2024 겨울학기 동국대학교SW역량강화캠프

6일차. BFS 1

● 자료구조(Data Structure)

▶ 데이터를 효율적으로 접근하고 수정하기 위한 데이터의 저장 방식

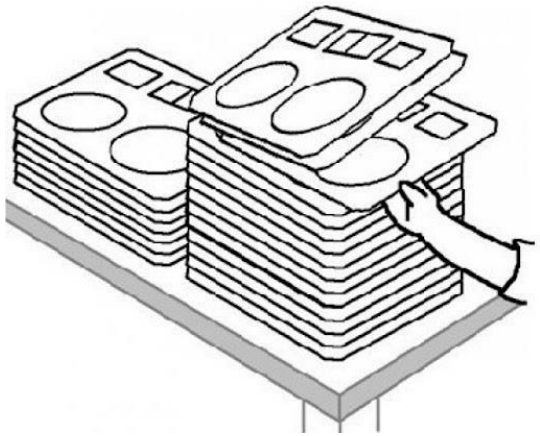
▶ 배열, 리스트, 스택, 큐, 덱, 힙, 트리맵 등등

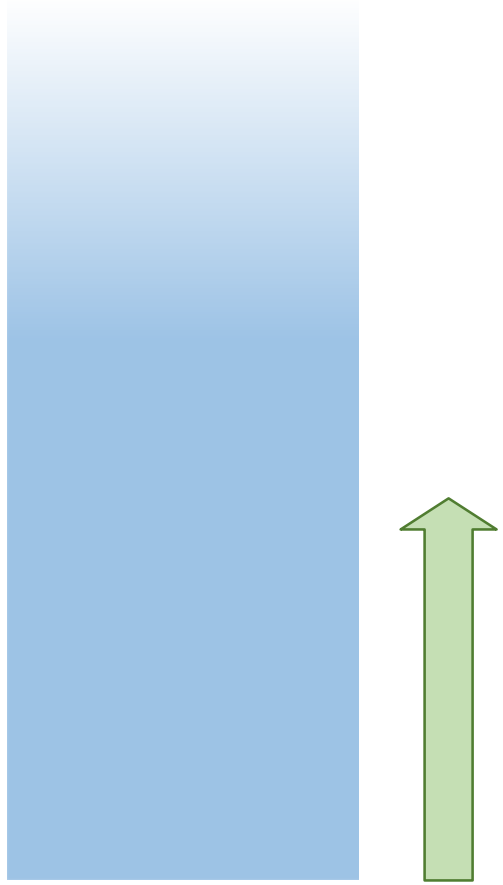
● 스택(Stack)

▶ 접시 쌓기

▶ 1번 → 2번 → 3번 순서로 쌓았다면, 꺼낼 때에는 3번 → 2번 → 1번

▶ FILO(First In Last Out) , LIFO(Last In First Out)





● 스택 (101)

문제

스택을 배운 서영이는 직접 스택 프로그램을 구현하고 싶어졌다.

서영이가 설계한 프로그램의 명령어들은 아래와 같다.

push x : 정수 x를 출력하고, 그 수를 스택에 넣는다. (x는 int형 범위 이내의 수이다.)

pop : 스택에서 가장 위에 있는 정수를 빼고, 그 수를 출력한다. 만약 스택에 들어있는 정수가 없는 경우에는 -1을 출력한다.

size: 스택에 들어있는 정수의 개수를 출력한다.

empty: 스택이 비어있으면 1, 아니면 0을 출력한다.

top: 스택의 가장 위에 있는 정수를 출력한다. 만약 스택에 들어있는 정수가 없는 경우에는 -1을 출력한다.

end : 프로그램을 종료한다.

서영이를 도와 위의 명령어를 수행하는 프로그램을 작성해보자. (유효한 명령어만 주어진다고 한다.)

