나는코더다
선발대비 모의고사
Solution booklet

14004 구재현

Results (GSHS Aggregated)

이름	jewel	floodfill	frog	game	farming	heavylight	newtown	physics	total
김현수	100	43	100	100	25	0	0	0	368
유재민	0	43	87	59	40	0	36	41	306
신승원	0	43	87	0	0	100	21	41	292
오선재	0	43	87	19	0	43	0	41	233
박민솔	0	43	50	59	15	0	15	41	223
이상헌	23	43	100	0	0	0	36	0	202
최재현	0	43	50	0	30	0	36	0	159

Results (GSHS 10/28)

이름	jewel	floodfill	frog	game	total
김현수	100	43	100	100	343
유재민	0	43	87	59	189
김동현	23	43	50	59	175
이상헌	23	43	100	0	166
박민솔	0	43	50	59	152
오선재	0	43	87	19	149
신승원	0	43	87	0	130
윤혁중	0	0	13	100	113
백인규	0	29	13	59	101
최재현	0	43	50	0	93
최종민	23	0	0	19	42
박민수	23	0	13	0	36

Results (GSHS 11/19)

이름	farming	heavylight	newtown	physics	total
신승원	0	100	21	41	162
유재민	40	0	36	41	117
오선재	0	43	0	41	84
박민솔	15	0	15	41	71
최재현	30	0	36	0	66
이상헌	0	0	36	0	36
김현수	25	0	0	0	25

Best Unofficial Competitors

- ainta (1차) : 100 / 100 / 100 / 100
- august14 (1차) : 100 / 100 / 100 / 100
- q11 (2차) : 0 / 20 / 21 / 41
- JKJeong (1차) : 0 / 16 / 100 / 59. sutekine!!!!

Solutions for Day 1

1. 보석 광산 (jewel)

- 문제요약이 쉽지 않음..
- Category : Data Structures
- 출처 : http://wcipeg.com/problem/ccc15s2p6
- 100 23 23 23 23 0 0 0 0 0

Idea 1

- 상당히 복잡하게 꼬여있는 문제라 단순화해서 최적화하기 쉽지 않은 문제인데..
- 일단, 관점을 살짝 틀어서, C[i, j]라는 걸 정의하자.
- C[i, j] -> 1 <= i, j <= N - K + 1에 대해서, [i, i+K-1] & [j, j+K-1] 을 만족하는 부분 정사각형을 팠을 때 얻는 이익
- 저렇게 하면 K 걱정 없이 counting만 해주면 되니까!

Idea 2

- 보석 하나가 있을 때는, [Px - K + 1, Px] & [Py - K + 1, Py] 정사각형 영역에 1의 이익을 더한다. 변홧값 배열로 O(1)에 가능한 연산.
- 보석이 여러개 있을 때는, 저러한 정사각형 영역의 합집합에 1의 이익을 더하면 된다.
- 같은 보석은.. 많아야 4개다! 포함배제 + 변홧값 배열을 사용해서 보석이 주어졌을 때 C[i, j]를 채울 수가 있음.
- C[i, j]를 채우는데, O(2^S * M + N^2)의 시간이 소요됨.

Idea 3

- 이제 들어오는 쿼리는 C[i, j]의 정사각형 영역을 지우는 쿼리와. C[i, j]>= V인 격자의 개수를 세는 쿼리임.
- 일단, V <= M이라 봐도 무방함.
- 정사각형 영역을 변홧값 배열을 사용해서 지운 후, 쿼리 2 전에 전처리할 시 **O(N^2 + M)에 서브태스크 2까지 풀 수 있음.**
- 한편, C[i, j] = T인 개수를 가지고 있는 binary indexed tree를 만들어 줄 수 있음. 정사각형 영역을 잘 지워주는 구조와 동반할 시. 2번 쿼리는 O(lgM)에 해결 가능.

Idea 4

- 그래서 이제 정사각형 영역을 잘 지워주는 구조가 필요한데.. std::map!
- X좌표마다 key가 Y좌표이고 value가 C[i, j]인 map을 잡음.
- lower_bound 연산으로 들어가서, 해당 정사각형 안에 있을 때까지 다 날림 (BIT 연산과 동반)
- BIT를 같이 쓰니. N^2lgN + QNlgN. **AC.**
- 왜 쿼리당 NlgN이냐고요? **한번 지운 건 다시 안 지우니. 그 부분을 따로 생각하면, lower bound 연산 하나만 세주면 됩니다.**

Appendix. Better Ideas

- 사실 이런 류의 문제는 std::set 없이, union-find 류의 path compression으로도 잘 풀림. X좌표 마다 union find를 잡는다.
- 구간에 대해서 날려줘야 할때, 먼저 해당 컴포넌트에 있는 값을 본다음에. 다음 컴포넌트로 넘어가고. (pos++) union 해준다.
- http://koistudy.net/?mid=prob_page&NO=2011 문제와 유사.
- 고로. 최종 시간 복잡도는 **O(2^S * M + N^2lgN + QN).** 쿼리당 N 미만의 방법이 있는지 궁금함.
- 실험 결과 앞 풀이보다 2배 빨랐다.

2. Flood Fill (floodfill)

- 택시 거리 D 이하인 정점들에 간선을 이을 때, 컴포넌트의 수와 최대 크기 컴포넌트는?

- Category : Graph, Tree, Data Structures

- 출처 : tjsct.wikidot.com/usaco-open08-gold

- 43 43 43 43 43 43 43 29 0 0

Subtask 1, 2, 3 (43점)

- 1. M이 크지만 좌표범위와 D를 줄여서, 일반적인 격자 그래프를 만들고 Flood Fill을 돌려도 통과되는 서브태스크.

