Topic 5: Breadth First Search (BFS)

Competitive Programming I (Fall 2019)

Ninghui Li (Purdue University)

Kattis: Grid (Required)

Problem from Southeast USA Regional 2015

You are on an n×m grid where each square on the grid has a digit on it. From a given square that has digit k on it, a move consists of jumping exactly k squares in one of the four cardinal directions. A move cannot go beyond the edges of the grid; it does not wrap. What is the minimum number of moves required to get from the top-left corner to the bottom-right corner?

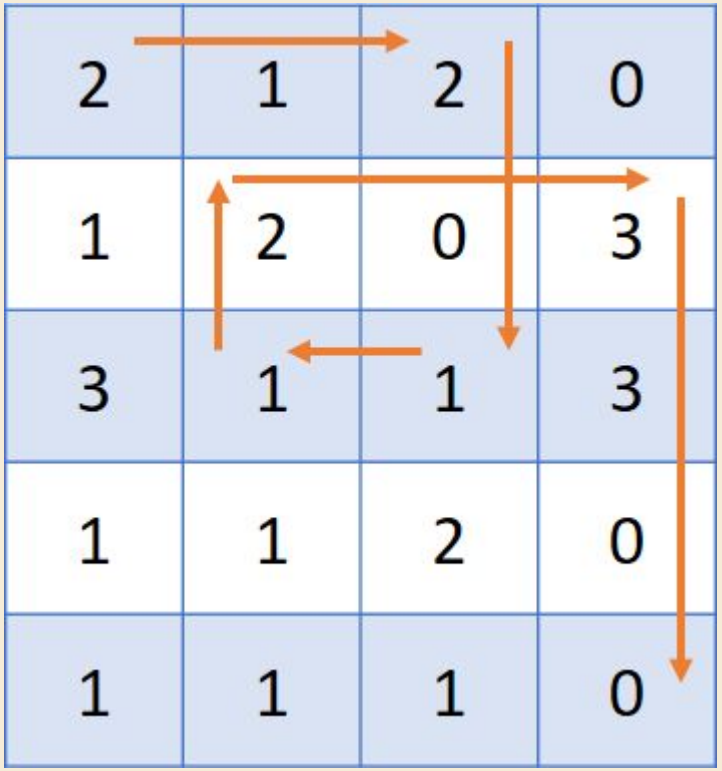
Grid: An Example

Start at (0,0), which contains 2.

Goal is to go to bottom-right corner (4,3)

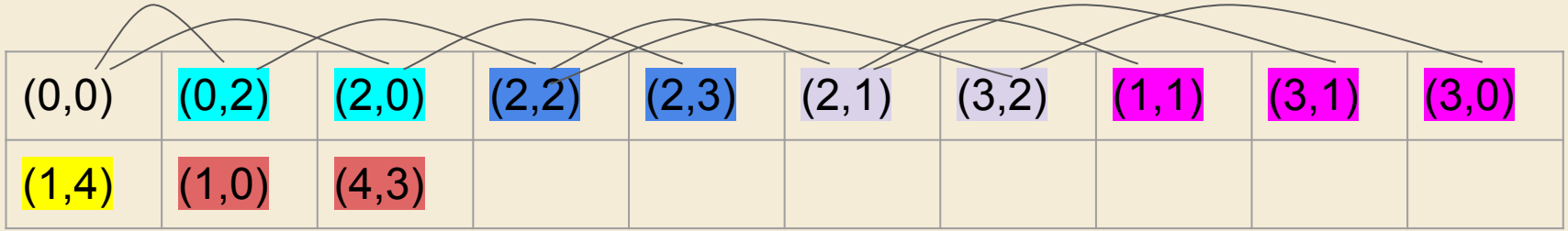
Minimal number of steps is 6

Path shown on the picture.



Breadth First Search

2 (0)	1	2 (1)	0
1 (6)	2 (4)	0 (3)	3 (5)
3 (1)	1 (3)	1 (2)	3 (2)
1 (4)	2 (4)	2 (3)	0 (5)
1 (5)	1	1 (5)	0 (6)