핵심 코드

▶ Stack 자료구조는 java.util.Stack 에 정의되어 있습니다.

python : from collections import deque 또는 리스트 사용
C++: #include<stack>

```
import java.util.Stack;
```

▶ Stack 자료구조는 다음과 같이 선언할 수 있습니다.

```
Stack<Integer> stack = new Stack<Integer>();
// stack은 이렇게 선언합니다.
// 참조형 변수로만 선언 가능합니다 int(X) char(X) Integer(O) Character(O)
// Stack<object> stack_name = new Stack<object>();
```

핵심 코드

▶ Stack에 들어있는 원소의 수는 size 메소드를 통해 알 수 있습니다.

```
if(str.equals("size")) {
    int x = stack.size(); // 스택에 쌓여있는 원소의 개수를 x에 저장합니다.
    sb.append(x+"\n");
}
```

▶ Stack이 비어 있는 지의 여부는 empty 메소드를 통해 알 수 있습니다.

```
if(str.equals("empty")) {
    Boolean isEmpty = stack.empty(); // isEmpty에 스택이 비어있는지 여부를 저장합니다.
    if(isEmpty) sb.append("1\n");
    else sb.append("0\n");
}
```

핵심 코드

▶ Stack에 값을 쌓는 것은 push 메소드를 이용하면 됩니다. (add 함수도 동일합니다)

```
if(str.equals("push")) {
    int x = Integer.parseInt(st.nextToken());
    stack.push(x); // stack에 x를 쌓습니다.
    sb.append(x+"\n");
}
```

▶ Stack에서 가장 위에 쌓여 있는 값은 peek 메소드를 이용하여 접근할 수 있습니다.

```
if(str.equals("top")) {
    if(stack.empty()) { // 문제 조건에 따라 스택이 비어있다면 -1을 출력합니다.
        sb.append("-1\n");
        continue;
    }
    int x = stack.peek(); // 스택의 맨 위에 쌓인 값을 x에 저장합니다.
    sb.append(x+"\n");
}
```

핵심 코드

▶ Stack에서 값을 빼내는 것은 pop 메소드로 할 수 있습니다.

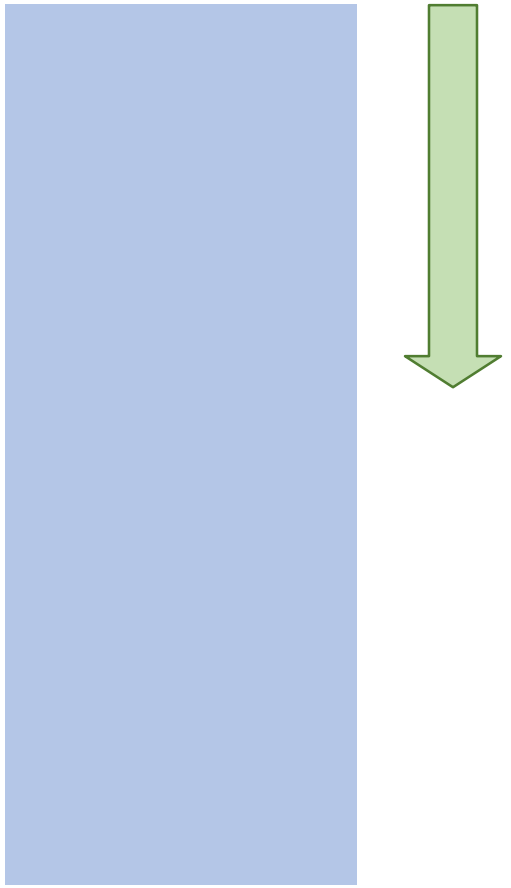
```
if (str.equals("pop")) {
    if (stack.empty()) { // 문제 조건에 따라 스택이 비어있다면 -1을 출력합니다.
        sb.append("-1\n");
        continue;
    }
    int x = stack.pop(); // 스택의 맨 위에 쌓인 값을 x에 저장하고 스택에서 제거합니다.
    /*
     * 또는 다음과 같이 쓸 수도 있습니다.
     * int x = stack.peek(); // 스택의 맨 위에 쌓인 값을 x에 저장합니다.
     * stack.pop(); // 스택의 맨 위에 쌓인 값을 제거합니다.
     */
    sb.append(x + "\n");
}
```

