```
(*
The input consists of two things:
  text:
  write "-" for a black field and a "x" for a tree. All other characters are ignored.

    numText: Number of tents. First for all columns,
then a comma, then for all rows *)

minusAndX = "
------xx-x---x-----x-
--------x------------
-x--x----x-----x-x---
------x------------x-
x-xx----xx-----x-x-x-
-x----x----x---------
x-xx----x--x---x----x
-x--x---------x-xx---
x----------x--------x
---x----xx-x-x-------
x--x---------x--x--x-
-------x---------x-x-
--------x-x---------x
--x--------x----x----
----x--x----x-----x-x
x----x----xx---x-----
---x--x----------x---
x-----x-------x------
-x---x----x-x-x------
--xx-------x----x---x
-x---x--x---x---x---x
";
numText = "827171634555262543428,362365463617253453628";


input = StringReplace[StringCases[minusAndX, x_ /; x == "-" || x == "x"], "x" → "b"];


(* The whole programm is a "If"-Command, so that Abort[] works correctly. *)
If[

 (* Test, if numText has the corect form *)
 StringCount[numText, ","] ≠ 1, Print["numText has to contain exactly one comma."];
 Abort[],
 (* numInCol contains all numbers before the comma in
  numText. It says for each column, how many tents there are. *)
 numInCol = ToExpression[StringCases[
    StringCases[numText, xx__ ~~ "," → xx]〚1〛, DigitCharacter]];
 (* numInRow contains all numbers after the comma in numText. *)
 numInRow = ToExpression[
   StringCases[StringCases[numText, "," ~~ xx__ → xx]〚1〛, DigitCharacter]];
 (* n is the number of rows. *)
 n = Length[numInRow];
 (* m is the number of columns. *)
 m = Length[numInCol];
 (* Some input mistakes *)
```

```
If[Union[input] =!= {"-", "b"}, Print["input must contain only '-' and 'b'."];
 Abort[]];
If[Length[input] ≠ n * m, Print["The number of fields (", Length[input],
  "), does not coincide with the number of rows times the number of columns (",
  n * m, ")."];
 Abort[]];
treesTotal = Count[input, "b"];
If[Total[numInRow] ≠ treesTotal,
 Print["The number of tents in all rows (", Total[numInRow],
  ") does not coincide with the total number of trees (", treesTotal, ")."];
 Abort[]];
If[Total[numInCol] ≠ treesTotal,
 Print["The number of tents in all columns (", Total[numInCol],
  "), does not coincide with the total number of trees (", treesTotal, ")."];
 Abort[]];


(* Maximum number of tents for each
 row/culumn (independant of numInRow/numInCol) *)
maxInRow = Table[m, {n}];
maxInCol = Table[n, {m}];

(* Number of tents aldeady built *)
builtInRow = Table[0, {n}];
builtInCol = Table[0, {m}];

(* If a row/column already contains its required number of tents,
it is no longer considered. *)
doneRow = Table[0, {n}];
doneCol = Table[0, {m}];

(* For black fields, neighbours is the number of neighbour trees,
that don't have a fixed tent yet. *)
neighbours = Table[0, {n}, {m}];
checkTentAnz = Table[0, {n}, {m}];

(* make input a matrix *)
mat = Partition[input, m];
(*
Each entry in mat can be one of the following characters \:
 "-" black
 "b" Tree, that still needs a tent
 "ψ" Tree, that has a fixed tent
 "." green
 "Δ" Tent, that has a fixed tree (the number of Δ and ψ is always equal )
 "z" Tent, that needs a tree
 "g" This field or one neighbour field containing also a "g"
 (horizontal neighbour for mazZA / vertikal for matCol) is a tent,
if the row/column contains its maximum number of tents
 "u" green, if the row (for matRow) or column
 (for matCol) contains its maximum number of tents
 "h" this corresponds to another entry "h". Exactly one has to contain a tent,
the other one contains grass.
