SCPC 6회 1차예선 풀이 + SCPC 1회 2차예선 풀이

서울대학교 컴퓨터공학부 18학번 김동현

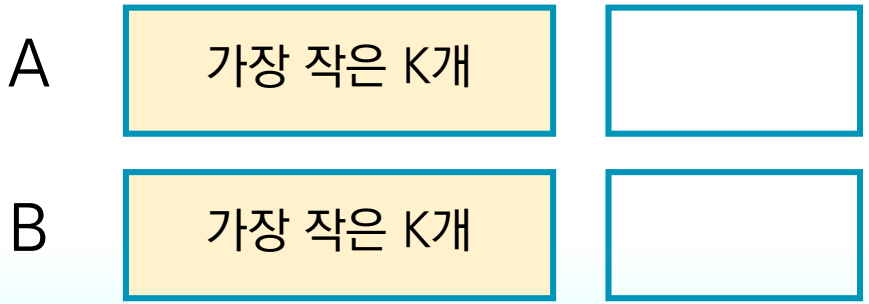
SCPC 6회 1차예선 풀이

1번 - 다이어트

- 길이 N의 배열 A, B가 있다.
- 총 K일 동안, 하루에 배열 A와 B에서 원소를 하나씩 선택하려고 한다.
- 한 번 선택한 원소는 앞으로 다시 선택할 수 없다.
- K일 동안 각 날짜의 (A에서 고른 값 + B에서 고른 값) 중 최댓값을 최소화하라.

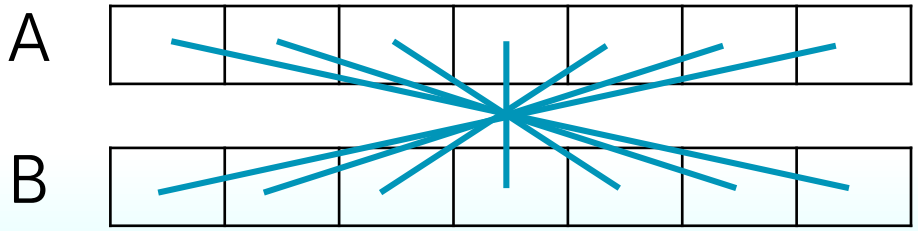
1번 - 다이어트

- 우선, A와 B에서 각각 가장 작은 K개 중에서만 고르면 좋을 것 같습니다.
- 가장 작은 K개를 고르지 않은 날이 있다면, 그것을 가장 작은 K개 중 하나로 교체해도 답이 나빠지지 않습니다.
- 이제, 각 날짜마다 어떤 쌍을 골라야지 합의 최댓값이 최소화될지 생각해 봅시다.



1번 - 다이어트

- A와 B가 각각 오름차순 정렬되어 있다고 합시다.
- 어떤 두 날짜에 각각 A[a]+B[b], A[c]+B[d] (a<c, b<d)를 골랐다고 합시다.
- 이를 A[a]+B[d], A[c]+B[b]를 고른 것으로 바꾸어 주면 답이 나빠지지 않음을 알 수 있습니다.
- 이런 쌍을 계속 풀어주다 보면, 결국 아래 그림처럼 짝을 짓는 것이 최적임을 알 수 있습니다.



1번 - 다이어트

- 코드는 매우 간단합니다.

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

  vint a(n), b(n);
  for(int &x : a) cin >> x;
  for(int &x : b) cin >> x;
  sort(all(a));
  sort(all(b));

  int ans = 0;
  for(int i = 0; i < k; i++) ans = max(ans, a[i] + b[k - 1 - i]);
  cout << ans << '\n';
}
```

2번 - 카드 게임

- A와 B가 번갈아가면서 게임을 한다. A가 먼저 행동한다.
- 1 이상 K 이하의 수가 하나씩 써진 카드들이 각각 N개씩 두 개의 더미로 쌓여 있다. (1 ≤ N ≤ 3,000)
- 각 턴마다 A 또는 B는 두 더미 중 하나를 골라 위에서부터 카드를 1장 이상 가져가야 한다. 단, 가져가는 카드들에 적힌 수의 합이 K 이하가 되어야 한다.
- 카드를 더 이상 가져가지 못 하는 사람이 진다.
- 카드 더미의 모든 가능한 (N+1)^2개의 상태들 중, 둘 다 최선을 다할 때 A가 이기는 상태의 수와 B가 이기는 상태의 수를 각각 구하여라.

2번 - 카드 게임

- 두 명이서 게임을 하는 문제는 매우 유명한 문제입니다.
- 가장 쉬운 형태의 2인 게임 문제는 보통 다음과 같은 DP를 정의하면 매우 쉽게 풀립니다.
- 현재 특정 상태에 있을 때, 지금 자기 차례인 사람이 이기는가?
- 이렇게 생긴 DP를 계산할 때는 아래 사실을 기억하면 좋습니다.
- 다음으로 갈 수 있는 상태들 중 지는 상태가 하나라도 있다면 지금 상태는 이기는 상태이다.
- (= 다음으로 갈 수 있는 상태들이 모두 이기는 상태라면 지금 상태는 지는 상태이다.)

2번 - 카드 게임

- 이 문제에서는 (무려 지문에서 언급하듯이) 상태가 총 O(N^2)개 있으니, DP를 다음과 같이 정의하여 계산해 봅시다.
- D[x][y] : 두 카드 더미가 각각 x개, y개 남은 상태에서, 지금 차례인 사람이 이기는가? (이긴다면 1, 진다면 0)
- D[0][0] = 0임을 쉽게 알 수 있습니다.
- 현재 상태에서 갈 수 있는 다음 상태는 아래와 같습니다.
  - 첫 번째 카드더미의 (a+1 ~ x번째 수의 합)이 K를 넘지 않는 (z, y)
  - 첫 번째 카드더미의 (b+1 ~ y번째 수의 합)이 K를 넘지 않는 (x, b)

2번 - 카드 게임

- 각 상태마다 가능한 다음 상태가 최대 O(N)개 있기 때문에 DP를 그냥 계산하면 O(N^3)입니다.
- 최적화를 위해서는 적절한 전처리가 필요합니다.
- 두 더미가 독립적이기 때문에, (x, y)에서 가장 많이 갈 수 있는 (z, y)와 (x, w)에 해당하는 z 및 w는 각각 O(N)번만 미리 계산해 두면 됩니다.
- DP를 계산하면서 각 행/열마다 D[x][0] ~ D[x][y]의 합 같은 것을 저장해 놓으면 다음 상태들 중에 DP 값이 0인 상태가 있는지 여부를 O(1)에 판별할 수 있습니다.

2번 - 카드 게임

- 입력 및 전처리 부분입니다.
- rs, cs 배열은 dp의 행/열 누적합 배열입니다.
- al, bl 벡터는 각 상태마다 갈 수 있는 다음 상태가 어디까지인지를 저장합니다.
  - 이 배열은 O(N^2)에 채워도 됩니다.

