삼성전자 SW 역량테스트 B형, 어떻게 공부해야하나요?

킹갓제네럴충무공 박트리 2018. 10. 21. 17:14

A형을 통과한 대학교 재학생 or 졸업생 분들이나 삼성전자 SW직군에 입사한 사원분들은 SW 역량테스트 B형 일명 Professional 등급을 취득하고자 시험을 볼 것이다.

이 포스팅은 B형(Pro) 시험을 보는 사람들에게 시험 유형에 대한 정보와 공부하면 좋을 배경지식 그리고 내가 생각하는 B형 테스트의 통과 컷, 풀어보면 좋을 문제 등을 작성하고자 한다.

들어가기 앞서 뇌피셜 B형 난이도는 100 * A형 난이도이다. 상당한 난이도 갭이 있음을 인지하고 공부하면 되겠다.

** 2018-10-31 초안 작성

배경지식

먼저 공부해야하는 배경지식들에 대해 적어보겠다. B형 시험부터는 STL이 사용불가능 하므로 자주 사용하는 STL의 경우 구현법을 알아야 하겠다. [stl 구현한 깃허브 // 변수명이 이상하거나 중요함수 몇개가 없는 등 업데이트가 더 필요하다.]

시험장에서 시험 시스템 로그인 하면 화면 11시 방향에 레퍼런스 코드? or 레퍼런스가 있다 그 안에 꽤 많은 코드들이 있으므로 굳이 암기까지 할 필요는 없고 동작 과정을 이해하고 응용하는 법을 많이 공부하는 것이 좋다.

-- 필수

- 해싱 : B형보는데 해싱을 모르고 응용할 줄 모르면 바로 혀 깨물어야됨
- 링크드리스트 구현 : 링크드리스트 쓰는 문제 진짜 많이 나온다
- 트리 구현 : 자식 수가 안 정해진 트리를 구현하는 법은 무적권 알아야 됨
- 메모이제이션 : 한 번 계산한 값을 다시 계산안하는 개념이 있어야됨
- 비트마스킹 : 변수를 쪼개서 공간복잡도를 아끼면서 저장하거나 다양한 비트 연산을 이용해서 코드를 최적화 하는 방법들
- 이분탐색 : 핵 기본적인 개념
- 분할정복 : 가끔식 출제 되는듯
- Sort : 소팅 알고리즘은 퀵소트는 필수고 여유있으면 머지, 계수, 기수까지는 공부해보자.
- Heap : 힙도 막상 필요할 때가 있음 스케쥴링 문제라던가 스케쥴링 문제라던가
- Queue, Stack : B형 시험보러 갈껀데 큐 스택 못짜면 반성해야됨

-- 알아두면 좋은 것들

- Trie : 문자열 관련된 문제에서 강력함. 거의 Trie나 해싱이 정해인 정도 Trie에 해싱을 끼얹거나
- LCA(Lowest Common Ancestor) : 디렉토리 구조나 가계도 이런 유형에서 등장 할 만한 기법
- BST(Binary Search Tree) : bbst까지는 너무 어렵고 bst 정도? 사실 힙으로 어느정도 커버가 되는 부분
- Segment Tree : 알아두면 쓸 일이 언젠가 생기지 않을까?
- Sqrt Decompisition : 개념은 쉽고 유용한데 쓸 일이 많은 유형인지는 잘 모르겠다

B형(Pro) 유형 파악

배경지식을 충분히 공부했다면 B형(Pro) 문제가 어떤식으로 나오는지 유형을 파악해보자. 먼저 SWEA 사이트에서 [블록 부품 맞추기] 라는 문제를 검색해보자.

이 문제를 읽어보면 알 수 있듯이 우리는 주어진 문제에 맞게 어떠한 함수를 1개 이상 만드는 것이 문제의 목표이고 main 함수를 임의로 수정할 수 없다. 우리가 제출한 코드는 어떠한 함수의 호출 횟수를 최소화 하거나, 문제에서 정의한 스코어 산정 방식으로 다양하게 점수가 계산된다.

[블록 부품 맞추기] 문제를 4시간을 맞춰 풀어 보도록 하자.

