CS 370 Final Project: Build a String

Taylor He, Jacob Manzelmann, Thomas Osterman

I pledge my honor that I have abided by the Stevens Honor System.

In this assignment, we solve the problem Build a String on Hackerrank:

https://www.hackerrank.com/challenges/build-a-string/problem

Greg wants to build a string, S. Starting with an empty string, he can perform 2 operations:

1. Add a character to the end of S for A dollars.

2. Copy any substring of S, and then add it to the end of S for B dollars.

Calculate minimum amount of money Greg needs to build S.

\*\*\*When I write O(n), I mean θ(n), but they’re like the same thing practically

========= Dynamic Programming Approach: O(B/A \* n3) ========

Keep track of the lowest costs so far in cost[] as we build up the new string

Iterate through the length of the string from 0 to length(cost)

replace the cost array with the min cost.

First assume that the minimum is appending a single character, and then check if the built string is a substring to the rest of the string, and proactively set the future cost array index to be the minimum of itself and the current cost plus the substring cost B.

\* There are some caveats to this approach. We should not be copying if:

a. Copying does not exceed the built string length

b. The result of copying does not exceed the final string length

c. The substring does not appear in the future

The approach described above is much easier to follow in Pythonic [pseudo]code:

int A the cost of appending a character

int B the cost of appending a substring copy of S

string S the string we want to build

solve(A, B, S):

cost = [0] + ([MAX\_INT] \* len(S))

copy\_length = min(1, B/A) # when does it make sense to copy?

**for** i in range(1, len(cost)):

# First get the minimum of itself and the append operation

cost[i] = min(cost[i], cost[i-1] + A)

j = copy\_length

**while** j <= i and i + j < len(cost) and **S[i:i+j] in S[:i]**:

cost[i+j] = min(cost[i+j], cost[i] + B)

j += 1

return cost[-1]

Results: This simple approach passed 10/20 test cases, times out on the rest



==========================================================

How do we improve on the dynamic programming approach?

========= Solution 2: DP with Rabin-Karp Search Algo O(n3) ========

There are 3 pain points in the code in Solution 1:

1. The outer for loop is O(n), but it is necessary to populate the cost array
2. The while loop is O(n), but it is necessary to check
3. Finding a pattern takes an O(n \* copy\_length) time

Rolling Hash: Rabin-Karp Search Algo O(n), MIT PDF attached

**Objective**: Figure out if string S1 is a substring of S2, length n1 n2

Brute force: intuitive O(n1\*n2)

Rolling Hash: O(n2) – special hash that goes through each character in string

1. Hash S1 to get h(S1): **O(n1)**
2. Hash the first length n1 substring of S: **O(n1)**
3. Use the rolling hash method to calculate the subsequent O(n2) substrings in S, comparing the hash values to h(S1): **O(n2)**
4. If a substring hash value does match h(S1), do a string comparison on that substring and P, stopping if they do match and continuing if they do not. **O(n1)** This speeds up the algorithm and as long as the total time spent doing string comparison is O(n2), then the whole algorithm is also O(n2).

\*\*\* A practical application of the algorithm is [detecting plagiarism](https://en.wikipedia.org/wiki/Plagiarism_detection). Given source material, the algorithm can rapidly search through a paper for instances of sentences from the source material, ignoring details such as case and punctuation. Because of the abundance of the sought strings, single-string searching algorithms are impractical.

Code is below:

RabinKarp(s1, s2):

# Is s1 a pattern in s2?

hashed\_pattern =

solve(A, B, S):

cost = [0] + ([MAX\_INT] \* len(S))

copy\_length = min(1, B/A)

**for** i in range(1, len(cost)):

cost[i] = min(cost[i], cost[i-1] + A)

j = copy\_length

**while** j <= i and i + j < len(cost) and **RabinKarp(S[i:i+j], S[:i])**:

cost[i+j] = min(cost[i+j], cost[i] + B)

j += 1

return cost[-1]