정과세 2차 문제
solution sketch

14004 구재현

Problem Set

- 수행평가 1 : Implementation
- 수행평가 2 : Greedy, DP, Data Structures
- Stop Making Sense : Geometry
- 예상 난이도 : 수행1 < Sense < 수행2

Problem Set

- Stop Making Sense의 이름은 아무렇게나 지은 것입니다. 문제와 관련이 없습니다.
- 수행평가 2는 원래 알고리즘 수행에 내려고 했던 문제입니다. 데이터와 디스크립션 모두 준비했었지만 모종의 이유로 지하철 문제가 대신 올라갔습니다. (이유가 기억이 안남)
- 수행평가 1은 위 문제를 재민이한테 설명할 때, 재민이가 잘못 이해하고 풀었던 버전입니다. (원래 재민이한테 주프 출제하라고 던질 계획이었음) 수행평가 2와 input data가 똑같습니다.
- 출제자가 시간이 없는 고로 빠르게 갑니다.

수행평가 1 Solution 1

- M >= 2라고 가정하겠습니다. M = 1일때는 답이 무조건 n+1, 1이며 예외처리 해주면 됩니다.

- 먼저 답이 O(ln(N) / ln(M)) 이라는 사실을 쉽게 증명할 수 있습니다. (M ^ T <= N - T + 1을 만족하는 모든 T는 답이 될 수 있음)

- 답의 길이를 알면 문제를 푸는데 도움이 될까요?

수행평가 1 Solution 2

- 답의 길이를 T로 정한다면, N - T + 1개의 substring이 생깁니다.
- vector에 이 스트링을 통째로 넣고, 이를 set에 넣습니다. (set<vector<int>>)
- set의 크기가 M^T가 아니면 답이 될 수 있습니다. 아니면 답이 안됨
- 이 판별은 O(NT * lgN)에 가능합니다.
- 모든 T에 대해서 시도하면 O(Nlg^3N)에 작동합니다
- 실전에서 빠르게 작동하며, 데이터가 약해서 특히 더 빠릅니다. (T를 크게 강제한 데이터가 없음) 모범 풀이는 저렇게 작성되었습니다.

수행평가 1 Solution 3

- Rabin-Karp 해시를 사용하면 O(NIg^2N)에 풀 수 있습니다.
- 모든 T에 대해서 시도를 하는데 해시값을 O(N)에 열거할 수 있고 서로 다른 문자열의 개수를 O(NIgN)에 계산할 수 있습니다.
- 문자열의 범위가 long long 안에 들어오기 때문에 해시 충돌이 나지 않음을 보장할 수 있습니다.
- O(NIgN)도 가능합니다. 생략.

수행평가 2 (20점)

- 수행평가 1의 “연속할 필요가 없는” 버전으로 풀이는 완전히 다릅니다.
- 20점 풀이는 모든 수열을 백트래킹으로 나열하는 것이 되겠죠.
- 수열 S가 있을 때 수열 T의 부분수열인지를 판별하는 것은 어떻게 할까요? O(N+M) 그리디 알고리즘이 있습니다.

수행평가 2 (20점)

- 그리디 알고리즘은 정말 간단합니다. 길이 N의 수열이 있을 때, M의 원소들을 순서대로 돌면서, N에 똑같은 원소가 나올때까지 포인터를 옮기는 겁니다.
- 적당히 커팅하면 20점이 충분히 나올 겁니다.
  - int pointer = 0;
  - for(all elem in M)
    - while(M[elem] != N[pointer]) pointer++;
    - pointer++;

수행평가 2 (50점)

- DP를 돌릴 수 있는 optimal substructure를 관찰할 수 있을까요? 충분히 관찰할 수 있습니다.

- DP[i] = 그리디 알고리즘으로 [1, i]까지의 수열이 매칭되었다고 칠 때, 최대 수열의 길이로 정의합니다.

- DP[i] = DP[j] + 1일텐데, j에서 바로 i까지의 수열을 매칭시켰다면, [j+1, i-1] 사이에 원소 a[i]가 없어야 할 것입니다.

- 답은 그러한 DP 값 중 최대인 값입니다.

