DP Optimization 1

나정휘
https://justiceHui.github.io/

목차

- Convex Hull Trick
- Li Chao Tree
- Hirschberg’s Algorithm
- Divide and Conquer Optimization
- Monotone Queue Optimization
- Aliens Trick

Convex Hull Trick

Convex Hull Trick

- 이런 문제를 생각해 보자.
  - N개의 일차 함수 $f_i(x) = a_ix + b_i$ 가 있을 때
  - x_j가 주어지면 max{ $f_i(x_j)$ } 를 구하는 문제
  - 또는 일차 함수를 추가할 수도 있음

  - 단순하게 구현하면 O(NQ)
  - Convex Hull Trick을 이용하면 더 빠르게 해결할 수 있음

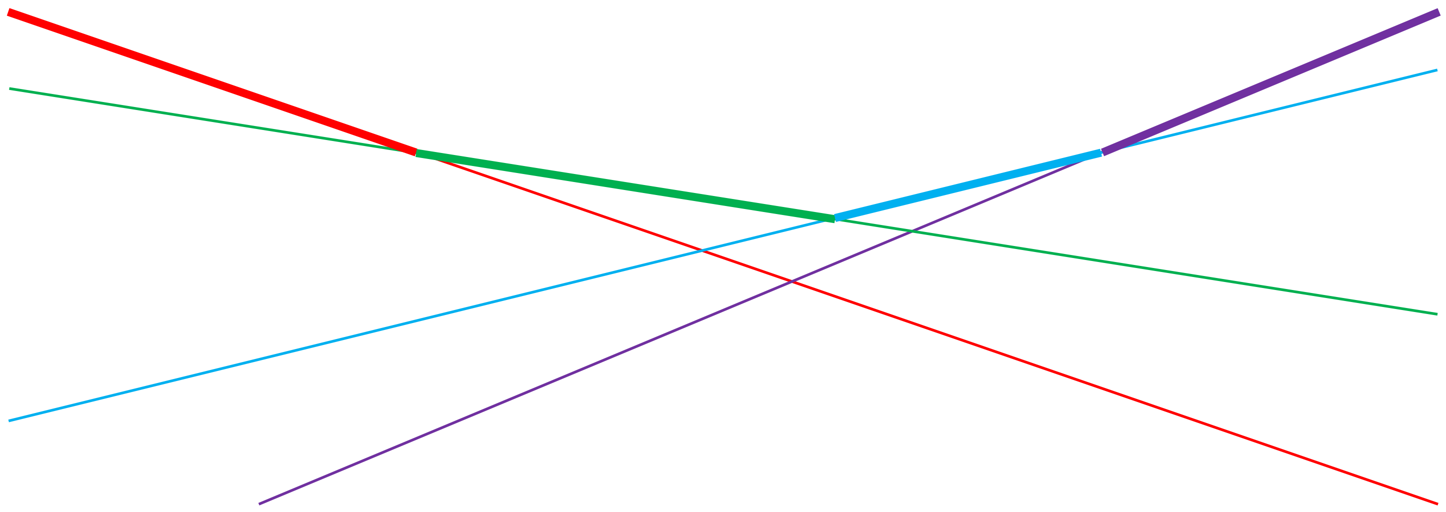
Convex Hull Trick

- Convex Hull Trick
  - 제약 조건에 따라 다양한 방법이 존재함
    - 직선 추가를 N번, 최댓값 쿼리를 Q번 하는 경우
    - $A_i \le A_{i+1}$, $x_j \le x_{j+1}$ 일 때 O(N + Q)
    - 일반적인 상황에서 O((N + Q) sqrt N)
    - 일반적인 상황에서 O((N + Q) log N)
    - 일반적인 상황에서 O((N + Q) log X)

    - 오늘은 하늘색으로 표시한 두 가지 방법을 다룸

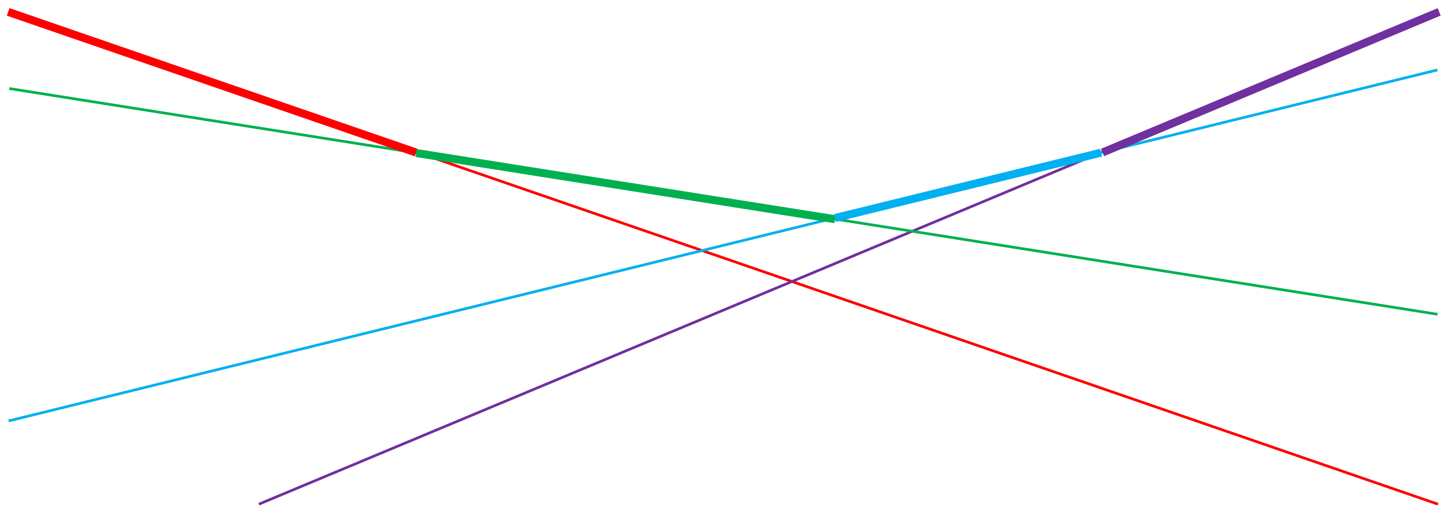
Convex Hull Trick

- Convex Hull Trick
  - $f(x) = \max f_i(x)$의 개형은 어떤 형태일까?
    - 아래로 볼록한 함수 – 이름이 Convex Hull Trick인 이유
      - 뒤에서 최대가 되는 직선일 수록 기울기가 큼
    - 각 직선이 최대인 구간을 모두 알고 있다면 최댓값 쿼리를 효율적으로 처리할 수 있음



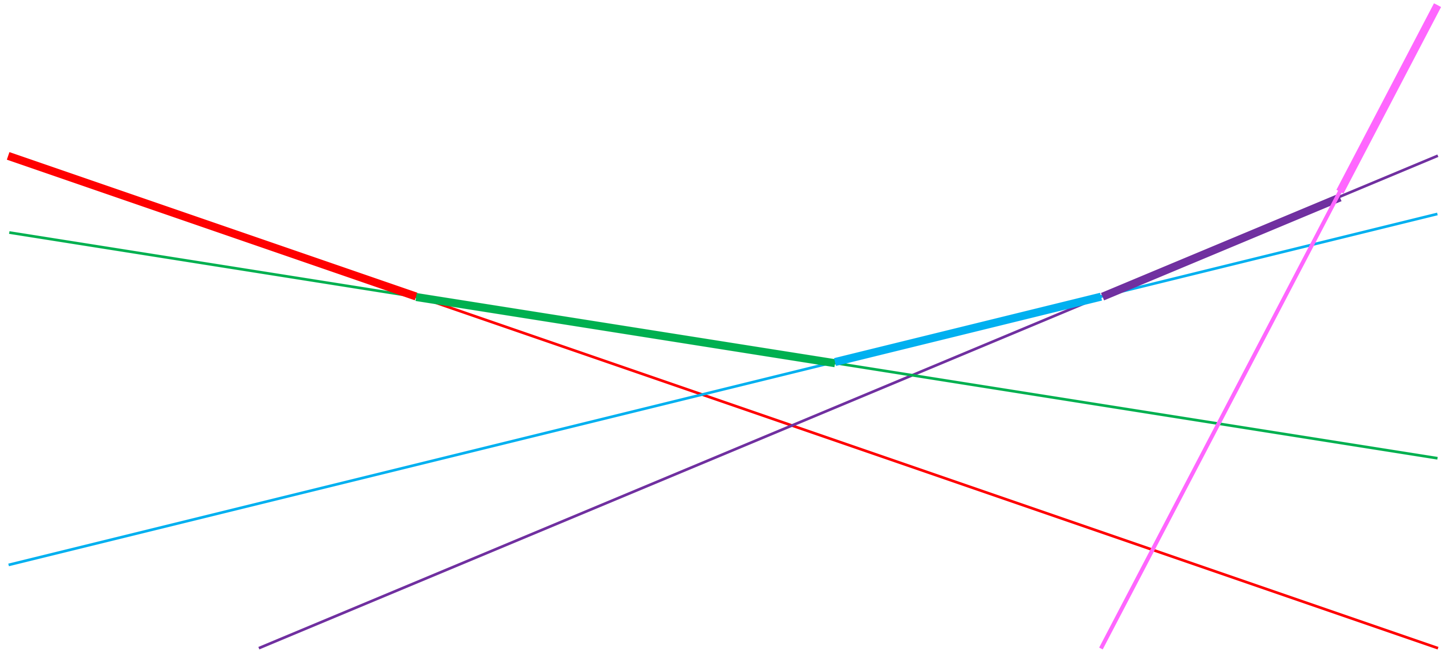
Convex Hull Trick

- Convex Hull Trick
  - 구현해야 하는 연산
    - add_line(a, b): f(x) = ax + b 직선 추가, 각 직선이 최대가 되는 구간 갱신
    - get_max(x): x에서의 최댓값 출력



