정과세 풀이집

14004 구재현

(Help Koosaga Get a Job)

강의실 배정 (interval)

- Category : Greedy, Sorting
- First Solver : 14067 오선재 (15분)

Introduction

- 유명한 문제! (Interval Graph Coloring)
- 저번 나코더 NP-hard 시간에 수업하다가, 이 문제가 없는 걸 깨닫고 냈음
- 정말 풀이가 많은데, 그 중 가장 간단한 풀이를 소개

Greedy Strategy

- 선분을 스캔하면서, 가장 많은 선분이 쌓여있는 좌표엔, 몇개의 선분이 쌓여있는지 검사. 최댓값을 T라 하자.
- 당연히, T 미만의 Coloring은 존재할 수 없다.
- 그리고, T개의 색깔로 Graph Coloring이 가능하다.
- 자세한 증명은 생략.

Implementation

- 좌표 압축 -> 배열 사용시 O(n^2)
- 좌표 압축 -> 변홧값 배열 사용시 O(nlgn)
- std::map 사용시 O(nlgn)..
- 다양합니다
- 세번째가 제일 짧음.

Greedy Strategy 2

- 실제 강의실을 배정하듯이 그리디 전략을 짜는 법 (simulation). 이 풀이가 인기가 제일 많았던 거 같음.
- 강의실을 저장하는 heap을 만들자. (std::set도 good)
- 만약 쓸 수 있는 (=끝점이 작은) 강의실이 heap에 없다면, 만들자 (끝점을 힙에 추가)
- 있다면 그 강의실에 현재 강의를 배정하자.
- 이 개념을 추상화하면 풀이 1로 귀결됨. 고로 요놈도 증명 생략.
- lower_bound(s.begin(), s.end() …) 쓰지 마세요!

김치 (kimchi)

- Maximize (i - j) * Ti + Vj (j >= i - D)
- N <= 200000 였지만
- 대충 짰더니 메모리 320MB 먹어서 (STL ㅅㅂ)
- 코이스터디 감안하고 N <= 100000 잡음.
- First Solver : 14069 유재민 (45분)
- Category : Data Structures

Subtask 1. CHT

- 일단 이 경우부터 먼저 생각을 해 봅시다.
- i * Ti - (Vj - j * Ti) 라는 식으로써..
- 아주 전형적인! Convex Hull Trick 식입니다.
- **Q : CHT는 DP 최적화 기법 아닙니까?**
- **A : 틀렸습니다. CHT는 선분을 다루는 자료구조입니다.**

Subtask 1. CHT

- i * Ti – (Vj – j * Ti)
- (j, Vj) 를 포함하는 선분 집합을 가지고 있습니다. -Ti는 증가합니다. 기울기도 증가합니다.
- 1. Query -> Amortized O(1)
- 2. Insert -> Amortized O(1)
- Total Complexity is O(N)

How to deal with N > D?

- D < N이면 음..
- 그냥 CHT에서 삭제하면 안되냐고요? 반례가 있습니다.
- 적어도 우리가 아는 범위에서 **CHT는 원소 삭제가 가능한 자료구조가 아닙니다.**
- **이런 상황, 다른 문제에서는 어떻게 해결했었나요??**

Model Solution

- 세그먼트 트리는 이런 상황에서 일반적으로 적용 가능한 자료구조입니다.
- 문제에서 요구하는 것은 구간 쿼리이기 때문에, 세그먼트 트리를 통해서 효율적으로 해결 가능합니다.
- 즉, 세그먼트 트리의 노드 [i, j]는 구간 i - j의 CHT를 가지고 있으며, lgN개의 노드에 선분 삽입. lgN개의 노드에서 쿼리를 때리면 됩니다.

Segment Tree with CHT

- 공간 복잡도를 따져봅시다.
- 공간 복잡도 T(n) = O(n) + 2T(n/2) = O(nlgn). nlgn의 복잡도를 가지고 있습니다. 각 노드에 들어가는 원소 수입니다.
- CHT는 합쳤을 때 원소 수만큼의 시간이 소요되므로 O(nlgn)에 모든 작업이 가능합니다.
- Merge Sort Tree와 비슷한 요령입니다. 노드 안에는 STL deque를 넣으면 됩니다.

