Geometry 2

나정휘
https://justiceHui.github.io/

목차

- 계산 기하
- 2차원 기하 물체의 표현
- CCW
- 선분 교차 판별
- 다각형 내부 판별
- 볼록 껍질
- 볼록 다각형 내부 판별
- 가장 먼 두 점
- 가장 가까운 두 점
- 볼록 다각형의 접선을 이용한 최적화

복습

- 지난 시간에 한 것
  - 점 — 벡터로 표현
  - CCW — 두 벡터의 방향, 외적의 부호로 판별
  - 선분 교차 — CCW 4번으로 판별 가능
  - 다각형 내부 판별 — 반직선과 다각형의 교점 개수, O(N)번의 선분 교차
  - 볼록 껍질 — 모든 점을 포함하는 가장 작은 볼록 다각형, Graham’s scan O(N log N)
  - 볼록 다각형 내부 판별 — 교점은 최대 2개, CCW 3 + $\log_2 N$번으로 판별 가능

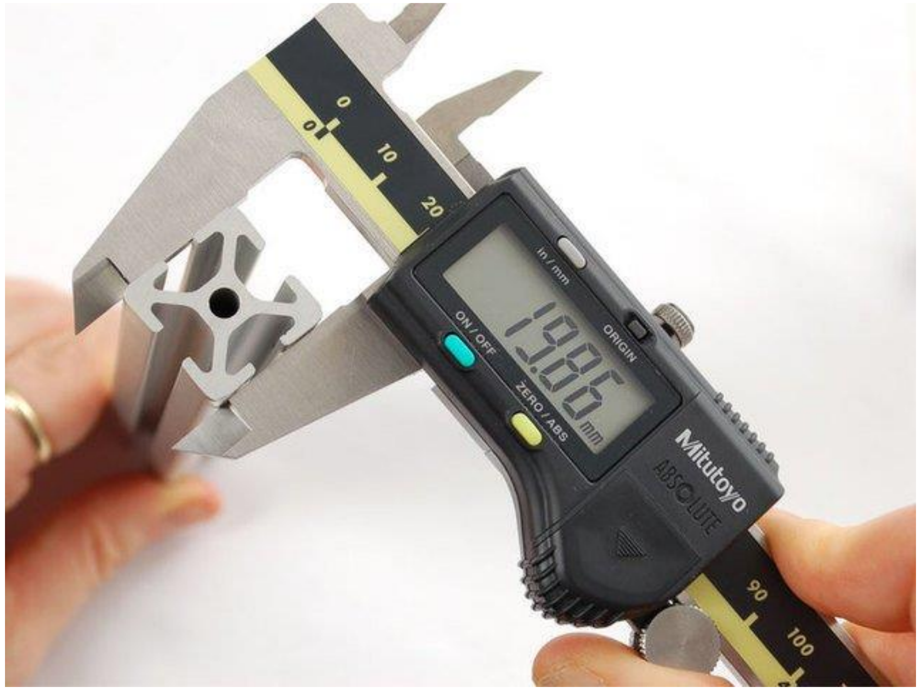
가장 먼두점

가장 먼 두 점

- 가장 먼 두 점은 항상 볼록 껍질의 꼭짓점
  - 일단 볼록 껍질을 구하고 생각하자.

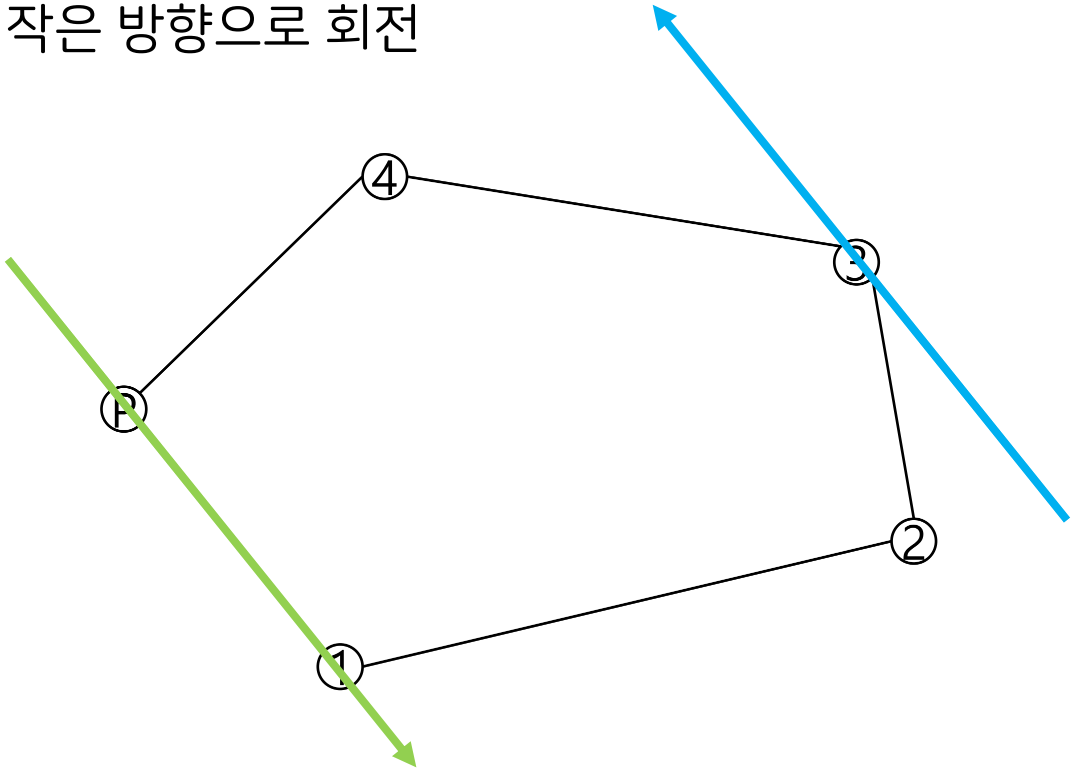
가장 먼 두 점

- 가장 먼 두 점은 항상 볼록 껍질의 꼭짓점
  - 일단 볼록 껍질을 구하고 생각하자.
- 버니어 캘리퍼스 : 길이를 정밀하게 측정하는 자의 일종



