SCPC 4회 2차예선 풀이 + SCPC 5회 2차예선 풀이

서울대학교 컴퓨터공학부 18학번 김동현

SCPC 4회 2차예선 풀이

1번 - Quick Sort

- 길이 N의 배열 A[i]가 주어진다.
- Quick Sort란, 어떤 Pivot A[p]를 잡아서 i < p일 경우 A[i] < A[p], i > p 일 경우 A[i] > A[p]를 항상 만족하도록 만드는 것을 말한다.
- 주어진 배열에서 이미 Pivot의 조건을 만족하는 위치가 몇 개인지 출력하여라.
- $1 \leq N \leq 200,000$

1번 - Quick Sort

- 각 위치 x에 대해 만족해야 하는 조건은 아래 두 가지입니다.
  - [1, x−1] 구간에 있는 모든 i에 대해 a[i] 〈 a[x]
  - [x+1, n] 구간에 있는 모든 i에 대해 a[i] 〉 a[x]
- p[i] = max(a[1], a[2], …, a[i]), s[i] = min(a[i], a[i+1], …, a[n])이라고 정의하면 아래와 같이 바꿀 수 있습니다.
  - (x 〉 1일 경우) p[x−1] 〈 a[x]
  - (x 〈 n일 경우) s[x+1] 〉 a[x]
- p와 s 배열은 반복문 한 번으로 간단하게 계산할 수 있습니다.
- O(N)만에 문제를 해결할 수 있습니다.

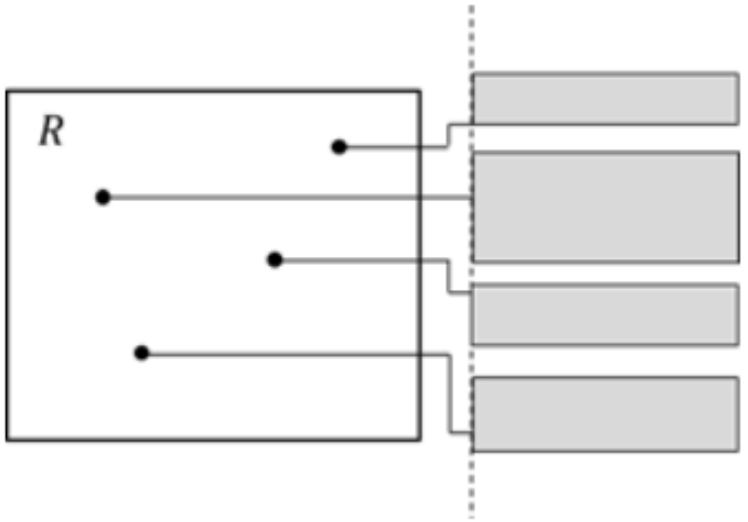
1번 - Quick Sort

- 인덱스를 좌우로 여유롭게 사용하면 끝 점 체크를 안 해도 됩니다.

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

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

  vint p(n + 1), s(n + 2);
  for(int i = 1; i <= n; i++) p[i] = max(p[i - 1], a[i]);
  s[n + 1] = int(2e9);
  for(int i = n; i >= 1; i--) s[i] = min(s[i + 1], a[i]);

  int r = 0;
  for(int i = 1; i <= n; i++) r += (p[i - 1] < a[i] && a[i] < s[i + 1]);
  cout << r << '\n';
}
```

2번 - 메모지

- 평면 상에 메모지가 하나씩 필요한 점 N개가 주어진다. 각 점의 y좌표는 모두 다르다.
  - $1 \le N \le 10,000$
- 각 점에 배당된 메모지마다 높이가 주어진다.
- 메모지는 모든 N개의 점 오른쪽에, 각각에 대응된 점의 y좌표 순서대로 놓이게 된다.
- 각 점은 배당된 메모지와 수평선으로 이어지거나, 수평/수직 꺾은선으로 이어져 있어야 한다.
- 메모지의 위치를 적절히 배정하여 꺾은선을 최소로 사용했을 때, 그 개수를 출력하라.



2번 - 메모지

- 뭔가 DP가 하고 싶어집니다.
- 그런데 점화식이 약간 까다롭습니다.
- 지금까지 몇 개 썼는지, 꺾은선이 지금까지 몇 개인지만 가지고는 약간 부족합니다.
- 그리디 맛을 약간 섞어서, DP 점화식을 아래와 같이 한 번 정의해 봅시다.
- D[i][j] : 아래에서 1~i번째 메모지까지 배치했고, 그 중 수평선이 j개 이상일 때 가능한 i번째 메모지의 최소 윗변 높이
  - 동일한 상황일 때는 i번째 메모지 높이가 낮을수록 좋기 때문입니다.
  - j개 '이상'으로 정의하면 식이 약간 간단해집니다.

2번 - 메모지

- D[i][j]의 점화식을 생각해 봅시다.
- 두 가지 경우가 있습니다. 둘 중 최솟값을 취하면 됩니다.
- (1) 꺾은선을 쓰는 경우
  - D[i-1][j] 바로 위에 메모지를 딱 붙이는 경우가 무조건 최적입니다.
  - 이 과정에서 꺾은선이 안 생길 수도 있기 때문에 j개 “이상”이라고 정의해야 올바른 정의입니다.
- (2) 수평선을 쓰는 경우
  - i번째 점의 y좌표를 메모지가 지나야 합니다. 즉, D[i-1][j-1] ≤ (i번째 점 y좌표)를 만족할 때만 가능합니다.

2번 - 메모지

- 여기서 문제가 한 가지 있는데, DP 배열의 크기가 너무 큽니다.
  - 4byte 정수를 10000^2개 잡으면 400MB입니다.
- 이를 해결하는 기법으로 “토글링”이라는 것이 있습니다.
- D[i][j]를 계산할 때 참조하는 값이 D[i-1][*] 뿐이므로, i에 해당하는 축의 길이를 2로 줄여도 상관이 없습니다!
- 보통 D[i][j]를 참조할 때 실제 배열에서는 arr[i%2][j]를 참조하는 식으로 구현합니다.
- 토글링을 할 경우에는 쓰던 배열에다 계속 값을 다시 쓰기 때문에 초기화에 조금 더 신경을 써야 합니다.

2번 - 메모지

- 참고로, https://www.acmicpc.net/problem/14752 이 문제랑 완전히 동일한 문제입니다. (17 ICPC 한국 인터넷예선 G번)
- 2번치고는 꽤 어려운 문제라서, 이걸 예전에 본 적이 있는지 여부가 영향을 끼쳤을지도 모르겠네요….
- 이 외에도 SCPC에는 “어디서 본 거 같은” 문제가 꽤 자주 나오는 편이니 평소에 백준 열심히 하시면 좋습니다(?)

2번 - 메모지

- 토글링을 편하게 구현하기 위해서 vector의 경우 & (참조형) 변수, 다차원 배열의 경우 포인터 변수를 적절히 활용하면 좋습니다.
  - cd : 지금 갱신하는 dp배열 (D[i])
  - pd : 참조할 dp배열 (D[i-1])

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

  vpii a(n);
  for(pii &p : a) cin >> p.x >> p.x >> p.y;
  sort(all(a));
  a.insert(a.begin(), pii(0, 0));

  vector<vint> d(2, vint(n + 2));
  const static int INF = int(1.05e9);
  d[0][0] = -INF;
  d[0][1] = INF;
  for(int i = 1; i <= n; i++) {
    vint &cd = d[i & 1], &pd = d[~i & 1];
    cd[0] = -INF;
    for(int j = 1; j <= i; j++) {
      cd[j] = min(
        pd[j] + a[i].y,
        (pd[j - 1] <= a[i].x ? max(a[i].x, pd[j - 1] + a[i].y) : INF)
      );
    }
    cd[i + 1] = INF;
  }

  int r = 0;
  for(r = 0; d[n & 1][r + 1] < INF; r++);
  cout << n - r << '\n';
}
```

