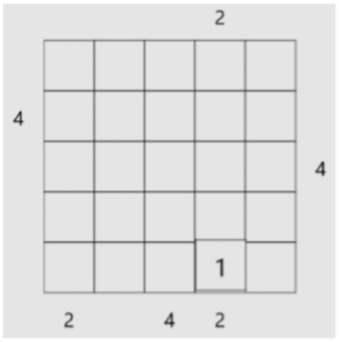
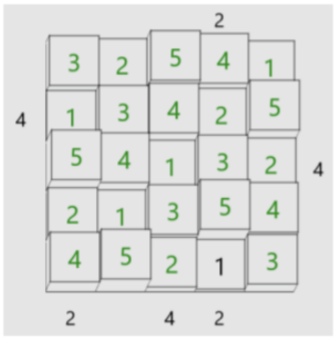
Skyscrapers (aka Towers)

Find a 'general' way to solve puzzles of the following kind:





For the precise rules, variants and other details, see:
https://brainbashers.com/skyscrapers.asp.
See also "Simon Tatham's Puzzles“.

Your solution should be able to solve at least 'hard' 4x4 and 5x5 puzzles.

Introduction:

The Skyscrapers puzzle, also known as Towers, is a logic-based number-placement puzzle that requires solvers to fill a square grid with skyscrapers of different heights, subject to various constraints. The intrigue of the Skyscrapers puzzle lies in its simplicity of rules paired with the complexity of solving, offering a satisfying challenge to puzzle enthusiasts.

Problem Definition:

A Skyscrapers puzzle is defined by a square grid. Each cell within the grid represents the base of a skyscraper, and the solver's task is to determine the height of each skyscraper, which is indicated by a number. The following rules and constraints govern the placement of these numbers:

1. **Grid Size**: The size of the grid, denoted by sizesize, determines the dimensions of the puzzle. A $size \times size$ grid requires the solver to place skyscrapers of heights $\{1, \dots, size\}$ in each row and column.
2. **Unique Heights**: Each row and column must contain unique skyscraper heights, with no repetitions allowed. This means that in a $size \times size$ grid, each row and column will contain all numbers $\{1, \dots, size\}$.
3. **Visibility Clues**: Along the edges of the grid, numerical clues are provided. These clues indicate the number of skyscrapers visible from that vantage point. A skyscraper is visible if there are no taller skyscrapers in front of it from that perspective.
4. **Height Constraints**: The height of a skyscraper in a cell is represented by a positive integer from $\{1, \dots, size\}$. The number 1 represents the shortest skyscraper, while the number $size$ represents the tallest skyscraper in the grid.

The objective of the Skyscrapers puzzle is to fill the entire grid with numbers that satisfy all the given constraints, ensuring that the number of visible skyscrapers from each edge matches the provided clues. Solving the puzzle requires logical deduction, pattern recognition, and sometimes a bit of trial and error.

Variables:

- **size:**
  Const $size$ represents the size of the grid:
  $$size \in Z^+$$
- **grid:**
  Function $grid(i, j)$: Represents the height of the skyscraper at row i and column j, where $i, j \in \{1, \dots, size\}$.
  $$grid: \{1, \dots, size\} \times \{1, \dots, size\} \rightarrow \{1, \dots, size\}$$

Functions:

- **MaxTower**
  A function $MaxTower(x, y)$, where $x, y \in \{1, \dots, size\}$ returns the maximum of two values.
  $$MaxTower: : \{1, \dots, size\} \times \{1, \dots, size\} \rightarrow \{1, \dots, size\}$$
  $$MaxTower(x, y) = \max(x, y)$$
- **Count Visible**:
  A function that counts the number of visible skyscrapers from a certain perspective in a row or column.
  $$\underbrace{CountVisible: \{1, \dots, size\} \times \dots \times \{1, \dots, size\}}_{(size\ time)} \rightarrow \{1, \dots, size\}$$

  We can express the function $CountVisible4(a, b, c, d)$ sums up the values of an indicator function that returns 1 if a skyscraper is visible and 0 otherwise.