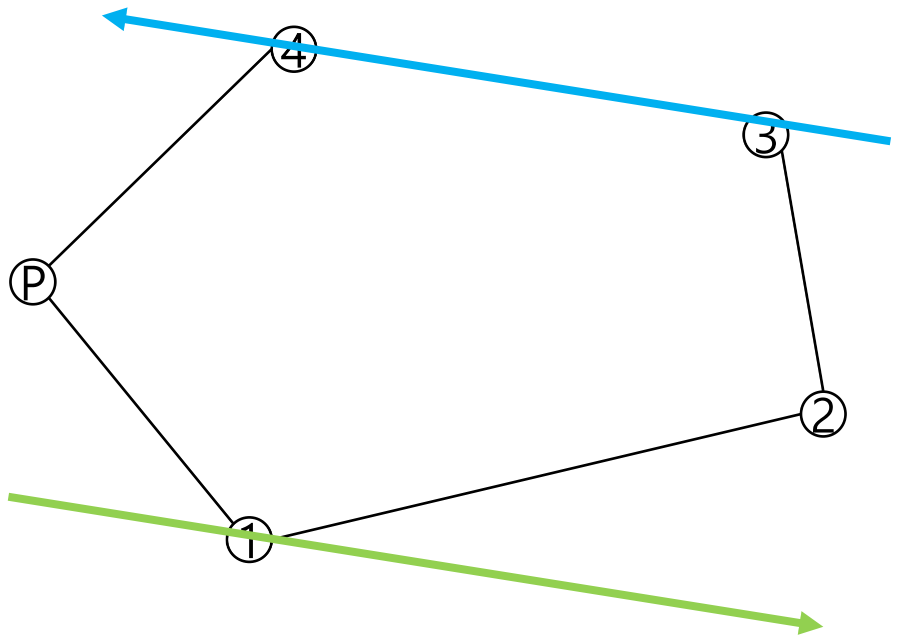
가장 먼 두 점

- 모든 변의 각도에서 캘리퍼스를 이용해 거리를 재면 그중 최댓값이 가장 먼 두 점
  - 둘 중 각도가 더 작은 방향으로 회전



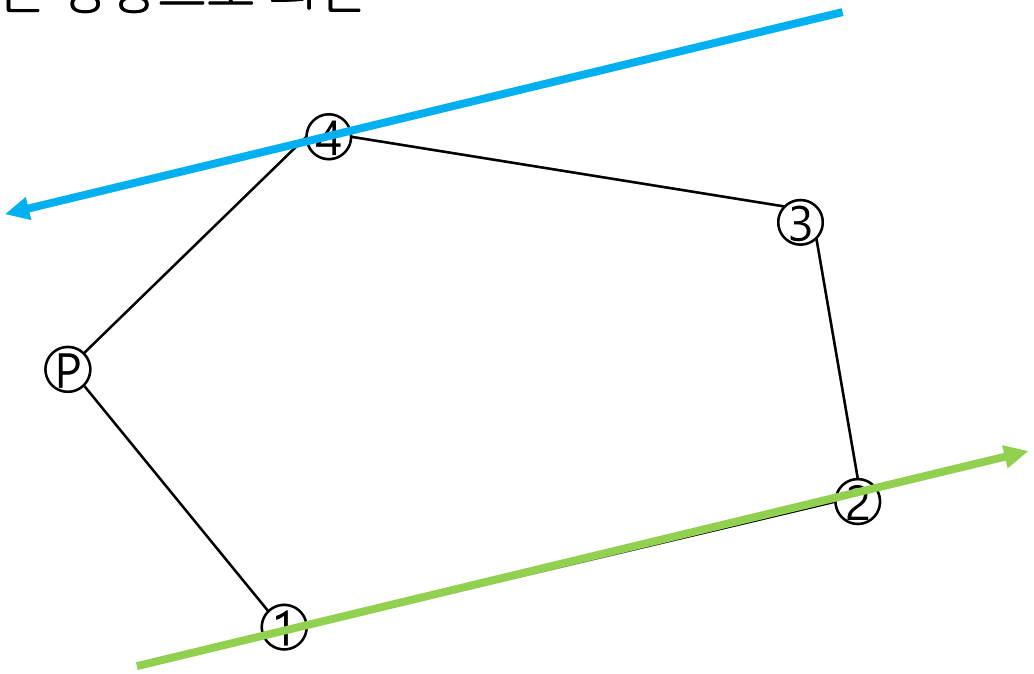
가장 먼 두 점

- 모든 변의 각도에서 캘리퍼스를 이용해 거리를 재면 그중 최댓값이 가장 먼 두 점
  - 둘 중 각도가 더 작은 방향으로 회전



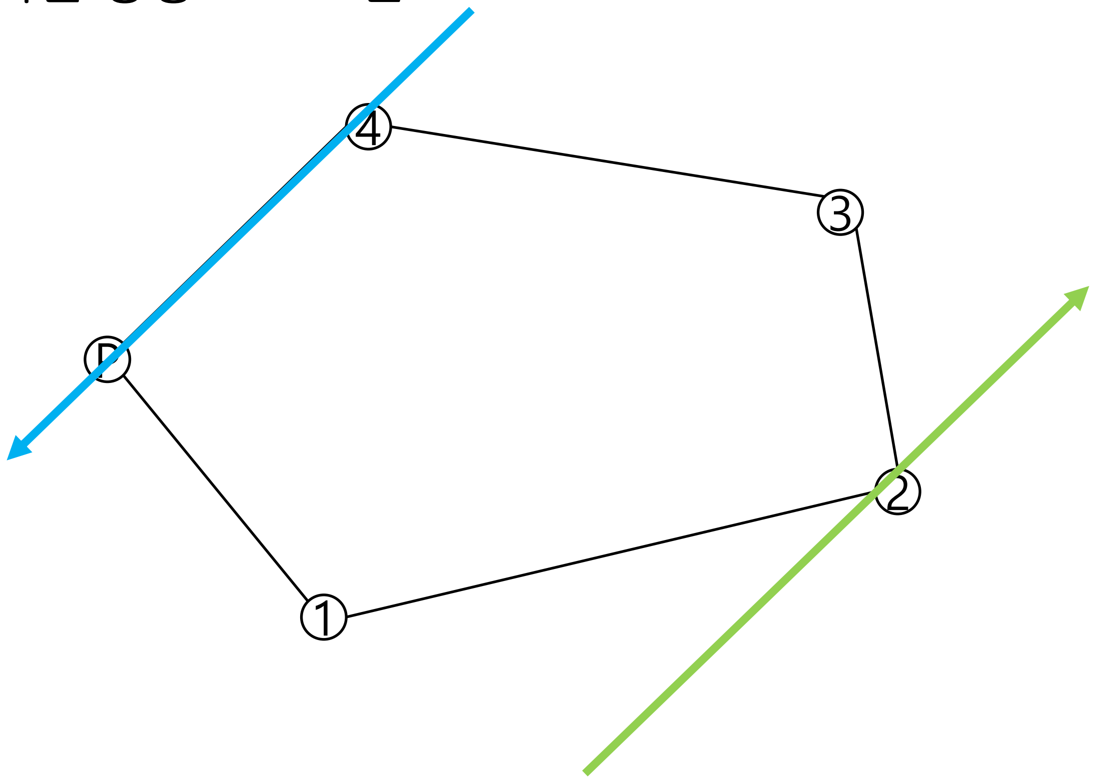
가장 먼 두 점

- 모든 변의 각도에서 캘리퍼스를 이용해 거리를 재면 그중 최댓값이 가장 먼 두 점
  - 둘 중 각도가 더 작은 방향으로 회전



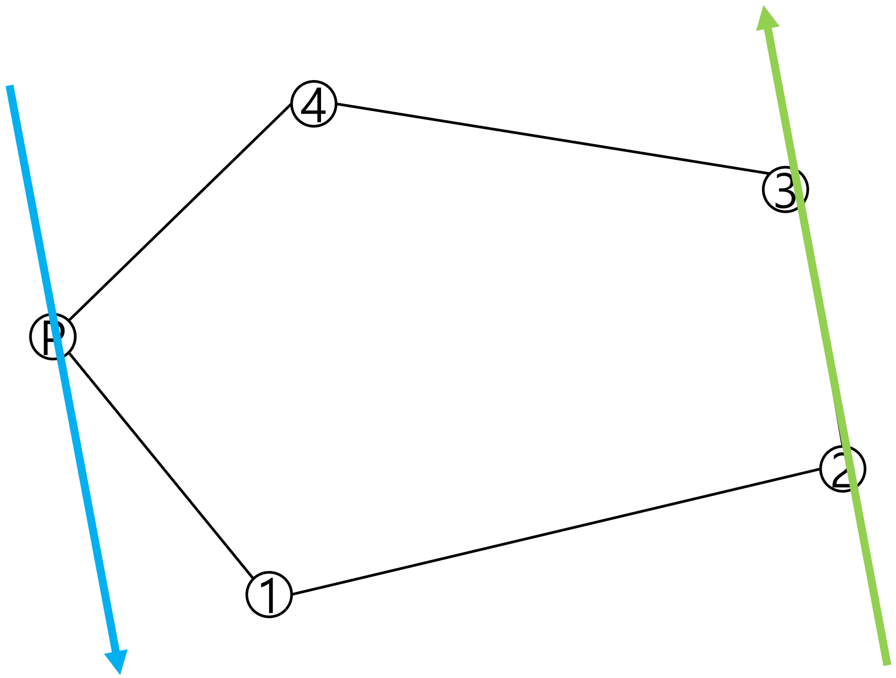
가장 먼 두 점

- 모든 변의 각도에서 캘리퍼스를 이용해 거리를 재면 그중 최댓값이 가장 먼 두 점
  - 둘 중 각도가 더 작은 방향으로 회전



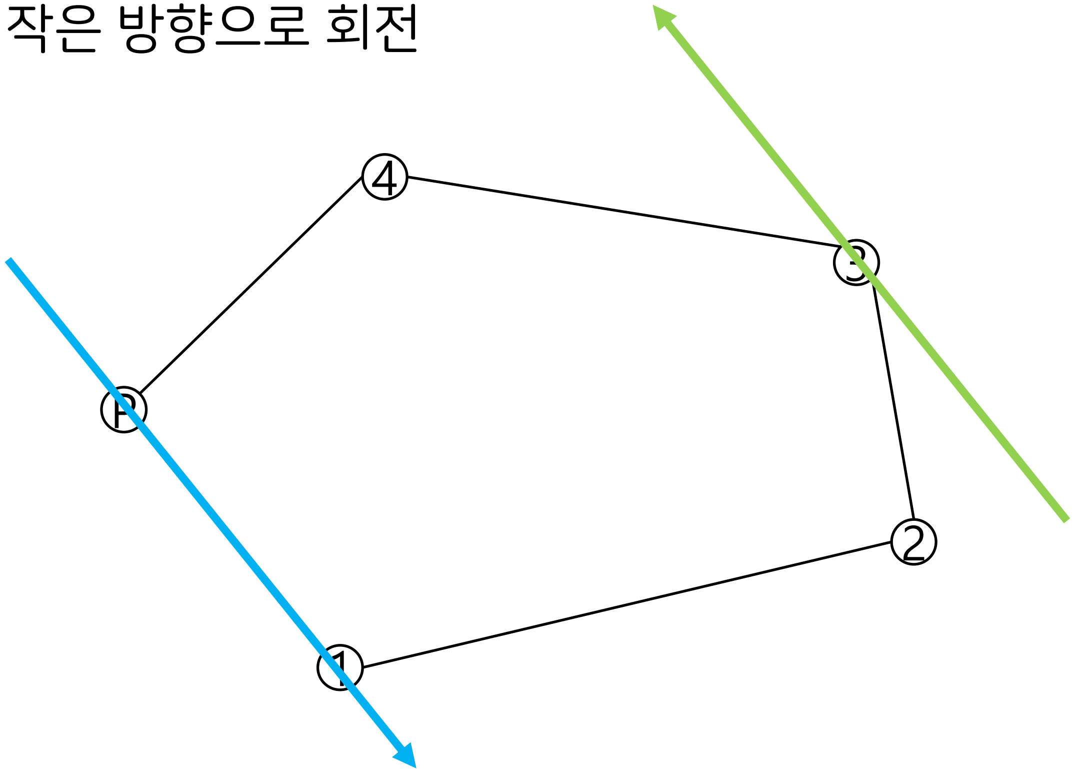
가장 먼 두 점

- 모든 변의 각도에서 캘리퍼스를 이용해 거리를 재면 그중 최댓값이 가장 먼 두 점
  - 둘 중 각도가 더 작은 방향으로 회전



가장 먼 두 점

- 모든 변의 각도에서 캘리퍼스를 이용해 거리를 재면 그중 최댓값이 가장 먼 두 점
  - 둘 중 각도가 더 작은 방향으로 회전



가장 먼 두 점

- 문제점
  - 특정 각도에서 극단에 있는 점을 어떻게 찾지?
  - 실수형으로 각도를 정확하게 비교할 수 있을까?

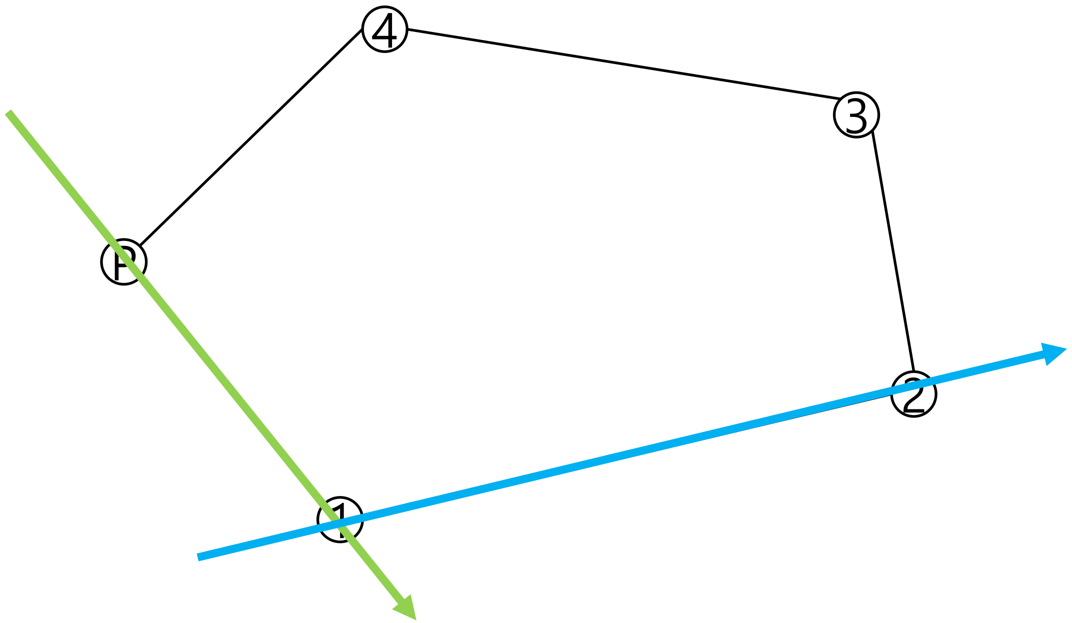
- 캘리퍼스에 닿는 두 점은 각자 한쪽 방향으로만 회전함 → 투 포인터
- 각도를 직접 계산하지 않고 비교할 수 있음 → CCW

가장 먼 두 점

- Rotating Calipers
  - 0번 점을 A, 1번 점을 C라고 하자.
  - A의 다음 점을 B, C의 다음 점을 D라고 하자.
  - A와 C의 거리를 정답에 반영
  - 벡터 AB와 CD가 정반대에 가까워지도록 A 또는 C를 한 칸 이동
    - 벡터 AB와 CD의 외적이 0보다 크거나 같으면 C를 이동
  - A가 0번 점으로 돌아올 때까지 반복

