ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Institute of Theoretical Computer Science
Bernd Gärtner, Rasmus Kyng, Angelika Steger, David Steurer, Emo Welzl

Solution 1

We prove the lemma by induction on the size n of the insertion permutation (or equivalently, the resulting tree).

Induction base case. If $n = 0$ or $n = 1$, the lemma trivially holds.

Induction step. Let $n \geq 2$ and suppose that the lemma holds for all insertion permutations of size strictly less than n. Let $\pi = (\pi(1), \ldots, \pi(n))$ be a permutation drawn uniformly at random from $\mathfrak{S}_n$. The first element $\pi(1)$ will become the root of the tree T_π. Since the distribution of $\pi(1)$ is u.a.r. from $[n]$, the root of the tree is chosen uniformly at random, as required by the construction of $\tilde{\mathcal{B}}_{[n]}$.

Now let us tackle the two subtrees of the root: Let $k \in [n]$. We want to show that, conditioned on $\pi(1) = k$, the distribution of the left subtree of the root is the same as $\tilde{B}_{\{1,\ldots,k-1\}}$, and the distribution of the right subtree of the root is the same as $\tilde{B}_{\{k+1,\ldots,n\}}$. To this end, let π^- be the sequence of elements in π smaller than k, and let π^+ be the sequence of elements in π larger than k. Note that the insertion sequence will send the keys in π^- (π^+) in exactly this order to the left (right) subtree of the root. Since π^- and π^+ are uniformly random permutations of their respective element sets $\{1, \ldots, k-1\}$ and $\{k+1, \ldots, n\}$, we can apply the induction hypothesis to obtain that the left and right subtrees of the root are distributed just as we stated.

We conclude that this process produces a tree T_π that is distributed like $\tilde{\mathcal{B}}_{[n]}$.

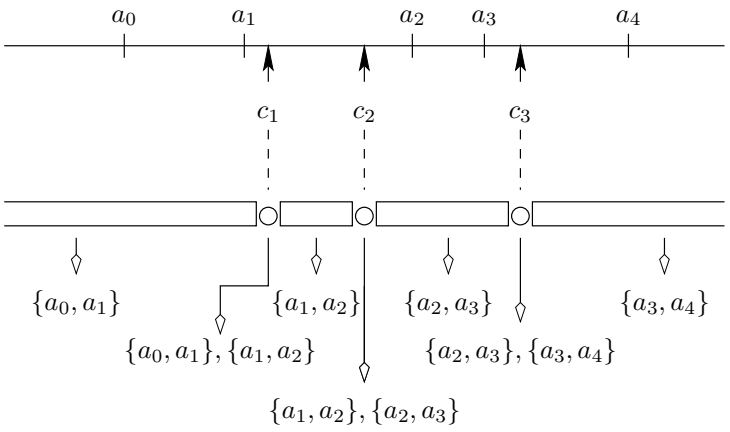
Solution 2

This problem lends itself to a straightforward solution using the "locus approach" as described in the lecture. First we have to find the regions of equal answer. While it is trivial to see where the regions with the same closest point lie, it is not so obvious for the closest pair. Note that when we move a query point from $-\infty$ to $+\infty$, we will see each pair $\{a_i, a_{i+1}\}$ as the correct answer for some interval. The transition from $\{a_{i-1}, a_i\}$ to $\{a_i, a_{i+1}\}$ occurs as soon as a_{i+1} is closer to q than a_{i-1}. The threshold position for this is obviously at $(a_{i-1} + a_{i+1})/2$. Therefore we may proceed as follows.

First sort the n elements $a_i \in S$ such that $a_0 < a_1 < \cdots < a_{n-1}$. Then partition the real line into the following intervals and points:

$$(-\infty, c_1), c_1, (c_1, c_2), c_2, (c_2, c_3), c_3, \ldots, c_{n-2}, (c_{n-2}, \infty),$$

where $c_i := (a_{i-1} + a_{i+1})/2$ for every $i \in \{1..n-2\}$. The next picture schematically shows the construction.



Each interval is associated with a nearest neighbor pair. At the points c_i, there are two possible pairs and the way the task is described, the answer for $c_i = q$ is ambiguous as both $\{a_{i-1}, a_i\}$ and $\{a_i, a_{i+1}\}$ fulfill the condition.

