T	M	M	M	T	T	T
M	M	M	M	T	M	M
T	T	M	T	T	M	T
T	M	M	T	M	M	M
T	T	T	T	T	T	M
T	T	T	T	T	T	M

L : 1 H : 5

< r_1 r_2 >

< c_1 c_2 >

I

$|(r_2 - r_1) * (c_2 - c_1)| \geq H$

II

contains TYPE M / T

count(M) ∧ count(T) ≥ L

	0	1	2	3	4
0	T	T	T	T	T
1	T	M	M	M	T
2	T	T	T	T	T

$r_1 = 0$ $c_1 = 0$

$r_2 = 0$ $c_2 = 0$

countT(r_1, r_2) = 2

countM(r_1, r_2) = 0

2 T

3 M

T	M	M	M	T	T	T
M	M	M	M	T	M	M
T	T	M	T	T	M	T
T	M	M	T	M	M	M
T	T	T	T	T	T	M
T	T	T	T	T	T	M

$$\begin{pmatrix} 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

Given a Matrix $A = [a_{ij}]$ of binary integers for all $i \in \{1, \dots, m\}$ and $j \in \{1, \dots, n\}$.
Non negative integers L, H.

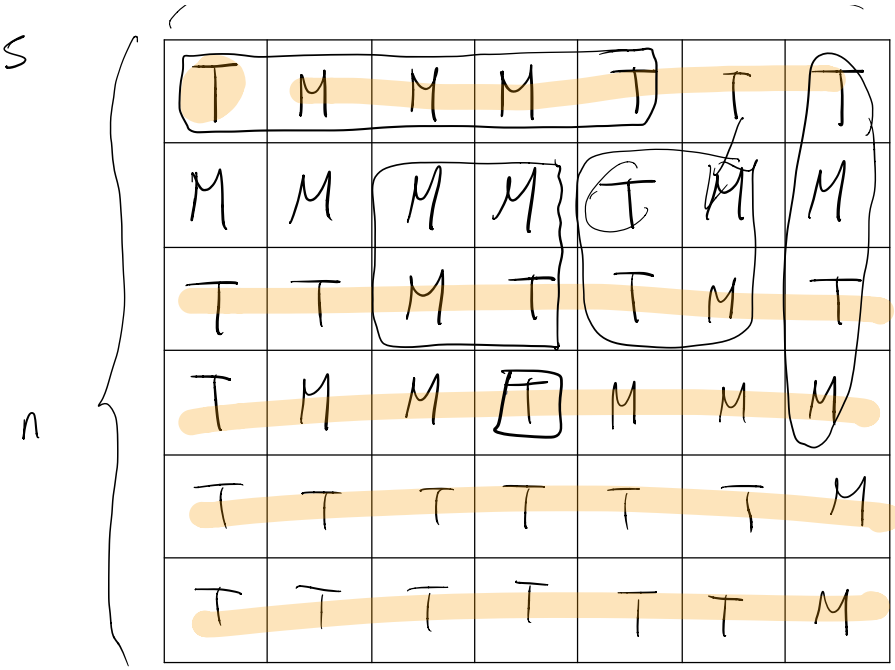
Find n disjoint sets S of $\{1, \dots, m\} \times \{1, \dots, n\}$ such that

$$\bigcup_{j=1}^{n} S_j = \{1, \dots, m\}$$

$$\sum_{i \in S_j} a_{ij} \leq H$$

$$\Sigma$$

H = 5



n

L T

L M

ArrayList < Celle > libere;
Foreach int i in m
  Foreach int j in n
check condiz ← check if i,j in libere
  .yes : slice ← generate slice
  .no: continue
check slice.area ≤ H
  . no: return previous recursion step
  . yes: check slice. # tomato ≥ L
    . yes : check slice. # mush ≥ L
      . yes : libere ← libere - slice
      . no : ArrayList<Celle> adjacent ← adj(slice)
        ArrayList<Celle> posM ← FindM(adjacent)
        check posM.size == 0
          . no : For cell in posM

---

prendiMpiùVicinaAccettabile(); // ritorna una Slice con le M accettabile più vicine NULLABLE CONTA ANCHE aggiunte alla current slice