- 2. X 좌표, Y 좌표가 다른데 어떻게 인접하는가? 답은 항상 (n, 1)이다. 구현은 http://koistudy.net/?mid=prob_page&NO=10 참고.

- 3. D가 크기 때문에 1 / 2와 같은 요령이 먹히지 않는다. V = O(M), E = O(M^2)의 그래프를 만든 후, DFS / BFS / Union-Find 등을 돌리면 된다.

1

- 일단 골치아픈 taxicab 좌표를 돌리자.

- 입력 (X, Y)가 주어지면, 점을 (X+Y, X-Y)로 변환할 수 있다. (45도 돌리는 것과 같음. 일차변환 배우셨나요?)

- 45도 돌리면, D 이하의 점을 나타내는 다이아몬드가 정사각형으로 변환된다! 맨하탄 좌표에서 엄청 자주 쓰는 트릭이니 알아두길.

- 이제 Max(|Xj - Xi|, |Yj - Yi|) <= D인 점을 찾으면 됨. (Chebyshev 거리나 Supremum Norm이라고 부름)

2

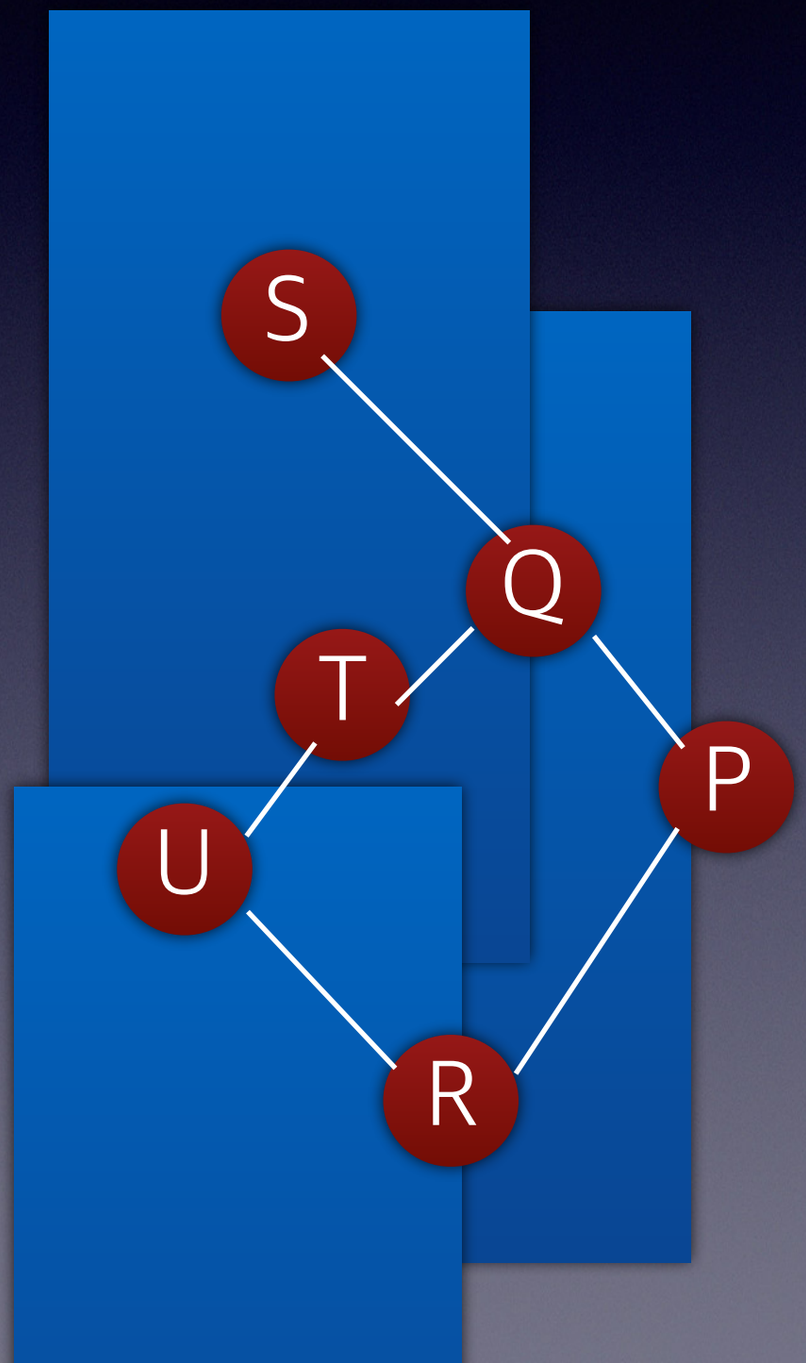
- 일단 알아 둬야 할 것은 “연결성” 이다.
- **문제에서 요구하는 건 두 정점이 연결되어 있는지만 확인하면 됨.**
- 이는, 트리 (포레스트) 만 만들어서 가지고 가도 된다는 뜻이다. 즉 간선을 모두 볼 필요가 없고. 한 O(M) 개 정도만 봐도 된다는 것.
- 핵심은 추리는 것이다. **어떠한 간선을 이으면 연결성만 볼 수 있고 개수도 bound 시킬 수 있을까?**

3

- 앞서 본인이 tree를 언급했기 때문에 한번 트리를 생각해 보자. M개의 점에 대해서 taxicab minimum spanning tree 를 크루스칼로 구하면, 풀 수 있는가?
- 두 정점 [i, j]가 연결되어 있을 조건은 i - j를 잇는 경로에서, 거리 최댓값이 D 이하면 됨. Kruskal MST니까 필요 충분 조건.
- 고로, MST에서 거리가 D 이하일때 union 시켜주면 풀 수 있음.
- ~~Voronoi Diagram을 사용하면 된다~~ 8방향으로 line sweeping하는 꽤 복잡한 Taxicab MST 알고리즘이 있다. 이를 사용하면 AC.
- https://www.topcoder.com/community/data-science/data-science-tutorials/line-sweep-algorithms/

4

- 물론 그런걸 정해로 주진 않았다.
- 어차피 undirected니 왼쪽으로 가는 에지만 생각해보자…
- 감이 오는가?
- 2개 정도의 정점에만 연결시키면, 나머지 정점과도 연결 시킬수 있다!