 *)
```

```
(* matRow is mat with row analysis (see below) *)
(* matCol is mat with column analysis *)
matRow = mat;
matCol = mat;

(* rowUpdate updates the matrix matRow. This does,
for example: m[[1]]={----b-b-----} → mTmpZ[[1]]={ggggb0b0u0u0}*)
(* In grass blocks of even length,
"-" is replaced by "g". In grass blocks of odd length,
"-" is replaced by "u", but only at even positions.*)


rowUpdate = z ↦ Module[{temp = mat⟦z⟧, mode = 0},
   (* If a row already contains its required number of tents,
   it is not considered. *)
   If[doneRow⟦z⟧ == 0,

    (* Count the tents built in that row. *)
    builtInRow⟦z⟧ = Count[temp, x_ /; x == "z" ∨ x == "Δ"];

    (* Replace each second "-" by a "u" *)
    Do[If[temp⟦j⟧ == "-" ∧ temp[[j - 1]] == "-", temp⟦j⟧ = "u"], {j, 2, m}];

    (* Now go backwards. Leave all grass blocks of odd length
     as they are. Replace all grass block of even length by "g". *)
    (* mode 0: nothing - 1: in a grass block of even length - 2:
     is a grass block of odd length *)
    Do[
     Which[mode == 0,
      If[temp[[m - j]] == "u", mode = 1;
       temp[[m - j]] = "g"];
      If[temp[[m - j]] == "-", mode = 2],
      mode == 1,
      If[MemberQ[{"-", "u"}, temp[[m - j]]], temp[[m - j]] = "g", mode = 0],
      mode == 2,
      If[! MemberQ[{"-", "-"}, temp[[m - j]]], mode = 0]
     ], {j, 0, m - 1}
    ];

    (* For the constellation "-b-",
    where both of the  "-" have no other neighbour tree, maxInRow[[z]] has
     to be decremented. In that case replace both of the "-" by a "h". *)
    Do[If[
      temp⟦j⟧ == "b" ∧
       temp⟦j - 1⟧ == "-" ∧
       temp⟦j + 1⟧ == "-" ∧
       neighbours⟦z, j - 1⟧ == 1 ∧
       neighbours⟦z, j + 1⟧ == 1,
      temp[[j - 1]] = "h";
      temp[[j + 1]] = "h";
     ],
     {j, 2, m - 1}];
```

```
      (* The remaining "-" give the maximum number of tents,
      that can be built in that row. The "g" and "h" and
       counted half (in every other such field may be a tent) *)
      maxInRow⟦z⟧ = builtInRow⟦z⟧ + Count[temp, "-"] +
        Count[temp, "g"] / 2 + Count[temp, "h"] / 2;

      matRow⟦z⟧ = temp;

      (* If num and max coincide, run maxBuildRow *)
      If[numInRow⟦z⟧ == maxInRow⟦z⟧, maxBuildRow[z]];

      (* If they coincide up to one, run almostMaxBuildRow *)
      If[numInRow⟦z⟧ + 1 == maxInRow⟦z⟧, almostMaxBuildRow [z]];

      (* If num and built coincide, run reachedRow *)
      If[numInRow⟦z⟧ == builtInRow⟦z⟧, reachedRow[z]];
     ]
    ];



  (* The same as rowUpdate for columns: *)
  colUpdate = s ↦ Module[{temp = mat⟦All, s⟧, mode = 0},
     If[doneCol⟦s⟧ == 0,
      builtInCol⟦s⟧ = Count[temp, x_ /; x == "z" ∨ x == "Δ"];
      Do[If[temp⟦i⟧ == "-" ∧ temp[[i - 1]] == "-", temp⟦i⟧ = "u"], {i, 2, n}];
      Do[
       Which[mode == 0,
        If[temp[[n - i]] == "u", mode = 1;
         temp[[n - i]] = "g"];
        If[temp[[n - i]] == "-", mode = 2],
        mode == 1,
        If[MemberQ[{"-", "u"}, temp[[n - i]]], temp[[n - i]] = "g", mode = 0],
        mode == 2,
        If[! MemberQ[{"-", "-"}, temp[[n - i]]], mode = 0]
       ], {i, 0, n - 1}
      ];
      Do[If[
        temp⟦i⟧ == "b" ∧
         temp⟦i - 1⟧ == "-" ∧
         temp⟦i + 1⟧ == "-" ∧
         neighbours⟦i - 1, s⟧ == 1 ∧
         neighbours⟦i + 1, s⟧ == 1,
        temp[[i - 1]] = "h";
        temp[[i + 1]] = "h";
       ],
       {i, 2, n - 1}];
      maxInCol⟦s⟧ =
       builtInCol⟦s⟧ + Count[temp, "-"] + Count[temp, "g"] / 2 + Count[temp, "h"] / 2;
      matCol⟦All, s⟧ = temp;
      If[numInCol⟦s⟧ == maxInCol⟦s⟧, maxBuildCol[s]];
      If[numInCol⟦s⟧ + 1 == maxInCol⟦s⟧, almostMaxBuildCol [s]];
      If[numInCol⟦s⟧ == builtInCol⟦s⟧, reachedCol[s]];
     ]
  ];



(* Run this, if column z has to contain the
 maximum number of tents to fulfill the column number. *)
maxBuildRow = z ↦ Module[{temp = 0},
    (*Print["zeileMax ",z];*)
    (* Replace all "-" by tents. *)
    Do[
     If[matRow⟦z, j⟧ == "-", mat⟦z, j⟧ = "z"; checkTentNeighbours[z, j]];
     (* Replace black neighbours of "g" by "." (green). This can be done,
     since "g" means that either this or the next field in the row is a tent. *)
     If[matRow⟦z, j⟧ == "g",
      If[z ≠ n ∧ mat⟦z + 1, j⟧ == "-", mat⟦z + 1, j⟧ = "."];
      If[z ≠ 1 ∧ mat⟦z - 1, j⟧ == "-", mat⟦z - 1, j⟧ = "."]