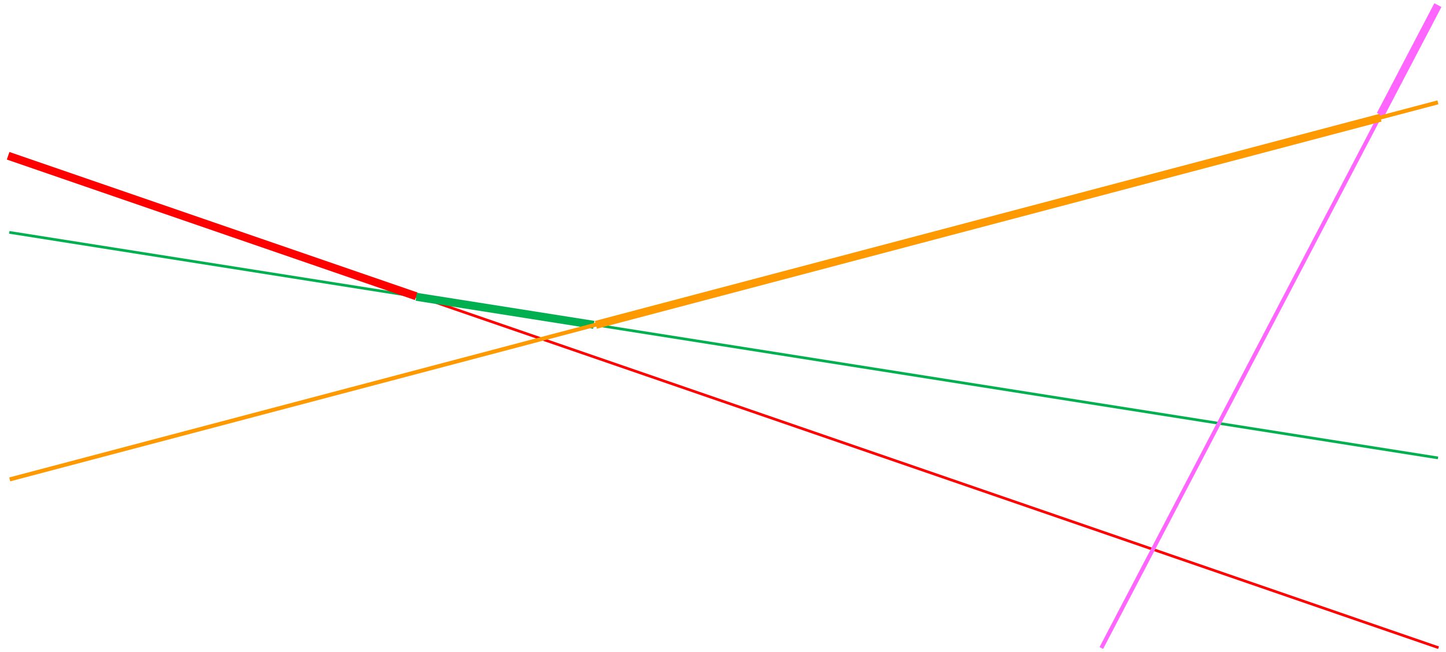
Convex Hull Trick

- Convex Hull Trick
  - 구현해야 하는 연산
    - add_line(a, b): f(x) = ax + b 직선 추가, 각 직선이 최대가 되는 구간 갱신
    - get_max(x): x에서의 최댓값 출력



Convex Hull Trick

- Convex Hull Trick
  - 구현해야 하는 연산
    - add_line(a, b): f(x) = ax + b 직선 추가, 각 직선이 최대가 되는 구간 갱신
    - get_max(x): x에서의 최댓값 출력



Convex Hull Trick

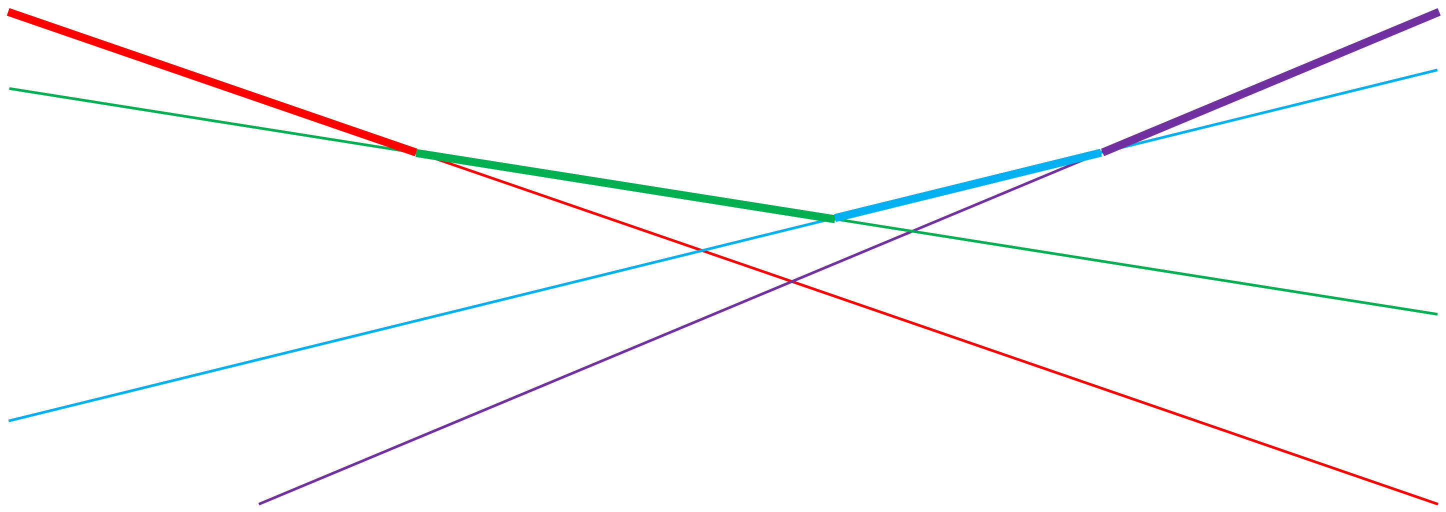
- Linear Time CHT
  - 제약 조건: $A_i \le A_{i+1}$, $x_j \le x_{j+1}$
  - 설명의 편의를 위해 $A_i < A_{i+1}$이라고 하자.

  - add line 연산
    - 새로 추가되는 직선 L은 기울기가 가장 큰 유일한 직선이므로 최대가 되는 구간 존재
    - L이 기존 직선의 구간을 가린다면, 기울기가 가장 큰 연속한 몇 개의 직선을 가림

Convex Hull Trick

- Linear Time CHT
  - 제약 조건: $A_i \le A_{i+1}$, $x_j \le x_{j+1}$
  - 설명의 편의를 위해 $A_i < A_{i+1}$이라고 하자.

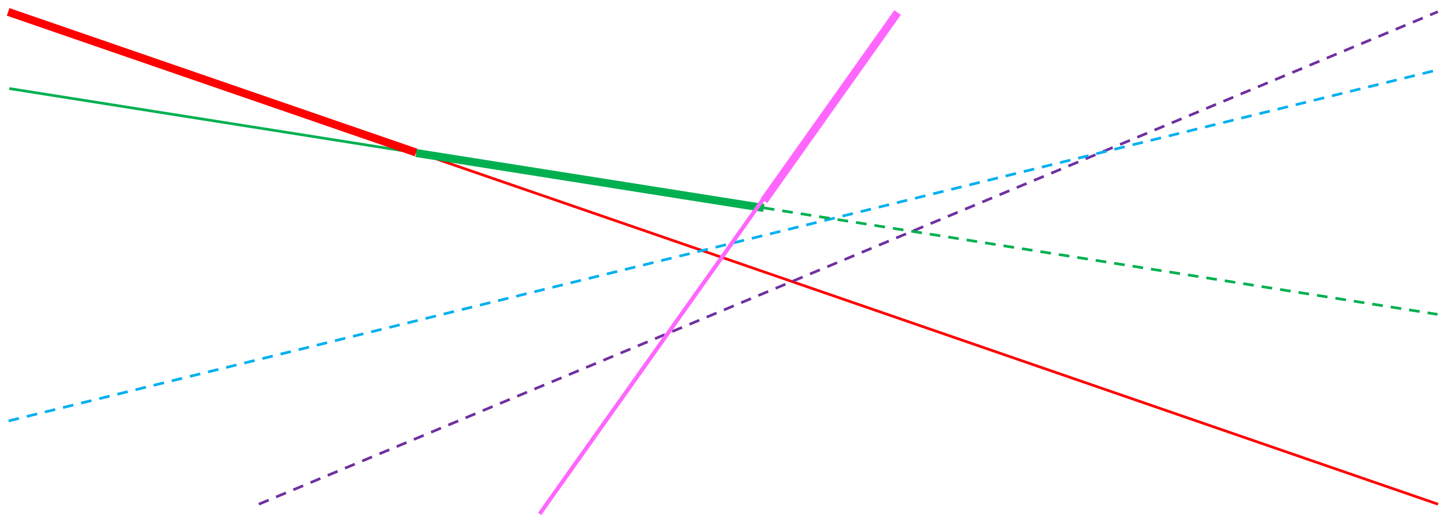
  - add line 연산
    - 새로 추가되는 직선 L은 기울기가 가장 큰 유일한 직선이므로 최대가 되는 구간 존재
    - L이 기존 직선의 구간을 가린다면, 기울기가 가장 큰 연속한 몇 개의 직선을 가림



Convex Hull Trick

- Linear Time CHT
  - 제약 조건: $A_i \le A_{i+1}$, $x_j \le x_{j+1}$
  - 설명의 편의를 위해 $A_i < A_{i+1}$이라고 하자.

  - add line 연산
    - 새로 추가되는 직선 L은 기울기가 가장 큰 유일한 직선이므로 최대가 되는 구간 존재
    - L이 기존 직선의 구간을 가린다면, 기울기가 가장 큰 연속한 몇 개의 직선을 가림



Convex Hull Trick

- Linear Time CHT
  - 제약 조건: $A_i \le A_{i+1}$, $x_j \le x_{j+1}$
  - 설명의 편의를 위해 $A_i < A_{i+1}$이라고 하자.

  - add line 연산
    - 새로 추가되는 직선 L은 기울기가 가장 큰 유일한 직선이므로 최대가 되는 구간 존재
    - L이 기존 직선의 구간을 가린다면, 기울기가 가장 큰 연속한 몇 개의 직선을 가림
    - k개의 직선이 가려진다고 하면, 뒤에 있는 k-1개는 완전히 가려지고 1개는 일부 또는 전부가 가려짐

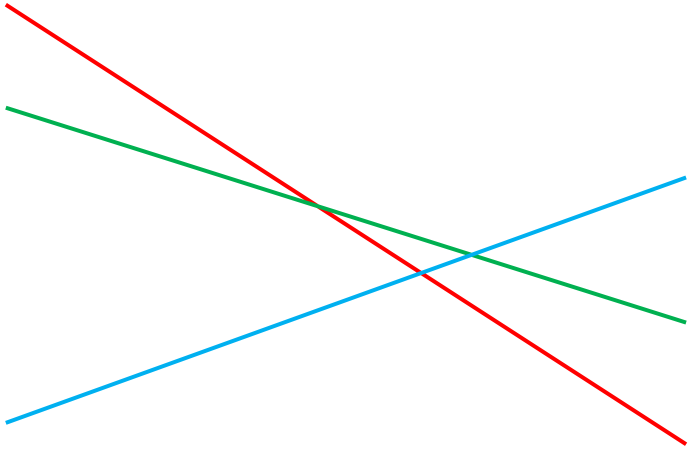
    - 최근에 추가된 직선부터 제거되니까 스택으로 관리할 수 있음

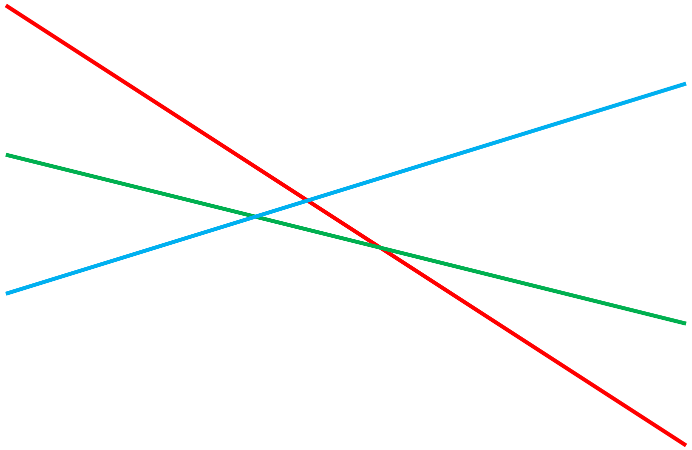
Convex Hull Trick

- Linear Time CHT
  - add line 연산
    - 빨간색: 스택 뒤에서 2번째 직선
    - 초록색: 스택 맨 뒤에 있는 직선
    - 하늘색: 새로 추가하는 직선