```
const int N = 3005;
int d[N][N], rs[N][N], cs[N][N];

void solve() {
  int n, k;
  cin >> n >> k;

  for(int i = 0; i <= n; i++) {
    fill(d[i], d[i] + n + 1, 0);
    fill(rs[i], rs[i] + n + 1, 0);
    fill(cs[i], cs[i] + n + 1, 0);
  }

  vint a(n + 1), b(n + 1);
  for(int i = 1; i <= n; i++) cin >> a[i];
  for(int i = 1; i <= n; i++) cin >> b[i];

  vint al(n + 1), bl(n + 1);
  for(int i = 1; i <= n; i++) {
    al[i] = bl[i] = i;
    int s = 0;
    for(int s = 0; al[i] >= 0 && s <= k; s += a[al[i]--]);
    for(int s = 0; bl[i] >= 0 && s <= k; s += b[bl[i]--]);
  }
```

2번 - 카드 게임

- 실제 DP값을 채우는 부분입니다.
- 인덱스 끝 부분에 대한 처리를 잘 해주어야 합니다.

```
  int awin = 0, bwin = 0;
  for(int i = 0; i <= n; i++) for(int j = 0; j <= n; j++) {
    if(i && cs[i - 1][j] - (al[i] < 0 ? 0 : cs[al[i]][j]) < i - 1 - al[i]) d[i][j] = 1;
    if(j && rs[i][j - 1] - (bl[j] < 0 ? 0 : rs[i][bl[j]]) < j - 1 - bl[j]) d[i][j] = 1;
    if(i + j == 0) d[i][j] = 1;
    (d[i][j] ? awin : bwin)++;
    rs[i][j] = (j ? rs[i][j - 1] : 0) + d[i][j];
    cs[i][j] = (i ? cs[i - 1][j] : 0) + d[i][j];
  }

