Introduction to Algorithms October 25, 2004
Massachusetts Institute of Technology 6.046J/18.410J
Professors Piotr Indyk and Charles E. Leiserson Handout 17

Problem Set 5

Reading: Chapters 15, 16

Both exercises and problems should be solved, but *only the problems* should be turned in. Exercises are intended to help you master the course material. Even though you should not turn in the exercise solutions, you are responsible for material covered in the exercises.

Mark the top of each sheet with your name, the course number, the problem number, your recitation section, the date and the names of any students with whom you collaborated.

Three-hole punch your paper on submissions.

You will often be called upon to "give an algorithm" to solve a certain problem. Your write-up should take the form of a short essay. A topic paragraph should summarize the problem you are solving and what your results are. The body of the essay should provide the following:

1. A description of the algorithm in English and, if helpful, pseudo-code.
2. At least one worked example or diagram to show more precisely how your algorithm works.
3. A proof (or indication) of the correctness of the algorithm.
4. An analysis of the running time of the algorithm.

Remember, your goal is to communicate. Full credit will be given only to correct algorithms which are *which are described clearly*. Convoluted and obtuse descriptions will receive low marks.

Exercise 4-1. Do Exercise 15.2-1 on page 338 in CLRS.

Exercise 4-2. Do exercise 15.3-4 on page 350 in CLRS.

Exercise 4-3. Do exercise 15.4-4 on page 356 in CLRS and show how to reconstruct the actual longest common subsequence.

Exercise 4-4. Do exercise 16.1-3 on page 379 in CLRS.

Exercise 4-5. Do exercise 16.3-2 on page 392 in CLRS.

Problem 4-1. Typesetting

In this problem you will write a program (real code that runs!!!) to solve the following typesetting problem. **Because of the trouble you may encounter while programming, we advise you to START THIS PROBLEM AS SOON AS POSSIBLE.**

You have an input text consisting of a sequence of n words of lengths $\ell_1, \ell_2, \ldots, \ell_n$, where the length of a word is the number of characters it contains. Your printer can only print with its built-in Courier 10-point fixed-width font set that allows a maximum of M characters per line. (Assume that $\ell_i \leq M$ for all $i = 1, \ldots, n$.) When printing words i and $i+1$ on the same line, one space character (blank) must be printed between the two words. Thus, if words i through j are printed on a line, the number of extra space characters at the end of the line—that is, after word j—is $M - j + i - \sum_{k=i}^{j} \ell_k$.

To produce nice-looking output, the heuristic of setting the cost to the square of the number of extra space characters at the end of the line has empirically shown itself to be effective. To avoid the unnecessary penalty for extra spaces on the last line, however, the cost of the last line is 0. In other words, the cost *linecost*(i, j) for printing words i through j on a line is given by

$$\textit{linecost}(i,j) = \begin{cases} \infty & \text{if words } i \text{ through } j \text{ do not fit into a line,} \\ 0 & \text{if } j = n \text{ (i.e. last line),} \\ \left(M - j + i - \sum_{k=i}^{j} \ell_k\right)^2 & \text{otherwise.} \end{cases}$$

The total cost for typesetting a paragraph is the sum over all lines in the paragraph of the cost of each line. An optimal solution is an arrangement of the n words into lines in such a way that the total cost is minimized.

(a) Argue that this problem exhibits optimal substructure.

(b) Define recursively the value of an optimal solution.

(c) Describe an efficient algorithm to compute the cost of an optimal solution.

(d) Write code (in any language you wish—even Visual Java ++ :-)[1]) to print an optimal arrangement of the words into lines. For simplicity, assume that a word is any sequence of characters not including blanks—so a word is everything included between two space characters (blanks).

(d) requires 5 parts: you should turn in the code you have written, and the output of your program on the two input samples using two values of M (the maximum number of characters per line), namely $M = 72$ and $M = 40$, on each input sample.

Sample 1 is from *A Capsule History of Typesetting* by Brown, R.J. Sample 2 is from *Out of Their Minds*, by Shasha, Lazere. Remember that collaboration, as usual, is allowed to solve problems, but you must write your program by yourself.

Here is what Sample 1 should look like when typeset with $M = 50$. Feel free to use this output to debug your code.

