3주차

2020.11.9 – 11. 15

그리디 알고리즘2 – 큰 수의 법칙

N : 숫자 개수 ,M : 총 더하는 횟수 ,K : 최대 더할 수 있는 횟수
"Idea : 1번째로 큰 숫자와 2번째로 큰 숫자만 알면 된다. (K+1)번째 숫자를, 1번째로 큰 수와 2번째로 큰 수 같으면 그대로, 다르면 2번째로 큰 수를 더해준다.

```
N, M, K = map(int, input().split())

nums = [int(x) for x in input().split()]
# 정렬(큰수->작은수)
data = sorted(nums, reverse = True)
sum_data = 0
idx_max = data[0]
idx_next_max = data[1]
cnt = 0

while 1 :
    cnt = cnt+1
    # K번 반복된 경우
    if cnt % K == 0 :
        # 일단 가장 큰수 더해준다.
        sum_data = sum_data + idx_max
        # K번 다음(K+1)에 더할 수를 지정해준다.
        cnt = cnt+1
        # 가장 큰 수 다음 숫자가 다르면 그 다음으로 큰 수 더함
        if idx_max != idx_next_max :
            sum_data = sum_data + idx_next_max
        # 가장 큰 수 다음 숫자 같으면 가장 큰 수 더함
        else :
            sum_data = sum_data + idx_max
    # K번 반복되지 않은 경우
    else :
        # 가장 큰 수 더함
        sum_data = sum_data + idx_max |
    # 탈출조건 (M번 반복된 경우)
    if cnt == M :
        break
print(sum_data)
```

그리디 알고리즘3 – 숫자 카드 게임

N x M 형태의 입력 받은 숫자 중에서 행(가로) 기준으로 가장 작은 값들을 추출하고, 그 중에서 가장 큰 값을 추출한다.
*** 행(가로), 열(세로)

```
# n,m = 행,열
n,m = map(int, input().split())

ans_arr = []

for i in range(n):
    # 행(가로) 단위로 입력받은 데이터 중
    data = [int(x) for x in input().split()]
    # 최소값을 찾은 다음
    min_data = min(data)
    # 그 중에서
    ans_arr.append(min_data)
# 최대값을 찾는다.
print(max(ans_arr))
```

그리디 알고리즘4 – 1이 될 때까지

N이 1이 될 때 까지 최소 횟수를 구하시오
1. N에서 1을 뺀다.
2. N을 K로 나눈다.(N이 K로 나누어지는 경우만 가능)

```
n, k = map(int, input().split())
cnt = 0

while 1 :

    # n을 k로 나눈 나머지가 0이 아닌경우 -1
    if n%k != 0 :
        n = n-1
        cnt = cnt+1
        # 탈출조건 n이 1인 경우
        if n==1 :
            print(cnt)
            break

    # n을 k로 나눈 나머지가 0인 경우
    else :
        n = n // k
        cnt = cnt+1
        # 탈출조건 n이 1인 경우
        if n==1 :
            print(cnt)
            break
```

구현1 - 상하좌우

```python
n = int(input())
x,y = 1, 1
plans = input().split()
# L R U D
dx = [0,0,-1,1]
dy = [-1,1,0,0]
move_types = ['L','R','U','D']

for plan in plans :
    for i in range(len(move_types)):

        # 방향과 일치하면 이동
        if plan == move_types[i] :
            nx = x + dx[i]
            ny = y + dy[i]

        # 맵에서 벗어나면 넘어감
            if nx < 1 or nx > n or ny < 1 or ny > n :
                continue

            # 그렇지 않으면 이동
            x = nx
            y = ny

# 마지막 도착지 출력
print(x, y)
```

구현2 - 시각

N이 입력되면 00시 00분 00초부터 N시 59분 59까지 모든 시각 중에서 3이 하나라도 포함된 모든 경우의 수를 구하시오

```
N = int(input())

cnt = 0
# 0시부터 N시까지
for i in range(0,N+1):
    # 0분부터 59분까지
    for j in range(0,60):
        # 0초부터 59초까지
        for k in range(0,60):
            # 3이 있다면(시+분+초)
            if '3' in str(i) + str(j) + str(k):
                cnt = cnt+1
print(cnt)
```

