2021 여름학기 동국대학교 SW역량강화캠프

12일차. BFS 1

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

```
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<>();
```

▶ 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");
}
```

▶ 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");
}
```

▶ 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");
}
```

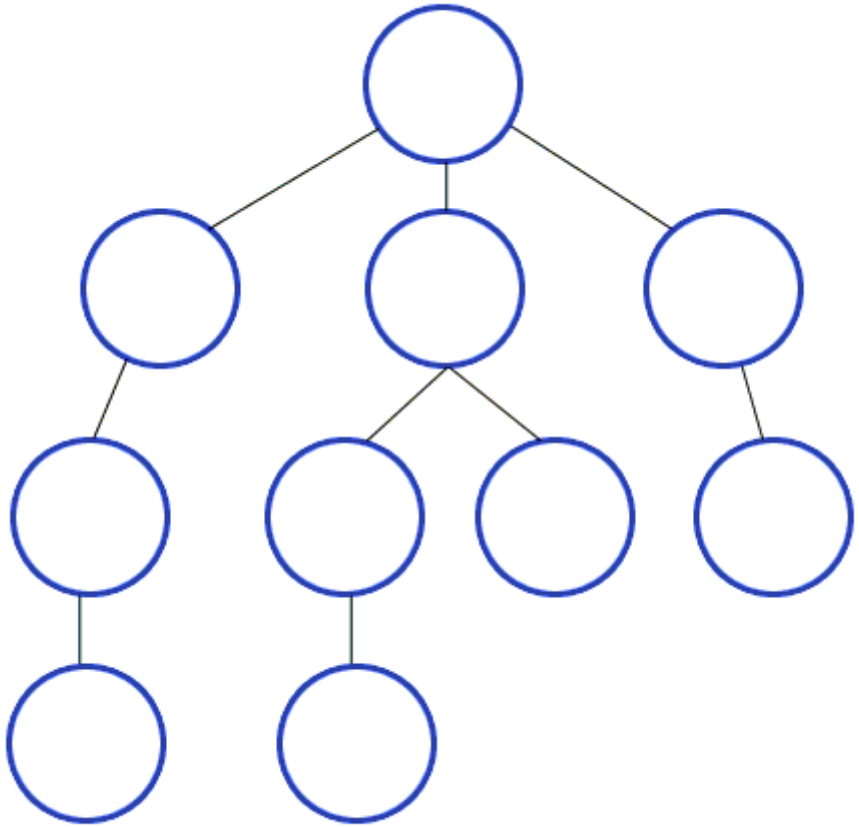
▶ Java에서는 queue에서 back의 기능을 하는 메소드가 구현되어 있지 않습니다.

● BFS (너비우선탐색)

▶ DFS와는 다르게 재귀함수가 아닌 Queue를 이용해 그래프 탐색

▶ 답이 되는 경로가 여러가지여도 언제나 최단경로를 찾음을 보장함

▶ 최단, 최소해 찾을 때 사용



● DFS와 BFS (4321)

문제

그래프를 DFS로 탐색한 결과와 BFS로 탐색한 결과를 출력하는 프로그램을 작성하시오. 단, 방문할 수 있는 정점이 여러 개인 경우에는 정점 번호가 작은 것을 먼저 방문하고, 더 이상 방문할 수 있는 점이 없는 경우 종료한다. 정점 번호는 1번부터 N번까지이다.

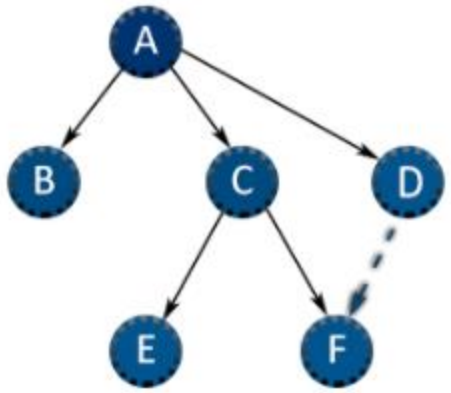
입력

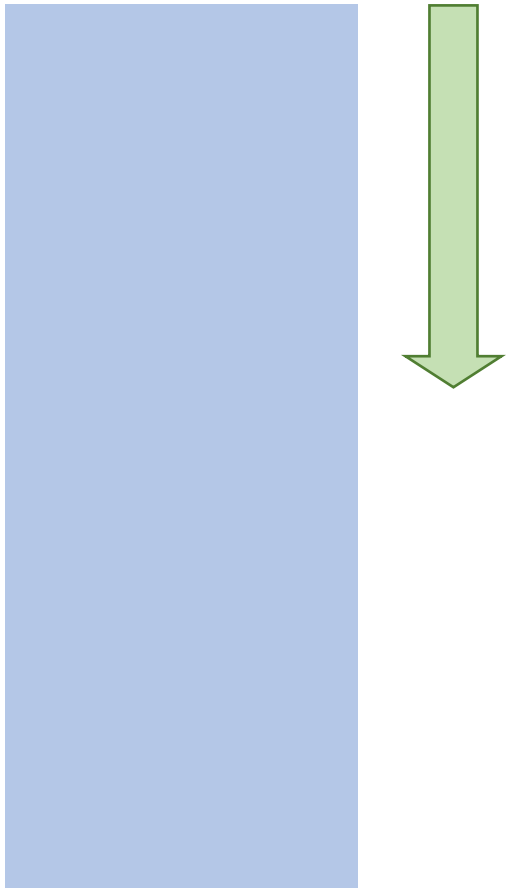
첫째 줄에 정점의 개수 N($1 \le N \le 1,000$), 간선의 개수 M($1 \le M \le 10,000$), 탐색을 시작할 정점의 번호 V가 주어진다. 다음 M개의 줄에는 간선이 연결하는 두 정점의 번호가 주어진다. 어떤 두 정점 사이에 여러 개의 간선이 있을 수 있다. 입력으로 주어지는 간선은 양방향이다.

출력

첫째 줄에 DFS를 수행한 결과를, 그 다음 줄에는 BFS를 수행한 결과를 출력한다. V부터 방문된 점을 순서대로 출력하면 된다.

▶ BFS 정점의 탐색 순서를 출력





핵심 코드

