Homework #1

RELEASEDATE: 2025/10/13

DUEDATE:2025/11/03, 23:59 on iLearning 3.0

1. *You need to submit your code to the designated place on iLearning 3.0.*
2. *As for coding, C and C++ are allowed.*
3. *Discussions on course materials and homework solutions are encouraged. But you should write the final solutions alone and understand them fully. Books, notes, and Internet resources can be consulted, but not copied from.*
4. *For each question, there will be a testcase (e.g. the testcase for question 1 is testcase1.txt) Please note that the testcase you received may not necessarily to be the same as the testcase used for grading. Please consider edge cases.*
5. *Let your program read the corresponding testcase and print the result to a corresponding text file (e.g. the output for question 2 should be output2.txt).*
6. *For each question, we have provided a corresponding answer (e.g assume you're solving problem 3, the corresponding answer is ans3.txt), and your output should be same as answer.*
7. *For each question, your program file should be named DS{problem number}_{student ID}.c. (e.g. assume your student ID is 71140561387 and you're solving question 3, your program file should be DS3_7114056138.c)*
8. *Each question is scored independently, but partial credit is not awarded; full credit is given only for complete correctness.*
9. *If you have any questions, please feel free to email the TA at any time.*

Teaching Assistant:

- Name:
- Email: g114056138@mail.nchu.edu.tw
- Lab: WCCCLab (S901)

1 Check the Order

In a logistics warehouse, goods must arrive in a fixed order: 1, 2, 3, …, N. According to the company's regulations, every arriving item must first be placed on the storage rack and registered before it can be shipped out.

However, due to limited warehouse space, when a new item is placed on the rack, the existing items must be pushed deeper inside. As a result, only the outermost item can be taken out first, and items stored deeper can only be retrieved after the ones in front of them are removed.

Now, the logistics company wants to load the goods according to a specific "shipping order." You must determine whether this shipping order can be achieved under the above rules.

Input

The first line of input gives the number of test cases, T ($1 \leq T \leq 50$). Then T test cases follow each described in the following way. The first line of each test case contains the integer N described above. For each of the next lines of the test case there is a permutation of 1, 2, … , N.

Output

The output contains the lines corresponding to the lines with permutations in the input. A line of the output contains ‘Yes’ if it is possible to ship out goods in the order required on the corresponding line of the input. Otherwise, it contains ‘No’.

Sample Input

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

Sample Output

```
Yes
No
```

2 The Most Powerful Incantation

In an ancient kingdom, the royal library preserves many mysterious scrolls. It is said that a scroll reveals its true power only when the text on it can be divided into segments called incantations.

An incantation follows a strict rule: its characters must read the same from left to right and from right to left. For example, “girafarig” is an incantation, while “magic” is not.

The librarian tells you that any text on a scroll can be divided into several segments such that each segment is an incantation. However, if the text is divided too finely, the scroll’s power will be weakened. Only by splitting the text into the minimum number of incantation segments can its full magic be unleashed.

Examples:

1. The scroll “girafarig” is itself an incantation, so it only needs to be split into 1 segment.

2. The scroll “magic” does not contain any non-trivial incantations, so it must be split into (m, a, g, i, c), for a total of 5 segments.

3. The scroll “ppap” can be split into (p, pap), for a total of 2 segments.

Your task: Given a string written on a scroll, determine the minimum number of incantation segments required to divide it.

Input

Input begins with the number T of test cases. Each test case consists of a single line of between 1 and 1000 lowercase letters, with no whitespace within.

Output

For each test case, output a line containing the minimum number of groups required to partition the input into groups of incantations.

Sample Input

```
3
girafarig
magic
ppap
```

Sample Output

```
1
5
2
```

3 Maximize the Energy

In the futuristic Galactic Racing League, each racer's spacecraft is equipped with a large number of energy chips, whose energy values are recorded as a continuous sequence of digits. However, due to a malfunction in the race control center's transmission system, all the chip energy values have been concatenated into one long string without any separators.

As a technician, you must devise the optimal strategy for distributing the chip energies by splitting this digit string into multiple valid energy chip values, subject to the following rules:

1. Each chip energy value after splitting must not contain leading zeros. However, the single-digit value 0 itself is considered valid.