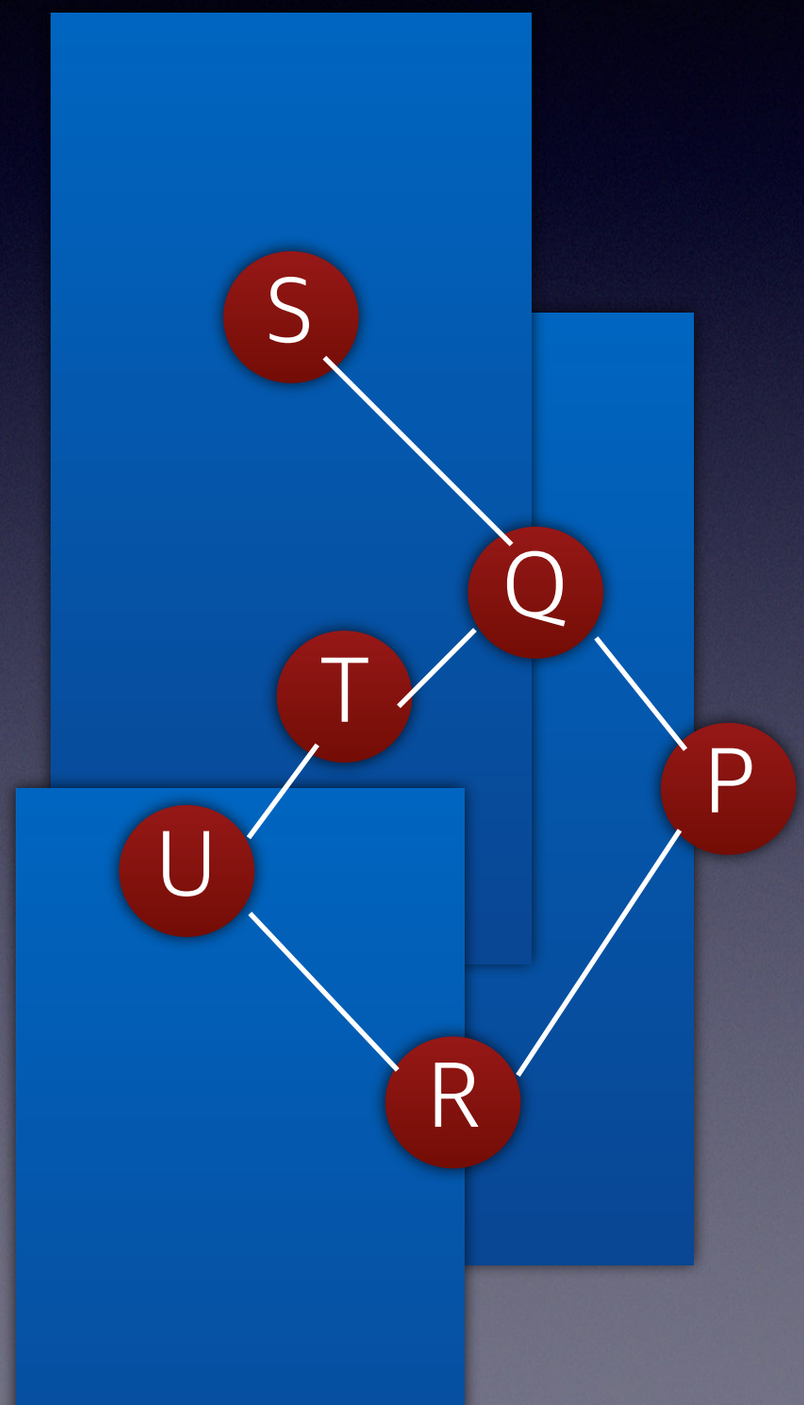


5

- 결국은 직사각형 범위 안에 있는 점들을 모두 덮는 게 핵심인데,

- 앞에 있는 점 2개만 봐도. 직사각형 범위 안의 모든 점을 덮을 수 있다.

- **보지 않은 점들은, 앞에 있는 두 점에 의해 연결되어 있기 때문에!**



6

- x축 기준으로 plane sweeping.
- 점 P(x, y)가 있을 때, P의 왼쪽 위 바닥과 오른쪽 위 바닥을 고려해보자.
- 왼쪽 위 바닥을 덮기 위해서, 왼쪽 위 범위 안에 있는 x좌표 최대의 점 하나를 고른다.
- 왼쪽 아래 바닥을 덮기 위해서, 왼쪽 아래 범위 안에 있는 x좌표 최대의 점 하나를 고른다.
- 또한, 같은 x좌표인 점 두개 역시 붙여야 한다!

7

- 같은 x좌표인 아래점을 연결시키는 것은 쉽다.
- 왼쪽 위 범위, 왼쪽 아래 범위는 [Y - D, Y] | [Y, Y+D] 등으로 표현 가능하다. 거기서 X좌표가 가장 작은 점 두개를 고른다.
- 좌표 압축 + 세그먼트 트리를 쓰면 그러한 점을 구할 수 있다. (Range Max Query)
- 이제 에지수가 3M개인 그래프가 생긴다. subtask 3과 똑같이 Flood Fill을 하면 된다.