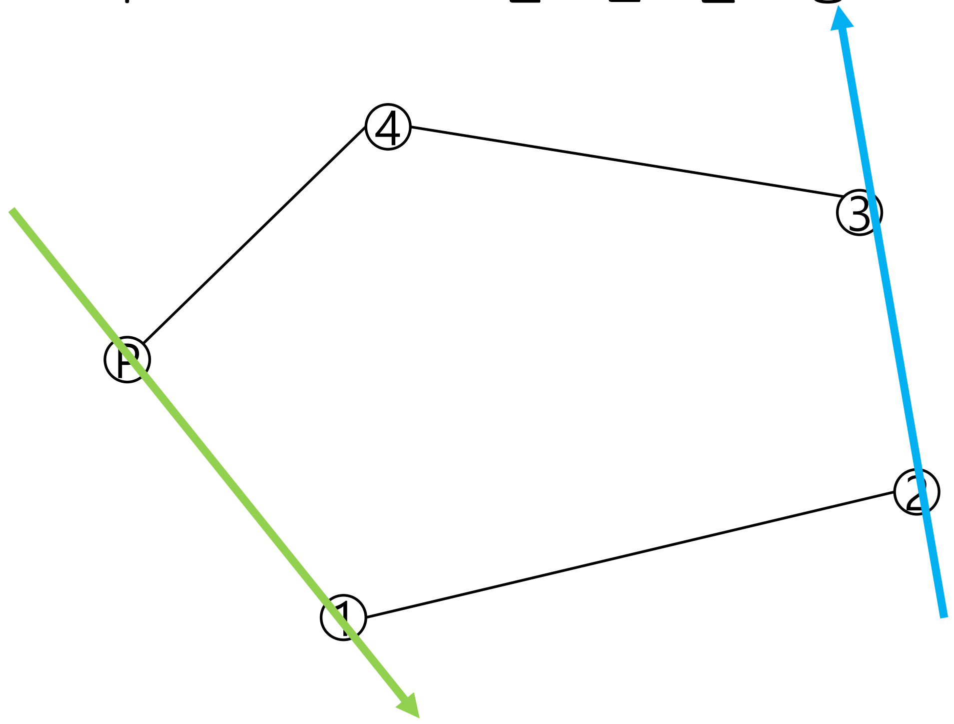
가장 먼 두 점

- 벡터 AB와 CD가 정반대에 가까워지도록 A 또는 C를 한 칸 이동
  - 벡터 AB와 CD의 외적이 0보다 크거나 같으면 C를 이동



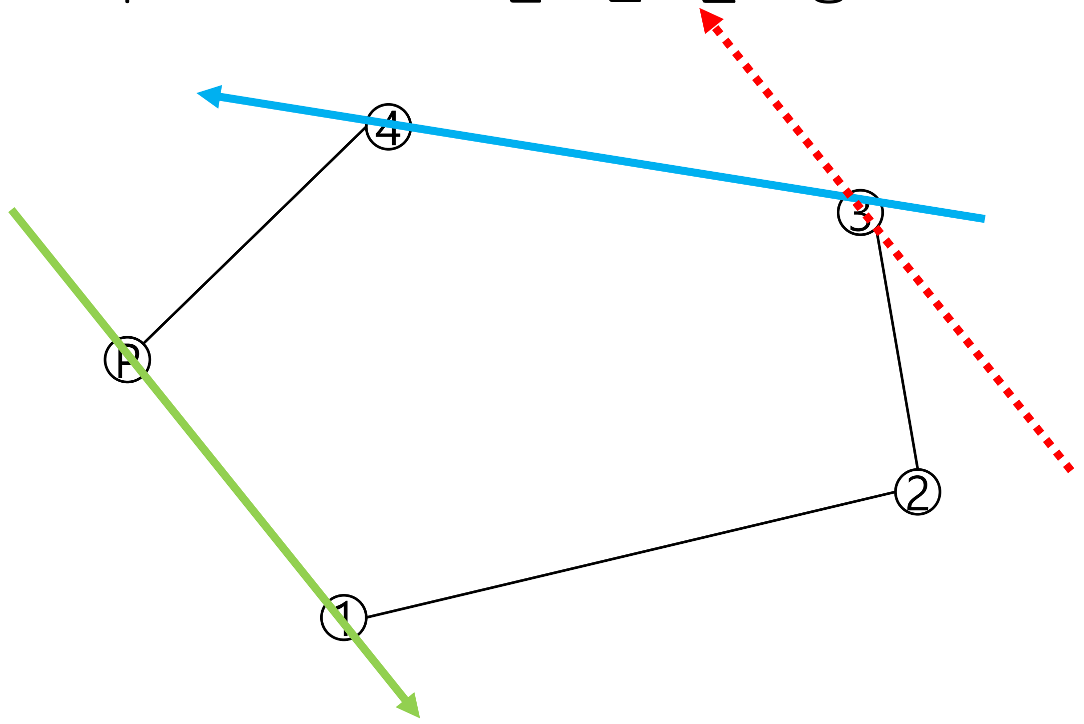
가장 먼 두 점

- 벡터 AB와 CD가 정반대에 가까워지도록 A 또는 C를 한 칸 이동
  - 벡터 AB와 CD의 외적이 0보다 크거나 같으면 C를 이동



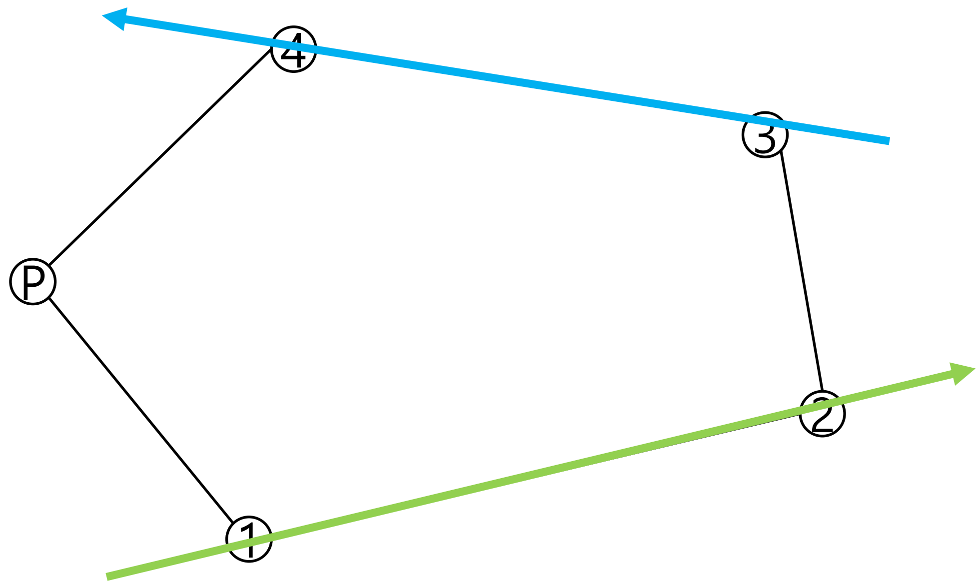
가장 먼 두 점

- 벡터 AB와 CD가 정반대에 가까워지도록 A 또는 C를 한 칸 이동
  - 벡터 AB와 CD의 외적이 0보다 크거나 같으면 C를 이동



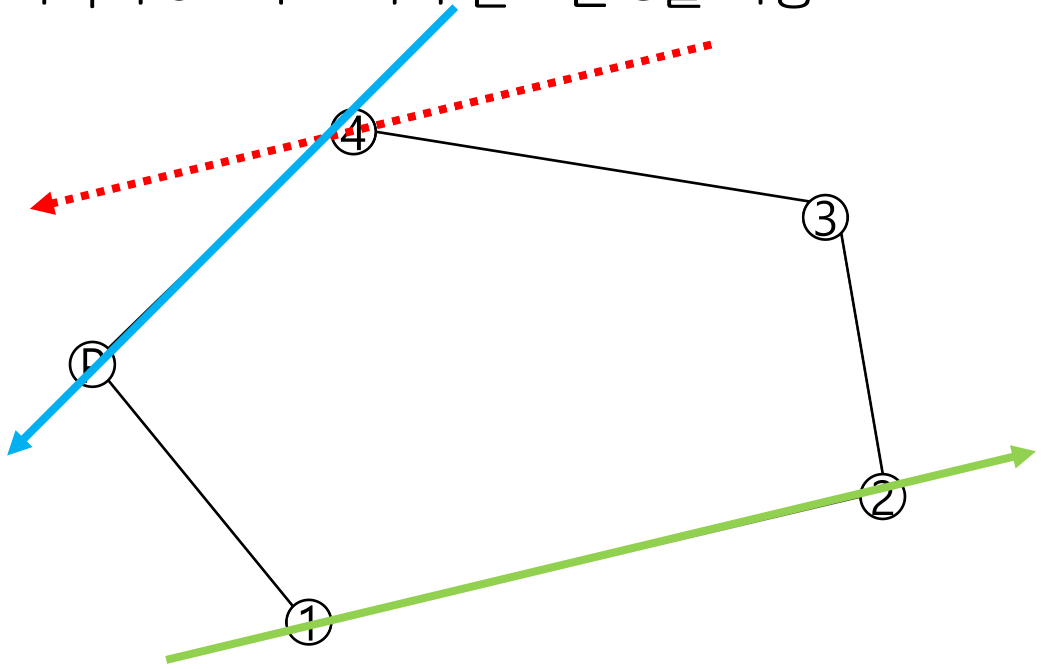
가장 먼 두 점

- 벡터 AB와 CD가 정반대에 가까워지도록 A 또는 C를 한 칸 이동
  - 벡터 AB와 CD의 외적이 0보다 크거나 같으면 C를 이동



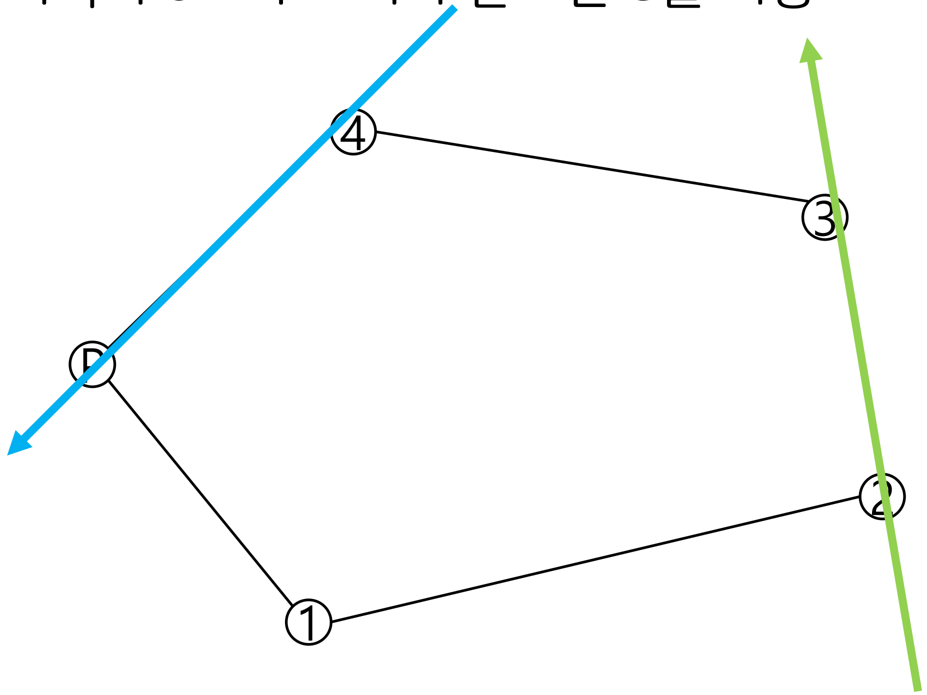
가장 먼 두 점

- 벡터 AB와 CD가 정반대에 가까워지도록 A 또는 C를 한 칸 이동
  - 벡터 AB와 CD의 외적이 0보다 크거나 같으면 C를 이동



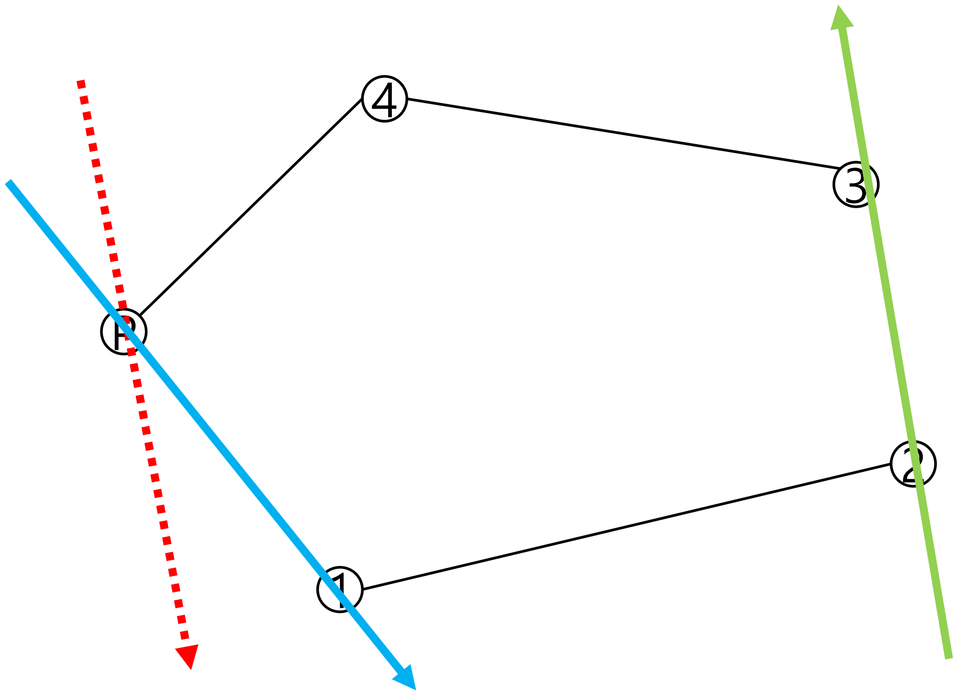
가장 먼 두 점

- 벡터 AB와 CD가 정반대에 가까워지도록 A 또는 C를 한 칸 이동
  - 벡터 AB와 CD의 외적이 0보다 크거나 같으면 C를 이동



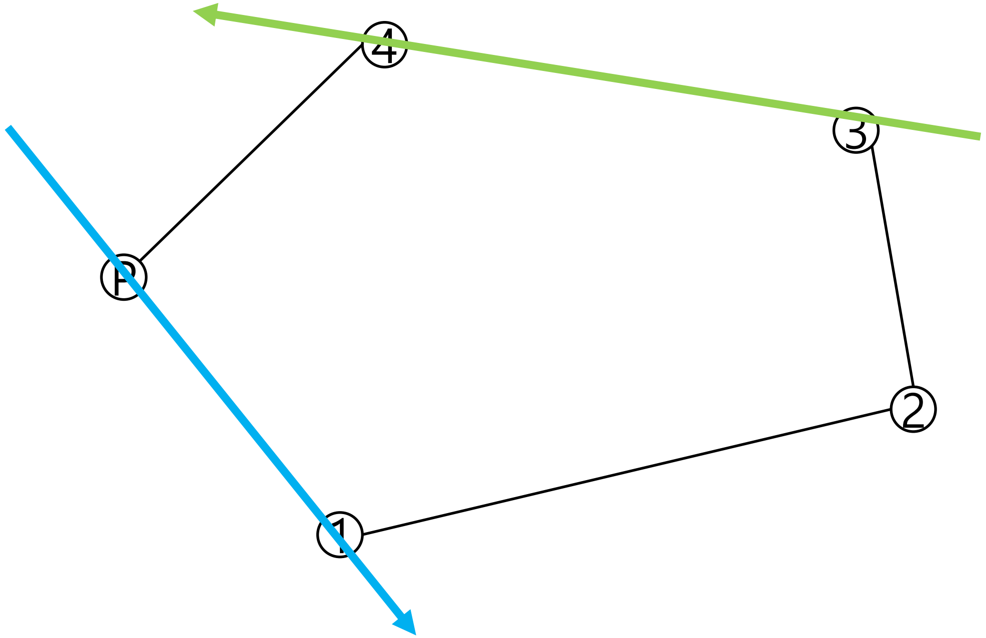
가장 먼 두 점

- 벡터 AB와 CD가 정반대에 가까워지도록 A 또는 C를 한 칸 이동
  - 벡터 AB와 CD의 외적이 0보다 크거나 같으면 C를 이동



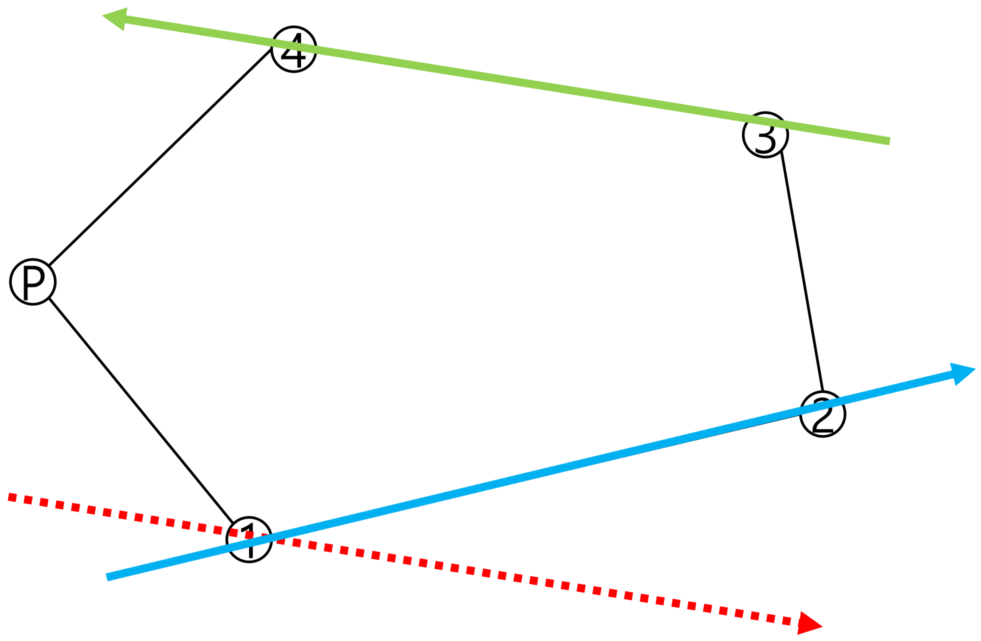
가장 먼 두 점

- 벡터 AB와 CD가 정반대에 가까워지도록 A 또는 C를 한 칸 이동
  - 벡터 AB와 CD의 외적이 0보다 크거나 같으면 C를 이동



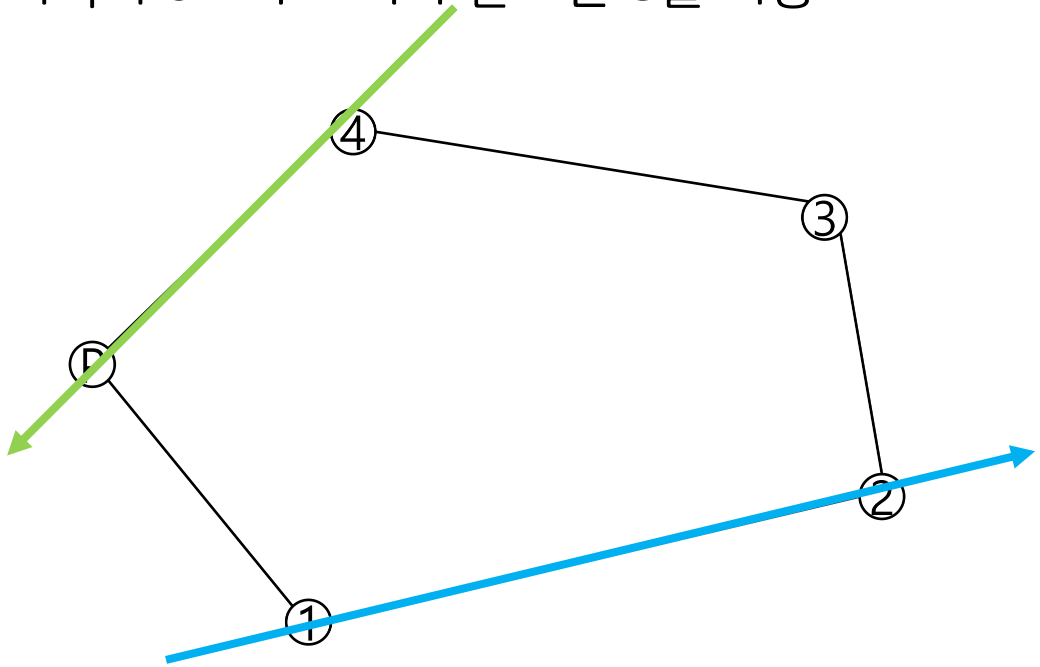
가장 먼 두 점

- 벡터 AB와 CD가 정반대에 가까워지도록 A 또는 C를 한 칸 이동
  - 벡터 AB와 CD의 외적이 0보다 크거나 같으면 C를 이동



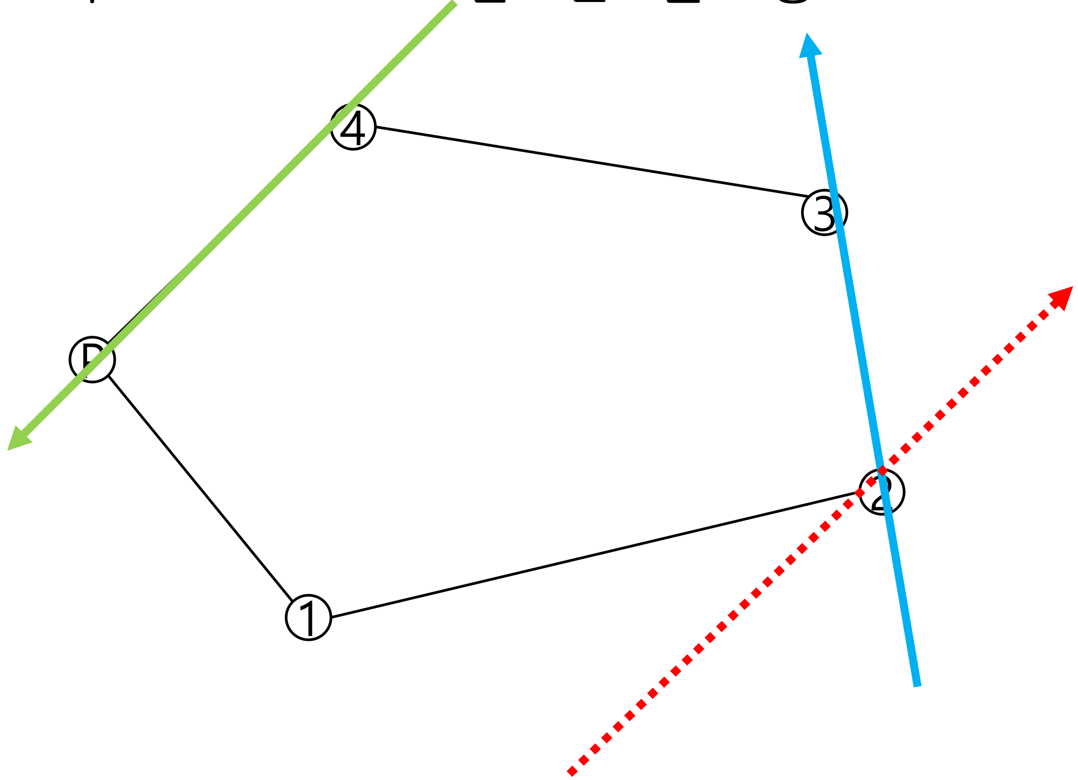
가장 먼 두 점

- 벡터 AB와 CD가 정반대에 가까워지도록 A 또는 C를 한 칸 이동
  - 벡터 AB와 CD의 외적이 0보다 크거나 같으면 C를 이동



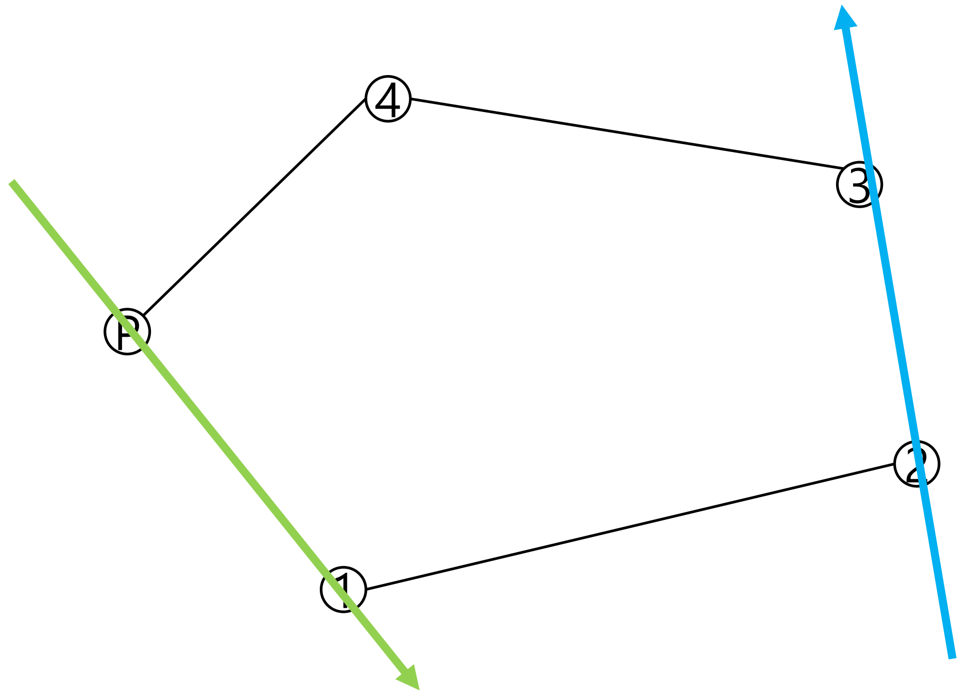
가장 먼 두 점

- 벡터 AB와 CD가 정반대에 가까워지도록 A 또는 C를 한 칸 이동
  - 벡터 AB와 CD의 외적이 0보다 크거나 같으면 C를 이동



가장 먼 두 점

- 벡터 AB와 CD가 정반대에 가까워지도록 A 또는 C를 한 칸 이동
  - 벡터 AB와 CD의 외적이 0보다 크거나 같으면 C를 이동



가장 먼 두 점

```cpp
Point operator - (const Point &p1, const Point &p2){
    return {p2.x - p1.x, p2.y - p1.y};
}