2. Each chip energy value must fall within the range of a 32-bit signed integer.

Your goal is to determine a way of splitting the string so that the total sum of all chip energies is maximized, thereby granting the racer the greatest possible acceleration and advantage in the competition.

Input

The input begins with an integer T (≤ 500) which indicates the number of test cases followed. Each of the following test cases consists of a string of at most 200 digits.

Output

For each input, print out required answer in a single line.

Sample Input

```
6
1234554321
5432112345
000
121212121212
2147483648
11111111111111111111111111111111111111111111111111111
```

Sample Output

```
1234554321
543211239
0
2121212124
214748372
5555555666
```

4 Minimum Pushes to Exit the Ruin

In the distant future, within an abandoned interstellar ruin, you are a pioneer (represented by 2) who has been stranded because your spaceship has run out of energy. During your exploration, you discover a massive object that can serve as your ship's energy core. To restart the spaceship and escape, you must push this heavy energy core (represented by 3) to the location of your spaceship (represented by 4). The ruin is structured as a vast m × n grid, where each cell may be:

1. Solid ground (0): passable terrain,
2. Collapsed zone (1): completely impassable.

You must proceed carefully, since you cannot walk through the energy core, nor can you step onto collapsed zones. Your movement is limited to the four cardinal directions: upward, downward, leftward, or rightward.

To push the energy core, you must stand on an adjacent square next to it and move in the core's direction. Each push consumes a huge amount of energy, so your goal is to minimize the number of pushes required to move the core to the spaceship.

Your task: Calculate the minimum number of pushes required to move the energy core to the spaceship. If it is impossible, output -1.

Notes:

- Solid ground = 0
- Collapsed zone = 1
- Your starting position = 2
- Energy core = 3
- Spaceship location = 4

You only need to count the number of pushes, not the actual path taken. You cannot walk through the core or collapsed zones, but you may walk over the spaceship’s location.

Input

The first line of input gives the number of test cases, T (1≤T≤50). Then T test

cases follow each described in the following way. The first line of each test case contains m (1≤m≤20) and n (1≤n≤20) separated by a space, which indicates that ruin is a m × n matrix. Then m lines follow each of the n containing row i. Row contains exactly n elements integer separated by a space. j-th number in row i is the element ruin[i][j] of matrix you have to process.

Output

For each input produce one line of output. This line contains an integer which indicates the number of minimum number of pushes to move the box to the target. If there is no way to reach the location of spaceship, return -1.

Sample Input

```
3
6 6
1 1 1 1 1 1
1 4 1 1 1 1
1 0 0 3 0 1
1 0 1 1 0 1
1 0 0 0 2 1
1 1 1 1 1 1
6 6
1 1 1 1 1 1
1 4 1 1 1 1
1 0 0 3 0 1
1 1 1 1 0 1
1 0 0 0 2 1
1 1 1 1 1 1
6 6
1 1 1 1 1 1
1 4 0 0 1 1
1 0 1 3 0 1
1 0 0 0 0 1
1 0 0 0 2 1
1 1 1 1 1 1
```