3. 개구리 (frog)

- Parent[i] = Minimum j s. t Hj > Hi라 할때, 각 정점에 대해서 Ji번째 Level Ancestor를 구하라.
- 메모리 제한이 15MB다. ~~이런 IOI!~~ 사실상 메모리 제한을 극복하는게 주 과제였던 문제.
- Category : Stack. Ad-Hoc. (Data Structures for some subtasks)
- 출처 : http://wcipeg.com/problem/boi09p4
- 100 100 87 87 50 50 50 13 13 0

Subtask 2 (50점)

- 문제에 엄청 Direct하게 Level Ancestor를 구하라고 써져 있다..
- Sparse Table이라는 DP 기술을 써서 O(NlgN) 메모리 / O(NlgN) time에 풀 수 있다.
- dp[0][i] 는 스택을 사용해서 잘 구해주고.
- dp[i][j] = j의 2 ^ i번째 조상이라 할 때.
- dp[i][j] = dp[i-1][dp[i-1][j]] 를 통해서 DP 테이블을 채우면 됨.
- 많은 참가자들이 Sparse table을 알고 있었다! 굿굿

Subtask 3 (87점)

- Subtask 2에서 메모리를 잘 압축하거나. Subtask 4에서 메모리를 잘 압축하지 못한 사람을 위한 서브태스크였음.

- 서브태스크 2에서 메모리를 압축하는 방법은 :
  - 32 정도만 압축하는 3N 크기의 sparse table을 잡음. (simillar to bucketing) (유재민)
  - 쿼리가 모두 오프라인이라는 점을 사용해서, sparse table을 toggling해줌. 총 4N까지 줄일 수 있음. (오선재)

Subtask 3 (87점)

- 해당 구조가 트리라는 점을 사용해서 암시적인 트리를 만들고, 깊이 우선 탐색 스택에서 dep[i] - Ji 번째 원소를 반환하는 방법도 있었음. (신승원)

- 정해랑 상당히 유사하지만 깊이 우선 탐색에서 벗어나야지 메모리가 줄어듬.

Solution

- 먼저 서브태스크 2를 풀기 위해서 썼던 스택을 생각해보면, 스택의 맨 앞 원소가 내 조상이다.
- Stk[size - T]의 조상은 Stk[size - T - 1]인가?
- 넣을 때 아니었으면 들어가겠음? ㅋㅋㅋ 당연하다.
- 고로, Stk[size - Ji]를 찍는 문제다.
- 그냥 짜면 메모리가 4N이다 (87점). Ji는 한번 계산한 후 필요가 없으니 거기에 답을 저장하면 3N에 풀 수 있다.

여담

- 메모리 제한이 4N은 안되고 3N은 되는 수준으로 빡셌음. 이런 점에서는 상수에 상당히 깐깐한 interactive 스럽기도 하다고 생각.
- 말리기 좋은 문제였다. 그러라고 냈음..
- 서브태스크 3의 DFS 풀이를 서브태스크 4로 확장시키는 방법..? DFS postorder 순서가 1, 2, 3, 4 … N 이라는 점을 관찰하면 된다. 이 점을 알면 거꾸로 DFS를 돌리면 됨.

4. 게임 (game)

- 문제요약 생략
- 사실 얘도 메모리 좀 빡셌다 (0.5GB 풀이가 있긴 함)
- 발로 풀라고 낸 문제였다.. 풀이 해야 하는데 쓸말이 없더라.
- Category : Dynamic Programming
- 출처 : http://www.ioi-jp.org/camp/2015/2015-sp-tasks/index.html
- 100 59 59 59 59 19 19 0 0 0

Subtask 1 / 2 (59점)

- 백트래킹을 짜면 경우의 수가 2^N이니까 19점이 나온다.
- 현재 뽑을 수 있는 3개의 원소. 덱의 맨 위 원소로 DP 식을 세우면 된다.
- DP[p1, p2, p3, peak] =
- if(ok(p1, peak)) update to DP[p2, p3, p3+1, p1] + V[p1]
- if(ok(p3, peak)) update to DP[p1, p2, p3+1, p3] + V[p3]
- DP[1, 2, 3, 0]이 답이다. 시간 복잡도는 O(n^4)이다.

Subtask 3 (100점)

- 사실 저 알고리즘의 시간 복잡도는 O(n^3)이다.
- p3이 p2 + 1이거나 peak + 1을 만족해야 해서 경우의 수가 별로 없거든..
- DP1[p1, p2, peak] = DP[p1, p2, p2+1, peak]
- DP2[p1, p2, peak] = DP[p1, p2, peak+1, peak]
- 앞 식에 이걸 대입하면 답은 DP1[1, 2, 0]
- O(n^3)

Solutions for Day 2

1. 농사 (farming)

- LIS 문제의 응용!
- Category : Data Structures, Dynamic Programming
- 출처 : http://www.ioi-jp.org/camp/2015/2015-sp-tasks/index.html
- 40 30 30 0 0 0 0 0

Subtask 2 (30점)

- 백트래킹 (혹은 비트마스킹) 을 통해 Subtask 1을 맞을 수 있습니다.
- Subtask 2 이상의 무언가를 노리려면 DP식을 만들어야 합니다.
- 2차항 DP식이 있고 1차항 DP식이 있지만, 2차항 DP식은 n^2 미만으로 빠르게 짜기 곤란한 고로 1차항 DP 기준으로 설명하겠습니다.
- 2차항 DP는 좀 느려서 재귀 풀어야 통과될겁니다. 토글링도 해야할지도..

Subtask 2 (30점)

- 먼저 단순히 한 방향만 고려해줘도 전혀! 상관없다는 점을 알아둡시다. / \ 과 같은 산 모양은 반쪽으로 쪼갤 수 있습니다. 조금 있다 더 설명하겠습니다.

- DP[i] = i번 식물의 열매를 팔 때의 최대 이익

- .. 이라 하면, 이제 이를 다른 부분문제와 연관지어야 합니다.