3번 – Bitonic Paths

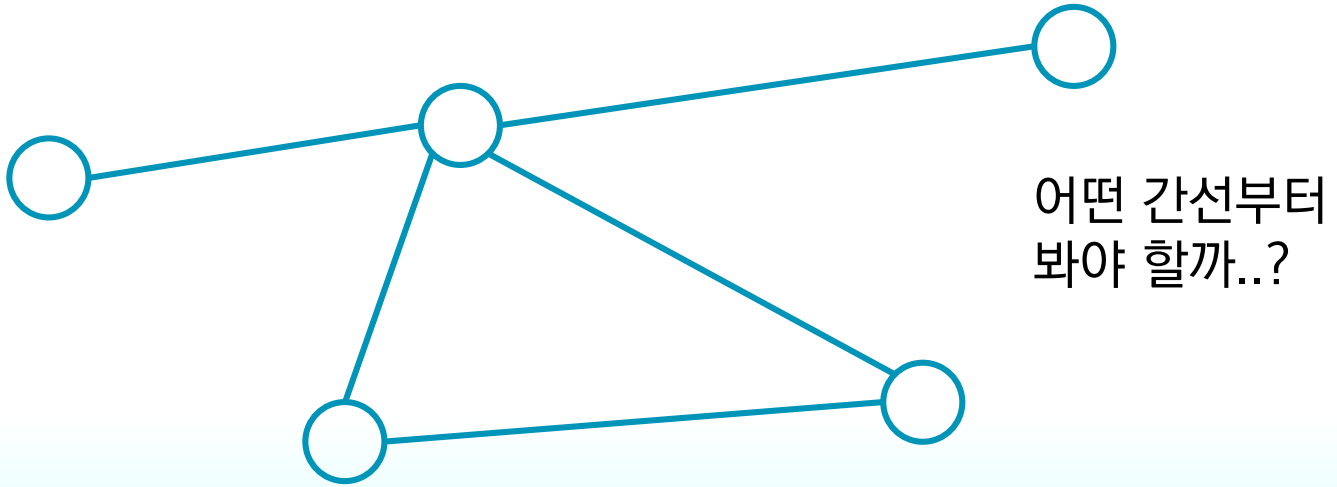
- 정점 N개, 간선 M개로 이루어진 양방향 가중치 그래프가 주어집니다.
- s에서 t로 가는 바이토닉 경로란, 경로 상의 어떤 정점 v가 존재해서 s에서 v로 가는 동안에는 간선 가중치가 단조증가하고, v에서 t로 가는 동안에는 단조감소하는 경로를 말합니다. (v는 s 또는 t일 수 있습니다)
- 1번에서 N번 정점으로 가는 바이토닉 경로 중 가장 짧은 것의 길이를 출력하세요.
- $1 \leq N,M \leq 100,000$
- $1 \leq$ (각 간선 가중치) $\leq 200,000$

3번 - Bitonic Paths

- 임의의 바이토닉 경로는 1번, N번 정점에서 각각 출발하는 단조증가 경로로 쪼갤 수 있습니다.
- 모든 정점 x에 대해 1번, N번 정점에서 각각 x번 정점까지 단조증가 경로로 도달하는 최소 거리를 합해 보아서 그 중 최솟값이 답이 됩니다.
- 시작점을 하나 정했을 때, 다른 모든 정점까지의 최단거리 단조증가 경로는 어떻게 구할까요?

3번 - Bitonic Path

- 간선을 가중치 순서대로 정렬해놓고 하나씩 보면 뭔가 될 것 같습니다.
- 모든 간선의 가중치가 다 다를 경우에는 잘 작동하는 방법입니다.
- 가중치가 같은 간선이 여러 개 있을 경우에는 아래 그림처럼 갱신 순서가 애매한 경우가 발생합니다.