pair<Point, Point> RotatingCalipers(const vector<Point> &v){
    int n = v.size();
    ll mx = 0; Point a, b;

    for(int i=0, j=0; i<n; i++){
        while(j + 1 < n && CCW(Point(0,0), v[i+1] - v[i], v[j+1] - v[j]) >= 0){
            ll now = Dist(v[i], v[j]);
            if(now > mx) mx = now, a = v[i], b = v[j];
            j++;
        }
        ll now = Dist(v[i], v[j]);
        if(now > mx) mx = now, a = v[i], b = v[j];
    }
    return {a, b};
}
```

질문

참고 사항

- 좌표 범위가 [0, X]일 때 볼록 다각형의 꼭짓점이 될 수 있는 격자점은 $O(N^{2/3})$가지
  - BOJ 9240 로버트 후드는 X = 2000이므로 볼록 껍질을 구한 뒤 $O(N^2)$ 브루트 포스 해도 됨
  - 증명: https://codeforces.com/blog/entry/62183?#comment-461811

가장 가까운 두 점

가장 가까운 두 점

- 다양한 풀이가 존재
  - 오늘 다루는 방법
    - $O(N \log^2 N)$ 분할 정복
    - $O(N \log N)$ 분할 정복
    - $O(N \log N)$ 스위핑
    - $O(N \log N \log X)$ 버킷
  - 다루지 않는 방법
    - $O(N)$ 랜덤 알고리즘
      - https://cse.hkust.edu.hk/mjg_lib/Classes/COMP3711H_Fall14/lectures/Closest_Pairs.pdf

  - 사실 4가지 방법 다 비슷한 관찰을 이용

가장 가까운 두 점

- 분할 정복
  - 점들을 x좌표 오름차순으로 정렬하자.
  - 분할: 적당한 x_m에 대해, 직선 $x = x_m$을 기준으로 두 개의 부분 문제로 분할 후 각각 해결
  - 정복: 직선 $x = x_m$을 기준으로 서로 반대편에 있는 점들의 거리를 고려

  - 목표: $T(N) = 2T(N/2) + O(N \log N) = O(N \log^2 N)$
    - $O(N \log^2 N)$을 알면 $O(N \log N)$은 쉬움

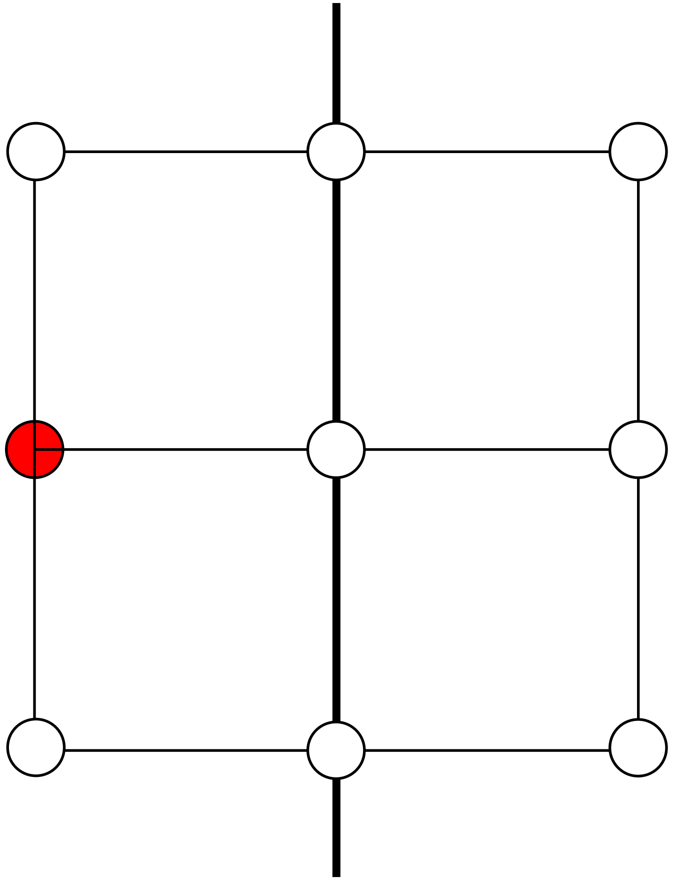
가장 가까운 두 점

- 분할 정복
  - 분할 단계
    - 양쪽에 균등하게 점이 나눠지도록 x_m을 잡자.
      - x 좌표 오름차순으로 정렬했을 때 중앙에 오는 점의 x좌표
    - $T(N) = 2T(N/2) + \text{something}$
  - 정복 단계
    - 두 부분 문제에서 계산한 최소 거리를 d라고 하자.
      - 정복 단계에서는 $|x - x_m| \le d$인 점만 고려해도 됨
      - 또한 $|y_1 - y_2| > d$인 두 점 (x_1, y_1), (x_2, y_2)는 고려하지 않아도 됨
    - 두 조건을 모두 만족하는 점들의 쌍은 몇 개일까?

가장 가까운 두 점

- 분할 정복
  - 각 점마다 6개의 점만 봐도 충분
  - 같은 영역에 있는 점까지 보더라도 8개