● 큐

▶ 대기열

▶ 1번 → 2번 → 3번 순서로 넣었다면, 꺼낼 때에도 1번 → 2번 → 3번

▶ FIFO(First In First Out)



● 직접 만든 큐 (121)

문제

준함이는 STL에서 지원하는 큐 기능을 믿지 않는다.
그러나 원하는 프로그램을 짜기 위해 큐 기능이 어떻게든 필요했던 준함이는 당신에게 큐 기능을 구현해 줄 것을 요청했다.
준함이가 원하는 큐 기능은 다음 명령들을 처리하는 것이다.

```
push X: 정수 X를 큐에 넣는다.
pop: 큐에서 가장 앞에 있는 수를 빼고, 그 수를 출력한다. 만약, 큐에 들어있는 정수가 없는 경우에는 -1을 출력한다.
size: 큐에 들어있는 정수의 개수를 출력한다.
empty: 큐가 비어있으면 1을, 아니면 0을 출력한다.
front: 큐의 가장 앞에 있는 정수를 출력한다. 만약 큐에 들어있는 정수가 없는 경우에는 -1을 출력한다.
back: 큐의 가장 뒤에 있는 정수를 출력한다. 만약 큐에 들어있는 정수가 없는 경우에는 -1을 출력한다.
```

준함이를 위해 큐 기능을 구현해주자

핵심 코드

▶ Queue 자료구조는 java.util.Queue 에 정의되어 있습니다.
하지만 Queue는 LinkedList로 구현되어 있기에, LinkedList도 import 해주어야 합니다.

python : from collections import deque 또는 리스트 사용
C++: #include<queue>

```
import java.util.Queue;
import java.util.LinkedList;
```

▶ Queue 자료구조는 다음과 같이 선언할 수 있습니다.

```
Queue<Integer> queue = new LinkedList<>();
// Queue는 LinkedList로 구현되어있습니다.
// 참조형 변수로만 선언 가능합니다.
// Queue<Object> queue_name = new LinkedList<>();
```

핵심 코드

▶ queue에 들어있는 원소의 수는 size 메소드를 통해 알 수 있습니다.

```
if(str.equals("size")) {
    int x = queue.size(); // queue에 들어있는 원소의 개수를 x에 저장합니다.
    sb.append(x+"\n");
}
```

▶ queue가 비어 있는 지의 여부는 isEmpty 메소드를 통해 알 수 있습니다.

```
if(str.equals("empty")) {
    Boolean isEmpty = queue.isEmpty(); // isEmpty에 큐가 비어있는지 여부를 저장합니다.
    if(isEmpty) sb.append("1\n");
    else sb.append("0\n");
}
```

핵심 코드

▶ Queue에 값을 넣는 것은 offer 메소드를 이용하면 됩니다. (add 함수도 동일합니다)

```
if(str.equals("push")) {
    int x = Integer.parseInt(st.nextToken());
    queue.offer(x); // queue에 x를 넣습니다.
    back = x;
}
```

▶ Queue에서 가장 먼저 들어간 값은 peek 메소드를 이용하여 접근할 수 있습니다.

```
if(str.equals("front")) {
    if(queue.isEmpty()) { // 문제 조건에 따라 큐가 비어있다면 -1을 출력합니다.
        sb.append("-1\n");
        continue;
    }
    int x = queue.peek(); // 큐에 가장 먼저 들어간 값을 x에 저장합니다.
    sb.append(x+"\n");
}
```