- j번 식물을 판 후, i번 식물을 팔았다는 식으로 연관지읍시다.

Subtask 2 (30점)

- DP[i]가 DP[j]에서 값을 끌어온다는 것은 j 다음으로 i를 팔 것이라는 말이며, 당연히 Hj <= Hi를 만족해야 합니다. (LIS!)
- 그 사이에 있는 풀들 중에 Hi보다 더 잘 자란 풀들은 모두 뽑아버려야 합니다.
- DP[i] = Max(j < i, Hj <= Hi) DP[j] + Cost(j+1, i-1, Hi) + Pi
- Cost(s, e, x) : Sum (s <= i <= e, Hi > x) Ci. 계산에 O(n).
- DP 배열을 채우는 데 O(n^3)의 시간이 들어서, Subtask 2가 됩니다.
- 배열을 뒤집어서 똑같이 계산하면, (정방향 R, 역방향 L), DPR[i] + DPL[i] - P[i] 중 최댓값이 답입니다. 올라갔다 내려갔다는 이렇게 하면 쉽습니다.

Subtask 3 (40점)

- Subtask 3을 하는 방법은 그냥 j를 돌릴때 루프를 잘 해 주면 됩니다.
- 그것도 싫으면 좌표압축 후 2차원 부분합도 괜찮아요.
- Subtask 2를 짤 수 있으면 3은 그냥 쉽습니다. 아무튼 시간 복잡도 O(n^2)를 달성했습니다.
- 이제 nlgn은 어떻게 할까요??

Model Solution 1

- DP[i] = Max(DP[j] + S(j+1, i-1, Hi)) + Pi (Hj <= Hi, j < i)
- 식과 LIS와의 유사성
- 문제를 푸는데 가장 유용한 방법은 **LIS의 풀이법을 고민해보는 것.**
- 그리디 / 세그먼트 트리 방법이 있으나
- 그리디는 아마 잘 안될 거임 ㅇㅇ. 세그먼트 트리 사용.

Model Solution 2

- S(j+1, i-1, Hi)라는 함수가 가장 큰 걸림돌
- 2차인자를 받아서 저렇게 해버리니 최적화가 안됨
- plane sweeping에서 답을 찾자.
- Hi 감소순 정렬을 하고 DP를 잘 돌리면
- S(j+1, i-1, Hi) 라는 함수를 S[i-1] - S[j]로 바꿀 수 있지 않을까??

Model Solution 3

- 이쪽에 기대를 가지고.. plane sweeping을 시작하자.
- Hi 증가순으로, 같으면 i 증가순으로 정렬한 후 문제를 풀자.
- dp[i] = Max(j < i) (dp[j] + S[i-1] - S[j]) + P[i]
- 이후, Hi가 변할때 마다 S 배열을 갱신해 주면 됨.
- 일단 아직까지도 O(n^2)이다. 바꿀 수 있는 여지가 있나?

Model Solution 4

- 식을 약간 풀어주면
- dp[i] = Max(j < i) (dp[j] - S[j]) + P[i] - S[i-1]
- 꽤 괜찮은 식이다. dp[j] - S[j]의 최댓값을 반환하는 세그먼트 트리가 있으면 되니까.
- dp[j] - S[j]는 그때그때 세그먼트 트리에 삽입하면 된다
- 그런데 S[j]가 동적으로 변하는데????

Model Solution 5

- dp[i] = Max(j < i) (dp[j] - S[j]) + P[i] - S[i-1]
- Hi가 바뀔때마다 S[i]가 갱신되는데, 잘 생각해보면 [i, N] 구간에 -Ci를 더해주는 갱신이었음.
- dp[j] - S[j]를 가지고 있는 세그먼트 트리면, Hi가 바뀔때마다 +Cj를 갱신해주는 구간 갱신을 해주면 됨
- 즉, 세그먼트 트리가 구간 최댓값과, 구간 갱신을 지원하면 됨. 이는 Lazy Propagation으로 가능함 (설명 생략)

Model Solution 6

- 이제 풀이를 정리함.
- 1. Hi 순으로, 같으면 i 순으로 입력된 값을 정렬, Si 하나 구할때 쓰는 BIT, dp - S 배열 저장하는 세그먼트 트리 하나 준비.
- 2. Hi가 변할때마다, dp[j] - S[j] 세그먼트 트리에 구간 갱신과 함께, S[i] 하나 구할때 쓰는 BIT 갱신.
- 3. dp[i] = Max(j < i) (dp[j] - S[j]) + P[i] - S[i-1] 일때. S[i-1]은 BIT 에서 구해주고, Max값은 세그먼트 트리에서 가져옴. 그 후 dp[i] - S[i] 값을 세그먼트 트리에 삽입.
- 구현에 약간 차이가 있을 수도 있지만 대략 이렇게 하면 풀 수 있음.

Comments

- 40점에서 100점으로 올라갈 때는 꽤 복잡한 자료구조가 필요합니다. (구간 add + 구간 max query) 많이 풀면 익숙해지니까 열심히 하세요.
- 출제 당시 만점이 나오지 않을 것이라 예상했으며 실제로도 이 문제는 만점자가 없었습니다. 다만 만점 풀이를 생각한 후 코딩을 시도했던 사람은 있었음 (유재민)
- 근데 30점 미만이 그렇게 많은 건 의외였어요 (..)