  - $|x - x_m| \le d$인 점 (x, y)를 모두 모은 다음
  - y좌표 오름차순으로 정렬하고
  - $0 \le y' - y \le d$인 점 (x', y')만 고려하면 됨



가장 가까운 두 점

- 분할 정복
  - 전체 시간 복잡도는 $T(N) = 2T(N/2) + O(N \log N) = O(N \log^2 N)$
    - y좌표 정렬할 때 $O(N \log N)$이 필요

  - $T(N) = 2T(N/2) + O(N) = O(N \log N)$ 방법
    - y좌표를 merge sort로 정렬
    - f(l, m)과 f(m+1, r)이 종료되면 [l, m]과 [m+1, r] 구간의 점은 각각 y좌표로 정렬된 상태
    - 두 구간의 점을 $O(N)$ 시간에 정렬할 수 있음

가장 가까운 두 점

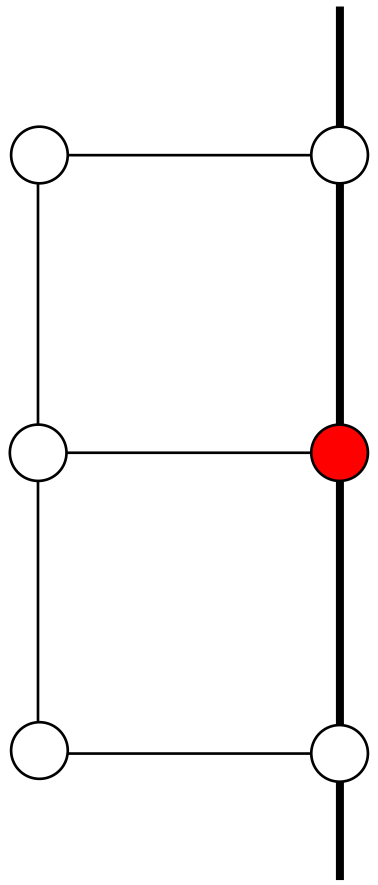
```cpp
#include <bits/stdc++.h>
#define x first
#define y second
using namespace std;
using ll = long long;
using Point = pair<ll, ll>;

inline ll Square(ll v){ return v * v; }
inline ll Dist(const Point &p1, const Point &p2){
    return Square(p2.x - p1.x) + Square(p2.y - p1.y);
}
struct CompareY{
    bool operator () (const Point &p1, const Point &p2) const {
        return tie(p1.y,p1.x) < tie(p2.y,p2.x);
    }
};

int N;
Point A[101010], B[101010];

ll DnC(int l, int r){
    if(l == r) return LLONG_MAX;
    int m = (l + r) / 2;
    ll xm = A[m].x, d = min(DnC(l, m), DnC(m+1, r));
    merge(A+l, A+m+1, A+m+1, A+r+1, B+l, CompareY());
    copy(B+l, B+r+1, A+l);

    vector<Point> P;
    for(int i=l; i<=r; i++) if(Square(A[i].x - xm) < d) P.push_back(A[i]);
    for(int i=0; i<P.size(); i++){
        for(int j=i-1; j>=0; j--){
            if(Square(P[i].y - P[j].y) < d) d = min(d, Dist(P[i], P[j]));
            else break;
        }
    }
    return d;
}

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    cin >> N;
    for(int i=1; i<=N; i++) cin >> A[i].x >> A[i].y;
    sort(A+1, A+N+1);
    cout << DnC(1, N);
}
```

질문

가장 가까운 두 점

- 스위핑
  - 점들을 x좌표 오름차순으로 정렬하자.

