2020년
ALPS 여름방학
비대면 스터디

- 2주차 -

문제풀이

참여방법

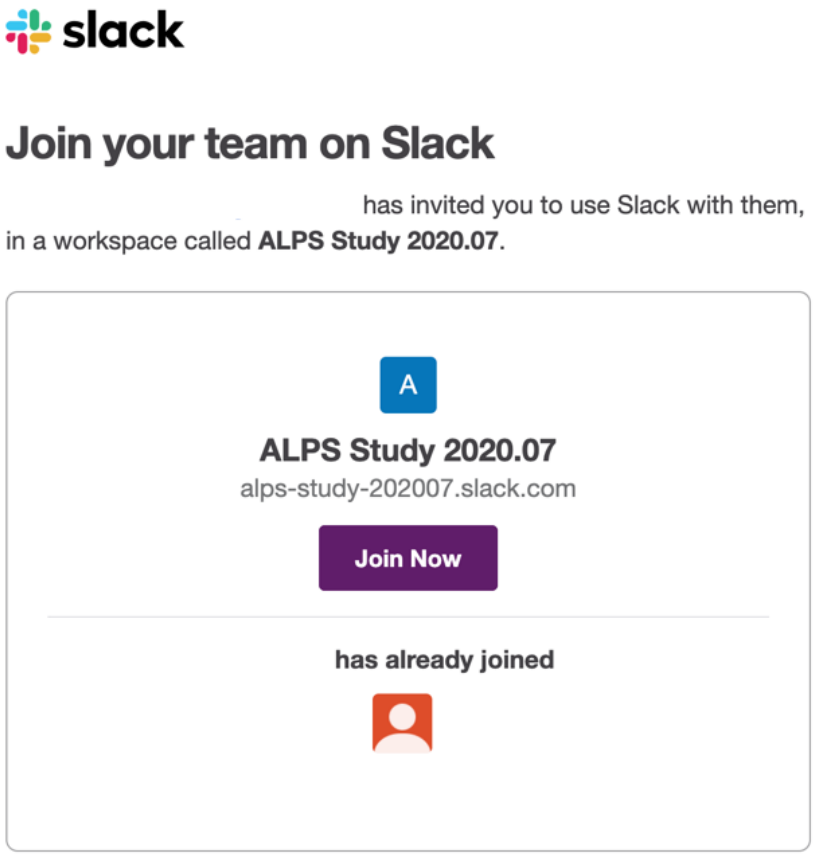
- 슬랙 - 코드 공유용으로 사용
- 구글 meet - 화면공유 및 음성으로 코드 설명 및 서로간 피드백

(https://meet.google.com/exm-dcvh-gcr) 링크는 계속 고정입니다.

항상 위 링크로 들어오시면 됩니다.

슬랙과 meet를 같이 띄우고 진행해주시면 감사하겠습니다.

