구한다.
- Segment Tree + Lazy Propagation
- Fenwick Tree
- Prefix Sum
- 구현은 쉽다.

고기 잡이

- 그물의 왼쪽/위 경계에 물고기가 걸치는 경우만 보면 된다.
 - 그렇지 않는 경우에는 경계에 걸치도록 이동시켜주는 것이 이득이다.

고기 잡이

- 모든 쌍 (i, j) 에 대해
 - 그물의 왼쪽 경계에 i 번째 물고기가 있고
 - 그물의 위쪽 경계에 j 번째 물고기가 있을 때
 - 포획할 수 있는 물고기의 수를 구해주면 된다.
- $O(M^3)$ 에 풀 수 있다.

케이크 자르기 2

- $D(l, r)$ = 현재까지 가져간 가장 왼쪽 위치는 l , 오른쪽 위치는 r 일 때 최댓값
 - ioi의 차례에는 $(l-1)$ 과 $(r+1)$ 중 더 큰 값을 가져간다.
 - joi의 차례에는 $(l-1)$ 과 $(r+1)$ 중 가져갔을 때 최종적으로 더 큰 결과를 갖게 되는 쪽을 가져간다.
 - 사실 $(l-1)$, $(r+1)$ 이 아니라 $(l+n-1)\%n$ 과 $(r+1)\%n$ 이다.
- 아무튼 $O(N^2)$ 에 풀린다.

버스 노선

- 선형이면 쉬운데 원형이라서 귀찮다.
- 0번과 N-1번 사이에 있는 도로를 지나는 노선과 그렇지 않은 노선으로 분류하자.
 - 안 지나는 노선 : A그룹
 - 지나는 노선 : B그룹

버스 노선

- A그룹을 먼저 보자. (0 ~ N-1을 안 지나는 노선)
- 두 노선 a, b가 아래 조건을 만족하면 a는 없어도 된다.
 - $S[b] \leq S[a] < E[a] \leq E[b]$

버스 노선

- A그룹만 보자. (0 ~ N-1을 안 지나는 노선)
- 두 노선 a, b가 아래 조건을 만족하면 a는 없어도 된다.
 - $S[b] \leq S[a] < E[a] \leq E[b]$
- 시작 지점 오름차순, 도착 지점 내림차순으로 정렬하자.
 - 뒤에 있는 노선이 앞에 있는 노선을 포함하는 경우가 없다.

버스 노선

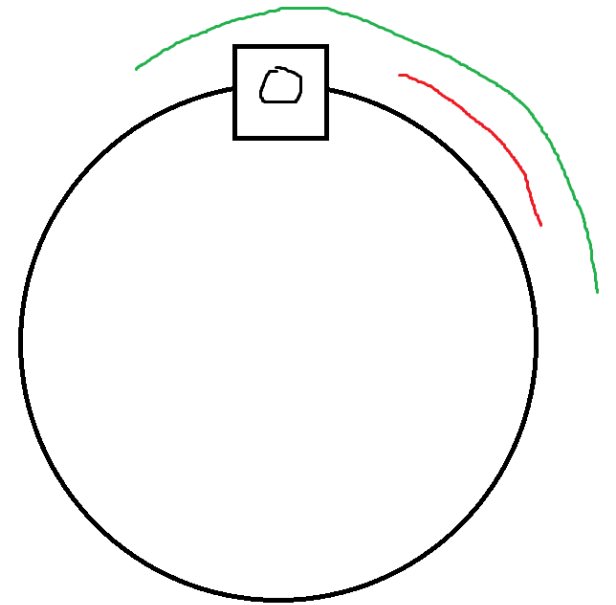
- B그룹을 보자. ($0 \sim N-1$ 을 지나는 노선)
 - A그룹에 있는 노선이 B그룹을 포함하는 경우는 없다.
 - B그룹이 A그룹을 포함 / B그룹이 또 다른 B그룹을 포함 2가지만 보자.

버스 노선

- B그룹이 또 다른 B그룹을 포함하는 경우
 - 노선의 종료 지점에 N을 더해준 뒤 A그룹처럼 처리해주면 된다.
- $N = 9$, $[7, 2]$, $[8, 1]$ 노선이 있는 경우
 - $[7, 11]$, $[8, 10]$ 으로 바꿔주면 A그룹처럼 처리해줄 수 있음

버스 노선

- B그룹이 A그룹을 포함하는 경우
- 빨간색을 A그룹의 노선, 초록색을 B그룹의 노선이라고 하자.
 - 빨간색의 종료지점 \leq 초록색의 종료지점
 - B그룹 노선들의 종료지점의 최댓값을 구하면 된다.



