

# Day5 DP

선린인터넷고등학교 소프트웨어과  
30610 나정휘

# 문제 목록

- BOJ10211 Maximum Subarray
- BOJ10844 쉬운 계단 수
- BOJ15966 군계일학
- BOJ11049 행렬 곱셈 순서
- BOJ10836 여왕벌
- BOJ5550 헌책방
- BOJ12013 248 게임
- BOJ2213 트리의 독립집합
- BOJ17435 합성함수와 쿼리
- BOJ5463 건포도
- BOJ1413 박스 안의 열쇠
- BOJ12920 평범한 배낭
- BOJ12008 262144
- BOJ6171 땅따먹기

# Maximum Subarray

- Kadane's Algorithm

```
int n, a[1010];
void solve(){
    cin >> n;
    for(int i=1; i<=n; i++) cin >> a[i];

    int mx = -1e9, now = 0;
    for(int i=1; i<=n; i++){
        // now : i번째 원소를 마지막 원소로 갖는 부분 배열의 최댓값
        now = max(now, 0) + a[i];
        ans = max(ans, now);
    }
    cout << ans << "\n";
}
```

# 쉬운 계단 수

- $D(i, j)$  =  $i$ 번째 자리까지, 마지막 자리가  $j$ 인 경우의 수
- $D(i, j) = D(i-1, j-1) + D(i-1, j+1)$

# 군계일학

- 1씩 증가하는 가장 긴 부분 수열을 찾는 문제
  - LIS 비슷하게?
- $D(i, v) = 1 \sim i$ 번째 원소에서,  $j$ 로 끝나는 1씩 증가하는 최대 길이
  - $D(i, A[i]) = \max\{ D(i-1, A[i]), D(i-1, A[i]-1) + 1 \}$ 
    - $i$ 번째 원소를 선택하지 않거나,  $i$ 번째 원소를 선택하거나

# 군계일학

- 10만 \* 100만짜리 테이블을 굳이 잡아야 하나?
- $D(v)$  = 지금까지 본 원소들에서  $v$ 로 끝나는 최대 길이
  - 크기 100만짜리 테이블

```
int dp[1010101];
for(int i=1; i<=n; i++){
    int t; cin >> t;
    dp[t] = max(dp[t], dp[t-1] + 1);
}
```

# 행렬 곱셈 순서

- $D(i, j)$  =  $i$ 번째 행렬부터  $j$ 번째 행렬까지 곱할 때 최소 비용
  - 구간에 대한 DP
- $D(i, i+1) = R[i] * C[i] * C[i+1]$
- $D(i, j) = \min\{ D(i, k) + D(k+1, j) + R[i] * C[k] * C[j] \}$ 
  - $i \sim k$ 번째 행렬(  $R[i]$  by  $C[k]$  )을 곱할 때 최소 비용
  - $k+1 \sim j$ 번째 행렬(  $R[k+1]$  by  $C[j]$  )을 곱할 때 최소 비용
  - 2개의 행렬을 곱한 비용

# 여왕벌

- 풀이1.  $O(NM + N^2)$
- 풀이2.  $O(M + N^2)$

메모리	시간
4484 KB	268 ms
4344 KB	1560 ms



# 여왕벌

- 첫 번째 행&열의 값은 각 날짜마다  $O(M)$ 에 갱신 가능
  - 총  $O(NM)$
- 첫 행&열의 최종 값을 이용해 나머지는 DP로 채워주면 됨
  - $D(i, j) = \max\{ D(i-1, j), D(i, j-1) \}$
  - $O(M^2)$
- 전처리  $O(NM)$ , DP  $O(M^2)$

# 여왕벌

- $\text{border[]} = \{ (M, 1), (M-1, 1), \dots, (1, 1), (1, 2), \dots, (1, N) \}$ 
  - $\text{border}[0] \sim \text{border}[a-1]$ 까지 0 더하고
  - $\text{border}[a] \sim \text{border}[a+b-1]$ 까지 1 더하고
  - $\text{border}[a+b] \sim$  에 2 더하고
- Prefix Sum!
  - 모든 쿼리에 대해  $\text{sum}[a]++$ ;  $\text{sum}[a+b]++$ ;
  - 마지막에  $\text{for}(\text{int } i=1; ; i++) \text{sum}[i] += \text{sum}[i-1];$
- 전처리  $O(N + M)$ , DP  $O(M^2)$