  Variables:
  $$MaxA = a$$
  $$MaxB = MaxTower(a, b)$$
  $$MaxC = maxTower(MaxB, c)$$

  The Function represented as:

$$CountVisible4(a,b,c,d) = \sum_{i=1}^{4} \begin{cases} 1, & i = 1 \land a > 0 \\ 1, & i = 2 \land b > MaxA \\ 1, & i = 3 \land c > MaxB \\ 1, & i = 4 \land d > MaxC \\ 0, & otherwise \end{cases}$$

We can express the function $CountVisible5(a,b,c,d)$ sums up the values of an indicator function that returns 1 if a skyscraper is visible and 0 otherwise.

Variables:

$$\begin{aligned} &MaxA = a \\ &MaxB = MaxTower(a,b) \\ &MaxC = maxTower(MaxB,c) \\ &MaxD = maxTower(MaxC,d) \end{aligned}$$

The Function represented as:

$$CountVisible5(a,b,c,d,e) = \sum_{i=1}^{5} \begin{cases} 1, & i = 1 \land a > 0 \\ 1, & i = 2 \land b > MaxA \\ 1, & i = 3 \land c > MaxB \\ 1, & i = 4 \land d > MaxC \\ 1, & i = 5 \land e > MaxD \end{cases}$$

Constraints:

- **Distinct Rows and Columns**:

Every row and column must contain distinct numbers from 1 to the size of the grid.

For columns, this can be expressed as:

$$\forall i \in \{1,2,\ldots,n\},\ \{grid(i,1), grid(i,2),\ldots, grid(i,n)\} = \{1,2,\ldots,n\}$$

For columns, this can be expressed as:

$$\forall j \in \{1,2,\ldots,n\},\ \{grid(1,j), grid(2,j),\ldots, grid(n,j)\} = \{1,2,\ldots,n\}$$

- Each cell must contain a value between 1 and the size of the grid.

$$\forall i,j \in \{1,2,\ldots,size\}, grid(i,j) \in \{1,2,\ldots,size\}$$

- The clues provided at the edges of the grid dictate how many skyscrapers should be visible from that perspective. This is modeled by asserting that the count of visible skyscrapers (as calculated by the Count Visible function) must match the given clues.

$$CountVisible\big(grid(i,1), grid(i,2), \ldots, grid(i,size)\big) = Clue(i)$$
$$or$$
$$CountVisible\big(grid(i,size), grid(i,size-1), \ldots, grid(i,1)\big) = Clue(i)$$

$$CountVisible\big(grid(j,1), grid(j,2), \ldots, grid(j,size)\big) = Clue(j)$$
$$or$$
$$CountVisible\big(grid(j,size), grid(j,size-1), \ldots, grid(j,1)\big) = Clue(j)$$

Where $Clue(i)$ and $Clue(j)$are the clues given for the i^{tn}row and j^{tn}column respectively.

