Problem Solving

Lab #11

Department of Computer Engineering
Kyung Hee University

Problem 1: Edit Distance

- Definition

$$\mathrm{ED}(s[1..i], t[1..j]) = \min \begin{cases} \mathrm{ED}(s[1..(i-1)], t[1..(j-1)] + \mathrm{match}(s[i], t[j]) \\ \mathrm{ED}(s[1..i\,], t[1..(j-1)] + \mathrm{insert}(t[j]) \\ \mathrm{ED}(s[1..(i-1)], t[1..(j-1)] + \mathrm{delete}(s[i]) \end{cases}$$

,where match(), insert(), and delete() represent the costs for edit operations.

- Using the costs defined in the textbook, compute the edit distances of the following pairs of words, manually. Submit your results including the computation steps.
  - milk, meal
  - cat, pet
- Write the program for computing the edit distance between two strings and check whether your computations are correct. The program should also output the edit steps.

Problem 2: Different Cost Functions

- Repeat the previous page problems (manual computation and writing program) using the following cost functions.
  - match: 0 if two characters are same. Do not allow substitution of different characters.
  - insert: 1
  - delete: 2

Problem 3: Approximate Substring Matching

- Given a pair of a string and a text, write a program that finds out the best match of the string in the given text.
- Use the edit distance function defined in the textbook.

Problem 4: Longest Common Subsequences

- What are the longest common subsequences of the following pairs of sequences?
  - elephant and television
  - sweater and waste
- Write a program that finds out the longest common subsequences of the given two sequences.

Problem 5: Elevator Optimization

Write a program that solves the following problem:

I work in a very tall building with a very slow elevator. It is especially frustrating for me when people press the buttons for many neighboring floors (say 13, 14, and 15) as I go from ground to the top floor. My trip upward is interrupted three times, once at each of these floors. It would be far more polite for the three of them to agree to press only 14, and have the floor 13 and 15 people take the stairs up or down for one floor. They could use the walk, anyway.

Your task is to write an elevator optimization program. The riders all enter their intended destinations at the beginning of the trip. The elevator then decides which floors it will stop at along the way. We limit the elevator to making at most *k* stops on any given run, but select the floors so as to minimize the total number of floors people have to walk either up or down. You can assume the elevator is smart enough to know how many people want to get off at each floor.

We assume that the penalty for walking up a flight of stairs is the same as walking down one – remember, these people can use the exercise. In this spirit, management proposes to break ties among equal-cost solutions by giving preference to stopping the elevators at the lowest floor possible, since it uses less electricity. Note that the elevator does not necessarily have to stop at one of the floors the riders specified. If riders specify floors 27 and 29, it can decide to stop at floor 28 instead.

Hints of Problem 5

- This is an example of a typical programming/algorithm problem which can be neatly
- solved by dynamic programming.
- Recall that dynamic programming algorithms are based on recursive algorithms. So we have to find out the recurrence relation first.
- Deciding the best place to put the *k*th stop depends on the cost of all possible solutions with $k-1$ stops. If you can tell me the cost of the best of the relevant partial solutions, I can make the right decision for the final stop.
- Efficient dynamic programming algorithms tend to require *ordered* input. The fact that the passenger's destinations can be ordered from lowest to highest is important. Consider a trip that stops at floor $f2$, after an initial stop at $f1$. This second stop can be of absolutely no interest to any passenger whose real destination is at or below $f1$. This means that the problem can be decomposed into pieces. If I want to add a third stop $f3$ above $f2$, plotting the best location for it requires no knowledge of $f1$.

Hints of Problem 5

- So we smell dynamic programming here. What is the algorithm? We need to define a cost function for the partial solutions which will let us make bigger decisions.

  Let m[i][j] denote the minimum cost of serving *all* the riders using exactly *j* stops, the last of which is at floor *i*.

- Can this function help us place the (*j* +1)st stop given smaller values? Yes. The (*j* +1)st stop must, by definition, be higher than the previous (*j*th) stop at floor *i*. Further the new stop will only be of interest to the passengers seeking to get above the *i*th floor. To figure out how much it can help, we must properly divide the passengers between the new stop and *i* based on which stop they are closer to. This idea defines the following recurrence:

$$m_{i,j+1} = \min_{k=0}^{i}(m_{k,j} - \text{floors\_walked}(k,\infty) + \text{floors\_walked}(k,i) + \text{floors\_walked}(i,\infty))$$

- What does the recurrence mean? If the last stop is at *i*, the previous stop has to be at some $k < i$. What will such a solution cost? We must subtract from $m_{k,j}$ the cost of servicing all passengers above *k* (i.e., floors_walked(*k,*∞)), and replace it by the (presumably) reduced cost of adding a stop at *i* (i.e., floors_walked(*k, i*) +floors_walked(*i,*∞)).

- The key is the function floors walked(a,b), which counts the total number of floors walked by passengers whose destinations are between the two consecutive stops *a* and *b*. Each such passenger goes to the closest such stop.