  cout << awin << ' ' << bwin << '\n';
}
```

3번 - 사다리 게임

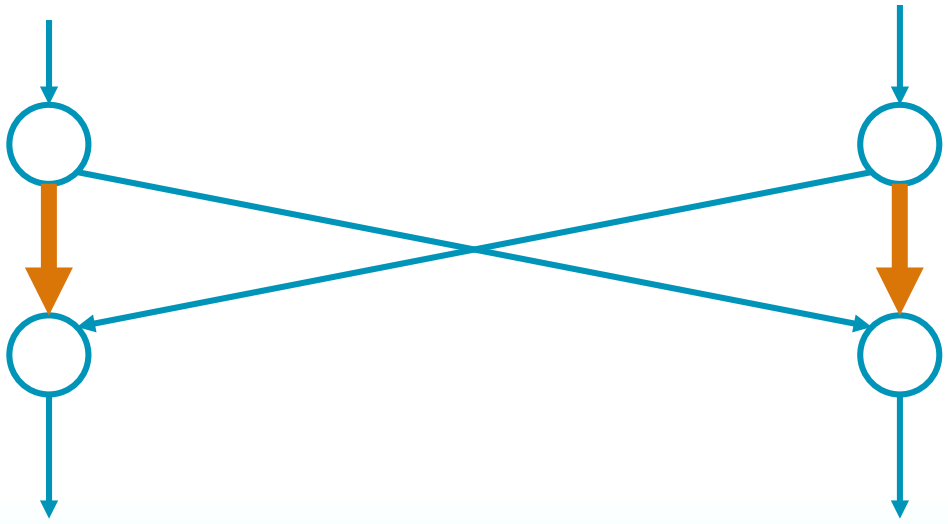
- 세로 기둥이 N개, 가로 기둥이 K개 있는 사다리가 있습니다.
  - 제한이 정확히 기억은 안 나는데 아마 둘 다 2,000 정도였던 거 같습니다.
- 모든 가로 기둥의 y좌표는 다릅니다.
- 쿼리가 총 Q (≤ 10^5) 번 주어지는데, 각 쿼리마다 두 정수 x, y가 주어집니다.
- x번 기둥의 맨 위에서 사다리를 타고 내려가 y번 기둥에 마지막으로 도착하기 위해 최소 몇 개의 가로 기둥을 제거해야 하는지를 출력하세요.

3번 - 사다리 게임

- DP로 해결하는 방법이 있고, 그래프로 적절히 모델링하여 푸는 방법이 있습니다.
- 두 방법 모두 근본적으로 같은 방법입니다.
- DP로 해결하는 쪽이 코드가 훨씬 짧긴 합니다.
- 저는 그래프로 풀었으니 그래프 풀이를 설명하겠습니다(?)

3번 - 사다리 게임

- 사다리를 위에서 아래로 타고 내려가면서, 각 가로 기둥마다 아래 그림과 같이 정점과 간선을 추가해주면 됩니다.
  - 정점을 약간 낭비하긴 하는데, 별로 상관 없습니다.



파란(얇은) 간선 : 비용 0 (제거할 필요 없음)

노란(두꺼운) 간선 : 비용 1 (제거해야 함)

3번 - 사다리 게임

- 각 시작점마다 최단거리 알고리즘을 한 번 씩 수행해 주면 모든 (시작점, 끝점) 쌍에 대한 답을 O(정점 수 * 간선 수) 정도에 구해줄 수 있습니다.
  - Dijkstra를 써도 통과한다고는 합니다.
  - 저는 0-1 BFS(후술)를 구현하였습니다.
- 정점 수와 간선 수 모두 N+K에 비례합니다.
- 주어진 그래프가 DAG (사이클이 없음) 임을 이용해서, 가로 간선을 하나 추가할 때마다 O(N)번의 DP 갱신으로 모든 (시작점, 끝점) 쌍에 대한 답을 업데이트 해 줄 수도 있습니다.

3번 - 사다리 게임

- 0-1 BFS란, 간선의 가중치가 0 또는 1일 때 BFS와 유사하게 최단 거리를 O(N+M)에 구할 수 있는 알고리즘입니다.
- BFS 코드에서 몇 줄만 바꾸고 추가하면 0-1 BFS가 됩니다.
  - queue → deque
  - deque에 정점을 새로 넣을 때, 이번에 본 간선의 가중치가 1이었다면 push_back (뒤에 추가), 가중치가 0이었다면 push_front (앞에 추가)
- 뒤 슬라이드의 코드를 참고하세요.

3번 - 사다리 게임

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

int v = n + 4 * k;
vint lst(n);
iota(all(lst), 0);
vector<vpii> e(v);

for(int i = 0; i < k; i++) {
  int x, y;
  cin >> x >> y;
  x--; y--;
  int xn = n + 4 * i, yn = n + 4 * i + 2;
  e[lst[x]].emplace_back(xn, 0);
  e[lst[y]].emplace_back(yn, 0);
  e[xn].emplace_back(xn + 1, 1);
  e[xn].emplace_back(yn + 1, 0);
  e[yn].emplace_back(xn + 1, 0);
  e[yn].emplace_back(yn + 1, 1);
  lst[x] = xn + 1;
  lst[y] = yn + 1;
}

vector<vint> qry(n);
for(int i = 0; i < m; i++) {
  int x, y;
  cin >> x >> y;
  qry[x - 1].push_back(y - 1);
}
```

```
vint d(v);
deque<int> dq;
int ans = 0;
for(int i = 0; i < n; i++) {
  fill(all(d), v + 1);
  d[i] = 0;
  dq.push_back(i);

  while(!dq.empty()) {
    int x = dq.front();
    dq.pop_front();

    for(const pii &p : e[x]) {
      if(d[p.x] <= d[x] + p.y) continue;
      d[p.x] = d[x] + p.y;
      if(p.y) dq.push_back(p.x);
      else dq.push_front(p.x);
    }
  }

  for(int j : qry[i]) ans += (d[lst[j]] > v ? -1 : d[lst[j]]);
}

cout << ans << '\n';
```

0-1 BFS

4번 - 범위 안의 숫자

- 0~9의 숫자로 이루어진 길이 N(≤ 50,000)의 문자열이 주어집니다.
- 문자열의 특정 위치 딱 하나를 골라 거기 적힌 숫자를 '1'로 바꿀 수 있습니다. (안 해도 됩니다.)
- 이제, 주어진 정수 k (1 ≤ k ≤ min(9,N)) 에 대해, 문자열의 모든 길이가 k인 연속한 부분문자열을 10진수로 읽어서 N−k+1개의 수를 얻습니다. (leading zero가 있어도 무방합니다)
- 주어진 정수 m이 있을 때, 정수 a를 마음대로 골라서 N−k+1개의 수들 중 [a, a+m] 구간에 포함된 수가 몇 개인지 셉니다. 이 때 최대로 포함시킬 수 있는 개수를 X라고 합시다.
- 숫자를 바꿀 위치를 잘 선택해서 (또는 바꾸지 않아서) X를 최대화하세요.

4번 - 범위 안의 숫자

- 문자열의 숫자 하나를 바꿀 때, N-k+1개의 수들 중 바뀌는 (있었다가 없어지거나, 없었다가 생기는) 수들의 개수는 O(k)개 임을 알 수 있습니다.
- 결국, 모든 자리에서 다 숫자를 다 바꿔 보았을 때 나타나는 수의 총 가짓수는 O(Nk) 개 있습니다.
- 어떤 수 집합에 대해서 길이 m짜리 구간에 가장 많이 포함시킬 수 있는 수의 개수, 즉 X를 구하면서 수의 삽입/삭제 쿼리를 빠르게 처리할 수 있는 자료 구조가 있으면 좋을 것 같습니다.

