SCPC 1회 1차예선 풀이

서울대학교 컴퓨터공학부 18학번 김동현

SCPC 문제 템플릿

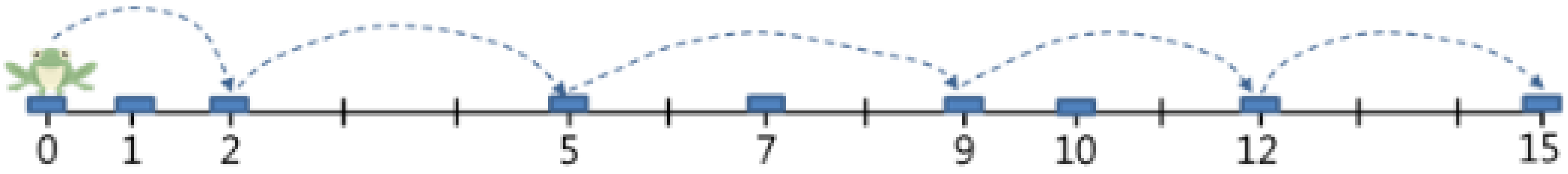
- SCPC 문제는 기본적으로 파일 하나에 테스트 케이스가 여러 개
- 오른쪽과 같이 생긴 코드를 사용하면 편리

- 테스트 케이스가 여러 개 있더라도 시간복잡도 계산에는 크게 신경쓰지 않아도 됨

```
void solve() {
  // 여기에 각 TC 해결하는 코드 작성
}

int main() {
  int tc;
  scanf("%d", &tc);
  for(int i = 1; i <= tc; i++) {
    printf("Case #%d\n", i);
    solve();
  }
  return 0;
}
```

1번 - 개구리 뛰기

- 개구리가 좌표 0에 위치
- N개의 돌이 서로 다른 정수 좌표에 위치 (1 ≤ N ≤ 10^6)
- 한 번의 점프로 최대 K만큼 이동 가능
- 마지막 돌까지 이동하기 위한 최소의 점프 횟수 구하기
  - 마지막 돌까지 도달할 수 없다면 -1 출력



1번 - 개구리 뛰기

- 최소 횟수로 뛰려면 매번 뛸 때마다 최대한 오른쪽으로 뛰면 됨.
- 돌 좌표가 오름차순으로 들어오므로 입력을 정렬할 필요가 없음
- 돌을 순서대로 보면서 현재 있는 돌에서 갈 수 있는 가장 오른쪽 돌로 점프하는 것을 반복하면 됨.
- 시간 복잡도는 O(N)

1번 – 개구리 뛰기 코드

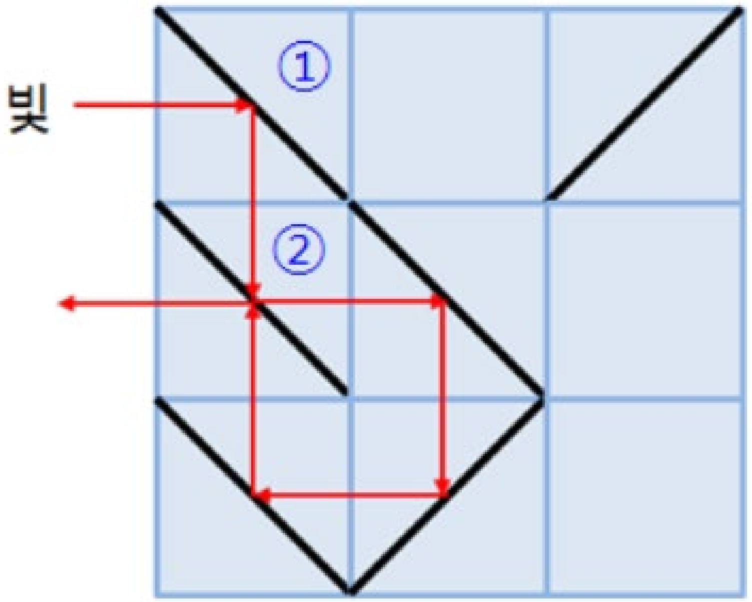
- v[0]에는 0 (시작 위치) 저장
- 한 번 뛸 때마다 가능한 최대한 오른쪽으로 감
- 어떤 돌에서 한 칸도 앞으로 못 갈 경우 불가능(-1 출력)

- Q. for문 안에 for문이 있으니까 O(N^2) 아닌가요?
- A. i와 j 둘 다 감소하지 않고 증가만 하기 때문에 O(N)입니다.

```
void solve() {
  int n;
  cin >> n;

  vector<int> v(n + 1);
  for(int i = 1; i <= n; i++) cin >> v[i];

  int k;
  cin >> k;

  int ans = 0;
  for(int i = 0, j; i < n; i = j) {
    for(j = i; j < n && v[j + 1] <= v[i] + k; j++);
    if(j == i){ ans = -1; break; }
    ans++;
  }
  cout << ans << '\n';
}
```

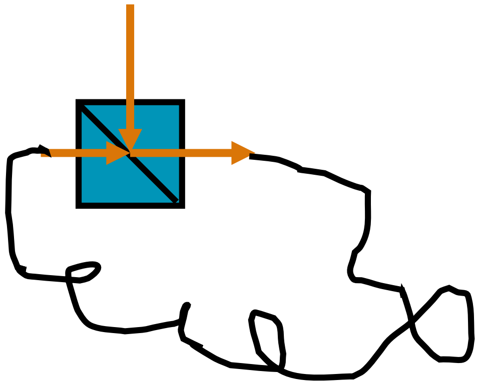
2번 – 방속의 거울

- N × M 격자 ($1 \le N,M \le 1000$)
- 각 칸에는 거울이 있거나 빈 칸
- 거울은 45도 각도로 놓임
- 빛을 맨 위 왼쪽 격자칸 왼쪽에서 쐈을 때, 빛이 거치는 서로 다른 거울이 총 몇 개인지 구하기
  - 한 거울에 두 번 부딪혀도 한 번으로 계산