참여방법 / 슬랙



메일로 보내드린
초대장에 Join Now를
클릭하시면
입장 가능합니다

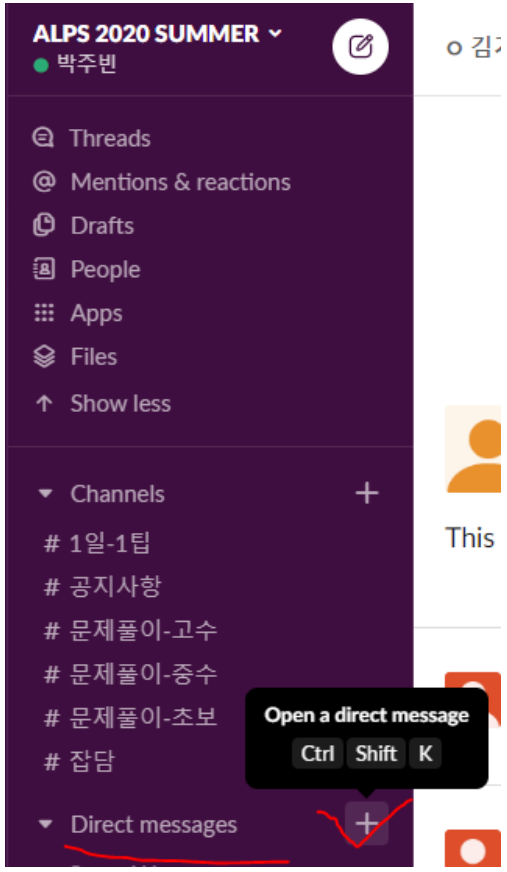
참여방법/구글 meet

- https://meet.google.com/exm-dcvh-gcr
- 해당 링크로 접속
- 해당 스터디 진행영상은 참여하지 못하신 분들을 위해 녹화 후 유튜브 채널에 업로드할 예정입니다^^
- 혹시 녹화를 원하지 않는 분들은 말씀해주세요.
- 아래는 영상이 올라갈 주소 입니다.
- https://www.youtube.com/playlist?list=PL9gVcwpebJSJJ80vNpdrYAhyG6PXxbgq_

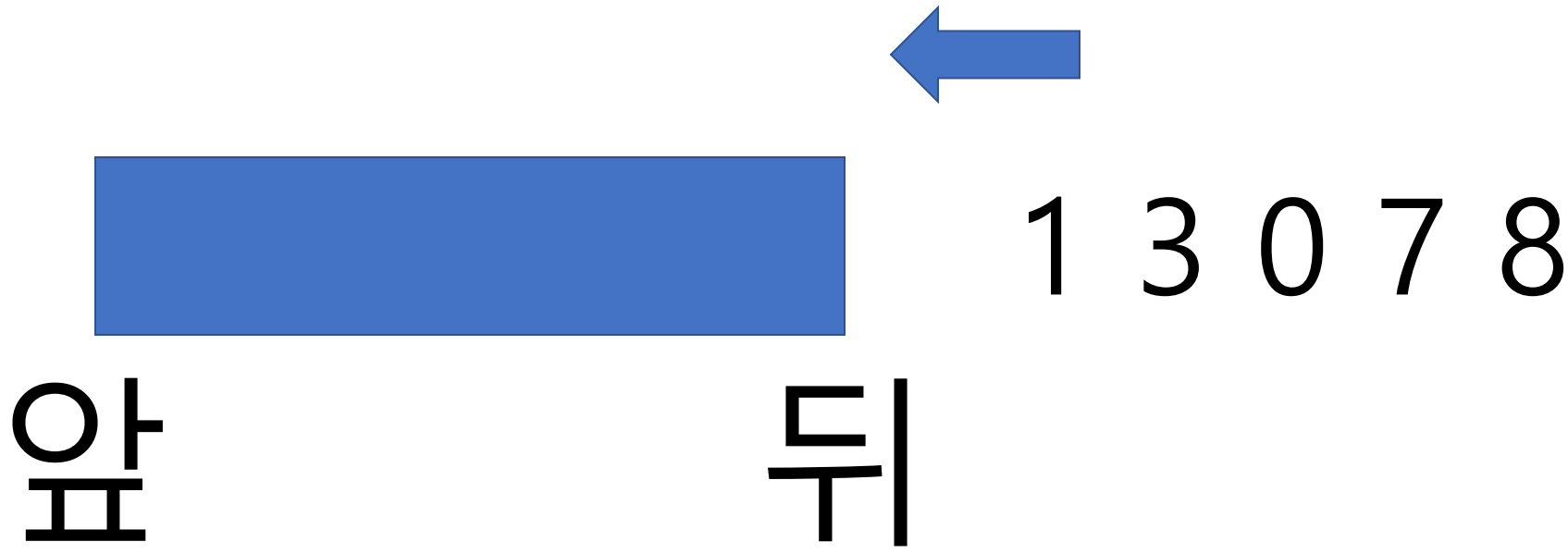
시작하기 전에..

- 먼저 **50분** 동안 모두에게 문제 푸는 시간이 주어집니다.
  - 만약 궁금하신 문제나 풀리지 않는 문제가 있다면
  - 질문을 하시되, **익명**으로도 가능하니 스터디 운영진에게 **슬랙의 Direct Message**를 통해 알려주세요.

- 나머지 **1시간**에는 풀이 설명을 진행할 것입니다.
  - 시청은 자유이며, 여러분이 **해당 문제**의 설명을 듣지 않아도 된다고 판단하시면 그 시간에 다른 문제를 푸셔도 좋습니다!



