

SCPC 5회 1차예선 풀이

서울대학교 컴퓨터공학부 18학번 김동현

1번 - 오르락 내리락

- 1 이상의 정수에 다음과 같은 작업을 반복하여 1로 만들려고 한다.
 - 홀수라면 1을 더한다.
 - 짝수라면 2로 나눈다.
- N_1 과 N_2 가 주어지면 $N_1, N_1 + 1, \dots, N_2$ 를 각각 1로 만드는 데 필요한 작업 횟수를 모두 더한 값을 출력하라.
- $1 \leq T \leq 10,000$
- $1 \leq N_1 \leq N_2 \leq 10^6$

1번 - 오르락 내리락

- 매우 간단한 DP로 10^6 이하의 모든 정수 k 에 대해 k 를 1로 만드는 데 필요한 작업 횟수를 구할 수 있습니다.
 - k 가 짝수 $\rightarrow (k/2$ 의 작업 횟수) + 1
 - k 가 홀수 $\rightarrow (k+1$ 의 작업 횟수) + 1 = $((k+1)/2$ 의 작업 횟수) + 1
- 2개씩 끊어서 계산하면 쉽게 구할 수 있습니다.

1번 - 오르락 내리락

- T가 크기 때문에, 각 테스트 케이스에 대해 일일이 답을 더하면 안 됩니다.
- 이는 누적합 배열로 간단하게 해결할 수 있습니다.
- $\text{sum}[k] = (1 \sim k \text{ 사이의 모든 자연수에 대해 작업 횟수의 합})$ 이라 합시다.
- $\text{sum}[k] = \text{sum}[k-1] + (k \text{의 작업 횟수})$ 이므로 $\text{sum}[k]$ 를 쉽게 $O(N)$ 에 구할 수 있습니다.
- 각 테스트 케이스에 대해, 정답은 $\text{sum}[N_2] - \text{sum}[N_1 - 1]$ 입니다.

1번 - 오르락 내리락

- sum 배열의 값은 각 TC를 풀기 전에 미리 채워두어야 함을 유의하세요.

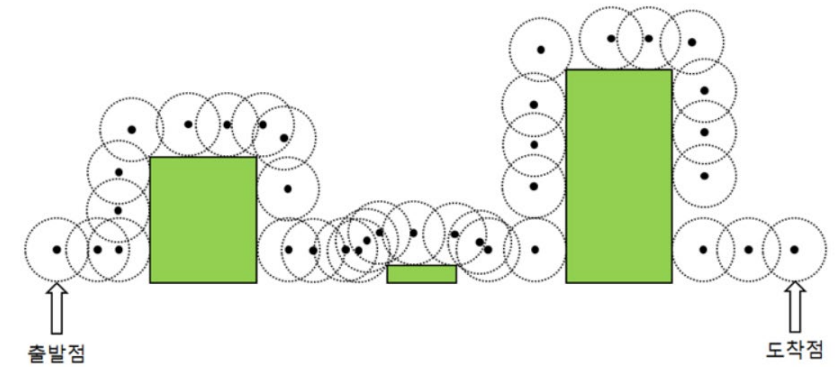
```
const int N = 1000005;
vll sum(N);

void solve() {
    int s, e;
    cin >> s >> e;
    cout << (sum[e] - sum[s - 1]) << '\n';
}

sum[1] = 0;
sum[2] = 1;
for(int i = 3; i + 1 < N; i += 2) {
    sum[i + 1] = sum[(i + 1) / 2] + 1;
    sum[i] = sum[i + 1] + 1;
}
for(int i = 1; i < N; i++) sum[i] += sum[i - 1];
```

2번 - 공 굴리기

- 반지름 R 의 공을 오른쪽 그림과 같이 출발점에서 도착점으로 굴린다.
- 장애물이 총 N 개 ($1 \leq N \leq 1000$) 있다.
- 각 장애물들은 어떤 x 좌표 구간 $[l, r]$ 에 높이 h 로 세워져 있다.
- 각 장애물들은 $2R$ 이상 떨어져 있으며, 출발/도착점에서 공이 바닥에 닿는다.
- 공의 중심이 이동한 총 거리를 출력하라.