Range Queries

- 결국 들어오는 노드가 nlgn개이니 O(nlgn)에 총 문제가 해결되는 것은 자명합니다.
- 원소 삽입은 lgN개의 노드에 Update를,
- 구간 쿼리는 lgN개의 노드에 Query를 날리면 됩니다.
- 구간 갱신이 없기에 비재귀로 쉽게 코딩 가능합니다.

Lessons

- 1. **Segment Tree는 일반화하기 쉬운 자료구조입니다.** 노드에 Max, Min만 넣을 필요가 없습니다. Convex Hull Trick을 넣어도 잘 작동하는 자료구조입니다.

- 2. **Convex Hull Trick는 DP와 아무런 상관이 없습니다.** 그저 선분 집합을 관리하기 좋은 자료구조이고, DP를 돌릴때 주로 활용될 뿐입니다.

- 3. **문제들은 다 비슷비슷합니다.** CHT 점화식이라 어려우면 더 쉬운 문제로 바꿔서 풀면 됩니다. 그러면 세그먼트 트리의 필요성이 나옵니다.

- .. 저도 숙지해야할 교훈들입니다.

Another Solution (by gs14069)

- 1. 분할 정복 최적화로 알려진 아이디어를 사용하는 방법이 있습니다. 김치를 꺼낸 점에 대해서 담궈야 하는 점은 단조적인 양상을 띄기 때문에, NlgN의 단순 계산으로 해결됩니다.

- 이 풀이는 정해보다 메모리 / 시간 / 코드 모든 면에서 훨씬 우월합니다! 갓재민 찬양해..

- 다만, 정해는 온라인 풀이인데 비해서 이 풀이는 오프라인이라는 점. 고로 Vj가 아니라 DPj면 (아마) 풀 수 없습니다. (Vj로 줘도 다른 풀이가 없을 거라 생각했거든요 ㅠㅠ)

- 사실 문제를 딱 보면 CHT로 접근하기 쉽상이라 분할 정복을 떠올리는게 쉬운 게 아님.

Another Solution (by gs14026)

- 조금 더 정석적인 분할 정복 기법을 사용한 풀이로서 모든 쌍을 다 봐야 하는 일이 있을때 분할 정복이 유용하다는 점을 비집고 들어갑니다.

- D 이하의 거리만 보는 CHT가 필요한데, 시작점과 끝점의 집합이 서로소이면 이는 O(N)에 해결됩니다.