# 헌책방

- 동일한 장르의 책을 매입하는 경우
  - 가격이 비싼 것을 매입하는 것이 무조건 이득
- $C(i, j)$  =  $i$ 번째 장르의 책을  $j$ 권 매입할 때 최대 가격
- $D(i, j)$  = 1.. $i$ 번째 장르를 총  $j$ 권 매입할 때의 최대 가격
- $C$ 는 그리디하게 구하고,  $D$ 는 DP로 구하면 됨

# 248 게임

- $D(i, j)$  =  $i$ 번째 수부터  $j$ 번째 수까지 **하나의 수로** 합칠 때 최댓값
  - 하나의 수로 합치지 못하면 0
- $D(i, j) = \max\{ D(i, k) + 1 \}$ 
  - if  $D(i, k) = D(k+1, j)$

# 트리의 독립집합

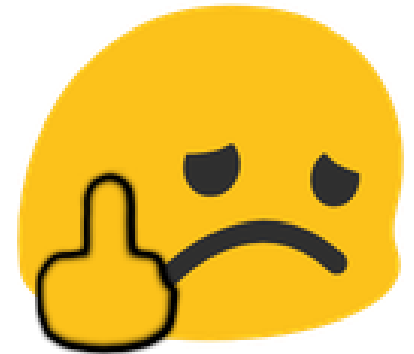
- 트리 DP
  - $D(v, \text{state})$  =  $v$ 를 루트로 하는 서브 트리의 상태가 state일 때 ~~~
- $D(v, \text{flag})$  =  $v$ 를 루트로 하는 집합에서
  - $\text{flag} = 0$  :  $v$ 를 집합에 넣지 않을 때
  - $\text{flag} = 1$  :  $v$ 를 집합에 넣을 때
  - 최대 독립 집합의 크기
- 역추적 파이팅!

# 합성함수와 쿼리

- Sparse Table
- $D(x, i) = f_{2^i}(x)$
- $D(x, i) = D(D(x, i-1), i-1)$

# 건포도

- 업데이트가 없을 때 직사각형 영역의 합을 구하는 방법
  - 2D Segment Tree - 구데기
  - 2D Fenwick Tree - 구데기
  - Merge Sort Tree - 차라리 PST를 쓴다
  - Persistent Segment Tree - 좌표 범위도 좁은데 굳이?
- 2D Prefix Sum - 편안



# 건포도

- 2D Prefix Sum - 전처리  $O(NM)$ , 쿼리  $O(1)$

```
int a[55][55];
for(int i=1; i<=n; i++) for(int j=1; j<=m; j++){
    cin >> a[i][j];
}

// init
for(int i=1; i<=n; i++) for(int j=1; j<=m; j++){
    a[i][j] += a[i][j-1];
    a[i][j] += a[i-1][j];
    a[i][j] -= a[i-1][j-1];
}

// query
int x1, x2, y1, y2;
cout << a[x2][y2] - a[x2][y1-1] - a[x1-1][y2] + a[x1-1][y1-1];
```



# 건포도

- $D(i, j, n, m)$  =  $(i, j)$ 부터 가로 길이가  $n$ , 세로 길이가  $m$ 인 초콜릿을 쪼개는 최소 비용
- $D(i, j, n, m) =$ 
  - $\min\{ D(i, j, k, m) + D(i+k, j, n-k, m) \} + \text{직사각형 영역 합}$
  - $\min\{ D(i, j, n, k) + D(i, j+k, n, m-k) \} + \text{직사각형 영역 합}$
- $N = M$ 인 경우  $O(N^5)$
- TL 3초 & 상수가 작아서 잘 돌아감
- $N^6$ 도 뚫린다는 소문이 있을 정도로 잘 돌아감

# 평범한 배낭 2

- Do You Know Knapsack Problem?
- 각 물건을 최대 하나만 가져갈 수 있는 문제는 쉽게 풀림
  - $O(NM)$
  - 각 물건의 개수 제한이 없어도 비슷하게 풀림
- **개수 제한  $K$**

# 평범한 배낭 2

- Naïve Solution
  - 각 물건을 K개씩 만들어주면  $O(NMK)$  - TLE
- 각 물건을 K개씩 만들고 탐색을 하면 TLE가 날 수 밖에 없다.
  - K개보다 덜 만들 수 있을까?