    - 빨간색 – 초록색 교점을 x_1, 초록색 – 하늘색 교점을 x_2라고 하자.
      - 초록색이 최대인 구간: $[x_1, x_2)$
      - 하늘색이 최대인 구간: $[x_2, \infty)$

    - 만약 $x_1 \ge x_2$ 이면 초록색은 최대가 될 수 없음





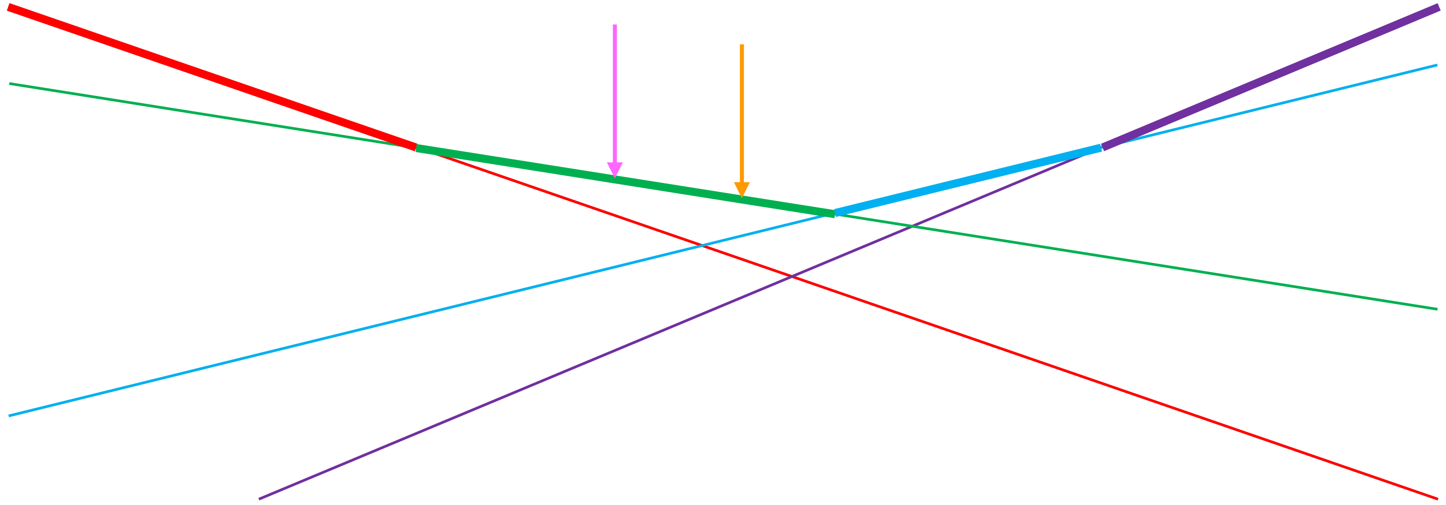
Convex Hull Trick

- Linear Time CHT
  - add line 연산

```
struct StackCHT{
    struct Line{
        ll a, b; // y = ax + b
        Line(ll a, ll b, ll c) : a(a), b(b) {}
        ll f(ll x){ return a * x + b; }
    };
    vector<Line> v;
    bool chk(const Line &a, const Line &b, const Line &c) const {
        // (a.b - b.b) / (b.a - a.a) >= (b.b - c.b) / (c.a - b.a)
        return (__int128_t)(a.b - b.b) * (c.a - b.a) >= (__int128_t)(b.b - c.b) * (b.a - a.a);
    }
    void add_line(Line l){
        while(v.size() >= 2 && chk(v[v.size()-2], v.back(), l)) v.pop_back();
        v.push_back(l);
    }
    ll get_max(ll x){
        // @TODO
    }
};
```

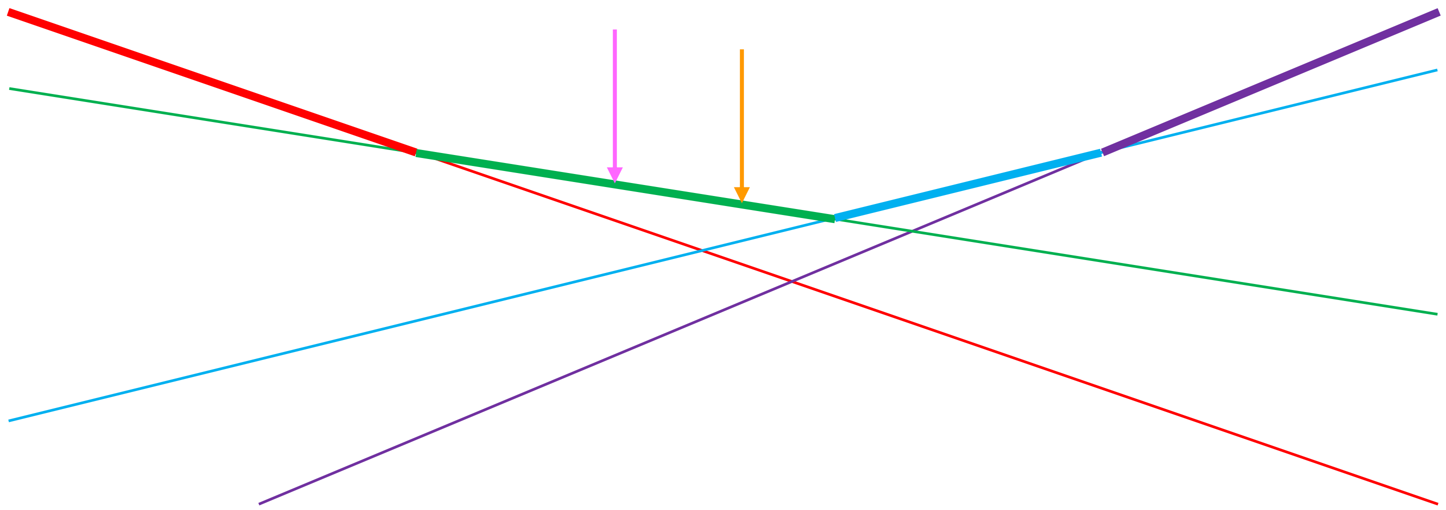
Convex Hull Trick

- Linear Time CHT
  - get max 연산
    - $x_j \le x_{j+1}$을 가정했으므로 구하고자 하는 직선의 기울기는 감소하지 않음
    - 이전 쿼리에서 정답이었던 직선(분홍색 화살표, p)과 x_j(주황색 화살표)를 생각해 보자.



Convex Hull Trick

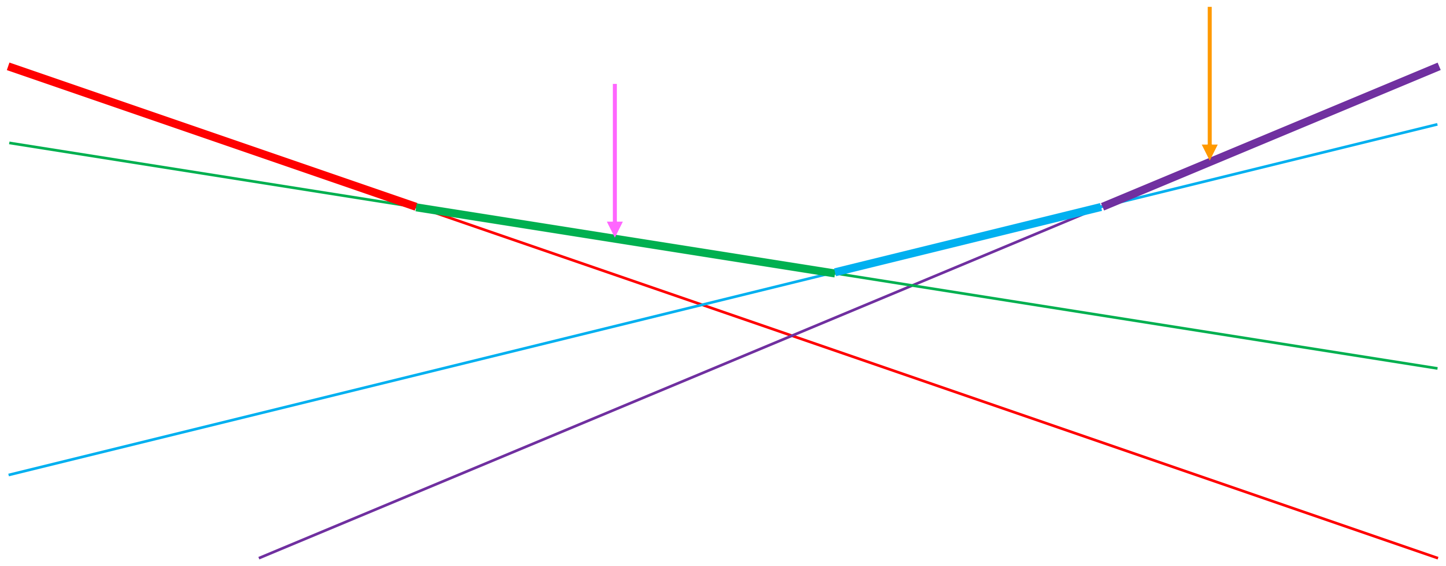
- Linear Time CHT
  - get max 연산
    - $x_j \leq x_{j+1}$을 가정했으므로 구하고자 하는 직선의 기울기는 감소하지 않음
    - 이전 쿼리에서 정답이었던 직선(분홍색 화살표, p)과 x_j(주황색 화살표)를 생각해 보자.
      - 만약 xj < p와 p+1의 교점이면 p번째 직선에서 최대



Convex Hull Trick

- Linear Time CHT
  - get max 연산
    - $x_j \le x_{j+1}$을 가정했으므로 구하고자 하는 직선의 기울기는 감소하지 않음

    - 이전 쿼리에서 정답이었던 직선(분홍색 화살표, p)과 x_j(주황색 화살표)를 생각해 보자.
      - 만약 xj < p와 p+1의 교점이면 p번째 직선에서 최대
      - xj ≥ p와 p+1의 교점이면 기울기가 더 큰 직선에서 최대



Convex Hull Trick

- Linear Time CHT
  - get max 연산
    - while(p + 1 < v.size() && x >= cross(p, p+1)) p++;
    - return v[p].f(x);

    - 단순히 이렇게 구현하면 add line 연산에서 직선을 제거할 때 문제가 발생할 수 있음
      - 직선을 많이 삭제해서 v의 크기가 p보다 작아지는 경우

      - 어차피 p 보다 앞에 있는 직선은 더 이상 신경 안 써도 되니까
      - p보다 앞에 있는 직선을 삭제하지 않으면 됨

Convex Hull Trick

```cpp
constexpr ll INF = 0xc0c0c0c0c0c0c0c0;
struct Line{
    ll a, b;
    Line() : Line(0, INF) {}
    Line(ll a, ll b) : a(a), b(b) {}
    ll f(ll x){ return a * x + b; }
};
struct StackCHT{
    vector<Line> v; int pv;
    StackCHT(){ clear(); }
    void clear(){ v.clear(); pv = 0; }
    bool chk(const Line &a, const Line &b, const Line &c) const {
        // (a.b - b.b) / (b.a - a.a) >= (b.b - c.b) / (c.a - b.a)
        return (__int128_t)(a.b - b.b) * (c.a - b.a) >= (__int128_t)(b.b - c.b) * (b.a - a.a);
    }
    bool chk(const Line &a, const Line &b, ll x) const {
        // x >= (a.b - b.b) / (b.a - a.a)
        return (__int128_t)x * (b.a - a.a) >= a.b - b.b;
    }
    void add(ll a, ll b){
        Line l(a, b);
        if(pv < v.size() && v.back().a == l.a){
            if(l.b < v.back().b) l = v.back();
            v.pop_back();
        }
        while(v.size() >= pv+2 && chk(v[v.size()-2], v.back(), l)) v.pop_back();
        v.push_back(l);
    }
    ll query(ll x){
        while(pv+1 < v.size() && chk(v[pv], v[pv+1], x)) pv++;
        return v[pv].f(x);
    }
};
```

Convex Hull Trick

- Linear Time CHT
  - $A_i \geq A_{i+1}$ 이고 최솟값을 구하고 싶다면 두 번째 chk 함수의 부등호 방향을 반대로 사용하면 됨
    - b.a – a.a와 c.a – b.a가 음수가 되기 때문
    - INF 값도 적당히 수정

  - 부등호 방향이 헷갈린다면…
    - 최댓값을 구할 때 while(v[pv].f(x) ≤ v[pv+1].f(x)) pv++;
    - 최솟값을 구할 때 while(v[pv].f(x) ≥ v[pv+1].f(x)) pv++;

질문?