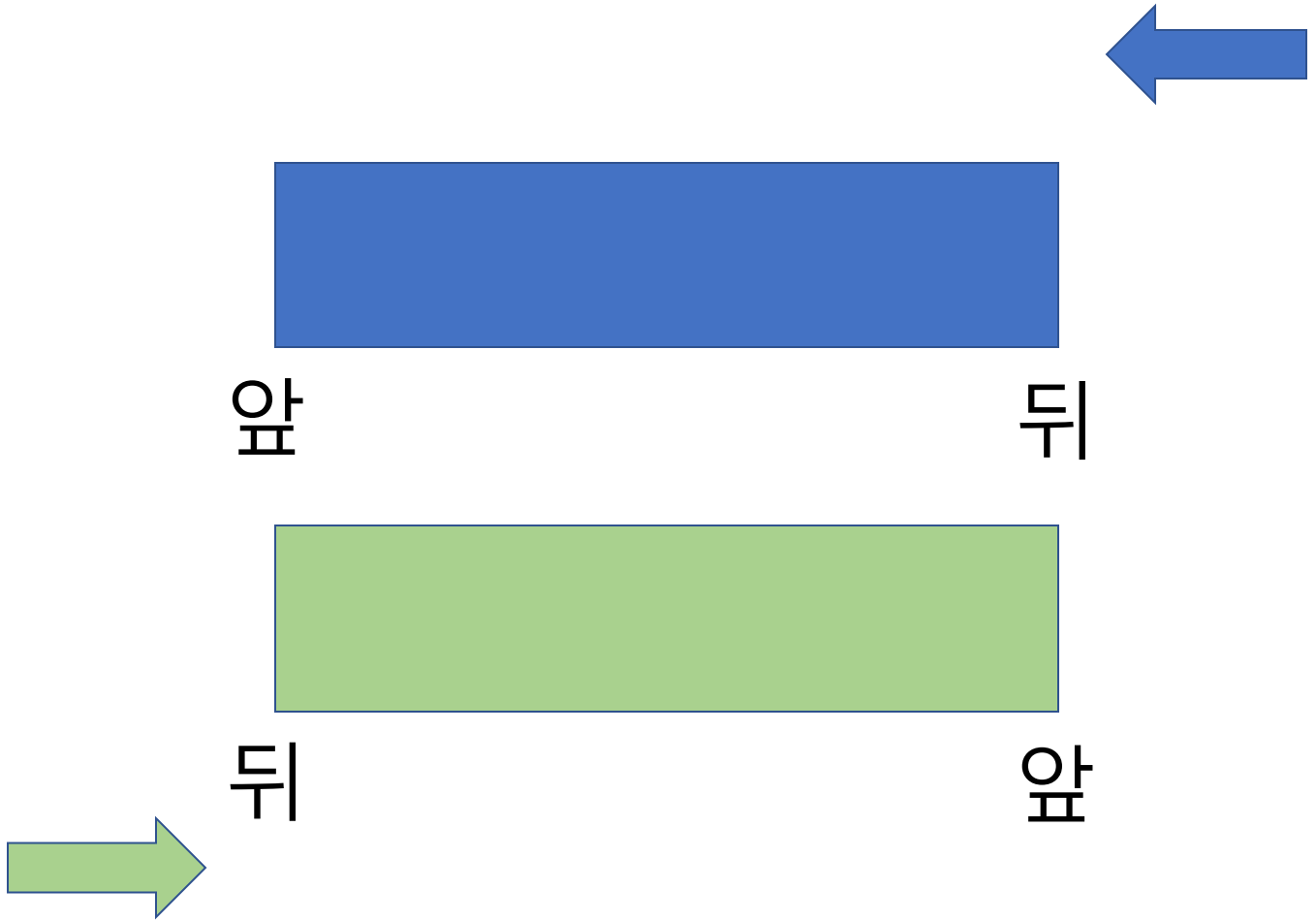
queue



dequeue? deque? 덱? 디큐?

double-**e**nded **queue**

여기 두 개의 큐가 있습니다.
이걸 하나로 합치면..



데이터를 양쪽으로 삽입/제거 가능한 dequeue 완성!

