Programming Assignment 1

Programming Assignment 1

By Jacob Noah Lettick && Jared Moore

Implementation

We solved the Dynamic Programming problem by first populating an empty table and then filling this with calculated values using the reccurence relation $dp[i,j] = \min(dp[i-1,j-1], dp[i,j-1], dp[i+1,j-1])$. We can then use this table to solve our problem by finding the minimum value of the bottom of the filled out DP table (which corresponds to the end of the lowest energy path). We used a class with a parent object to track which location a current pixel came from (right, center, left), and trace back to the top using these parnet objects to do our solution in one pass.

Benchmarking

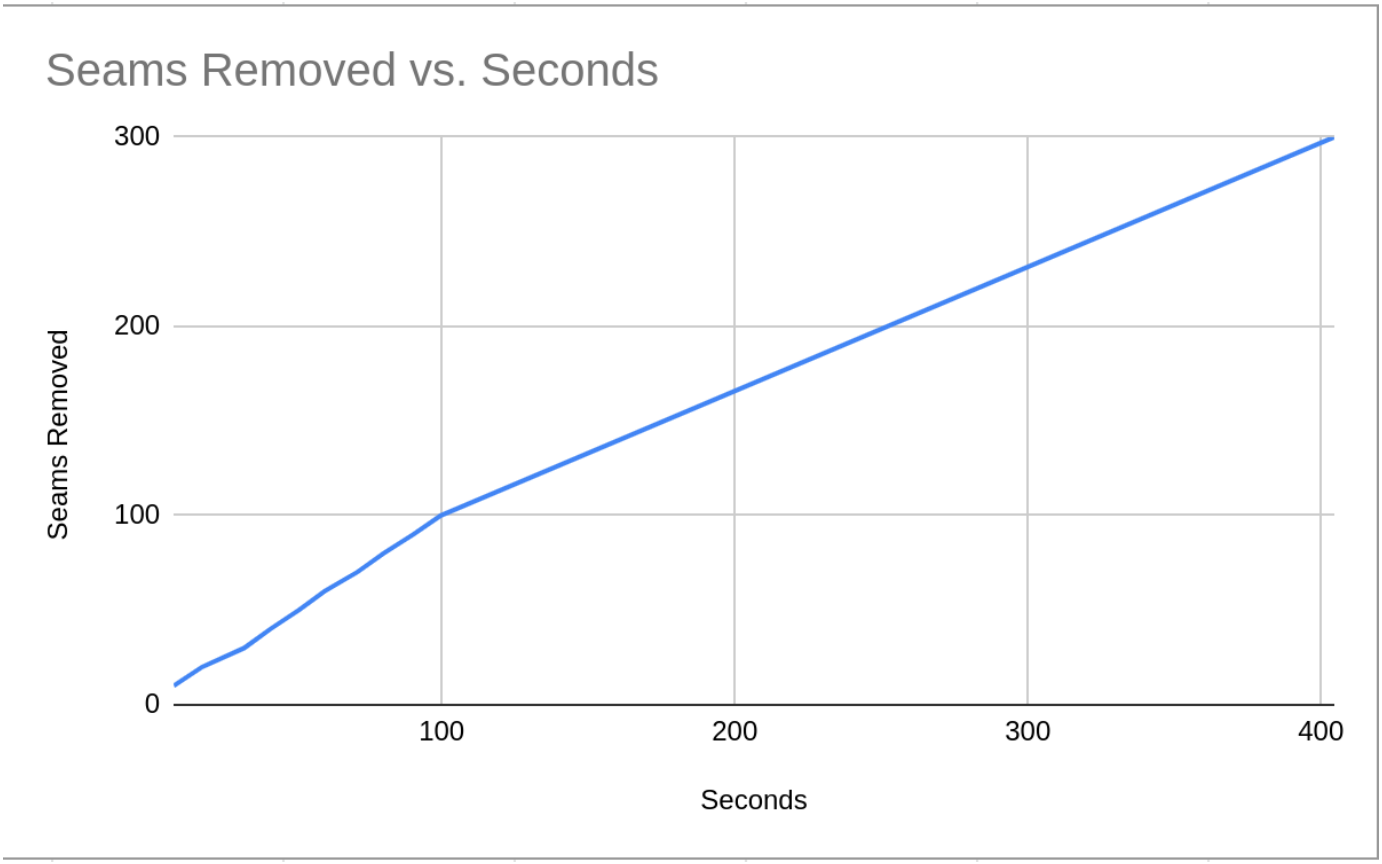
Naive Solution

The naive solution was dramatically slower, taking an average of **~600 seconds**, and only was able to be run on photos less than 16x16 in size. A naive solution would not be an effective way of solving this problem. The naive solution also doesn't necessarily return the optimal solution, just an optimal solution given the location.

DP Solution

The Dynamic Programming solution was much faster, taking an average of **.8907 seconds** per seam over 100 itterations, and was able to effectively find and remove seams on the example photographs. Some other solutions that could benefit the runtime is not recalculating the table for each call (as was necessary due to the nature of the startercode), which could reduce the runtime of seams after the first from ($M * N + M + N$) to ($M + N$). One thing we noticed was the runtime gets faster as the size of the image reduces, meaning the time for removing the nth seam is generally faster than the n-1th

seam.



Analysis

1. What is the reccurence for a horizontal sum?

The original subproblem was solved with the reccurence relation $dp[i,j] = \min(dp[i-1, j-1], dp[i, j-1], dp[i+1, j-1])$, in which **i is on the X-Axis**, and **j is on the Y-Axis**. To remove a horizontal seam by computing the horizontal sum, we would invert these variables as follows: $dp[i,j] = \min(dp[i+1, j-1], dp[i+1, j], dp[i+1, j+1])$ where **dp[i+1,j-1] is down to the right**, **dp[i+1,j] is center to the right**, and **dp[i+1,j+1] is up to the right**

---

2. Assume m > 1. Show that the set of potential vertical seams grows in complexity atleast exponential in n.

The set of vertical seams grows exponentially as m>1 because the number of subproblems needed to be solved is now 3 for each additional pixel of startpoint and the area of the image increases exponentially as the width or height increase. This forms, an exponential reccurence relation ($M * N$). This makes sense, because the traversal of the table is directly related to the size of the table, and is why a reccursive solution to the problem gives such terrible runtimes.

---

3. What is the asymptotic time complexity of the dynamic programming algorithm?

There is one subproblem for each pixel, with the potential for 3 subsequent subproblems. There is a constant amount of work for each subproblem. This means we will do $W * H$ opperations, where W is

the width of the image, and H is the height of the image. After building the table, we need to go through W once more to find the minimum of the bottom row. We then again traverse up from that min bottom pixel equivalent to H. This gives us a total time complexity of $O(W * H + W + H)$ where W = Width and H = Height.

---

4. Does the time complexity of both the DP and non-DP algorithm make sense given the results of the benchmarking?

The Time complexities of each make sense given the benchmarking. A DP solution only needs to go through the table once to contruct it ($M * N$) and then find the minimum pixel and backtrace from there, giving it a time complexity directly proportional to the size of the image. On the other hand, the reccursive soution to this problem created 3 subproblems for each problem it solved, and without memoization quickly fails with images above a size of 16x16 pixels