버스 노선

- A그룹이 A그룹 포함 : 정렬 후 적절히 처리
- B그룹이 B그룹 포함 : 도착 지점에 N 더하고 A그룹처럼 처리
- B그룹이 A그룹 포함 : $\max(\text{B그룹 노선 도착지점})$ 구해서 비교
- deque 쓰면 구현 잘 할 수 있음.

```
#include <bits/stdc++.h>
using namespace std;

struct Info{
    int st, ed, idx;
    bool operator < (const Info &t) const {
        return make_pair(st, -ed) < make_pair(t.st, -t.ed);
    }
};

int n, m, back;
Info a[505050];
deque<Info> dq;

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    cin >> n >> m;
    for(int i=1; i<=m; i++){
        cin >> a[i].st >> a[i].ed; a[i].idx = i;
        if(a[i].st > a[i].ed) back = max(back, a[i].ed), a[i].ed += n;
    }
    sort(a+1, a+m+1);
    for(int i=1; i<=m; i++){
        if(dq.empty() || dq.back().ed < a[i].ed) dq.push_back(a[i]);
    }
    while(dq.size() && dq.front().ed <= back) dq.pop_front();
    sort(dq.begin(), dq.end(), [](Info a, Info b){ return a.idx < b.idx; });
    for(auto i :dq) cout << i.idx << " ";
}
```

회전 테이블

- $D(a, b, c, d)$
 - 1번 사람이 a 개
 - 2번 사람이 b 개
 - 3번 사람이 c 개 먹었고
 - 마지막으로 d 번 사람이 먹었을 때의 최소 회전
- $O(P1 * P2 * P3 * 3)$

Milk Pumping

1. 최단 경로를 구한다.
2. (유량/비용)을 구해서 최댓값을 갱신한다.
3. 1번에서 구한 경로 상에서 용량이 가장 작은 간선을 제거한다.
4. 1과 N이 연결되어 있다면 1번으로 돌아간다.

Springboards

- 오른쪽/위로만 이동
 - DP?

Springboards

- 각 스프링보드의 시작점까지 최단 거리만 알면 된다.
- $\text{dst}[i]$ = i 번째 스프링보드의 시작점까지 가는 최단 거리

Springboards

- 각 스프링보드의 시작점까지 최단 거리만 알면 된다.
- $dst[i]$ = i 번째 스프링보드의 시작점까지 가는 최단 거리
 - $(0, 0)$ 부터 걸어서 가면 $x1[i] + y1[i]$
 - 적당한 j 에 대해 j 번째 스프링보드를 거쳐서 가면
 - $dst[j] + (x1[i] - x2[j]) + (y1[i] - y2[j])$
 - $(x1[i] + y1[i]) + (dst[j] - x2[j] - y2[j])$
 - 최솟값 취하면 된다.

Springboards

- 현재 i 번째 스프링보드를 보고 있다면 $dst[i]$ 는
 - $x2[j] \leq x1[i] \ \& \ y2[j] \leq y1[i]$ 인 모든 j 에 대해
 - $(x1[i] + y1[i]) + (dst[j] - x2[j] - y2[j])$ 의 최솟값
- 시작점과 끝점들을 x, y 좌표 오름차순으로 정렬해주면
 - $j < i$ 일 때 $x[j] > x[i] \ \& \ y[j] > y[i]$ 인 경우가 없다.
 - 즉, i 번째 점을 보고 있을 때 필요한 모든 j 는 이미 처리된 상태

Springboards

- 스윙핑을 하자.
- 현재 보고 있는 점이 스프링보드의 도착점이라면
 - $(dst[j] - x2[j] - y2[j])$ 를 업데이트 해야 하고
- 현재 보고 있는 점이 스프링보드의 시작점이라면
 - $dst[i]$ 를 $(x1[i] + y1[i]) + (dst[j] - x2[j] - y2[j])$ 의 최솟값으로 갱신해야 한다.

Springboards

- 점들의 x좌표는 이미 단조증가 하므로 y좌표만 신경 써도 된다.
- 구간의 최솟값을 관리하는 세그먼트 트리를 만들자.
- 현재 보고 있는 점이 스프링보드의 도착점이라면
 - y좌표에 $(dst[j] - x2[j] - y2[j])$ 를 저장한다.
 - `minUpdate(y2[j], dst[j] - x2[j] - y2[j]);`
- 현재 보고 있는 점이 스프링보드의 시작점이라면
 - $dst[i] = minQuery(0, y1[i]) + x1[i] + y1[i];$

Springboards

- 좌표 범위가 10억이다.
- Dynamic Segment Tree를 구현하면 된다.

Wall Clocks

- 각 사람의 시야를 “구간”으로 표현할 수 있다.
- 원형에서 문제를 풀기 전에 선형을 먼저 생각해보자.

Wall Clocks

- 선형인 경우
- 끝점 기준으로 정렬한 다음
- 최대한 뒤쪽에 시계를 배치하는 그리디로 풀 수 있다.
- $O(N \log N)$

Wall Clocks

- 원형이긴 한데 $N \leq 1000$ 이다.
- 선형일 때의 $O(N \log N)$ 풀이를 N 번 반복해주면 될 것 같다!

Wall Clocks

- 원형이긴 한데 $N \leq 1000$ 이다.
- 선형일 때의 $O(N \log N)$ 풀이를 N 번 반복해주면 될 것 같다!
- 구간 하나 잡아서 시작점 기준으로 끊어주면 원형이 선형으로 바뀐다.
- N 번의 시도에서 시계를 가장 적게 쓰는 경우의 개수를 출력