BOJ-10866 덱 (난이도:하)

https://www.acmicpc.net/problem/10866

BOJ-10866 덱

- C++의 std::deque 또는 std::list 라이브러리를 사용하세요.
- 둘 다 아래 함수가 구현되어 있으니 그대로 적용하면 됩니다!
  - push_front
  - push_back
  - pop_front
  - pop_back
  - size
  - empty
  - front
  - back

BOJ-10866 덱

- std::deque 사용

https://github.com/lee20h/ALPS_2020_Summer_Study/blob/master/Jubin/week2/boj-10866-try1.cpp

- std::list 사용

https://github.com/lee20h/ALPS_2020_Summer_Study/blob/master/Jubin/week2/boj-10866-try2.cpp

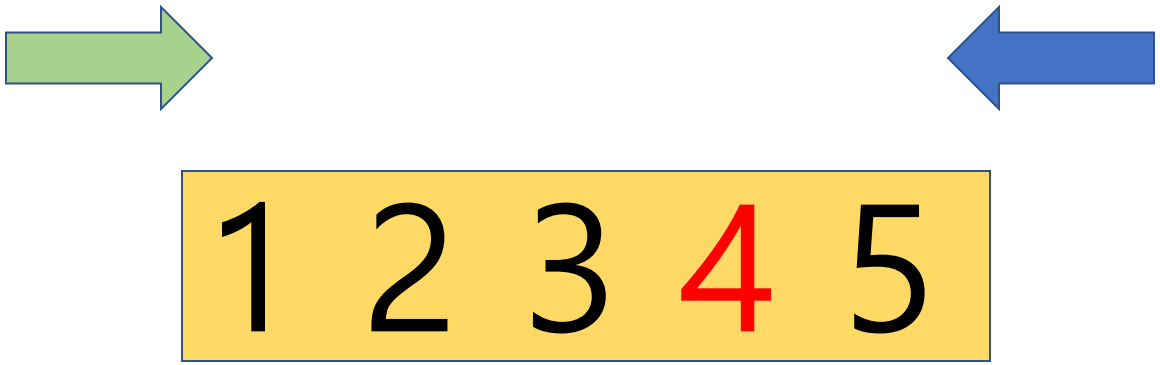
BOJ-1021 회전하는 큐 (난이도:하)

https://www.acmicpc.net/problem/1021

BOJ-1021 회전하는 큐

- { 1, 2, 3, … , N-1, N } 수열을 순서대로 나열합니다.
- 1번째 연산에 해당하는, 큐의 front만을 뽑아야 합니다.
- 따라서 2, 3번째 연산으로 원하는 숫자를 front로 옮겨야 합니다.
- (2번째 + 3번째 연산 횟수의 최소값)을 구하는 것이 목표
- 세 가지 연산 명세를 봤을 때 **이 자료구조를 써야겠다!** 라는 감이 오시나요?

BOJ-1021 회전하는 큐



- Q1) 몇 번의 pop_front를 해야 4를 얻을 수 있나요?
- Q2) 몇 번의 pop_back를 해야 4를 얻을 수 있나요?

BOJ-1021 회전하는 큐

- 눈치채셨나요?
- (Q1 답)과 (Q2 답)을 더하면 (덱의 사이즈)가 나온다는 것을..
- (Q1 답)과 (덱의 사이즈)만 알아도, (Q2 답)을 구할 수 있겠죠?

- 최종적으로 min(Q1 답, Q2 답)을 얻을 수 있습니다!

BOJ-1021 회전하는 큐

- https://github.com/lee20h/ALPS_2020_Summer_Study/blob/master/Jubin/week2/boj-1021.cpp

BOJ-5430 AC
(난이도:중)

https://www.acmicpc.net/problem/5430

BOJ-5430 AC

단순하게 **주먹구구식**으로 생각합시다.

- R 함수를 실행하면 배열을 뒤집고
- D 함수를 실행하면 첫 번째 원소를 지운다!

- 하지만 뭔가 불편하다..

BOJ-5430 AC

- 배열의 길이가 100,000 이고, R을 1번 호출한다면..
- 100,000 번 모두 R로 호출한다면..

- 시간 초과!

BOJ-5430 AC

- 배열을 직접 뒤집지 말자! 덱을 사용해볼까요?