구현3 - 나이트

```
n = input()

x = int(n[1])
y = ord(n[0])-97+1

# 경우의 수(오2아1,왼2아1,오2위1,왼2위1,아2오1,아2왼1,위2오1,위2왼1)
move_step = [(1,2),(1,-2),(-1,2),(-1,-2),(2,1),(2,-1),(-2,1),(-2,-1)]

result = 0
for step in move_step :
    # move row and column
    nx = x + step[1]
    ny = y + step[0]

    if nx >= 1 and nx <= 8 and ny >= 1 and ny <= 8 :
        result = result+1

print(result)
```

스택 큐

```
# stack = 후입선출
stack = []

stack.append(5)
stack.append(3)
stack.append(1)

stack.pop()
print(stack)
print(stack[::-1])
```

```
[5, 3]
[3, 5]
```

```
# queue = 선입선출
from collections import deque

queue = deque()
queue.append(5)
queue.append(4)
queue.append(3)

queue.popleft()
print(queue)
queue.reverse()
print(queue)
```

```
deque([4, 3])
deque([3, 4])
```

DFS – 깊이 우선 탐색

```python
def dfs(graph,v,visited) :
    # 방문 시 True
    visited[v] = True
    print(v, end=' ')
    for i in graph[v] :
        # 아직 방문하지 않았으면 계속 탐색
        if not visited[i] :
            dfs(graph, i, visited)

graph = [
    [], # 0번 노드
    [2,3,8], # 1번 노드
    [1,7], # 2번 노드
    [1,4,5], # 3번 노드
    [3,5], # 4번 노드
    [3,4], # 5번 노드
    [7], # 6번 노드
    [2,6,8], # 7번 노드
    [1,7] # 8번 노드
]

# 0번 ~ 8번 노드
visited = [False]*9
dfs(graph,1,visited)
```

