K - Teardown

Cameron Reuschel, David Schantz
Table of Contents

- The Problem
- Debunking Approaches
- Problem Structure
- The Algorithm
- Implementation Tips
- Summary
The Problem
Problem Description

Bulldozer Time!

Given:
- Many buildings along a long, straight road
- modelled as individual square blocks

Objective:
- Level all the buildings
- by getting all blocks on the ground
- by moving any block left or right
- with as few moves as possible
What is a Move?
Input

Number of columns

10
1 3 0 0 1 9 1 1 1 1

Height of each column in blocks
Output

Minimum number of moves needed to get all blocks to level 0

13
Pause the video and play a little!

https://xdracam.itch.io/teardown
### Definitions

<table>
<thead>
<tr>
<th>n</th>
<th>Number of columns</th>
</tr>
</thead>
<tbody>
<tr>
<td>Block</td>
<td>Single block with clearly defined xy-coordinates</td>
</tr>
<tr>
<td>h</td>
<td>Height of a block = y-coordinate</td>
</tr>
<tr>
<td>m</td>
<td>Number of blocks with h &gt; 0</td>
</tr>
<tr>
<td>Column</td>
<td>A specific x-coordinate</td>
</tr>
<tr>
<td>Gap</td>
<td>A column without any blocks</td>
</tr>
<tr>
<td>Stack</td>
<td>Multiple adjacent blocks in the same column</td>
</tr>
<tr>
<td>Split</td>
<td>Separation of a column into three parts that either go left, right or are leveled in place</td>
</tr>
</tbody>
</table>

![Diagram showing definitions](image-url)

Obvious Problem Characteristics

- always possible to find a solution
  - infinite gaps to the left and right of the instance
- solution is not unique
  - many different moves and orders can lead to the same or equivalent outcomes
- each problem instance has mirror version with left/right swapped
  - so the order with which we iterate the instance does not matter
Intuitive Heuristics

- after pushing blocks into a direction, it makes no sense to push them back
- good idea to move many blocks at once
- moving towards closer/足够 enough gaps is better

- moving blocks at h=0 is useless

Or is it?
Debunking Approaches
Move all blocks into same direction?
Find a column $c$.
Move all blocks with $x \leq c$ to the left & all blocks with $x > c$ to the right
For each column, move left or right individually?
Problem Structure
Problem Complexity

- depends on number of blocks above ground level = $m$
- hard to solve in linear time
  - we need to split a stack in the middle sometimes
  - we can’t know where to split in advance
  - so we need to consider all splits

- up to $10^9$ columns with $10^5$ blocks each ($m < 10^{14}$)

$10^{14}$ bytes = 100 terabytes

⇒ we cannot possibly use $O(m)$ memory

As much data as the LHC generates in one second!
Upper and Lower Bounds

- $m = \text{number of blocks above ground level}$

- need a minimum of $m$ moves
  - every move can only level at most one block
  - blocks on floor are already leveled

- need a maximum of $2m$ moves
  $\Rightarrow \text{2-approximation strategy}$
Basic Solution Idea

Partition all blocks with $h > 0$ into non-overlapping intervals

- Every block in an interval is leveled with the same strategy
- In *Left/Right intervals*, all blocks are moved in the same direction until leveled
- In a *NoOp interval*, all blocks are leveled with the 2-approximation strategy
  - every block in a NoOp interval requires exactly 2 moves to be leveled

⇒ Partitioning of *all* blocks with $h > 0$ into non-overlapping intervals so that the *sum of required moves* is minimal
Visualization: Interval Partitioning

required moves: 4+2+3+4+1 = 14
Definition: Left / Right Intervals

- every block with $h > 0$ is moved in the same direction until leveled
- contain a start stack and a continuous sequence of complete columns
  - can include gap columns outside the problem instance!