- 고로 T(N) = T(N/2) * 2 + O(N) = O(NlgN)에 문제 해결

Slower Solution

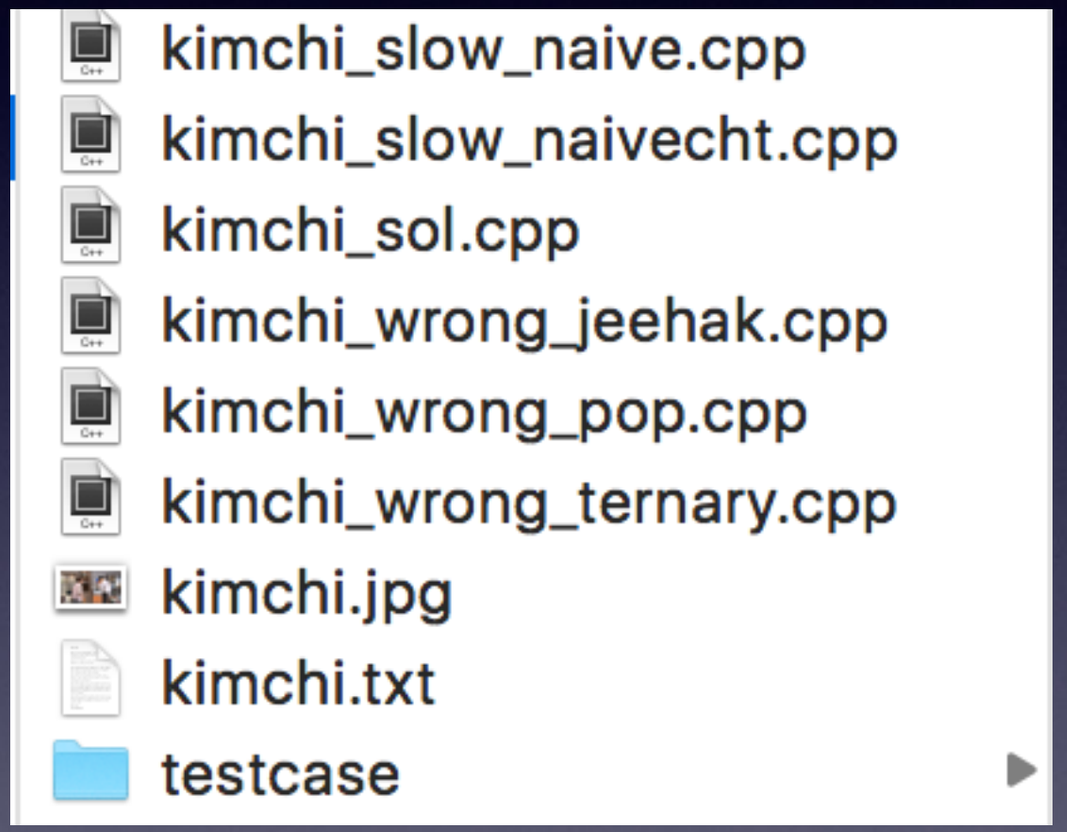
- 조금 더 느린 풀이들도 있습니다. TL을 1000ms로 유지하고 싶어서 악랄하게 막지는 않았습니다. (막을라 그랬으면 N=500000 ㅋㅋㅋ)
- N이 코이스터디 감안해서 10만으로 줄었다는 점을 감안하면, 맞기 상대적으로 쉽지 않을까 생각해봅니다.
- 먼저, lgN 대신 Sqrt(N)의 버킷을 사용하는 방법이 있으며,
- Convex Hull Trick 내에서 이진 탐색을 사용하는 방법이 있습니다.
- 이진탐색은 스무스하게 통과합니다.. 버킷은 모르겠음

Test Data Generation

- 틀린 프로그램 5개를 작성할 예정입니다.
- 아으 시바 진짜 다 작성하고 뻑데이터 만들었습니다. 하루종일 해도 힘들더라고요. 하지만 이래도 야매로 맞는 사람이 나올거 같아요 ㅋㅋㅋㅋ
- 테스트 케이스는 총 30개로 코이스터디에 부담 안 갈 정도로 적당히 만들었습니다. 30개 이하로는 안됩니다. 30개 정말 꽉꽉 채워넣었습니다.
- 대충 데이터 만들면 쉽게 맞을 수 있는 문제라, 테스트 데이터 만드는거 **굉장히 힘들었습니다**. 이야기를 공유하려 합니다.
- 테스트 만드는 거 **굉장히** 힘들었지만, 코드포스 Polygon 시스템과 이상헌(<<중요) 의 도움으로 그나마 편했습니다. 감사합니다 (_)

Wrong Codes

- slow_naive : O(ND).
- slow_cht : 모범답안과 같지만, CHT 내부를 선형시간에 순회함.
- wrong_jeehak : 길이를 무조건 최대화하고, 25000번만 가봄. (폴리곤에는 TL 5초로 실험함)
- wrong_pop : CHT에서 선분 기울기가 i - D 미만이면 무조건 pop 시키고 봄.
- wrong_ternary : 모든 선분이 CHT에 있다고 가정하고, 배열 상에서 이분 / 삼분탐색. 코드에는 이분 탐색으로 구현함



how to hack it 1

- slow_naive : 생략.
- slow_cht : CHT 안에 항상 선분의 개수가 최대화되면 됩니다. 장독대 가치를 적당히 단조감소 형태로 주면 TLE가 납니다.
- wrong_jeehak : 모두 랜덤으로 주면 Ti도 엄청 크고 Vi도 엄청 큰데, Ti는 길이가 곱해지니 긴것만 보는 걸로 이득을 볼 수 있죠. Ti를 너프시키고 Vi를 조금 괴상하게 주면 됩니다.

how to hack it 2

- wrong_pop : 앞에 있는 선분보다 가중치가 작지만, 앞에 있는 선분이 사라지면 영향력이 있는 선분을 만들면 됩니다. 말이 쉽지 큰 데이터에 대해 일반적으로 작동하는 경우를 만들기 힘들어서, 그냥 폴리곤 스트레스 돌렸습니다. 폴리곤 없었으면 저 풀이 AC뜨고 멘붕하는 저를 볼 수 있었음..