```
Queue<Integer> queue = new LinkedList<>();
visit = new boolean[N + 1];

queue.offer(V); // 큐에 처음 점 V를 offer
visit[V] = true; // visit 배열은 방문 여부보다는 큐에 들어간 적 있는지 여부를 확인한다.(중복 offer 방지)
while (!queue.isEmpty()) {
    int x = queue.poll(); // queue에서 꺼낸 값 x
    sb.append(x + " "); // x를 출력
    for (int y : graph.get(x)) { // x와 인접한 y
        if (!visit[y]) { // y를 큐에 넣은 적이 없다면
            queue.offer(y); // y를 queue에 넣고 visit에 체크
            visit[y] = true;
        }
    }
}
System.out.println(sb.toString());
```

● 미로 탐험(13)

문제

N×M크기의 배열로 표현되는 미로가 있다.

```
1  0  1  1  1  1
1  0  1  0  1  0
1  0  1  0  1  1
1  1  1  0  1  1
```

미로에서 1은 이동할 수 있는 칸을 나타내고, 0은 이동할 수 없는 칸을 나타낸다. 이러한 미로가 주어졌을 때, (1, 1)에서 출발하여 (N, M)의 위치로 이동할 때 지나야 하는 최소의 칸 수를 구하는 프로그램을 작성하시오. 한 칸에서 다른 칸으로 이동할 때, 서로 인접한 칸으로만 이동할 수 있다.

위의 예에서는 15칸을 지나야 (N, M)의 위치로 이동할 수 있다. 칸을 셀 때에는 시작 위치와 도착 위치도 포함한다.

문제 해설

▶ BFS를 이용하여 dist 배열을 채운다.

▶ dist[x][y] = (1,1)로부터의 최소 칸 수

▶ 초기값: dist[1][1] = 1, queue = [{1,1}]

▶ Queue에는 이차원 좌표를 넣어야 하므로 Integer[]이나 2개의 정수를 담을 수 있는 class/struct를 선언하여 사용

핵심 코드