     ],
     {j, m}
    ]
  ];

(* The same for columns: *)
maxBuildCol = s ↦ Module[{temp = 0},
    (*Print["spalteMax ",z];*)
    Do[
     If[matCol⟦i, s⟧ == "-", mat⟦i, s⟧ = "z"; checkTentNeighbours[i, s]];
     If[matCol⟦i, s⟧ == "g",
      If[s ≠ m ∧ mat⟦i, s + 1⟧ == "-", mat⟦i, s + 1⟧ = "."];
      If[s ≠ 1 ∧ mat⟦i, s - 1⟧ == "-", mat⟦i, s - 1⟧ = "."]
     ],
     {i, n}
    ]
  ];



(* Run this, if column z has to contain the maximum
 number of tents MINUS ONE to fulfill the column number. *)
almostMaxBuildRow = z ↦ Module[{temp = 0},
    Do[
     (* Replace "-" in the row above and below by ".",
     if the "-" is diagonal to two "-" in the row considered. This can be done
      since one of the two "-" in the row considered has to be a tent. *)
     If[matRow⟦z, j - 1⟧ == "-" ∧ matRow⟦z, j + 1⟧ == "-",
      If[z > 1 ∧ mat⟦z - 1, j⟧ == "-", mat⟦z - 1, j⟧ = "."];
      If[z < n ∧ mat⟦z + 1, j⟧ == "-", mat⟦z + 1, j⟧ = "."];
     ],
     {j, 2, m - 1}
    ]
  ];


(* The same for columns: *)
almostMaxBuildCol = s ↦ Module[{temp = 0},
    Do[
     If[matCol⟦i - 1, s⟧ == "-" ∧ matCol⟦i + 1, s⟧ == "-",
      If[s > 1 ∧ mat⟦i, s - 1⟧ == "-", mat⟦i, s - 1⟧ = "."];
      If[s < m ∧ mat⟦i, s + 1⟧ == "-", mat⟦i, s + 1⟧ = "."];
     ],
     {i, 2, n - 1}
    ]
   ];


(* Run this, if row z has reached its required number of tents. *)
reachedRow = z ↦ Module[{temp = 0},
    (* Replace all "-" by "." *)
    Do[
     If[mat⟦z, j⟧ == "-",
      mat⟦z, j⟧ = "."
     ],
     {j, m}
    ];
    doneRow⟦z⟧ = 1;
   ];


(* The same for columns: *)
reachedCol = s ↦ Module[{temp = 0},
    Do[
     If[mat⟦i, s⟧ == "-",
      mat⟦i, s⟧ = "."