Convex Hull Trick

- BOJ 13263 나무 자르기
  - 점화식을 세우는 건 쉬움
    - D[1] = 0
    - D[i] = min(B[j] * A[i] + D[j])
    - A[i] < A[i+1], B[j] > B[j+1]

  - Convex Hull Trick을 이용해 점화식 계산
    - min 내부에 있는 식에서 i는 상수 취급
    - 기울기가 B[j], y 절편이 D[j]인 일차 함수
    - 추가되는 직선들의 기울기가 감소하고, 최솟값을 구하고자 하는 x 좌표는 증가하므로 CHT 사용

Convex Hull Trick

```
ll N, A[101010], B[101010], D[101010];

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    cin >> N;
    for(int i=1; i<=N; i++) cin >> A[i];
    for(int i=1; i<=N; i++) cin >> B[i];

    StackCHT H;
    D[1] = 0; H.add(B[1], D[1]); // f_i(x) = B[i]x + D[i]
    for(int i=2; i<=N; i++){
        D[i] = H.query(A[i]); // D[i] = min{ B[j]A[i] + D[j] }
        H.add(B[i], D[i]);
    }
    cout << D[N];
}
```

질문?

Li Chao Tree

Li Chao Tree

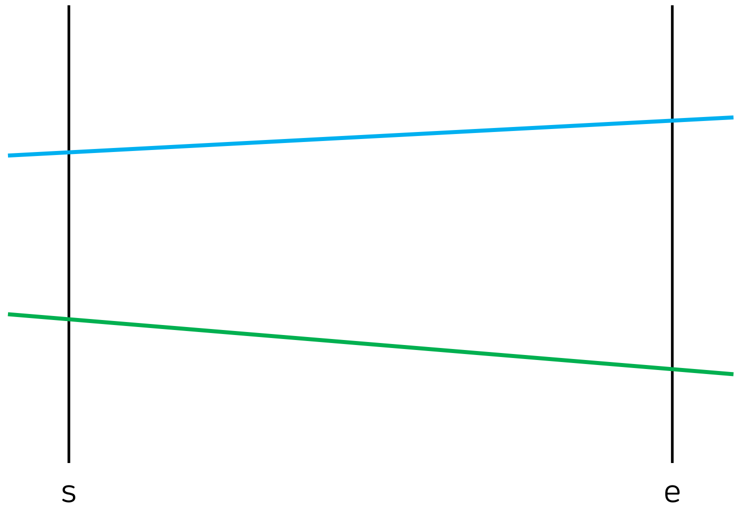
- Li Chao Tree
  - 다이나믹 세그먼트 트리에서 직선을 관리
    - [s, e]를 관리하는 정점에서는 조상에 있는 직선 제외하고 [s, e] 중 절반 이상에서 더 큰 직선을 저장
    - [s, e] 구간에서 최대가 될 수 있는 모든 직선은 그 정점의 조상 또는 자손에 저장되어 있는 상태

  - 어떤 지점 x에서의 최댓값을 구하는 방법
    - 루트 정점부터 시작해서 x를 나타내는 리프 정점까지 이동
    - 경로 상에 있는 정점들에 저장된 직선에서 f(x)의 최댓값

  - 직선 추가는 어떻게?

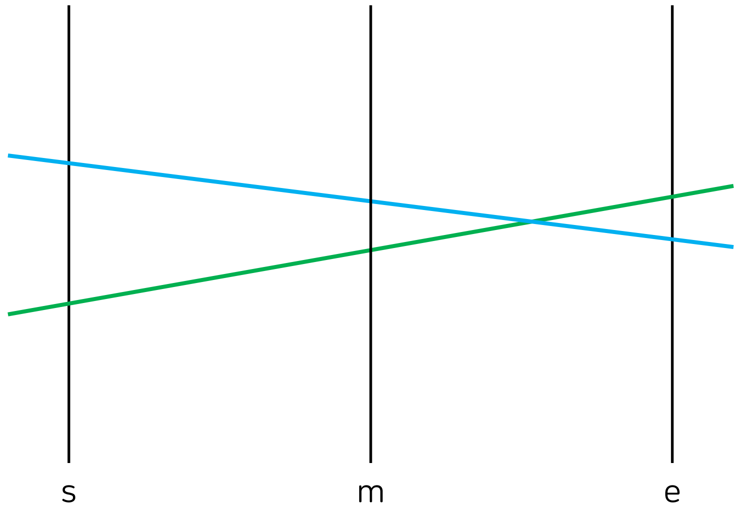
Li Chao Tree

- Li Chao Tree
  - 직선 추가
    - 원래 있던 직선과 새로 추가할 직선 중 왼쪽 끝 지점에서 더 작은 것을 low, 높은 것을 high라고 하자.
    - 만약 오른쪽 끝 지점에서도 low ≤ high 라면 정점에 high 저장하고 중단
      - [s, e] 구간에서 최대인 직선을 모두 저장했으므로 더 갱신할 필요 없음



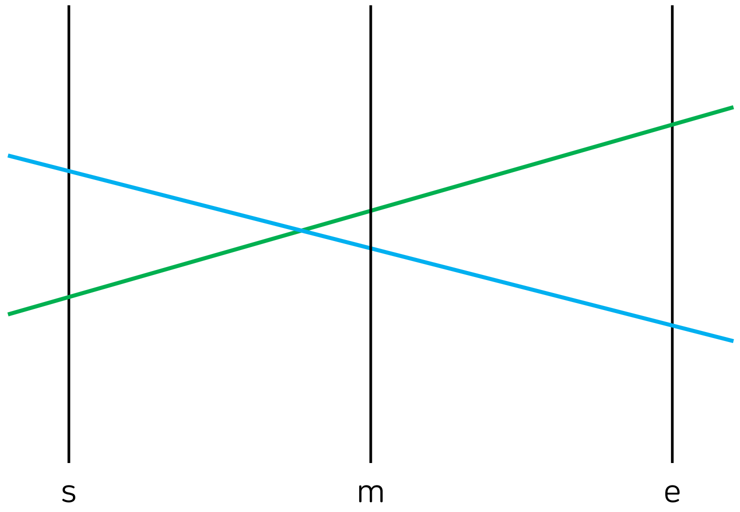
Li Chao Tree

- Li Chao Tree
  - 직선 추가
    - 원래 있던 직선과 새로 추가할 직선 중 왼쪽 끝 지점에서 더 작은 것을 low, 높은 것을 high라고 하자.
    - 두 직선에 [s, e]에서 교차한다면 중간 지점 m에서의 대소 관계를 본다.
      - 만약 m에서 low ≤ high 라면 절반 이상의 구간에서 high가 더 큰 상태
      - [s, m]에서는 항상 high가 크고 [m, e]에서는 경우에 따라 다름
      - 현재 정점에 high 저장하고 오른쪽 자식 정점에 low 갱신



Li Chao Tree

- Li Chao Tree
  - 직선 추가
    - 원래 있던 직선과 새로 추가할 직선 중 왼쪽 끝 지점에서 더 작은 것을 low, 높은 것을 high라고 하자.
    - 두 직선에 [s, e]에서 교차한다면 중간 지점 m에서의 대소 관계를 본다.
      - 만약 m에서 low ≥ high 라면 절반 이상의 구간에서 low가 더 큰 상태
      - [m, e]에서는 항상 low가 크고 [s, m]에서는 경우에 따라 다름
      - 현재 정점에 low 저장하고 왼쪽 자식 정점에 high 갱신



Li Chao Tree

```
constexpr int RANGE = 1 << 18;
constexpr ll INF = 0xc0c0c0c0c0c0c0c0;
struct Line{
    ll a, b;
    Line() : Line(0, INF) {}
    Line(ll a, ll b) : a(a), b(b) {}
    ll f(ll x) const { return a * x + b; }
};

struct LiChaoTree{
    struct Node{
        int l, r; Line v;
        Node() : l(-1), r(-1), v() {}
    };
    vector<Node> T;
    LiChaoTree(){ clear(); }
    void clear(){ T.clear(); T.emplace_back(); }
    void update(Line v, int node=0, int s=-RANGE, int e=+RANGE){
        Line lo = T[node].v, hi = v;
        if(lo.f(s) > hi.f(s)) swap(lo, hi);
        if(lo.f(e) <= hi.f(e)){ T[node].v = hi; return; }
        int m = (s + e) / 2;
        if(lo.f(m) < hi.f(m)){
            T[node].v = hi;
            if(T[node].r == -1) T[node].r = T.size(), T.emplace_back();
            update(lo, T[node].r, m+1, e);
        }
        else{
            T[node].v = lo;
            if(T[node].l == -1) T[node].l = T.size(), T.emplace_back();
            update(hi, T[node].l, s, m);
        }
    }
    void add(ll a, ll b){ update(Line(a, b)); }
    ll query(ll x, int node=0, int s=-RANGE, int e=+RANGE) const {
        ll res = node != -1 ? T[node].v.f(x) : INF;
        if(node == -1 || s == e) return res;
        int m = (s + e) / 2;
        if(x <= m) res = max(res, query(x, T[node].l, s, m));
        else res = max(res, query(x, T[node].r, m+1, e));
        return res;
    }
};
```

Li Chao Tree

- Li Chao Tree
  - 시간 복잡도
    - Update, Query: O(log X)
    - 범위가 X인 세그먼트 트리와 동일
  - 공간 복잡도
    - Update 마다 최대 1개의 정점을 추가로 생성
    - 따라서 전체 공간 복잡도는 O(N)
      - 무조건 리프 노드까지 내려가야 하는 일반적인 동적 세그먼트 트리와 다름
  - 제약 조건
    - 직선 삭제 불가능
    - 열심히 구현하면 마지막으로 추가한 직선은 삭제 가능

Li Chao Tree

- BOJ 12795 반평면 땅따먹기
  - Li Chao Tree 구현 연습 문제

  - Li Chao Tree 구현하기 귀찮고 팀 노트를 사용할 수 있는 상황이라면…
    - LineContainer 복붙
    - https://github.com/kth-competitive-programming/kactl/blob/main/content/data-structures/LineContainer.h

질문?