2번 - 방속의 거울

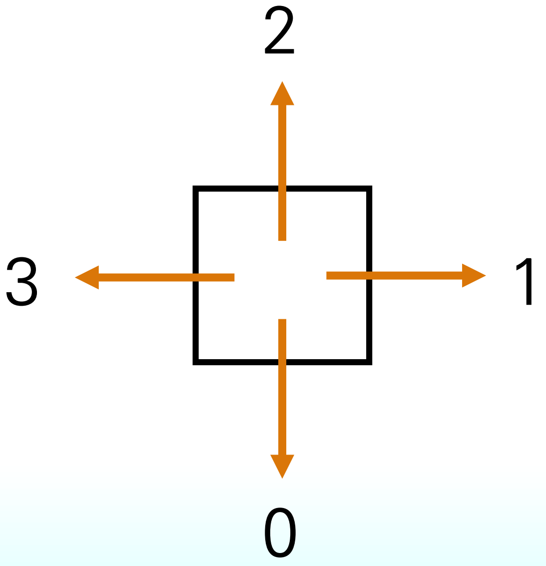
- 기본적인 사실 몇 가지를 먼저 관찰
  - 매 순간 빛의 상태는 (현재 격자 칸 좌표, 방향) 으로 결정됨
  - 바깥에서 들어온 빛은 유한한 step 후에 다시 밖으로 나감
    (다시 못 빠져나온다고 가정하면 사이클이 있어야 한다.
    그런데 사이클을 만들려면 서로 다른 두 방향에서 들어와서
    한 방향으로 나가는 거울 칸이 있어야 함.
    그런 칸은 만들 수 없으므로 모순)
  - 격자 칸 하나를 많아야 4번 방문함
    (5번 이상 방문했으면 2번 거친 상태가 존재 → 사이클 → 모순)

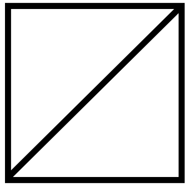
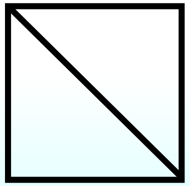


- 결론 : 빛의 움직임을 하나하나 시뮬레이션하면 O(NM)에 풀 수 있다!

2번 - 방속의 거울

- 시뮬레이션을 구현하는 것 자체는 어렵지 않음
- 코드를 어떻게 짧고 간결하게 구현할까?
- 방향을 index로 나타내 보자!



원래 방향	0	1	2	3
	3	2	1	0
	1	0	3	2

2번 – 방속의 거울

- 빈칸, /, \ 각각에 대한 방향 전환 정보를 ndir 배열에 저장
- 시뮬레이션은 아래 과정의 반복
  - 현재 칸에 방문 체크
  - 현재 보고 있는 방향으로 한 칸 이동
    - 이동 후에 격자판 밖으로 벗어났으면 종료
  - 새로 이동한 칸의 종류에 따라서 방향을 전환

```
const static int dx[4] = {1, 0, -1, 0}, dy[4] = {0, 1, 0, -1};
const static int ndir[3][4] = {
  {0, 1, 2, 3},
  {3, 2, 1, 0},
  {1, 0, 3, 2}
};

vector<vint> vis(n, vint(n, 0));
for(
  int x = 0, y = 0, d = 1;
  0 <= x && x < n && 0 <= y && y < n;
  x += dx[d], y += dy[d]
) {
  vis[x][y] = 1;
  d = ndir[b[x][y] - '0'][d];
}
```

3번 – 균일수

- N을 b진법(b≥2)으로 표현했을 때 각 자릿수가 모두 같다면 “균일수”
- N을 균일수가 되게 하는 b의 최솟값을 구하기 (1 ≤ N ≤ 10^9)

N	b		
36	8	⇨	44
63	2	⇨	111111

3번 – 균일수

- b를 하나 정했을 때 그것이 N을 균일수로 만드는지는 진법 변환을 직접 해 봄으로써 바로 알 수 있습니다. 시간 복잡도는 $O(\log_b N)$ 입니다.
- $b \geq N+1$일 때는 무조건 균일수가 되므로 (한 자리 수) 가능한 b의 후보는 $O(N)$개 있습니다.
- N이 최대 10^9으로 애매하게 크기 때문에 모든 값을 다 해 보기는 힘듭니다.
- 계산을 덜 할 수 없을까요?

3번 – 균일수

- N이 b진법으로 3자리 수가 되려면 $b^2+b+1 \le N$을 만족해야 합니다. 즉, b가 대충 $\sqrt{N}$ 을 넘으면 N을 b진법으로 썼을 때 2자리 수가 됩니다.
- N이 2자리 균일수가 될 수 있는지를 따로 빠르게 검사할 수 있다면 $b^2+b+1 \le N$인 b에 대해서만 진법 변환을 수행해도 됩니다!
- N이 2자리 균일수인 b는 어떤 k ($1 \le k < b$)에 대해 $k(b+1)=N$을 만족해야 합니다. 즉, b+1이 N의 약수이고 $N/(b+1)$이 $b-1$ 이하여야 합니다.
- N의 모든 약수는 $O(\sqrt{N})$ 만에 알 수 있으므로, 위의 사실들을 종합하면 전체 문제를 $O(\sqrt{N}\log N)$에 해결할 수 있습니다!
  - log는 진법 변환 부분 때문에 붙습니다.

