Building skyscrapers

Basic terminology:

- The term 4-adjacent describes cells (or skyscrapers) that share a side.
- 4-reachable means "reachable by walking between 4-adjacent empty cells".
- Additional graph terminology will be used with the prefix "4-", meaning that the edges of the graph are defined by 4-adjacency.
- Skyscrapers that are 4-reachable from "infinity", as described in the statement, will be called 4-outside.
- A collection of skyscrapers is 8-connected if it forms a single connected component in a graph in which skyscrapers that share a corner or a side are considered adjacent.
- A skyscraper in an 8-connected collection is called an 8-articulation if after its removal the rest is no longer 8-connected.

Let's look at the problem in reverse: Any valid order of building, seen backwards, is a process in which we n times select a 4-outside skyscraper that is currently not an 8-articulation and we remove it.

The key observation to our solution is that the following statement is true: Each 8-connected figure contains at least one 4-outside non-8-articulation. Thus, when you are removing skyscrapers, you can never get stuck. This means that the lexicographically maximal sequence (the one you should construct in subtasks of type 2) can be constructed greedily by always removing the largest-numbered 4-outside non-8-articulation among all the remaining skyscrapers.

A detailed proof is quite technical and will be provided later.

Subtasks 4, 6 and 7

Checking reachability from infinity is easy. We can store the 4-connected components of empty cells, remember which component corresponds to the outside and whenever we make a new empty cell, we can update this information with the simple "merge smaller to larger" strategy. We just have to choose which empty cells to consider, since the naive estimate is $\mathcal{O}(n^2)$ empty cells; however, empty cells which have no full 8-neighbours in the original figure are not interesting, so we are left with $\mathcal{O}(n)$ interesting cells and the total time complexity of calculating connected components is $\mathcal{O}(n \log n)$.

When is a full cell c an articulation? Let's look at all its regions that aren't isolated empty corners (since their neighbours on the border of c are 8-connected, so the following argument doesn't apply). For each pair of different regions r_1 and r_2, if there is no 4-path of empty cells connecting them, then for each pair of full non-adjacent 8-neighbours of c (c_1 and c_2) such that c_1, r_1, c_2, r_2 are in this order around c, there is an 8-path of full cells connecting c_1 and c_2 that does not contain c. Vice versa, if there is a path connecting two such cells c_1 and c_2 that does not contain c, then there is no 4-path connecting r_1 and r_2. Since for each pair of non-corner regions r_1 and r_2, we can find 2 cells c_1 and c_2, we get an equivalent definition of a non-articulation: all non-corner regions belong to different 4-connected components of empty cells.

This simplifies things. We're already updating connected components, so the information about articulation-ness of cells is local – if we know the empty 4-neighbours and connected components of a full cell, we can compute whether it is an articulation in $\mathcal{O}(1)$ time. This information only needs recomputing if empty neighbours appear or are moved to a different component.

Now, we only need to realise that we already have to look at all empty cells that appear or are moved to a different component. It's done in the "merge smaller to larger" union-find for empty cells – when merging component K_s into component K_l, we have to look at all cells from K_s to update which component they belong to, and we only create an empty cell $\mathcal{O}(n)$ times along with that. For each cell at which we look in the union-find, we can just check all its full 8-neighbours, and for each of these neighbours, we can update the information abouts its neighbours, decide whether it is an articulation and if it is reachable from the outer component, all in $\mathcal{O}(1)$ time.

We are left with updates saying "this cell can be removed" / "this cell cannot be removed". With them, it's easy to keep the set of removable cells in amortised $\mathcal{O}(\log n)$ and choose the cell to remove in each step in $\mathcal{O}(\log n)$. The subtask type is irrelevant, we could just as well support queries "remove this cell or report that it's not allowed".

Obviously, the answer is "NO" if and only if the final configuration isn't 8-connected.

Total time complexity: $\mathcal{O}(n \log n)$, memory complexity $\mathcal{O}(n)$.

In subtask 6, we can afford to store the graph as a 2D array, with complexity $\mathcal{O}(n \log n + r_{max} c_{max})$. Also, we are checking if a cell is an articulation in $\mathcal{O}(d)$, where $d = 8$ is the number of neighbours of a cell, but less efficient implementations like finding regions and checking all pairs of regions in $\mathcal{O}(d^2)$ become possible.

In subtask 4, we only need to determine if it's possible to remove a skyscraper in $\mathcal{O}(1)$. Whenever we're removing a skyscraper, we can simply find the last removable one, then update the connected components, either by merging existing ones or running BFS on the interesting cells. This works in $\mathcal{O}(n^2)$.

Subtasks 3 and 5

When we are allowed to choose the output sequence, a much simpler approach becomes possible:

- First, build an arbitrary skyscraper.
- Store the set of skyscrapers which haven't been built, but are 8-adjacent to those which have been built.
- While we need to build a skyscraper, pick the rightmost one from this set (with the maximum c_i); if there is more than one such skyscraper, pick the one with the maximum r_i. Then, update the set.

This takes $\mathcal{O}(n \log n)$ time when the skyscrapers are stored efficiently (e.g. in an STL `set<>`), or $\mathcal{O}(n^2)$ by brute force.

The set of built skyscrapers is 8-connected, but it isn't immediately clear why we can't fail by closing an 8-cycle and then building a skyscraper inside it. However, it makes sense – informally, before building the last skyscraper from this cycle, we can already build everything on its inner border, and if the final configuration is 8-connected and there is something inside the cycle we still need to build, there will be a skyscraper we haven't built yet on this border with greater c_i (or equal c_i and greater r_i). We'll build everything inside a cycle before we finish the cycle itself.

Subtask 2

We can use the approach for $t = 1$, or we can guess that the hypothesis about existence of a removable skyscraper is true. Then, we can choose a skyscraper to remove by checking which ones can be removed (by removing and running BFS on the remaining graphs of empty and full cells). If we do this n times, the time complexity is $\mathcal{O}(n^3)$.

Subtask 1

We can try all $\mathcal{O}(n!)$ possible orders and check if they are valid.