# 평범한 배낭 2

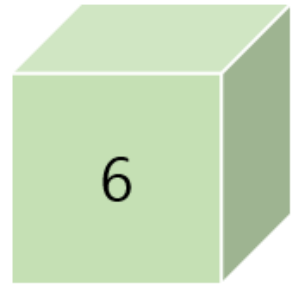
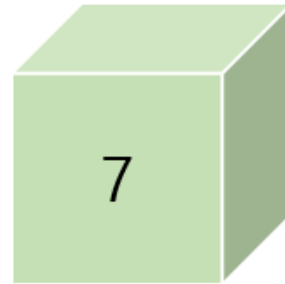
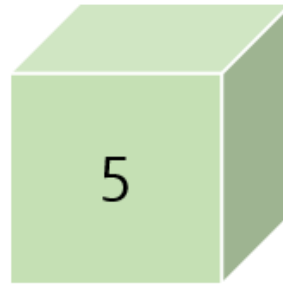
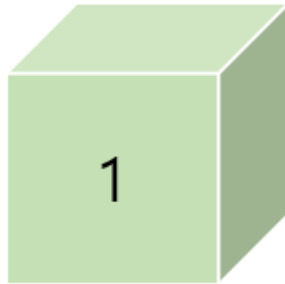
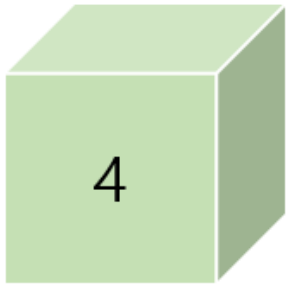
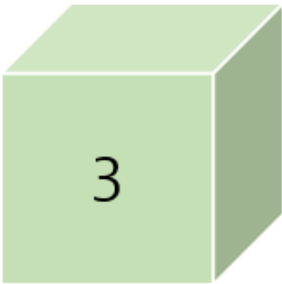
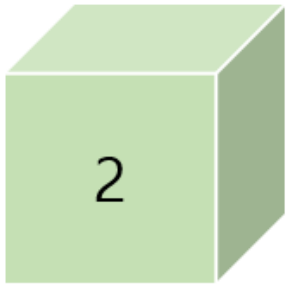
- $K = 14$ 라고 가정하자.
- 1개, 2개, 4개,  $14 - (1 + 2 + 4) = 7$ 개 있는 패키지를 만들면 된다.
  - 이진법
- 각 물건을  $O(\log K)$ 개의 패키지로 쪼갤 수 있고
- $O(N \log K)$ 개의 패키지이므로  $O(NM \log K)$ 에 풀 수 있다.