3번 – 균일수

- ans 변수는 n+1로 초기화
  (한 자리 균일수 중 가장 작은 b)

- 첫 번째 for문 : 2자리 균일수에 대해 고려

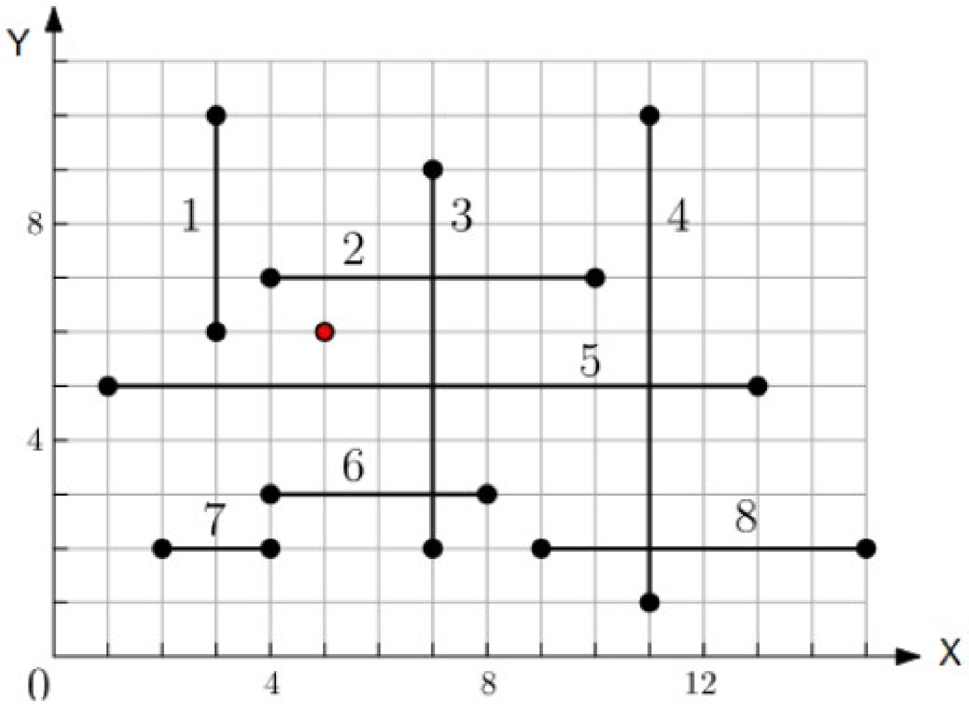
- 두 번째 for문 : 3자리 이상 균일수에 대해 고려

```
int ans = n + 1;

for(int i = 1; i * i <= n; i++) {
  if(n % i == 0) {
    if(i <= n / i - 2) ans = min(ans, n / i - 1);
  }
}

for(int b = 2; b * b + b + 1 <= n; b++) {
  int valid = 1;
  int digit = n % b;
  int m = n;
  while(m) {
    if(m % b != digit){ valid = 0; break; }
    m /= b;
  }
  if(valid) ans = min(ans, b);
}
```

4번 - 회로판 위의 배터리

- N개의 수직/수평 선분 있음
  (1 ≤ N ≤ 100)
- 좌표평면 위에 점을 하나 찍어야 함
  (실수 좌표 가능)
- 찍은 점의 좌표를 (x, y)라 하고, 각 선분의 양 끝점을 (a_i, b_i), (c_i, d_i)라 할 때 다음을 최소화하는 점을 찾아서 그 값을 출력:

$$\max_{1\le i\le n} \min \begin{Bmatrix} \max\{|a_i - x|, |b_i - y|\}, \\ \max\{|c_i - x|, |d_i - y|\} \end{Bmatrix}$$

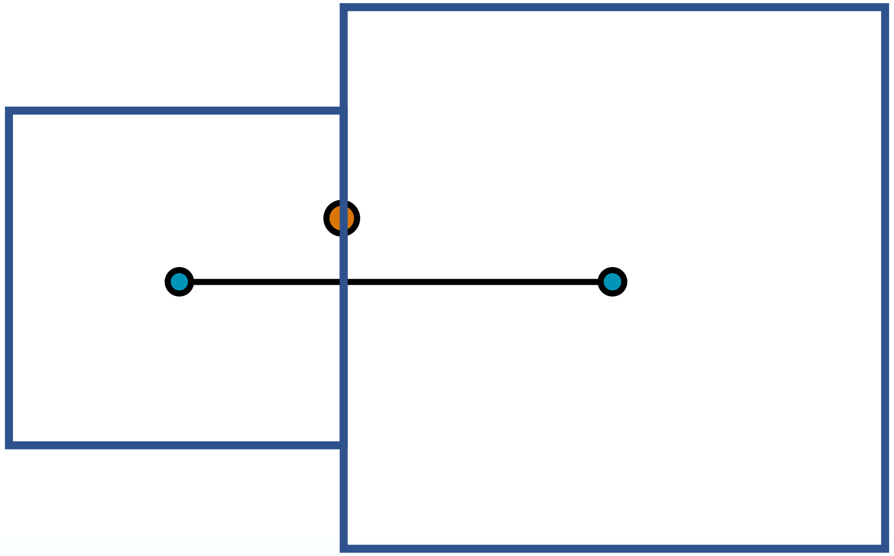


4번 – 회로판 위의 배터리

- 문제가 참 복잡합니다..
- 우선 문제에서 최소화하라고 시키는 값의 의미를 살펴봅시다.