Grid: Code to Read Input and Initialize (in Java)

```
static final int[] dx ={0,0,1,-1}, dy={1,-1,0,0};
static int R, C;
static int[][] A, dist; // A is input, dist stores distances
public static void main(String[] args) {
  Scanner in = new Scanner(System.in);
  R = in.nextInt();    C = in.nextInt();
  A = new int[R][C];
  dist = new int[R][C];
  for (int i=0; i<R; i++) {
    String line = in.next();
    for (int j=0; j<C; j++) {
      A[i][j] = line.charAt(j) - '0';
      dist[i][j] = Integer.MAX_VALUE;
    }
  }
  System.out.println(solve());
}
```

```
static int solve() {
  Deque<int[]>  q = new ArrayDeque<int[]>();
  q.addLast(new int[]{0,0});  // row, column
  dist[0][0] = 0;
  while (! q.isEmpty()) {
    int[] x = q.removeFirst();
    int i = x[0], j=x[1], d = dist[i][j];
    for (int k=0; k<4; k++) {
      int ni = i + dx[k]*A[i][j],  nj = j + dy[k]*A[i][j];
      if (ni == R-1 && nj == C-1)    return d+1;  // reached goal
      if (0<=ni && ni<R && 0<=nj && nj<C && A[ni][nj]!=0 && ?? ){
        q.addLast(new int[]{ni, nj});
        dist[ni][nj] = d+1;
      }
    }
  }
  return -1;
}
```

Is this condition necessary?

`dist[ni][nj]==Integer.MAX_VALUE`

For checking whether it has been added

BFS PseudoCode and Key Points

- Have a data structure for maintaining a state for each node/cell.
  - Often an array when nodes are easily numbered
  - State is often an integer (such as distance from origin)
- Uses a queue, which is initialized to include the first reachable node(s)
- Invariance: Whenever adding a node to the queue, update its state (e.g., distance) to reflect that fact.
- While queue is not empty, extract the head, for all its neighbors, check their state, if they are not already in the queue, add it, and update the state.

Kattis: Rings (Required)

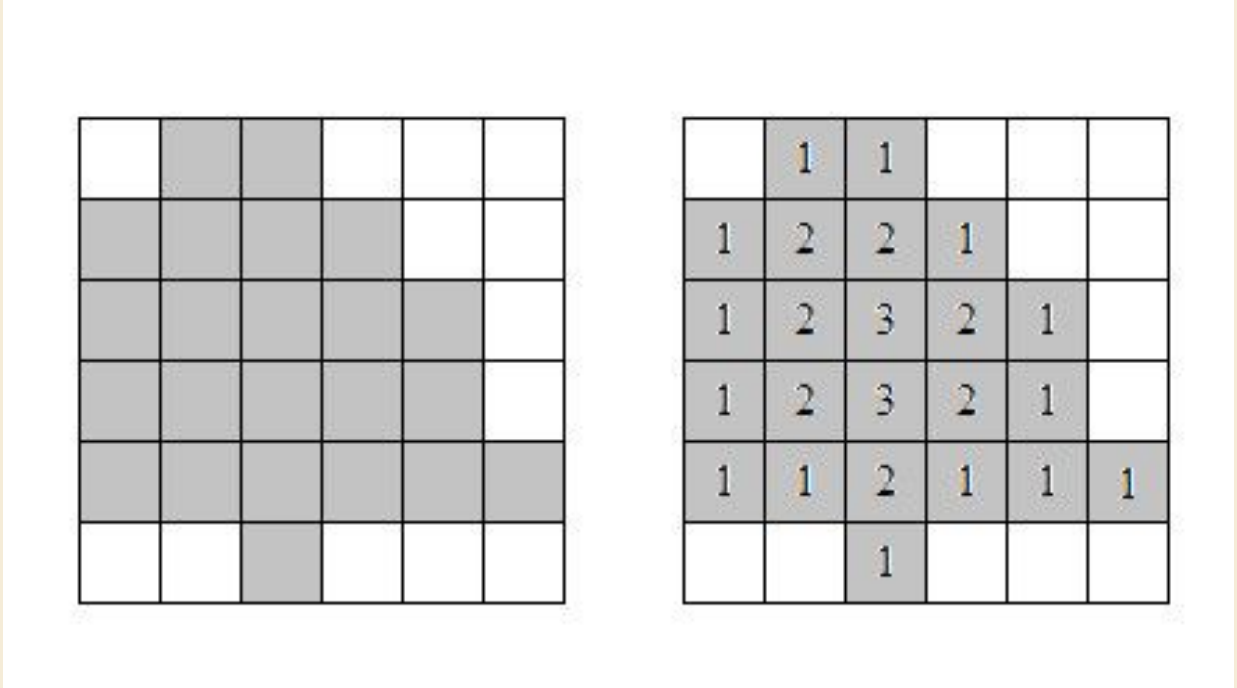
Problem from 2015 ICPC East Central North America Regional

Given a cross section of a tree on a two dimensional grid, with the interior of the tree represented by a closed polygon of grid squares. One assigns rings from the outer parts of the tree to the inner as follows: calling the non-tree grid squares “ring 0”, each ring n is made up of all those grid squares that have at least one ring (n−1) square as a neighbor (sharing a common edge).