2. HLD (heavylight)

- 그래프에서의 Heavy Light Decomposition의 개수 세기
- Category : Tree, Dynamic Programming
- 출처 : https://www.codechef.com/LTIME21/problems/HLDOTS
- 100 43 0 0 0 0 0 0

Subtask 2 (43점)

- dp[pos, light] : pos번째 노드를 보는 중이며, 1 - pos 간 경로에서 light edge를 “light” 개 사용함.
- 2 ^ light > n -> 0가지.
- heavy edge를 정확히 한개 이으니, 그냥 모두 다 시도해봅니다.
- [0 .. numSon-1] 개의 자식이 있을 때, H번 노드에 heavy edge를 이었다 치면, dp[son[0], light + 1] * dp[son[1], light + 1]… dp[son[H], light] … 의 경우의 수가 있습니다.
- 이걸 계산하는 게 O(N)이며 모든 자식에 대해서 하면 노드당 O(N^2) 입니다.

Subtask 2 (43점)

- 노드당 O(N^2)면 터지지 않을까요? 합하면 N^3인데?
- (자식수 + 자식수 + 자식수)^2 >= 자식수^2 + 자식수^2..
- 고로 다 합쳐도 O(N^2)입니다.
- 그리고 로그만큼 dp 잡아서 N^2lgN임.

Model Solution

- ai = dp[son[i], light + 1], bi = dp[son[i], light] 일 경우
- 구하는 것이 b1a2a3a4.. + a1b2a3a4 … 임을 알 수 있습니다.
- 가장 간단한 식 전개는 Sigma(b1 / a1 + b2 / a2) * Phi(ai) 겠지만 애석하게도 나눗셈이 힘듭니다.
- 심지어 주어지는 숫자가 소수도 아니지요! (디스크립션에도 mod inverse를 쓰지 말라는 류의 경고가 있습니다.)

Model Solution

- 부분합의 개념과 비슷한 부분 곱을 통해서 문제를 해결하면 됩니다.
- prefix[i] = prefix[i-1] * ai, suffix[i] = suffix[i+1] * ai 모두 O(n)에 계산 가능합니다.
- prefix와 suffix를 벡터에 저장한 후, Sum(prefix[i-1] * bi * suffix[i+1])을 모두 계산해주면 됩니다.
- 재귀함수 내부에 벡터를 선언해야 한다는 점을 명심하세요.

Comments

- Codechef라는 인도판 코드포스에서 나왔던 문제입니다.
- 가장 쉬운 문제로 의도하고 출제했으며, 실제로도 만점자가 나왔던 유일한 문제지만, 문제조차 읽지 않은 사람들이 너무 많았네요 ^^;
- 대회 때 이런 일이 일어나면 100% 본인 책임이니 문제를 열심히 읽읍시다.

3. 신도시 (newtown)

- Category : Data Structures, Computational Geometry, Convex Hull, Binary Search
- 출처 : https://www.acmicpc.net/problem/10746
- 36 36 36 21 15 0 0 0

Subtask 1 (21점)

- 문제에서 요구하는 것은 주어지는 점들을 모두 한쪽으로 몰아넣을 수 있는지에 대한 질문을 대답하는 것입니다.
- 한쪽으로 몰린다는 뜻은, 점들이 모두 부등식 Ax + By + C에서 부호가 같다는 뜻이겠죠. (0인 경우 제외)
- 그러한지 아닌지는 O(N)에 판별할 수 있습니다. 고로 O(QN)에 쉽게 풀 수 있습니다.
- 대회에서 가장 쉬운 서브태스크 중 하나였습니다.

Subtask 2 (36점)

- 이번엔 질의들이 X / Y축에 평행합니다.
- 그러한 경우 X축과 Y축을 따로 생각해줄 수 있습니다.
- 1차원 상의 문제로 생각을 해보면, 최댓값 최솟값만 영향을 주죠?
- 고로 X/Y축 최대/최소 등을 들고가면 풀립니다.
- 생각보다 코딩에서 많이 미끄러졌었던 서브태스크.

Subtask 3 (66점)

- Subtask 3에서는 점이 모두 주어지고 쿼리를 빠르게 처리해야 합니다.

- 이 과정에서 알아둬야 할 것은 볼록 껍질 이외의 점들은 전혀 필요가 없다는 부분입니다.

- 그렇다면, 볼록 껍질을 알고 있으면 빠르게 문제를 해결할 수 있을까요?

Subtask 3 (66점)

- 볼록 껍질을 윗껍질과 아랫껍질로 나눕시다. (X축 최소점 / 최대점을 기준으로 갈랐다고 생각하면 편함)
- 아랫껍질과 직선이 만났는지를 판별하는 법은? 윗껍질의 점들의 Ax + By + C 값를 살펴보면, 기울기가 단조증가 합니다. 이는 삼진 탐색이 가능함을 시사합니다.
- 윗껍질도 동일합니다.
- 삼진 탐색은 O(lgN)에 가능합니다. 윗껍질과 아랫껍질을 잘 만들면 됩니다.

Andrew’s Monotone Chain

- 여담으로 윗껍질과 아랫껍질을 갈라야 할 때 볼록껍질을 만들 수 있는 꿀 알고리즘으로 Andrew’s Monotone Chain이라는 게 있습니다.
- 먼저 X축 순으로 정렬한 후에 (그냥 정렬) 쭉 돌면서, Graham Scan처럼 조건을 만족하지 않는 점을 뺍니다. 이렇게 아랫껍질을 만듭니다.
- 이제 아까 그 점들을 뒤집고 똑같은 걸 하면 윗껍질이 생깁니다.
- Graham’s Scan으로도 물론 풀리지만, 이러한 류의 문제는 이게 더 편하죠. 알고리즘도 어렵지 않습니다.

Model Solution

- Subtask 4를 풀기 위해서는 삽입이 빠르게 진행되어야 합니다.
- X축 기준으로 나눴을 때 삽입을 시키는 데 걸리는 시간은 O(N)입니다. 삽입 정렬처럼 X축 아귀가 맞는 데 적당히 끼워넣고, 양옆을 지워주면서 볼록 껍질을 잘 맞춰주는 거죠.
- 그리고 알다시피 이러한 루틴을 최적화시키는 건 쉽지 않습니다. Splay Tree가 도움이 될.. (까요?) 더 쉬운 방법들을 봅시다.