```
Queue<Integer[]> queue = new LinkedList<>();
queue.offer(new Integer[] { 1, 1 });
dist[1][1] = 1;

while (!queue.isEmpty()) {
    Integer[] tmp = queue.poll();
    int x = tmp[0], y = tmp[1];
    for (int i = 0; i < 4; i++) {
        int newx = x + dx[i], newy = y + dy[i]; // (x,y)와 인접한 점 (newx, newy)
        if (newx < 1 || newx > N || newy < 1 || newy > M) // OOB 체크
            continue;
        if (arr[newx][newy] == 1 && dist[newx][newy] == 0) { // (newx, newy)가 queue에 들어간 적 없는 '1'인 좌표라면
            dist[newx][newy] = dist[x][y] + 1; // (newx, newy)까지의 거리는 (x,y)까지의 거리 + 1
            queue.offer(new Integer[] { newx, newy }); // queue에 (newx, newy) offer
        }
    }
}
```

● 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 해준다
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);
```

● 어부바 (3557)

문제

강아지 루시를 기르는 선기쌤은 매번 산책을 하는데 힘이 듭니다. 언제나 알고리즘의 효율성을 중시하는 선기쌤답게 산책을 할 때에도 최대한 에너지 효율적으로 움직이려고 합니다.

선기쌤은 한 칸을 이동하는데 A만큼 에너지가 필요합니다. 그리고 루시는 한 칸을 이동하는데 B만큼 에너지가 필요합니다. 하지만 선기쌤과 루시가 같은 칸에서 만나 어부바를 하게 된다면 둘이 합쳐서 한칸 움직이는데 P만큼 에너지가 필요합니다.

산책을 하는 공원은 1번부터 N번까지 지점들로 이루어져 있고 선기쌤은 1번, 강아지 루시는 2번 그리고 집은 N번에 위치해 있습니다. 각 지점들 사이의 도로와 A, B, P의 값이 주어질 때 최대한 효율적으로 움직여서 선기쌤과 루시가 집에 도착할때까지 필요한 에너지를 구해주세요.

입력

첫째 줄에 A, B, P, N, M이 주어집니다. 각 숫자는 최대 40,000이고 A, B, P, N은 문제에서 설명한 값이며 M은 연결하는 도로의 개수입니다. N은 최소 3입니다.
둘째줄부터 M개 줄에 걸쳐 도로로 연결된 두 지점이 공백을 사이에 두고 주어집니다.

출력

선기쌤과 루시가 집에 도착하기 위해 필요한 에너지의 합의 최소값을 출력하세요.

문제 해설

▶ BFS를 이용한 최단경로 풀이

▶ 모든 정점 x들 중

dist(1, x) * A + dist(2, x) * B + dist(x, N) * P

의 최소값을 구하면 됩니다. (dist(A,B)는 A와 B 사이의 거리)

▶ 1, 2, N번 정점에서 BFS를 한 번씩 돌리면, 원하는 거리값을 전부 저장할 수 있다.

핵심 코드

```
dist = new int[N + 1][3];
int[] arr = new int[] { 1, 2, N };
for (int i = 0; i < 3; i++) {
    Queue<Integer> queue = new LinkedList<>();
    int start = arr[i];
    dist[start][i] = 1;
    queue.offer(start);
    while (!queue.isEmpty()) {
        int x = queue.poll();
        for (int y : graph.get(x)) {
            if (dist[y][i] == 0) {
                dist[y][i] = dist[x][i] + 1;
                queue.offer(y);
            }
        }
    }
}
long ans = Long.MAX_VALUE;
for (int i = 1; i <= N; i++) {
    if (dist[i][0] == 0 || dist[i][1] == 0 || dist[i][2] == 0)
        continue;
    long tmp = (long) A * (dist[i][0] - 1) + B * (dist[i][1] - 1) + P * (dist[i][2] - 1);
    if (ans > tmp)
        ans = tmp;
}
System.out.println(ans);
```