     ],
     {i, n}
    ];
    doneCol⟦s⟧ = 1;
   ];

(* If a tent is built at position {i,j},
all neighbour fields (also diagonal), that aren't tents or trees, become green. *)
checkTentNeighbours = {i, j} ↦ Module[{temp = 0},
    (*Print["checkTentNeighbours ",i," ",j];*)
    If[FreeQ[{"z", "Δ"}, mat⟦i, j⟧],
     Print["Abort: checkTentNeighbours[", i, j, "] run, although there is no tent."];
     Abort[]];

    (* In the row below *)
    If[i ≠ n,
     If[j ≠ 1 ∧ mat⟦i + 1, j - 1⟧ == "-", mat⟦i + 1, j - 1⟧ = "."];
     If[mat⟦i + 1, j⟧ == "-", mat⟦i + 1, j⟧ = "."];
     If[j ≠ m ∧ mat⟦i + 1, j + 1⟧ == "-", mat⟦i + 1, j + 1⟧ = "."];
    ];

    (* In the same row *)
    If[j ≠ 1 ∧ mat⟦i, j - 1⟧ == "-", mat⟦i, j - 1⟧ = "."];
    If[j ≠ m ∧ mat⟦i, j + 1⟧ == "-", mat⟦i, j + 1⟧ = "."];

    (* In the row above *)
    If[i ≠ 1,
    If[j ≠ 1 ∧ mat⟦i - 1, j - 1⟧ == "-", mat⟦i - 1, j - 1⟧ = "."];
    If[mat⟦i - 1, j⟧ == "-", mat⟦i - 1, j⟧ = "."];
    If[j ≠ m ∧ mat⟦i - 1, j + 1⟧ == "-", mat⟦i - 1, j + 1⟧ = "."];


    (* Furthermore, the tent should be checked (see below) *)
    If[mat⟦i, j⟧ == "z", checkTent[i, j]];
   ];
  ];

(* At position {i,j} is a "z",
which can become a "Δ" if it has exactly one unsatisfied neighbour tree ("b"). *)
checkTent = {i, j} ↦ Module[{neighbourTrees = 0},
   If[mat⟦i, j⟧ ≠ "z", Print["Abort: checkTent[",
     i, " ", j, "] run, although this tent is already a 'Δ'."];
    Abort[]];

   (* Count the unsatisfied neighbour trees "b". *)
   neighbourTrees = If[i < n ∧ mat⟦i + 1, j⟧ == "b", 1, 0] + If[i > 1 ∧ mat⟦i - 1, j⟧ == "b", 1,
      0] + If[j < m ∧ mat⟦i, j + 1⟧ == "b", 1, 0] + If[j > 1 ∧ mat⟦i, j - 1⟧ == "b", 1, 0];
   If[neighbourTrees == 0, Print["Abort: The tent at ", i,
     " ", j, " has no free tree left."];
    Abort[]];
   (* If there is only one neighbour "b",
   make this "b" a "Ψ" (run "satisfyTree") and the "z" a "Δ". *)
   If[neighbourTrees == 1, mat⟦i, j⟧ = "Δ";
    (* Search corresponding "b" and replace it *)
    If[i < n ∧ mat⟦i + 1, j⟧ == "b", satisfyTree[i + 1, j]];
    If[i > 1 ∧ mat⟦i - 1, j⟧ == "b", satisfyTree[i - 1, j]];
    If[j < m ∧ mat⟦i, j + 1⟧ == "b", satisfyTree[i, j + 1]];
    If[j > 1 ∧ mat⟦i, j - 1⟧ == "b", satisfyTree[i, j - 1]];
   ];
  ];



(* The tent at position {i,j} got a fixed tree. So we check if
 any neighbour trees of this tent can now also get a fixed tent. *)
satisfyTent = {i, j} ↦ Module[{temp = 0},
   If[FreeQ[{"-", "z"}, mat⟦i, j⟧], Print["Abort: satisfyTent[",
     i, " ", j, "] run, although there is no '-' or 'z'."];
    Abort[]];
   checkTentAnz[[i, j]] ++;
   mat⟦i, j⟧ = "Δ";
   (* For all neighbour trees: Check if they can be satisfied now. *)
   If[i < n ∧ mat⟦i + 1, j⟧ == "b", checkTree[i + 1, j]];
   If[i > 1 ∧ mat⟦i - 1, j⟧ == "b", checkTree[i - 1, j]];
   If[j < m ∧ mat⟦i, j + 1⟧ == "b", checkTree[i, j + 1]];
   If[j > 1 ∧ mat⟦i, j - 1⟧ == "b", checkTree[i, j - 1]];
  ];



(* The tree at position {i,j} got a fixed tent. So check if any
 other neighbour tent of this tree can now also get a fixed tree. *)
```

```
satisfyTree = {i, j} ↦ Module[{temp = 0},
   If[mat⟦i, j⟧ ≠ "b", Print["Abort: satisfyTree[",
     i, " ", j, "] run, although this field contains no 'b'."];
    Abort[]];
   mat[[i, j]] = "Ψ";
   (* For all neighbour tents: Check if it now has exactly one neighbour tree. *)
   If[i < n ∧ mat⟦i + 1, j⟧ == "z", checkTent[i + 1, j]];
   If[i > 1 ∧ mat⟦i - 1, j⟧ == "z", checkTent[i - 1, j]];
   If[j < m ∧ mat⟦i, j + 1⟧ == "z", checkTent[i, j + 1]];
   If[j > 1 ∧ mat⟦i, j - 1⟧ == "z", checkTent[i, j - 1]];
   (* For all neighbour black fields: They now have an unsatisfied neighbour tree
      less. It this reduces this number to zero, the field is a ".".  *)
   If[i < n ∧ mat⟦i + 1, j⟧ == "-", If[--neighbours⟦i + 1, j⟧ == 0, mat[[i + 1, j]] = "."]];
   If[i > 1 ∧ mat⟦i - 1, j⟧ == "-", If[--neighbours⟦i - 1, j⟧ == 0, mat[[i - 1, j]] = "."]];
   If[j < m ∧ mat⟦i, j + 1⟧ == "-", If[--neighbours⟦i, j + 1⟧ == 0, mat[[i, j + 1]] = "."]];
   If[j > 1 ∧ mat⟦i, j - 1⟧ == "-", If[--neighbours⟦i, j - 1⟧ == 0, mat[[i, j - 1]] = "."]]