- wrong_ternary : 저건 대충 랜덤 때려도 틀리더라고요..

- 테스트 데이터 정보를 모두 공개했지만, 이 정보의 악의적인 사용은 도움이 안된다는 것도 기억해주세요.

토큰 게임 (tokens)

- N <= 40

- 갓문제.

- Category : Ad-Hoc. Greedy.

Subtask 1. N <= 10

- 승원이는, 자신의 여러 수 중에서 재현이가 어떤 수를 선택하든, 가장 못하게 되는 수를 선택할 것이다.
- 게임을 흔히 이런 식으로 재귀적으로 표현한다! 복잡도는 대략 O(N!) 이다.
- 전 안해봐서 모르지만, 이렇게 하면 될거에요.

Model Solution

- 40 이하의 N에 대해서는, 2^(N/2)의 복잡도를 쉽게 예상할 수 있다.
- 하지만 실제로는.. 2^N도 어렵지 않나요? (아님 말고요..)
- **의도한 시간 복잡도는 입력에 선형, O(N^2) 입니다.** 이것보다 느린 풀이는 잘 모르겠네요.

Different Approach

- 토큰은 T1, T2라는 두개의 서로소 집합으로 분할됩니다. T1이 승원이 거라고 합시다.
- Sum(i∈T1 & j∈T1) A[i][j] - Sum(i∈T2 & j∈T2) A[i][j] 를 최대화 해야 하고요!
- 근데....
- Sum(i∈T1 | j∈T1) A[i][j] - Sum(i∈T2 | j∈T2) A[i][j] 는요??
- 네. 같은 식입니다. ((i∈T1 & j∈T2) | (i∈T2 & j∈T1)) A[i][j] 가 더해지고 빼집니다.

충격과 공포의 식전개...

- Maximize X =
- Sum(i∈T1 & j∈T1) A[i][j] - Sum(i∈T2 & j∈T2) A[i][j]
- = Sum(i∈T1 | j∈T1) A[i][j] - Sum(i∈T2 | j∈T2) A[i][j]
- = Sum(i∈T1) A[i][j] + Sum(j∈T1) A[i][j] - Sum(i∈T2) A[i][j] - Sum(j∈T2)A[i][j] - X
- **2X = Sum(i∈T1) A[i][j] + Sum(j∈T1) A[i][j]- Sum(i∈T2) A[i][j] - Sum(j∈T2)A[i][j].**

아직 잘 모르겠다고요?

- Maximize Sum(i∈T1) A[i][j] + Sum(j∈T1) A[i][j]- Sum(i∈T2) A[i][j] - Sum(j∈T2)A[i][j]

- Let X[i] = Sum(j<N) A[i][j], Y[i] = Sum(j<N) A[j][i]

- Maximize Sum(i∈T1)(X[i] + Y[i]) - Sum(i∈T2)(X[i] + Y[i])

- Maximize 2 * Sum(i∈T1)(X[i] + Y[i]) - Sum(i<N)(X[i] + Y[i])

- Maximize Sum(i∈T1)(X[i] + Y[i])

...

- Maximize Sum(i∈T1)(X[i] + Y[i])
- X[i] = i행의 합. Y[i] = i열의 합이라 할 때. 재현이가 안 가져간 토큰 중 X[i] + Y[i]가 가장 큰 것을 고릅니다. 끝!

- IOI 2014 Friend처럼 조금 깨는 풀이의 문제입니다. 이런걸 풀려면 운과 머리가 모두 좋아야 하죠.
- 사실 이 문제는 조금 찍기가 수월한 문제긴 한데.. N이 작아서 오히려 엄두를 못 내지요. (실제로 대회 때 다들 그랬던 거 같고)
- 문제 풀이의 심리학..
- 구현 방법은 생략하겠습니다. 정렬 한번으로 init O(N^2) / query O(1)에도 해결이 가능합니다. 모범 답안은 O(N^2) / O(N)입니다.