Make Fossil 정리 및 풀이

안녕하세요! Make Fossil 스터디를 연 박홍빈이라고 합니다. 이런 PS류의 공부는 처음이라 익숙하지 않았음에도 불구하고 다들 잘 따라와 주셔서 정말 감사합니다. 이 문서를 읽는다는 것부터 굉장히 열심히, 또 잘 한다는 증거이니 계속 열심히 공부하여 저 같은 사람보다 더 잘하여 그때는 오히려 저를 멱살 잡고 끌어올려주는 사람이 될 수 있기를 빕니다.

(PS : problem solving의 약자, 알고리즘 문제를 푸는 것을 말합니다.)

이 문서는 앞으로 스터디가 진행 될 때마다 그 전 스터디 내용을 복습할 수 있도록, 또 문제 풀이를 알고 싶거나 그 풀이를 코드로 어떻게 구현해야 할 지 막막할 때 참고할 수 있도록 만든 문서입니다. 계속 업데이트 및 수정, 첨언이 들어갈 예정이니 피드백은 물론 환영이고, 질의나 메모가 필요하다고 생각되신다면 이 문서를 바로 수정해 주시면 됩니다! 그럼 바로 시작하겠습니다!

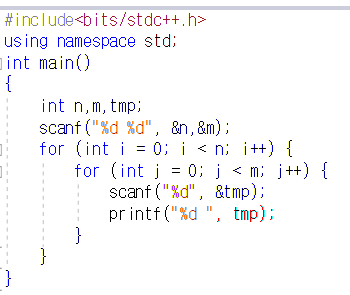
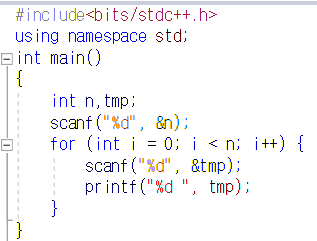
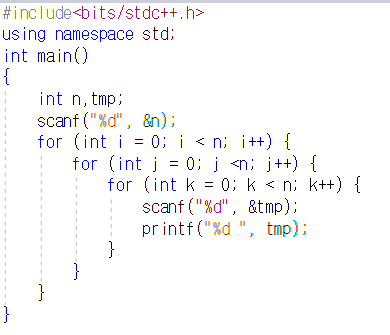
Day1

첫 수업에서는 시간복잡도와 STL에 대해 다루어 보았습니다. 시간복잡도란 “프로그램이 최악의 상황에서 어느정도로 느리게 도는가?”를 보여주는 척도라고 생각하면 됩니다. 예를 들어보겠습니다.

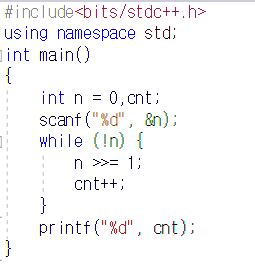
어떤 프로그램이 무슨 수 n를 입력 받던 1부터 1억까지 출력한 후 1부터 n까지 출력하는 프로그램이 있다고 생각해봅시다. 이 경우 n이 10이나 100처럼 작다면 프로그램의 시간은 1~1억까지 출력하는데에 있어 대부분의 시간을 쓸 것입니다. 하지만 만약 n이 굉장히 크다면??? n이 10억, 100억 정도로 매우 크다면 반대로 1부터 n까지 출력하는데에 프로그램의 대부분의 시간이 쓰일 것입니다. 이렇게 우리는 극단적인 상황을 생각해 봐야 합니다. 지금 예시의 경우처럼 n이 굉장히 클 때가 최악의 상황입니다. 이 경우 앞에 1억번 출력하는 시간은 무시해도 될 정도가 될 것이고 이 경우 n개를 출력하는 것이 가장 중요한 경우가 될 것입니다.