  ];



(* Check if a tree has exactly one possible field for its tent and if so,
satisfy the tree and this tent. *)
checkTree = {i, j} ↦ Module[{neighbourTents = 0, neighbourFree = 0},
   If[mat⟦i, j⟧ ≠ "b",
    Print["Abort: checkTree[", i, " ", j, "] run, although there is no 'b'."];
    Abort[]];
   neighbourTents =
    If[i < n ∧ mat⟦i + 1, j⟧ == "z", 1, 0] +
     If[i > 1 ∧ mat⟦i - 1, j⟧ == "z", 1, 0] +
     If[j < m ∧ mat⟦i, j + 1⟧ == "z", 1, 0] +
     If[j > 1 ∧ mat⟦i, j - 1⟧ == "z", 1, 0];
   neighbourFree =
    If[i < n ∧ mat⟦i + 1, j⟧ == "-", 1, 0] +
     If[i > 1 ∧ mat⟦i - 1, j⟧ == "-", 1, 0] +
     If[j < m ∧ mat⟦i, j + 1⟧ == "-", 1, 0] +
     If[j > 1 ∧ mat⟦i, j - 1⟧ == "-", 1, 0];
   (* At least one neighbour field of the
    tree needs to be an unsatisfied tent or a free field. *)
   If[neighbourTents + neighbourFree == 0, Print["Abort: The tree at ",
     i, " ", j, " has no possible free neighbour tent."];
    Abort[]];
   (* If there is only one possible field for the tent
    and this is a free field, make it a Δ: *)
   If[neighbourTents == 0 ∧ neighbourFree == 1, mat⟦i, j⟧ = "Ψ";
    If[i < n ∧ mat⟦i + 1, j⟧ == "-", mat⟦i + 1, j⟧ = "Δ";
     checkTentNeighbours[i + 1, j]];
    If[i > 1 ∧ mat⟦i - 1, j⟧ == "-", mat⟦i - 1, j⟧ = "Δ";
     checkTentNeighbours[i - 1, j]];
    If[j < m ∧ mat⟦i, j + 1⟧ == "-", mat⟦i, j + 1⟧ = "Δ";
     checkTentNeighbours[i, j + 1]];
    If[j > 1 ∧ mat⟦i, j - 1⟧ == "-", mat⟦i, j - 1⟧ = "Δ";
     checkTentNeighbours[i, j - 1]];
   ];
   (* If there is only one possible field for the
    tent and this is an unsatisfied tent, satisfy the tent: *)
   If[neighbourTents == 1 ∧ neighbourFree == 0, mat⟦i, j⟧ = "Ψ";
    If[i < n ∧ mat⟦i + 1, j⟧ == "z", satisfyTent[i + 1, j]];
    If[i > 1 ∧ mat⟦i - 1, j⟧ == "z", satisfyTent[i - 1, j]];
    If[j < m ∧ mat⟦i, j + 1⟧ == "z", satisfyTent[i, j + 1]];
    If[j > 1 ∧ mat⟦i, j - 1⟧ == "z", satisfyTent[i, j - 1]];
   ];
   (* This is a special case that is sometimes
    needed: If there are exactly two possible positions for the tent,
   these positions are diagonal and the field next to these positions is "-",
   this field has to be green. To illustrate this
    - -                 - .
