15-750: Graduate Algorithms January 18, 2016

Lecture 17: 2D Closest Pair using Hashing

Lecturer: Gary Miller *Scribe: Yanzhe Yang, Yao Liu*

1 Introduction to Hashing and the Problem

Imagine that we want to maintain a set of keys, which belong to a huge space of keys $\mathcal{U}$. Often, the space can be larger than universal large constant:

$$|\mathcal{U}| \geq 10^{100} \geq 10^{60}$$

However, the size of key set $K \subset \mathcal{U}$ would not be such huge. Usually $|K| \approx 10^{15}$. If we want to maintain the set of keys so that we can insert, lookup, and delete any keys, our solution so far has been maintaining a ordered set so that we can do those options in $O(\log n)$ times.

1.1 Hashing

We can also maintain such a set by hashing. Hash function is a random mapping from the space of key $\mathcal{U}$ to another smaller space T (table). Usually we set the size of T, denoted $|T|$, to be approximately equal to $|K|$. We know that hashing method has the property:

Claim 1.1. *Hashing table can insert, lookup, or delete any single element in $O(1)$ expected time.*

1.2 The Closest Pair Problem

The closest pair problem is defined as:

$$\begin{aligned} \text{Input:} \quad & P \subset \mathbb{R}^2;\ P \subset \text{Unit Box};\ |P| = n \\ \text{Output:} \quad & CP(P) = \underset{p \neq q, p, q \in P}{\operatorname{argmin}} \|p - q\|_2 \end{aligned}$$[1]

Now we place the points into boxes using hashing. The main idea is that we partition unit box into boxes of side length α. We define boxes partition G_α as a grid partition of boxes with length of α. See figure 1. Note that if α is small, say 10^{-10} = one over 10 billion, then 10^{20} boxes is too big!

Recall that we can use hashing to map the keys to a smaller space rather than key universe. Here, in this problem, the key universe is the universe of name of all boxes. The key space is the space of names of boxes containing a point. Note that there are n points in total, so the size of hash table is $O(n)$ (some boxes might contain several points). We also can use dynamic sizing to maintain the hash table.

Lemma 1.2. *Hashing points into its box is O(1) time.*

Definition 1.3. (extended neighbor) If B is a box of G_α. Then the extended neighbor of B is all the 9 boxes next to B and B itself. See figure 2. We denote it $\text{Ext}(B)$.

[1] For the compactness, later we may also treat the output of CP as the distance between p and q. That is because we want to be consistent with Gary's lecture note in this definition and later, also that is easy to compute $\|p - q\|_2$ in $O(1)$ time given q and p.

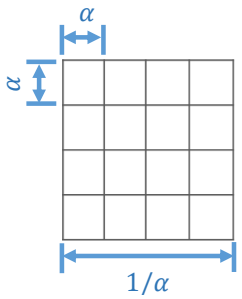


Figure 1: A grid boxes partition G_α. There are in total $\left(\frac{1}{\alpha}\right)^2$ boxes in the partition.

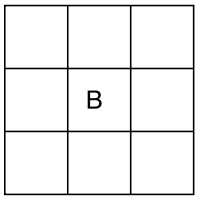
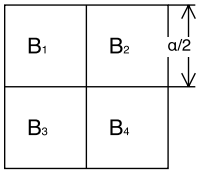


Figure 2: Ext(B)

Lemma 1.4. *(Packing Lemma) Let B be a box with side length α, $\alpha \leq CP(P)$, and $P \subseteq B$. Then $|P| \leq 4$.*

Proof. Split B into 4 boxes:



The diameter of each B_i is $\alpha/\sqrt{2} \leq \alpha \leq CP(P)$. Thus each B_i contains at most one point. □

2 Test α Algorithm

Before introducing the algorithm to calculate $CP(P)$, we need a procedure satisfying

Definition 2.1.

$$Test(\alpha > 0, P) = \begin{cases} \beta < \alpha & \text{if } \exists p \neq q \in P, \text{ s. t. } \|p - q\|_2 = \beta < \alpha \\ \alpha & \text{if CP}(P) = \alpha \\ \text{False} & \text{otherwise} \end{cases}$$

Here is a procedure $Test(\alpha, P)$:
Let $\mathbb{P}_i = \{P_1, P_2, ..., P_{i-1}\}$.

