USACO JAN12 Problem 'combos' Analysis

Solution Notes (Richard Peng): A clear first solution would be one that tries all 3^K sequences of length K. Consider the following small minor modification of it: instead of counting the number of points at the end, when we enumerate the character at i, we count the number of points obtained by combos ending at position i. Then since the combos have length at most $L \leq 15$, we only need to track the last 15 characters in the sequence. Therefore the state of the previous 15 values in the sequence uniquely determines the state of our enumeration, giving a $O(3^L LNK)$ time algorithm. However, note that we can do even better. Suppose we're at position i and there is some position $j < i$ such that the sequence of characters between position j and i is not a substring of any combo. Then there cannot be any combo that starts before position j that ends at a position after i. Hence, we only need to track the part of the sequence that's a substring of one of the combos. More precisely, we identify the earliest j, or the longest suffix of the sequence so far that's a substring of some combo. Since there are only $O(NL^2)$ possible substrings of one of the combos, this immediately brings the number of states to something manageable. Since each state only be modified by adding one of 3 characters (A, B, C) to it, the transitions to position $i + 1$ can also be precomputed in $O(N^2L^4)$ time naively. In total this gives a $O(N^2L^4 + NL^2K)$ algorithm, which is sufficient for full points. Note however a key detail is that multiple combos can end at position i, making it necessary to precompute a score for each state as well.

Several further improvements are possible, with the most immediate being that instead of considering substrings of combos, we consider only prefixes. This is because if a combo ends after i, it must have started somewhere earlier. Once again, special care is needed to ensure that the state tracks the longest suffix that's a prefix of some combo. This gives a $O(N^2L^2+NLK)$ time algorithm.

Here is an implementation of a slightly different algorithm from problem author Neal Wu, in which he considers only states where one of the combos is matched exactly at position i, with a bit more work done computing transitions.

```
#include <cstdio>
#include <cstring>
#include <cassert>
#include <algorithm>
using namespace std;

FILE *input = fopen ("combos.in", "r"), *output = fopen ("combos.out", "w");

const int N_MAX = 105, LEN_MAX = 105, K_MAX = 1005;

int N, K, combo_len[N_MAX];
char combos[N_MAX][LEN_MAX];
int contains[N_MAX][LEN_MAX], attach[N_MAX][N_MAX];
int dp[K_MAX][N_MAX];

bool match(int a, int b, int offset)
{
    int a_len = combo_len[a], b_len = combo_len[b];
    assert(offset <= b_len && b_len - offset <= a_len);
    int a_start = a_len - (b_len - offset);
    return strncmp(combos[a] + a_start, combos[b], b_len - offset) == 0;
}

int main()
```

```
{
    fscanf(input, "%d %d", &N, &K);

    for (int i = 0; i < N; i++)
    {
        fscanf(input, "%s", combos[i]);
        combo_len[i] = strlen(combos[i]);
    }

    memset(contains, 0, sizeof(contains));
    for (int i = 0; i < N; i++)
    {
        contains[i][0] = 0;

        for (int end = combo_len[i], offset = 0; end >= 0; end--, offset++)
        {
            int occur = 0;

            for (int j = 0; j < N; j++)
                if (end >= combo_len[j] && strncmp(combos[i] + end - combo_len[j],
                         combos[j], combo_len[j]) == 0)
                    occur++;
            contains[i][offset + 1] = contains[i][offset] + occur;
        }
    }

    for (int a = 0; a < N; a++)
        for (int b = 0; b < N; b++)
            for (int len = max(1, combo_len[b] - combo_len[a]);
                len <= combo_len[b]; len++)
                if (match(a, b, len))
                {
                    attach[a][b] = len;
                    break;
                }

    memset(dp, -63, sizeof(dp));
    for (int i = 0; i < N; i++)
        dp[combo_len[i]][i] = contains[i][combo_len[i]];

    int max_points = 0;

    for (int k = 0; k <= K; k++)
        for (int i = 0; i < N; i++)
            if (dp[k][i] >= 0)
            {
                max_points = max(max_points, dp[k][i]);

                for (int j = 0; j < N; j++)
                {
                    int new_k = k + attach[i][j];

                    if (new_k <= K)
                        dp[new_k][j] = max(dp[new_k][j],
                                dp[k][i] + contains[j][attach[i][j]]);
                }
            }

    fprintf(output, "%d\n", max_points);
}
```