    b -      becomes    b -
   *)
   If[neighbourTents == 0 ∧ neighbourFree == 2,
    If[i > 1 ∧ j > 1 ∧ mat⟦i - 1, j⟧ == "-" ∧
      mat⟦i, j - 1⟧ == "-" ∧ mat⟦i - 1, j - 1⟧ == "-", mat⟦i - 1, j - 1⟧ = "."];
    If[i > 1 ∧ j < m ∧ mat⟦i - 1, j⟧ == "-" ∧ mat⟦i, j + 1⟧ == "-" ∧ mat⟦i - 1, j + 1⟧ == "-",
     mat⟦i - 1, j + 1⟧ = "."];
    If[i < n ∧ j > 1 ∧ mat⟦i + 1, j⟧ == "-" ∧ mat⟦i, j - 1⟧ == "-" ∧ mat⟦i + 1, j - 1⟧ == "-",
     mat⟦i + 1, j - 1⟧ = "."];
    If[i < n ∧ j < m ∧ mat⟦i + 1, j⟧ == "-" ∧ mat⟦i, j + 1⟧ == "-" ∧ mat⟦i + 1, j + 1⟧ == "-",
     mat⟦i + 1, j + 1⟧ = "."];
   ]
  ];



(*----------------------------------------- Program
  starts here  -----------------------------------*)

(* Make each "-" to ".", if it has no neighbour tree.  *)
Do[If[
  mat⟦i, j⟧ == "-",
  neighbours⟦i, j⟧ =
   If[i > 1 ∧ mat⟦i - 1, j⟧ == "b", 1, 0] +
    If[j > 1 ∧ mat⟦i, j - 1⟧ == "b", 1, 0] +
    If[i < n ∧ mat⟦i + 1, j⟧ == "b", 1, 0] +
    If[j < m ∧ mat⟦i, j + 1⟧ == "b", 1, 0];
  If[neighbours[[i, j]] == 0, mat[[i, j]] = "."]
 ],
 {i, n}, {j, m}
];

Do[
 altmat = mat;
  (* Update all rows and columns *)
 Do[rowUpdate[i], {i, n}];
 Do[colUpdate[j], {j, m}];

 (* Check all trees *)
 Do[If[mat⟦i, j⟧ == "b", checkTree[i, j]], {i, n}, {j, m}];

 (* If nothing has changed, break *)
 If[altmat == mat, Print[i]; Break[]],
 {i, 20}
 ];

 out = Join[(x ↦ {x}) /@ Prepend[numInRow, "out"], Prepend[mat, numInCol], 2];
 Print[MatrixForm[out]];

]
```

```
9
```

```
out 8 2 7 1 7 1 6 3 4 5 5 5 2 6 2 5 4 3 4 2 8
 3  . . . . . . Ψ Ψ △ Ψ △ . . Ψ . . . . . Ψ △
 6  . △ . . △ . △ . Ψ . . . . △ . △ . △ . . .
 2  . Ψ . . Ψ . . . △ Ψ △ . . . . Ψ . Ψ . . .
 3  △ . △ . . . Ψ . . . . . . . . . . . . Ψ △
 6  Ψ . Ψ Ψ △ . △ . Ψ Ψ △ . . . △ Ψ △ Ψ △ Ψ .
 5  z b z . . . Ψ . △ . . Ψ △ . . . . . . . △
 4  b . b Ψ △ . △ . Ψ . △ Ψ . . . Ψ △ . . . Ψ
 6  z b z . Ψ . . . △ . . . . △ Ψ . Ψ Ψ △ . △
 3  Ψ . . . △ . . . . . △ Ψ . . . . △ . . . Ψ
 6  △ . △ Ψ . . . △ Ψ Ψ . Ψ △ Ψ △ . . . . △ .
 1  Ψ . . Ψ . . . . . △ . . . Ψ . . Ψ . . Ψ .
 7  △ . . △ . . △ Ψ . . . . . △ . . △ Ψ △ Ψ △
 2  . . . . . . . . Ψ △ Ψ △ . . . . . . . . Ψ
 5  . △ Ψ . △ . . △ . . . Ψ . . . △ Ψ . . . △
 3  . . . . Ψ . . Ψ . . . △ Ψ △ . . . △ Ψ . Ψ
 4  Ψ . . . △ Ψ △ . . △ Ψ Ψ . . . Ψ . . . . △
 5  △ . △ Ψ . . Ψ . . . . △ . . . △ . Ψ △ . .
 3  Ψ . . . . △ Ψ △ . . . . . △ Ψ . . . . . .
 6  △ Ψ △ . . Ψ . . . △ Ψ △ Ψ . Ψ △ . . . . △
 2  . . Ψ Ψ △ . . . . . . Ψ . . . . Ψ △ . . Ψ
 8  △ Ψ △ . . Ψ △ . Ψ △ . △ Ψ △ . △ Ψ . . △ Ψ
```

```
Join[Join[{{0, 0, 0}}, Table[{maxInRow[[i]], numInRow[[i]], i}, {i, n}]],
  Join[{Range[m]}, matRow], 2] // MatrixForm
Join[Table[{i}, {i, -2, n}], Join[{maxInCol}, {numInCol}, {Range[m]}, matCol], 2] //
 MatrixForm
Join[Table[{i}, {i, 0, n}], Join[{Range[m]}, mat], 2] // MatrixForm
```

```
0 0  0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21
3 3  1 . . . . . . b Ψ △  Ψ  △  .  .  Ψ  .  .  .  .  .  Ψ  △
6 6  2 . △ . . △ . △ . Ψ  .  .  .  .  △  .  △  .  △  .  .  .
