Algorithm Assignment 4 report

Author: 410821305 Michael Hsueh

Problem 1. Parking problem

Q1. Design a greedy algorithm to solve the problem.

Here, I will explain the code of my algorithm.

The steps are the following below:

1. The user will input if there is a parking space or a car. (first input line: 1 means there is a parking space, 0 means don't; second input line: 1 means there is a car, 0 means don't.). If the count of the cars is greater than the parking spaces, than the error message occurs.
2. Then the program will start to initialize the array called "shortest", there are 2 rows in this array. The first row is the parking space that the car can go with the shortest moving time, the second row is the distance (or the moving time) to the parking space.
3. Run a loop to go through the "shortest" array to find the shortest length that the car can go to the parking space.
4. Place the car into the parking space. Which means that we

need to refresh the “shortest” array and the input arrays so that when running the next time, the same event will not happen.

5. Run procedure 3 and 4 until all the cars are in the parking space.
6. Print out the answer.

Q2. Prove the optimality of your algorithm.

The solution is optimal since we choose the shortest length every single time. However, there is a question that if we run the for loop to find the shortest length, we need to find the parking space at the left of the car when the parking space is the shortest at the right of the car and also the left of the car. For example:

Parking space: 1 0 1 0

Car’s location: 0 1 0 1

It is obvious that the first car we met must go to the left parking space, which this algorithm does.

Q3. Analyze the time complexity of your algorithm.

Assume there are total n spaces for parking spaces and cars, and there are m cars on the road.

We need to run until all of the cars are in the parking space, while every time we need to refresh the “shortest” array. We need time complexity O(n^2). Therefore, the time complexity of this algorithm is O(mn^2).

Q4. Write a program and test it properly.

At file “Q4.1 parking problem\ A4.1 - Parking problem.cpp” and “Q4.1 parking problem \ test cases.JPG”.

Problem 2. Shortest sentence problem

Q1. Design a greedy algorithm to solve the problem.

Here, I will explain the code of my algorithm.

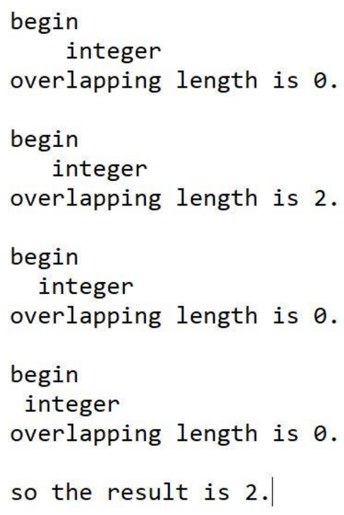
The steps are the following below:

1. Input a number n, represents the total keywords that the user will input.
2. Input all the keywords until the user inputs n keywords, store each string into an array.
3. Copy the array into a vector.
4. Run each possible pair of strings in the vector. Say string str1 and string str2. Then, run min (str1.length(), str2.length()) times to test the overlapping length. For

each pair, find the overlapping length. After go through all of the possible pair of strings, we have the maximum overlapping length and the two strings that we want to combine.

The way to compute the overlapping length:

Example: str1 = “begin”, str2 = “integer”:



```
begin
    integer
overlapping length is 0.

begin
   integer
overlapping length is 2.

begin
  integer
overlapping length is 0.

begin
 integer
overlapping length is 0.

so the result is 2.
```

5. Then, we start to combine the string. The formula is:

   str1 = str1 + { [str2 – (str2[0] to str2[overlapping length - 1]) }.

6. Keep doing the procedure 4 and 5 until the size of the vector is 1.

7. Print out the answer.

Q2. Prove the optimality of your algorithm.

We should get the maximum overlapping length after go through every possible pair of strings.

This algorithm seems optimal. However, it is not always correct, this algorithm is not optimal. Here, I will show the counter example:

Total keywords: 3

Keyword1: abcdef

Keyword2: efde

Keyword3: defab

By the algorithm above, the order are as follows:

1. Combine “abcdef” and “defab”, we get “abcdefab”. Since the maximum overlapping length is 3.
2. Combine “abcdefab” and “efde”, we get “abcdefabefde”. Since the maximum overlapping length is 0.
3. The result is “abcdefabefde”, the sentence length is 12.

However, unfortunately, the sentence length could be 11, the description are as follows:

1. Combine “efde” and “defab”, we get “efdefab”, The overlapping length is 2.
2. Combine “abcdef” and “efdefab”, we get “abcdefdefab”,

the overlapping length is 2.

3. The result is “abcdefdefab”, the sentence length is 11.

There is an example that using greedy approach algorithm is not optimal.

Q3. Analyze the time complexity of your algorithm.

Assume the count of the keywords are n, for each pair of strings, the minimum length is m.

We need to run the possible pair of strings. However, for each pair of strings, we need to run square time to find the longest overlapping length within to given strings, so the time complexity is O(n^2m^2).

Q4. Write a program and test it properly.

At file “Q4.2 shortest sentence string problem\A4.2 shortest sentence string problem.cpp” and “Q4.2 shortest sentence string problem\test case.JPG”.