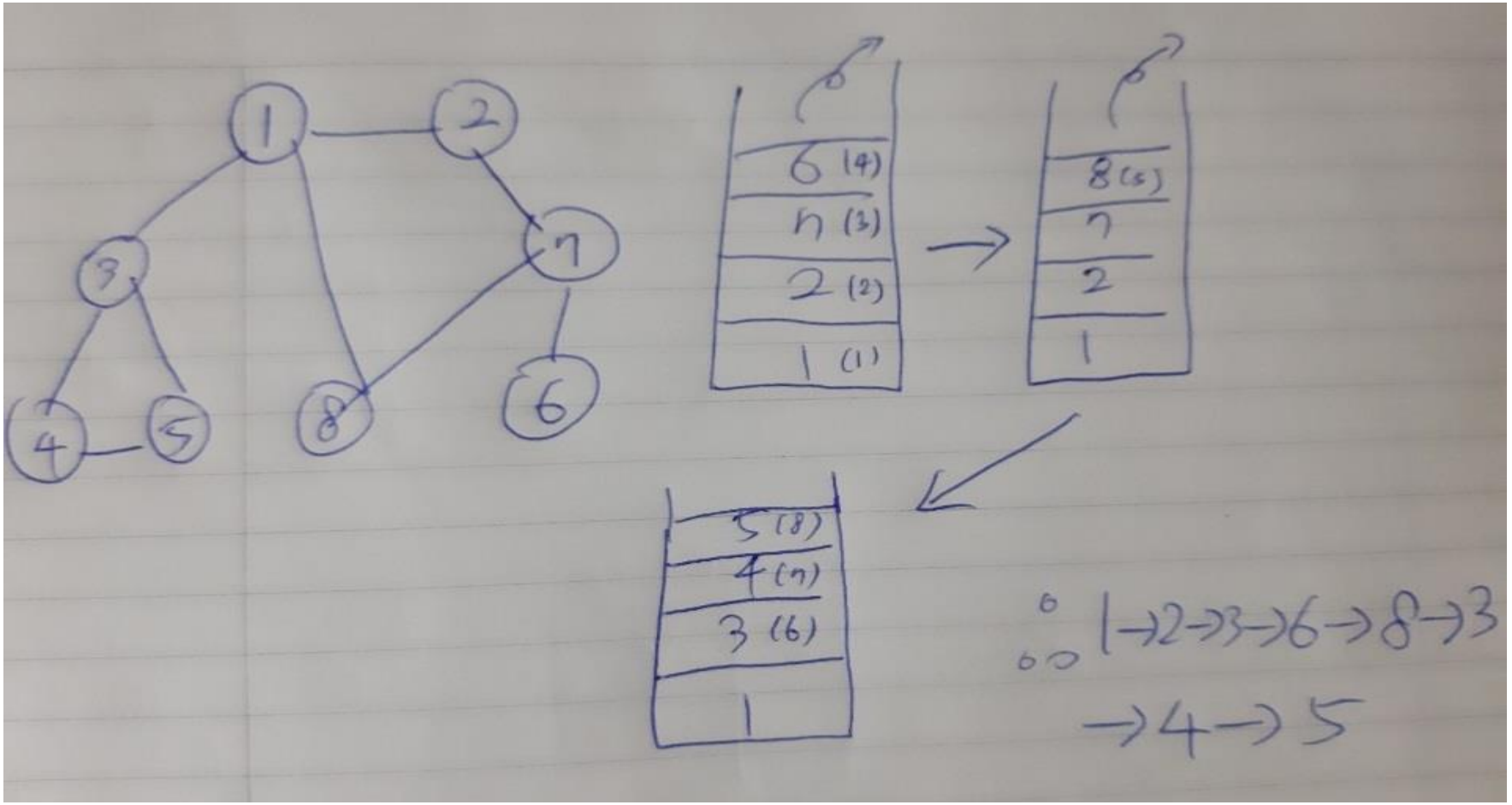
```
1 2 7 6 8 3 4 5
```

작은 번호의 Node부터 탐색하는 것이 국룰

1. 작은 번호부터 스택에 넣는다.
2. 더 이상 갈 수 없으면 POP한다.
3. 반복한다.

=> 스택에 넣은 순서가 탐색 순서임

DFS



BFS – 너비 우선 탐색

```python
from collections import deque
def bfs(graph,start,visited) :
    # queue 만들기
    queue = deque([start])
    # 방문 시 True
    visited[start] = True
    # queue가 빌때까지 반복
    while queue :
        # 가장 먼저 들어온 원소 pop
        v = queue.popleft()
        print(v, end=' ')
        for i in graph[v] :
            # 아직 방문하지 않았으면 계속 탐색
            if not visited[i] :
                # 새로 방문하면 append하고, True만들기
                queue.append(i)
                visited[i] = True
graph = [
    [], # 0번 노드
    [2,3,8], # 1번 노드
    [1,7], # 2번 노드
    [1,4,5], # 3번 노드
    [3,5], # 4번 노드
    [3,4], # 5번 노드
    [7], # 6번 노드
    [2,6,8], # 7번 노드
    [1,7] # 8번 노드
]

# 0번 ~ 8번 노드
visited = [False]*9
bfs(graph,1,visited)
```