Hirschberg's Algorithm

Hirschberg’s Algorithm

- BOJ 18438 LCS 5
  - LCS를 구하는 문제
  - 인데 메모리 제한이 4MB

  - LCS의 “길이”는 토글링을 이용하면 시간 복잡도 O(NM), 공간 복잡도 O(N+M)에 구할 수 있음
  - 실제 LCS를 복원하는 건 어떻게 해야 할까?

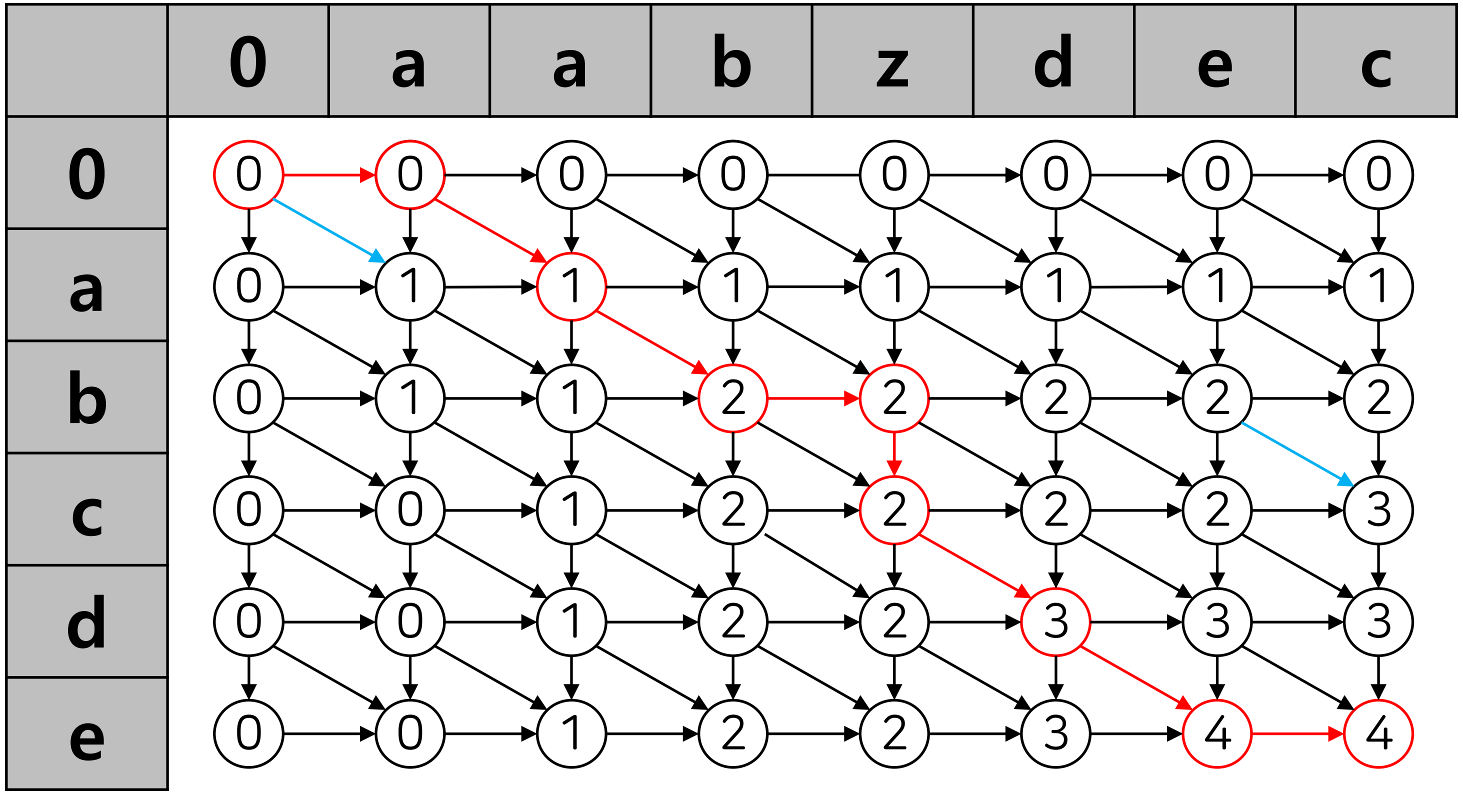
Hirschberg’s Algorithm

- BOJ 18438 LCS 5
  - 점화식을 다시 생각해 보자.
    - D(i, j) ← D(i-1, j)
    - D(i, j) ← D(i, j-1)
    - D(i, j) ← D(i-1, j-1) + (0 또는 1)

  - 오른쪽/아래/오른쪽 아래로 이동할 수 있는 격자 그래프
  - 평면 그래프라는 성질을 이용해 보자.

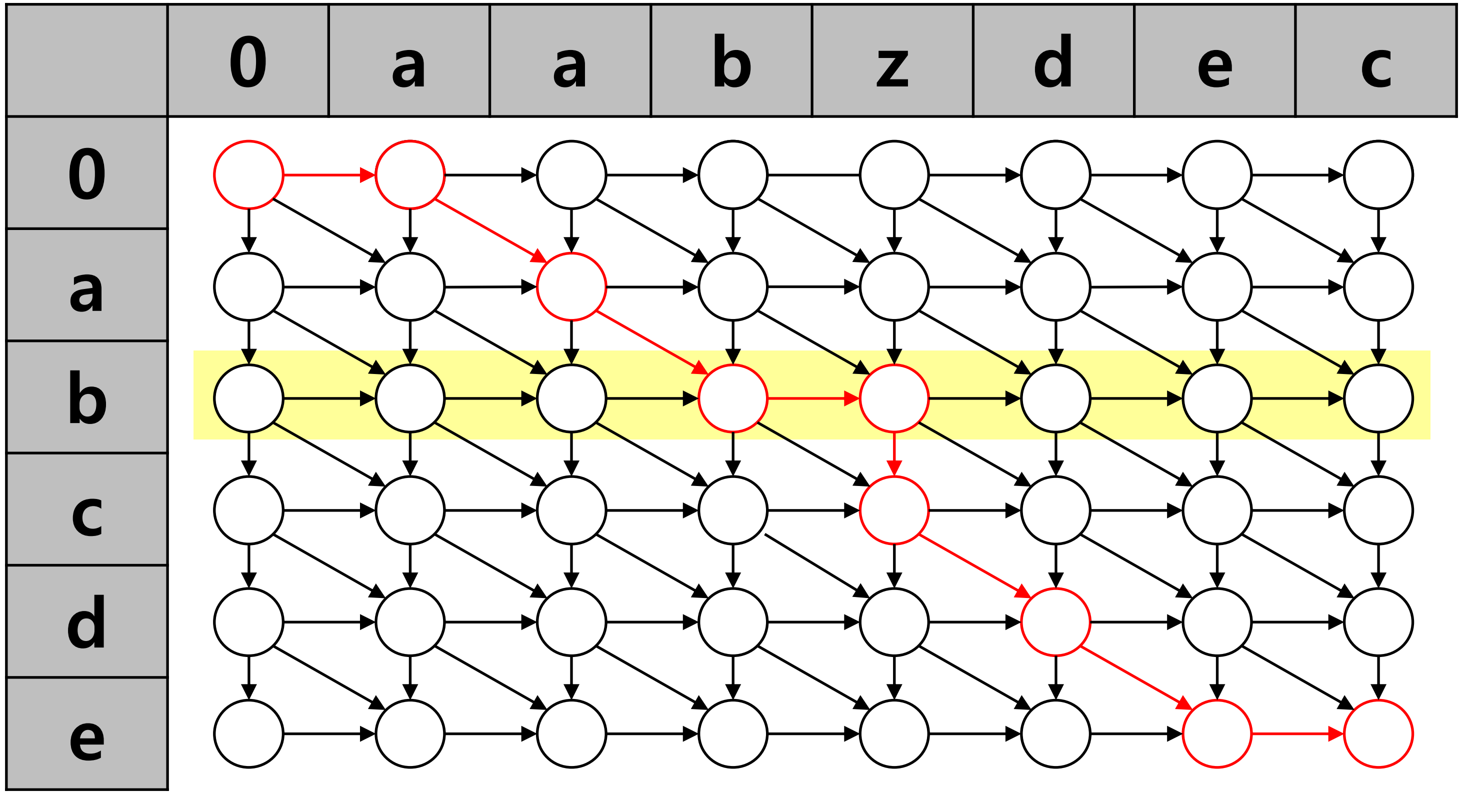
Hirschberg’s Algorithm

- BOJ 18438 LCS 5
  - 오른쪽/아래로 가는 가중치 0 간선, 대각선으로 가는 가중치 0 또는 1 간선
  - (0, 0)에서 (N, M)으로 가는 최장 경로를 구하는 문제