2 2  3 . Ψ . . Ψ . . . △  Ψ  △  .  .  .  .  Ψ  .  Ψ  .  .  .
3 3  4 △ . △ . . . Ψ . .  .  .  .  .  .  .  .  .  .  .  Ψ  △
6 6  5 Ψ . Ψ Ψ △ . △ . Ψ  Ψ  △  .  .  .  △  Ψ  △  Ψ  △  Ψ  .
5 5  6 z b z . . . Ψ . △  .  .  Ψ  △  .  .  .  .  .  .  .  △
4 4  7 b . b Ψ △ . △ . Ψ  .  △  Ψ  .  .  .  Ψ  △  .  .  .  Ψ
6 6  8 z b z . Ψ . . . △  .  .  .  .  △  Ψ  .  Ψ  Ψ  △  .  △
3 3  9 Ψ . . . △ . . . .  .  △  Ψ  .  .  .  .  △  .  .  .  Ψ
6 6 10 △ . △ Ψ . . . △ Ψ  Ψ  .  Ψ  △  Ψ  △  .  .  .  .  △  .
3 1 11 Ψ . . b – . . . .  △  .  .  .  Ψ  .  .  b  g  g  b  .
7 7 12 △ . . △ . . △ Ψ .  .  .  .  .  △  .  .  △  Ψ  △  Ψ  △
2 2 13 . . . . . . . . Ψ  △  Ψ  △  .  .  .  .  .  .  .  .  Ψ
5 5 14 . △ Ψ . △ . . △ .  .  .  Ψ  .  .  .  △  Ψ  .  .  .  △
3 3 15 . . . . Ψ . . Ψ .  .  .  △  Ψ  △  .  .  .  △  Ψ  .  Ψ
5 4 16 Ψ . . . △ Ψ △ . .  △  Ψ  Ψ  .  .  .  b  .  g  g  .  △
5 5 17 △ . △ Ψ . . Ψ . .  .  .  △  .  .  .  △  .  Ψ  △  .  .
3 3 18 Ψ . . . . △ Ψ △ .  .  .  .  .  △  Ψ  .  .  .  .  .  .
6 6 19 △ Ψ △ . . Ψ . . .  △  Ψ  △  Ψ  .  Ψ  △  .  .  .  .  △
2 2 20 . . Ψ Ψ △ . . . .  .  .  Ψ  .  .  .  .  Ψ  △  .  .  Ψ
8 8 21 △ Ψ △ . . Ψ △ . Ψ  △  .  △  Ψ  △  .  △  Ψ  .  .  △  Ψ
```

```
-2  8  2  7  1  7  1  6  3  4  5  5  5  2  6  3  5  4  3  4  2  8
-1  8  2  7  1  7  1  6  3  4  5  5  5  2  6  2  5  4  3  4  2  8
 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21
 1  .  .  .  .  .  .  Ψ  Ψ  △  Ψ  △  .  .  Ψ  .  .  .  .  .  Ψ  △
 2  .  △  .  .  △  .  △  .  Ψ  .  .  .  .  △  .  △  .  △  .  .  .
 3  .  Ψ  .  .  Ψ  .  .  .  △  Ψ  △  .  .  .  .  Ψ  .  Ψ  .  .  .
 4  △  .  △  .  .  .  Ψ  .  .  .  .  .  .  .  .  .  .  .  .  Ψ  △
 5  Ψ  .  Ψ  Ψ  △  .  △  .  Ψ  Ψ  △  .  .  .  △  Ψ  △  Ψ  △  Ψ  .
 6  z  b  z  .  .  .  Ψ  .  △  .  .  Ψ  △  .  .  .  .  .  .  .  △
 7  b  .  b  Ψ  △  .  △  .  Ψ  .  △  Ψ  .  .  .  Ψ  △  .  .  .  Ψ
 8  z  b  z  .  Ψ  .  .  .  △  .  .  .  .  △  Ψ  .  Ψ  Ψ  △  .  △
 9  Ψ  .  .  .  △  .  .  .  .  .  △  Ψ  .  .  .  .  △  .  .  .  Ψ
10  △  .  △  Ψ  .  .  .  △  Ψ  Ψ  .  Ψ  z  Ψ  △  .  .  .  .  △  .
