TIN093 Algorithms - Assignment 2

Mirco Ghadri

September 1, 2024

Problem 3

a) The distance between 2 consecutive houses H_i and H_{i+1} is denoted as x_i and the distance between any pair of houses i and j is denoted as $y_{i,j}$. If we naively calculate the distance between each pair of house, by summing the values of the distances of the houses between them, we get a running time of $O(n^3)$. Why? Assume we start at House $i = 1$ and take the distance to each House $j = 2 \cdots n$. The distance from house 1 to house 2 becomes $y_{ij} = y_{12} = x_1$. The distance to house 3 becomes $y_{1,3} = x_1 + x_2$ since we sum up the distances in between. The distance to any house j from house $i = 1$ is calculated as:

$$y_{i,j} = \sum_{i=1}^{j-1} x_i \tag{1}$$

Assume we calculate the distance x_i in constant time and let t denote this time, i.e $t(1)$. If we sum up 2 x_i values, this then takes $t(2)$ time. If we sum up n x_i values, this takes $t(n)$ time. Hence, the time to calculate any $y_{i,j}$ value takes $t(j - i)$ time, because we sum up $j - i$ x_i values. Why is this important? Because it allows us to derive the time bound of calculating the distance from a fixed house i to all other houses j that are greater than it. Let $T(i, n)$ denote the total time taken to calculate the distance from house i to all other houses larger than it. For the case $i = 1$ we get:

$$T(i, n) = \sum_{k=1}^{n-1} t(i). \tag{2}$$

which is an arithmetic sum of the form $1 + 2 + 3 \cdots n - 1$. An arithmetic sum of this form has the general formula:

$$\sum_{i=1}^{n} i = n(n + 1)/2. \tag{3}$$

For the general case, it becomes:

$$T(i,n) = \sum_{k=1}^{n-i} t(k) \tag{4}$$

which is also an arithmetic sum of the form $1+2+3\cdots n-i$. Now, we take the sum of $T(i,n)$ for all possible values of i, i.e $i = 1\cdots n-1$. Let T denote the total time this takes. We have:

$$T = \sum_{i=1}^{n-1} T(i,n) = \sum_{i=1}^{n-1}\sum_{k=1}^{n-i} t(i) = \sum_{i=1}^{n-1} t\left(\frac{i(i+1)}{2}\right) \tag{5}$$

This can be simplified further:

$$\sum_{i=1}^{n-1} t\left(\frac{i(i+1)}{2}\right) = \frac{1}{2}\cdot\sum_{i=1}^{n-1} i(i+1) = \frac{1}{2}\cdot\sum_{i=1}^{n-1}(i^2+i) = \frac{1}{2}\cdot\left(\sum_{i=1}^{n-1} i^2 + \sum_{i=1}^{n-1} i\right) \tag{6}$$

This can be solved and simplified further. We know the formulas for the sum of squares and the sum of the first $n-1$ natural numbers:

$$\sum_{i=1}^{n-1} i^2 = \frac{(n-1)n(2n-1)}{6}$$

$$\sum_{i=1}^{n-1} i = \frac{(n-1)n}{2}$$

Substituting these into the expression gives:

$$\frac{1}{2}\cdot\left(\frac{(n-1)n(2n-1)}{6} + \frac{(n-1)n}{2}\right)$$

Next, we'll find a common denominator to add these fractions. The least common denominator between 6 and 2 is 6, so:

$$= \frac{1}{2}\cdot\left(\frac{(n-1)n(2n-1)}{6} + \frac{3(n-1)n}{6}\right)$$

Now, combine the fractions:

$$= \frac{1}{2}\cdot\frac{(n-1)n\left((2n-1)+3\right)}{6}$$

Simplify the expression inside the parentheses:

$$= \frac{1}{2}\cdot\frac{(n-1)n(2n+2)}{6}$$

Factor out the 2 from $2n+2$:

$$= \frac{1}{2}\cdot\frac{(n-1)n\cdot 2(n+1)}{6}$$

Simplify by canceling out the 2:

$$= \frac{(n-1)n(n+1)}{6}$$

So we have that the total time it takes to find the distance between all pairs of houses using the naive approach is:

$$T = \frac{(n-1)n(n+1)}{6} \in O(n^3) \tag{7}$$

The question is, can we do better?