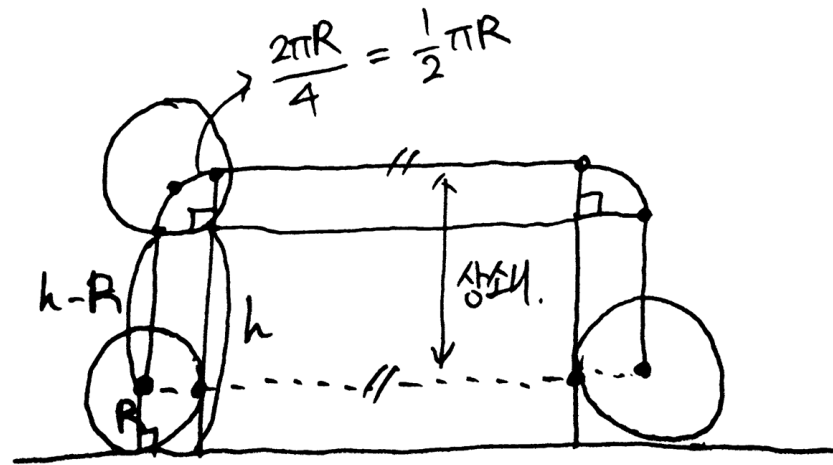


2번 - 공 굴리기

- 갑자기 웬 수학 문제가 나왔습니다;;
- 일단 장애물이 없을 때 중심 이동거리는 당연히 출발점과 도착점의 좌표 차이입니다.
- 각 장애물들이 충분히 멀리 떨어져 있으니, 장애물이 중심 이동 거리에 미치는 영향은 서로 독립적입니다.
- 장애물이 중간에 하나 있을 때 중심 이동 거리가 얼마나 늘어나는지 구해 봅시다.

2번 - 공 굴리기

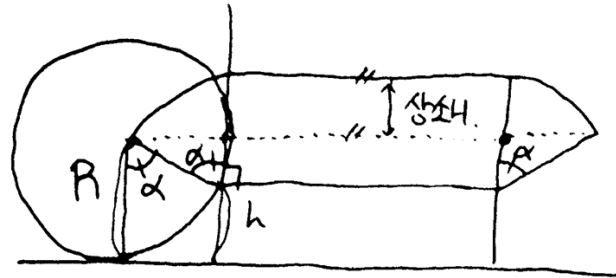
- (1) $R \leq h$
- 공이 잠시 수직으로 이동하는 때가 있습니다.
- 늘어나는 거리는 $2(h-R) + \pi R - 2R = 2h + (\pi-4)R$ 입니다.




$$2(h-R) + 2 \cdot \frac{1}{2}\pi R - 2R = 2h + (\pi-4)R$$

2번 - 공 굴리기

- (2) $R \gg h$
- 이제는 공이 수직으로 올라가지 않습니다.
- 공이 장애물과 만나는 순간을 잘 생각해서 마찬가지로 식을 써 보면, 늘어난 거리는 $2R \cos^{-1} \left(1 - \frac{h}{R} \right) - 2\sqrt{2Rh - h^2}$ 이 됩니다.



$$\therefore 2R \cos^{-1} \left(1 - \frac{h}{R}\right) - 2\sqrt{2Rh - h^2}$$

$R-h$  $\rightarrow \alpha = \cos^{-1}\left(\frac{R-h}{R}\right)$
 $\sqrt{R^2 - (R-h)^2} = \sqrt{2Rh - h^2}$