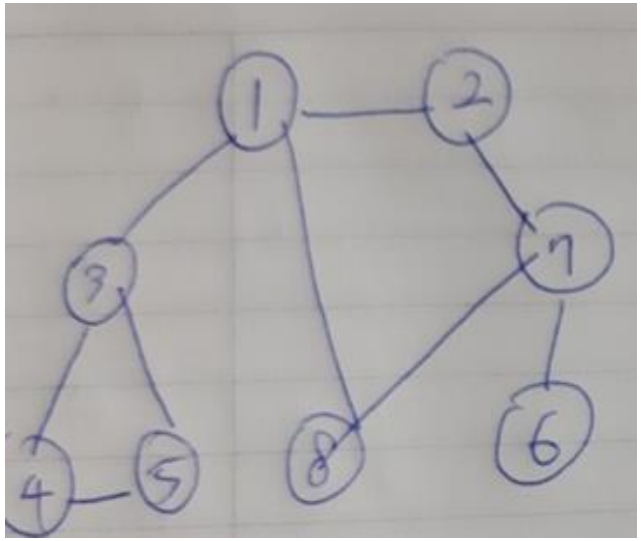
```
1 2 3 8 7 4 5 6
```

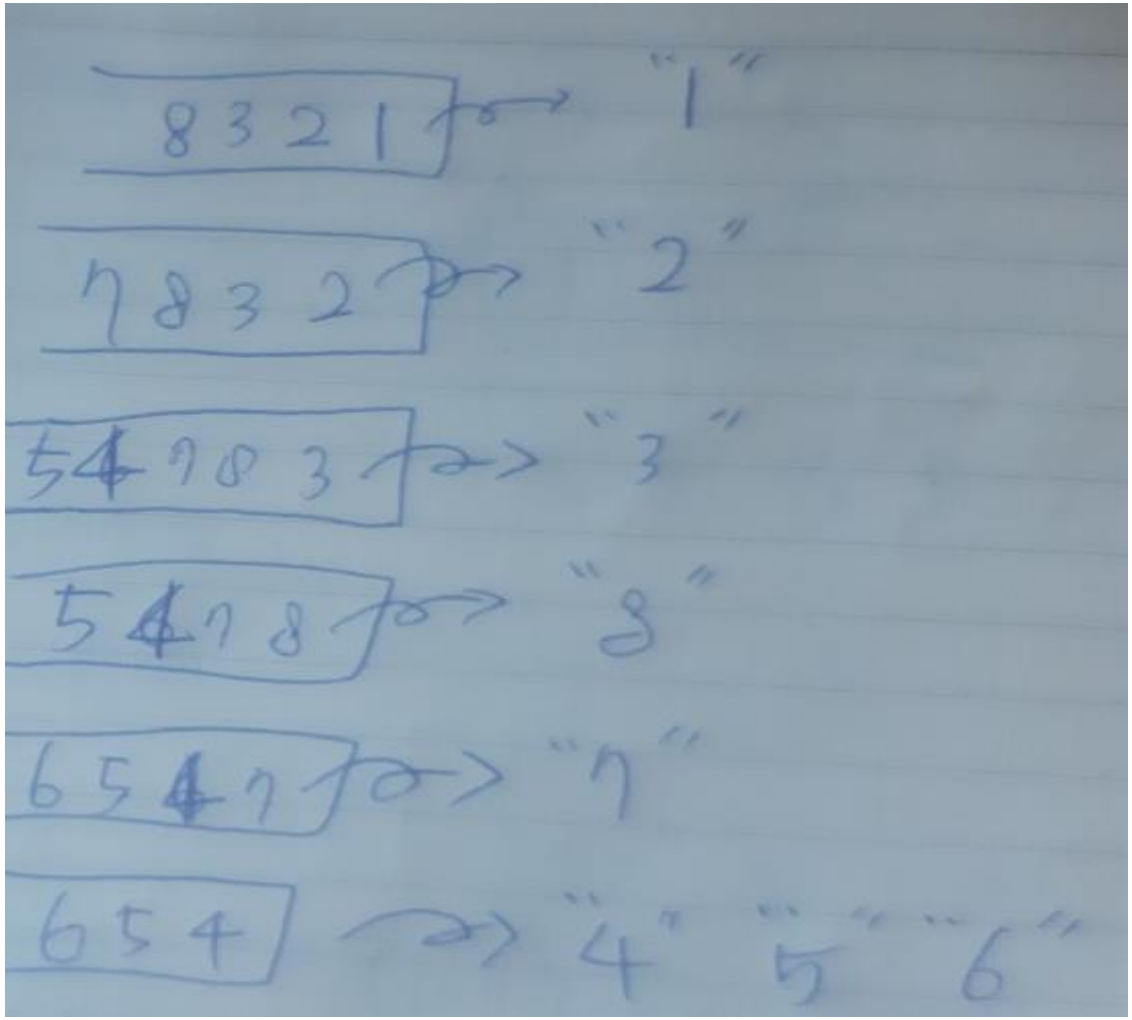
작은 번호의 Node부터 탐색하는 것이 국룰

1. 작은 번호부터 큐에 넣는다.
2. 더 이상 갈 수 없으면 POP하고, 연결된 Node를 넣는다.
3. 반복한다.

=> 큐에서 pop하는 순서가 탐색 순서임

BFS





CodeUp 1405

```
num = 5
nums = [1,2,3,4,5]

idx = 0
for i in range(num):
    # 첫번째 입력 숫자 인덱스 잡기
    idx = i
    for k in range(num) :
        # 첫번째 인덱스부터 출력하는데,
        print(nums[idx], end=' ')
        idx = idx+1
        # 마지막에 도달하면 idx 0부터 다시 시작함
        if idx == num:
            idx = 0
    print('')
```

```
1 2 3 4 5
2 3 4 5 1
3 4 5 1 2
4 5 1 2 3
5 1 2 3 4
```