# 평범한 배낭 2

- $O(NM)$  풀이는 <https://moonrabbit2.tistory.com/3> 에서 확인

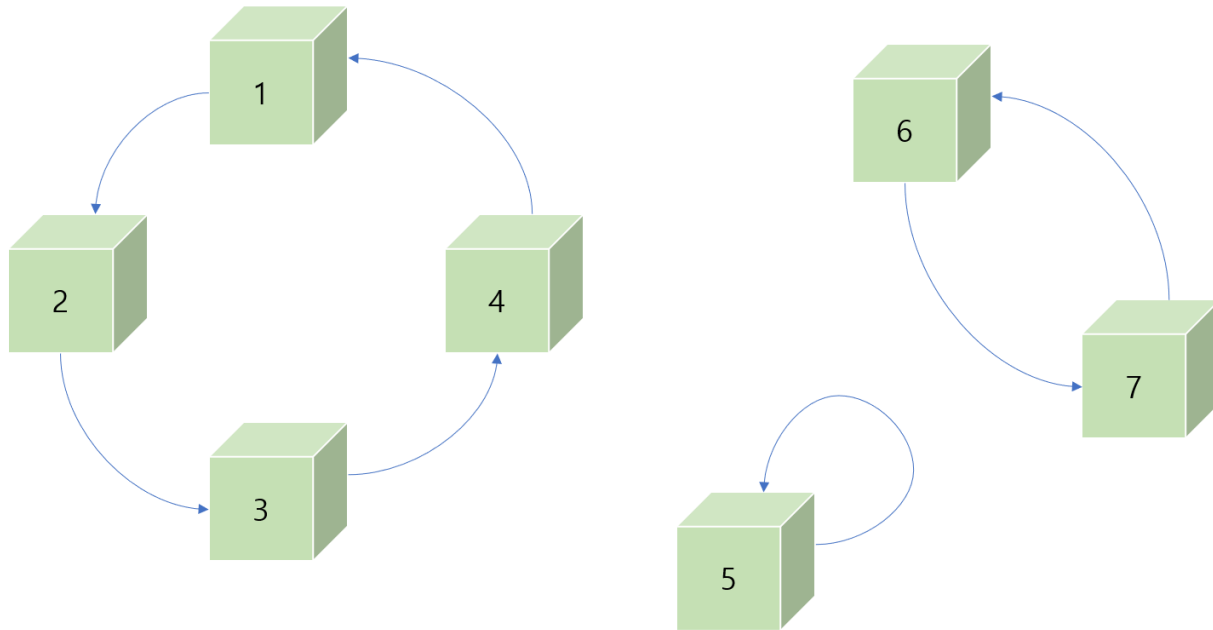
# 박스 안의 열쇠

- 1번 박스 -> 2번 열쇠, 2번 박스 -> 3번 열쇠, ...



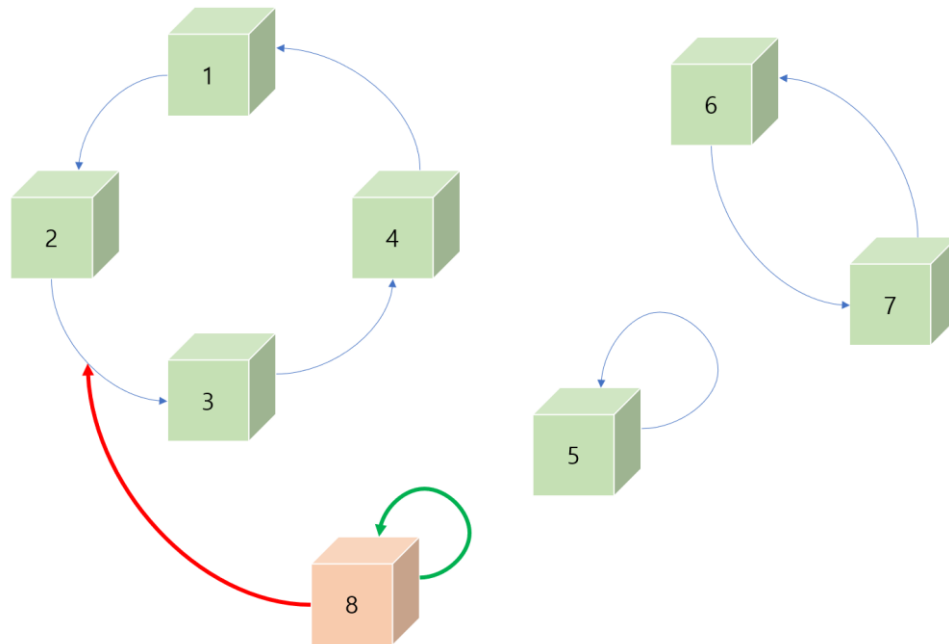
# 박스 안의 열쇠

- 그래프로 나타내면
- 폭탄 하나를 이용해 사이클을 모두 열 수 있다.



