LECTURE 10

CONVEX HULL TRICK

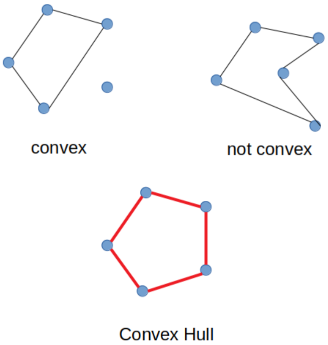
Po Hao Chen

4/8 2022

BOSTON UNIVERSITY

Convex Hull

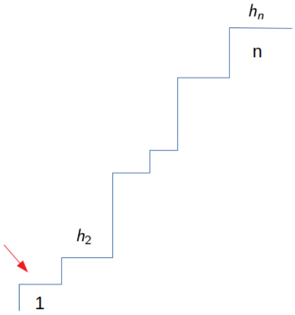
What is a convex hull?



- Convex Hull is the smallest convex set that contains ALL points
- For the mathematicians: $\forall x, y \in C$, $\lambda x + (1 - \lambda) y \in C$ conv(S): $\{\sum \lambda_i x_i\}$, where $x_i \in S, \lambda \in \Delta_k, k \in \mathcal{N}$

Dynamic Programming

There are n staircases, each one has a height of h_i.
The *cost* to go from i to j is $(h_j - h_i)^2 + C$.
What is the minimum cost to reach the last staircase from the first?
Constraint: $n \leq 2 * 10^5$



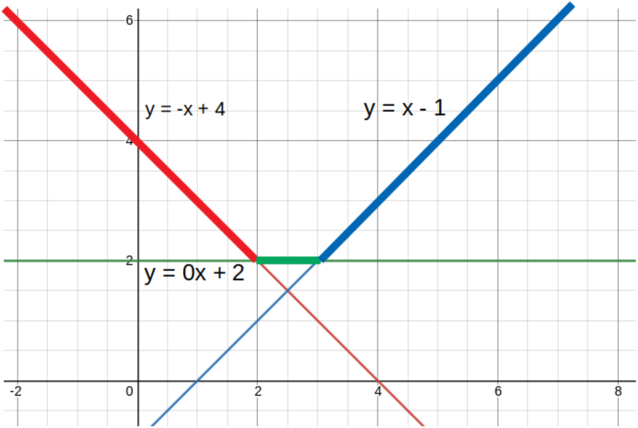
Dynamic Programming

Statement: There are n staircases, each one has a height of h_i.
The *cost* to go from i to j is $(h_j - h_i)^2 + C$.
What is the minimum cost to reach the last staircase from the first?

DP Recurrence: dp[j] $= min_{i \leq j}(dp[i] + (h_j - h_i)^2 + C)$
Time Complexity: $O(n^2)$

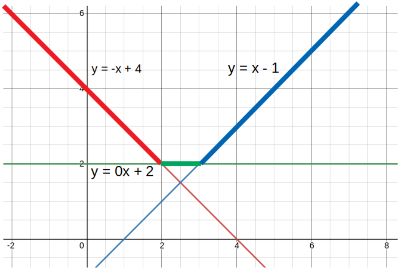
Geometry to Optimizes DP

The Convex Hull Trick is a dynamic programming optimization technique, it speeds up the time complexity by exploiting geometric properties.



The Intersection Point

Notice that the segment is only optimal till the intersection point with another line.



$y = m_1 x + c_1$ and $y = m_2 x + c_2$

$m_1 x + c_1 = m_2 x + c_2 \rightarrow x = \frac{c_2 - c_1}{m_1 - m_2}$

Observation: The above formula shows the optimal segments.

Code

We will create a custom struct for line.