Example

Constraints:

Number of rows, columns ≤100



Rings: Analysis

- First find the outer layer, which consists of
  - a trunk cell adjacent to white square
  - or, a trunk on the border
- Add cells in the outer layer to the queue
- What numbering scheme should we use if we use just one 2d int array?
  - White cell, trunk cell (unvisited)
  - Visited trunk cell (at different rings)

Rings Code: Set up the Queue (Code for reading data not included).

```
Deque<int[]> q = new ArrayDeque<int[]>();
for (int i=0; i<R; i++) for (int j=0; j<C; j++) {
  if (A[i][j] == 0) {    // white space
    for (int k=0; k<4; k++) {
      int nx = i + dx[k], ny = j + dy[k];
      if (0<=nx && nx <R && 0<=ny && ny<C && A[nx][ny]==-1) {
        A[nx][ny] = 1;
        q.addLast(new int[]{nx,ny});
      }
    }
  } else if (A[i][j]=-1 && (i==0||i==R-1||j==0||j==C-1)) {
    A[i][j] = 1;
    q.addLast(new int[]{i, j});
  }
}
```

Trunk cells initialized to -1.
Positive value indicate ring #.

Why need the green code?

Rings: BFS Code (code for printing output not included)

```
int max_ring = 1; // to store maximum number of rings
                  // needed to generate output
while (! q.isEmpty()) {
  int[] c = q.removeFirst();
  for (int k=0; k<4; k++) {
    int nx = c[0] + dx[k], ny = c[1] + dy[k];
    if (0<=nx && nx <R && 0<=ny && ny<C && A[nx][ny]==-1) {
      A[nx][ny] = A[c[0]][c[1]]+1;
      q.addLast(new int[]{nx,ny});
      max_ring = Math.max(max_ring, A[nx][ny]);
    }
  }
}
```

LeetCode: Open the Lock

You have a lock with 4 circular wheels. Each wheel has 10 slots: '0', '1', '2', '3', '4', '5', '6', '7', '8', '9'. The wheels can rotate freely and wrap around (we can turn '9' to be '0', or '0' to be '9'). Each move consists of turning one wheel one slot.

The lock initially starts at '0000'. You are given a list of deadends, meaning if the lock displays any of these codes, the wheels of the lock will stop turning and you will be unable to open it.

Given a target, return the minimum total number of turns required to open the lock, or -1 if it is impossible.

Open the Lock: Examples

```
Input: target = "0202"
deadends = ["0201","0101","0102","1212","2002"],
Output: 6
Explanation: A sequence of valid moves would be
"0000" -> "1000" -> "1100" -> "1200" -> "1201" -> "1202" -> "0202".

Note that a sequence like "0000" -> "0001" -> "0002" -> "0102" ->
"0202" would be invalid, because the wheels of the lock become
stuck after the display becomes the dead end "0102".
```

From “0000”, how many states can one go to in one step?

Open the Lock Code (in C++)

```
int openLock(vector<string>& deadends, string target) {
  unordered_set<string> blocked(deadends.begin(), deadends.end());
  if (blocked.count("0000")) return -1;
  unordered_set<string> visited({"0000"});
  queue<string> q{{"0000"}};
  ...
  // bfs code on next slide
  ...
  return -1;
}
```

Can I change this to for (int k=0; k<q.size(); k++) ?

```
int layer = 1;
while (!q.empty()) {
  for (int k = q.size(); k > 0; k--) {
    string str = q.front(); q.pop();
    for (int i = 0; i < str.size(); i++) {
      for (int j = -1; j <= 1; j+=2) {
        string nstr = str;
        nstr[i] = ((str[i] - '0') + 10 + j) % 10 + '0';
        if (nstr == target) return layer;
        if (!visited.count(nstr) && !blocked.count(nstr)) {
          q.push(nstr);
        }
      }
    }
  }
  layer++;
}
```

These 4 purple lines enumerate through all 8 reachable states from current one.

Using Layered BFS

- One can keep track of the distance from origin source by using a variable during BFS.
- This way, one needs need to remember that a state has been reached.
- Using a for loop inside bfs loop, which goes over all nodes of a certain distance.