# 박스 안의 열쇠

- 새로운 8번 박스가 들어간다고 하자.
  - 기존 사이클 중 한 곳에 들어가거나 (빨간색)
  - 혼자서 새로운 사이클을 구성하거나 (초록색)





# 박스 안의 열쇠

- 기존 사이클에 들어가는 경우
  - 간선 중간에 들어가는 것
  - N번째 박스를 N-1개의 간선 중 하나를 선택해서 삽입
- 혼자서 새로운 사이클을 만드는 경우
  - 폭탄이 하나 더 필요함

# 박스 안의 열쇠

- $D(i, j)$  =  $i$ 개의 열쇠를 정확히  $j$ 개의 폭탄으로 얻는 경우의 수
- $D(i, j) = D(i-1, j) * (i-1) + D(i-1, j-1)$
- 모든 열쇠를 얻을 확률
  - $(1..M)$ 개의 폭탄으로 얻는 경우의 수 /  $(1..N)$ 개의 폭탄으로 얻는 경우의 수

# 262144

- $[i, k]$  구간을 잘 합쳐서  $X$ 를 만들고
- $[k+1, j]$  구간을 잘 합쳐서  $X$ 를 만들었다면
  - $[i, j]$  구간을 잘 합쳐서  $X+1$ 을 만들 수 있다.

# 262144

- Sparse Table을 생각해보자.
  - 크기가  $2^{i-1}$ 인 구간 2개를 붙여서  $2^i$ 인 구간을 만든다.
  - $ST(x, i) = x$ 에서 시작하는 크기가  $2^{i-1}$ 인 구간의 끝점의 다음 위치
- 262144 문제는
  - $(i-1)$ 을 만들 수 있는 구간 2개를 붙여서  $i$ 를 만든다.
  - $D(x, i) = x$ 에서 시작해서  $i$ 를 만들 수 있는 구간의 끝점의 다음 위치?

# 262144

- $(i \sim k)$  번째 원소들을 사용해서 **하나의 수**  $j$ 를 만들 수 있다면
  - $D(i, j) = k+1$ 로 정의
  - 그러한  $k$ 가 존재하지 않는다면 0으로 초기화
- 
- $A[i] = j \rightarrow D(i, j) = j+1$
  - $D(i, j-1) = 0 \rightarrow D(i, j) = 0$
  - $D(i, j) = D(D(i, j-1), j-1)$

262144

- 배열의 각 원소는 최대 40
- 정답의 최댓값은  $40 + O(\log N)$

# 땅따먹기

- $W[i] > W[j] \ \& \ H[i] > H[j]$  이면  $j$ 는 신경 쓰지 않아도 됨
- 직사각형의 높이가 증가하도록, 너비는 감소하도록 정렬
- 연속한 직사각형을 한 번에 구매하는 것이 이득

# 땅따먹기

- $D(i)$  =  $i$ 번째 직사각형까지 구매했을 때의 최소 비용
- $D(i) = \min\{ D(j-1) + H[i] * W[j] \}$ 
  - $O(N^2)$
  - 최적화 필요