2번 - 공 굴리기

- 어차피 계산은 컴퓨터가 하기 때문에 굳이 식 정리를 끝까지 안 해도 됩니다.
- π 의 값은 저렇게 쭉 써도 되고(?) $\text{acos}(-1)$ 의 값을 써도 됩니다.

```
int s, e, n;
ld r;
cin >> r >> s >> e >> n;

ld ans = e - s;
const static ld PI = 3.14159265358979323846264;
for(int i = 0; i < n; i++) {
    ld x, y, h;
    cin >> x >> y >> h;
    if(h < r) {
        ans += 2 * acos(1 - h / r) * r - 2 * sqrt(2 * r * h - h * h);
    }
    else {
        ans += 2 * (h - r) + r * PI - 2 * r;
    }
}
cout << ans << '\n';
```

3번 - 점프

- 0번 칸부터 X번 칸까지 여러 번의 점프를 통해 도달하려고 합니다.
- 점프는 아래의 두 가지 중 하나가 가능합니다.
 - 1칸을 뛩니다.
 - (바로 전에 뛩 칸 수 + 1)칸을 뛩니다.
- $f(X)$ 를 “X번 칸까지 뛰는 데 필요한 최소 점프 횟수”라고 합니다.
- x, y 가 주어지면 $f(x), f(x+1), \dots, f(y)$ 중 최댓값을 구하세요.
- $1 \leq x \leq y \leq 10^{11}$

3번 - 점프

- 문제를 처음 보면 참 막막합니다.
- 이런 경우에, 작은 제한에 대해서 일단 먼저 문제를 푼 다음 그것을 바탕으로 규칙을 찾으면 도움이 되는 경우가 꽤 있습니다.
- 우선, 제한이 작은 부분문제를 풀어 봅시다.
 - 1번 부분문제는 x, y 가 10^4 이하입니다.

3번 - 점프

- $d[i]$ 를 i 번 칸까지 오는 데 필요한 최소 점프 횟수라고 정의하면, 아래와 같은 코드로 $O(N^{1.5})$ 정도에 비례하는 시간에 DP 값을 모두 구할 수 있습니다.
 - 마지막에 몇 칸 뛰었는지를 같이 저장하는 2차원 DP를 해도 됩니다. 이 때는 X 번 칸에 도달할 때까지 가장 멀리 뛴 점프가 $O(\sqrt{X})$ 칸 정도를 뛰기 때문에 역시 시간복잡도가 $O(N^{1.5})$ 이 됩니다.

```
d[0] = 0;
for(int i = 0; i < N; i++) {
    for(int j = 1, k = 1; i + k < N; k += ++j) {
        d[i + k] = min(d[i + k], d[i] + j);
    }
}
```

3번 - 점프

- 이제 DP 값을 몇 개 출력해보고 규칙이 있는지 살펴봅시다.
- 0번에서 시작해서 점프를 계속 전보다 한 칸 씩 더 뛰었을 때 거치는 칸들(1번, 3번, 6번, ...)을 기준으로 DP 값을 출력해보면 아래와 같습니다.

```
int c = 1;
for(int i = 1; i <= 20; i++) {
    for(int j = 0; j < i + 1; j++) {
        printf("%3d ", d[c++]);
    }
    puts("");
}
```

1	2																		
2	3	4																	
3	4	5	5																
4	5	6	6	7															
5	6	7	7	8	8														
6	7	8	8	9	10	9													
7	8	9	9	10	11	10	11												
8	9	10	10	11	12	11	12	13											
9	10	11	11	12	13	12	13	14	14										
10	11	12	12	13	14	13	14	15	15	14									
11	12	13	13	14	15	14	15	16	16	15	16								
12	13	14	14	15	16	15	16	17	17	16	17	18							
13	14	15	15	16	17	16	17	18	18	17	18	19	19						

3번 - 점프

- 뭔가 아래와 같은 이동 방식이 최적인 것 같아 보입니다..?
 - X번 칸을 넘어가기 직전까지 0번에서 최대한 멀리 뛰기를 반복한 뒤, 그 때 도달한 위치를 Y라 하면 $dp[X-Y]$ 번 더 뛴다.
- 그런데 중간에 반례가 있는 것 같습니다... (빨간 원)

