CSE 5311-001 Lab Assignment 3

Due May 2

Goals:

1. Understanding of string preprocessing techniques.

2. Application of a greedy heuristic to the shortest common superstring problem (SCS): Given a set of strings, find the shortest string such that every input string is a substring of the generated string. This problem has applications in data compression and in DNA sequencing. Since SCS is NP-hard, fast heuristics are important.

Requirements:

1. Write a C program that implements the standard greedy technique to find an approximate SCS for a set of strings. Your program must compile and execute on `omega.uta.edu`.

2. Submit your C code on Canvas before 5:00 p.m. on Monday, May 2.

Getting Started:

1. The input (`stdin`) will consist of the following:

   a. A line with `n` (number of strings). `n` ≤ 100.
   b. `n` lines, each including a single string with no more than 50 symbols.

2. Before doing other processing, your program should verify that none of the input strings is a substring of any other input string. If this condition is violated, an error message and termination should result. There is no loss of generality in making this assumption, but it avoids a few tedious situations.

3. The "standard" greedy technique is conceptually simple. Given a set of strings, determine an ordered pair of strings X=UV and Y=VW that maximizes |V|. Next, replace X and Y with UVW. Apply this technique until one string remains. (This algorithm has been conjectured as producing a superstring that is no more than twice as long as the SCS.)

4. Your program should only compute the maximum overlap for each ordered pair of strings *once* and, in spite of the simple description, will not need to compute any other overlaps. Instead, you may view the problem as approximating a longest Hamiltonian path in a weighted, directed graph: each vertex corresponds to a string and the weight on each edge (X,Y) is the maximum overlap.

   The greedy technique corresponds to concatenating paths, but only at the endpoints. Care is needed to avoid cycles.

5. Your output should print the strings, one per line, to make the overlaps comprising the superstring obvious.

For example, suppose the input was:

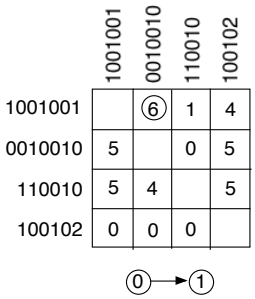
```
4
1001001
100102
0010010
110010
```

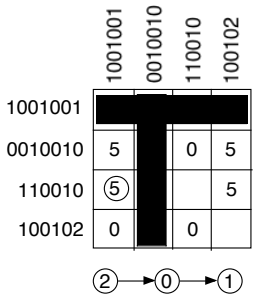
The output could be

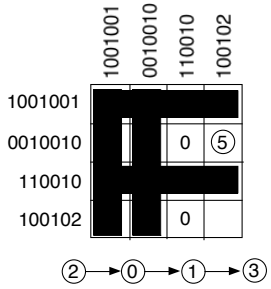
```
1001001
 0010010
        110010
         100102
```

or

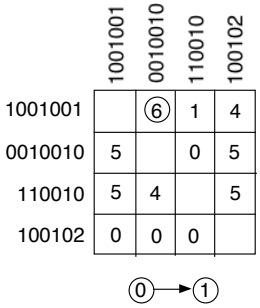
```
110010
 1001001
  0010010
    100102
```

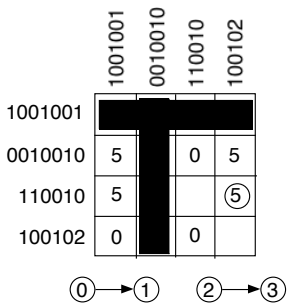


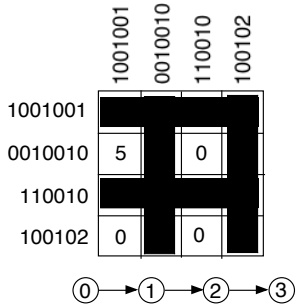




110010
1001001
0010010
100102







1001001
0010010
110010
100102