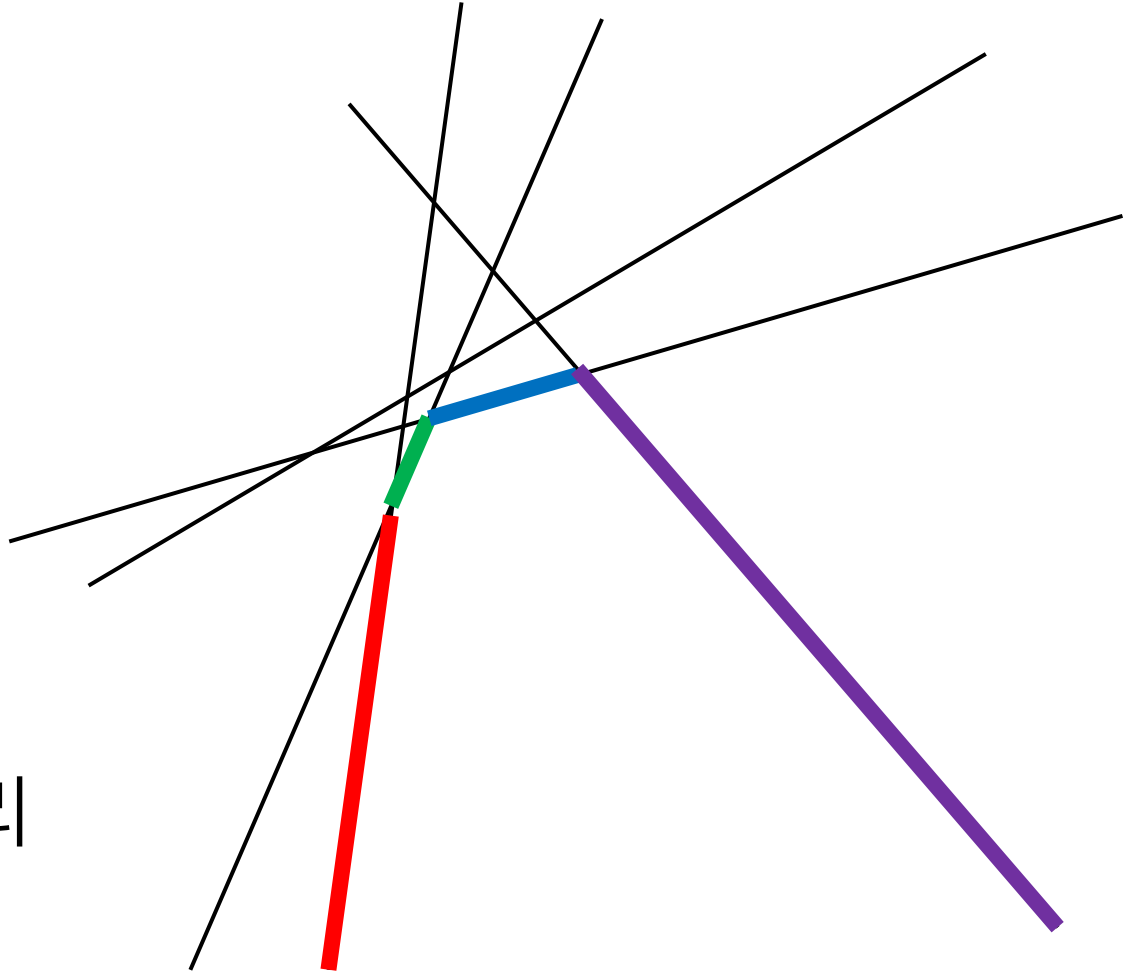


# 땅따먹기

- $D(i) = \min\{ D(j-1) + H[i] * W[j] \}$
- $W[j] = a, D(j-1) = b, H[i] = x$  치환
  - $D(i) = \min\{ ax+b \}$
  - 여러 개의 직선이 주어졌을 때 임의의 x좌표에서의 최솟값
- $W$ (기울기)는 감소,  $H$ (x좌표)는 증가