4번 - 범위 안의 숫자

- N-k+1개의 수들이 정해졌을 때 X를 어떻게 구하는지 생각해 봅시다.
- 여러 방법이 있겠지만, 삽입/삭제가 원활한 방법을 찾아야 합니다.
- 문제를 살짝 바꿔서, 각각의 수를 구간으로 대응시키고 구간을 점으로 바꾸어 봅시다. 즉, 어떤 수 x를 구간 [x, x+m]로 바꾸고, 수들을 포함하는 구간 [a, a+m]을 점 a로 바꾸면 문제가 다음과 같이 바뀝니다
- N-k+1개의 길이 m짜리 구간들이 가장 많이 겹쳐져 있는 점에는 구간이 몇 개 있는가?
- 구간에 수 더하기 / 구간의 최댓값 구하기 연산을 지원하는 자료구조가 있으면 문제를 해결할 수 있습니다.

4번 - 범위 안의 숫자

- Lazy Propagation Segment Tree를 짜면 앞에서 말한 문제를 해결할 수 있습니다.
- 자세한 구현은 코드를 참고하도록 합시다.
- 수의 범위가 크기 때문에 실제로 고려해야 되는 O(Nk)개의 수들만 적절히 남기도록 좌표압축 등의 처리를 해 줘야 합니다.

4번 - 범위 안의 숫자

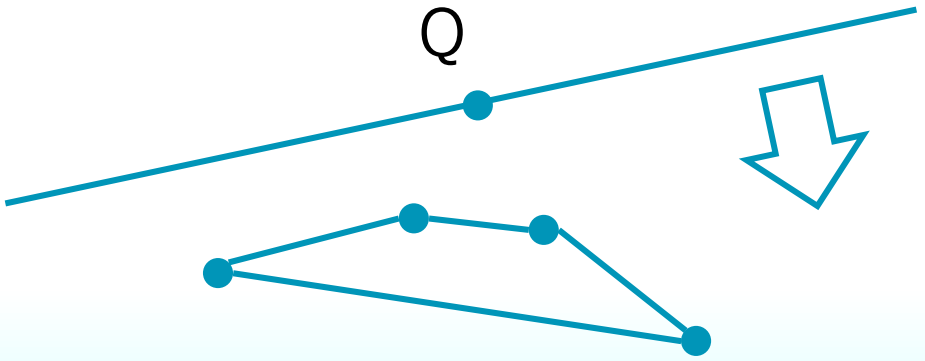
- 코드가 너무 길어서 따로 첨부하지는 않습니다.
  - GitHub에 전체 코드가 있습니다.
- Lazy Propagation Segment Tree에 대해서는 다른 자료를 통해 공부하면 좋을 것 같습니다.

5번 – 우범 지역

- 2차원 평면 상에 N개의 서로 다른 점 P_1, …, P_N이 있다. 각 점에서는 일정 확률로 범죄가 일어난다.
- 또 다른 점 Q가 주어지는데, 임의의 i, j에 대해 Q, P_i, P_j가 일직선상에 있지 않다.
- 범죄가 일어난 점들로 Convex Hull을 그렸을 때 Q가 그 안에 포함될 확률을 구하여라.
- $1 \leq N \leq 100,000$

5번 - 우범 지역

- 우선, Q를 원점에 가져다 놓아 봅시다.
- 이제 나머지 점들은 Q와의 '각도'만이 중요한 조건임을 알 수 있습니다.
- 구체적으로, P_i들 중 몇 개를 선택하여 Convex Hull을 만들었을 때 그것이 Q를 포함하지 않는다는 것은 선택된 점들이 Q를 지나는 어떤 직선의 한 쪽에만 놓인다는 것과 동치입니다.