수행평가 2 (50점)

- DP[i]와 함께 Cnt[i]를 가지고 가는데, Cnt[i]는 DP[i]를 최소화하는 경우의 수겠죠?

- DP 테이블을 갱신하면서, Cnt[i] == (dp[j] + 1 == dp[i]를 만족하는 Cnt[j]의 합)으로 정의하면 해결 가능합니다.

- 서술한 알고리즘을 적당한 복잡도로 구현하면 50점을 긁을 수 있습니다.

수행평가 2 Solution 1

- 서술한 알고리즘을 다듬기 위해서 Nxt[i, j]를 정의합니다.
- Nxt[i, j] -> [1, i-1] 구간에서 값이 j인 가장 오른쪽 값의 인덱스.
- DP[i] = Max(DP[Nxt[i, j]]) + 1 for all 1 <= j <= M
- 으로 정의 가능합니다. Count는 알아서 잘..

수행평가 2 Solution 2

- Nxt[i-1] 배열과 Nxt[i] 배열의 차이는 원소 하나밖에 안 납니다. 당연하죠?

- 크기 M의 Range Minimum Query를 사용합니다. RMQ의 i번째 인덱스는, 해당 상태에서의 DP[Nxt[pos,i]] 값을 가지고 있습니다.

- 차이가 원소 하나 뿐이니, RMQ에 갱신 한번, 쿼리 한번 돌아가면 풀 수 있습니다.

수행평가 2 Solution 3

- Cnt[i]를 빠르게 구하기 위해서, CntSum[DP[i]]를 정의하겠습니다.
- CntSum[Val] == DP[i]의 값이 Val이며, Nxt[] 배열에서 active한 원소들의, 경우의 수의 합
- RMQ와 비슷한 요령으로 각 i당 상수 시간에 관리 가능하며 갱신 가능합니다.
- 이렇게 하면 풀 수 있습니다.
- 코딩 팁 : N+1 위치에 더미 원소를 넣어보세요 (a[N+1] == 0 이라던가)

Stop Making Sense (20점)

- 문제에서 요구하는 것이 볼록 껍질의 넓이임을 빠르게 간파할 수 있습니다.

- 볼록 껍질을 계산하는데는 O(NlgN)의 시간이 소요됩니다.

- 해당 점을 빼놓고 볼록 껍질을 계산하는 것을 반복하면 쉽게 O(N^2lgN)에 풀 수 있습니다.

Stop Making Sense Solution 1

- 먼저 중요한 것은 컨벡스 헐 안에 있는 점만 뺐을 때 넓이가 변한다는 점입니다.

- 고로 아닌 점들은 신경을 안써도 됩니다.

- 컨벡스 헐 다각형에서, 원점을 기준으로 0 ~ H-1로 인덱스를 매깁시다.

- 이 때 또 하나 관찰할 것은 i (1 <= i <= H-2) 번째 점을 빼면, [0, i-1] 과 [i+1, H-1] 구간의 다각형은 그대로라는 점입니다. i 기준 양옆 부채꼴만 변화가 생깁니다.

Stop Making Sense Solution 2

- 그러면, 원점 정렬된 점에 대해서, [i-1, i+1] 각도 구간에 포함된 애들만 다시 컨벡스 헐 알고리즘을 돌리고, 이어붙이면 어떨까요? 물론 i번째 점은 제외하고요.
- i-1, i+1번째 점은 완벽하게 껍질 상에서 유지된 채로 쉽게 넓이를 구할 수 있습니다.
- 부분합을 사용해서 볼록껍질의 넓이를 잘 저장했다면, 넓이는 lsum[i-1] + rsum[i+1] + [i-1, i+1 구간 껍질 넓이]가 되겠습니다.
- 시간은 [i-1, i+1] 구간의 점들의 개수입니다.

Stop Making Sense Solution 3

- [i-1, i+1] 구간을 모두 덮으면 원주를 두번 덮는 꼴이 됩니다. 고로 한 점은 많아야 두번 처리됩니다.

- 고로 앞서 설명한 알고리즘은 O(NlgN)에 문제를 해결합니다.

- 여담으로, 껍질 상 0번째 점과 H-1번째 점이 처리가 안되는데, O(NlgN) 컨벡스헐을 두번 더 돌리면 됩니다.