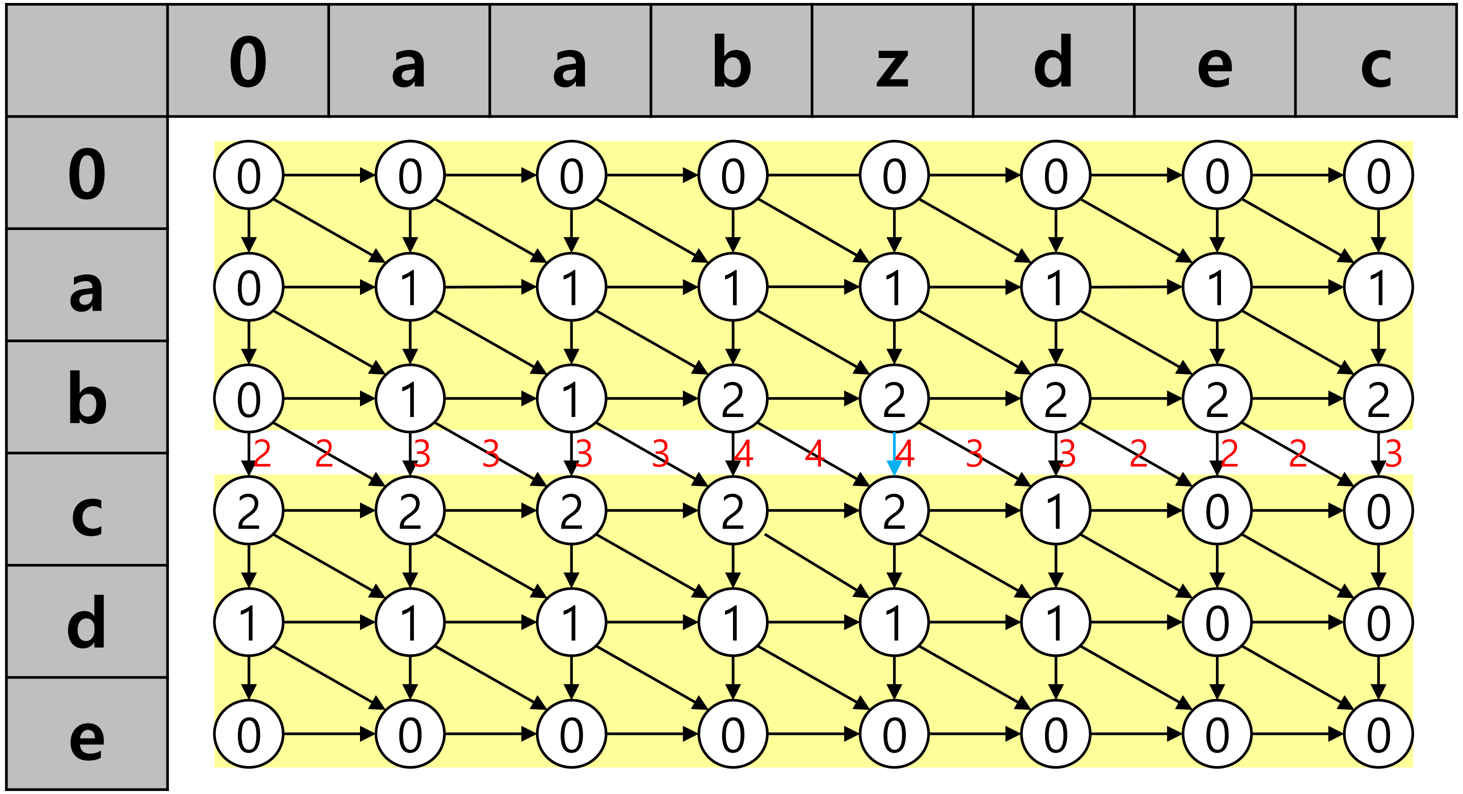
Hirschberg's Algorithm

- BOJ 18438 LCS 5
  - 임의의 i에 대해, (0, 0)에서 (N, M)으로 가는 모든 경로는 항상 i행의 어떤 정점 (i, j)를 지남
  - (0, 0) → (N, M) 경로를 (0, 0) → (i, j)와 (i, j) → (N, M)으로 분할할 수 있음



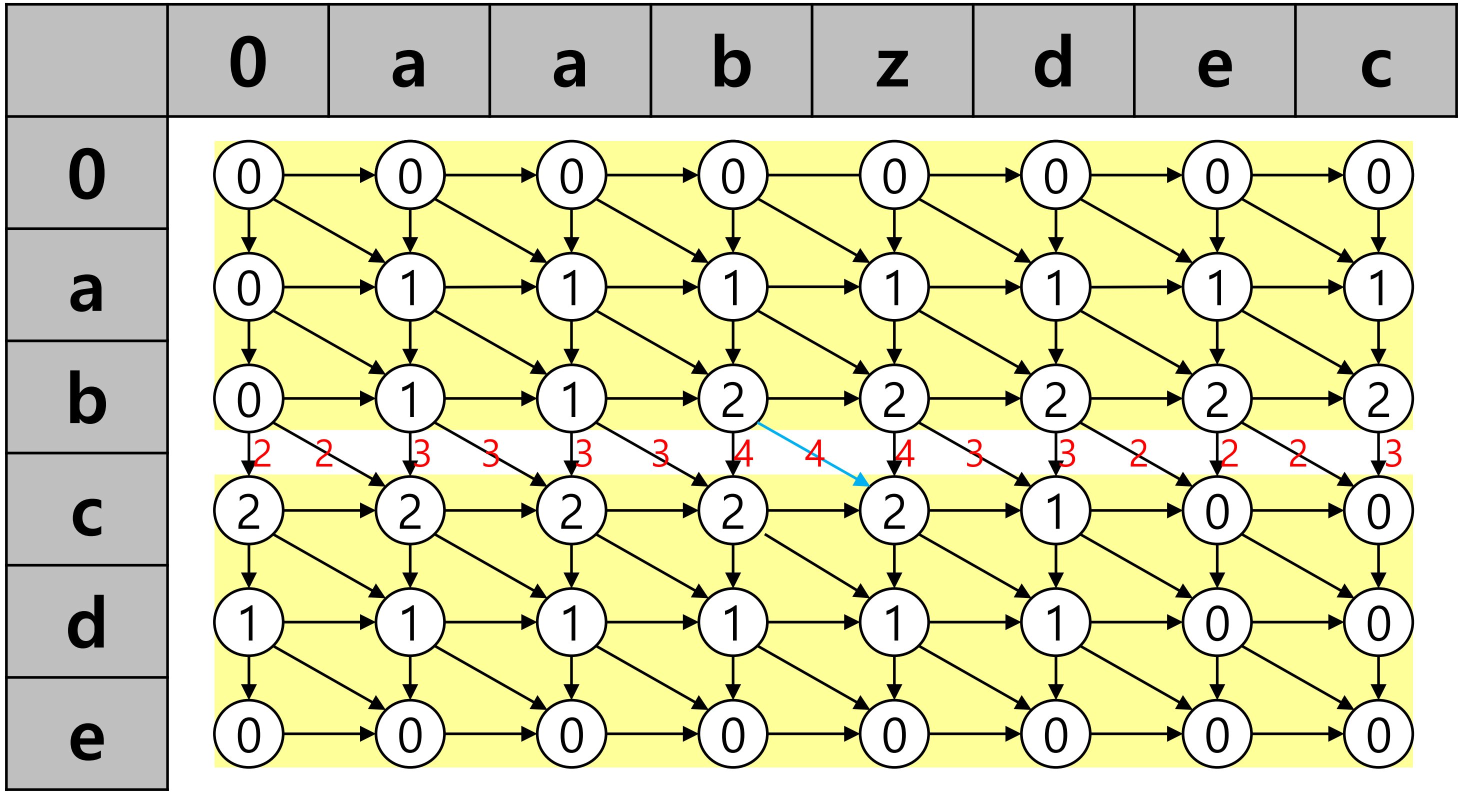
Hirschberg’s Algorithm

- BOJ 18438 LCS 5
  - 경로는 (0, 0) → (i, j) → (i+1, k) → (N, M) 꼴
  - (0, 0) → (i, j) → (i+1, k) → (N, M)의 거리가 최대가 되는 j와 k를 구하면 됨 (0 ≤ k-j ≤ 1)



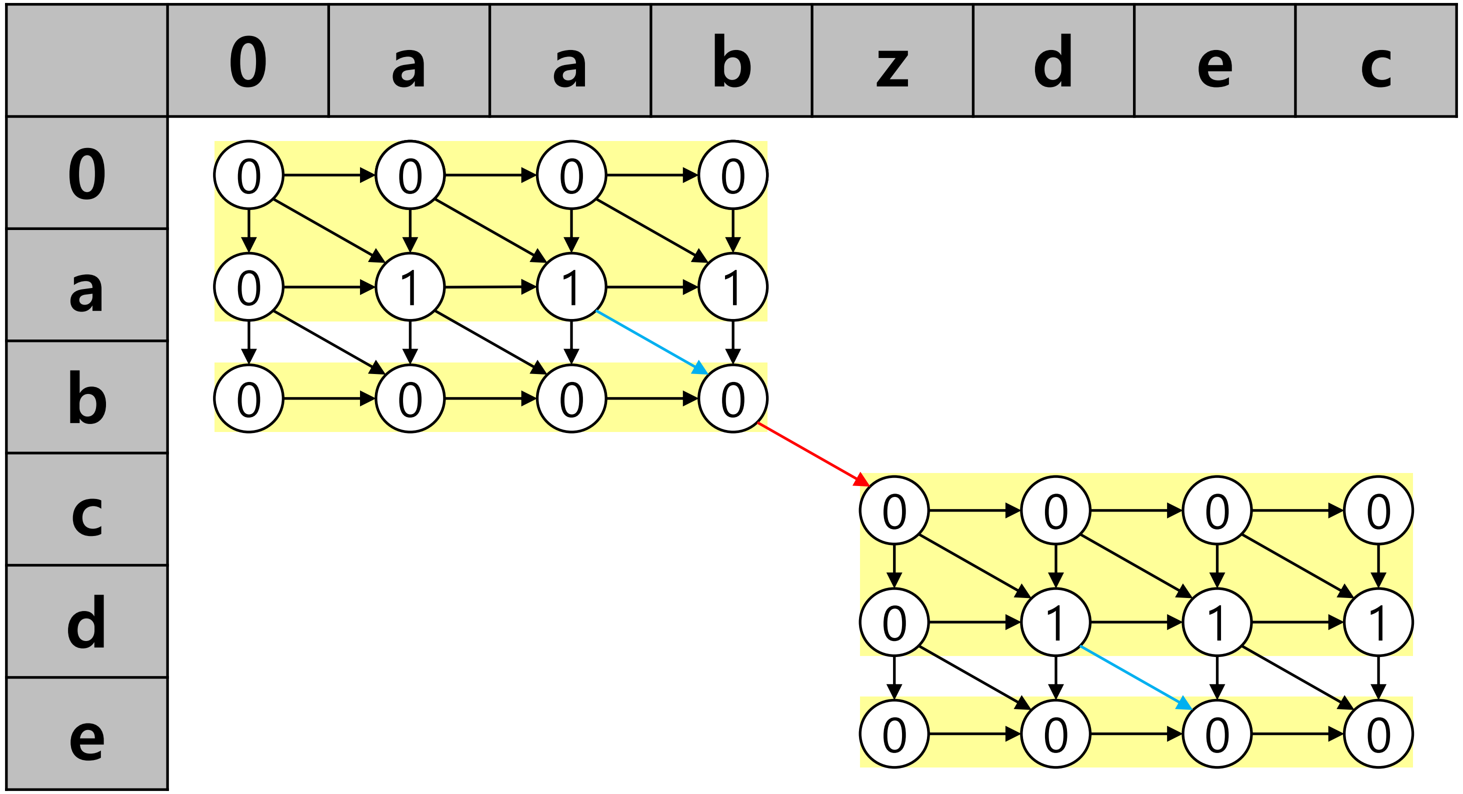
Hirschberg’s Algorithm

- BOJ 18438 LCS 5
  - 경로는 (0, 0) → (i, j) → (i+1, k) → (N, M) 꼴
  - (0, 0) → (i, j) → (i+1, k) → (N, M)의 거리가 최대가 되는 j와 k를 구하면 됨 (0 ≤ k-j ≤ 1)