$\max\{|a_i - x|, |b_i - y|\}$ (Chebyshev 거리)

(a_i, b_i)를 중심으로 하는 정사각형의 둘레가 (x, y)를 지날 때, 그 정사각형의 한 변 길이의 절반



어떤 점으로부터 Chevyshev 거리가 x 이하인 점들의 집합 = 그 점을 중심으로 하는 한 변 길이 2x의 정사각형!

4번 – 회로판 위의 배터리

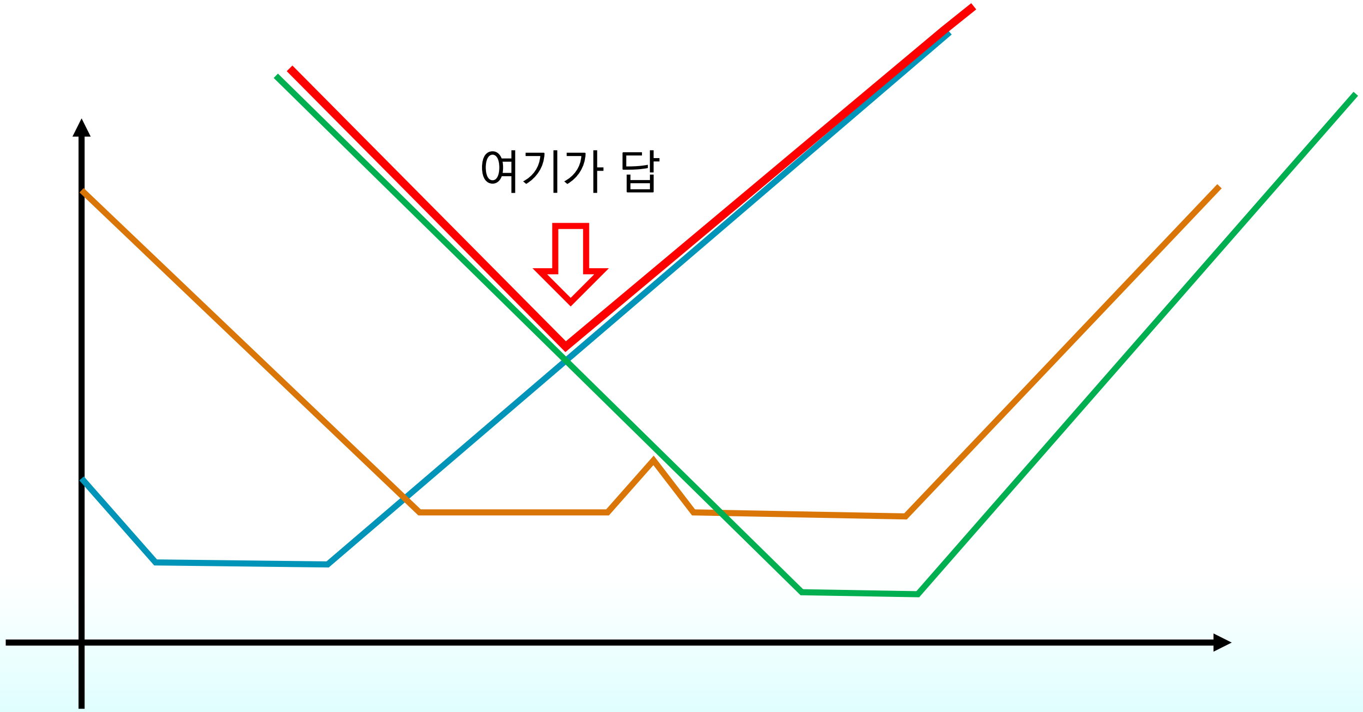
- 문제를 이대로 풀기에는 조금 힘들어 보입니다.
- 이럴 때 쓰는 기법 중 하나가 “답에 대한 이분 탐색” 입니다.
  - 흔히 Parametric Search라고 부릅니다.
- 답이 정확히 얼마인지 바로 구하기는 힘들지만,
  답이 어떤 수 x 이하인가를 판별하기는 좀 더 쉽습니다.
- 이것을 할 수 있다면, 답이 얼마인지 이분탐색을 할 수 있습니다.
  - 지금까지 답이 [s, e] 구간 내에 있음을 알아냈다고 합시다.
  - m = (s+e)/2라 하고, 답이 m 이하인지 아닌지 알아봅시다.
  - 그 결과에 따라 답의 구간을 절반으로 줄일 수 있습니다.

4번 – 회로판 위의 배터리

- 더 가기 전에 확인해야 할 중요한 관찰이 하나 있습니다:
  답은 항상 $\frac{k}{2}$ 꼴입니다. (k는 정수)
  - 정확히 말하면, $(\frac{a}{2}, \frac{b}{2})$ (a, b는 정수) 꼴의 점만 고려해도 답을 찾을 수 있습니다.
- 이를 엄밀하게 증명하기는 생각보다 어렵지만, 대충 그렇다고 가정하고 풀면 풀립니다(?)
- 그래도 증명을 하면 좋으니, 나름대로 증명하는 방법을 설명하겠습니다.