Solution 1. Bucketing

- 이런 류의 삽입 문제에 일반적으로 사용되는 테크닉은 버킷입니다. 삽입에 Sqrt(N)이 걸리는 자료구조죠.
- Sqrt(N)으로 블록을 나눠서, 삽입을 하고, 양옆으로 볼록 껍질이 아닌 점을 빼준 후, 질의가 들어올 때마다 버킷 전체를 이분 탐색.
- 버킷은 random access가 가능하도록 잘 짜여 있어야 겠죠?
- 아마 <Sqrt(N), lg^2N> 정도에 해결이 가능할 거 같습니다. 참고로, 입풀이입니다 ㅎㅎ

Solution 2. Bucketing 2

- 더 쉬운 방법이 있습니다.
- 먼저 정렬된 리스트가 있으면 선형 시간에 볼록 껍질을 만들 수 있는 걸 알 수 있습니다.
- “버퍼”를 잡읍시다. 버퍼는 새로 추가된 점을 포함하는 배열로써, Sqrt(N)번 마다 볼록 껍질에 합쳐집니다. 껍질이 버퍼를 합치는 데 드는 시간은 O(N) 이며, 이 과정은 Sqrt(N) 번 시행됩니다.
- 이 과정 때문에 버퍼의 크기는 Sqrt(N)으로 제한됩니다. 껍질에서 이진 탐색, 버퍼에서 선형 탐색 하면 쿼리당 O(Sqrt(N))의 시간이 소요됩니다. 고로 (N+Q)Sqrt(N)이 소요됩니다.
- 이 풀이에서 나온 방법 중 코딩이 제일 쉬운 풀이인거 같네요.

Solution 3. Binary Transform

- 정확한 이름을 모르겠어서 정해에 언급된 논문 이름으로 부르겠습니다.. http://usaco.org/current/data/sol_fencing_gold.html

- Binary Transform을 사용하면 이 문제를 (N+QlgN)lgN에 해결할 수 있습니다. 버킷을 사용하지 않는 방법입니다. (버킷을 일반화시킨 방법에 속하죠)

- 이 방법은 버킷 대신, 2^k 크기의 컨벡스 헐들을 lgN개 가지고 있습니다.

Solution 3. Binary Transform

- 이 방법은 버킷 대신, 2^k 크기의 컨벡스 헐들을 lgN개 가지고 있습니다.
- 각 컨벡스 헐이 가지고 있는 원소의 수는 정확히 2^k개여야 합니다.
- 만약에 전체 원소 수가 5개면, 2^2 / 2^0 버킷이 차있죠.
- 19면, 2^4 / 2^1 / 2^0…
- 버킷이 가지고 있는지 없는지는 사이즈의 이진 표현과 연관되어 있습니다.

Solution 3. Binary Transform

- 삽입의 경우에 대해서 문제를 해결할 때는 받아올림의 개념을 생각해봅시다.
- 점이 하나가 삽입되었을 경우에, 만약에 자료 구조의 크기가 2^k를 넘는다면, 분명히 2^1 크기일 겁니다. 이 자료구조를 비워주고 2^1 자료구조에 점을 채워야 할 것입니다.
- 2^1 자료구조에 점이 비어있다면, 점이 들어오면서 자료구조가 찰 것입니다. 차 있다면, 2^2 크기가 되겠죠. 받아올림이 될 것입니다. 반복..

Solution 3. Binary Transform

- 감이 오나요?
- 일단 쿼리에 대해서 살펴봅시다. 모든 컨벡스 헐이 차 있다고 가정을 하면 O(lgN)개의 컨벡스 헐에서 O(lgN) 짜리 이진 탐색을 하므로 O(lg^2N)의 시간이 듭니다. 쿼리는 O(lg^2N)입니다.
- 자료구조 자체의 시간은 얼마가 될까요? 2^k 크기의 버킷에 대해서 초기화되는 횟수는 N / 2^k 번입니다. 초기화에 걸리는 시간은, Merge Sort의 요령으로 O(Size)에 가능합니다. 고로 각 컨벡스 헐에 대해서 O(N)이 걸립니다. 귀찮으면 O(NlgN) 해도 큰 차이는 없습니다.
- 이러한 컨벡스 헐이 O(lgN) 있으니, O(NlgN)이 걸립니다. 고로 복잡도는 O((N + QlgN)lgN)입니다.

Solution 3. Binary Transform

- 다만, 삼진 탐색 조건을 만족하기 위해서는 컨벡스 헐에서 원소가 빠져야 합니다. 이러한 경우에는 2^k가 정확히 아귀가 맞지 않는다는 것을 알 수 있습니다.

- 해결을 위해서는, 2^k 크기를 넘기기만 하면 그냥 위쪽 버킷으로 넘겨버리면 됩니다.

- 원소가 더 줄었기 때문에 앞 문제보다 시간 복잡도가 느릴 일은 없습니다.

- 사실 입풀이긴 해서.. 안될수도 있는데 (될거에요!) 정 안되면, 쓰레기 점들을 지우지 말고 다른 저장소에 보관한 후, 합쳐도 되요. 그러면 2^k를 항상 만족하죠.

Comments

- 만점 방지용으로 낸 문제였습니다. 사실 66점은 있을 줄 알았는데..

- 섭테 4의 Binary Transform은 저도 풀이 작성하면서 처음 배워본 방법입니다. 의도된 정해는 버킷이었습니다.

- 사실 저도 저 문제에 대한 코드가 없습니다 ㅠㅠ 시간이 되면 짤게요..

