L-Triominoes – solution

Domagoj Bradač

September 2nd, 2021

1 An $\mathcal{O}(2^W \cdot HW)$ solution using dynamic programming

We describe a dynamic programming solution which is standard for problems of this type. We build the tiling from the bottom up and from left to right as follows. Suppose $1 \leq r \leq H, 1 \leq c \leq W$ and we have so far tiled every square (r', c') with $(r', c') \leq (r, c)$ (and potentially some other squares in the process). In the next step, we put a triomino covering the current square (r, c). Note that there are at most four possible ways to do this (see Figure 1)

What do we need to keep track of? By assumption, we have already tiled all squares $(r', c') < (r, c)$. It is easy to check that the only other squares we might have additionally tiled are the following:

- (r, c') with $c' \geq c$;
- $(r + 1, c')$ with $c' \leq c$.

(See Figure 2.)

In total, there are $W + 1$ squares which might additionally be tiled. Hence, we can describe the state of our dynamic program as (r, c, m), where (r, c) is the position of the current square and m is a bitmask of $W + 1$ bits describing which additional squares are tiled. As mentioned, the DP transition can be done in time $O(1)$, yielding the total running time $\mathcal{O}(2^W \cdot HW)$.

2 The directed graph on bitmasks

To discuss the remaining solutions, we introduce a directed graph D. Its vertex set consists of all 2^W bit-vectors of length W. Such a vector represents a partially tiled row: 1's correspond to occupied squares and 0's to squares which need to be tiled. For two bit-vectors (u, v), the edge $u \to v$ appears in D if we can complete the tiling of the row partially tiled according to bitmasks u such that the next row has partial tiling

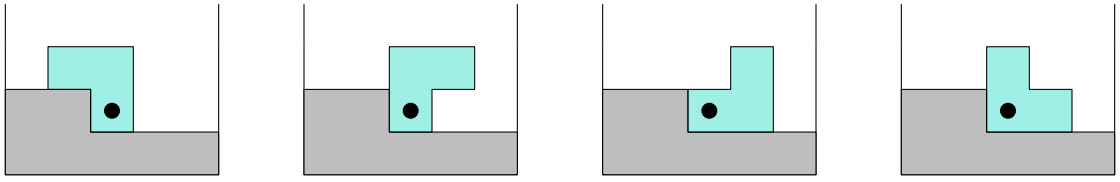

Figure 1: Four possible ways to continue tiling. Gray squares mark the previously tiled squares and the dot marks the current square (r, c).

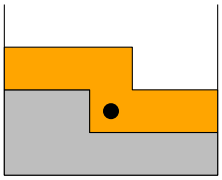

Figure 2: Squares we need to keep track of in a DP solution

Figure 3: There are two ways to continue tiling from the bitmask 101.

corresponding to bitmask v. E.g., suppose $W = 3$ and $u = 101$. There are two ways to finish the tiling of this row (Figure 3). Hence, there are two edges going from 101: $(101, 110)$ and $(101, 011)$.

Note that we can construct the entire digraph D in time $\mathcal{O}(2^{2W} \cdot W)$ using the abovementioned dynamic programming approach.

How can we use this digraph to obtain a faster solution? Let $1 \leq r \leq H$ be an arbitrary row. Suppose we can tile, bottom-up as before, the first $r-1$ rows, possibly tiling some additional squares in row r. Let S be the set of squares in row r which are either tiled or initially missing and let v_S denote the vertex in D corresponding to S. We say this tiling produces the set S in row r. Define

$$U_r = \left\{ v_S \in V(D) \mid \text{ there is a tiling which produces } S \text{ in row } r \right\}.$$

Now let $v_S \in U_r$ and consider a tiling of the first $r-1$ rows which produces S in row r. Let T be an arbitrary subset of squares in row $r+1$ and let v_T its corresponding vertex in D. Let us for the moment ignore any squares missing in row $r+1$. Then, by definition of D, we can extend our tiling such that the set of tiled squares in row $r+1$ is T if and only if (v_S, v_T) is an edge in D. Consider a set T we can achieve this way. If it contains any of the squares missing in row $r+1$ then this tiling is invalid. Otherwise, we add the missing squares to T, and add T to the set U_{r+1} of possible configurations in the next row.

A naïve implementation of the above yields an algorithm with running time $\mathcal{O}(2^{2W} \cdot H + K)$, worse than the dynamic programming approach.

However, this approach can be vastly improved. Let us first identify the simpler problem we want to attack. Suppose $1 \leq r < s \leq H$ and there are no tiles between rows r and s, inclusively. Given U_r, we need to be able to quickly determine U_s. In the setting of our directed graph D, it is enough to solve the following subproblem $O(k)$ times.

Problem 2.1. Given a set $U \subseteq V(D)$ and a positive integer ℓ, determine all vertices $v \in V(D)$ for which there exists a vertex $u \in U$ and a walk of length ℓ from v to u.