이 문제를 잘 읽어보면 3만개의 블록들을 절약비용의 합을 최대화하도록 2개씩 매칭 시키는 문제이다. 곰곰히 생각해보면 어떤 한 모양에 대해서 이 모양과 매칭되는 모양도 정확히 한 개 존재하고 그렇다면 해당하는 모양끼리 가장 긴(최대 기둥 높이) 모양부터 서로 매칭 시켜주면 된다는 것을 알 수있다. [여기까지가 풀이에 대한 기초적 접근]

먼저 주어진 블록을 길이(최대 기둥 높이) 내림차순으로 정렬하고 해당하는 i번째 블록에 대해 [이것에 대응되는 상대편 블록모양을 찾아야함], [해당하는 블록모양중 가장 긴 길이의 블록을 구해야함] 이 2가지 테스크를 해결해야 한다. 그리고 찾은 2 블록을 서로 매칭시켜주면 해결된다.

[대응되는 상대편 블록모양 찾기] 주어진 블록에서 base를 빼면 [0,2] 값을 가지는 4x4 행렬로 바뀌고 이 모양과 결합 될 수 있는 모양은 일단 수직이든 수평이든 한 번 flip한 다음 90도씩 4번 rotate 한 모양 들이다. [2차원 배열 flip과 rotate하는 법도 알아두면 좋다.]

[대응되는 블록모양 중 가장 긴 길이의 블록을 구하기] 를 해결하는 가장 naive 하는 방법은 주어진 블록들을 전부 순회하면서[O(N)] 내가 지금 구한 대응되는 모양과 일치하는지[O(4*4)] 를 하나하나 구하는 방법이다. 이런 방식으로 구현하면 전체 시간복잡도가 대략 O((N^2)*4*(4*4)) 정도 나오게 되는데 너무 느리다!! [나중에 언급할 팁이지만 실제로 Pro 시험장이라면 먼저 이런 시간복잡도를 구현하고 Output이 정확히 나오는걸 확인한 후 최적화 작업을 시작하는 것이 좋다!!] [여기까지 구현한 코드]

가장 먼저 접근 할만한 최적화는 4x4 2차원 배열을 변수 하나로 치환시키는 것이다. 해당하는 배열은 [0,2]값을 가지는 16개의 1차원 배열로 생각할 수 있고 3진수로 접근하면 3^16 가지의 수를 표현하는 것이니 10진수로 [0,3^16) 에 해당하는 값으로 2차원 배열을 치환할 수 있다. 이렇게 하면 2개의 모양이 같은지 확인할 때 단순히 변수의 동등비교로 해결가능하다.

이렇게 우리가 블록모양을 변수 하나로 [해싱]이 가능하다면 대응되는 모양찾기를 더 빠르게 할 접근법이 많아진다.

첫번째 접근법은 또다른 1차원 배열에 {블록모양, 최대 기둥 높이, 인덱스} tuple 로 저장해두고 소팅하면 이후 같은 블록 모양 구간[l, r] 을 이분탐색으로 쉽게 찾을 수 있고 그 중 가장 긴 기둥을 포함하는 블록모양부터 매칭시켜주면 된다.

두번째 접근법은 vector<int> vec[3^16] 과 같은 2차원 동적배열로 O(1)에 해당 벡터에 접근해 가장 긴 기둥을 포함하는 블록모양부터 매칭시켜주면 된다. 여기서 약간의 팁은 애초에 push할 때 작은 블록부터 넣으면 자연스럽게 맨 뒤가 가장 긴 블록으므로 맨 뒤부터 pop해주면 된다. 따라서 vector 가 아닌 list나 stack을 사용해도 무방하다.

세번째 접근법은 3^16이 너무 커서 vector<int> vec[3^16] 선언되지 않을 때 우리는 map<int,vector<int> > 같은 자료구조를 만들면 좋겠지만 map을 구현하는 것이 너무나도 어려울 때 사용하는 방법이다. 비교적 간단한 방법이고 흔히 [좌표압축] 이라고 알려진 웰노운인데 주어진 데이터의 개수가 총 N개이고 데이터의 범위가 [0,3^16) 인 상황이므로 결국 주어진 데이터들을 다시 넘버링을 하면 최악 [0,N) 범위안에서 다시 넘버링 할 수 있음을 알 수 있다. 이러한 방식으로 다시 넘버링 하고 결국 vector<int> vec[N] 에 해당하는 공간만 선언해 준뒤 해당하는 위치를 이분탐색을 통해 O(logN)에 찾고 두번째 접근법과 같은 방식으로 해결 할 수 있다.

