Problem
Two companies, Apricot Rules LLC and Banana Rocks Inc., are sharing the same datacenter. The datacenter is a ma rix of R rows and C columns, with each cell containing a single server tower. Each tower contains intellectual proper y belonging to exactly one of the two companies.

At first, they built walls on the edges between cells assigned to different companies. This allowed orthogonally adjac nt cells belonging to the same company to remain connected. Also, two cells x and y are considered connected if x is connected to a cell that is, directly or indirectly, connected to y. With this definition, it was possible that two cells as igned to the same company were not connected, which was unacceptable.

Both companies agreed to build narrow hallways running through cell corners that allow two diagonally adjacent cel s to be connected directly. Let us write (i, j) to represent the cell at row i and column j. At most one narrow hallway an be built through any given vertex, which means either (i, j) and (i + 1, j + 1) can be connected, or (i + 1, j) and (i, + 1) can be connected, or neither pair, but not both. Of course, only hallways between cells assigned to the same co pany can be built.

Given a matrix where each cell is labeled A or B depending on which company it is assigned to, find a way to add c nnections through diagonal adjacencies such that all As are connected and all Bs are connected.

Input
The first line of the input gives the number of test cases, T. T test cases follow. Each test case begins with one line c ntaining two integers R and C, the number of rows and columns of the matrix representing the datacenter. Then, ther are R more lines containing C characters each. The j-th character on the i-th of these lines Mi,j is either A or B, indi ating which company owns the cell at (i, j).

Output
For each test case, first output one line containing Case #x: y, where x is the test case number (starting from 1) and y is IMPOSSIBLE if there is no way to assign the diagonal connections such that the A cells are connected and the B ells are connected, or POSSIBLE otherwise. Then, if you output POSSIBLE, output R - 1 more lines of C - 1 charac ers each. These characters must correspond to a valid arrangement as described in the statement above. The j-th char cter of the i-th of those lines must be \ if cells (i, j) and (i + 1, j + 1) are to be connected, / if cells (i + 1, j) and (i, j +) are to be connected, or . if neither pair is to be connected.

Limits
$1 \leq T \leq 100$.
Time limit: 20 seconds per test set.
Memory limit: 1GB.
$2 \leq C \leq 100$.
Mi,j = uppercase A or uppercase B, for all i and j.
Mi,j = uppercase A for at least one pair of i and j.
Mi,j = uppercase B for at least one pair of i and j.

Test set 1 (Visible)
$2 \leq R \leq 4$.

Test set 2 (Hidden)
$2 \leq R \leq 100$.

Sample

Input

Output

```
3
2 2
AB
BA
2 3
AAB
ABB
3 4
BBAA
BABA
BBAA


Case #1: IMPOSSIBLE
Case #2: POSSIBLE
..
Case #3: POSSIBLE
//\
.//
```

In Sample Case #1, the pair of A cells and the pair of B cells need to be connected, but since both connections woul
 have to cross the same vertex, at most one of the connections can exist.

In Sample Case #2, the cells are already connected in the required way in the input, so no additional connections are ecessary. Note that you can add unnecessary valid connections, so another valid answer would be //, but \. would be rong.

In Sample Case #3, there are also multiple solutions, one of which is displayed.

Solution:

```
#ifdef _MSC_VER
#define _CRT_SECURE_NO_WARNINGS
#endif

#include <bits/stdc++.h>

using namespace std;

typedef long long int64;
typedef unsigned long long uint64;
#define two(X) (1<<(X))
#define twoL(X) (((int64)(1))<<(X))
#define contain(S,X) (((S)&two(X))!=0)
#define containL(S,X) (((S)&twoL(X))!=0)
const double pi=acos(-1.0);
const double eps=1e-11;
template<class T> inline void ckmin(T &a,T b){if(b<a) a=b;}
```

```
template<class T> inline void ckmax(T &a,T b){if(b>a) a=b;}
template<class T> inline T sqr(T x){return x*x;}
typedef pair<int,int> ipair;
#define SIZE(A) ((int)A.size())
#define LENGTH(A) ((int)A.length())
#define MP(A,B) make_pair(A,B)
#define PB(X) push_back(X)
#define FOR(i,a,b) for(int i=(a);i<(b);++i)
#define REP(i,a) for(int i=0;i<(a);++i)
#define ALL(A) A.begin(),A.end()
using VI=vector<int>;
using VS=vector<string>;

VI f;

#define ID(x,y) ((x)*(m)+(y))

int getf(int p)
{
 return (f[p]<0?p:(f[p]=getf(f[p])));
}
void mergef(int a,int b)
{
 a=getf(a);
 b=getf(b);
 if (a!=b) f[a]=b;
}
pair<bool,VS> solve(VS a)
{
 int n=SIZE(a);
 int m=LENGTH(a[0]);
 f=VI(n*m,-1);
 REP(x,n) REP(y,m)
 {
  if (x+1<n && a[x][y]==a[x+1][y]) mergef(ID(x,y),ID(x+1,y));
  if (y+1<m && a[x][y]==a[x][y+1]) mergef(ID(x,y),ID(x,y+1));
 }
 VS ret(n-1,string(m-1,'.'));
 for (char key='A';key<='B';key++) REP(x,n-1) REP(y,m-1)
 {
  if (ret[x][y]=='.' && a[x][y]==key && a[x+1][y+1]==key && getf(ID(x,y))!=getf(ID(x+1,y+1)))
  {
   mergef(ID(x,y),ID(x+1,y+1));
   ret[x][y]='\\';
  }
  if (ret[x][y]=='.' && a[x+1][y]==key && a[x][y+1]==key && getf(ID(x+1,y))!=getf(ID(x,y+1)))
  {
   mergef(ID(x+1,y),ID(x,y+1));
   ret[x][y]='/';
  }
 }
 VI cnt(2);
 REP(x,n) REP(y,m) if (f[ID(x,y)]<0) ++cnt[a[x][y]-'A'];
 return MP(cnt[0]<=1 && cnt[1]<=1,ret);
}

int main()
{
#ifdef _MSC_VER
 freopen("input.txt","r",stdin);
#endif
 std::ios::sync_with_stdio(false);
 int testcase;
 cin>>testcase;
 for (int case_id=1;case_id<=testcase;case_id++)
 {
  int n,m;
  cin>>n>>m;
  VS a(n);
  REP(i,n) cin>>a[i];
  auto ret=solve(a);
  printf("Case #%d: ",case_id);
  if (ret.first)
  {
   printf("POSSIBLE\n");
   REP(i,SIZE(ret.second)) printf("%s\n",ret.second[i].c_str());
  }
  else
  {
   printf("IMPOSSIBLE\n");
  }
 }
 return 0;
}
```