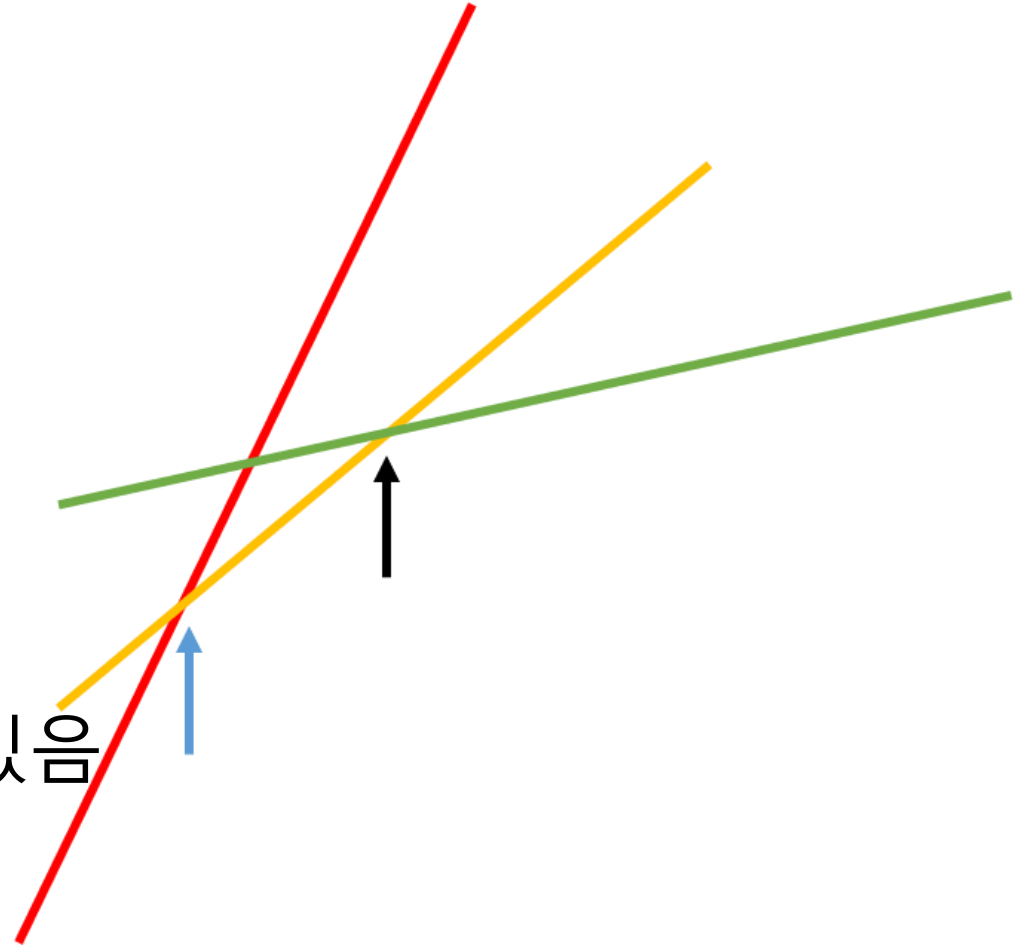
# 땅따먹기

- 최소가 되는 선분들을 보자
- 기울기가 감소하는 볼록 함수
- 해야할 일
  - 최소가 되는 선분들의 집합을 관리
  - 선분 집합에서 최솟값 찾기
- 당연히 기울기가 감소하도록 관리



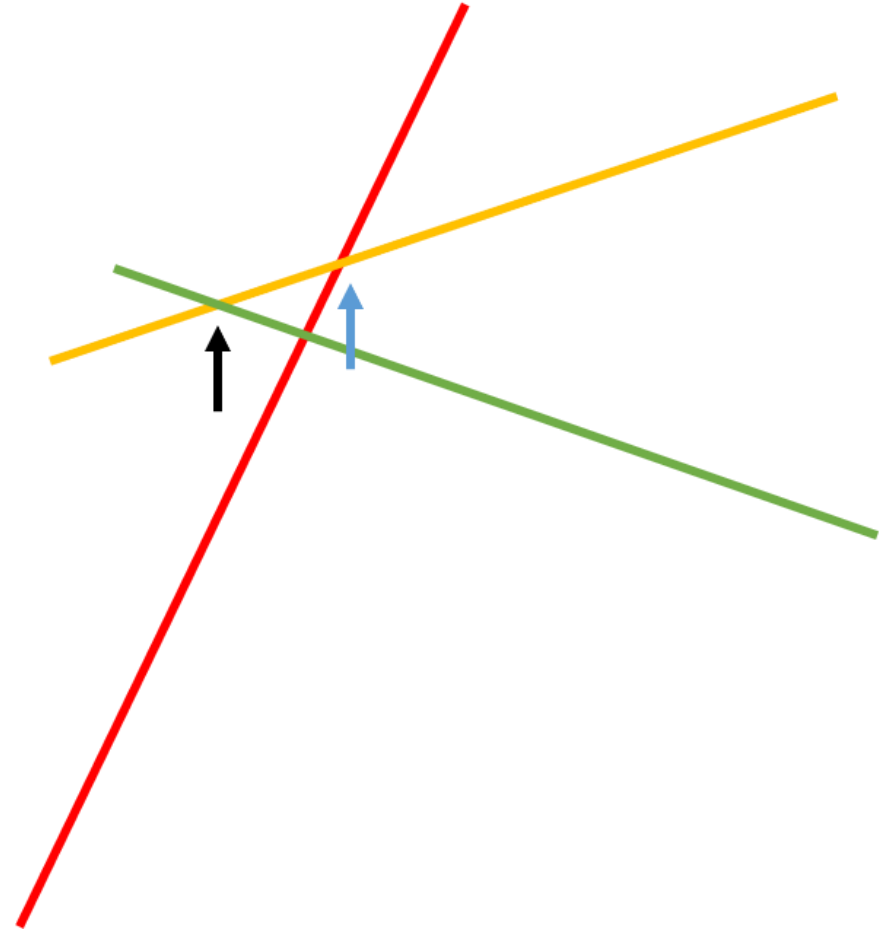
# 땅따먹기

- 최소가 되는 선분 집합 관리
- 빨간 직선과 노란 직선이 있을 때  
초록 직선을 삽입하는 상황
- 초록 직선이 들어와도  
노란 직선이 최소가 되는 구간이 남아있음