Z3 SMT Solution

5x5 grid

code:

```
; Define a 4x4 grid or 5x5 grid
(define-const size Int 5)

(declare-fun grid (Int Int) Int)

(assert (forall ((j Int))
  (=> (and (>= j 1) (<= j size))
    (distinct
      (grid 1 j)
      (grid 2 j)
      (grid 3 j)
      (grid 4 j)
      (ite (= size 5) (grid 5 j) 0)  ; conditional for 5x5
    )
  )
))
; Assert distinct values for rows
```

```
(assert (forall ((i Int))
   (=> (and (>= i 1) (<= i size))
      (distinct
         (grid i 1)
         (grid i 2)
         (grid i 3)
         (grid i 4)
         (ite (= size 5) (grid i 5) 0)  ; conditional for 5x5
      )
   )
))

(assert (forall ((i Int) (j Int))
   (=> (and (>= i 1) (<= i size) (>= j 1) (<= j size))
      (and (>= (grid i j) 1) (<= (grid i j) size))
   )
))

(define-fun MaxTower ((x Int) (y Int)) Int
   (ite (> x y) x y)
)

(define-fun count-visible-4 ((a Int) (b Int) (c Int) (d Int)) Int
   (let ((maxA a)
         (maxB (MaxTower a b))
         (maxC (MaxTower (MaxTower a b) c)))
      (+
         (ite (> a 0) 1 0)
         (ite (> b maxA) 1 0)
         (ite (> c maxB) 1 0)
         (ite (> d maxC) 1 0)
      )
   )
)
(define-fun count-visible-5 ((a Int) (b Int) (c Int) (d Int) (e Int)) Int
   (let ((maxA a)
         (maxB (MaxTower a b))
         (maxC (MaxTower (MaxTower a b) c))
         (maxD (MaxTower (MaxTower (MaxTower a b) c) d)))
      (+
            (ite (> a 0) 1 0)
            (ite (> b maxA) 1 0)
            (ite (> c maxB) 1 0)
            (ite (> d maxC) 1 0)
            (ite (> e maxD) 1 0)
        )
    )
)
(assert
    (=
        (count-visible-5
            (grid 1 1)
            (grid 2 1)
            (grid 3 1)
            (grid 4 1)
            (grid 5 1)
        )
        2
    )
)
(assert
    (=
        (count-visible-5
            (grid 1 3)
            (grid 2 3)
            (grid 3 3)
            (grid 4 3)
            (grid 5 3)
        )
        4
    )
)
(assert
    (=
        (count-visible-5
            (grid 1 5)
            (grid 2 5)
            (grid 3 5)
            (grid 4 5)
            (grid 5 5)
    )
    4
  )
)
(assert
  (=
    (count-visible-5
      (grid 3 5 )
      (grid 3 4 )
      (grid 3 3 )
      (grid 3 2 )
      (grid 3 1 )
    )
    4
  )
)

(assert
  (=
    (count-visible-5
      (grid 5 1 )
      (grid 5 2 )
      (grid 5 3 )
      (grid 5 4 )
      (grid 5 5 )
    )
    3
  )
)

(assert
  (=
    (count-visible-5
      (grid 3 1 )
      (grid 3 2 )
      (grid 3 3 )
      (grid 3 4 )
      (grid 3 5 )
    )
    2
  )
)
(assert
  (=
    (count-visible-5
      (grid 4 1 )
      (grid 4 2 )
      (grid 4 3 )
      (grid 4 4 )
      (grid 4 5 )
    )
    2
  )
)

(assert
  (=
    (count-visible-5
      (grid 3 1 )
      (grid 3 2 )
      (grid 3 3 )
      (grid 3 4 )
      (grid 3 5 )
    )
    2
  )
)

(check-sat)
(get-value (


  (grid 1 1) (grid 2 1) (grid 3 1) (grid 4 1) (grid 5 1)
  (grid 1 2) (grid 2 2) (grid 3 2) (grid 4 2) (grid 5 2)
  (grid 1 3) (grid 2 3) (grid 3 3) (grid 4 3) (grid 5 3)
  (grid 1 4) (grid 2 4) (grid 3 4) (grid 4 4) (grid 5 4)
  (grid 1 5) (grid 2 5) (grid 3 5) (grid 4 5) (grid 5 5)
))
```

Output:

Z3:

```
sat
(((grid 1 1) 4)
 ((grid 2 1) 5)
 ((grid 3 1) 2)
 ((grid 4 1) 3)
 ((grid 5 1) 1)
 ((grid 1 2) 3)
 ((grid 2 2) 1)
 ((grid 3 2) 5)
 ((grid 4 2) 2)
 ((grid 5 2) 4)
 ((grid 1 3) 1)
 ((grid 2 3) 2)
 ((grid 3 3) 4)
 ((grid 4 3) 5)
 ((grid 5 3) 3)
 ((grid 1 4) 5)
 ((grid 2 4) 4)
 ((grid 3 4) 3)
 ((grid 4 4) 1)
 ((grid 5 4) 2)
 ((grid 1 5) 2)
 ((grid 2 5) 3)
 ((grid 3 5) 1)
 ((grid 4 5) 4)
 ((grid 5 5) 5))
```

