Lecture 3: Nearest Neighbors I

1/13/2020

Lecturer: Kanat Tangwongsan *Scribe: Kriangsak THUIPRAKHON*

1 Example: Depth of Treap

the previous lecture, we have proven that the height of a treap to be fairly balanced with the expected depth of $\log_2 n$. Here is another derivation of the height of a treap using chernoff hoeffding bounds.

Let $A_{i,j}$ be an indicator random variable where,

$$A_{i,j} = \begin{cases} 1 & \text{if } j \text{ is an ancestor of } i \\ 0 & \text{otherwise} \end{cases}$$

Note that for a fix i, all $A_{i,j}$'s are independent, meaning Chernoff-Hoeffding bounds apply.
In this case, the depth of the treap is,

$$\begin{aligned} \text{depth}(X_i) &= \sum_{j=1}^{n} A_{1,j} \\ &= 1 + \underbrace{\sum_{j=1}^{i-1} A_{i,j}}_{L} + \underbrace{\sum_{j=i+1}^{n} A_{i,j}}_{R} \end{aligned}$$

In the previous lecture, it was shown that the height of a treap is about $\log_2 n$ with high probability. This mean we want to bound something in the following form
Claim:

$$Pr[\text{Depth} \leq k \ln n\,] = 1 - \underbrace{Pr[\text{Depth} > k \ln n\,]}_{*} \geq 1 - \frac{1}{n^\alpha}$$

where the term $\frac{1}{n^\alpha}$ is the error probability mentioned in the previous lecture. The goal now is to bound $*$ using union bound.

$$Pr[\,\text{Depth} > k \ln n\,] \leq Pr[\,\text{left} > \frac{k}{2} \ln n\,] + Pr[\,\text{right} > \frac{k}{2} \ln n\,]$$

Applying the powerful Chernoff-Hoeffding bounds, we then have

$$Pr[\text{ left} > (1 + \frac{k}{2} - 1)\ln n\,] \le \exp\{-\frac{(\frac{k}{2}-1)^2}{3} \times \underbrace{2\ln n\}}_{*}$$
$$\le \frac{1}{n^\alpha} \quad \text{where } \alpha = \frac{(\frac{k}{2}-1)^2}{3}$$

Note that * is bounded by the deepest height of a treap. The same taken to the second term, we will have

$$\mathbb{P}[\text{depth} \le k \ln n] \ge 1 - \frac{1}{n^\alpha}$$

2 The Power of Grid

In this last bit of the lecture, we will explore the a simple idea whereby the space is sub-divided into grid cells. By doing so, it turns out to be extremely powerful.

2.1 Closest Pair in 2D

Given the input as a set ***P*** of *n* points in 2D, and the goal is to compute the closest pair of points. There are a few ways to solve such a problem. One may try to go about writing two for-loops but of course the algorithm delivers with $O(n^2)$. Another approach to this problem is to solve it using divide and conquer yeilding $O(nlogn)$. However, there is a faster, and simpler, algorithm that delivers the closest pair in $O(n)$, with allowance of additional operations.

2.2 Verification

let $CP(Q)$ denote the distance between the closest pair of points of Q, where Q is a set of points. if someone claims CP(Q) is *r*. How do we verify this quickly?
We can in fact do this in linear time:

- Build a grid of size *r* by *r*
- Dump all the points into the cells by bucketing/hashing. In fact, **Q(x,y)** goes to the coordinate $(\lfloor x/r \rfloor, \lfloor y/r \rfloor)$.
- If any cell has more than 9 points, $CP(Q) < r$. Note that if the cell is sub-divided into a 3-by-3 sub-cells, Pigeon hole says one of them has at least two points, but then this sub-cell's diameter is strictly less than r
- For each operation, we will look at at most 81 points. That is, we will be looking at the neighboring 8 points for all sub-cells containing at most 9 points.

- Hence, linear time!

2.3 A Closest Pair Algorithm

1. Permute the given input set $\boldsymbol{P}$ of points randomly, say $\boldsymbol{P}$ = $\langle p_1, p_2, ..., p_n\rangle$ is a permutation of the given input.
2. start with $r_2 = ||p_1 - p_2||_2$. In this case, the above grid will be sub-divided into an r-by-r grid where $r = r_2$
3. $\forall i \in \{3, 4, 5, ..., n\}, p_i$ will be added . When added, there algotithm shall perform:
   - Check whether p_i forms a new closest pair.This requires the checking of p_i to its original and other 8 neighboring cells, all of which contain at most 9 points.
   - If p_i becomes the new closest pair, rebuild the grid using the new closest pair distance.
   - Continue with this grid otherwise.

   Let $\boldsymbol{X}$ be the indicator random variable,

$$X_i = \begin{cases} 1 & \text{if } p_i \text{ forms a new closest pair} \\ 0 & \text{otherwise} \end{cases}$$

The total cost is then given by the resurrance,

$$\boldsymbol{T}(n) = \boldsymbol{O}(1) + \sum_{i=3}^{n} (\boldsymbol{O}(1) + X_i \cdot i)$$

Claim: $\mathbb{P}[\boldsymbol{X}_i = 1] \leq \frac{2}{i}$, where p_i is randomly drawn from the permuted input. The probability of a point randomly drawn forms the new closest pair is given by two cases:

- The closest pair is unique and they are x–y. Then either one of them will be picked as p_i with $2/i$ probability.
- The closest pair is not unique, then the probability that p_i becomes the ***new*** closest pair is 0.

Hence, the expected running time is then,

$$\begin{aligned} \mathbb{E}[\boldsymbol{T}(n)] &\leq \boldsymbol{O}(1) + n + \underbrace{\sum_{i=3}^{n} [\boldsymbol{X}_i \cdot i]}_{2/i} \\ &= \boldsymbol{O}(n) + \sum_{i=3}^{n} [\frac{2}{i} \cdot i] \\ &= \boldsymbol{O}(n) \end{aligned}$$