이 것처럼 입력이나 혹은 다른 변수에 따라 프로그램의 시간이 결정나는 경우 그 입력이나 다른 변수로 프로그램의 작동 시간을 말하는 것이 시간복잡도 중 빅-오(big - O) 표현법입니다. 정의는 다음과 같습니다.

이 정의에 대해서는 나중에 수학적 감을 통해 알 수 있겠지만 지금은 이런 복잡한 정의는 ‘이해가 되면 좋다’ 정도입니다. 정의 없이 경험적으로 이해가 될 수 있도록 시간복잡도의 예제를 살펴봅시다.

각각의 프로그램들을 봅시다. 각 프로그램들은 입력 n을 받고 그 n을 이용하여 for문을 돕니다. 이제 우리가 신경써야 할 것은 n이 커질 때 프로그램의 작동 시간이 어떻게 바뀌느냐 하는 것입니다. 첫 번째 프로그램의 경우 n이 커지던 말던 main 함수 밑의 2 줄은 똑같이 작동합니다. 하지만 for문의 경우 n이 커지면 그 만큼 더 많은 숫자들을 입력받고 출력합니다. 즉 이 경우 시간복잡도는 이 됩니다. 두 번째 프로그램의 경우는 i, j가 0부터 n-1까지 돌면서 입력을 받습니다. 이 경우는 이며 마지막 프로그램의 경우 입니다.

모든 프로그램의 시간복잡도가 위 프로그램들처럼 단순하면 좋을텐데 아쉽게도 항상 이렇게 쉬운 경우만 있지는 않습니다. 다음의 경우를 보겠습니다.



이 프로그램의 경우 언뜻 봐서는 시간복잡도를 알아내기 힘들어 보입니다. 답부터 말하자면 이 경우의 시간복잡도는 입니다. 그 이유는 n>>=1에 있습니다. n>>=1은 n이 1/2씩 감소하게 됩니다. 1씩 감소하는 것보다 매우 빠르게 감소하며 이 감소 추세를 알아내면 시간복잡도를 찾아낼 수 있을 것입니다. n이 일반적인 상황에서는 얼마나 빨리 감소할지 감이 잘 잡히지 않으므로 2의 거듭제곱에서의 상황을 생각해봅시당! n이 이라면 while문을 6번 돌면 그 뒤에는 n의 값이 1이 되고 그 다음에는 0이 되므로 while문은 7번 돌것입니다. n이 이라면 한 번 더 많은 8번 돌겠지요. 이때 64~128 사이의 숫자는 64보다는 많이 128보다는 적게 돌 것이 확실하니 7~8번 정도 돌 것이라고 예상할 수 있습니다. 이 숫자는 n이 커지면 n이 꼭 2의 거듭제곱이 아니더라도 으로 근사할 수 있습니다. 따라서 이 프로그램의 시간복잡도는 입니다.

밑의 표는 많이 나오는 시간복잡도를 n에 대해 후순위부터 적어놓은것 입니다. 프로그램이 나누어져 있어 시간복잡도가 여러 개라면 밑의 것들 중 가장 아래에 있는 한 가지 항만 적으면 됩니다.(ex. 이 같이 있다면 총 시간복잡도는 입니다.)

|  |  |
| --- | --- |
|  | 입력값과 상관 없이 걸리는 시간이 일정할 때 입니다. 문제의 답이 항상 YES이거나 NO일 때 이 시복입니다. |
|  | 위의 프로그램이 대표적 예시입니다. |
|  | PS를 하다 보면 어려운 문제일수록 이 시복인 경우가 많습니다. 어려운 풀이를 생각하기 전에 간단한 풀이를 생각해 봅시다!! |
|  | 정렬 알고리즘이나 분할정복 등에서 흔히 쓰이며 대부분의 PS 문제 시복일 확률이 많습니다. |
|  | PS를 하며 쉽게 보이는 시복은 아니나 충분히 고려해 봐야 합니다. |
|  | 구현이 어렵거나 값을 효율적으로 줄이는게 문제인 경우가 많습니다. |
|  | 사실상 정풀의 시복인 경우는 없으며 가장 낮은 부분점수의 시복인 경우가 많습니다. |

PS를 할 때, 안에 들어가는 값이 정말 간단한 연산(비트연산)인 경우 2~5억정도, 그렇지 않다면 1억 이하정도로 되어야 1초 안에 실행이 된다고 생각하시면 편합니다. 이제 저희는 드디어 PS를 접할 준비가 된 겁니다. 이제 PS에 본격적으로 입문할 일만 남았습니다.