1	2													
2	3	4												
3	4	5	5											
4	5	6	6	7										
5	6	7	7	8	8									
6	7	8	8	9	10	9								
7	8	9	9	10	11	10	11							
8	9	10	10	11	12	11	12	13						
9	10	11	11	12	13	12	13	14	14					
10	11	12	12	13	14	13	14	15	15	14				
11	12	13	13	14	15	14	15	16	16	15	16			
12	13	14	14	15	16	15	16	17	17	16	17	18		
13	14	15	15	16	17	16	17	18	18	17	18	19	19	

3번 - 점프

- 그런데 반례를 다 찍어 봐도, n 이 작을 때만 좀 나타나다가 안 나타나는 것 같기도 합니다..
- 놀랍게도, 아래 나온 반례를 제외하면 앞에서 말한 성질이 성립합니다!
 - 증명은 일단 생략합니다.

```
c = 1;
for(int i = 1; i <= 1500; i++) {
    int st = c;
    for(int j = 0; j < i + 1; j++, c++) {
        if(d[c] != d[st] + d[j]) {
            printf("counter : %d (%d + %d != %d)\n", c, d[st], d[j], d[c]);
        }
    }
}
```

```
counter : 20 (5 + 4 != 8)
counter : 119 (14 + 7 != 20)
counter : 461 (29 + 10 != 38)
counter : 594 (33 + 11 != 43)
counter : 1172 (47 + 13 != 59)
```


3번 - 점프

- 이제 임의의 k 에 대해, 대충 \sqrt{k} 정도까지의 DP값을 모두 알고 있으면 $f(k)$ 의 값을 빠르게 구할 수 있습니다!
- 제한이 10^{11} 이니, 45만 정도까지 DP값을 미리 구해놓으면 됩니다.
- K 가 작으면 구해놓은 DP값을 그대로 쓰면 되고, K 가 커지면 0번에서 시작해 K 를 넘어가기 직전까지 최대한 멀리 뛴 뒤 남은 칸 수에 해당하는 DP값을 더해주면 됩니다.
 - K 를 넘어가기 직전까지 얼마나 멀리 뛰는지는 이분 탐색 등을 통해 빠르게 구할 수 있습니다.
- 하지만 x 와 y 사이의 모든 값을 빨리 구하기는 역부족입니다...

3번 - 점프

- 아까 DP값을 적어놓은 삼각형을 보면, 삼각형의 각 열은 **아래로 내려갈수록 감소하지 않음**을 알 수 있습니다.
- 즉, 구간이 너무 길다면 맨 뒤에서부터 **모든 열을 다 덮을 때까지만 점프 횟수를 구해도 충분합니다!**
- 아까 잡은 상한 (45만) 만큼만 고려해도 충분합니다.

1	2													
2	3	4												
3	4	5	5											
4	5	6	6	7										
5	6	7	7	8	8									
6	7	8	8	9	10	9								
7	8	9	9	10	11	10	11							
8	9	10	10	11	12	11	12	13						
9	10	11	11	12	13	12	13	14	14					
10	11	12	12	13	14	13	14	15	15	14				
11	12	13	13	14	15	14	15	16	16	15	16			
12	13	14	14	15	16	15	16	17	17	16	17	18		
13	14	15	15	16	17	16	17	18	18	17	18	19	19	