3번 - Bitonic Path

- 결국 구하는 것이 “최단거리”이므로, 그냥 Dijkstra 알고리즘을 간선 가중치마다 한번씩 돌려주면 어떨까요?
- f(E) = ElogE가 아래로 볼록하기 때문에, 가중치마다 한번씩 그래프를 구축하고 Dijkstra 알고리즘을 수행해도 O(ElogE)임을 알 수 있습니다.
- 정확히 (해당 가중치의 간선 개수)에 비례하도록 코드를 주의깊게 구현해야 합니다.

3번 - Bitonic Path

- st번 정점에서 시작하는 단조증가 최단경로를 구하는 부분입니다.
- 초기 세팅 단계에서 priority_queue에 한 번 넣은 정점을 또 넣지 않도록 유의해야 합니다.

```
auto f = [&](vll &d, int st) {
  fill(all(d), INF);
  d[st] = 0;
  vector<vint> e(n + 1);
  vint c(n + 1);
  priority_queue<pli, vector<pli>, greater<pli>> pq;
  for(int t = 1; t < K; t++) {
    for(pii &p : edges[t]) {
      e[p.x].push_back(p.y);
      e[p.y].push_back(p.x);
      if(!c[p.x]) { c[p.x] = 1; pq.emplace(d[p.x], p.x); }
      if(!c[p.y]) { c[p.y] = 1; pq.emplace(d[p.y], p.y); }
    }

    while(!pq.empty()) {
      pli c = pq.top();
      pq.pop();
      if(c.x != d[c.y]) continue;
      for(int i : e[c.y]) {
        if(d[i] > c.x + t) {
          d[i] = c.x + t;
          pq.emplace(d[i], i);
        }
      }
    }

    for(pii &p : edges[t]) {
      if(c[p.x]) { c[p.x] = 0; e[p.x].clear(); }
      if(c[p.y]) { c[p.y] = 0; e[p.y].clear(); }
    }
  }
};
```

4번 - 지진

- N일 간의 지진파의 세기가 주어진다. ($1 \leq N \leq 10,000$)
- 연속된 M일에 해당하는 패턴이 하나 주어진다. 패턴을 이루는 값들은 모두 다르다. ($1 \leq M \leq 300$)
- N일 중 연속한 M일을 하나 잡아서 각 날짜에 패턴 값을 순서대로 대응시켰다고 하자. 이 중 K개 이하의 (데이터, 패턴) 쌍을 지워서 남은 데이터들의 크기 순서가 패턴과 정확히 일치하도록 할 수 있다면, 지진이 발생할 가능성이 있는 기간이다.
- 지진이 발생할 가능성이 있는 연속한 M일을 잡는 방법의 가짓수를 구하여라.

4번 - 지진

- 문제를 참 복잡하게도 써 놓았습니다….
- 패턴의 각 수들은 크기 순서만 중요하고, 나타나는 수들이 모두 다르므로 순열로 적절히 바꿔 줄 수 있습니다.
- 연속한 M일을 하나 잡았을 때 O(M) 내지는 O(MlogM) 시간 정도에 지진 발생 가능성을 판별할 수 있다면 문제를 풀 수 있습니다.
- K일 이하를 지워서, 즉 M-K일 이상을 남겨서 패턴과 데이터의 순서 관계가 정확히 일치하도록 했다는 것은 무슨 뜻일까요?

4번 - 지진

- M일간의 데이터를 “패턴 순열 순서대로” 읽었다고 해 봅시다.
  - 예를 들어 패턴이 “3 1 4 2”라면 2번째 -> 4번째 -> 1번째 -> 3번째 순서대로 읽는 것입니다.
- 남은 데이터들이 패턴과 순서 관계가 일치한다는 것은 남은 데이터들이 패턴 순열 순서대로 읽었을 때 증가수열이라는 것과 같은 말입니다.
- 즉, 패턴 순열 순서대로 보면서 LIS(최장 증가 부분수열)의 길이를 구하고, 그것이 M-K 이상인지 아닌지 보면 됩니다!
- LIS는 O(MlogM)에 구할 수 있습니다.

4번 - 지진

- 4번 치고 코드가 매우 짧습니다….
- 개인적으로 2번보다 쉬울 수도 있다고 생각합니다.

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

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

  vint c(m);
  iota(all(c), 0);
  sort(all(c), [&](int x, int y){ return b[x] < b[y]; });

  int ans = 0;
  vint d(m + 1);
  const static int INF = int(1e9);
  for(int i = 0; i + m <= n; i++) {
    fill(all(d), INF);
    d[0] = -INF;
    for (int j = 0; j < m; j++)
      *lower_bound(all(d), a[i + c[j]]) = a[i + c[j]];
    if(d[m - k] < INF) ans++;
  }

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

5번 – 히스토그램

- 계급이 N개 있는 히스토그램이 주어진다. 계급 i의 도수를 H(i)라고 하자.
- 이 히스토그램을 비감소 (H(i) ≤ H(i+1)을 만족하는) 히스토그램으로 바꿀 것인데, 아래와 같이 정의되는 오차를 최소화할 것이다.