위 같은 상황에서 할 수 있는 네번째 접근법은 [0,3^16) 에 해당하는 수를 또다시 해싱해서 충분히 전역변수로 선언할 만한 크기를 만들고, 같은 방식으로 구현하면 해당하는 위치를 찾는데 평균(1)에 찾을 수 있고 위와 같은 방식으로 문제를 해결 할 수 있다.

나는 세번째 방법으로 구현하였고, 내가 구현한 코드는 여기에 올려두었다.. 실제 제출 할 때 sort와 vector는 내 깃허브에 있는 코드를 그냥 복붙했고 1300ms 의 시간으로 통과했다.

이 문제에 대한 개인적인 B형(Pro) 등급 통과 기준은 2000ms ~ 3000ms 라고 생각한다.

B형(Pro)이 어떤 방식으로 출제되고 어떻게 평가하는지 이해가 되는가? 정리를 해보자면 [단순히 문제의 조건에 맞게 구현하면 통과 가능했던 A형]과 달리 B형 에서는 다양한 방법으로 [O(n) 복잡도의 코드를 O(logn) 혹은 O(1) 로 최적화 하는 과정]이 필요하다. 그리고 대다수의 B형 문제들이 상당한 [구현력] 을 요구하고 있다.

이 문제를 4시간 안에 풀이조차 생각하지 못했다면 자료구조에 대한 응용력 혹은 문제를 접근하는 방법 자체가 많이 부족하다. 이 부분에 초점을 맞춰 우선 공부하고 만약 대략적인 구조는 설계했으나 구현을 완벽히 못했다면 구현력을 충분히 연습하는게 좋다.

풀어볼 만한 문제들

기본적으로 탄탄한 구현력이 기반이 되어야 다양한 접근법들을 시도해보고 깨질 수 있다. 구현력을 조지는 문제들을 많이 풀어보는게 도움이 된다. 물론 해당하는 문제를 하루종일 세월아~~~ 내ㅜ월으러어얼아 ~~ 푸는게 아니라 정확히 시간을 정하고 딱 풀어야한다.

** 추천 받습니다. (B형(Pro)와 유사하거나 구현량이 으마으마한 문제들)

[BOJ 고스택 3425]

[BOJ 큐빙 5373]

[oj.uz 보물 찾기] ** 아이디어 문제

[oj.uz cmp] ** 아이디어 문제

[BOJ Yut Nori 15778]

[BOJ Piet 15949]

[BOJ QR 2680]

[BOJ 그날의 너 15827]

[BOJ Iceberg Orders 11743] ** 아이디어 + 구현 , 어려움

실전 B형(Pro) Tip!!

평소에 삼성전자 소프트웨어 역량테스트(공채)를 보는 지인들에게는 항상 문제를 30분 이해 및 설계하고, 30분 코딩하고, 30분 디버깅해서 총 3시간 동안 2문제를 풀어라 해당시간이 넘어가면 화장실을 가던가, 다른 문제를 보라고 조언을 하는 편이다.

마찬가지로 B형 시험에 있어서 세워야할 계획은 [1시간 문제이해 및 설계, 2시간 구현 및 디버깅으로 정확한 아웃풋 출력, 1시간 최적화] 이다.

먼저 문제를 꼼꼼히 이해하고 최대한 Naive한 방법부터 접근해 보는 것이 좋다. 처음부터 가능한 복잡도까지 생각하기보다 비교적 쉽게 가능한 복잡도를 먼저 생각하고 여기서 더 최적화할 방법이 존재하는지 1시간 정도 까지는 충분히 고민해본다. 문제를 난독증으로 잘못 읽을 수도 있으니 1시간쯤지나면 기분전환으루다가 화장실 한 번 다녀오고(만약 인개원이라면 화장실이 일단 기분전환 엄청됨 ㅇㅇ), 문제를 다시 꼼꼼히 읽은 뒤 지금까지 생각한 뒤 가장 좋은 방법으로 구현을 시작하자!