Given this data structure and a query point q, a pair of nearest neighbors can be found in $\mathcal{O}(\log n)$ time. Preprocessing can be done in $\mathcal{O}(n \log n)$ time by any optimal sorting algorithm and $\mathcal{O}(n)$ space.

Solution 3

Let us first give the proof the exercise asked for. Then we would like to elaborate on the purpose of this exercise.

Proof. Observe that $\ell : y = 2kx - k^2$ is the tangent of the parabola $g : y = x^2$ at point (k, k^2): The slope of line ℓ is $2k$ which is the derivative of g at position k and (k, k^2) lies on both, ℓ and g. Thus $k = a_i$ for any $i \in \{0, \ldots, n-1\}$ implies that (a_i, a_i^2) lies on ℓ and the polygon C. For the other direction let us assume that C intersects ℓ at some point p. Because the parabola is a strictly convex function[1], g touches C only at the points (a_i, a_i^2). Therefore by also using that ℓ is a tangent of g, we get $p = (a_i, a_i^2)$ for some $i \in \{0, \ldots, n-1\}$. □

This exercise showcases a way to prove a lower complexity bound for a geometric problem. Such so-called *negative results* ("there is *no* algorithm better than...") are often very hard to prove because the argument has to go over all possible algorithms, a family of objects so complex that we are today hardly able to understand it. In the present

[1] Recall: a function $f : \mathbb{R} \to \mathbb{R}$ is strictly convex iff $\forall \lambda \in (0,1), \forall x_1, x_2 \in \mathbb{R}, x_1 \neq x_2$,

$$\lambda f(x_1) \; + \; (1-\lambda) f(x_2) \; < \; f\big(\, \lambda x_1 \; + \; (1-\lambda) x_2 \big)\,.$$

case, the above geometric considerations show that deciding whether a (query) line hits a (preprocessed) convex polygon consisting of n points cannot be any easier than deciding whether a (query) number $q \in \mathbf{R}$ is contained in a (preprocessed) set of n keys $S \subseteq \mathbf{R}$. Because if that were easier, then we could preprocess the set S of keys by generating the points $P := \{(x, x^2) | x \in S\}$, preprocess them as a polygon and then each time a query $q \in \mathbf{R}$ is asked, instantiate the line $l_q : y = 2qx - q^2$ and use the (faster) algorithm to check whether l_q hits it. Thence if we have a lower bound for the effort needed to answer the query $q \in S$, the same bound applies to the polygon and query line problem. So do we have such a lower bound?

You have learnt in your first year already that searching for a key in a set of n keys needs a least $\Omega(\log n)$ steps, which is also achievable, e.g. through binary search. *Searching* here means to say for the query number q, which is the next-smallest (or next-largest, or both) key in S. The argument there was the following: Since there are at least $n + 1$ possible answers the algorithm needs to be able to give, at least $\lceil \log(n + 1) \rceil$ bits of information have to be acquired to distinguish between them. So if the only thing we can do with the numbers is comparing two of them, we need at least $\lceil \log(n + 1) \rceil$ comparisons.

Does this now imply together with the exercise that deciding whether a query line hits a polygon needs at least $\Omega(\log n)$ steps and the algorithm we saw in the lecture is therefore optimal? Not yet really, we have to be extremely careful. Here are two flaws: firstly, we have proved that deciding whether a query line hits a polygon is as difficult as deciding whether a real q is in a set of reals S. That's not the same as *searching*, it is a decision problem. Nobody asks you to say which is the next-smallest or next-largest key in S, you just have to say *yes* or *no*. That might be easier than the lower bound we know. Secondly, what was proved in earlier classes was that searching in a set S of comparable keys which you cannot do anything else with but *comparing* them takes $\Omega(\log n)$ of these comparisons. But here we do not have such a set of keys, we have *real numbers*. There is more we can do with real numbers than comparing. The computational model we are considering when designing geometric algorithms is usually a model where basic operations like addition, subtraction, multiplication, division and taking roots can be evaluated in constant time. So if we really want to prove that our algorithm is optimal in this powerful model of computation, we have to prove that every algorithm that just *decides* whether a number $q \in \mathbf{R}$ is in a preprocessable set S of keys carries out at least $\Omega(\log n)$ basic arithmetic operations.

Proving this is very cumbersome and would exceed the scope of this course by far. But Michael Ben-Or has done it in [1].

