CS601 Algorithms and Complexity **Jul-Nov 2021**

Homework (No submission)

Lecture 11-12 (Aug 30-31)

- Let $G(V, E)$ be a graph with edge weights and $V = V_1 \cup V_2$ be a partition of vertices. Let C be the set of cut edges connecting V_1 with V_2, i.e.,

  $$C = \{(u, v) : u \in V_1, v \in V_2\}.$$

  Let e^* be the minimum weight edge in C. Prove that there must exists a minimum weight spanning tree containing e^*.

- Suppose you are given a set of n course assignments today, each of which has its own deadline. Let the i-th assignment have deadline d_i and suppose to finish the i-th assignment it takes ℓ_i time. Given that there are so many assignments, it might not be possible to finish all of them on time. If you finish an assignment at time t_i which is more than its deadline d_i, then the difference $t_i - d_i$ is called the lateness of this assignment (if $t_i < d_i$ then the lateness is zero). Since you want to maintain a balance among courses, you want that the maximum lateness over all assignments is as small as possible. You want to find a schedule for doing the assignments which minimizes the maximum lateness over all assignments. Can you show that a greedy algorithm will give you an optimal solution?

  - Greedy Strategy 1: Do the assignments in increasing order of their lengths (ℓ_i).
  - Greedy Strategy 2: Do that assignment first whose deadline is the closest.
  - Greedy Strategy 3: Do that assignment first for which $d_i - \ell_i$ is the smallest.

  Two of these strategies don't work. Give examples to show that they don't work. One of the strategies actually work. Prove that it works by arguing that there is an optimal solution which agrees with the first step of the greedy algorithm.

  Example: $d_1 = 20, \ell_1 = 10$, $d_2 = 40, \ell_2 = 20$, $d_3 = 60, \ell_3 = 30$. If the assignments are done in order $(1, 3, 2)$ then the maximum lateness will be 20 (for assignment 2). If the assignments are done in order $(1, 2, 3)$ then the maximum lateness will be 0.

- Given a set of intervals, you need to assign a color to each interval such that no two intersecting intervals have the same color. Design an efficient algorithm find a coloring with minimum number of colors.

Lecture 13 (Sep 2)

- You are going on a car trip from city A to city B that will take multiple days. On the way, you will encounter many cities. You plan to drive only during the day time and on each night you will stay in one of the intermediate cities. Suppose you can drive at most d kilometers in a day. You are given the distances of the intermediate cities from city A, say, $d_1, d_2, \ldots, d_n$. And distance of B from A is d_{n+1}. You are also given the costs of staying in various cities for one night, say, $c_1, c_2, \ldots, c_n$. Find a travel schedule, that is, in which all cities you should do a night stay, such that your total cost of staying is minimized.

- A subsequence of a string is obtained by possibly deleting some of the characters without changing the order of the remaining ones. For example, *ephn* is a subsequence of *elephant*.

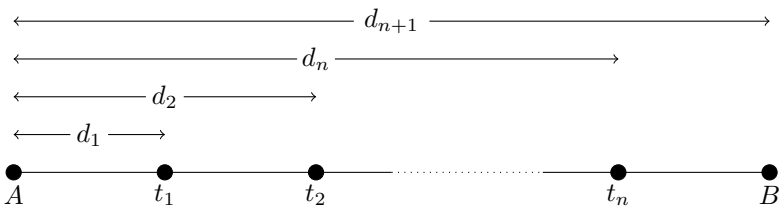

Figure 1:

You are given a string A of length n and another string B of length m ($\leq n$). If we want to check whether A contains B as a subsequence, there is greedy algorithm for it: For $1 \leq i \leq m$, match the character $B[i]$ with its first occurrence in A after the matching of $B[i-1]$.

For example, if $A = bacbcbabcacba$ and $B = bcbca$, then the greedy approach will match B as follows (shown in red).

$$bacbcbabcacba$$

Now, suppose to match with the j-th character in string A, you have to pay a cost p_j. And to match B with a subsequence in A, you have to pay the sum of costs of the matched characters.

In the above example, if the prices for the characters in A were $2, 5, 3, 1, 2, 5, 3, 1, 3, 1, 2, 4, 1$ then the matching $bacbcbabcacbaa$ has cost $2 + 3 + 1 + 2 + 3 = 11$. While the matching $bacbcbabcacba$ has cost only $1 + 2 + 1 + 2 + 1 = 7$.

Design an algorithm that given A, B and $p_1, p_2, \ldots, p_n$, can find the minimum cost subsequence in A that can be matched with B. Ideally your algorithm should run in time $O(mn)$.