  - 점을 차례대로 훑으면서 최소 거리를 구함
    - i – 1번째 점까지 고려했을 때 최소 거리를 d라고 하자
    - i번째 점과 x 좌표가 d 이하로 차이나는 점만 고려해도 됨
    - i번째 점과 y 좌표가 d 이하로 차이나는 점만 고려해도 됨
    - 고려해야 하는 점의 개수는 최대 5개



가장 가까운 두 점

- 스위핑
  - std::set에 x 좌표가 d 이하로 차이나는 점 관리
    - lower_bound/upper_bound를 이용해 y 좌표 d 이하로 차이나는 점의 구간을 구할 수 있음

  - 시간 복잡도
    - 각 점은 최대 1번 삽입/삭제 : O(log N)
    - 각 점마다 lower_bound/upper_bound 1번씩 호출 : O(log N)
    - 각 점마다 iterator 증가 연산 O(1)번 : O(log N)
    - 각 점마다 O(log N)이므로 전체 O(N log N)

가장 가까운 두 점

```cpp
#include <bits/stdc++.h>
#define x first
#define y second
using namespace std;
using ll = long long;
using Point = pair<ll, ll>;

inline ll Square(ll v){ return v * v; }
inline ll Dist(const Point &p1, const Point &p2){
    return Square(p2.x - p1.x) + Square(p2.y - p1.y);
}
struct CompareY{
    bool operator () (const Point &p1, const Point &p2) const {
        return tie(p1.y,p1.x) < tie(p2.y,p2.x);
    }
};

int N;
Point A[101010];

ll Sweeping(){
    set<Point, CompareY> S;
    ll res = Dist(A[1], A[2]);
    S.insert(A[1]); S.insert(A[2]);
    for(int i=3, j=1; i<=N; i++){
        while(j < i && Square(A[i].x - A[j].x) >= res) S.erase(A[j++]);
        ll d = sqrt(res) + 2;
        auto it1 = S.lower_bound(Point(LLONG_MIN, A[i].y-d));
        auto it2 = S.upper_bound(Point(LLONG_MAX, A[i].y+d));
        while(it1 != it2) res = min(res, Dist(A[i], *it1++));
        S.insert(A[i]);
    }
    return res;
}

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    cin >> N;
    for(int i=1; i<=N; i++) cin >> A[i].x >> A[i].y;
    sort(A+1, A+N+1);
    cout << Sweeping();
}
```

질문

가장 가까운 두 점

- 버킷
  - 거리가 d 이하인 두 점은 한 변이 d/√2인 축에 평행한 정사각형으로 감쌀 수 있음
    - 대각선 길이가 d인 정사각형
    - 한 변의 길이가 d/√2인 정사각형 안에 두 점이 존재한다면 두 점의 거리는 d 이하

가장 가까운 두 점

- 버킷
  - 거리가 d 이하인 점이 존재하는지 판별하는 결정 문제를 이용해 파라메트릭 서치
    - k = floor(d/√2)라고 하자.
    - 점 (x, y)를 (floor(x/k), floor(y/k)) 기준으로 정렬
    - 같은 순서쌍이 존재하면 정사각형에 두 점이 존재하므로 정답은 d 이하
    - 존재하지 않는다면 (floor(x/k)±2, floor(y/k)±2) 범위의 버킷만 고려해도 됨
    - 각 버킷에는 점이 하나만 존재, 해시맵이나 이분 탐색으로 찾을 수 있음

    - 결정 문제를 O(N log N)에 해결할 수 있으므로 최적화 문제는 O(N log N log X)

질문

참고 사항

- 이렇게 하면 O(N√N)이라는 이야기를 들었는데 확실하진 않음
  - 랜덤한 각도 theta에 대해, 좌표 평면을 theta 만큼 회전하고 x 좌표 기준으로 정렬

```
for(int i=0; i<n; i++)
  for(int j=i-1; j>=0; j--)
    if(A[i].x – A[j].x < res) res = Dist(A[i], A[j]);
    else break;
```

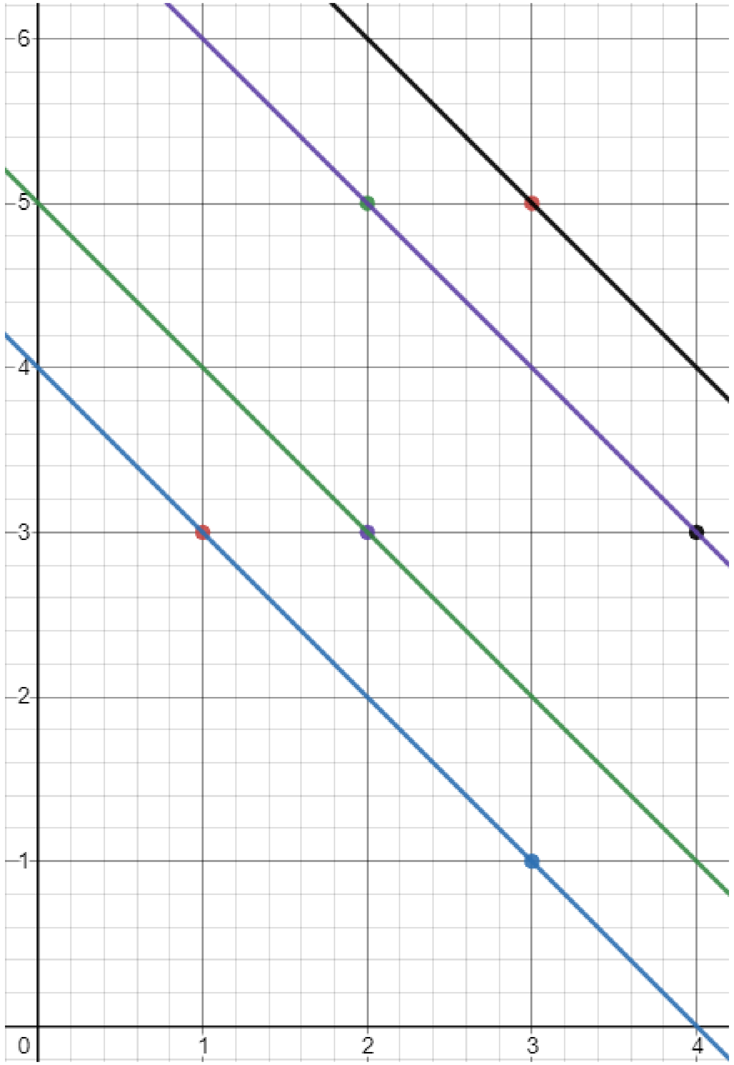
볼록 다각형의 접선을 이용한 최적화

볼록 다각형의 접선

- N개의 점 (x_i, y_i)가 주어진다.
- 실수 a가 주어지면 a*x_i + y_i의 최댓값을 구하자.
    - http://www.jungol.co.kr/bbs/board.php?bo_table=pbank&wr_id=3019

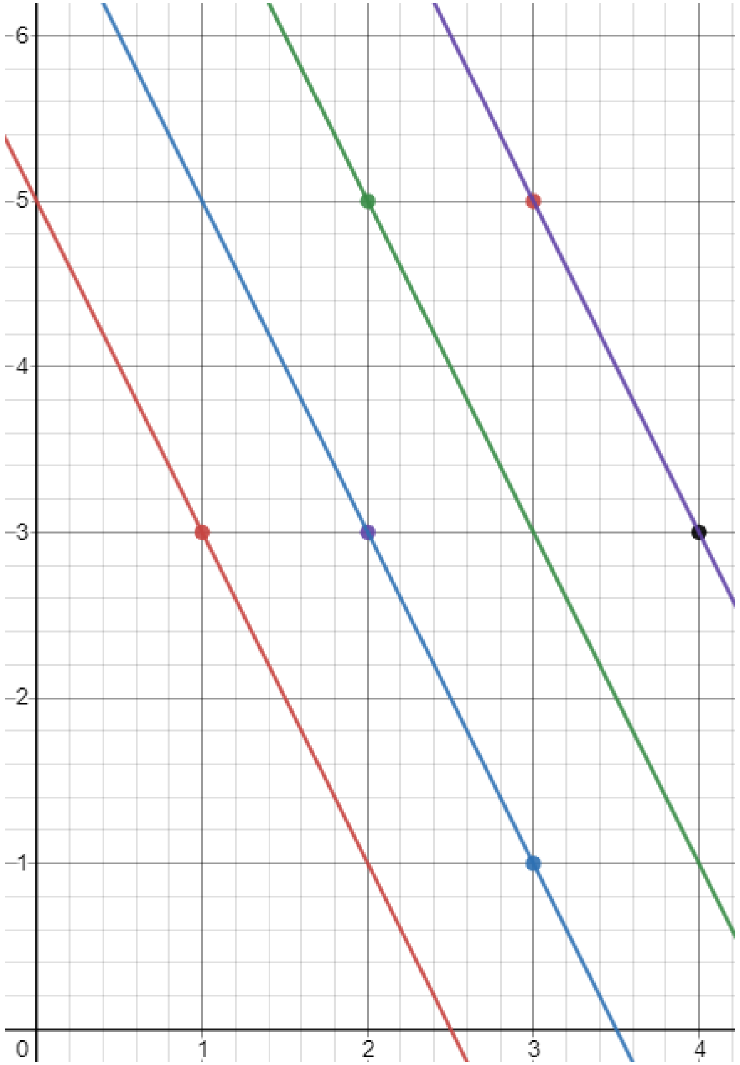
볼록 다각형의 접선

- N개의 점 (x_i, y_i)가 주어진다.
- 실수 a가 주어지면 a*x_i + y_i의 최댓값을 구하자.
    - a = 1이면 y_i – x_i가 같은 점들끼리 값이 동일함



볼록 다각형의 접선

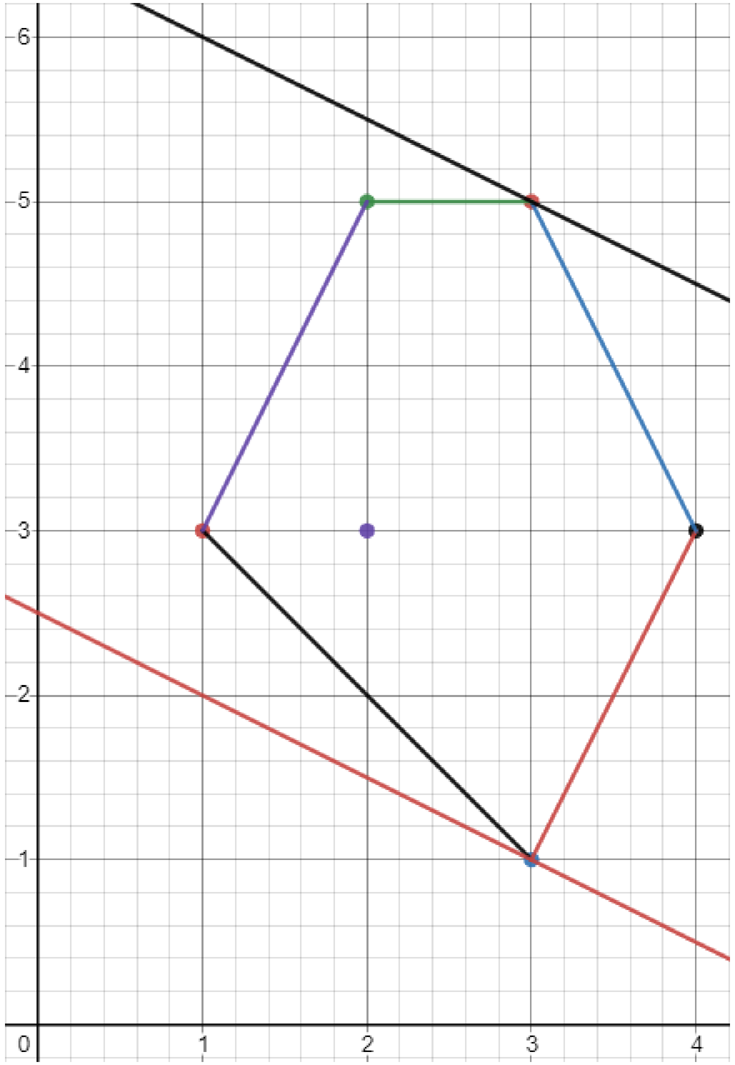
- N개의 점 (x_i, y_i)가 주어진다.
- 실수 a가 주어지면 a*x_i + y_i의 최댓값을 구하자.
  - a = 1이면 y_i – x_i가 같은 점들끼리 값이 동일함
  - a = 2이면 y_i – x_i*2가 같은 점들끼리 값이 동일함