- 함수 R을 실행할 때
  - bool 형 변수를 만들어서 뒤집힌 상태를 저장

- 함수 D를 실행할 때,
  - 뒤집혔으면, pop_front 를 호출하고
  - 안 뒤집혔으면, pop_back 를 호출

BOJ-5430 AC

- n 입력 후 엔터를 쳤다!
  - scanf 또는 cin을 썼다면, '₩n' 값은 아직 입력되지 않고 버퍼에 남음
  - getchar() 또는 cin.get() 로 '₩n'을 반환 받고 그대로 흘려버리세요!
- 그 다음 '[' 가 들어오는데, 이걸 어떻게 처리할까요?
  - 그 한 줄이 끝날 때까지 한 문자씩 얻어오세요.
  - 숫자, 콤마,], ₩n, EOF 인 경우를 나누어서 생각하세요.

BOJ-5430 AC

…

3 ₩n

[114,2,31] ₩n

…

…

2 ₩n

[4,30] EOF

BOJ-5430 AC

- https://github.com/lee20h/ALPS_2020_Summer_Study/blob/master/Jubin/week2/boj-5430.cpp

BOJ-3078 좋은 친구 (난이도:중)

https://www.acmicpc.net/problem/3078

BOJ-3078 좋은 친구

- 큐를 통해 작업한다.
- 큐의 크기는 어떻게 유지 되어야 할지 고민해본다.
- 나와 같은 이름의 친구들 어떻게 ans에 더해갈지 생각해본다.

BOJ-3078 좋은 친구

- 큐는 K크기만큼 유지한다.
- 현재 큐에 들어있는 친구들의 이름길이는 따로 cnt[]배열에 차곡차곡 갯수를 넣어둔다.
- 이때 새로운 값이 들어오면 기존 cnt[]배열에 들어있는 이름길이 갯수를 통해 나의 짝을 결정할 수 있다.
- 그리고 현재 값을 큐에 넣어주고 cnt[현재이름길이]++ 해준다.
- 반면 큐의 size가 K를 초과하게 되면 pop수행 및 cnt[q.top]--

BOJ-3078 좋은 친구

- https://github.com/lee20h/ALPS_2020_Summer_Study/blob/master/KiwanKim/3078.cpp

BOJ-1966 프린터 큐 (난이도:중)

https://www.acmicpc.net/problem/3078

BOJ-1966 프린터 큐

- 최우선 순위의 작업이 나올때 까지 pop, push해준다.
- 중복된 숫자가 나올 수 있기 때문에

목표작업인지 아닌지 판단하는 방법은 어떻게 할지 고민한다.

BOJ-1966 프린터 큐

- queue<int,bool>을 통해 pair<현재 숫자, 목표작업인지 아닌지>사용
- cnt[]배열을 통해 현재 queue에 들어있는 작업을 파악
- cnt[]배열을 통해 가장 높은 우선순위이면 pop과 동시에 ans++

그렇지 않다면 다시 pop과 동시에 다시 push

BOJ-1966 프린터 큐

- https://github.com/lee20h/ALPS_2020_Summer_Study/blob/master/KiwanKim/1966.cpp
- http://melonicedlatte.com/algorithm/2018/02/18/230705.html

BOJ-1655 가운데를 말해요 (난이도:상)

https://www.acmicpc.net/problem/1655

BOJ-1655 가운데를 말해요

- 우선순위 큐를 활용하는 문제
- 서로 다른 두개의 우선순위 큐를 통해 시간을 단축할 수 있다.

BOJ-1655 가운데를 말해요

- left_pq,right_pq를 만듭니다.
- 두 pq는 서로간 균형을 유지하게 되는데 이 과정에서 mid(가운데값)을 구하게 됩니다.
- Left_pq에는 작은 것들 right_pq에는 큰 값들이 들어갑니다.
- 그리고 새로운 값이 들어오면 둘 중 어느 곳에 넣을지 비교하게 되는 이 과정에서 mid(가운데값)을 고려하게 됩니다.
- 이때 right_pq에는 넣고자 하는 value(값)을 -value로 push해주면 가장 작은 value값이 top으로 나오게 되어 연산이 간단해집니다.

BOJ-1655 가운데를 말해요

- https://github.com/lee20h/ALPS_2020_Summer_Study/blob/master/KiwanKim/1655.cpp (제가 쓴 코드)
- https://www.crocus.co.kr/625 (블로그)

다음 3주차에는..

- 그리디 알고리듬을 배울 예정입니다.
- 시청해 주셔서 감사합니다.
- 설문조사에 참여해주세요!