CS 341 A3 Q4

Tareef Dedhar - 20621325

June 15, 2018

1 Bookshelf Problem Part 1

1.1 Description/Correctness

First, create an array of length n. Each element is a flag that indicates if there is a solution for a substring of length i+1 (since array[0] is length 1). This should be all false at the start.

Then, loop from 0 through n (as i, stopping before n). Each time, if the substring of the input from 0 thru i is a word, then this is definitely a valid prefix, and we can flag the array at i as true.

If not, we will have an internal loop from j = 0 to i. This loop checks if there is a valid prefix from 0 thru j (since this is cached already), and the substring from j + 1 to i, as this would mean there is a combination that is valid by splitting the input into multiple words (a valid solution with any number of words + our substring). If we get a match at any point, we can flag our prefix array[i] as true and break.

After this loop completes, we have a prefix array that is true wherever there exists a solution for a prefix of our input string of length[i + 1]. So, we can return a the array at index[n-1], since that will be the entire string.

1.2 Pseudocode

```
bool wordBreak(set d, String s)
{
  int len = s.length();
  if (!len) return true;

  // prefixes[i] means there's a solution for s.substring(0, i), or of length i + 1
  bool * prefixes = calloc(len * sizeof(bool));

  // go through options for each length of substring
  for (int i = 0; i < len; ++i)
  {
    // if this prefix is a work, then flag it as true
    if (isWord(s.substring(0, i + 1))) prefixes[i] = true;
    else
    {
      // check each set of possible matches with this string split in two, since we stored the substrin
      for (int j = 0; j < i; ++j)
      {
        if (prefixes[j] && isWord(s.substring(j + 1, i + 1)))
        {
          prefixes[i] = true;
          break;
        }
      }
    }
  }

  return prefixes[len - 1];
}
```

1.3 Time Complexity

First, we retrieve the length of our string and make a boolean check on that length. This is constant time. Second, we create a 0-filled array of length n. This takes $\Theta(n)$ time. Now, we have a loop that loops n times.

Each time, we: Use isWord to check a value, and if it is true set a value in an array. This is 2 constant time operations. Or, we loop from 0 up to i (as j), and check an array index and use isWord again, and potentially set an array value. This is up to 3 constant time operations, done up to i times (which is up to n since i loops up to n). This inner loop is Θ(n). This loop does O(n) work (since sometimes it is Θ(1), other times Θ(n)), and does that n times. This means it is O(n^2).

2 Bookshelf Problem Part 2

2.1 Modification to output the solution in words

In order to print the solution out, we create an array of strings of length n. Each index correpsonds to the same index in the array of booleans from part a. Whenever a boolean is flagged as true, the substring used for that value is stored in the array of strings at the same index. Then, once our algorithm produces "true", we print out the array at index (n - 1), and each succesive index where that index is equal to the previous minus the length of the printed string. Since the isWord function uses a substring and takes constant time, we can assume we can also use a substring of the input with the same efficiency, so the building of this string array should take constant time. As for printing it out, we loop up to n times (if the substrings are all length n), so this would be Θ(n) at the end, and therefore would not affect the runtime.