핵심 코드

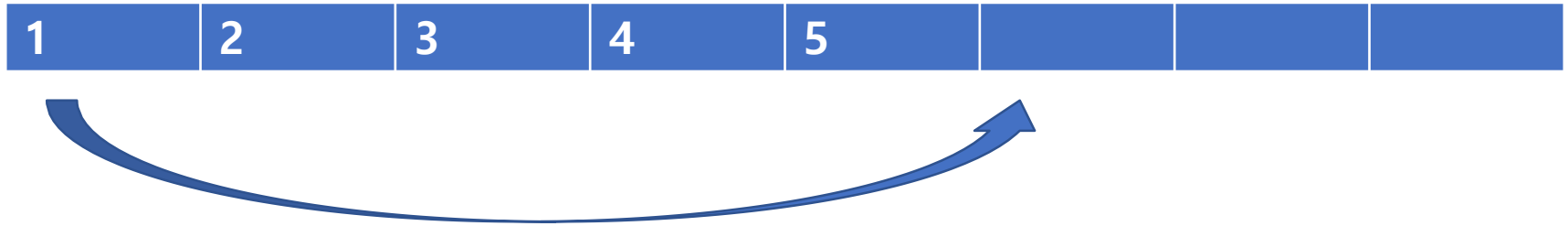
▶ Queue에서 값을 빼내는 것은 poll/remove 메소드로 할 수 있습니다.

```
if (str.equals("pop")) {
    if (queue.isEmpty()) { // 문제 조건에 따라 큐가 비어있다면 -1을 출력합니다.
        sb.append("-1\n");
        continue;
    }
    int x = queue.poll(); // 큐에 가장 먼저 들어간 값을 x에 저장하고, 큐에서 제거합니다.
    /*
     * 또는 다음과 같이 쓸 수도 있습니다
     * int x = queue.peek(); // 큐에 가장 먼저 들어간 값을 x에 저장합니다.
     * queue.remove(); // 큐에 가장 먼저 들어간 값을 큐에서 제거합니다. queue.poll(); 을 사용해도 됩니다.
     */
    sb.append(x + "\n");
}
```

▶ Java에서는 queue에서 back의 기능을 하는 메소드가 구현되어 있지 않습니다.

● 큐의 활용

▶ 구현 문제에서는 대기열, 맨 뒤로 보내기 연산 등을 구현할 때 쓰임



▶ 큐는 보통 구현에 활용되기보다는 BFS에서 사용된다.

● 조세퍼스 문제 (5)

문제

조세퍼스 문제는 다음과 같다.

1번부터 N번까지 N명의 사람이 원을 이루면서 앉아있고, 양의 정수 M(≤ N)이 주어진다. 이제 순서대로 M번째 사람을 제거한다. 한 사람이 제거되면 남은 사람들로 이루어진 원을 따라 이 과정을 계속해 나간다. 이 과정은 N명의 사람이 모두 제거될 때까지 계속된다. 원에서 사람들이 제거되는 순서를 (N, M)-조세퍼스 순열이라고 한다. 예를 들어 (7, 3)-조세퍼스 순열은 3, 6, 2, 7, 5, 1, 4 이다.

N과 M이 주어지면 (N,M)-조세퍼스 순열을 구하는 프로그램을 작성하시오.

● 큐를 이용한 풀이

▶ M번째 수를 꺼내는 작업은 다음과 같습니다.

1. 맨 앞 수를 맨 뒤로 보내는 작업을 M-1번 반복합니다.
2. 맨 앞 수를 큐에서 제거하고 출력합니다.

