Lecture 18: April 26, 2001

Randomized algorithms:

- Pattern matching
- Close Pair

Pattern Matching

Input: Two strings $x = x_1 \cdots x_n$ (the text) and $y = y_1 \cdots y_m$ (the pattern). (Assume all characters are just bits, for simplicity.)

Task: Check if pattern appears in text and if so, report such an occurrence.

Obvious: For every i check if $x^{(i)} = x_i \cdots x_{i+m-1} = y$,

Running time: $\Theta(mn)$.

Best known: $\Theta(m+n)$, but quite complicated.

Today: Simple (well, almost) and fast algorithm.

Solution Idea

- Let $x^{(i)} = x_i \cdots x_{i+m-1}$.
- Consider the m-bit strings $x^{(1)}$ and y and assume they are distinct.
- Can think of each as a *long* integer and the problem as one of checking if $x^{(1)} = y$.
- Want to reduce this to a small integer comparison (but that later)

Hashing

Want a function h such that

1. For all i we have $h(y) = h(x^{(i)})$ if and only if $y = x^{(i)}$
2. We can compute all values $h(x^{(i)})$ in linear time

For $s = s_0 \ldots s_{m-1}$ define

$$h(s) = s_0 2^0 + s_1 2^1 + \ldots + s_{m-1} 2^{m-1}$$

How to compute $h(x^{(i)})$? Easy, given $h(x^{(i-1)})$, since

$$h(x^{(i)}) = 1/2(h(x^{(i-1)}) - x_{i-1}) + 2^{m-1} x_{m+i-1}.$$

Now we just have to deal with the long integer problem.

Computation modulo random prime

Ideas:

- hash using $h(s) \bmod p$, not just $h(s)$
- pick p to be a random prime; see if $h(x^{(1)}) - h(y) \equiv 0 (\mathrm{mod}\ p)$.

Analyzing the idea

- What kind of primes are bad?
- Suppose $h(x^{(1)}) - h(y) = 0 (\mathrm{mod}\ p)$, then p must divide $h(x^{(1)}) - h(y)$.
- How many such primes exist? At most $\log_2(|x^{(1)} - y|) \leq m$.
- Random prime from a large set is likely to be good!

More details

- Let Π be the set of the $2mn$ smallest primes.
- Can prove that largest element has $O(\log mn)$ bits. (So small we will assume it fits into one memory cell, just like we assume the index $i \in \{1, \ldots, n\}$ fits into one cell.)
- Then for random $p \in \Pi$, the probability that $h(x^{(1)}) - h(y) = 0 (\mathrm{mod}\ p)$ is at most $\frac{1}{2n}$.
- The probability that there exists an index $i \in \{1, \ldots, r$ such that $x^{(i)} \neq y$, but $x^{(i)} - y = 0 (\mathrm{mod}\ p)$ is at most $\frac{1}{2}$.
- In particular, the expected number of "false matches" modulo p is at most $\frac{1}{2}$.
- Thus we get an algorithm correct with constant probability.

Final remarks

- Can verify if a reported "match" is correct $\Rightarrow$ zero-error algorithm
- Food for thought: How to find a random element of the set Π ?

Close pair

- Given: a set P of n two-dimensional points, i.e., $P \subset \{0 \dots U\}^2$; and number $r > 0$
- Want: check if there is any pair of distinct points $p_i, p_j \in P$ such that the distance between them is at most r

Randomization

Idea: Impose a regular square grid onto the plane, such that each cell has side length $4r$.

Fact: For two points p and q within distance $\leq r$ from each other, the probability that p and q end up in the same cell is at least $1/2$.

Proof: The probability that p and q are separated by horizontal line is at most $1/4$, same for vertical line. Thus, the probability that p and q is not separated is at least $1 - 1/4 - 1/4 = 1/2$. So, with constant probability, a "close pair" is located in one bucket.

Our algorithm will have $1/2$ probability of correctness.

Algorithm

- Translate the grid *at random*
- For each cell c containing at least one point, compute the set B_c containing all points in that cell (this can be done in $O(n)$ using hashing)
- For all such sets B_c, search for a pair $p, q \in B_c$ such that p and q are within distance r from each other

How to implement the last step efficiently ? All points could go to the same bucket...

Packing bound

Fact: If $|B_c| \geq 256/\pi$, then B_c *must* contain a "close" pair of points.

Proof: Assume that for all pairs of points $p, q \in B_c$, the distance between p and q is $> r$. Then, if for each point $p \in B_c$ we create a ball of radius $r/2$ around p, then no two balls can intersect. However, the area of intersection of any such ball with the cell c is at least $1/4 \cdot \pi(r/2)^2$. Since the area of the cell c is $(4r)^2$, it follows there are at most

$$\frac{(4r)^2}{1/4 \cdot \pi(r/2)^2} = 256/\pi$$

balls.

Wrapping up

- If there is B_c with cardinality $> 256/\pi$, we know a close pair must exist
- Otherwise, enumeration of all pairs of points from any B_c takes constant time
- There are at most n non-empty sets B_c, so we are done!

Final remark: do we really need to shift the grid at random ? And how to *find* the close pair ?

References

- All the algorithms of this lecture were discovered in the 1970's.
- The pattern matching algorithm is due to R. M. Karp and M. O. Rabin.
- The close pair algorithm is also due to M. O. Rabin (he actually showed how to find the *closest* pair, though).
- A good reference for this lecture is the Chapter on Algebraic Techniques from the book of Motwani and Raghavan.