$$\sum_{i=1}^{N}(H(i)-H'(i))^2$$

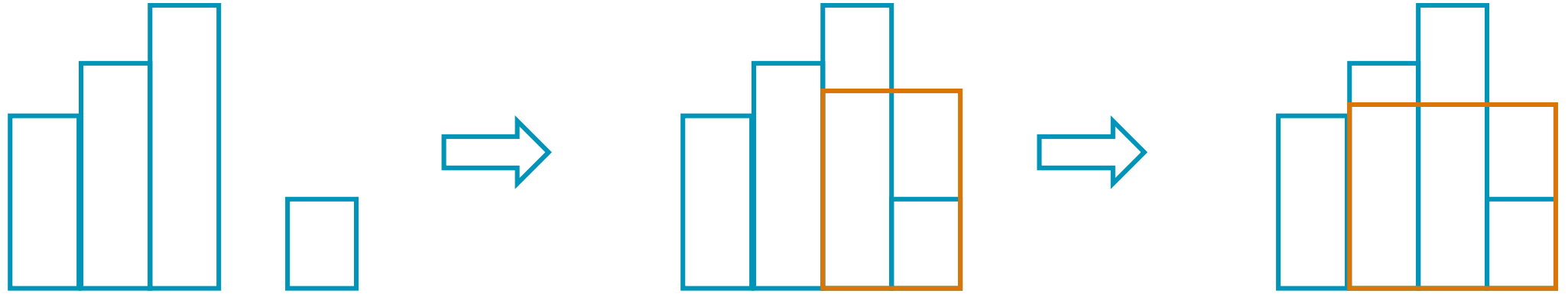
- 최소화한 오차를 출력하라.
- 1 ≤ N ≤ 1,000,000

5번 - 히스토그램

- 일단, 어떤 구간의 값을 다 같게 바꾼다고 했을 때 오차를 최소화하는 값은 구간의 평균임을 알 수 있습니다.
  - 식이 계수가 양수인 이차함수 꼴이라서, 축의 위치를 구해보면 됩니다.
  - 그리고 이 때 오차는 (구간의 분산) * (구간 길이) 가 됩니다.
- 이제, 다음과 같은 알고리즘을 생각해 볼 수 있습니다.
  - 히스토그램에 기둥을 1~N 순서대로 오른쪽으로 붙여 나갑니다.
  - 결과 히스토그램은 단조증가하는 구간 평균 여러 개로 이루어집니다.
  - 매번 기둥을 붙인 다음에 (뒤에서 2번째 구간 평균) ≤ (맨 마지막 구간 평균)을 만족할 때까지 맨 뒤 2개의 구간을 합쳐서 전체 구간 평균으로 값을 바꿉니다.
- 이제 이걸 짜서 내면 맞습니다.

5번 - 히스토그램

- 대회 당시에는 그냥 짜서 냈더니 맞길래 (^^;;) 왜 맞는지에 대해서는 자세히 생각을 안 해 봤습니다.
- 알고리즘에 대해 좀 더 자세히 설명하자면, 아래 그림과 같이 작동합니다.



H	3	4	5	2
H’	3	4	5	

H	3	4	5	2
H’	3	4	3.5	3.5

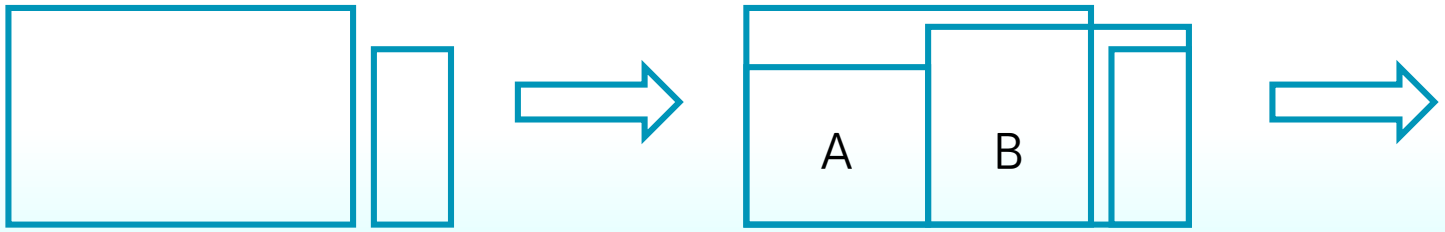
H	3	4	5	2
H’	3	11/3	11/3	11/3

5번 - 히스토그램

- 이 알고리즘이 왜 올바른지에 대해 잠시 살펴봅시다.
- 결국 전체 히스토그램을 구간 평균이 단조증가하는 여러 구간들로 partitio하는 것이 목표인데, 답을 최소화하려면 히스토그램을 최대한 잘게 나누는 것이 좋습니다.
- 앞서 제시한 알고리즘이 히스토그램을 가장 잘게 나눈다는 것을 귀납법을 통해 증명해 봅시다.

5번 - 히스토그램

- 어떤 시점까지 알고리즘이 구한 partition이 있고, 그 뒤에 기둥 하나를 더 붙이려고 합니다.
- 그 기둥이 마지막 구간 평균보다 높이가 같거나 높다면 그냥 그 기둥을 구간 하나로 해서 새로 뒤에 붙이면 끝입니다.
- 그게 아니라면, 원래 partition 중 뒤의 몇 개를 맨 뒤 기둥과 합치게 될 것인데, 이 때 원래 partition의 구간 중 하나를 쪼개서 일부만 합치게 되는 경우가 없습니다. (가장 잘게 나누었다는 가정에 모순)



5번 - 히스토그램

- 뒤에서부터 합쳐 나가다가 조건을 만족하게 되는 순간 바로 멈추므로, 그것보다 더 잘게 나눌 수는 없을 것입니다.
- 즉, 매번 기둥을 추가할 때마다 항상 가장 잘게 나눈 partition을 계속 유지할 수 있게 됩니다!
- 구간을 정했을 때 거기서 발생하는 오차는 앞서 말했듯이 분산을 구하는 것이므로, H(i)들의 합과 (H(i))^2들의 합을 가지고 식을 쓸 수 있습니다. 누적합을 이용하면 O(1)에 구할 수 있습니다.
- 인접한 구간끼리 평균을 비교하고 합치는 것은 스택을 이용하면 O(N)에 할 수 있습니다.