핵심 코드

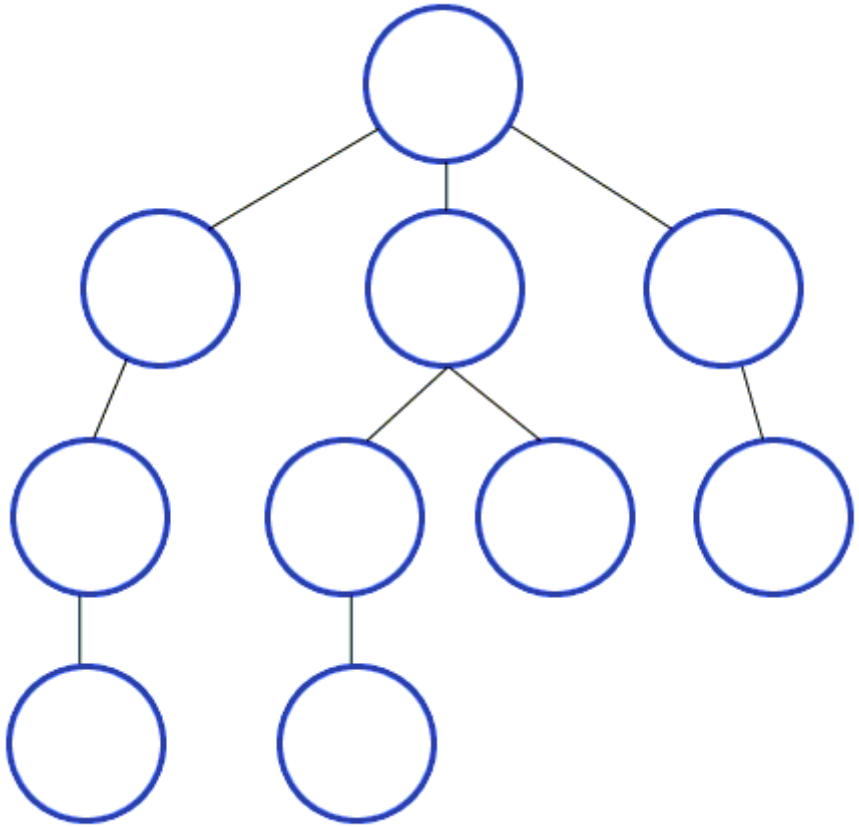
```
for (int i = 1; i <= N; i++) { // 큐에 1~N의 수를 차례로 넣는다.
    queue.offer(i);
}

while (!queue.isEmpty()) { // 큐가 빌 때까지 반복한다.
    for (int i = 1; i < M; i++) { // M-1개는 맨 뒤로 보내고
        int x = queue.poll();
        queue.offer(x);
    }
    int x = queue.poll(); // M번째는 빼내서 출력한다.
    sb.append(x + " ");
}
```

● BFS (너비우선탐색)

▶ DFS와는 다르게 재귀함수가 아닌 Queue를 이용해 그래프 탐색

▶ 답이 되는 경로가 여러가지여도 언제나 최단경로를 찾음을 보장함

▶ 최단, 최소해 찾을 때 사용



● 포탈 (2861)

문제

윌리는 좌표 X에서 좌표 Y까지 포탈을 이용하여 이동하려고 한다.
포탈을 이용하면 좌표 A에서 좌표 A+1, A-1, 2*A 중 한 좌표로 이동할 수 있다.
X와 Y를 입력받아 좌표 X에서 좌표 Y까지 이동하기위한 포탈의 최소 사용 횟수를 구해보자

입력

첫 줄에 X,Y가 공백을 사이에 두고 주어진다. 두 값 모두 0 이상 10만 이하이다.

출력

좌표 X에서 좌표 Y까지 이동하기위한 포탈의 최소 사용 횟수를 출력한다.

문제 해설

▶ BFS를 이용한 풀이

▶ 좌표 x에서는 x+1, x−1, 2*x 의 좌표를 탐색 가능

▶ dist[x] = 좌표 x로 이동하기 위해 필요한 최소 연산 수

▶ dist 값이 0이라는 것이 의미하는 것은? (1. 아직 방문하지 않은 좌표 2. x==X)

1. 예외 처리를 해 준다
2. dist[x] = (좌표 x로 이동하기 위해 필요한 최소 연산 수) + 1
3. dist 초기값 −1

핵심 코드

```
Queue<Integer> queue = new LinkedList<>();
dist[X] = 1; // 배열 전체를 -1로 초기화하거나, dist[X]를 1로 설정한 후 dist[Y]-1 출력
queue.offer(X);

while(!queue.isEmpty()) {
    int x = queue.poll();
    if(x + 1 <= 200000 && dist[x+1] == 0) {
        dist[x+1] = dist[x] +1;
        queue.offer(x+1);
    }
    if(x - 1 >= 0 && dist[x-1] == 0) {
        dist[x-1] = dist[x] +1;
        queue.offer(x-1);
    }
    if(2 * x <= 200000  && dist[2*x] == 0) {
        dist[2*x] = dist[x] +1;
        queue.offer(2*x);
    }
}
System.out.println(dist[Y]-1);
```