3번 - 점프

- DP값을 구하는 부분은 아까 짰 코드를 그대로 쓰면 됩니다.
- sum은 어떤 i를 넘어가기 직전까지 최대한 멀리 뛴 위치, st는 그 때 점프 횟수입니다.

```
void solve() {
    ll s, e;
    cin >> s >> e;

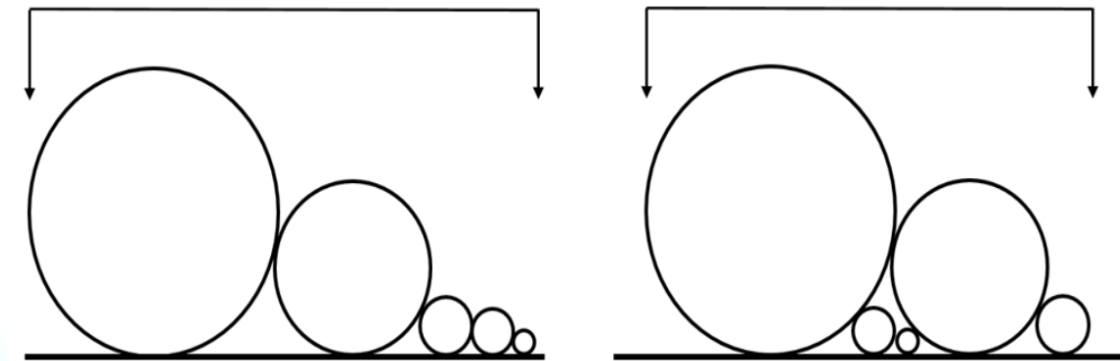
    ll bnd = max(s, e - N);

    int st = 1;
    ll sum = 1;
    while(sum <= e) sum += ++st;
    sum -= st--;

    int ans = 0;
    for(ll i = e; i >= bnd; i--) {
        if(i < N) ans = max(ans, d[i]);
        else {
            while(sum > i) sum -= st--;
            ans = max(ans, st + d[i - sum]);
        }
    }
    cout << ans << '\n';
}
```

4번 - 파이프

- 옆에서 봤을 때 원형인 파이프가 N개 있다. 각각의 반지름이 주어진다.
- 모든 파이프가 바닥에 닿도록 어떤 순서로 배열할 것이다.
- 파이프를 적절한 순서로 배열하여 파이프가 존재하는 x좌표의 구간을 최대한 좁게 만들어라. (각 파이프의 중심의 x좌표를 출력해야 함)
- $3 \leq N \leq 100$

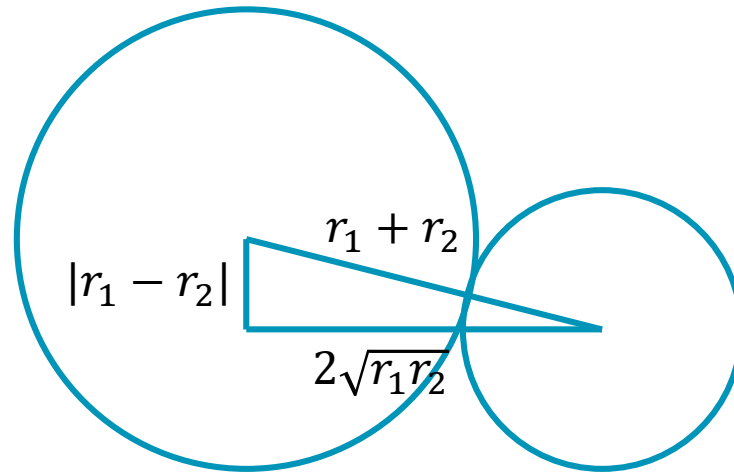


4번 - 파이프

- 점수를 또 이상하게 줍니다...
- 기준이 4회 때보다는 약간 널널합니다.
 - 어떤 테스트 케이스에 대해서 주최 측의 답보다 더 좋은 답을 내면, 그걸로 다른 테스트 케이스에서 못 한 걸 메꿀 수 있습니다.
- 이번에도 역시 일단 적당한 그리디를 짜면 좋을 것 같습니다.
- 그런데 이 문제 같은 경우는 순서를 정한 다음에 최대한 붙여서 배열하는 것을 일단 먼저 해결해야 합니다.