PS에 입문을 했다면 가장 먼저 배워야 할 것은 자료구조와 알고리즘입니다. 우선 이 두 가지가 무엇인지를 알아야합니다. PS를 할 때는 데이터를 수십만개에서 많게는 수백만개를 다루어야 합니다. 이 때, 원하는 작업을 효율적으로 하기 위해서 데이터들을 어떤 형식에 맞추어 잘 보관하고 있어야 합니다. 그렇게 자료들을 효율적으로 관리하기 위한 형식이 바로 자료구조입니다. 알고리즘은 이렇게 자료구조로 관리하는 데이터들을 이용하여 원하는 결과를 만들어내는 효율적인 절차입니다. 이제 자주 사용하는 자료구조와 알고리즘에는 무엇이 있는지 알아보겠습니다.

1. Stack

첫 번째는 스택입니다. 다들 ~ 몇 스택 적립! 같은 말을 한 번쯤은 들어봤을 것 입니다. 스택이라는 것은 데이터를 쌓는 자료구조입니다. 쉽게 이미지화하기 위해서는 문이 굉장히 좁은 엘리베이터를 생각하시면 될 것 같습니다. 엘리베이터 안에 사람이 많아지면 먼저 탄 사람은 문을 통해 나가는 것이 힘들어 나갈 수 없고 나중에 탄 사람이 먼저 내리게 될 것입니다. Stack도 마찬가지입니다. 데이터들을 일렬로 세우는데 가장 마지막에 추가된 녀석이 가장 먼저 나가게 되며 우리가 볼 수 있는 값 역시 현재 stack에 있는 데이터들 중 가장 마지막에 추가된 데이터 뿐 입니다.

1. Queue

Queue의 경우는 stack과 반대입니다. queue에서 볼 수 있는 값은 queue 안에 있는 값들 중 가장 먼저 들어간 값 뿐 입니다. 또 queue에서 빼는 것 역시 queue에서 가장 먼저 들어간 값만 가능합니다. 물건을 구매할 때 줄은 맨 뒤에서 추가되지만 맨 앞 자리만 구매가 가능하고 줄이 주는 것 역시 맨 앞에서 주는 것만 가능한 것을 생각하시면 이해가 빠르실 것 같습니다.

1. Deque

Deque는 stack과 queue를 섞어놓았다고 생각하면 편합니다. 줄의 양 끝에서 삽입, 삭제, 값값 가져오는 것 모두 가능합니다.

1. Priority\_queue

여기서부터 어려운 자료구조입니다. Heap 자료구조 중 한 가지이며 데이터들 중 가장 큰 값(혹은 가장 작은 값)을 찾고싶을 때 사용합니다. 이 자료구조의 구현에 대해서는 다음에 트리를 배운 후 한 번 짚도록 하겠습니다.

이 외에도 더 많은 자료구조가 존재합니다만 (segment tree, set, map 등등)이런 자료구조는 잘 쓰이지 않거나 내용이 복잡하고 많기 때문에 밑에 stl 정리표를 참고하시거나 나중에 스터디에서 중점적으로 다루겠습니다.

알고리즘에는 굉장히 많은 종류가 있지만 기본적으로 알아야할 알고리즘들 중 stl에 있는 알고리즘을 중점적으로 본 뒤 나중에 스터디를 진행하며 다른 알고리즘들을 차례차례 배우도록 하겠습니다.