5번 - 히스토그램

- 실수 오차는 long double을 사용하면 보통 문제가 없는 듯 합니다.

```
int n;
cin >> n;

vll a(n + 1), s(n + 1), t(n + 1);
for(int i = 1; i <= n; i++) {
  cin >> a[i];
  s[i] = s[i - 1] + a[i];
  t[i] = t[i - 1] + a[i] * a[i];
}

auto c = [&](int l, int r) {
  return (t[r] - t[l - 1]) - ld(s[r] - s[l - 1]) * (s[r] - s[l - 1]) / (r - l + 1);
};

stack<int> st;
st.push(0);
ld ans = 0;
for(int i = 1; i <= n; i++) {
  while(st.top() > 0) {
    int t = st.top();
    st.pop();
    if((s[t] - s[st.top()]) * (i - t) < (s[i] - s[t]) * (t - st.top())) {
      st.push(t);
      break;
    }
    ans -= c(st.top() + 1, t);
  }
  ans += c(st.top() + 1, i);
  st.push(i);
}

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

SCPC 5회 2차예선 풀이

1번 - 소수 수열

- 각 자릿수가 1 이상인 자연수 N이 있을 때, 그 수를 써 놓고 숫자를 하나씩 지워 갈 것이다. 이 때, 지워가는 과정에서 나타나는 모든 수가 소수가 되도록 최대한 많이 지운 횟수를 f(N)이라고 하자. (N이 소수가 아니라면 0으로 정의한다)
- 두 자연수 A, B가 주어지면 f(A)와 f(B)의 대소관계를 비교하여라.
- $1 \leq A,B \leq 30,000$

1번 - 소수 수열

- 에라토스테네스의 체를 이용하면 1 이상 30,000 이하의 각 자연수에 대해 그 수가 소수인지 아닌지 바로 알 수 있습니다.
- f(N)은 재귀적으로 정의되므로, 정의에 입각하여 DP를 구현하면 매우 간단하게 구할 수 있습니다.
- 열심히 구현만 하면 됩니다.

```
const int N = 30005;
vint d(N);

void solve() {
  int x, y;
  cin >> x >> y;
  cout << (d[x] > d[y] ? 1 : d[x] < d[y] ? 2 : 3) << '\n';
}
```

쿼리 받고 출력하는 부분

```
vint p(N, 1);
p[1] = 0;
for(int i = 2; i < N; i++) {
  if(!p[i]) continue;
  for(int j = i * i; j < N; j += i) p[j] = 0;
}

for(int i = 2; i < N; i++){
  if(!p[i]) continue;
  for(int t = 1; t <= i; t *= 10) {
    int nxt = (i / (10 * t) * t) + (i % t);
    d[i] = max(d[i], d[nxt] + 1);
  }
}
```

DP 계산하는 부분

2번 – 유사도

- 길이 N의 두 수열 A와 B가 주어진다.
- 수열의 유사도란, A[i] = B[i]인 i의 개수를 의미한다.
- B에서 구간 하나를 정해 뒤집을 수 있다.
- 뒤집을 구간을 잘 정해서 유사도를 최대화하라.
- $1 \leq N \leq 5{,}000$

2번 - 유사도

- 원래 수열의 유사도는 O(N)에 간단하게 구할 수 있습니다.
- 뒤집을 수 있는 구간은 O(N^2)개 존재하니 이 때마다 유사도를 매번 다시 구하면 O(N^3)이 됩니다.
- 빠르게 구하기 위해서는, [s, e] 구간을 뒤집을 때와 [s+1, e−1] 구간을 뒤집을 때의 관계를 생각해 보면 됩니다.
- [s+1, e−1] 구간을 뒤집었을 때 유사도를 알고 있다면, [s, e] 구간을 뒤집을 때는 B에서 변한 곳이 B[s]와 B[e]가 바뀌었다 말고 없기 때문에 유사도의 변화를 O(1)에 구할 수 있습니다!
- 구간 길이가 짝수일 때와 홀수일 때를 나누어야 함에 주의합시다.

2번 - 유사도

- 개인적으로 for문 안에 변수를 여러 개 선언하는 것을 선호합니다.

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

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

  int bas = 0;
  for(int i = 0; i < n; i++) if(a[i] == b[i]) bas++;

  int ans = 0;
  for(int i = 0; i < n; i++) {
    int cur = 0;
    for(int s = i - 1, e = i + 1; 0 <= s && e < n; s--, e++) {
      cur += (a[s] == b[e]) + (a[e] == b[s]) - (a[s] == b[s]) - (a[e] == b[e]);
      ans = max(ans, cur);
    }
    cur = 0;
    for(int s = i, e = i + 1; 0 <= s && e < n; s--, e++) {
      cur += (a[s] == b[e]) + (a[e] == b[s]) - (a[s] == b[s]) - (a[e] == b[e]);
      ans = max(ans, cur);
    }
  }

  cout << (bas + ans) << '\n';
}
```