Algorithm 1 Test(α, P)

1: Make hash table H_α for grid G_α
2: Insert P_1 into G_α
3: **for** $i = 2$ to n **do**
4: Insert P_i into its box B
5: Compute $\min dist(P_i, \mathbb{P}_i \bigcap Ext(B)) = \beta$
6: **if** $\beta < \alpha$ **then**
7: **return** "$CP(P) \leq \beta < \alpha$"
8: **end if**
9: **if** $\beta = \alpha$ **then**
10: Flag $\leftarrow$ true
11: **end if**
12: **end for**
13: **if** Flag **then**
14: **return** "$CP(P) = \alpha$"
15: **else**
16: **return** "$CP(P) > \alpha$"
17: **end if**

Note: From Packing Lemma, there are at most 4 points in each box when the procedure is running the lines $3-5$ (otherwise the procedure would terminate at step 7 in the previous round of the cycle) , so Ext(P) contains at most 36 points. So computing $\min dist(P_i, \mathbb{P}_i \bigcap Ext(B))$ is $O(1)$.

Claim 2.2. *Test is linear time and correctly tests the relationship between CP(P) and* α.

3 2D CP Algorithm

Algorithm 2 $\overline{CP}(P)$

Input: $P \subseteq UnitBox$
Output: α

1: **if** $n \leq 4$ **then**
2: Check all pairs.
3: **end if**
4: Randomly permute $\mathbb{P} = \{P_1, ..., P_n\}$
5: $\alpha \leftarrow 1$
6: **while** $Test(\alpha, P) =$ "$\beta < \alpha$" **do**
7: $\alpha \leftarrow \beta$
8: **end while**
9: **return** α

3.1 Correctness

If $n \leq 4$, done.
By Lemma 1.4, if $n > 4$, then $\alpha < 1$.

3.2 Backward Analysis

Theorem 3.1. *$\overline{CP}(P)$ is expected linear time.*

Definition 3.2. α_i be random variable. $\alpha_i = CP(P_1, ..., P_i)$, $i \geq 2$.

Note that $\alpha_{i+1} \leq \alpha_i$. And we restart $Test$ (1) for each i s.t. $\alpha_i < \alpha_{i-1}$.
Then we need to know $Prob(\alpha_i < \alpha_{i-1})$. There are three cases.

1. There exists a unique closest pair within $\{P_1, P_2, ..., P_i\}$, say (P_j, P_k).
   Then, only removing P_j or P_k will cause a restart, so the $Test$ runs only when $j = 1$ or $k = i$.
   So $Prob(\alpha_i < \alpha_{i-1}) = \frac{2}{i}$.

2. There exists multiple closest pairs, and all these pairs include one point, say P_j (shown in Figure 3: Left). Then we need to restart only if we remove P_j. So, $Prob(\alpha_i < \alpha_{i-1}) = \frac{1}{i}$

3. There are two or more disjoint closest pairs (shown in Figure 3: Right). If we remove only one point from these pairs, we can see the closest distance will not change. So, we don't need to restart anymore, and $Prob(\alpha_i < \alpha_{i-1}) = 0$.

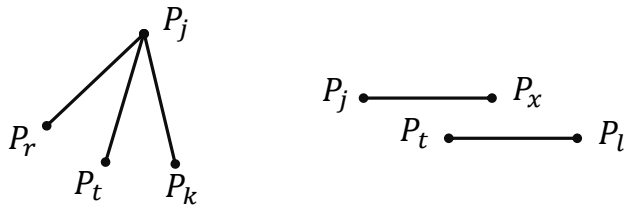


Figure 3: Left: Case 2. Right: Case 3

From the three cases, we learn $P(\alpha_i < \alpha_{i-1}) \leq \frac{2}{i}$. Each restart is $O(i)$ new work. So, the total expected work is

$$O\left(\sum \left(\frac{2}{i}\right) i\right) = O(n)$$

.