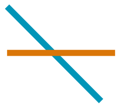
4번 - 회로판 위의 배터리

- 새로 찍을 점의 y좌표를 임의의 실수로 고정했을 때, x좌표에 따라 답이 어떻게 바뀌는지를 살펴봅시다.
- 아래 그림과 같이 기울기가 +1, 0, −1로만 이루어진 꺾은선들의 최댓값이 답이 됩니다.



4번 - 회로판 위의 배터리

- 이런 꺾은선들로 이루어진 “산봉우리”에서 가장 낮은 곳은 어디 있을까요?
- “계곡”, 즉 최댓값에 해당하는 선분의 기울기가 증가하는 지점입니다.
  - −1 → +1로 증가할 수도 있고, −1 → 0 이나 0 → 1도 가능합니다.


  - 이 지점들만 고려해도 전체 최솟값을 찾을 수 있습니다.

- (1) −1 → +1의 경우, 그런 지점의 x좌표는 무조건 $\frac{k}{2}$ 꼴입니다.
  - 각 점의 좌표가 정수이기 때문에, 기울기가 ±1인 선분의 y절편은 정수입니다.

4번 - 회로판 위의 배터리

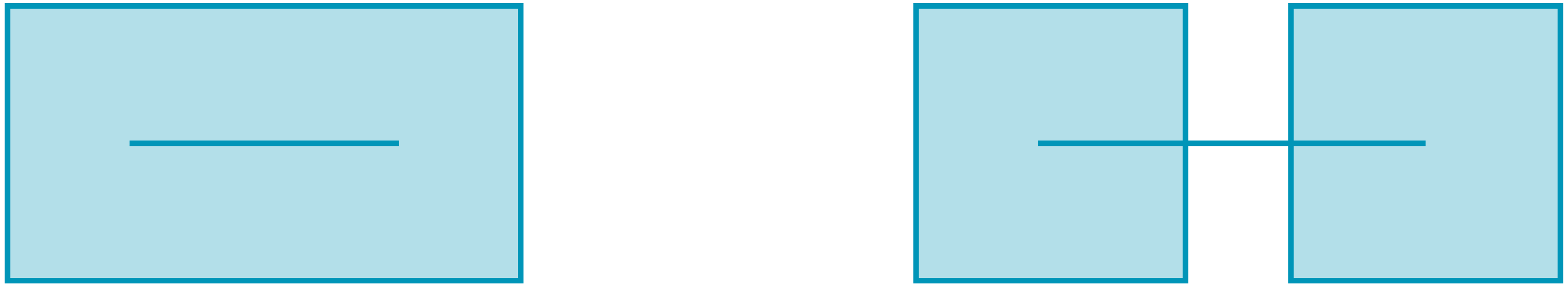
- (2) −1 → 0이나 0 → 1의 경우, 기울기 0인 선분을 따라 움직이면 무조건 $\frac{k}{2}$ 꼴의 x좌표에 도달할 수 있습니다.
  - 기울기 0인 선분들을 잠시 치웠다고 생각해 봅시다.
  - 현재 걸려 있던 기울기 0인 선분 아래를 보면, 꺾은선이 영원히 내려갈 수는 없으니 내려갔다가 올라오는 부분이 무조건 하나 있습니다.
  - (1)에 의해 그 부분의 좌표가 $\frac{k}{2}$ 꼴입니다! 이제 기울기 0인 직선을 다시 가져다 놓고 그 위로 가면 답이 변하지 않으면서 $\frac{k}{2}$ 꼴 좌표에 도달하게 됩니다.
- 이제 새로 찍을 점의 x좌표를 $\frac{k}{2}$ 꼴만 고려해도 된다는 것을 알았습니다.
- 같은 논리를 y좌표에 대해서 적용할 수 있습니다!

4번 - 회로판 위의 배터리

- 이제 입력으로 받은 모든 점의 좌표를 2배 하면 새로 찍는 점의 좌표가 (정수, 정수) 꼴이라고 가정할 수 있습니다.
- 드디어 Parametric Search를 구현할 수 있겠습니다...
- 답이 어떤 정수 x 이하인지 알고 싶다고 합시다.
- 모든 선분에 대해 선분의 각 끝점과 새로 찍을 점 간의 Chevyshev 거리 중 적어도 하나가 x 이하이면 됩니다.
- 각 선분에 대해 새로 찍을 점이 존재할 수 있는 곳이 영역 형태로 나오므로 이들 전체의 교집합이 존재하는지를 판별하면 될 것 같습니다.

4번 - 회로판 위의 배터리

- 영역의 모양은 선분의 길이에 따라 직사각형 1개 또는 2개로 나타납니다.



- 직사각형 2개로 나타나는 선분들이 있어서 아주 쉽게는 할 수 없습니다.

4번 - 회로판 위의 배터리

- 좌표 압축 + 2차원 누적합을 사용해서 이 문제를 O(N^2)에 해결할 수 있습니다.

(1) 좌표 압축

- 영역을 표현하는 직사각형이 총 O(N)개 있으니, 직사각형의 경계를 나타내는 x좌표와 y좌표 역시 O(N)개씩 있습니다.
- 모든 영역의 교집합 내부에 점을 하나 찍었다고 할 때, 그 점의 x좌표와 y좌표를 각각 직사각형 경계에 해당하는 좌표 중 하나로 옮길 수 있습니다.
  - 교집합의 "한 구석"으로 점을 끌어다 놓는다고 생각하면 됩니다.