충분한 시간동안 구현을 마치고 문제에서 요구하는 아웃풋이 정확하게 나왔다면 이제 충분한 시간을 가지고 최적화를 시도해보자. 블록 부품 맞추기를 할 때와 같이 무언가를 전처리 한다거나 정렬을 해서 이분탐색을 한다거나 해싱을 해서 어떤것을 좀더 빨리 찾고 빨리 접근 한다는 식이라서 찾고 바꾸고 계산하는 것들은 [전처리, 메모이제이션, 이분탐색, 해싱] 선에서 거의 다 끝난다. 가으으아으끔 [트라이, LCA, 세그먼트 트리] 를 사용하는 정도의 최적화도 필요할 수 있다.

이렇게 Naive 구현과 최적화부분을 나누어서 코딩을 하게 되면 [디버깅 하는 시간] 도 그만큼 줄어들게 된다. 당연하게도 구현부분에서 이미 정확한 Output이 나왔음이 보장되기 때문에 최적화 부분에서 뭘 바꿧는데 이제 안나오는건지 집중해서 디버깅 할 수 있기 때문이다.

이와 같은 이유로 평소에 STL을 많이 사용하는 STL충들은~~[벌레충이 아니라 충성할 충]~~ 로컬에서는 먼저 STL을 사용해서 구현을 완료하고 Output이 보장되면 그 때 남은 STL들을 구현하는 방식으로 코딩하는 것이 디버깅하기 쉽다.

지인들 중에서는 [Output이 나왔다고 그냥 제출하고 바로 퇴실하는 경우] 도 있었다. 물론 탈락이다. 맨 처음부터 합격선까지 최적화 했을 수도 있지만 본인이 그정도 클라쑤가 아니라고 생각된다면 내가 잘 풀었으면 다른사람들도 잘 풀었을 것이다 라고 생각하고, 남은 시간동안 정말 여러방면으로 생각해서 최적화를 시도해보도록 하자.

4시간 동안 잘 구현하고 충분히 최적화 했다면 좋은 결과가 있기를 기원하도록 하자~ 내가 좋아하는 말중에 [진인사대천명]이라는 말이 있는데 충분히 노력했드아아면 결과는 좋게 나올것이다!!

다양한 기법들

사실 본문에서 블록 부품 맞추기를 풀면서 간단하게 이용했던 4x4 2차원 배열을 변수로 치환하는 기법이나 좌표압축 기법은 사실 사용해 보지 않았다면 시험장에서 바로 떠올리기엔 어려운 기법들이다.

이 챕터에서는 B형(Pro) 등급에서 사용될 만한 다양한 기법들을 정리하겠다.

** 생각나는대로 추가하겠읍니다.

[차원 압축 : 모든 데이터는 (해당하는 데이터를 표현하는데 필요한 총 비트 수) / (sizeof(int) or sizeof(long long))로 압축가능 합니다.]

EX) 2차원 그리드에서 2x2 영역을 1x1 크기로 압축, 길이가 10인 1차원 배열을 길이 1로 압축 ....

[좌표 압축 : 데이터의 범위가 너무 크다면 등장하는 데이터들을 [0,N) 범위로 다시 넘버링 할 수 있습니다.]

[랜덤 엑세스 링크드 리스트 : 링크드 리스트의 단점이 랜덤 엑세스가 불가능하다는 것인데 해당하는 노드를 특정할 수 있고(id 라던지 value가 distinct 하다던지) 그 값이 충분히 작다면 링크드 리스트에 존재하는 모든 노드에 대해 해당하는 key를 전역변수로 노드 포인터를 저장하여 O(1)로 해당하는 노드에 접근 할 수 있습니다.]

EX) 랜덤 엑세스 + 중간 삽입, 삭제가 O(1)로 자유로운 자료구조 , BUT 해당하는 key의 값의 범위가 전역변수로 선언할 만큼 충분히 작아야함.

[이분탐색 + 휴리스틱 : 데이터가 유니폼하게 분포되어 있는 상황에서는 정확한 이분탐색이 가장 효율적일 것입니다. 하지만 데이터를 분석해봤을 때 유니폼하지 않고 특정값에 몰려있다면 확률적으로 해당부분을 포함하게 적절히 쪼개는 것이 좀 더 효율적임을 알 수 있읍니다.]