3 Matrix multiplication – $\mathcal{O}(2^{3W} \cdot K \log H)$

The first way to solve this problem uses a standard technique of matrix multiplication. It can be implemented to run in time $\mathcal{O}(2^{3W} \cdot K \log H)$. Using bitwise operations one can achieve a very small constant factor (at most 1/64 on modern architectures) compared to the usual implementation. Additionally, this approach

can be sped up to achieve running time $\mathcal{O}(2^{3W} \cdot \log H + 2^{2W} \cdot K \log H)$, which is left as an exercise to the reader. However, we decided not to award these speedups any additional points as they represent technical improvements to the solution which do not introduces any of the ideas required to solve the problem fully.

4 The full solution – $\mathcal{O}(2^W \cdot WK \cdot C(W))$.

The function $C(W)$ is a parameter which we will introduce formally later. To motivate our solution, we start with a definition.

Definition 4.1. The *period* of a directed graph is the greatest common divisor of the lengths of all (directed) cycles in the graph.

We immediately state a useful fact which in some form can be found in most textbooks about Markov chains (e.g. see [**?**]).

Fact 4.2. *Let $F = (V, E)$ be a strongly connected directed graph with period k. Then, V can be partitioned as $V = V_1 \dot\cup V_1 \dot\cup \ldots \dot\cup V_k$ such that for any $1 \le i \le k$, the vertices in V_i only have outgoing edges towards V_{i+1}, where we denote $V_{k+1} = V_1$. Additionally, there exists a constant $C = C(F)$, such that the following holds. Let $1 \le i, j \le k$, and let $u \in V_i, v \in V_j$ be arbitrary vertices. Then for any integer $\ell \ge C$, there is a walk from u to v of length ℓ if and only if $j - i \equiv \ell (\mathrm{mod}\, k)$.*

From now on suppose that D is strongly connected. This is actually not true and we will later describe the structure of D. However, solving the problem under the assumption that D is strongly connected captures the relevant ideas. Suppose we are given the period k of D as well as a partition $V(D) = V_1 \dot\cup V_1 \dot\cup \ldots \dot\cup V_k$ and the integer C from Fact 4.2. Then for given U and ℓ we can answer Problem 2.1 as follows. If $\ell \ge C$, find the set I of indices $1 \le i \le k$, such that $U \cap V_i \neq \emptyset$. For each $V_i, i \in I$, add the set $V_{(i+\ell) \bmod k}$ to the answer. This can be done in time $\mathcal{O}(|D|) = \mathcal{O}(2^W)$. If $\ell < C$, we can answer Problem 2.1 using dynammic programming in time $\mathcal{O}(2^W \cdot \ell W)$ as described in Section 1. Combining the two and recalling that we need to solve $\mathcal{O}(K)$ instances of Problem 2.1, we derive an algorithm with time complexity $\mathcal{O}\big(2^W \cdot WK \cdot C(W)\big)$, where $C = C(W)$ is the mentioned constant.

Finally, we describe the describe the structure of D. The cases $W = 2, 3$ have a different structure than larger values and these cases need to be handled separately. We leave this as a simple exercise for the reader. (Or see the attached source codes.) For $W \ge 4$:

a) If W is odd, there are two special vertices, those corresponding to masks $1010 \ldots 101$ and $0101 \ldots 010$. The former has no incoming edges (it is impossible to achieve this configuration by tiling the previous row) while the latter has no outgoing edges (it is impossible to continue the corresponding tiling). From now on we ignore these vertices and all of the following statements come with the exception of special vertices.

b) If W is not divisible by 3, then there is a single weakly connected component, it is also strongly connected, and has period 3. If W is divisible by 3, then there are three connected components V_0, V_1, V_2, all are strongly connected and have period 1. The partition from Fact 4.2 in the former case, and the connected components in the latter, correspond to residues modulo 3 of the number of 1's in the corresponding bitmasks.

c) For every non-trivial strongly connected component D' of D, we have $C(D') \le 11$. :-)

All of these statements can be verified with a piece of code which runs for a few minutes or less. In addition, the first point can easily be found with pen and paper. The second point can be guessed by considering residues modulo 3. The third point is difficult to guess precisely and is the most difficult to verify. That being said, obtaining (or guessing) a reasonably good upper bound on C is enough to obtain full points. Having determined the structure of D, we can hard-code it into our program and we do not need to explicitly generate the digraph D.

5 Subtasks

The following presents the subtasks for this problem and their intended solutions.

Subtask	Score	Constraint	Intended solution
1	10	$H \leq 1000$	Dynamic programming
2	7	$K = 0$	Pen and paper analysis
3	11	$W \leq 3$	This subtask can be solved using matrix multiplication. However, it can also be solved by doing a pen and paper analysis of the directed graph D. Such a solution of the subtask can lead to a full solution.
4	17	$4 \leq W \leq 6$	Matrix multiplication
5	35	$7 \leq W \leq 13$	Full solution
6	20	No special constraints	Full solution, correct handling of the special cases $W = 2, 3$.