4번 - 회로판 위의 배터리

- 즉, O(N^2)개의 격자점에 대해서만 그 점이 모든 영역의 교집합에 속하는지 여부를 알면 됩니다.
- 좌표 압축이란 이런 식으로 N개의 좌표값이 있을 때 이들의 순서 관계를 유지하면서 범위를 1~N으로 바꾸는 것입니다.
- 정렬을 통해 간단히 해결할 수 있으며, 자세한 방식은 코드를 보면서 다시 설명하겠습니다.

-1, 9, 9, 100, 12345, 987654321 ⇨ 1, 2, 2, 3, 4, 5

4번 – 회로판 위의 배터리

(2) 2차원 누적합

- 각 점에 대해 그 점이 모든 영역에 다 속하는지는 어떻게 알 수 있을까요?
- 각 선분에 해당하는 영역마다, 직사각형 내의 점에 +1씩을 더해 준다고 합시다.
- 모든 선분에 대해 이 작업을 한 뒤, 정확히 N이 적힌 점이 있다면 그 점이 바로 모든 영역의 교집합에 속하는 점이 됩니다.

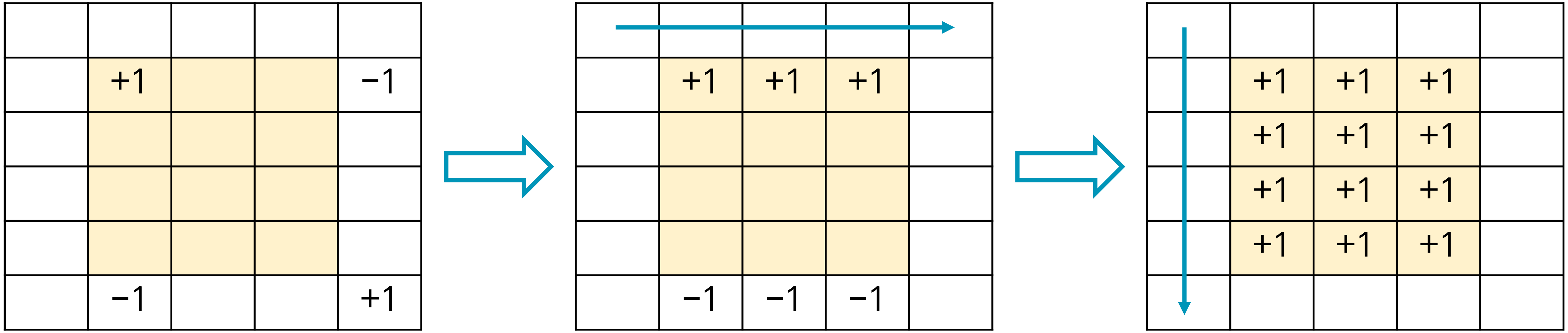
0	1	1	1
1	2	2	1
1	2	2	1
1	1	1	0

4번 - 회로판 위의 배터리

- 직사각형을 총 O(N)개 만들고, 각 직사각형 안에 점이 O(N^2)개 있으므로 일일이 1씩 더해주면 O(N^3)이 됩니다.
    - N ≤ 100 이라서 지금 생각해 보니 될 거 같기도 합니다(?)
- (아무튼) 2차원 누적합을 이용하면 이를 O(N^2)만에 할 수 있습니다!
- 각 직사각형에 대해, 배열에 우선 다음과 같이 체크를 해 줍니다.

4번 - 회로판 위의 배터리

- 체크를 다 했으면, 가로로 한 번 누적하고 세로로 한 번 누적하면 됩니다.
- 각 직사각형에 대해 대충 아래와 같은 식으로 작동합니다.



가로로 누적

세로로 누적

4번 – 회로판 위의 배터리

- 결국 답이 x 이하인지 판별하는 문제를 O(N^2)에 풀 수 있으니, 답에 대한 이분탐색을 수행하면 전체 문제를 O(N^2 log(좌표범위)) 에 풀 수 있습니다.
- N ≤ 100 이라서 의도한 풀이는 이것과 다른 것 같기도 한데, 일단 이렇게 푸는 게 제가 생각한 가장 빠른 풀이입니다.
  - 혹시 더 빠르게 푸신 분이 있다면 알려 주시면 감사하겠습니다..

4번 - 회로판 위의 배터리

- 입력 받는 부분 + 이분탐색 하는 부분입니다.
-
```
using pii = pair<int, int>;
#define x first
#define y second
```

이런 코드를 사용하면 STL pair를 편하게 사용할 수 있습니다.

```
int n;
cin >> n;

vector<pair<pii, pii>> v(n);
for(auto &p : v) {
  cin >> p.x.x >> p.x.y >> p.y.x >> p.y.y;
  p.x.x *= 2; p.x.y *= 2; p.y.x *= 2; p.y.y *= 2;
  if(p.x > p.y) swap(p.x, p.y);
}

ll l = 0, r = int(2.1e8);
while(l < r) {
  ll m = (l + r) / 2;
  if(f(m)) r = m;
  else l = m + 1;
}
if(l % 2 == 0) cout << l / 2 << '\n';
else cout << l / 2 << ".5\n";
```

4번 - 회로판 위의 배터리

- 함수 f (답이 x 이하가 될 수 있는지 판별하는 함수) 의 첫 부분입니다.
- C++ Lambda 함수를 사용하였습니다.
- Lambda 함수 안에서 Lambda 함수를 정의할 수 있습니다…
- `#define all(v) v.begin(),v.end()`