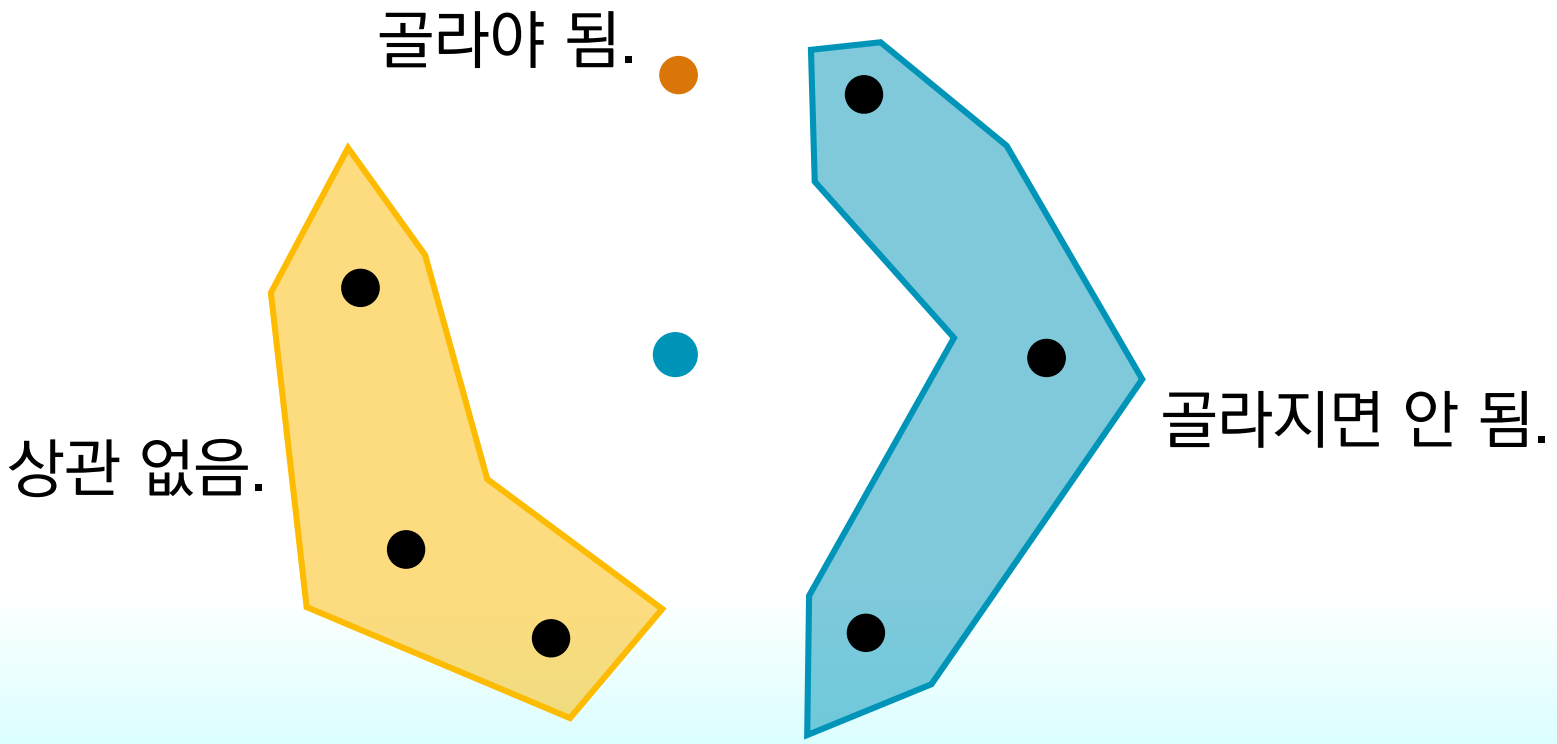


5번 - 우범 지역

- Q가 Convex Hull에 포함되지 않을 확률을 셀 수 있을 듯 합니다.
- 우선, 아무 점에서도 범죄가 일어나지 않을 확률을 더해줍니다.
- 그렇지 않다면, 어떤 점에서는 범죄가 일어났습니다.
- Q를 포함하지 않도록 선택된 임의의 점 부분집합에 대해, 가장 시계방향에 있는 점을 유일하게 찾을 수 있습니다. (180° 범위 내에 있으므로)
- 그 점을 고정시켜 놓고 생각하면, 각 점에 대해 "그 점이 가장 시계방향인 점이면서 Q를 포함하지 않도록 점들이 선택될 확률"을 다 구해주면 이들끼리는 겹치는 경우가 없게 됩니다!

5번 - 우범 지역

- 점 하나를 골랐을 때 그 점이 가장 시계방향인 점이면서 Q를 포함하지 않도록 선택될 확률은 (그 점을 고를 확률) * (그 점에서 시계방향으로 180° 범위의 점들을 하나도 안 고를 확률) 이 됩니다.



5번 - 우범 지역

- 점들을 반시계방향으로 돌아가게 정렬했다고 생각하면, “가장 시계방향인 점”의 인덱스를 증가시키면 “고르면 안 되는 범위”의 끝점 인덱스가 감소할 일이 없습니다.
- 즉, Two pointer 기법을 사용해서 모든 점에 대해 고르면 안 되는 범위를 알 수 있습니다.
- 안 고를 확률을 모두 곱하는 것은 구간 곱을 빠르게 처리하는 자료구조(세그먼트 트리)를 이용하면 됩니다.
  - 누적 곱 배열을 사용할 경우 1보다 작은 수를 많이 곱하면 그 값이 0이 되어 버려서 문제가 생깁니다.
  - log를 씌워서 누적합 배열로 바꿔줄 수 있습니다. (0 처리를 잘 해야 합니다)

5번 - 우범 지역

- 입력 및 각도 순 정렬 코드입니다.
- 점들을 1,4사분면 / 2,3사분면으로 나누어 각각 반시계로 돌아가게 정렬합니다.
- 배열이 원형이므로 같은 배열을 2번 복사해서 선형으로 펴면 편합니다.

```
int n;
cin >> n;

vector<po> v(n);
for(po &p : v) cin >> p.x.x;
for(po &p : v) cin >> p.x.y;
for(po &p : v) cin >> p.y;

pll ori;
cin >> ori.x >> ori.y;
for(po &p : v) { p.x.x -= ori.x; p.x.y -= ori.y; }

auto ccw = [](const pll &p, const pll &q) {
  ll t = p.x * q.y - q.x * p.y;
  return (t > 0) - (t < 0);
};
const static pll O(0LL, 0LL);
sort(all(v), [&](const po &p, const po &q){
  if((p.x < O) == (q.x < O)) return ccw(p.x, q.x) > 0;
  return p.x < q.x;
});
for(int i = 0; i < n; i++) v.push_back(v[i]);
```

5번 - 우범 지역

- 답을 구하는 부분입니다.
  - Segment Tree의 구현 부분은 생략하였습니다.
- Two pointer는 매우 간단하게 구현할 수 있습니다.
  - 기준점의 인덱스를 한 칸 오른쪽으로 옮길 때마다, 그 점에서 “더 시계방향에 있는 점”들만 포함되도록 j를 같이 옮겨 주면 됩니다.

```
Seg::init(2 * n);
for(int i = 0; i < 2 * n; i++) Seg::u(i, 1 - v[i].y);

ld ans = Seg::g(0, n - 1);
for(int i = n, j = 1; i < 2 * n; i++) {
  while(ccw(v[i].x, v[j].x) > 0) j++;
  ans += v[i].y * Seg::g(j, i - 1);
}

cout << (1 - ans) << '\n';
```

SCPC 1회 2차예선 풀이

1번 – 등차 수열

- 등차수열의 일부분을 나타내는 수열 B1, B2, …, BM이 주어진다.
- 이 수열이 포함되어 있었을 가능성이 있는 모든 등차수열에 대해, 서로 다른 공차가 몇 가지인지 구하여라.
- $1 \le M \le 100,000$

1번 - 등차 수열

- 우선, 수들이 모두 다르거나 모두 같아야 함을 알 수 있습니다.
  - 그렇지 않다면, 등차수열의 일부가 절대 될 수 없습니다.
- 수들이 모두 같다면 답이 1임도 알 수 있습니다.
  - 가능한 공차는 0으로 1가지뿐입니다.
- 이제, 수들이 모두 다를 때는 어떻게 하면 될까요?