4번 - 파이프

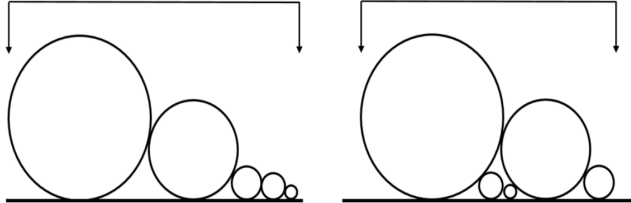
- 반지름이 각각 r_1, r_2 인 원 두 개를 바닥에 닿은 상태에서 딱 맞게 붙였다 고 하면, 두 원의 중심의 x좌표 차이는 $2\sqrt{r_1 r_2}$ 입니다.



- 어떤 원의 위치를 구할 때는 그 앞에 있는 모든 원에 대해 고려해야 함에 유의하세요. $O(N^2)$ 만에 정확한 위치를 다 구할 수 있습니다.

4번 - 파이프

- 이제 그럴듯한 그리디에 대해 생각을 해 봅시다.
- 뭔가 큰 파이프와 작은 파이프가 붙어 있으면 x좌표가 겹치는 부분이 많아서 절약이 될 것 같은 느낌입니다.

- 대충 이런 느낌 →→ 

- 파이프를 반지름 순으로 정렬한 뒤 그 순서대로 번호를 매겼다고 하고, 1, N, 2, N-1, 3, N-2, ... 이런 식으로 대충 채워봅시다.

4번 - 파이프

- 놀랍게도, 이걸 하면 198점이 나옵니다 (??)
- 저번에 언급했던 간단한 local search (파이프 두 개 순서를 바꿔서 답이 더 좋아지나 확인) 을 약간 가미하면 200점이 뜹니다.
- 4회 예선 때 만점자가 너무 적어서 난이도를 내린 느낌이 듭니다.

4번 - 파이프

- 순서를 주면 최대한 타이트하게 배치해서 답을 구해주는 함수입니다.
- 이런 걸 처음에 잘 짜두면 문제 풀기가 매우 편해집니다.

```
auto put = [&](vint &p, bool print) {
    vld pos(n);
    ld mn = -r[p[0]], mx = r[p[0]];
    for(int i = 1; i < n; i++) {
        for(int j = 0; j < i; j++) {
            pos[i] = max(pos[i], pos[j] + 2 * sqrt(r[p[i]] * r[p[j]]));
        }
        mx = max(mx, pos[i] + r[p[i]]);
        mn = min(mn, pos[i] - r[p[i]]);
    }

    if(print) {
        vint q(n);
        for(int i = 0; i < n; i++) q[p[i]] = i;
        for(int i = 0; i < n; i++) cout << pos[q[i]] << '\n';
    }

    return mx - mn;
};
```

4번 - 파이프

- 적당한 전략을 실행하는 부분입니다. 파이프를 정렬한 뒤 작은 것부터 2칸씩 점프하며 채웁니다.
- vector에서 `iota(all(v), 0);` 을 수행하면 첫 번째 원소부터 순서대로 0, 1, 2, ... 가 채워집니다.
- 인덱스 처리에 매우 주의합니다. 좀 헛갈립니다.

```
vint kth(n);  
iota(all(kth), 0);  
sort(all(kth), [&](int x, int y){ return r[x] < r[y]; });
```

```
vint p(n);  
int c = 0;  
for(int i = 0; i < n; i += 2) p[i] = kth[c++];  
for(int i = n - 1 - (n & 1); i >= 0; i -= 2) p[i] = kth[c++];  
ld mn = put(p, false);
```

4번 - 파이프

- 랜덤하게 두 파이프의 자리를 바꾸어 보면서 답이 더 좋아지면 그 쪽으로 갑니다.
- 100번씩만 더 해줘도 충분한 듯 합니다.

```
random_device rd;
mt19937 mt(rd());
uniform_int_distribution<int> rnd(0, n - 1);

int trial = 100;
while(trial--) {
    int x, y;
    do {
        x = rnd(mt);
        y = rnd(mt);
    } while(x == y);
    swap(p[x], p[y]);
    ld cur = put(p, false);
    if(mn > cur) mn = cur;
    else swap(p[x], p[y]);
}
```