b) Instead of calculating the distance between 2 pairs of houses which are not neighbors by summing up all the distances in between them, we can use a more clever approach. We use the fact that the distance y_{ij} can be written as:

$$y_{ij} = y_{ij-1} + x_{j-1} \tag{8}$$

So if we calculate the distance between House 1 and House 2, we don't need to calculate the distance between House 1 and House 3 by summing up the distances in between House 1 and House 3. Instead we can use the fact that the distance between House 1 and House 3 is equal to the distance between House 1 and House 2(which we already calculated) and the distance between House 2 and House 3. By doing it this way, the time it takes to find the distance between any 2 pair of houses is constant.
We have $\binom{n}{2}$ unique pair of houses. If we spend t(1) time finding the distance between each pair of house, this then becomes:

$$\binom{n}{2} = \frac{n(n-1)}{2} \implies t\left(\frac{n(n-1)}{2}\right) \in O(n^2) \tag{9}$$

c) The time bound of $O(n^2)$ can not be further improved. The reason is that we have $\binom{n}{2}$ unique pair of houses. If we spend constant time finding the distance between any pair of house, as we did in b), it will still take $O(n^2)$ time. The best algorithm we could use would be as follows:

1. Start at i=1 and find the distance to all houses $j = 2 \cdots n$.
2. For i=k, to find the distance from i to all other houses $j = k+1 \cdots n$, use the formula:

   $$y_{ij} = y_{1j} - y_{1i} \tag{10}$$

   Since y_{1j} and y_{1i} have already been calculated for all possible values, this indeed takes constant time.

Problem 4

a) A more clever algorithm would use the fact that the size of the warehouse rooms are already sorted in descending order. We start by selecting the left-most/largest room s_1 and the rightmost/smallest room s_n. 3 things can happen:

- $s_1 + s_n = s$ In this case we have already solved the problem and we stop.
- $s_1+s_n > s$ In this case, we know that we must select a room of smaller size, so we select s_2 and keep s_n unchanged (since s_1 added to any other room will be larger than s). Notice how we immediately remove all pairs containing s_1 from further consideration. This removes O(n) pairs just with 1 comparison which explains why our algorithm is so efficient.
- $s_1 + s_n < s$ In this case, we know that we must select a room of larger size, so we select s_{n-1} and keep s_1 unchanged (since s_n added to any other room will be smaller than s). Notice how we remove all pairs containing s_n from further consideration.

We repeat this algorithm. In each step, we either increase the number of the leftmost room or decrease the number of the rightmost room. We do this until we find a room where $s_{left} + s_{right} = s$ or until $s_{left} = s_{right}$ in which case we have found no solution. Clearly this algorithm runs in $O(n)$ time since we do at most $O(n)$ comparisons because the distance between the leftmost room and rightmost room decrease by 1 in each iteration of the algorithm. But how can we be sure that we have not missed any potential solution?

b) The clever algorithm we use does not try all pairs of rooms, since there are $\binom{n}{2} \in O(n^2)$ rooms and our algorithm runs in $O(n)$ time. So how can we be sure that it does not miss any potential solution? It is simple. We start with the rooms s_1 and s_n. We have 3 situations:

- $s_1 + s_n = s$ In this case we have found our solution and we stop
- $s_1 + s_n > s$ In this case we know that s_1 added to any other room will be larger than s, since if we add it to the smallest value it is larger than s. Hence we delete all pairs that s_1 is a part of and we move the left cursor 1 step to the right. This means we can ignore all pairs that s_1 is a part of in the future. We then repeat the same algorithm.
- $s_1 + s_n < s$ In this case, we know that s_n added to any other room will be smaller than s, since s_n added to the largest room is smaller than s. So we can delete all pairs that s_n is a part of. We then move the right cursor 1 step to the left and perform the same algorithm without considering s_n again, which we can safely do.

What does this tell us? It tells us that any step of the algorithm where the left-most room is denoted by s_{left} and the rightmost-room is denoted by s_{right}, we have excluded the possibility of all solutions containing the room s_{left-1} and earlier, and excluded all solutions containing the room $s_{right+1}$ and later. This proves that the algorithm works and it does so without explicitly considering all pairs of rooms.