볼록 다각형의 접선

- N개의 점 (x_i, y_i)가 주어진다.
- 실수 a가 주어지면 a*x_i + y_i의 최댓값을 구하자.
  - a = 1이면 y_i – x_i가 같은 점들끼리 값이 동일함
  - a = 2이면 y_i – x_i*2가 같은 점들끼리 값이 동일함

  - 볼록 껍질 구하고 기울기가 –a인 접선의 접점
    - 최솟값 : 아래에서 접하는 접선
    - 최댓값 : 위에서 접하는 접선
    - 윗 껍질 / 아랫 껍질 나누고 이분 탐색하면 됨



볼록 다각형의 접선

- http://boj.kr/fcf0c311792a41d1961f2a85f141e881

볼록 다각형의 접선

- N개의 점 (x_i, y_i)와 실수 a가 주어지면 a*x_i + y_i의 최댓값 구하기
- N개의 일차 함수 f_i(x) = a_i*x + b_i와 실수 x가 주어지면 f_i(x)의 최댓값 구하기

- y = ax + b
  - 일차 함수로 표현
  - 기울기 a = (y2 – y1) / (x1 – x2)
  - 절편 b = y1 – mx1
  - 장점 : 교점을 구하기 쉬움, 점과 동일하게 취급할 수 있음
  - 단점 : 기울기가 무한대일 때 예외 발생, 선분을 표현하기 힘듬

  - ax + by + c = 0으로 표현하는 경우도 존재
  - 기울기 무한대를 표현할 수 있지만 수식이 더러워짐

질문

볼록 다각형의 접선을 이용한 최적화

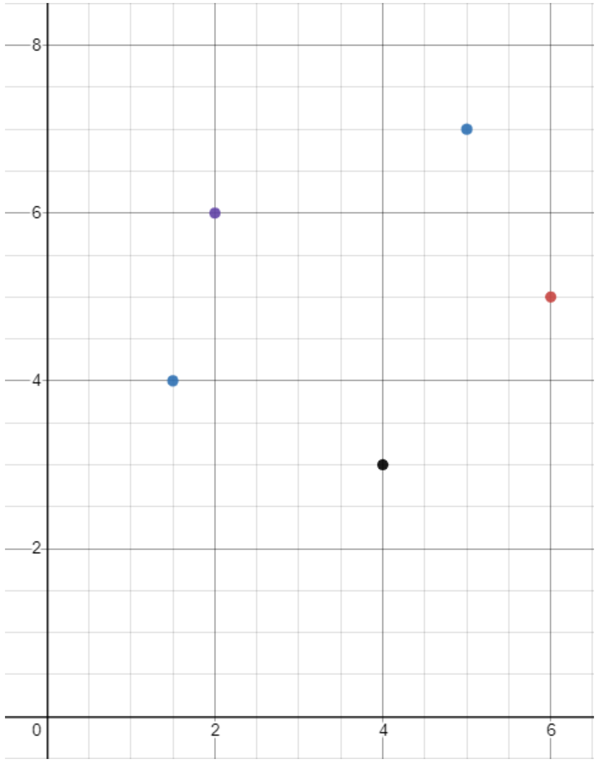
- N개의 점 (x_i, y_i)와 실수 a, b가 주어지면 a*x_i + b*y_i를 최대화/최소화
  - b * (a/b*x_i + y_i)이므로 기울기가 –a/b인 접선
  - 이걸 쓸 일이 있을까?

볼록 다각형의 접선을 이용한 최적화

- N개의 점 (x_i, y_i)와 실수 a, b가 주어지면 a*x_i + b*y_i를 최대화/최소화
  - b * (a/b*x_i + y_i)이므로 기울기가 –a/b인 접선
  - 이걸 쓸 일이 있을까?

  - 각 원소는 두 가지 종류의 가중치 A_i, B_i를 갖고 있고
  - 이들 중 몇 개를 선택해서 (sum A_i) * (sum B_i)의 합을 최소화하는 문제

  - 모든 경우에 대해, 좌표 평면에 (sum A_i, sum B_i) 점을 찍어보자

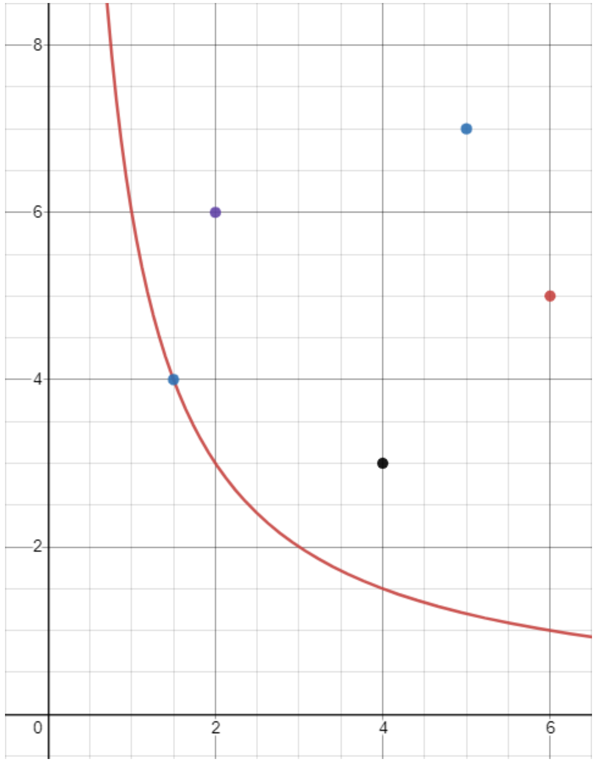


볼록 다각형의 접선을 이용한 최적화

- N개의 점 (x_i, y_i)와 실수 a, b가 주어지면 a*x_i + b*y_i를 최대화/최소화
  - b * (a/b*x_i + y_i)이므로 기울기가 –a/b인 접선
  - 이걸 쓸 일이 있을까?

  - 각 원소는 두 가지 종류의 가중치 A_i, B_i를 갖고 있고
  - 이들 중 몇 개를 선택해서 (sum A_i) * (sum B_i)의 합을 최소화하는 문제

  - 모든 경우에 대해, 좌표 평면에 (sum A_i, sum B_i) 점을 찍어보자
  - 최솟값이 c라면 다른 모든 점들은 xy = c 곡선의 위쪽에 존재함

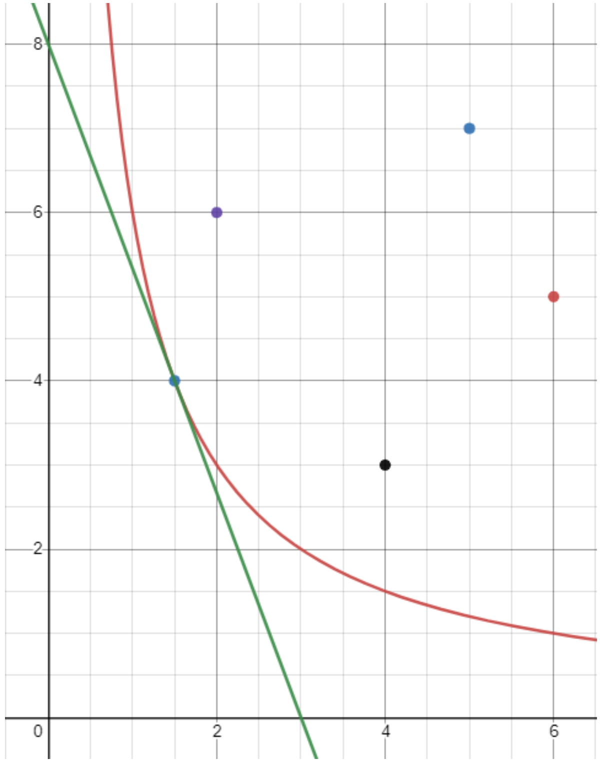


볼록 다각형의 접선을 이용한 최적화

- N개의 점 (x_i, y_i)와 실수 a, b가 주어지면 a*x_i + b*y_i를 최대화/최소화
  - b * (a/b*x_i + y_i)이므로 기울기가 –a/b인 접선
  - 이걸 쓸 일이 있을까?

  - 각 원소는 두 가지 종류의 가중치 A_i, B_i를 갖고 있고
  - 이들 중 몇 개를 선택해서 (sum A_i) * (sum B_i)의 합을 최소화하는 문제

  - 모든 경우에 대해, 좌표 평면에 (sum A_i, sum B_i) 점을 찍어보자
  - 최솟값이 c라면 다른 모든 점들은 xy = c 곡선의 위쪽에 존재함
  - 답이 되는 점을 A(x, y)라고 하면
  - ax + by = c가 되도록 하는 a, b가 존재 (기울기가 –b/a인 접선)
  - 모든 기울기에 대해 접점을 구한 뒤, 그 중 최솟값을 취하면 됨



볼록 다각형의 접선을 이용한 최적화

- 고려해야 하는 기울기 : 볼록 껍질에서 변의 기울기
  - 고려해야 하는 점의 개수가 너무 많아서 볼록 껍질을 직접 구할 수 없음
  - 볼록 껍질 위의 점의 개수 : min{ (좌표 범위)$^{2/3}$, N }
  - 기울기가 주어졌을 때, 그 기울기에 대한 접점을 T(N) 시간에 구할 수 있다면 $X^{2/3}T(N)$
    - ex) X = N이고 T(N) = N log N이면 전체 시간 복잡도는 $O(N^{5/3} \log N)$
    - 대충 N^2이 될 것 같은 입력 제한이면 의심해보자

질문

볼록 다각형의 접선을 이용한 최적화

- BOJ 5257 timeismoney
  - MST 비슷한 것을 만드는 문제
  - 각 간선은 X_i, Y_i라는 두 가지 종류의 가중치가 있음
  - 스패닝 트리의 가중치는 (sum X_i) * (sum Y_i)로 정의함
  - 가중치가 최소인 스패닝 트리를 구하는 문제

  - 1 ≤ N ≤ 200
  - 1 ≤ M ≤ 10’000
  - 1 ≤ X_i, Y_i ≤ 255

  - 좌표 범위는 200 * 255 = 51000
  - $51000^{2/3} \le 1400$

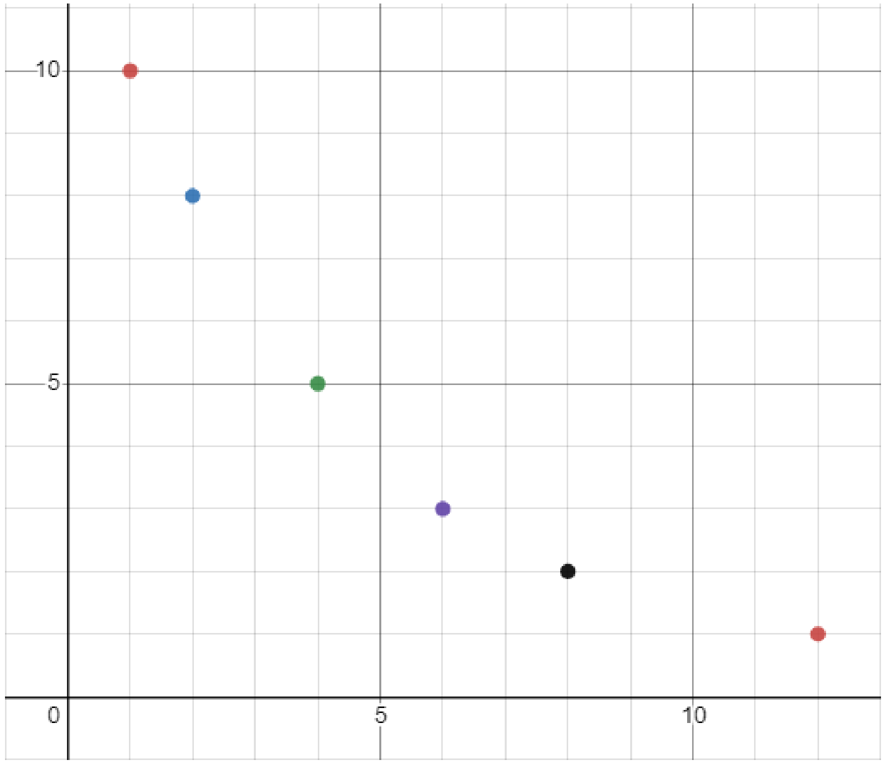
볼록 다각형의 접선을 이용한 최적화

- 기울기 -a/b가 주어지면 접점을 구하는 방법
  - a*(sum X_i) + b*(sum Y_i)를 최소화 해야 하므로
  - 간선을 aX_i + bY_i순으로 정렬하고 크루스칼
  - O(M log M)이므로 1400 * M log M에 문제를 풀 수 있다!