3번 - 드론 탐험

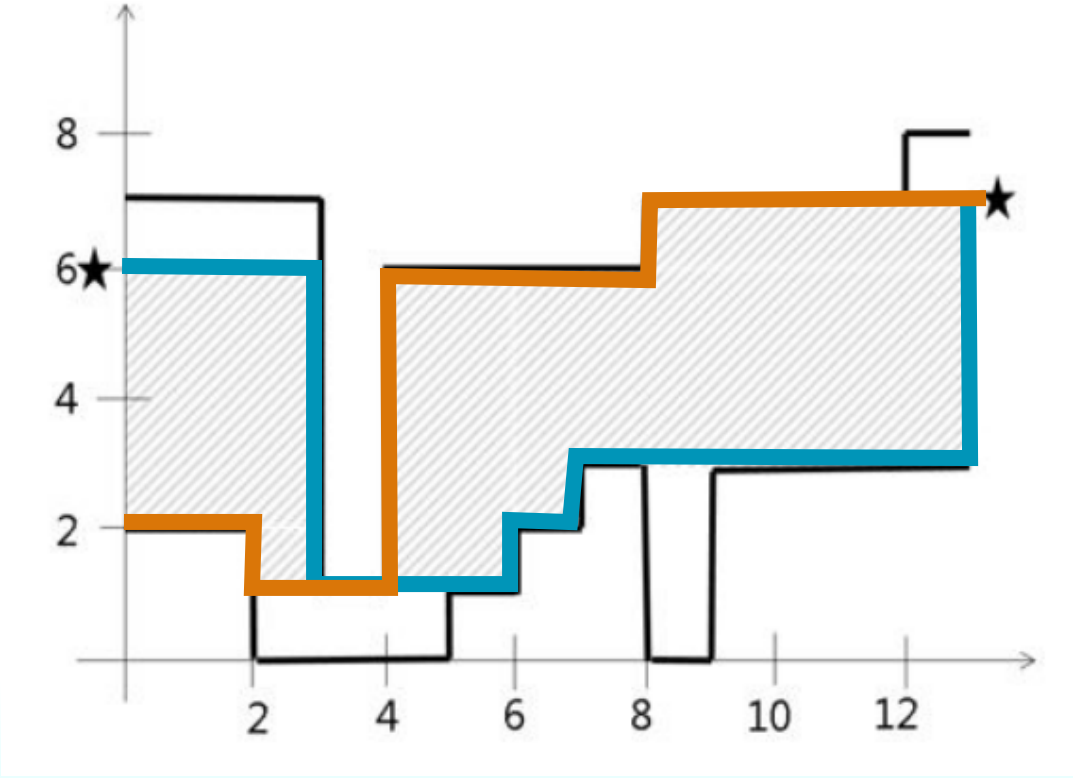
- 드론이 동굴의 (0, s)에서 출발하여 (L, e)에 도달하고자 합니다.
- 드론은 수직/수평으로만 움직일 수 있습니다.
- 동굴 지형은 천장과 바닥으로 나눌 수 있고, 각각 A개 / B개의 부분으로 나뉘어 각 부분마다 (길이, 높이)가 주어집니다.
- 드론이 최단 거리로 움직인다고 했을 때, 드론이 지나는 모든 경로는 어떤 영역의 형태로 나타납니다.
- 영역의 넓이를 구하세요.
- $1 \leq A,B \leq 100,000$

3번 - 드론 탐험

- 드론이 지나는 최단경로를 구하는 방법에 대해 일단 생각해 봅시다.
- (0, s)에서 출발해서 오른쪽으로 계속 간다고 생각해 봅시다.
- 이 때, 수직 이동은 미리 해서 이득을 볼 게 전혀 없음을 알 수 있습니다. 즉, 중간에 지형에 의해 막히는 시점마다 필요한 만큼만 수직 이동을 해 주면 됩니다.
- 일종의 Greedy 전략이라고 생각할 수 있습니다.
- (L, e)에서 출발해서 왼쪽으로 쭉 가면서 비슷한 이동을 해 줄 수 있습니다.

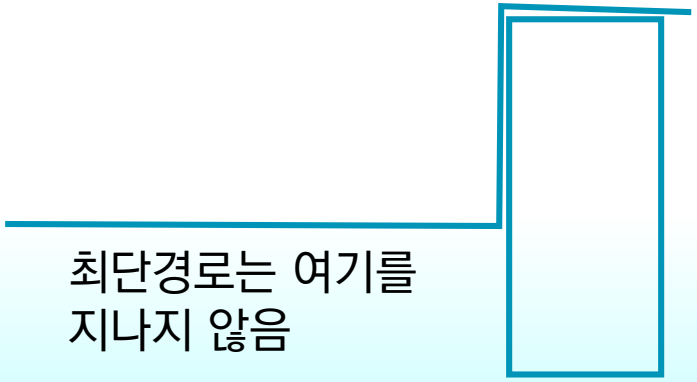
3번 - 드론 탐험

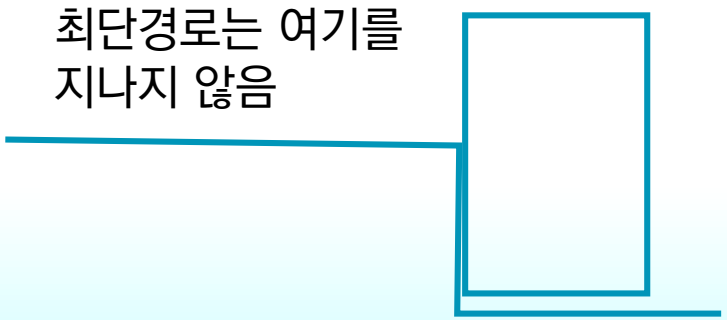
- 예제를 가지고 앞에서 말한 두 가지 이동을 해 보면 뭔가가 보입니다..?



3번 - 드론 탐험

- 왼쪽/오른쪽에서 각각 Greedy하게 수행한 이동이 영역의 경계를 이룸을 알 수 있습니다.
- 위/아래 관계는 중간에 바뀔 수 있지만, 두 이동 경로 사이에 끼인 경로만이 최단경로가 됩니다! 즉 두 경로 사이에 끼인 넓이를 구하면 됩니다.
- 이것이 성립하는 이유는 각 경로에서 수직 이동이 일어날 때마다 그보다 위(또는 아래)에 걸치는 경로가 최단경로가 절대 아님을 알 수 있기 때문입니다.





3번 - 드론 탐험

- 구현 시에 헷갈릴 수 있는 부분이 많은데, 한 가지 방법 중 하나는 동굴을 x좌표가 [a[i], a[i+1]]인 구간에서는 y좌표 [s[i], e[i]] 범위를 지날 수 있다 같은 식으로 표현하는 것입니다.
- 이렇게 하면 각 경로 역시 x좌표가 [a[i], a[i+1]]인 구간에서 높이가 h[i] 였다 와 같이 표현이 되므로 처리하기가 편해집니다.