Theorem 1 (Ben-Or). *Consider a computational model operating on reals by executing the basic operations addition, subtraction, multiplication, division, taking of the square root and comparisons ($<$, $>$ and $=$). Then for any preprocessed set $S \subset \mathbf{R}$ of size $|S| = n$, any algorithm answering for a query $q \in \mathbf{R}$ whether $q \in S$ needs to carry out in the worst case at least $\Omega(\log n)$ basic operations.*

Using this, we can infer that our $O(\log n)$ time algorithm for the line-hitting-polygon problem is in fact best possible. Even though the result turned out as expected, it has to be distinctly understood that this was by no means a priori obvious.

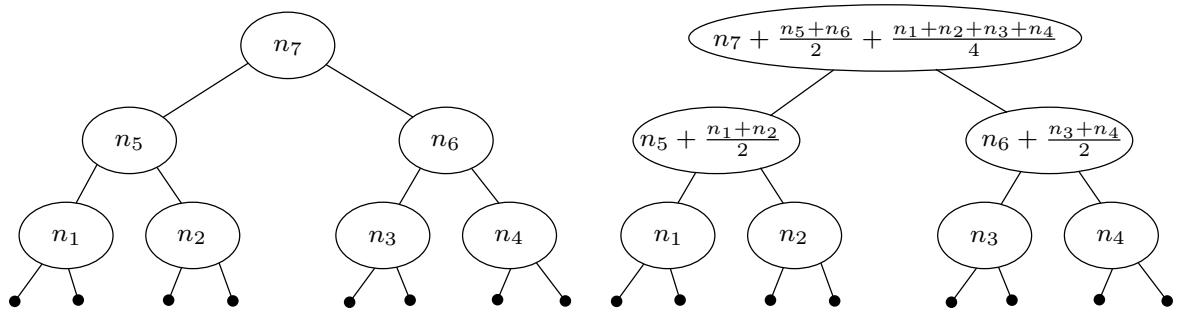
Solution 4

The lemma contains a chain of two inequalities:

$$\sum_{v \text{ inner node}} |\bar{S}_v| \leq 2 \cdot \sum_{v \text{ inner node}} |S_v| \leq 2n^2.$$

The second inequality is easy to derive: The number $|S_v|$ of coordinates for level v can be bounded by the number of edges that belong to level v, and the total number of edges does not exceed n^2.

So it remains to look at the first inequality. Let us first look at an example, so that we may believe the statement. Below we have depicted a tree to store $n = 7$ levels: On the left-hand side, the tree for the original sets S_v, and on the right-hand side, the tree with the 'enhanced' sets $\bar{S}_v$. We have annotated the nodes with the number ($|S_v|$ and $|\bar{S}_v|$, respectively) of x-coordinates stored in that node.



Indeed our example has

$$\begin{aligned}\sum_{v \text{ inner node}} |\bar{S}_v| &= \left(1 + \frac{1}{2} + \frac{1}{4}\right)(n_1 + n_2 + n_3 + n_4) + \left(1 + \frac{1}{2}\right)(n_5 + n_6) + n_7 \\ &\leq 2\,(n_1 + n_2 + n_3 + n_4 + n_5 + n_6 + n_7) \\ &= 2 \cdot \sum_{v \text{ inner node}} |S_v|.\end{aligned}$$

The example suggests that we should group the nodes of the tree by their depth. (We refrain to speak of the 'levels' of the tree, because that will almost certainly cause confusion vis-à-vis the levels v of the line arrangement). Thus, let h denote the height of the tree and for $0 \leq i \leq h$ let

$$m_i := \sum_{\substack{v \text{ inner node} \\ \text{at depth } i}} |S_v|, \qquad \bar{m}_i := \sum_{\substack{v \text{ inner node} \\ \text{at depth } i}} |\bar{S}_v|.$$

Now

$$\sum_{\nu \text{ inner node}} |\bar{S}_\nu| = \sum_{k=0}^{h} \bar{m}_k \leq m_h + \sum_{k=0}^{h-1} \left(m_k + \frac{\bar{m}_{k+1}}{2} \right) = \sum_{\nu \text{ inner node}} |S_\nu| + \frac{1}{2} \cdot \sum_{\nu \text{ inner node}} |\bar{S}_\nu| - \frac{\bar{m}_0}{2}$$

which yields the claim.

References

[1] Michael Ben-Or. *Lower bounds for algebraic computation trees.* In Proceedings of the Fifteenth Annual ACM Symposium on the Theory of Computing STOC, pages 80-86, 1983.