Labyrinth

TASK | STATISTICS

Time limit: 1.00 s **Memory limit:** 512 MB

You are given a map of a labyrinth, and your task is to find a path from start to end. You can walk left, right, up and down.

Input

The first input line has two integers n and m: the height and width of the map.

Then there are n lines of m characters describing the labyrinth. Each character is `.` (floor), `#` (wall), `A` (start), or `B` (end). There is exactly one `A` and one `B` in the input.

Output

First print "YES", if there is a path, and "NO" otherwise.

If there is a path, print the length of the shortest such path and its description as a string consisting of characters `L` (left), `R` (right), `U` (up), and `D` (down). You can print any valid solution.

Constraints

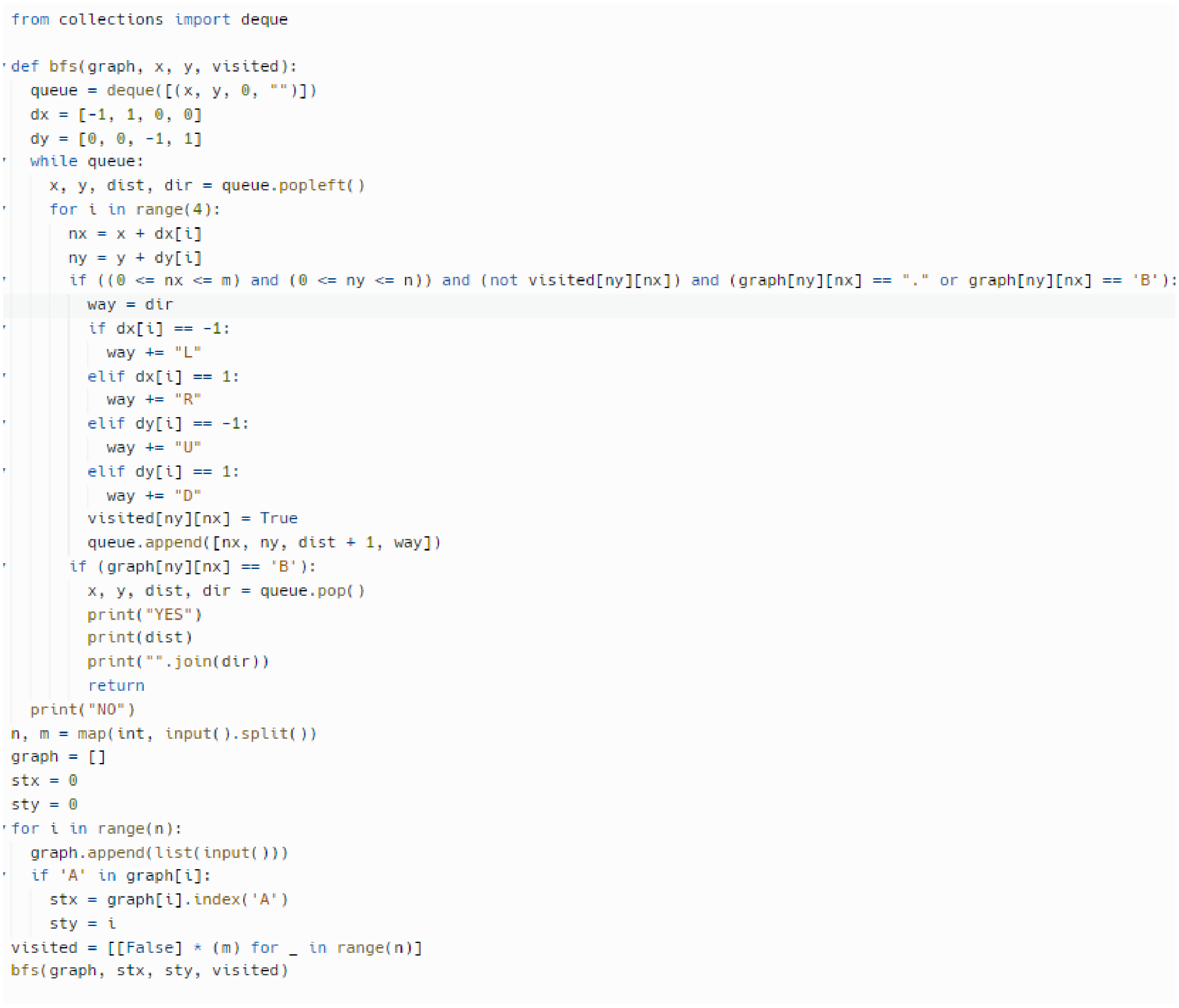
- $1 \le n, m \le 1000$

Example

Input:
```
5 8
########
#.A#...#
#.##.#B#
#......#
########
```

Output:
```
YES
9
LDDRRRRRU
```

Code



```python
from collections import deque

def bfs(graph, x, y, visited):
  queue = deque([(x, y, 0, "")])
  dx = [-1, 1, 0, 0]
  dy = [0, 0, -1, 1]
  while queue:
    x, y, dist, dir = queue.popleft()
    for i in range(4):
      nx = x + dx[i]
      ny = y + dy[i]
      if ((0 <= nx <= m) and (0 <= ny <= n)) and (not visited[ny][nx]) and (graph[ny][nx] == "." or graph[ny][nx] == 'B'):
        way = dir
        if dx[i] == -1:
          way += "L"
        elif dx[i] == 1:
          way += "R"
        elif dy[i] == -1:
          way += "U"
        elif dy[i] == 1:
          way += "D"
        visited[ny][nx] = True
        queue.append([nx, ny, dist + 1, way])
      if (graph[ny][nx] == 'B'):
        x, y, dist, dir = queue.pop()
        print("YES")
        print(dist)
        print("".join(dir))
        return
  print("NO")
n, m = map(int, input().split())
graph = []
stx = 0
sty = 0
for i in range(n):
  graph.append(list(input()))
  if 'A' in graph[i]:
    stx = graph[i].index('A')
    sty = i
visited = [[False] * (m) for _ in range(n)]
bfs(graph, stx, sty, visited)
```

The purpose of the problem is to find whether it is possible to reach place B from place A, the least number of places needed if possible, and the direction orders of the shortest path from place A to place B. The challenging part of the problem was to keep track of the directions in the process of BFS because the queue needed to contain not only distance but also directions.