- 필요한 기울기를 어떻게 빠르게 구하지?

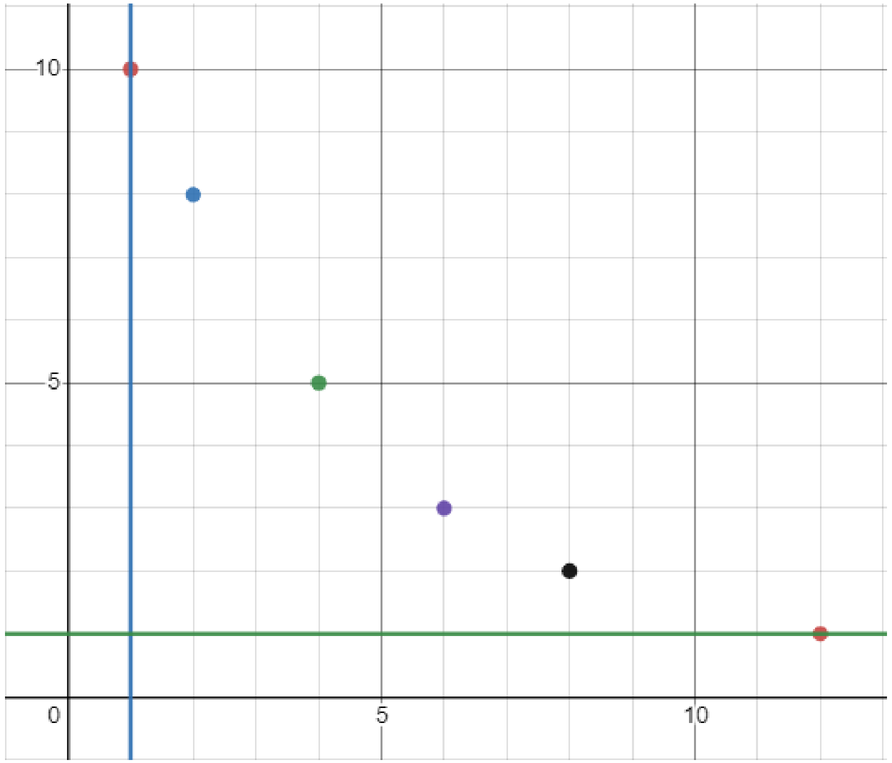
볼록 다각형의 접선을 이용한 최적화

- 가장 왼쪽에 있는 점 / 가장 아래에 있는 점은 쉽게 구할 수 있음



볼록 다각형의 접선을 이용한 최적화

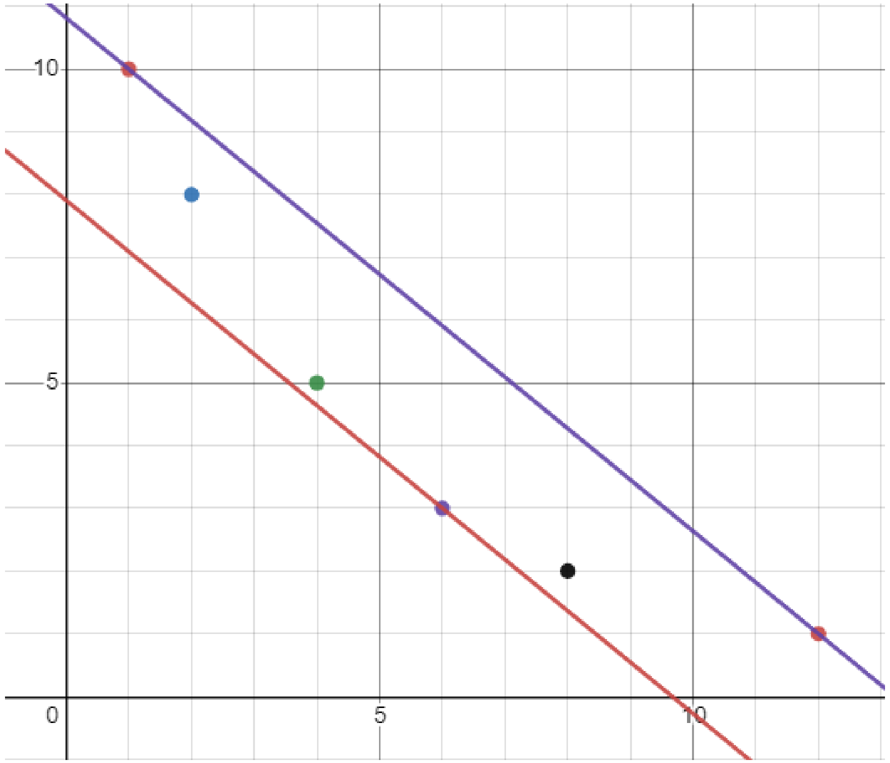
- 가장 왼쪽에 있는 점 / 가장 아래에 있는 점은 쉽게 구할 수 있음
  - 기울기가 1/0인 접선과 0/1인 접선



볼록 다각형의 접선을 이용한 최적화

- 가장 왼쪽에 있는 점 / 가장 아래에 있는 점은 쉽게 구할 수 있음
  - 기울기가 1/0인 접선과 0/1인 접선

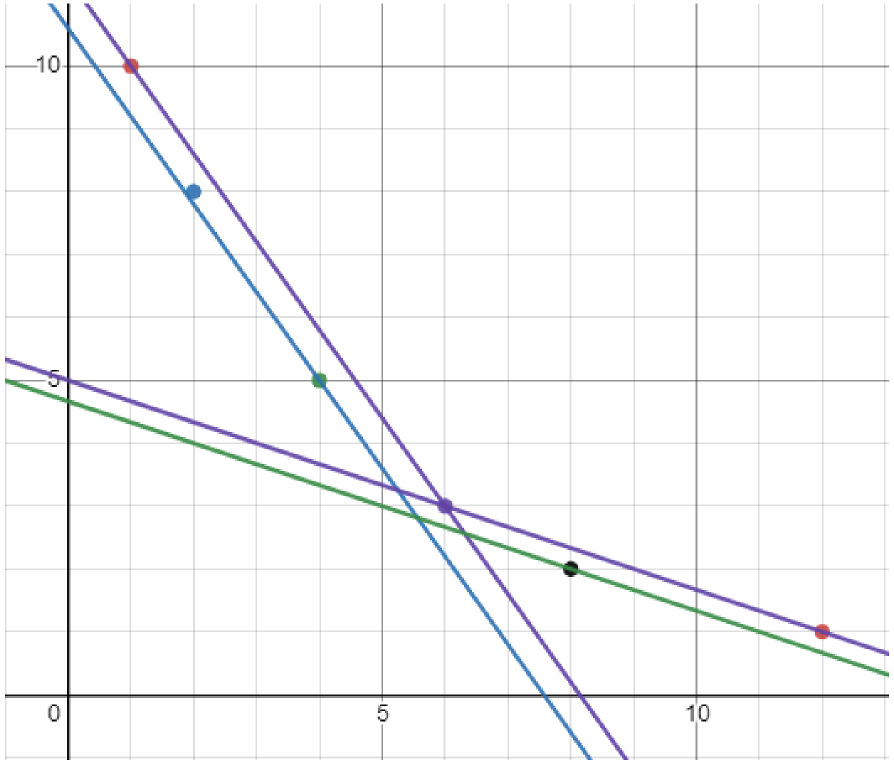
- 두 점을 잇는 직선의 기울기의 접점을 구하자



볼록 다각형의 접선을 이용한 최적화

- 가장 왼쪽에 있는 점 / 가장 아래에 있는 점은 쉽게 구할 수 있음
  - 기울기가 1/0인 접선과 0/1인 접선

- 두 점을 잇는 직선의 기울기의 접점을 구하자
  - 가장 왼쪽에 있는 점과 지금 찾은 점을 잇는 직선
  - 가장 아래에 있는 점과 지금 찾은 점을 잇는 직선
  - …
  - 분할 정복
    - 볼록 껍질 위의 점 개수 만큼만 호출됨



볼록 다각형의 접선을 이용한 최적화

```cpp
#include <bits/stdc++.h>
#define x first
#define y second
using namespace std;
using ll = long long;
using Point = pair<ll, ll>;

ll CCW(const Point &p1, const Point &p2, const Point &p3){
    return (p2.x - p1.x) * (p3.y - p2.y) - (p3.x - p2.x) * (p2.y - p1.y);
}

struct Edge{ int u, v, x, y; };
struct UnionFind{
    int P[222];
    UnionFind(){ clear(); }
    void clear(){ iota(P, P+222, 0); }
    int find(int v){ return v == P[v] ? v : P[v] = find(P[v]); }
    bool merge(int u, int v){
        u = find(u); v = find(v);
        if(u == v) return false;
        P[u] = v; return true;
    }
};

int N, M;
Edge E[10101];
UnionFind UF;
vector<pair<int,int>> MST;
ll OptX = 1e9, OptY = 1e9;
```

```cpp
Point Optimize(ll dy, ll dx){
    UF.clear();
    sort(E+1, E+M+1, [&](const Edge &a, const Edge &b){
        return dy*a.x + dx*a.y < dy*b.x + dx*b.y;
    });
    vector<pair<int,int>> now;
    ll sx = 0, sy = 0;
    for(int i=1; i<=M; i++){
        if(UF.merge(E[i].u, E[i].v)){
            sx += E[i].x; sy += E[i].y;
            now.emplace_back(E[i].u, E[i].v);
        }
    }
    if(sx*sy < OptX*OptY) OptX = sx, OptY = sy, MST = now;
    return { sx, sy };
}

void Solve(Point le, Point dw){
    Point pt = Optimize(le.y - dw.y, dw.x - le.x);
    if(CCW(le, pt, dw) > 0) Solve(le, pt), Solve(pt, dw);
}

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    cin >> N >> M;
    for(int i=1; i<=M; i++) cin >> E[i].u >> E[i].v >> E[i].x >> E[i].y;

    auto le = Optimize(1, 0), dw = Optimize(0, 1);
    Solve(le, dw);
    cout << OptX << " " << OptY << "\n\n";
    for(auto i : MST) cout << i.x << " " << i.y << "\n";
}
```

질문

더 공부할 거리

- 만약 이게 재밌다면…
  - K-D Tree
  - Rotate Sweep Line (A.K.A. Bulldozer Trick)
  - Half Plane Intersection
  - Dual Graph
  - Voronoi Diagram / Delaunay Triangulation