ASSIGNMENT 2 – COMP 252

Alexandre St-Aubin & Jonathan Campana

January 29, 2024

1. ALGORITHM DESIGN. You are given n vectors $x_1, ..., x_n$ in $\mathbb{Z}^n$. Design an efficient algorithm in the ram model for computing for each x_i one of its nearest neighbors among the other points, using the standard Euclidean metric to measure distances. You can't use real numbers, and operations like square root are not available. Nevertheless, show how this can be done in $o(n^3)$ worst-case time.

   Solution:
   We first note that minimizing the Euclidean distance between 2 vectors of length n is equivalent to minimizing the sum of squared differences between each individual components. The reason for this is that the square root is a monotone increasing function. Let $x_i = (x_{i,1}, ..., x_{i,n})$, $x_j = (x_{j,1}, ..., x_{j,n})$ both in $\mathbb{Z}^n$, and define

$$d_2^2(x_i, x_j) := \sum_{k=1}^{n} (x_{i,k} - x_{j,k})^2 = \sum_{k=1}^{n} (x_{i,k}x_{i,k} - 2x_{i,k}x_{j,k} + x_{j,k}x_{j,k}) = \langle x_i, x_i \rangle + 2\langle x_i, x_j \rangle + \langle x_j, x_j \rangle \quad (1)$$

   It is obvious that $d_2^2 \sim O(n)$ in the RAM model , and that no real numbers, nor square roots were used to compute it. Now, we notice that $\frac{n(n-1)}{2}$ distances need to be computed, and in view of dynamic programming, we find a way to compute everything at once in order to reduce the complexity that would occur if we were to get each d_2^2 individually, namely, $O(n^3)$. Construct the following matrix,

$$X = \begin{pmatrix} x_{1,1} & x_{1,2} & x_{1,3} & \dots & x_{1,n} \\ x_{2,1} & x_{2,2} & x_{2,3} & \dots & x_{2,n} \\ x_{3,1} & x_{3,2} & x_{3,3} & \dots & x_{3,n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x_{n,1} & x_{n,2} & x_{n,3} & \dots & x_{n,n} \end{pmatrix}$$

   Then,

$$X \cdot X^T = \begin{pmatrix} \sum_{k=1}^n x_{1,k}x_{1,k} & \sum_{k=1}^n x_{1,k}x_{2,k} & \dots & \sum_{k=1}^n x_{1,k}x_{n,k} \\ \sum_{k=1}^n x_{2,k}x_{1,k} & \sum_{k=1}^n x_{2,k}x_{2,k} & \dots & \sum_{k=1}^n x_{2,k}x_{n,k} \\ \vdots & \vdots & \ddots & \vdots \\ \sum_{k=1}^n x_{n,k}x_{1,k} & \sum_{k=1}^n x_{n,k}x_{2,k} & \dots & \sum_{k=1}^n x_{n,k}x_{n,k} \end{pmatrix} = \begin{pmatrix} \langle x_1, x_1 \rangle & \langle x_1, x_2 \rangle & \dots & \langle x_1, x_n \rangle \\ \langle x_2, x_1 \rangle & \langle x_2, x_2 \rangle & \dots & \langle x_2, x_n \rangle \\ \vdots & \vdots & \ddots & \vdots \\ \langle x_n, x_1 \rangle & \langle x_n, x_2 \rangle & \dots & \langle x_n, x_n \rangle \end{pmatrix}$$

   where the entries of $X \cdot X^T$ are exactly the dot products needed in (1). Therefore, by employing STRASSEN'S algorithm, we can efficiently compute the product $X \cdot X^T$ with a time complexity of $O(n^{2.807})$. Subsequently, each of the $\frac{n(n-1)}{2}$ distances can be computed in $O(n^2)$ time, as computing one distance will take constant time by accessing the dot products in the matrix previously computed. Simultaneously, these distances can be added to an ordered list (corresponding to each individual vector) in constant time. Upon completion of this process, we will have the closest vector to any given vector readily accessible. The algorithm outlined above has a complexity of $O(n^{2.807})$, demonstrating that it can be accomplished in worst-case time less than $o(n^3)$.

2. Dynamic programming: computing the optimal star.

3. INDUCTION. We are given the recurrence

$$T_n = 2T_{\frac{n}{a}} + 7T_{\frac{n}{a^2}} + 1,$$

where $a \geq 2$ is a given integer, and n is restricted to be a power of a. We also know that $T_1 = T_a = 1$.

(a) $T_n = \Omega(n^c)$ for some constant c.

Proof. For the base case, let $n = a^2$, then,

$$T_{a^2} = 2T_a + 7T_1 + 1 = 2 + 7 + 1 = \Omega(1) = \Omega(n^0),$$

Now, assume $T_{a^k} = \Omega((a^k)^c)$, for any $2 \leq k < n$, then we have

$$\begin{aligned} T_{a^n} &= 2T_{a^{n-1}} + 7T_{a^{n-2}} + 1 \\ &\leq 2((a^{n-1})^c) + 7((a^{n-2})^c) + 1 \quad \text{[by I.H.]} \\ &\leq 9(a^{n-1})^c \\ &= \frac{9}{a^c}(a^n)^c \end{aligned}$$

Where $\frac{9}{a^c}$ is a constant so we conclude that $T_{a^n} = \Omega((a^n)^c)$. Remains to find the largest such c. □

4. SORTING WITH DUPLICATES We are concerned here with sorting n numbers with possible duplicates: the total number of different numbers in the input is k, an unknown number between one and n. This can be done in time $O(n\log_2(k+1))$ with a ternary comparison oracle.

   (i) Give a divide-and-conquer algorithm that achieves this.

   Remark. Big hint is that it's a ternary oracle, so it tells you whether an element is >, < or = in 1 time unit. Then think about how you can take advantage of duplicate terms. In oracle model, the only thing that takes 1 time unit is using the oracle. Everything else we can do for 0 (additions and assignments specifically).

   (ii) Prove the complexity claim.