3번 - 드론 탐험

- 앞에서 말한 것처럼 동굴 지형을 전처리하는 코드입니다.

```
int l, s, e;
cin >> l >> s >> e;

int a;
cin >> a;
vint ax(a + 1), ah(a), tx(1);
for(int i = 0; i < a; i++) {
  cin >> ax[i + 1] >> ah[i];
  ax[i + 1] += ax[i];
  tx.push_back(ax[i + 1]);
}
int b;
cin >> b;
vint bx(b + 1), bh(b);
for(int i = 0; i < b; i++) {
  cin >> bx[i + 1] >> bh[i];
  bx[i + 1] += bx[i];
  tx.push_back(bx[i + 1]);
}
sort(all(tx));
tx.erase(unique(all(tx)), tx.end());

int n = int(tx.size()) - 1;
vint lh(n), uh(n);
for(int i = 0, ap = 0, bp = 0; i < n; i++) {
  if(ap < a - 1 && ax[ap + 1] <= tx[i]) ap++;
  if(bp < b - 1 && bx[bp + 1] <= tx[i]) bp++;
  uh[i] = ah[ap];
  lh[i] = bh[bp];
}
```

3번 - 드론 탐험

- 경로 두 개를 구해서 사이에 낀 영역의 넓이를 계산하는 코드입니다.

```
vint lpath(n), rpath(n);
for(int i = 0, h = s; i < n; i++) {
  h = max(lh[i], min(uh[i], h));
  lpath[i] = h;
}
for(int i = n - 1, h = e; i >= 0; i--) {
  h = max(lh[i], min(uh[i], h));
  rpath[i] = h;
}

ll ans = 0;
for(int i = 0; i < n; i++) {
  ans += ll(tx[i + 1] - tx[i]) * abs(rpath[i] - lpath[i]);
}
cout << ans << '\n';
```

4번 - 폭격

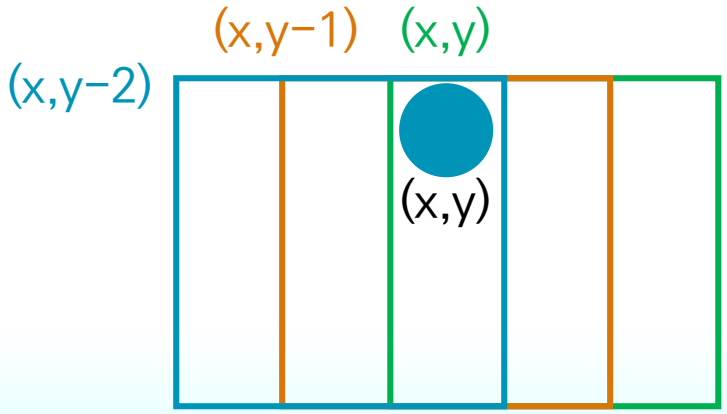
- M x N 격자가 있고, 몇몇 칸에는 공장이 있습니다.
- 한 번의 폭격으로 3 x 3 격자에 있는 공장을 모두 파괴할 수 있습니다.
- 최소 개수의 폭격으로 격자 전체의 모든 공장을 파괴하세요.
- $1 \leq M \leq 50$, $1 \leq N \leq 500$

4번 - 폭격

- 휴리스틱 문제입니다.
  - 올해 1차예선에는 없었는데, 2차예선에는 충분히 나올 수 있다고 봅니다.
- 다행히 이 문제도 “그럴듯한 그리디”를 잘 짜면 만점이 나오는 널널한(?) 편인 문제입니다.
- 그리디를 생각해 봅시다.

4번 - 폭격

- 아직 남아 있는 공장 중 가장 위쪽 행에서 가장 왼쪽에 있는 것을 하나 잡읍시다.
- 이 공장을 포함하는 폭격을 최소 1회는 해야 합니다.
- 해당 공장을 터트리는 폭격 중 고려할 것은 아래의 3가지 중 하나입니다.
- 가장 많은 공장을 터트릴 수 있는 폭격을 해 주는 것을 반복해 줍시다.



4번 – 폭격

- 폭격의 범위가 격자 밖으로 나가지 않도록 처리를 해 주어야 합니다.

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

vector<string> b(n + 2);
b[0] = b[n + 1] = string("0", m + 2);
for(int i = 1; i <= n; i++) {
  cin >> b[i];
  b[i] = "0" + b[i] + "0";
}