- Initialize with slope and y-intercept (b).
- Evaluate y = mx+b;
- Calculate intersection with another line.
  (take ceiling using $\lceil \frac{x}{y} \rceil = \frac{x+y-1}{y}$)

LineContainer

We will maintain a dequeue-like data structure sorted by the optimal intersection x value.
It holds the (line , optimal x)

- Add new lines
- Supports query for y

Rewriting Recurrence

Expand:

$dp[j] = dp[i] + (h_j - h_i)^2 + C = dp[i] + h_j^2 - 2h_jh_i + h_i^2 + C$

Observation: For some j, the highlighted terms stay constant

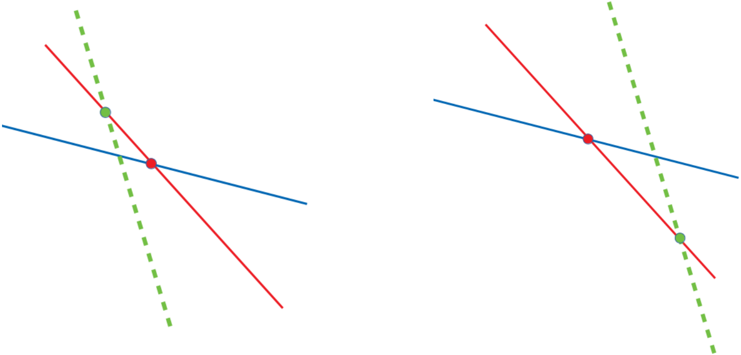
$dp[i] + h_j^2 - 2h_jh_i + h_i^2 + C$

Rearrange the terms, and let h_j be x since we want to find the minimal y for h_j.

$-2h_jh_i + (h_i^2 + dp[i]) + h_j^2 + C$

y = mx + b where m = $-2h_i$, b = $h_i^2 + dp[i]$, x = h_j,

Adding a Line



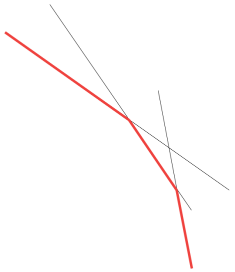
Adding a Line

```
add_line(slope, y-intercept):

    create new_line with input parameters

    while( at least 2 lines exists &&
            new_line is to the left of last_line )
        remove last_line from container


    if( container is empty ):
        add new_line with intersection of 0
    else:
        add new_line with intersection of "current" last_line
```

Monotonicity

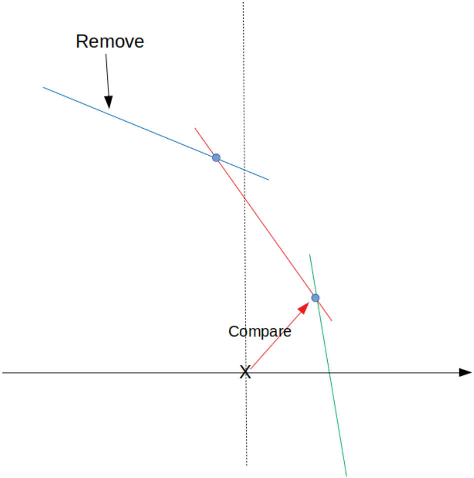
y = mx + b where m $= -2h_i$, x $= h_j$, b $= h_i^2 + dp[i]$

The slopes are monotonic.



for each x $= h_j$, the problem hints $h_j < h_{j+1}$, thus queries are monotonic.

Monotonicity



query(x):

while(at least an intersection exists):
  if (second to oldest line's intersection less than x):
    remove the oldest line
  else break

return the eval(x) with the appropriate line (the oldest remaining)

Query

Monotonicity determines the time complexity of our optimized algorithm.

- If slopes and queries are both monotone. We keep removing lines until query h_j is $\geq$ to x.
- If only slopes are monotone. Do not remove lines. Perform binary search to find the first x that our query h_j is $\geq$ to x.
- If slopes and queries are non-monotone. We will need a dynamic data-structure known as Li-Chao Tree.

Solution with Convex Trick

Recurrence: $-2h_j h_i + (h_i^2 + dp[i]) + h_j^2 + C$

y = mx + b where m = $-2h_i$, b = $h_i^2 + dp[i]$, x = h_j

- dp[0] = 0
- Initialize the first line
- dp[j] = query(h[j]) + $h_j^2 + c$
- addLine(m,b)
- dp[n-1] contains answer