Clearly defined by:

- start column index
- end column index
- number of blocks moved in start column (= start stack size)

**length** of an interval = number of included columns - 1
= end column index - start column index
Left / Right Intervals: Observations

- start stack always has at least 1 block with \( h > 0 \)
  - otherwise there would be nothing to move, so why include?
- for every block with \( h > 0 \), includes at least one matching gap
  - otherwise we could not have leveled that block in the interval
- end column is always a gap
  - interval ends as soon as we have found a gap for each non-leveled block
- a single column can include start stacks of both a left and a right interval

- required moves to level \( = \) length of the interval
  - in a right-interval, we need to move the leftmost block into the rightmost gap
  - the leftmost block is in the start stack, the rightmost gap is the end column
  - all other blocks on the way will be leveled before the leftmost block reaches the end gap
How Many Intervals?

- each block with $h > 0$ can be in either a left, right or noop interval
  \[\Rightarrow\text{up to } 3m \text{ possible intervals in a problem instance}\]

- up to $m$ non-overlapping intervals at the same time

  \[\Rightarrow O(2^m) \text{ interval partitionings to consider!}\]

  \[\text{but } m < 10^{14} \Rightarrow \text{impossible to calculate all partitionings}\]
The Algorithm
Incremental Calculation

Too many possible interval partitions

Cannot calculate them all

→ Dynamic Programming
Basic Approach

Iterate over columns from left to right

For each column \( x \), remember min number of moves required to level everything to the left (including \( x \)) in \( \text{movesUntil}[x] \)

---

**Input** : \( n, h \)

```plaintext
for x from 0 until n do
  // assume NoOp:
  \( \text{movesUntil}[x] \leftarrow \text{movesUntil}[x-1] + 2 \cdot \max(h[x] - 1, 0) \)
  consider left and right intervals separately

return \( \text{movesUntil}.last \)
```
Calculating a Left Interval

Naive approach: go left until we have gaps for all found blocks

→ Inefficient, will result in $O(m^2)$ runtime for left moves alone

Idea: Keep

- a stack of open gaps we found
- a counter how many gaps to the left of 0 have been filled
Calculating a Left Interval

Input : n, h

let gaps ← empty stack
let gapsFilledBeyondLeftBorder ← 0
let leftSplits ← 2-dim array
for x from 0 until n do
  leftSplits[x][0] ← movesUntil[x-1]
  if h[x] = 0 then
    push x to gaps
  else
    for y from 1 until h[x] do
      if gaps is not empty then
        leftBound ← gaps.pop()
      else
        gapsFilledBeyondLeftBorder += 1
        leftBound ← gapsFilledBeyondLeftBorder
      leftSplits[x][y] ← leftSplits[x][y-1] + 2
      let leftMoves ← movesUntil[leftBound] + x - leftBound
      if leftSplits[x][y] > leftMoves then
        leftSplits[x][y] ← leftMoves
      movesUntil[x] ← leftSplits[x][h[x]-1]
Calculating a Right Interval

Handle right intervals at their end column ➔ $x$ must be a gap

A search for each gap would be inefficient

Idea: New columns have to be leveled completely before a right interval can end

➔ Keep stack of possible right intervals

Each with $\{\text{leftCol}, \text{remainingBlocks}\}$
Calculating a Right Interval

Input : n, h

let openRightIntervals ← empty stack of \{ leftCol, remainingBlocks \}

for x from 0 until n do
    if h[x] > 1 then
        push \{ x, h[x] - 1 \} to openRightIntervals
    (left interval handling)
else if h[x] = 0 then
    movesUntil[x] ← movesUntil[x - 1]
    if openRightIntervals is not empty then
        let ri ← openRightIntervals.top
        let x₁ ← ri.leftCol
        let blocksTaken ← h[x₁] - ri.remainingBlocks
        let totalMoves ← leftSplits[x₁][blocksTaken] + x - x₁
        if totalMoves < movesUntil[x] then
            movesUntil[x] ← totalMoves
        ri.remainingBlocks ← 1
        if ri.remainingBlocks = 0 then
            openRightIntervals.pop()