- Given an array of integers, you want to find a subset with maximum total sum such that no two elements in the subset are adjacent. For example, for the array $\{6, 4, 3, 2, 1, 5\}$, the desired subset is $\{6, 3, 5\}$ with total sum 14. Design an $O(n)$-time algorithm for this problem, where n is the length of the array.

- Suppose there are n objects with their weights being $w_1, w_2, \ldots, w_n$ and their values being $v_1, v_2, \ldots, v_n$. You need to select a subset of the objects such that the total weight is bounded by W, while the total value is maximized. Your algorithm should run in time poly(n, W).

  Consider the case when the weights are too large, that is, they are exponential in n. Then the above algorithm is not really efficient. Suppose on the other hand, then values $\{v_1, v_2, \ldots, v_n\}$ are small. Can you design an algorithm running in time poly$(n, \sum_i v_i, \log W)$?

Lecture 14,15 (Sep 6, 7)

- The naive algorithm to multiply two matrices of dimensions $p \times q$ and $q \times r$ takes time $O(pqr)$. Suppose we have four matrices A, B, C, D which are $2 \times 4, 4 \times 3, 3 \times 2, 2 \times 5$ respectively. If you multiply $ABCD$ in the order $(AB)(CD)$, it will take time $2 \times 4 \times 3 + 3 \times 2 \times 5 + 2 \times 3 \times 5 = 84$, on the other hand if you multiply in the order $A((BC)D)$, it will take time $4 \times 3 \times 2 + 4 \times 2 \times 5 + 2 \times 4 \times 5 = 104$.

  Given matrices $A_1, A_2, \ldots, A_n$ with dimensions $p_1 \times p_2, p_2 \times p_3, \ldots, p_n \times p_{n+1}$, design an algorithm to find the order in which you should multiply $A_1 A_2 \cdots A_n$ which minimizes the multiplication time.

- Segmented Least Square (Kleinberg Tardos Section 6.3)

- Kleinberg Tardos Chapter 6 Exercise 4

- Kleinberg Tardos Chapter 6 Exercise 19. Can you do it in $O(|s| \cdot |x| \cdot |y|)$ time.

Lecture 16-18 (Sep 20-23)

- Let $A = \{a_1, a_2, \ldots, a_k\}$ be an alphabet of size k. You are given a code that maps each letter in A to a 0-1 string. You have decide whether the code is uniquely decodable. That is, are there two different strings over A whose encodings are same?

  For example, consider the following code: $a \to 01, b \to 010, c \to 00, d \to 10$. Here the two strings bb and acd have the same encoding. Thus, this code is not uniquely decodable.

  For another example, consider the following code: $a \to 01, b \to 10, c \to 0100, d \to 110$. This is uniquely decodable. Why? Suppose there are two different strings S_1, S_2 having same encoding. One of them (say S_1) must start with a and the other (S_2) must start with c, because $enc(a)$ is a prefix of $enc(c)$ and there is no other such pair of letters. Then the next letter in S_1 must be one whose encoding starts with 00, but there is no such letter.

  Design an efficient algorithm to decide whether a given code is uniquely decodable.

- Suppose you have a text where there is some letter whose frequency is more than 0.4. Prove that Huffman coding will map at least one letter to a length 1 encoding (not necessarily the letter with 0.4).

- Suppose we have a black and white image with 10^6 pixels (black or white). It is represented by a 0-1 string of length 10^6. Suppose at least 0.9 fraction of the bits are 0. We want to use Huffman coding to compress the image. Divide this string into 5 lakh chunks of two bits each. Each chunk will be seen as a character. There are four possible values any two bits can take – 00,01,10,11. Observe that 00 will be much more frequent than the other three. So, we go with the following code.

  $00 \to 0$

  $10 \to 10$

  $01 \to 110$

  $11 \to 111$

  There are many such images that we want to compress. The only information we are given is that each image has at least 0.9 fraction of the bits being 0. We don't know the exact frequencies of 00, 01, 10, 11. Suppose there is a particular image that just has repeated occurrences of 1 followed by nine 0's. For this image, the frequencies of (00,10,01,11) will be roughly like (0.8, 0.2, 0.0, 0.0). Our coding scheme will give 1.2 bits per character. That means your compressed output will have a size (12/20) of the original size. What do you think is the worst example of an image with respect to our coding. Can you give an upper bound on the compression ratio that is guaranteed for every image in our collection?

  What if your collection had images with at least 0.7 fraction of 0's. What is the guarantee you have now? Do you always get a compression? If not then can your situation improve if you instead used chunks of 3 bits?