1번 - 등차 수열

- B_1, B_2, …, B_M은 오름차순으로 주어집니다.
- 공차를 d라고 해 봅시다.
- B_2-B_1은 d의 배수입니다.
- B_3-B_2는 d의 배수입니다.
- …
- d는 B_2-B_1, B_3-B_2, …, B_M-B_{M-1}의 공약수입니다!
- 가능한 d의 가짓수는 최대공약수의 약수의 개수가 됩니다.

1번 - 등차 수열

- gcd는 C++17부터 STL 표준에 들어왔습니다.

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

  vll v(n);
  for(ll &x : v) cin >> x;

  if(v[0] == v[n - 1]) {
    cout << "1\n";
    return;
  }

  ll g = 0;
  for(int i = 1; i < n; i++) {
    if(v[i] == v[i - 1]) { g = -1; break; }
    g = gcd(g, v[i] - v[i - 1]);
  }

  int ans = 0;
  for(ll i = 1; i * i <= g; i++) {
    if(g % i == 0) ans += 1 + (i * i != g);
  }
  cout << ans << '\n';
}
```

2번 - MT 게임

- A학과 학생 a명, B학과 학생 b명이 게임을 합니다.
- A학과 학생들이 먼저 순서대로 차례를 가진 후 B학과 학생들이 순서대로 차례를 가집니다. 이것이 계속 반복됩니다.
- 각 차례마다 학생은 바로 전 사람이 부른 자연수 바로 다음 수부터 1~K개의 자연수를 이어서 부릅니다. (처음 시작하는 경우 1부터)
- N을 부르는 학생이 속한 학과가 집니다.
- N, K, a, b가 여러 번 주어지면 누가 이기는지 출력하세요.
- $1 \le N \le 1{,}000{,}000$, $1 \le K \le 1{,}000$, $1 \le a,b \le 10{,}000$

2번 - MT 게임

- 학생이 여러 명 있지만, 사실 같은 학과 학생들을 묶을 수 있습니다.
- 편의상 게임을 “N개의 돌이 있고, 각 학생은 1~K개의 돌을 가져갈 수 있다. 마지막 돌을 가져가는 팀이 진다” 라는 식으로 약간 변형합시다.
- A학과 팀은 매 차례마다 a개~aK개의 돌을 가져갈 수 있습니다.
- B학과 팀은 매 차례마다 b개~bK개의 돌을 가져갈 수 있습니다.
- “현재 차례에 돌이 i개 남아있을 때, 이기는가?” 라는 DP를 각 팀에 대해 정의하면 문제를 풀 수 있습니다.

2번 - MT 게임

- N이 100만 정도로 크기 때문에, DP 하나의 값을 빠르게 계산할 수 있어야 합니다.
- A팀이 돌이 i개 남아있을 때 이기기 위해서는 [i-aK, i-a] 구간에서 돌이 x만큼 남았을 때 B팀이 지는 x가 하나라도 존재하면 됩니다. B팀의 경우에도 유사하게 구간을 하나 잡을 수 있습니다.
- DP 누적합 배열을 만들면 그런 x가 존재하는지 여부를 O(1)에 판별할 수 있습니다.

2번 - MT 게임

- 돌이 적게 남았을 때 처리에 유의 해주면 됩니다.

```
void solve() {
  int a, b, c;
  cin >> a >> b >> c;

  const static int N = int(1e6) + 5;
  vector<int> d(N), e(N);
  for(int t = 0; t < c; t++) {
    int n, k;
    cin >> n >> k;

    for(int i = 1; i <= n; i++) {
      int nd =
        (i <= a) ? 0 :
        (i <= a * k) ? 1 :
        (e[i - a] - e[i - a * k - 1] < a * (k - 1) + 1);
      int ne =
        (i <= b) ? 0 :
        (i <= b * k) ? 1 :
        (d[i - b] - d[i - b * k - 1] < b * (k - 1) + 1);
      d[i] = d[i - 1] + nd;
      e[i] = e[i - 1] + ne;
    }