Handle all remaining intervals in stack
Right column is always lastRightBound + .remainingBlocks (lastRightBound is n - 1 for the first one)
Necessary Optimizations
Current Performance

Need to iterate over every block with h > 1
- Once for left-intervals, once for right-intervals
→ \( O(m) \) runtime

Need to save min move value for each possible split
→ \( O(m) \) memory
- Worst case: No Gaps
  → All columns on right-interval stack
  → Need to handle all splits at the end
  → Actually need all the values until the end

Remember: up to \( 10^{14} \) blocks
1 byte per block → 100 terabyte
\( O(m) \) definitely doesn’t work for extreme cases.
Idea: Implement **leftSplits** as sparse data

Step one: flatten the array

Observation if $h > 1$:

$$\text{leftSplit}[x][h(x)-1] == \text{leftSplit}[x+1][0]$$

( == movesUntil[x])
Idea: Implement leftSplits as sparse data

Assumption: No gaps

How do the values in the array develop?

+1 For most blocks
+2 For a new column
Idea: Implement \texttt{leftSplits} as sparse data

Generalizing to gaps:

- At every gap, the required moves do not increase (one non-positive change)

\[ \Rightarrow \text{Only } O(n) \text{ non-1 differences in } \texttt{leftSplits} \]
\[ \Rightarrow \text{If we only save non-1 differences, we can reduce the memory usage from } O(m) \text{ to } O(n) \]
Getting the required data

But how do we get the minimum number of moves until starting a left split when considering a right split?

- Use a search tree (C++ std::map, Java TreeMap)
- Keys: indices of old array
- When entry is present, then done
- If not, search the tree for the next smaller key
- Result: Value at present entry + difference between the keys

$\rightarrow O(\log n)$ lookup instead of $O(1)$

Logarithmic factors can often be ignored for actual runtimes 😊
Idea: Skip Left Interval calculations

When calculating left intervals, we only jump from gap to gap (at most $n$)

→ As long as we stay in bounds (left column index $\geq 0$), total left split calculation is in $O(n \log n)$, as there can be at most $n$ gaps

→ $O(m \log n)$ only applies when leaving bounds

Idea: If we do leave the left bound, there will be infinite gaps → Every additional block only adds $+1$ move

Since we don’t save those, we can simply break once we found a worthy (= better than 2-approx strategy) left split across the left bound → $O(n \log n)$
Idea: Cleanup right intervals faster

Same approach: Infinite consecutive gaps after handling all columns

- No need to find `.remainingBlocks` next gaps, can just calculate end column
- Partitionally leveling stack is not necessary, taking all blocks is optimal

Only have to handle all right splits ending before right bounds (at most $n$) and one right interval per column that exceeds bounds (at most $n$)

$\rightarrow O(n \log n)$ in total
Implementation Tips

- Use long (int64) for most numbers!
  - large instances can easily cause ints to overflow
  - performance will be fine, we promise

- Watch out for offsets!
  - +/- 1 issues can easily happen
  - depending on how you keep track of values

- Use expressive variable names!
  - which values are in/exclusive w.r.t. column indices?, etc

- Ignore micro-optimizations until the very end
  - can get up to factor ~3 faster
  - but algorithmic improvements can lead to ~1000 times faster code!

https://xkcd.com/1691/
Summary

Iterate through all columns and keep track of:
- min number of moves required to level everything so far
- number of blocks with $h > 0$ encountered so far (= key for leftSplits)

If height of column $> 1$:
- push to openRightIntervals
- calculate possible leftSplits by iterating through the blocks
- stop iterating early when all following blocks would only need 1 more move

If column is a gap:
- push it to the gaps stack
- check whether including the top of openRightIntervals yields a better result

After iterating, iterate backwards through openRightIntervals and check for a better result

⇒ $O(n \log n)$ runtime
⇒ $O(n)$ space