Python visual:

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

5x5 grid

code:

```
; Define a 4x4 grid or 5x5 grid
(define-const size Int 4)
(declare-fun grid (Int Int) Int)
(assert (forall ((j Int))
   (=> (and (>= j 1) (<= j size))
      (distinct
         (grid 1 j)
         (grid 2 j)
         (grid 3 j)
         (grid 4 j)
         (ite (= size 5) (grid 5 j) 0)  ; conditional for 5x5
      )
   )
))
; Assert distinct values for rows
(assert (forall ((i Int))
   (=> (and (>= i 1) (<= i size))
      (distinct
         (grid i 1)
         (grid i 2)
         (grid i 3)
         (grid i 4)
         (ite (= size 5) (grid i 5) 0)  ; conditional for 5x5
      )
   )
))
(assert (forall ((i Int) (j Int))
   (=> (and (>= i 1) (<= i size) (>= j 1) (<= j size))
      (and (>= (grid i j) 1) (<= (grid i j) size))
   )
))

(define-fun MaxTower ((x Int) (y Int)) Int
   (ite (> x y) x y)
)
(define-fun count-visible-4 ((a Int) (b Int) (c Int) (d Int)) Int
  (let ((maxA a)
        (maxB (MaxTower a b))
        (maxC (MaxTower (MaxTower a b) c)))
    (+
      (ite (> a 0) 1 0)
      (ite (> b maxA) 1 0)
      (ite (> c maxB) 1 0)
      (ite (> d maxC) 1 0)
    )
  )
)
(define-fun count-visible-5 ((a Int) (b Int) (c Int) (d Int) (e Int)) Int
  (let ((maxA a)
        (maxB (MaxTower a b))
        (maxC (MaxTower (MaxTower a b) c))
        (maxD (MaxTower (MaxTower (MaxTower a b) c) d)))
    (+
      (ite (> a 0) 1 0)
      (ite (> b maxA) 1 0)
      (ite (> c maxB) 1 0)
      (ite (> d maxC) 1 0)
      (ite (> e maxD) 1 0)
    )
  )
)
(assert
  (=
    (count-visible-4
      (grid 4 2)
      (grid 3 2)
      (grid 2 2)
      (grid 1 2)
    )
    2
  )
)
(assert
  (=
    (count-visible-4
      (grid 4 3)
```

```
        (grid 3 3)
        (grid 2 3)
        (grid 1 3)
      )
      2
    )
)
(assert
    (=
      (count-visible-4
        (grid 4 4)
        (grid 3 4)
        (grid 2 4)
        (grid 1 4)
      )
      2
    )
)
(assert
    (=
      (count-visible-4
        (grid 1 1 )
        (grid 1 2 )
        (grid 1 3 )
        (grid 1 4 )
      )
      4
    )
)
(check-sat)
(get-value (
    (grid 1 1) (grid 2 1) (grid 3 1) (grid 4 1)
    (grid 1 2) (grid 2 2) (grid 3 2) (grid 4 2)
    (grid 1 3) (grid 2 3) (grid 3 3) (grid 4 3)
    (grid 1 4) (grid 2 4) (grid 3 4) (grid 4 4)
))
```

Output:
Z3

```
sat
(((grid 1 1) 1)
 ((grid 2 1) 2)
 ((grid 3 1) 3)
 ((grid 4 1) 4)
 ((grid 1 2) 2)
 ((grid 2 2) 3)
 ((grid 3 2) 4)
 ((grid 4 2) 1)
 ((grid 1 3) 3)
 ((grid 2 3) 4)
 ((grid 3 3) 1)
 ((grid 4 3) 2)
 ((grid 1 4) 4)
 ((grid 2 4) 1)
 ((grid 3 4) 2)
 ((grid 4 4) 3))
```

Python visualisation:

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