● 소수 경로 (4701)

문제

소수를 유난히도 좋아하는 창영이는 게임 아이디 비밀번호를 4자리 '소수'로 정해놓았다. 어느 날 창영이는 친한 친구와 대화를 나누었는데:

"이제 슬슬 비번 바꿀 때도 됐잖아"
"응 지금은 1033으로 해놨는데... 다음 소수를 무엇으로 할지 고민중이야"
"그럼 8179로 해"
"흠... 생각 좀 해볼게. 이 게임은 좀 이상해서 비밀번호를 한 번에 한 자리 밖에 못 바꾼단 말이야. 예를 들어 내가 첫 자리만 바꾸면 8033이 되니까 소수가 아니잖아. 여러 단계를 거쳐야 만들 수 있을 것 같은데... 예를 들면... 1033 1733 3733 3739 3779 8779 8179처럼 말이야."
"흠...역시 소수에 미쳤군. 그럼 아예 프로그램을 짜지 그래. 네 자리 소수 두 개를 입력받아서 바꾸는데 몇 단계나 필요한지 계산하게 말야."
"귀찮아"
그렇다. 그래서 여러분이 이 문제를 풀게 되었다. 입력은 항상 네 자리 소수만(1000 이상) 주어진다고 가정하자. 주어진 두 소수 A에서 B로 바꾸는 과정에서도 항상 네 자리 소수임을 유지해야 하고, '네 자리 수'라 하였기 때문에 0039 와 같은 1000 미만의 비밀번호는 허용되지 않는다.

문제 해설

- BFS를 이용한 최소 이동횟수 풀이
- dist[x] = x를 문제의 조건을 만족하면서 방문하는 데까지 걸리는 최소 이동횟수
- 10000 미만의 모든 소수들을 에라토스테네스의 체를 이용하여 구하자
- 자리수를 바꾸는 연산은 /와 % 연산을 이용하여 계산할 수 있다.

핵심 코드

```
isPrime = new int[10001]; // 소수면 0, 아니면 1
for(int i=2; i<=100; i++) { // 에라토스테네스의 체
    if(isPrime[i] == 1) continue;
    for(int j=2*i; j<10000; j+=i) isPrime[j] = 1;
}
```

```
queue.offer(A); dist[A] = 1;
while(!queue.isEmpty()) {
    int x = queue.poll();
    for(int y: getlist(x)) {
        if(dist[y] == 0) {
            dist[y] = dist[x] + 1;
            queue.offer(y);
        }
    }
}
```

핵심 코드

	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50
51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70
71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90
91	92	93	94	95	96	97	98	99	100

핵심 코드

```
public static ArrayList<Integer> getlist(int x) {
    ArrayList<Integer> arr = new ArrayList<Integer>();
    for(int i=1 ; i<10; i++) { // 천의 자리를 i로
        int tmp = i*1000 + (x%1000);
        if(isPrime[tmp]==0 && x!=tmp) arr.add(tmp);
    }
    for(int i=0; i<10; i++) { // 백의 자리를 i로
        int tmp = (x/1000)*1000 + i*100 + (x%100);
        if(isPrime[tmp]==0 && x!=tmp) arr.add(tmp);
    }
    for(int i=0; i<10; i++) { // 십의 자리를 i로
        int tmp = (x/100)*100 + i*10 + (x%10);
        if(isPrime[tmp]==0 && x!=tmp) arr.add(tmp);
    }
    for(int i=0; i<10; i++) { // 일의 자리를 i로
        int tmp = (x/10)*10 + i;
        if(isPrime[tmp]==0 && x!=tmp) arr.add(tmp);
    }
    return arr;
}
```