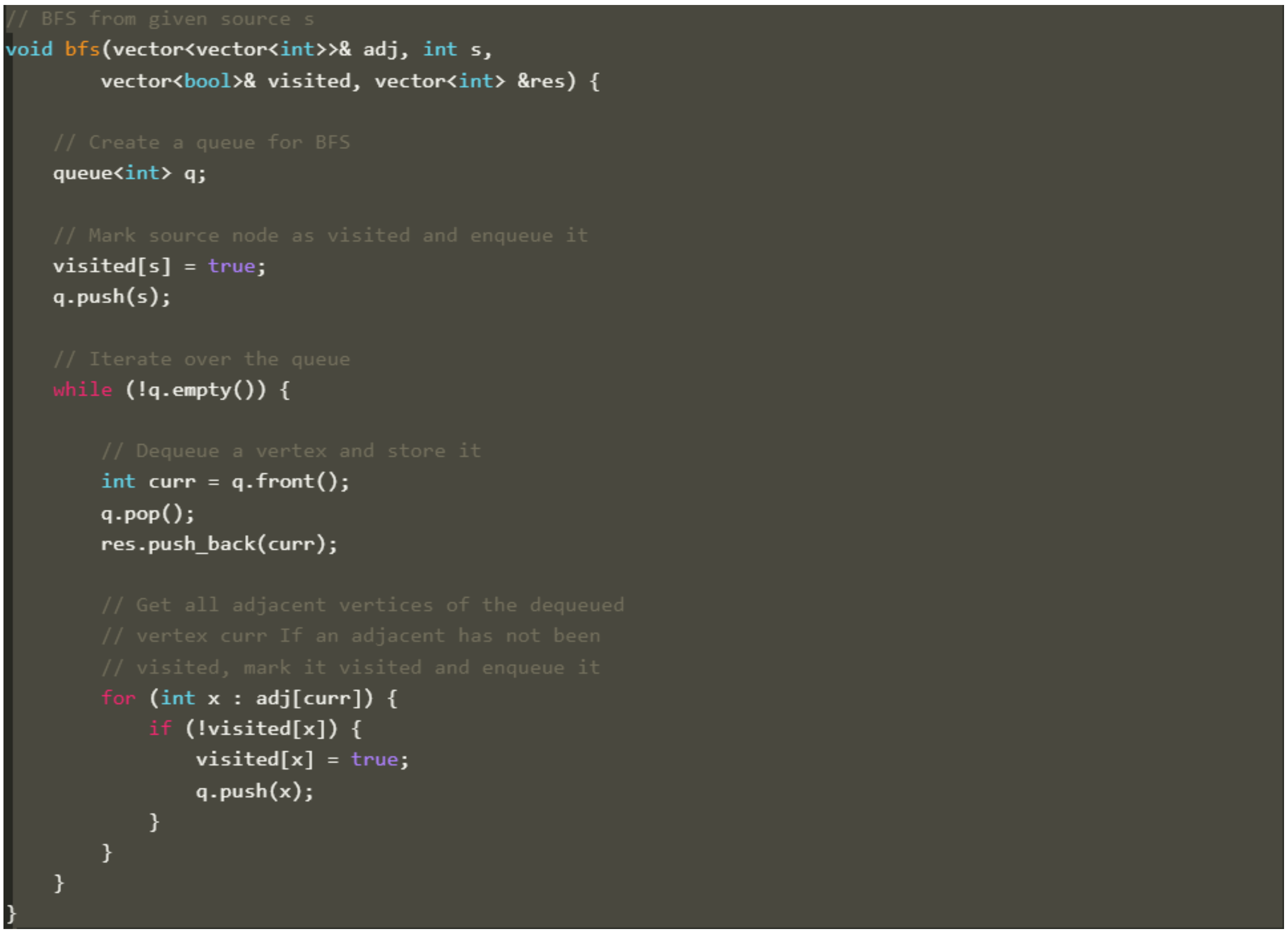
Sample Output

```
3

-1

5
```

Hint: Use Breadth First Search (BFS)



```cpp
// BFS from given source s
void bfs(vector<vector<int>>& adj, int s,
        vector<bool>& visited, vector<int> &res) {

    // Create a queue for BFS
    queue<int> q;

    // Mark source node as visited and enqueue it
    visited[s] = true;
    q.push(s);

    // Iterate over the queue
    while (!q.empty()) {

        // Dequeue a vertex and store it
        int curr = q.front();
        q.pop();
        res.push_back(curr);

        // Get all adjacent vertices of the dequeued
        // vertex curr If an adjacent has not been
        // visited, mark it visited and enqueue it
        for (int x : adj[curr]) {
            if (!visited[x]) {
                visited[x] = true;
                q.push(x);
            }
        }
    }
}
```

Submission File:

Please submit the homework to iLearning 3.0 before the deadline. You need to upload your coding part as a single ZIP compressed file to iLearning 3.0 before the deadline. The zip file should be named DS_1_{student ID}.zip (e.g assume your student ID is 7114056138, your zip file should be DS_1_7114056138.zip). The zip file should contain no directories and only the following items:

- all the source code
- an optional README, anything you want the TAs to read before grading your code