  이런 매크로를 정의하면 편합니다.

```
auto f = [&](ll x) {
  vint xs, ys;
  vector<pair<pii, pii>> rect;
  auto makerect = [&](pii a, pii b) {
    rect.emplace_back(a, b);
    xs.push_back(a.x);
    xs.push_back(b.x);
    ys.push_back(a.y);
    ys.push_back(b.y);
  };
  for(auto &p : v){
    if(abs(p.x.x - p.y.x) + abs(p.x.y - p.y.y) <= 2 * x) {
      makerect(pii(p.x.x - x, p.x.y - x), pii(p.y.x + x, p.y.y + x));
    } else {
      makerect(pii(p.x.x - x, p.x.y - x), pii(p.x.x + x, p.x.y + x));
      makerect(pii(p.y.x - x, p.y.y - x), pii(p.y.x + x, p.y.y + x));
    }
  }
  sort(all(xs));
  sort(all(ys));
```

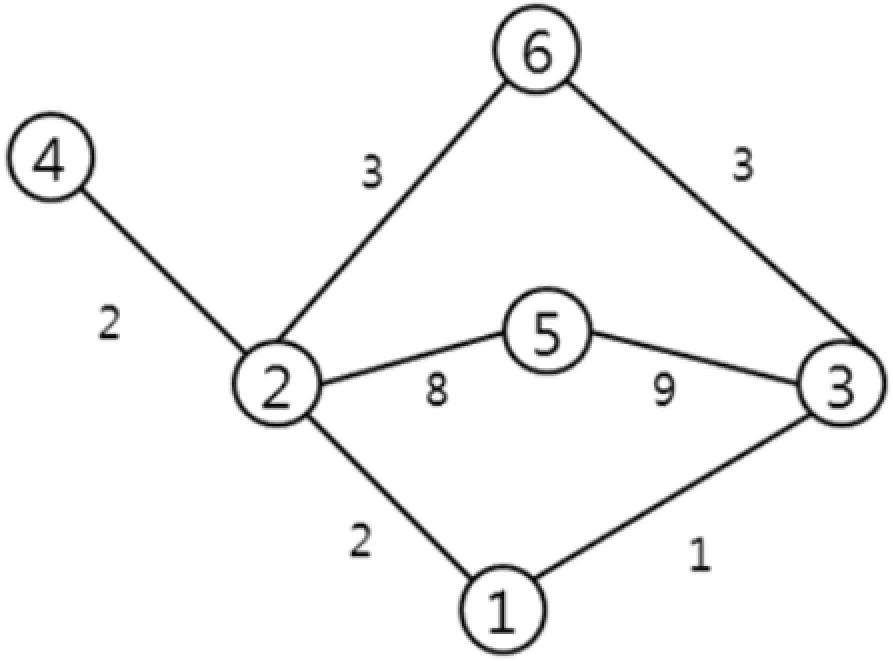
4번 – 회로판 위의 배터리

- 좌표 압축 + 2차원 누적합을 수행하는 부분입니다.
- cp 함수는 정렬된 좌표 배열과 좌표 값이 주어지면 압축된 값을 반환합니다.
- 2차원 누적합을 수행한 뒤 누적된 배열에 n 값이 있는지 검사합니다.

```
  vector<vint> d(xs.size() + 1, vint(ys.size() + 1, 0));
  auto cp = [](vint &v, ll x) {
    return int(lower_bound(all(v), x) - v.begin());
  };
  for(auto &r : rect) {
    int xl = cp(xs, r.x.x), xr = cp(xs, r.y.x);
    int yl = cp(ys, r.x.y), yr = cp(ys, r.y.y);
    d[xl][yl]++;
    d[xl][yr + 1]--;
    d[xr + 1][yl]--;
    d[xr + 1][yr + 1]++;
  }

  for(int i = 0; i < xs.size(); i++) for(int j = 1; j < ys.size(); j++)
    d[i][j] += d[i][j - 1];
  for(int j = 0; j < ys.size(); j++) for(int i = 1; i < xs.size(); i++)
    d[i][j] += d[i - 1][j];
  for(vint &v : d) if(*max_element(all(v)) == n) return 1;
  return 0;
};
```

5번 – 캠퍼스와 도로(1)

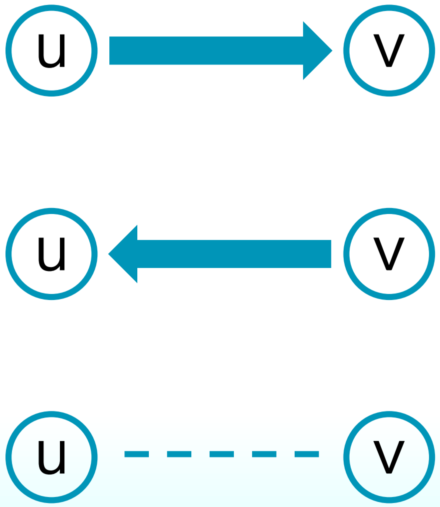
- 정점 N개와 양방향 가중치 간선 M개가 있는 그래프가 주어짐.
  - 1 ≤ N ≤ 1000, 1 ≤ M ≤ 5000
- 이 그래프 위의 임의의 두 정점 쌍에 대해, 차량은 두 정점 쌍 간의 최단경로로만 움직임
  - 단, 최단 경로가 여러 개라면 어떤 경로든 이용할 수 있음
- 이 조건 하에서 차량이 절대로 통과하지 않는 정점을 모두 구하기
  - 시작/끝 정점은 통과한 정점이 아님



5번 - 캠퍼스와 도로(1)