Another Version of Open the Lock (Using integers)

```
class Solution {
  const int bases[4] = {1000,100,10,1};
  int str_to_int(string s) {
    return (s.at(0)-'0') * 1000 + (s.at(1)-'0') * 100
          + (s.at(2)-'0') * 10 + s.at(3)-'0';
  }
  int move(int x, int p, int u) {
    int digit = (x / bases[p]) % 10;
    if (u==-1 && digit==0)     u=9;
    else if (u==1 && digit==9) u=-9;
    return x + u * bases[p];
  }
```

Open The Lock : Continued

```
public:
  Solution() {
    std::ios::sync_with_stdio(false);
    cin.tie(nullptr);
  }
  int openLock(vector<string>& deadends, string target) {
    vector<int>  d(10000, 0);
    queue<int> q;
    int  targetNum = str_to_int(target);
    for (int i=0; i<deadends.size(); i++)
      d[str_to_int(deadends[i])] = -1;
    if (d[0] == 0)
      q.push(0);
```

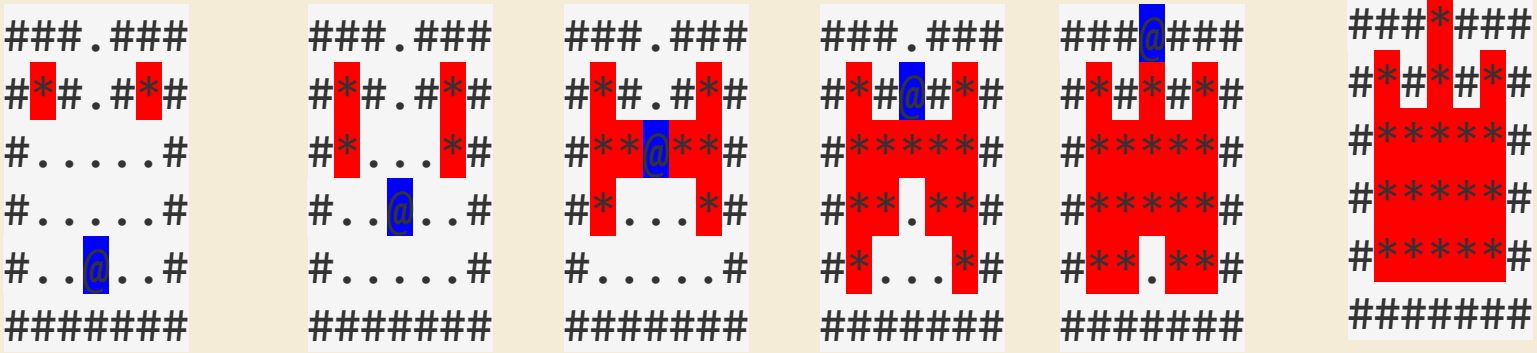
```
    while (! q.empty()) {
      int x = q.front();
      q.pop();
      int nl = d[x] + 1;
      for (int i=0; i<4; i++) for (int j=-1; j<=1; j+=2) {
        int y = move(x, i, j);
        if (y == targetNum) {
          return nl;
        }
        if (d[y] == 0) {
          d[y] = nl;
          q.push(y);
        }
      };
    };
    return -1;
  }
};
```

Kattis: Fire (Optional)

From Benelux Algorithm Programming Contest (BAPC) 2012

You are trapped in a building consisting of open spaces and walls. Some places are on fire. At each second, the fire will spread to all open spaces sharing an edge with . Walls will never catch fire To run to any of the four open spaces adjacent to you takes you exactly one second. You cannot run through a wall or into an open space that is on fire or is just catching fire, but you can run out of an open space at the same moment it catches fire.
Given a map of the building, decide how fast you can exit the building.

Fire: An Illustration



0	1	2	3	4	5

Kattis: Fire

- Need to keep track of fire status and person status
- Easiest solution is to use two separate 2D arrays
- First use BFS to compute the time that fire reaches each cell
- Then use BFS to see where one could go

LeetCode Shortest Bridge (Required)

In a given 2D binary array A, there are two islands. Now, we may change 0s to 1s so as to connect the two islands together to form 1 island. Return the smallest number of 0s that must be flipped. (It is guaranteed that the answer ≥ 1.)

1. `1 <= A.length = A[0].length <= 100`
2. `A[i][j] == 0 or A[i][j] == 1`

Shortest Bridge Example

- What should be the output for the map to the left?
- 2.
- How to compute the answer?
- Bruteforce: Use floodfill to find 2 islands, then for each cell in first and each cell in second, compute manhattan distance. Keep the smallest one.
- Complexity?
- O(N^2), where N is number of cells.

Shortest Bridge

- Find first island and floodfill all cells of the island to have a different value (e.g., 2).
- During the floodfill DFS, for each discovered 0 cell, add to the queue, and mark as having distance 1 from the first island (e.g., using value 3).
- Conduct BFS using the queue, until seeing a cell a cell that has value 1 (belonging to the 2nd island)