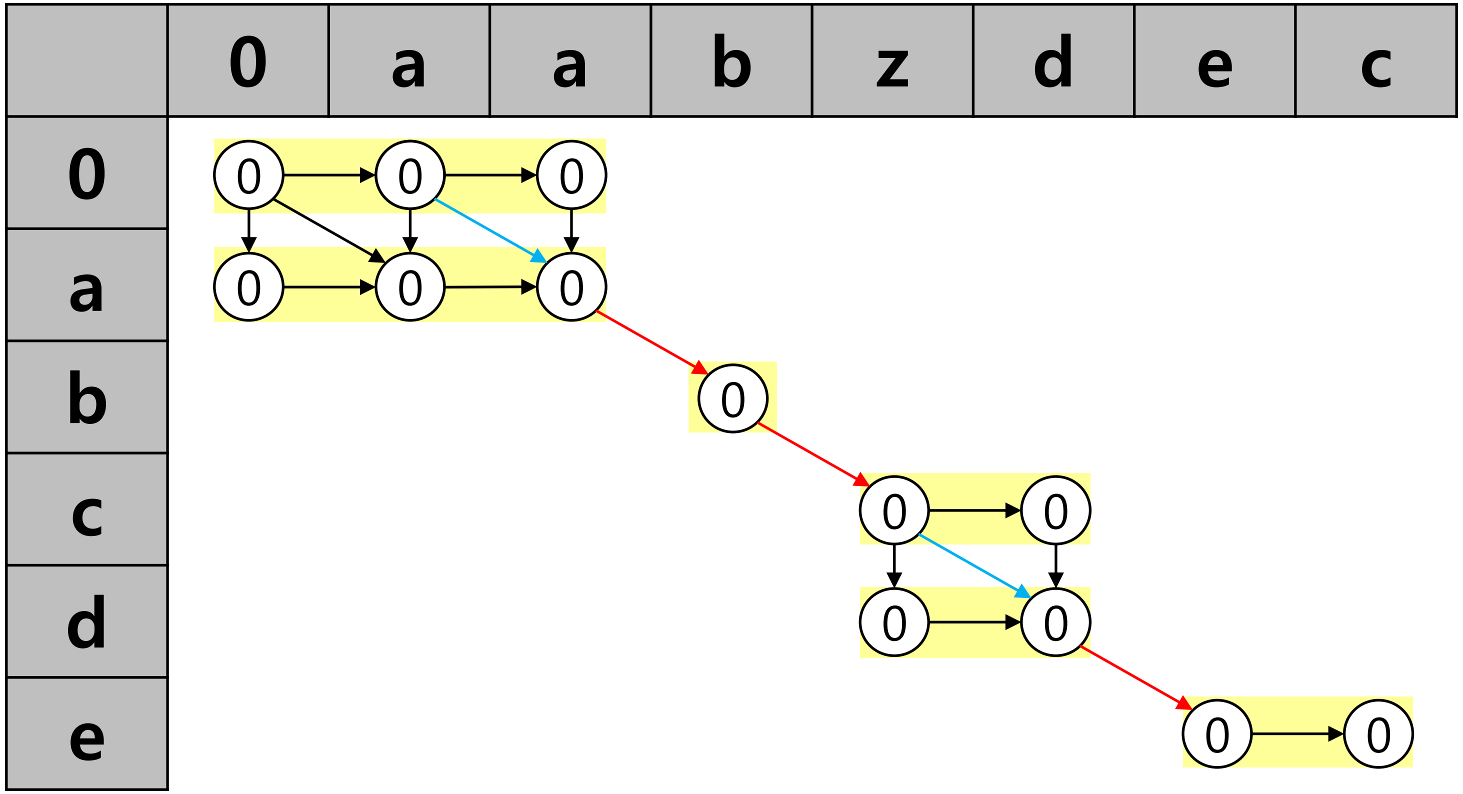
Hirschberg's Algorithm

- BOJ 18438 LCS 5
  - (i, j)에서 (i+1, k)로 가는 간선을 정답에 포함시키고
  - (0, 0) → (i, j)와 (i+1, k) → (N, M)은 동일한 방식으로 재귀적으로 계산



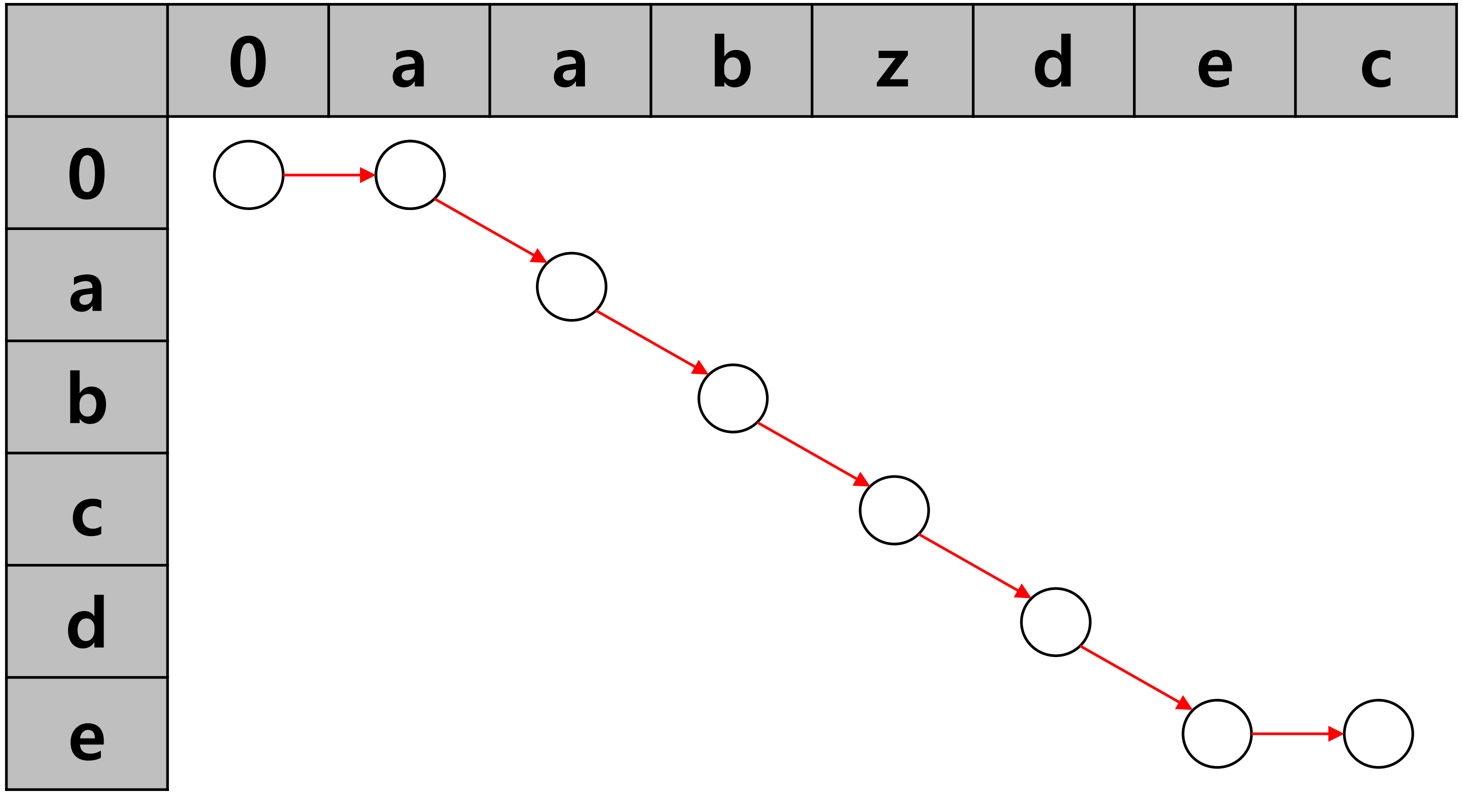
Hirschberg’s Algorithm

- BOJ 18438 LCS 5
  - (i, j)에서 (i+1, k)로 가는 간선을 정답에 포함시키고
  - (0, 0) → (i, j)와 (i+1, k) → (N, M)은 동일한 방식으로 재귀적으로 계산



Hirschberg's Algorithm

- BOJ 18438 LCS 5
  - (i, j)에서 (i+1, k)로 가는 간선을 정답에 포함시키고
  - (0, 0) → (i, j)와 (i+1, k) → (N, M)은 동일한 방식으로 재귀적으로 계산

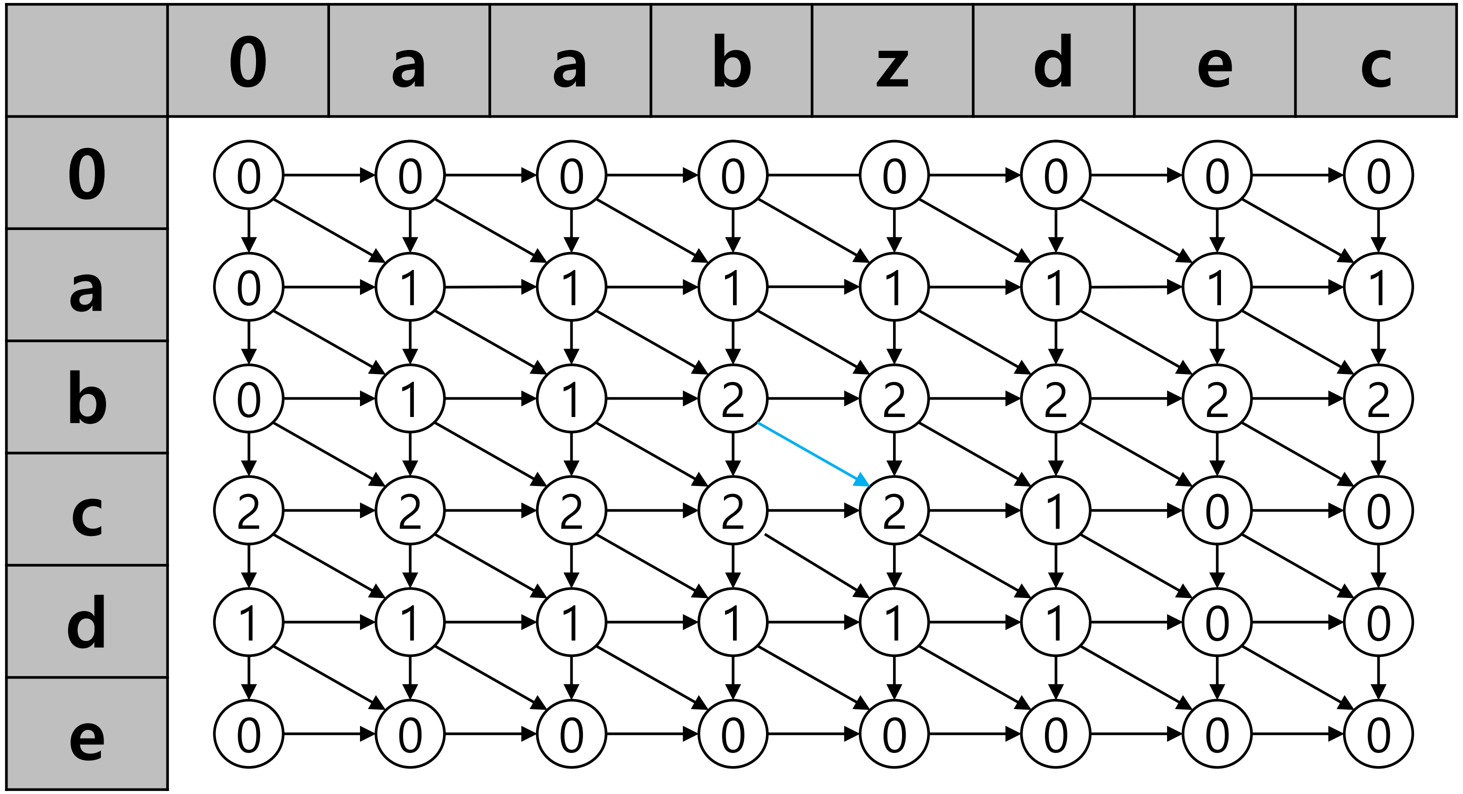


Hirschberg's Algorithm

- BOJ 18438 LCS 5
  - f(s, e, l, r) : (s, l)에서 (e, r)로 가는 최장 경로를 구하는 함수
    - A[s+1..e]와 B[l+1..r]의 LCS
    - n = e-s, m = r-l이라고 하자
  - 공간 복잡도
    - 입력 받은 문자열 O(N+M)
    - 함수 f에서 최단 거리를 계산하는데 O(m)
    - (i, j)에서 (i+1, k)로 가는 최적 간선을 찾는데 O(m)
  - 시간 복잡도
    - 함수 f에서 최단 거리를 계산하는데 O(nm)
    - (i, j)에서 (i+1, k)로 가는 최적 간선을 찾는데 O(m)
    - i를 s와 e의 중간 지점으로 잡으면?