5번 - 세포 키우기

- 2차원 평면상에서 (L, 0)과 (R, 0)을 잇는 선분 위에 세포를 하나 키울 것이다.
- 세포는 선분 위의 한 점을 중심으로 하는 정사각형 모양으로 자란다.
- 평면 위에 총 N개의 불순물이 있어서, 세포는 자라다가 경계에 불순물이 닿으면 그 순간 자라기를 멈춘다.
- 세포의 중심을 적절한 위치에 잡아서 한 변의 길이를 최대한 길게 키우려고 할 때, 가능한 최대 길이를 구하여라.
- $1 \leq N \leq 100,000$
- $|\text{모든 좌표}| \leq 10^{12}$

5번 - 세포 키우기

- 또 파라메트릭 서치 (답에 대한 이분 탐색) 문제입니다.
- 1차 예선에서만 벌써 3번째 보는 거 같습니다.
- 심지어, 이 문제 역시 모든 좌표에 2를 곱하면 풀기가 편해집니다.
 - 답이 홀수일 경우에는 $(\text{정수})/2$ 위치에 중심이 위치해야 하기 때문입니다.
- 모든 좌표에 2를 곱하고, 구하는 답을 “한 번 길이의 절반” 이라고 하면 일단 동일한 문제가 됩니다.

5번 - 세포 키우기

- 세포가 최소 X 만큼 클 수 있는가?를 판단한다고 합시다. (즉, 한 변의 길이가 $2X$ 가 될 수 있는가?)
- 세포를 딱 X 까지만 키운 다음에 불순물이 내부에 들어가지 않도록 배치할 수 있으면 됩니다.
- 각각의 불순물은 그 불순물의 y 좌표에 따라 아예 영향을 끼치지 못하거나, 특정 x 좌표 구간에 세포가 놓일 수 없도록 하는 역할을 합니다.
- 세포에게 영향을 주는 불순물들은 y 좌표에 상관 없이 모두 똑같은 역할입니다.

5번 - 세포 키우기

- 조금 생각해 보면, 영향을 주는 불순물들을 x 좌표 순으로 정렬했을 때 인접한 두 불순물 사이에만 세포를 끼워넣을 수 있음을 알 수 있습니다.
- 끼워넣을 수 있는지 판별은 매우 쉽습니다. 그냥 거리가 $2X$ 이상 떨어져 있으면 끼울 수 있는 것입니다.
- 선분의 양 끝점 처리에 유의하면 됩니다.
 - x 좌표가 $L-X$, $R+X$ 인 불순물 2개를 추가해 주면 매우 쉽게 처리됩니다.

5번 - 세포 키우기

- 입력을 받습니다.
- 좌표에 2를 곱합니다.
- 불순물들을 미리 x좌표 기준으로 정렬합니다.

```
void solve() {
    ll L, R;
    int n;
    cin >> L >> R >> n;
    L *= 2;
    R *= 2;

    vpll a(n);
    for(pll &p : a) {
        cin >> p.x >> p.y;
        p.x *= 2;
        p.y *= 2;
    }
    sort(all(a));
}
```


5번 - 세포 키우기

- 핵심 로직 및 답 출력 부분입니다.
- 코드가 참 간단합니다.
- 미리 불순물들을 정렬해 놓았기 때문에 각 결정 문제를 $O(N)$ 에 풀 수 있습니다.

```
auto f = [&](ll x) {
    vll v;
    v.push_back(L - x);
    for pll &p : a {
        if(abs(p.y) >= x || p.x < L - x || p.x > R + x) continue;
        v.push_back(p.x);
    }
    v.push_back(R + x);

    for(int i = 0; i + 1 < v.size(); i++) {
        if(v[i + 1] - v[i] >= 2 * x) return true;
    }
    return false;
};

ll l = 0, r = ll(1e13);
while(l < r) {
    ll m = (l + r + 1) / 2;
    if(f(m)) l = m;
    else r = m - 1;
}
cout << l << '\n';
}
```