    cout << "ba"[d[n] - d[n - 1]];
  }
  cout << '\n';
}
```

3번 – 마라톤

- (0, 0)에서 (N, M)으로 격자 선을 따라서, x좌표 또는 y좌표가 증가하는 방향으로만 이동하려 합니다.
- 어떤 격자점에는 생수통이 있습니다. 생수통이 있는 격자점을 K번 이상 지나쳐야 합니다.
- 각 격자점에는 고도 값이 있습니다. 두 격자점 사이를 이동할 때는 두 격자점 사이의 고도 차이만큼 힘이 듭니다.
- 생수통을 K번 이상 지나는 경로 중 가장 힘이 적게 드는 경로가 힘이 얼마나 드는지 구하세요.
- $1 \leq N,M \leq 100$, $1 \leq K \leq 10$

3번 - 마라톤

- x좌표 또는 y좌표가 증가하는 방향으로 움직이니까 딱 봐도 [x좌표][y좌표]를 가지고 DP를 하면 될 것 같습니다.
- 생수통을 몇 개 만났는지도 중요하므로, 그것까지 인자로 넣어주면 될 것 같습니다.
- 이제 그렇게 DP를 하면 됩니다. (??)
- 이게 다라서 쓸 말이 없네요..

3번 - 마라톤

- 생수통을 K개보다 많이 만났으면 그냥 K개 만난 걸로 해 줘도 똑같습니다.
  - 근데 이 처리를 안 해도 맞을 듯 합니다..
- DP 항이 길 경우에는 upd 함수 같은 것을 정의해 주면 편합니다.

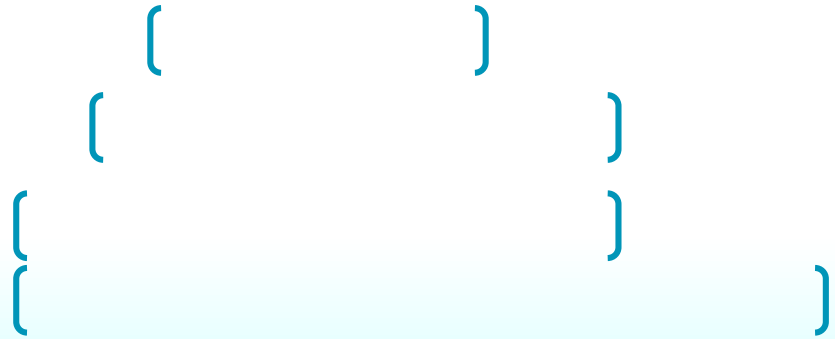
```
const int N = 105, K = 15;
int a[N][N], b[N][N], d[N][N][K];

void solve() {
  int n, m, k;
  cin >> m >> n >> k;
  for(int i = 0; i <= n; i++) for(int j = 0; j <= m; j++) {
    cin >> a[i][j];
    b[i][j] = (a[i][j] < 0);
    a[i][j] = abs(a[i][j]);
  }

  auto upd = [&](int &x, int v) { x = min(x, v); };
  memset(d, 0x3f, sizeof(d));
  d[0][0][0] = 0;
  for(int i = 0; i <= n; i++) {
    for(int j = 0; j <= m; j++) {
      for(int t = 0; t <= k; t++) {
        if(i < n) upd(d[i + 1][j][min(k, t + b[i + 1][j])],
                      d[i][j][t] + abs(a[i][j] - a[i + 1][j]));
        if(j < m) upd(d[i][j + 1][min(k, t + b[i][j + 1])],
                      d[i][j][t] + abs(a[i][j] - a[i][j + 1]));
      }
    }
  }
  cout << d[n][m][k] << '\n';
}
```

4번 – 최대 구간 중첩

- 구간이 N개 주어진다.
- 두 폐구간 [a, b]와 [c, d]에 대해, a ≤ c ≤ d ≤ b를 만족하거나 c ≤ a ≤ b ≤ d를 만족하면 둘은 중첩되었다고 합니다.
- 임의의 두 원소가 중첩되어 있도록 구간을 최대한 많이 골랐을 때 몇 개를 고를 수 있는지 출력하시오.
- 1 ≤ N ≤ 100,000

4번 - 최대 구간 중첩

- 임의의 두 원소가 중첩되어 있는 구간들의 집합을 생각해 봅시다.
- 그 집합에 속한 구간들을 시작점의 내림차순으로 정렬합시다.
  - 시작점이 같은 구간은 끝점 오름차순으로 정렬합시다.
- 이 때, 구간들의 끝점은 단조증가합니다. (감소하지 않습니다).
- 즉, 전체 구간들을 시작점 내림차순으로 정렬하고 나면, 거기서 끝점이 단조증가하는 구간들을 최대한 많이 고르는 문제가 됩니다!



4번 - 최대 구간 중첩

- 이 문제를 다른 말로 하면 최장 증가 부분 수열, 즉 LIS (Longest Increasing Subsequence) 의 길이를 구하는 문제가 됩니다.
  - 보통 이 문제는 강증가(strictly increasing) 수열을 고려하지만, 같은 값을 허용하더라도 크게 다르지 않습니다.
- 이 문제는 매우 유명한 문제 중 하나입니다.
- 코드가 매우 짧은 O(nlogn) 알고리즘이 잘 알려져 있습니다.
- 원리를 간단하게 소개하겠습니다.

4번 - 최대 구간 중첩

- D[i]를 “지금까지 만들 수 있는 길이가 i인 단조증가 부분수열들 중 가장 작은 맨 마지막 원소” 라고 합시다.
- 초기에는 D[0] = −inf, D[i] = inf (i ≥ 1) 입니다.
- N개의 수를 보면서 이 배열을 업데이트할 것입니다.
- 어떤 수 a가 맨 뒤에 하나 추가된다고 했을 때, D[x]가 바뀌는 x가 어디일지 생각해 보면 D[x−1] 〈= a이고 D[x] 〉 a를 만족해야 합니다.
- 그런 x는 많아야 1개 존재합니다!
- 그 위치는 이분탐색으로 간단하게 찾을 수 있습니다.

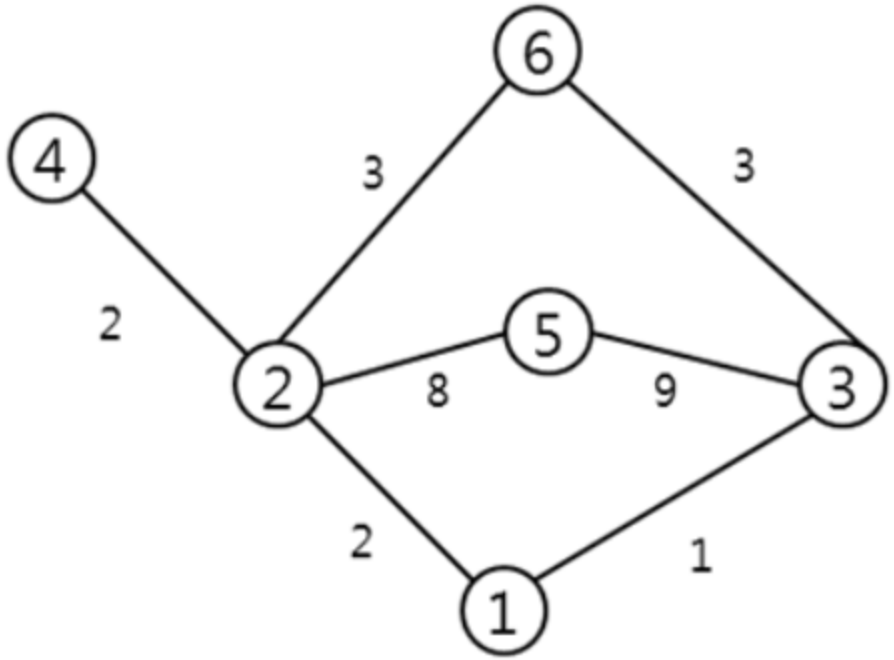
4번 - 최대 구간 중첩

- 코드가 매우 짧습니다.
- 어째 1차 예선보다 더 쉬운 거 같습니다…

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

  vpii v(n);
  for(auto &p : v) {
    cin >> p.x >> p.y;
    p.x *= -1;
  }
  sort(all(v));

  vint d(n + 1, int(1e9));
  d[0] = -int(1e9);
  for(auto &p : v) {
    *upper_bound(all(d), p.y) = p.y;
  }

  cout << int(lower_bound(all(d), int(1e9)) - d.begin() - 1) << '\n';
}
```