4. 물리학자 (physics)

- Category : Graph, Shortest Path
- 출처 : https://www.acmicpc.net/problem/5952
- 41 41 41 41 0 0 0 0

Shortest Path Algorithms

- 최단 경로 알고리즘으로는 크게 세가지가 알려져 있습니다. **모두 알아야 합니다.** 작동 시간 순으로 나열되었으며, 하이라이트된 부분은, 각 알고리즘의 중요한 특이점입니다.

- 1. Floyd-Warshall : DP에 기반한 **모든 시작점** 최단 경로 알고리즘, 음수 간선 처리 가능. 인접 행렬 사용. 짧은 코드. O(N^3)

- 2. Bellman-Ford : DP에 기반한 단일 시작점 최단 경로 알고리즘, 음수 간선 처리 가능. 인접 리스트 사용. O(NM)

- 3. Dijkstra : 그리디에 기반한 단일 시작점 최단 경로 알고리즘, 음수 간선 처리 **불가능**. 단 하나도 있으면 안됩니다. (사실 하나는 될지도..) 인접 리스트 사용. O(N^2 + M) or, O(MlgM)

Subtask 1, 2 (41점)

- 먼저 0점 이상을 노리려면 다익스트라가 음수 간선이 있을 때 최단 경로를 찾지 못한다는 것을 알아야 합니다. 섭테 1에 저격 데이터가 있습니다 (사실 대회중 넣었습니다..)

- 19점은 Floyd-Warshall, 41점은 Bellman-Ford 알고리즘을 사용하면 쉽게 풀 수 있습니다. 이번 대회 중 가장 쉬운 서브태스크였던 거 같습니다.

Branch and Bounding Bellman Ford?

- 잘만 짜면 Bellman-Ford 자체가 O(VE)보다 조금 더 빨리 도는 걸로 알려져 있죠. 물론 저격하기 쉽습니다.
- Bellman-Ford의 유명한 커팅으로 SPFA (Shortest Path Faster Algorithm) 이 존재합니다. 실제로 MCMF 같은 데 많이 사용되는 알고리즘입니다.
- http://wcipeg.com/wiki/Shortest_Path_Faster_Algorithm
- SPFA는 저격 데이터가 없는줄 알고 제가 그렇게 말하고 다녔는데 (…) V = 25000인데도 SPFA로 10초가 걸리는 데이터가 존재합니다. 특히 문제의 입력이 작아서 유혹당하기 쉬운 부분입니다.

Model Solution

- 일반적인 경우에 이 문제를 해결할 수 있었다면, 벨만 포드나 다익스트라같은건 배우지도 않았겠죠? 문제의 그래프의 특성을 관찰합시다.

- Ai에서 Bi로 타임머신을 통해 도달했다면, 도로나 타임머신으로 다시 돌아갈 수 없다는 것은 무슨 의미일까요? 단순히 음수 사이클이 없다는 뜻일까요? (없긴 없겠죠)

- 타임머신으로 연결되어 있는 두 도로는, 도로망에서 연결되어 있지 않다는 것을 뜻합니다.

Model Solution 2

- **타임머신으로 연결되어 있는 두 도로는, 도로망에서 연결되어 있지 않다는 것을 뜻합니다.**

- 도로망으로 연결되어 있는 도시들을 컴포넌트로 묶어봅시다.

- 컴포넌트 하나를 정점 하나로 대응시켜서 타임머신만 간선으로 보면, 그래프는 DAG임을 간파할 수 있습니다.

- 컴포넌트를 묶는 건, 도로망 하나를 잡고 Flood-Fill을 하는 게 가장 편리합니다. SCC를 구해도 되고 Union-Find를 구해도 되고 방법은 다양합니다.

Model Solution 3

- 우리는 DAG에서 최단 경로를 찾는 법을 알고, 양수 가중치 그래프에서 최단 경로를 찾는 방법을 압니다. 시간 복잡도도 모두 준 선형이죠!

- 이 두 과정을 섞는 방법을 얘기해야 하는데, 관찰의 과정을 얘기하기가 애매하니 바로 풀이를 찌르겠습니다.

Model Solution 4

- 먼저, 도로 컴포넌트를 정점으로 두고, DAG를 압축된 형태로 가지고 있습니다. (SCC와 비슷한 방법으로)
- 각 도로 컴포넌트마다 각자의 priority_queue를 가지고 있습니다. 다익스트라를 컴포넌트마다 돌리겠다는 말이죠.
- 초기에, 시작점이 있는 컴포넌트 priority_queue에 노드를 넣은 후.
- DAG를 위상정렬하면서 차례차례 문제를 해결합니다.

Model Solution 5

- DAG의 위상정렬은 큐를 통해서 진행할 것입니다. indegree가 0인 컴포넌트를 해결합시다.

- 먼저 priority_queue 안에 있는 원소들이 있을 것입니다. 이들을 뿌려주면서 다익스트라를 진행해 나갑니다. 각 정점까지 도달하는데 걸리는 시간을 알 수 있습니다.

- 해당 컴포넌트와 DAG로 연결된 컴포넌트들이 존재할텐데, s -> t로 w만큼의 시간이 걸린다면, t의 컴포넌트에 (dist[s] + w) weight의 노드를 넣어줍니다.

- 이걸 반복하면. 답을 찾겠죠? 시간 복잡도는 O(ElgE)입니다.

Comments

- KOI 지역본선 3번에서도 낭설이 대회를 망친(?) 사례가 간혹 있습니다. 이번엔 그런 경우는 없는 거 같지만 아무튼 조심합시다. 시간 제한과 인풋 제한에서 유혹당하기 좋은 거 같습니다.
- 다익스트라 저격 데이터는 https://www.acmicpc.net/problem/7145 참고. 이걸로 만들었습니다.
- ~~유쾌해~~
- 수고하셨습니다.