//Problem
You have been hired recently as the Chief Decision Maker (CDM) at a famous parcel delivery company, congratulat
ons! Customers love speedy deliveries of their parcels and you have decided to decrease the time it takes to deliver p
rcels around the world to win customers. You have introduced this idea to the authorities and they have allocated yo
enough budget to build at most one new delivery office.

The world can be divided into an R × C grid of squares. Each square either contains a delivery office or it does not.
ou may pick a grid square that does not already contain a delivery office and build a new delivery office there.

The delivery time of a parcel to a square is 0 if that square contains a delivery office. Otherwise, it is defined as the
inimum Manhattan distance between that square and any other square containing a delivery office. The overall deliv
ry time is the maximum of delivery times of all the squares. What is the minimum overall delivery time you can obta
n by building at most one new delivery office?

Note: The Manhattan distance between two squares (r1,c1) and (r2,c2) is defined as |r1 - r2| + |c1 - c2|, where |*| ope
ator denotes the absolute value.

Input
The first line of the input gives the number of test cases, T. T test cases follow. The first line of each test case contai
s the number of rows R and number of columns C of the grid. Each of the next R lines contains a string of C charact
rs chosen from the set {0, 1}, where 0 denotes the absence of a delivery office and 1 denotes the presence of a delive
y office in the square.

Output
For each test case, output one line containing Case #x: y, where x is the test case number (starting from 1) and y is th
minimum overall delivery time you can obtain after adding at most one additional delivery office.

Limits
Time limit: 15 seconds per test set.
Memory limit: 1GB.
1 ≤ T ≤ 100.
There is at least one delivery office in the initial grid.

Test set 1 (Visible)
1 ≤ R ≤ 10.
1 ≤ C ≤ 10.

Test set 2 (Hidden)
1 ≤ R ≤ 250.
1 ≤ C ≤ 250.

Sample

Input

Output

```
3
3 3
101
000
101
```

```
1 2
11
5 5
10001
00000
00000
00000
10001
```

```
Case #1: 1
Case #2: 0
Case #3: 2
```

In Sample Case #1, you get a minimum overall delivery time of 1 by building a new delivery office in any one of the five squares without a delivery office.

In Sample Case #2, all the squares already have a delivery office and so the minimum overall delivery time is 0. Not you have to add at most one delivery office.

In Sample Case #3, to get a minimum overall delivery time of 2, you can build a new delivery office in any of these quares: (2, 3), (3, 2), (3, 3), (3, 4), or (4, 3). Any other possibility results in a higher overall delivery time than 2.

```
// In the name of God

#include <iostream>
#include <algorithm>
#include <fstream>
#include <vector>
#include <deque>
#include <assert.h>
#include <queue>
#include <stack>
#include <set>
#include <map>
#include <stdio.h>
#include <string.h>
#include <utility>
#include <math.h>
#include <bitset>
#include <iomanip>
#include <complex>

using namespace std;

#define rep(i, a, b) for (int i = (a), i##_end_ = (b); i < i##_end_; ++i)
#define debug(...) fprintf(stderr, __VA_ARGS__)
#define mp make_pair
#define x first
#define y second
#define pb push_back
```

```
#define SZ(x) (int((x).size()))
#define ALL(x) (x).begin(), (x).end()

template<typename T> inline bool chkmin(T &a, const T &b) { return a > b ? a = b, 1 : 0; }
template<typename T> inline bool chkmax(T &a, const T &b) { return a < b ? a = b, 1 : 0; }
template<typename T> inline bool smin(T &a, const T &b)   { return a > b ? a = b : a;    }
template<typename T> inline bool smax(T &a, const T &b)   { return a < b ? a = b : a;    }

typedef long long LL;

const int N = (int) 355, mod = (int) 0;
int d[N][N];
int q[N * N];
int dx[] = {0, -1, 0, 1};
int dy[] = {1, 0, -1, 0};
string s[N];
void bfs(int n, int m) {
 int h = 0, t = 0;
 memset(d, 63, sizeof d);
 for (int x = 0; x < n; ++x)
  for (int y = 0; y < m; ++y)
   if (s[x][y] == '1')
    d[x][y] = 0, q[t++] = x * m + y;
 while (h != t) {
  int v = q[h++];
  int x = v / m, y = v % m;
  for (int d = 0; d < 4; ++d) {
   int nx = x + dx[d], ny = y + dy[d];
   if (nx >= 0 && nx < n && ny >= 0 && ny < m) {
    if (::d[nx][ny] > ::d[x][y] + 1) {
     ::d[nx][ny] = ::d[x][y] + 1;
     q[t++] = nx * m + ny;
    }
   }
  }
 }
}
int sum[4 * N][4 * N];
int check(int lim, int n, int m) {
 for (int x = 0; x < 2 * N; ++x) for (int y = 0; y < 2 * N; ++y) sum[x][y] = 0;
 int cnt = 0;
 for (int x = 0; x < n; ++x)
  for (int y = 0; y < m; ++y) {
   if (d[x][y] > lim) {
    ++cnt;
    int lx = x - lim, rx = x + lim;
    int lnx = lx + y;
    int lny = lx - y + m;
    int rnx = rx + y;
    int rny = rx - y + m;
    lny = max(lny, 0);
    lnx = max(lnx, 0);
    rny = max(rny, 0);
    rnx = max(rnx, 0);
    sum[lnx][lny]++;
    sum[rnx + 1][lny]--;
    sum[lnx][rny + 1]--;
    sum[rnx + 1][rny + 1]++;
   }
  }
 for (int x = 0; x < 2 * N; ++x)
  for (int y = 0; y < 2 * N; ++y) {
   int a = (x + y - m) / 2, b = x - a;
   if (x) sum[x][y] += sum[x - 1][y];
   if (y) sum[x][y] += sum[x][y - 1];
   if (x && y) sum[x][y] -= sum[x - 1][y - 1];
   if ((x & 1) == ((y + m) & 1) && a >= 0 && a < n && b >= 0 && b < m && sum[x][y] == cnt) {
    return 1;
   }
  }
 if (cnt == 0) return 1;
 return 0;
}
int main() {
 int tc;
 cin >> tc;
 for (int tt = 1; tt <= tc; ++tt) {
  cout << "Case #" << tt << ": ";
  int n, m;
  cin >> n >> m;
  for (int j = 0; j < n; ++j)
   cin >> s[j];
  bfs(n, m);
  int bl = -1, br = n + m + 1;
  while (bl < br - 1) {
   int bm = bl + br >> 1;
   if (check(bm, n, m)) {
    br = bm;
   } else {
    bl = bm;
   }
  }
  cout << br << '\n';
 }
}
```