11  Ψ  .  .  Ψ  .  .  .  .  .  △  .  .  .  Ψ  .  .  Ψ  .  .  Ψ  .
12  △  .  .  △  .  .  △  Ψ  .  .  .  .  .  △  .  .  △  Ψ  △  Ψ  △
13  .  .  .  .  .  .  .  .  Ψ  △  Ψ  △  .  .  .  .  .  .  .  .  Ψ
14  .  △  Ψ  .  △  .  .  △  .  .  .  Ψ  .  .  .  △  Ψ  .  .  .  △
15  .  .  .  .  Ψ  .  .  Ψ  .  .  .  △  Ψ  △  .  .  .  △  Ψ  .  Ψ
16  Ψ  .  .  .  △  Ψ  △  .  .  △  Ψ  Ψ  .  .  .  Ψ  .  .  .  .  △
17  △  .  △  Ψ  .  .  Ψ  .  .  .  .  △  .  .  -  △  .  Ψ  △  .  .
18  Ψ  .  .  .  .  △  Ψ  △  .  .  .  .  .  △  b  .  .  .  .  .  .
19  △  Ψ  △  .  .  Ψ  .  .  .  △  Ψ  △  b  .  b  △  .  .  .  .  △
20  .  .  Ψ  Ψ  △  .  .  .  .  .  .  Ψ  .  .  .  .  Ψ  △  .  .  Ψ
21  △  Ψ  △  .  .  Ψ  △  .  Ψ  △  .  △  Ψ  △  .  △  Ψ  .  .  △  Ψ
```

```
 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21
 1  .  .  .  .  .  .  Ψ  Ψ  △  Ψ  △  .  .  Ψ  .  .  .  .  .  Ψ  △
 2  .  △  .  .  △  .  △  .  Ψ  .  .  .  .  △  .  △  .  △  .  .  .
 3  .  Ψ  .  .  Ψ  .  .  .  △  Ψ  △  .  .  .  .  Ψ  .  Ψ  .  .  .
 4  △  .  △  .  .  .  Ψ  .  .  .  .  .  .  .  .  .  .  .  .  Ψ  △
 5  Ψ  .  Ψ  Ψ  △  .  △  .  Ψ  Ψ  △  .  .  .  △  Ψ  △  Ψ  △  Ψ  .
 6  z  b  z  .  .  .  Ψ  .  △  .  .  Ψ  △  .  .  .  .  .  .  .  △
 7  b  .  b  Ψ  △  .  △  .  Ψ  .  △  Ψ  .  .  .  Ψ  △  .  .  .  Ψ
 8  z  b  z  .  Ψ  .  .  .  △  .  .  .  .  △  Ψ  .  Ψ  Ψ  △  .  △
 9  Ψ  .  .  .  △  .  .  .  .  .  △  Ψ  .  .  .  .  △  .  .  .  Ψ
10  △  .  △  Ψ  .  .  .  △  Ψ  Ψ  .  Ψ  △  Ψ  △  .  .  .  .  △  .
11  Ψ  .  .  Ψ  .  .  .  .  .  △  .  .  .  Ψ  .  .  Ψ  .  .  Ψ  .
12  △  .  .  △  .  .  △  Ψ  .  .  .  .  .  △  .  .  △  Ψ  △  Ψ  △
13  .  .  .  .  .  .  .  .  Ψ  △  Ψ  △  .  .  .  .  .  .  .  .  Ψ
14  .  △  Ψ  .  △  .  .  △  .  .  .  Ψ  .  .  .  △  Ψ  .  .  .  △
15  .  .  .  .  Ψ  .  .  Ψ  .  .  .  △  Ψ  △  .  .  .  △  Ψ  .  Ψ
16  Ψ  .  .  .  △  Ψ  △  .  .  △  Ψ  Ψ  .  .  .  Ψ  .  .  .  .  △
17  △  .  △  Ψ  .  .  Ψ  .  .  .  .  △  .  .  .  △  .  Ψ  △  .  .
18  Ψ  .  .  .  .  △  Ψ  △  .  .  .  .  .  △  Ψ  .  .  .  .  .  .
19  △  Ψ  △  .  .  Ψ  .  .  .  △  Ψ  △  Ψ  .  Ψ  △  .  .  .  .  △
20  .  .  Ψ  Ψ  △  .  .  .  .  .  .  Ψ  .  .  .  .  Ψ  △  .  .  Ψ
21  △  Ψ  △  .  .  Ψ  △  .  Ψ  △  .  △  Ψ  △  .  △  Ψ  .  .  △  Ψ
```