Problem Set 1 - General

Question E

So in this question, we have two upper and lower bounds for a set of k strings. And we want to calculate the number of unique prefixes that we can have across those k strings. When solving this question, it really helped to first consider what the answer would be if there were no restrictions, and then sequentially, add the restrictions of:

1. The upper and lower bounds s and t.
2. The limit on the number of strings k

I did this in my brain at first, but I reckon it would be nice to apply it to the code when explaining to see how the thought process unfolded. In order to do this, consider if we had no bounds s and t, the number of unique prefixes that could occur between them is 2^n, where n is the length of the string. Now, if we write a for loop to multiply the number of unique prefixes at each step by 2, we have our base to apply the restrictions onto. I.e. for every i in n:

```
    ns = ns * 2
```

Now, when applying the restriction of s and t, notice that at every stage of i, the number of paths for prefixes doubles - however, if the lower bound (s) is set at b, it would remove a branch from this - the ‘a’ branch. Now if the higher bound (t) was set to a, it would also remove a branch. Hence, the code now is as follows post-applying the s and t restriction.

```
    int s_bound = s[i] == 'b' ? 1 : 0;
    int t_bound = t[i] == 'a' ? 1 : 0;

    ns = ns * 2 - s_bound - t_bound;
```

Finally, to apply the constraint of only having k strings, given we can only select up to k strings, the maximum number of prefixes that can contribute to the result at each step is k. So, for every position i, we’ll add the number of valid prefixes to our total result ans by adding the min of ns and k. Also, we cap the number of valid prefixes ns to k+1 to make sure that we aren’t unnecessarily tracking more paths than is needed or is possible. The +1 is just a flag to note for future iterations that we have more than k prefixes at the future step, capping it at k would mean we lose information as to whether we had k or more than k prefixes at the previous step. After this, our final loop looks like:

```
for (int i = 0; i < n; ++i) { // iterate through every letter
    int s_bound = s[i] == 'b' ? 1 : 0;
    int t_bound = t[i] == 'a' ? 1 : 0;

    ns = ns * 2 - s_bound - t_bound;

    ans += std::min(ns, k);
    ns = std::min(ns, k + 1);
}
```

And then we print out answer and happy days!