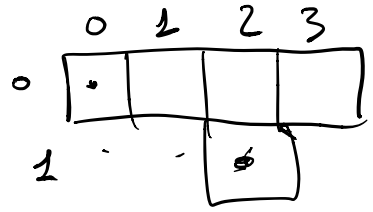
Alg:

```
PI ← getInitialPIC();
int x,y = PI.x,y;
int countH, countT;

visitedC = 0
currentCell = PI
ArrayList<Slices> solution = new ArrayList<>();
while (visitedC < n*m) {
    visitedC++;
    if(!currentCell.isVisited()) {
      newSlice = generateCorrectSlice(currentCell);
        if (newSlice ≠ NULL) {
            solution.add( newSlice)
        }

    }else {
      continue;
    }
    currentCell = nextCell(currentCell);
}
```

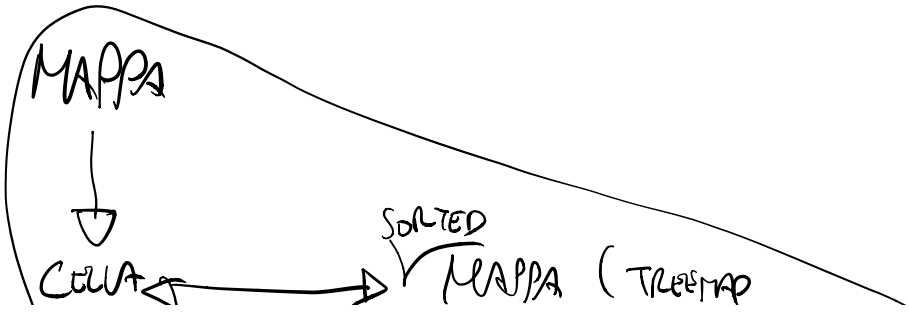
```
Slice generateCorrectSlice (currentCell){
    currentSlice = generateSlice(currentCell);
while(currentSlice.countM < L ∧ currentSlice.countT < L){
if (currentSlice.countM < L)
    M()
if (currentSlice.countT < L)
    T()
}
```

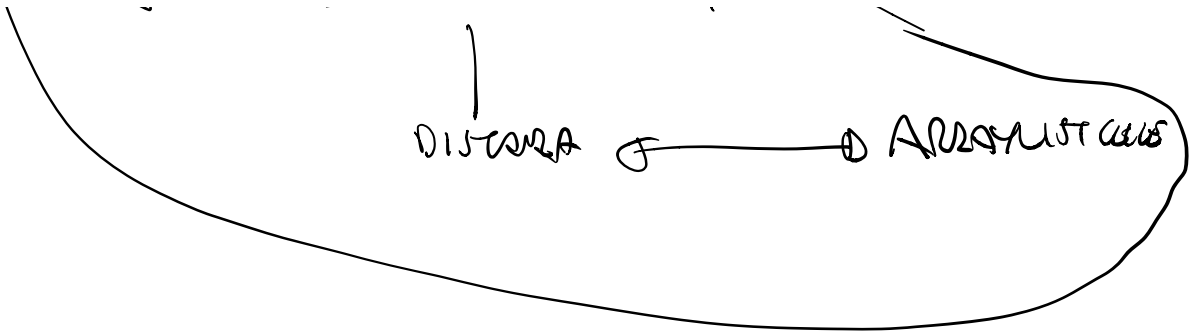
```
M()
    currentSlice ← prendiMpiùVicinaAccettabile. ();
    if (currentSlice.isNull()) {
            current.visited = true;
            return NULL;
    }

T()
    currentSlice ← prendiTpiùVicinaAccettabile. ();
    if (currentSlice.isNull()) {
            current.visited = true;
            return NULL;
    }
    return currentSlice;

Slice prendiMpiùVicinaAccettabile. (Slice) {
    while (!nonTrovato) {
        currentMap ← GMap.lookUp(slice.currentCell())
        for each (in currentMap.keys){// distanze
            closeCells ← currentMap.lookup(i);
            For each Cell in closeCells {
                if (c.type == T)
                    continue;
                else
                    sliceToCheck ← Slice.addCell(c)
                if (sliceToCheck.isAcceptable()
                    Foreach Cell c in sliceToCheck {c.setOccupied}
                    return sliceToCheck;
    }
```

```
    return NULL
}

public addCell(Cell c) {  // Slice has the 4 numbers
    if (c.x < this.c1) {
        this.c1 = c.x;
```



PER OGNI CELLA (10^6) ABBIAMO UNA MAPPA CHE MAPPA UN VALORE DELLA DISTANZA A PIÙ CELLE.




DISTANZA
ARROTONDAMENTO