[1]The solution will be written using C.

```
The first practical mechanized type casting
machine was invented in 1884 by Ottmar
Mergenthaler. His invention was called the
"Linotype". It produced solid lines of text
cast from rows of matrices. Each matrice was a
block of metal -- usually brass -- into which
an impression of a letter had been engraved or
stamped. The line-composing operation was done
by means of a keyboard similar to a typewriter.
A later development in line composition was
the "Teletypewriter". It was invented in
1913. This machine could be attached directly
to a Linotype or similar machines to control
composition by means of a perforated tape. The
tape was punched on a separate keyboard unit.
A tape-reader translated the punched code into
electrical signals that could be sent by wire to
tape-punching units in many cities simultaneously.
The first major news event to make use of the
Teletypewriter was World War I.
```

(e) Suppose now that the cost of a line is defined as the number of extra spaces. That is, when words i through j are put into a line, the cost of that line is

$$\textit{linecost}(i,j) = \begin{cases} \infty & \text{if words } i \text{ through } j \text{ do not fit into a line,} \\ 0 & \text{if } j = n \text{ (i.e. last line),} \\ M - j + i - \sum_{k=i}^{j} \ell_k & \text{otherwise;} \end{cases}$$

and that the total cost is still the sum over all lines in the paragraph of the cost of each line. Describe an efficient algorithm that finds an optimal solution in this case.

Problem 4-2. Manhattan Channel Routing

A problem that arises during the design of integrated-circuit chips is to hook components together with wires. In this problem, we'll investigate a simple such problem.

In ***Manhattan routing***, wires run on one of two layers of an integrated circuit: vertical wires run on layer 1, and horizontal wires run on layer 2. The height h is the number of horizontal tracks used. Wherever a horizontal wire needs to be connected to a vertical wire, a ***via*** connects them. Figure 1 illustrates several ***pins*** (electrical terminals) that are connected in this fashion. As can be seen in the figure, all wires run on an underlying grid, and all the pins are collinear.

In our problem, the goal is to connect up a given set of pairs of pins using the minimum number of horizontal tracks. For example, the number of horizontal tracks used in the routing channel of Figure 1 is 3 but fewer might be sufficient.

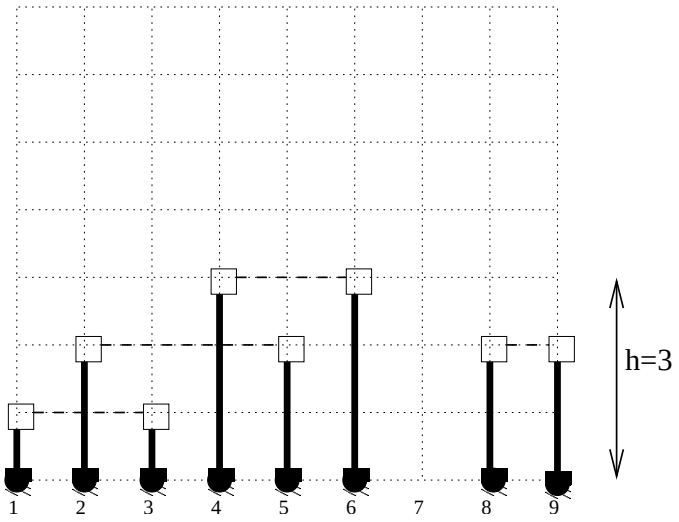

Figure 1: Pins are shown as circles. Vertical wires are shown as solid. Horizontal wires are dashed. Vias are shown as squares.

Let $L = \{(p_1, q_1), (p_2, q_2), \ldots, (p_n, q_n)\}$ be a list of pairs of pins, where no pin appears more than once. The problem is to find the fewest number of horizontal tracks to connect each pair. For example, the routing problem corresponding to Figure 1 can be specified as the set $\{(1, 3), (2, 5), (4, 6), (8, 9)\}$.

(a) What is the minimum number of horizontal tracks needed to solve the routing problem in Figure 1?

(b) Give an efficient algorithm to solve a given routing problem having n pairs of pins using the minimum possible number of horizontal tracks. As always, argue correctness (your algorithm indeed minimizes the number of horizontal tracks), and analyze the running time.