1. Sort

Sort는 값들을 특정 기준으로 정렬하는 것을 말합니다. 기본적으로는 오름차순 정렬이 일반적이며 정렬 방법은 매우 많습니다만 보통은 quick sort를 기반으로 정렬 알고리즘을 사용합니다. 여러 정렬 알고리즘에 대해서는 나중에 따른 문서로 만들어 보겠습니다. 다들 꼭 한 번 확인해서 알고리즘적 사고를 키워볼 수 있도록 합시다!!

1. Lower\_bound, upper\_bound

Lower\_bound, upper\_bound는 이미 정렬되어 있는 값들 중 상한과 하한을 찾는 함수입니다. 이분탐색을 이용하여 상한과 하한을 찾으며 stl에 구현이 되어 있으나 사실 이분탐색정도는 직접 구현하는 것이 더 효율적일 수 있습니다. (stl에서의 lower\_bound, upper\_bound을 원하는 방식에 끼워 맞추는게 더 까다로운 경우가 많습니다.)

이 밑에는 c++의 stl(standard template library)에 있는 자료구조와 알고리즘의 대략적 설명과 관련 설명이 있는 사이트의 링크입니다. 보고 익숙해지면 코딩할 때 편하나 무조건 외우진 마시고 문제를 풀 때마다 계속 찾아쓰다보면 언젠가 익숙해질 겁니다. 따라서 어려운 문제 1, 2개를 푸는 것보다 쉬운 문제들을 풀면서 사용에 익숙해지도록 합시다.

|  |  |  |
| --- | --- | --- |
| 이름 | 설명 | 링크 |
| vector | 동적 배열 | <http://www.cplusplus.com/reference/vector/vector/?kw=vector> |
| stack | 위 내용 참고 | <http://www.cplusplus.com/reference/stack/stack/?kw=stack> |
| queue | 위 내용 참고 | <http://www.cplusplus.com/reference/queue/queue/?kw=queue> |
| deque | 위 내용 참고 | <http://www.cplusplus.com/reference/deque/deque/?kw=deque> |
| Priority\_queue | 위 내용 참고 | <http://www.cplusplus.com/reference/queue/priority_queue/?kw=priority_queue> |
| set | 값이 들어올 때마다 정렬됨(값이 중복X, 중복이 필요할때는 multiset) | <http://www.cplusplus.com/reference/set/set/?kw=set> |
| map | Set과 비슷하나 값을 두 개씩 (key, value) 집어넣음. key값을 알면 value값을 찾을 수 있음 | <http://www.cplusplus.com/reference/map/map/?kw=map> |
| sort | 위 내용 참고 | <http://www.cplusplus.com/reference/algorithm/sort/?kw=sort> |
| Lower\_bound | 위 내용 참고 | <http://www.cplusplus.com/reference/algorithm/lower_bound/?kw=lower_bound> |
| Upper\_bound | 위 내용 참고 | <http://www.cplusplus.com/reference/algorithm/upper_bound/?kw=upper_bound> |

이 외에도 코딩을 하다가 자주 쓰일 것 같은 기능이나 필요한 기능들은 stl에 구현되어 있는 경우가 많으니 자주 찾아보는 습관을 들입시다! 하지만 본인이 구현하는게 더 빠르다고 생각되면 구현하시는 것도 좋은 방법입니다!

첫 날에는 PS에 입문하기 위한 준비를 하였고 그에 따른 준비문제들 4문제와 알고리즘 적 사고가 필요한 문제 2 문제를 출제하였습니다. 이에 관해서는 problem and solve 폴더를 참고해주세요.

Day2

PS를 입문할 준비가 모두 되었기 때문에 두 번째 날에는 PS를 맛보았습니다. 앞으로도 계속 많이 쓰이는 알고리즘인 완전탐색, DP, greedy, parametric search를 배워보았습니다. 각각 개념이 확실히 잡히는 것이 중요하기 때문에 다시 한 번 살펴보도록 합시다.