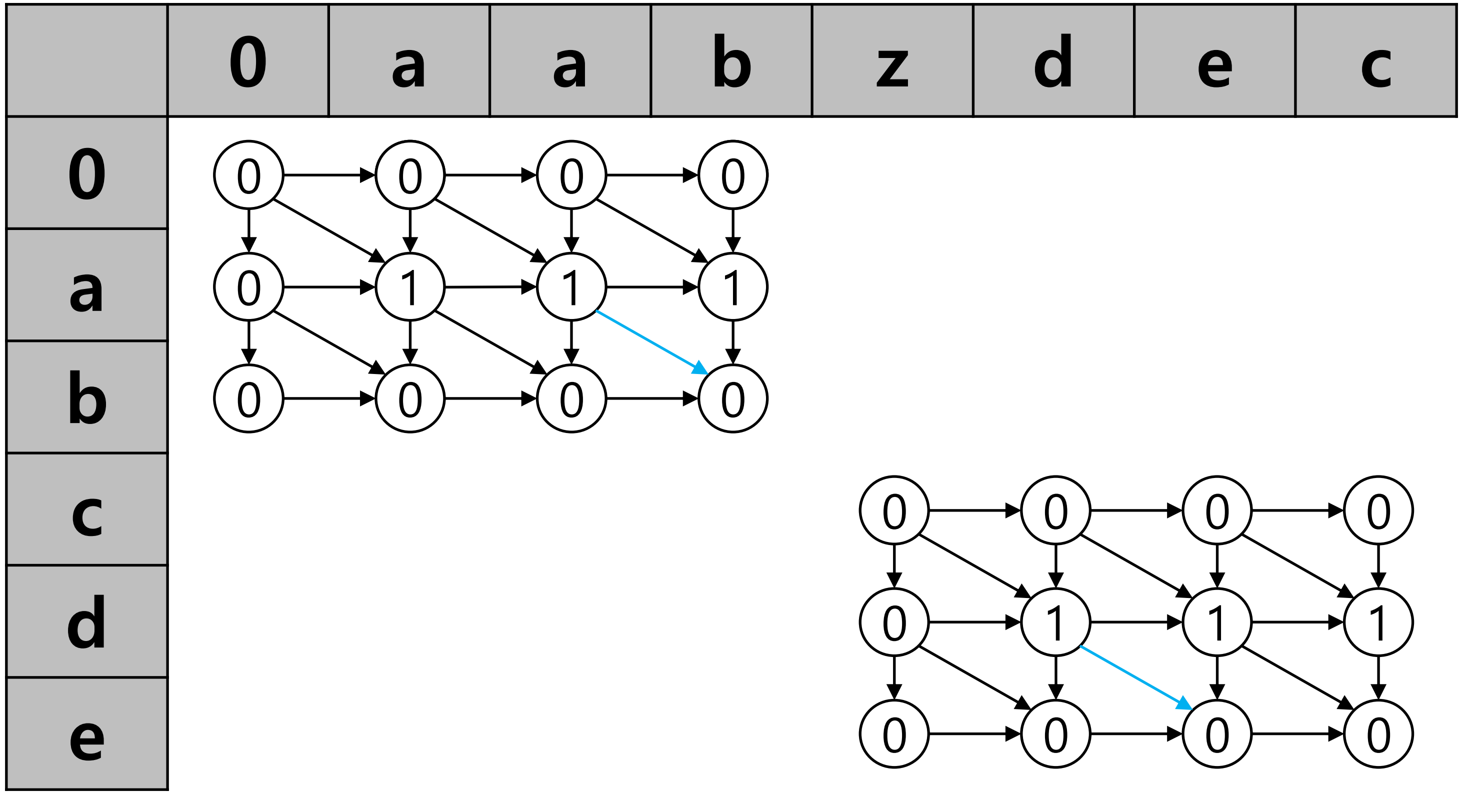
Hirschberg’s Algorithm

- BOJ 18438 LCS 5
  - NM



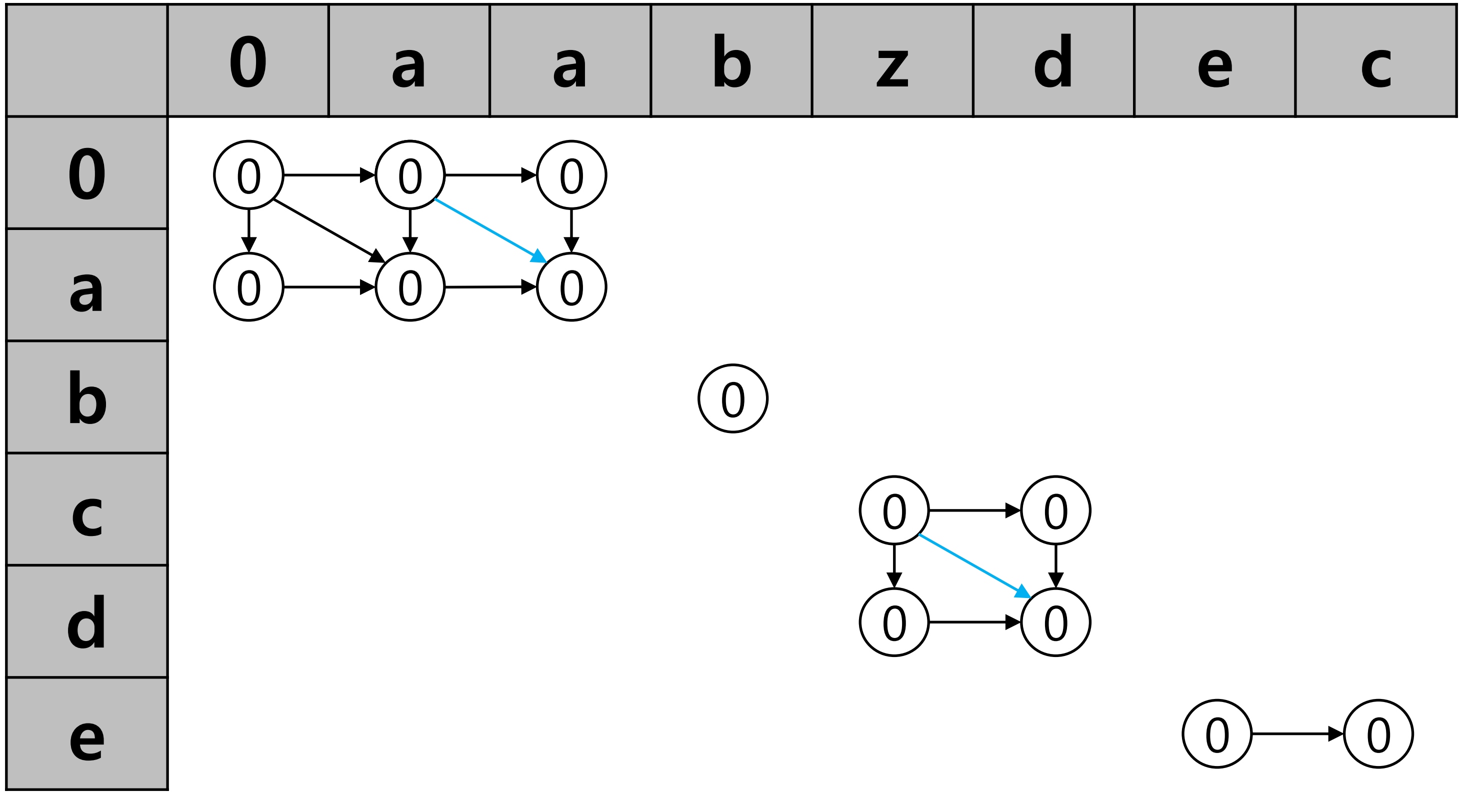
Hirschberg's Algorithm

- BOJ 18438 LCS 5
  - NM/2



Hirschberg's Algorithm

- BOJ 18438 LCS 5
  - NM/4



Hirschberg’s Algorithm

- BOJ 18438 LCS 5
  - 공간 복잡도
    - 입력 받은 문자열 O(N+M)
    - 함수 f에서 최단 거리를 계산하는데 O(m)
    - (i, j)에서 (i+1, k)로 가는 최적 간선을 찾는데 O(m)
    - 재귀 호출 깊이 O(log n)
    - 총 O(N+M)
  - 시간 복잡도
    - 함수 f에서 최단 거리를 계산하는데 O(nm)
    - (i, j)에서 (i+1, k)로 가는 최적 간선을 찾는데 O(m)
    - 총 NM + NM/2 + NM/4 + NM/8 + … = 2NM = O(NM)

Hirschberg’s Algorithm

```cpp
string Solve(const string &a, const string &b, int s, int e, int l, int r){
    int n = e - s, m = r - l, mid = (s + e) / 2;
    if(n <= 0 || m <= 0) return "";
    if(n == 1) return b.substr(l, m).find(a[s]) != string::npos ? string(1, a[s]) : "";
    if(m == 1) return a.substr(s, n).find(b[l]) != string::npos ? string(1, b[l]) : "";

    vector<vector<int>> dp(2, vector<int>(m+1));
    vector<int> dp_frt, dp_bck;

    for(int i=0; i<=m; i++) dp[0][i] = 0;
    for(int i=1; i<=mid-s; i++){
        dp[i&1][0] = dp[i-1&1][0];
        for(int j=1; j<=m; j++){
            if(a[s+i-1] == b[l+j-1]) dp[i&1][j] = dp[i-1&1][j-1] + 1;
            else dp[i&1][j] = max(dp[i-1&1][j], dp[i&1][j-1]);
        }
    }
    dp_frt = dp[mid-s&1];

    for(int i=0; i<=m; i++) dp[n&1][i] = 0;
    for(int i=n-1; i>mid-s; i--){
        dp[i&1][m] = dp[i+1&1][m];
        for(int j=m-1; j>=0; j--){
            if(a[s+i] == b[l+j]) dp[i&1][j] = dp[i+1&1][j+1] + 1;
            else dp[i&1][j] = max(dp[i+1&1][j], dp[i&1][j+1]);
        }
    }
    dp_bck = dp[mid-s+1&1];

    int mx = -1, idx1 = -1, idx2 = -1;
    char op = -1;
    for(int i=0; i<=m; i++){
        if(dp_frt[i] + dp_bck[i] > mx){
            mx = dp_frt[i] + dp_bck[i];
            idx1 = i; idx2 = i; op = -1;
        }
        if(i < m && a[mid] == b[l+i] && dp_frt[i] + dp_bck[i+1] + 1 > mx){
            mx = dp_frt[i] + dp_bck[i+1] + 1;
            idx1 = i; idx2 = i + 1; op = a[mid];
        }
    }

    auto le = Solve(a, b, s, mid, l, l+idx1), ri = Solve(a, b, mid+1, e, l+idx2, r);
    if(op == -1) return le + ri;
    else return le + op + ri;
}
```

질문?