- 시작점을 하나 고정하고 생각해 봅시다.
- Dijkstra 알고리즘 등으로 나머지 각 정점까지의 최단 거리를 구합니다.
- 각 도로에 대해, 그 도로는 어느 한 방향으로만 이용되거나 아예 이용되지 않습니다.

d[u] : 시작점에서 u까지 최단 거리
d[v] : 시작점에서 v까지 최단 거리
c   : u-v 간선의 가중치



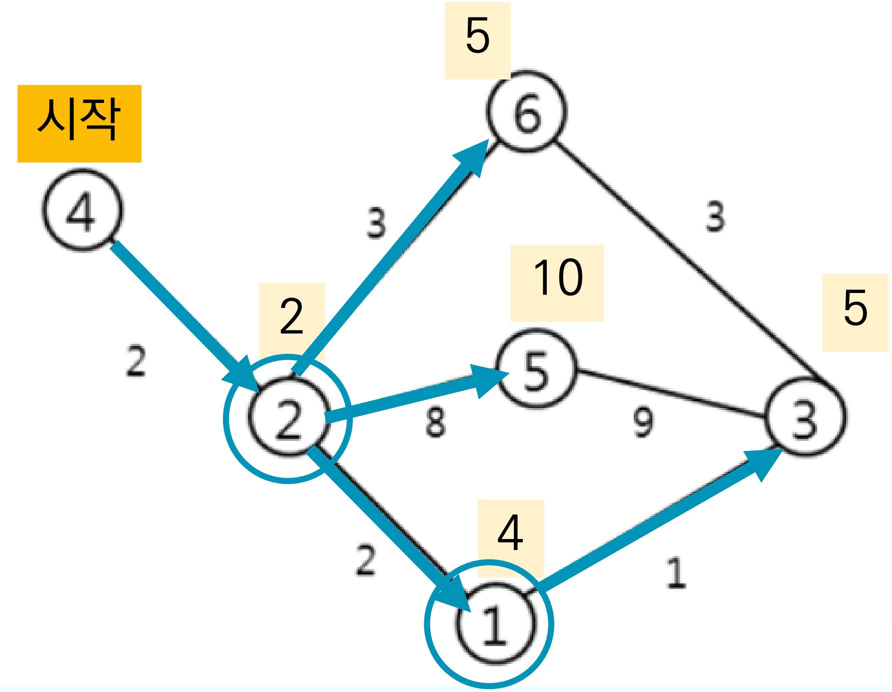
$d[u] + c = d[v]$

$d[u] = d[v] + c$

그 외

5번 – 캠퍼스와 도로(1)

- 어떤 정점에 대해 그 정점이 통과 정점이 될 수 있다는 것은,
  그 정점에서 나가는 방향으로 이용되는 도로가 존재한다는 것입니다.



4번 정점이 출발 정점일 때,
1, 2번 정점은 통과 정점이 될 수 있음

5번 – 캠퍼스와 도로(1)

- 모든 정점을 시작 정점으로 잡아 본 뒤, 어떤 정점에서 출발하더라도 통과 정점이 될 수 없는 정점들이 답이 됩니다.
- 시간 복잡도는 O(NMlogM) 입니다.
  - Dijkstra Algorithm의 가장 널리 알려진 구현은 O(MlogM)에 작동합니다.
  - Wikipedia 등을 보면 Fibonacci heap을 이용할 시 O(M+NlogN)에 구현할 수 있다고는 하는데, 아무도 그렇게 안 짜는 듯 합니다..

5번 - 캠퍼스와 도로(1)

- 입력 받는 부분입니다.
- vpii는 vector<pii> 입니다.
- 각 간선의 도착점과 가중치를 동시에 저장하는 여러 방법 중 하나입니다.

```
int n, m;
cin >> n >> m;

vector<vpii> e(n + 1);
for(int i = 0; i < m; i++) {
  int x, y, z;
  cin >> x >> y >> z;
  e[x].emplace_back(y, z);
  e[y].emplace_back(x, z);
}
```

5번 – 캠퍼스와 도로(1)

- Dijkstra 알고리즘의 구현입니다.
- Priority_queue는 기본적으로 max-heap이기 때문에, min-heap으로 쓰려면 오른쪽과 같이 써 주면 됩니다.
  - (현재 거리, 현재 정점)의 pair를 담으면 됩니다.

```
vector<int> d(n + 1), chk(n + 1, 0);
for(int st = 1; st <= n; st++) {
  fill(all(d), int(1e9));
  d[st] = 0;
  priority_queue<pii, vector<pii>, greater<pii>> pq;
  pq.emplace(0, st);

  while(!pq.empty()) {
    pii c = pq.top();
    pq.pop();
    if(c.x != d[c.y]) continue;

    for(pii i : e[c.y]) {
      if(d[i.x] > c.x + i.y) {
        d[i.x] = c.x + i.y;
        pq.emplace(d[i.x], i.x);
      }
    }
  }
```

5번 - 캠퍼스와 도로(1)

- For문의 아랫부분 및 답 출력 부분 입니다.
- 모든 시작 정점에 대해 다 해 봐도 체크가 안 된 정점들을 모두 출력하면 됩니다.

```
    for(int i = 1; i <= n; i++) {
      if(i == st) continue;
      for(pii j : e[i]) {
        if(d[j.x] == d[i] + j.y) {
          chk[i] = 1;
          break;
        }
      }
    }
}

cout << (count(all(chk), 0) - 1);
for(int i = 1; i <= n; i++) if(!chk[i]) cout << ' ' << i;
cout << '\n';
```