vpii ans;
auto cnt = [&](int x, int y) {
  if(x <= 0 || y <= 0) return -1;
  int r = 0;
  for(int i = 0; i < 3; i++) for(int j = 0; j < 3; j++)
    if(b[x + i][y + j] == '1') r++;
  return r;
};
auto bomb = [&](int x, int y) {
  ans.emplace_back(x, y);
  for(int i = 0; i < 3; i++) for(int j = 0; j < 3; j++)
    b[x + i][y + j] = '0';
};
```

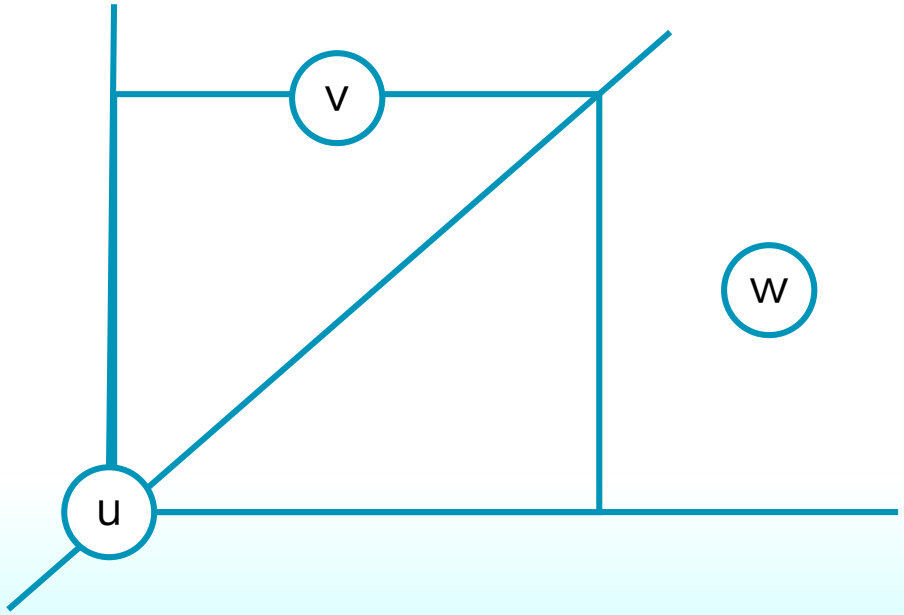
```
for(int i = 1; i <= n - 3; i++) {
  for(int j = 1; j <= m - 3; j++) {
    if(b[i][j] == '1') {
      int c[3] = {cnt(i, j - 2), cnt(i, j - 1), cnt(i, j)};
      int mx = max({c[0], c[1], c[2]});
      if(mx == c[0]) bomb(i, j - 2);
      else if(mx == c[1]) bomb(i, j - 1);
      else bomb(i, j);
    }
  }
}
for(int i = 1; i <= n - 3; i++) {
  if(b[i][m - 2] + b[i][m - 1] + b[i][m] > 3 * '0') bomb(i, m - 2);
}
for(int i = 1; i <= m - 3; i++) {
  if(b[n - 2][i] + b[n - 1][i] + b[n][i] > 3 * '0') bomb(n - 2, i);
}
if(cnt(n - 2, m - 2)) bomb(n - 2, m - 2);

cout << ans.size() << '\n';
for(pii &p : ans) cout << p.x << ' ' << p.y << '\n';
```

5번 - 존의 정사각형

- 한 변의 길이가 M인 축 평행 정사각형 내에 점이 N개 있다.
- 각 점에서 다음 조건을 만족하는 가장 큰 정사각형을 그릴 것이다.
  - 왼쪽 아래 꼭짓점이 해당 점이다.
  - 정사각형의 내부에 N개의 점 중 어떤 것도 포함되지 않는다.
  - 큰 정사각형 (길이 M짜리) 밖으로 나가지 않는다.
- 모든 점에 대해 그린 정사각형의 한 변의 길이들의 합을 출력하라.
- $1 \leq N \leq 500{,}000$

5번 - 존의 정사각형

- 각 점 (x[u], y[u])에 대해, 그려지는 정사각형의 한 변의 길이는 크게 두 가지 조건에 의해 결정됩니다.
  - x[v] > x[u], x[v] − y[v] ≤ x[u] − y[u]인 v에 대해 (y[v] − y[u])의 최솟값
  - y[w] > y[u], x[w] − y[w] ≥ x[u] − y[u]인 w에 대해 (x[w] − x[u])의 최솟값



5번 - 존의 정사각형

- 조건은 크게 (1) x (또는 y) 좌표 값 / (2) (x좌표 - y좌표) 값으로 표현됩니다.
- 우선, x - y 값을 기준으로 인덱스를 매겨주면 간단한 구간 쿼리로 (2)를 처리할 수 있을 것 같습니다.
- 이제, 점을 적절한 순서대로 고려하기만 하면 (1) 역시 처리할 수 있습니다.
  - (1) 조건이 x좌표일 경우에는 x좌표가 큰 점부터 보면 됩니다. x좌표가 같은 점은 y좌표가 작은 점부터 보면 됩니다.
  - (1) 조건이 y좌표일 경우에는 x와 y를 바꿔서 비슷하게 하면 됩니다.

5번 - 존의 직사각형

- x-y 조건은 점 갱신 / 구간 최솟값 구하기를 지원하는 세그먼트 트리를 구현하면 됩니다.
- 미리 각 점의 x-y 값들을 다 구해놓고 정렬해서 좌표압축을 해 두면 됩니다.
- 까놓고 보면 복잡한 테크닉이나 자료구조가 전혀 필요하지 않은 문제인데, 대회 때는 왠지 모르게 어렵게 풀었다가 만점을 못 받았던 기억이 있습니다..

5번 - 존의 직사각형

- 세그먼트 트리 구현은 Github에 가면 있습니다.

```
int l, n;
cin >> l >> n;

struct Pos { pii p; int i, ans; };
vector<Pos> a(n);
vint xs(n);
for(int i = 0; i < n; i++) {
  cin >> a[i].p.x >> a[i].p.y;
  xs[i] = a[i].p.x - a[i].p.y;
}
sort(all(xs));
for(int i = 0; i < n; i++) {
  a[i].i = int(lower_bound(all(xs), a[i].p.x - a[i].p.y) - xs.begin()) + 1;
}
```

```
sort(all(a), [&](Pos &a, Pos &b){
  return a.p.x == b.p.x ? a.p.y < b.p.y : a.p.x > b.p.x;
});
Seg::i(n, 1);
for(Pos &x : a) {
  x.ans = Seg::g(1, x.i) - x.p.y;
  Seg::u(x.i, x.p.y);
}

sort(all(a), [&](Pos &a, Pos &b){
  return a.p.y == b.p.y ? a.p.x < b.p.x : a.p.y > b.p.y;
});
Seg::i(n, 1);
for(Pos &x : a) {
  x.ans = min(x.ans, Seg::g(x.i, n) - x.p.x);
  Seg::u(x.i, x.p.x);
}

ll tot = 0;
for(Pos &x : a) tot += x.ans;
cout << tot << '\n';
```