5번 - 캠퍼스와 도로(2)

- 정점 N개와 양방향 가중치 간선 M개가 있는 그래프가 주어짐.
  - 1 ≤ N ≤ 500, 1 ≤ M ≤ 5,000
- 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 해당 정점을 막았을 때 최단경로로 절대 갈 수 없는 정점이 생기는 정점을 모두 구하기



5번 - 캠퍼스와 도로(2)

- 1회 1차예선 5번과 아주 약간 다릅니다.
- 풀이도 아주 약간 다릅니다.
- 풀이의 앞부분을 다시 되새겨봅시다.

5번 – 캠퍼스와 도로(2)

- 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

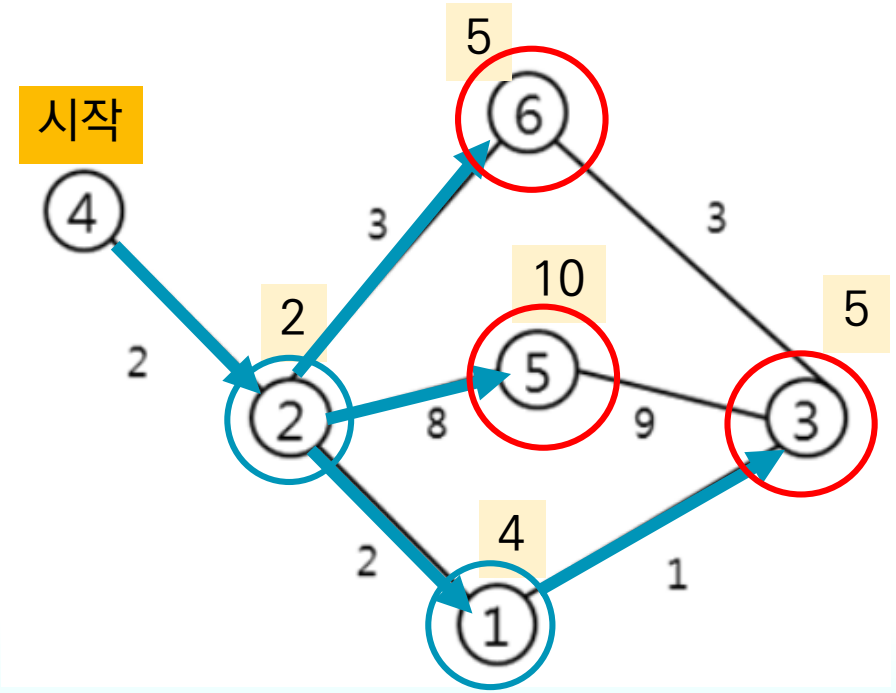
d[u] + c = d[v]

d[u] = d[v] + c

그 외

5번 - 캠퍼스와 도로(2)

- 어떤 정점을 막았을 때 절대 최단경로로 갈 수 없는 정점이 생긴다는 것은, 그 정점에서 indegree가 1인 정점으로 가는 간선이 있다는 것입니다.



4번 정점이 출발 정점일 때,
1번 정점을 막으면 3번 정점으로 못 가고,
2번 정점을 막으면 5,6번 정점으로 못 감

5번 - 캠퍼스와 도로(2)

- 캠퍼스와 도로 (1) 코드에서 이 부분만 바뀌고 나머지는 다 똑같습니다.

```
fill(all(ind), 0);
for(int i = 1; i <= n; i++) {
  for(pii j : e[i]) if(d[j.x] == d[i] + j.y) ind[j.x]++;
}
for(int i = 1; i <= n; i++) {
  if(i == st) continue;
  for(pii j : e[i]) {
    if(d[j.x] == d[i] + j.y && ind[j.x] == 1) {
      chk[i] = 1;
      break;
    }
  }
}
```