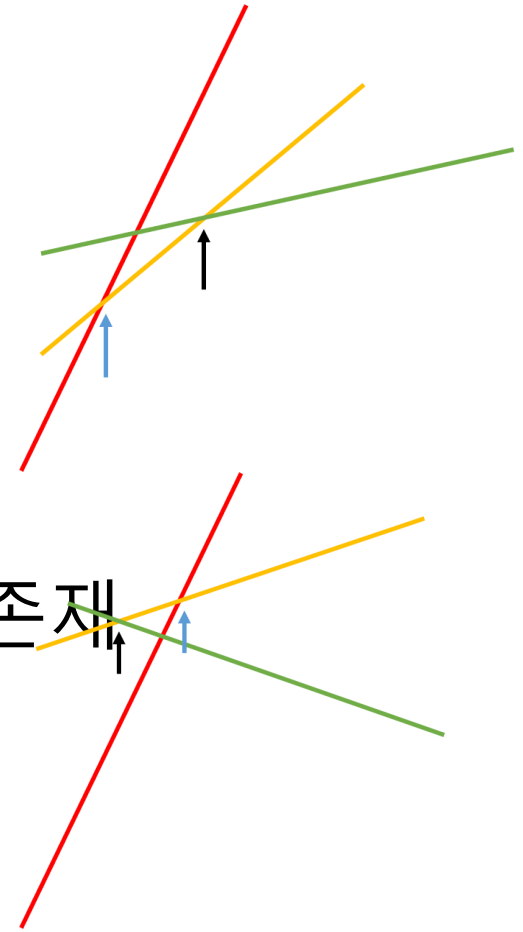
# 땅따먹기

- 최소가 되는 선분 집합 관리
- 빨간 직선과 노란 직선이 있을 때  
초록 직선을 삽입하는 상황
- 초록 직선이 들어오면  
노란 직선이 최소가 되는 구간이 없어짐



# 땅따먹기

- 빨간색, 노란색, 초록색 직선 : 1, 2, 3번 함수
- 1번 함수와 2번 함수의 교점의 위치 :  $x_1$
- 2번 함수와 3번 함수의 교점의 위치 :  $x_2$
- $x_1 < x_2$ 이면 2번 함수가 최소가 되는 구간이 존재
- $x_1 \geq x_2$ 이면 없음



# 땅따먹기

- 최소가 될 수 있는 직선들의 집합을 스택으로 관리
  - while( $x_1 \geq x_2$ ) stack.pop();
  - stack.push(line)
  - 들어오는 직선들의 기울기가 감소하지 않는다면 set이나 세그 써야함

```
void insert(int a, int b){ // ax+b
    Line line(a, b);
    while(stk.size() >= 2 &&
        cross(stk[-2], stk[-1]) > cross(stk[-1], line)){
        stk.pop_back();
    }
    stk.push_back(line);
}
```

# 땅따먹기

- 주어진  $x$ 좌표에서 최소값 찾기
  - 단, 주어진  $x$ 좌표들은 증가한다.
- 스택에서 포인터 관리해주면 된다.
  - $i$ 번째 직선과  $i+1$ 번째 직선의 교점  $\leq x$ 이면  $i$ 를 증가시킴
  - 최종적으로 가리키게 되는 직선이 최소가 됨
  - 주어진  $x$ 좌표가 증가하지 않으면 이분 탐색 하거나 세그

```
int pt;
int query(int x){
    while(pt+1 < stk.size() &&
          cross(stk[pt], stk[pt+1]) <= x){
        pt++;
    }
    return stk[pt].a * x + stk[pt].b;
}
```