Extended Algorithms Courses COMP3821/9801

Aleks Ignjatović

School of Computer Science and Engineering
University of New South Wales

Introduction to Randomised Algorithms

What are we going to do in this class

We will do:

- randomised data structures such as skip lists and randomised hash tables;
- randomised algorithms, such as Karger's min-cut algorithm;
- a bit of complexity theory, such as NP complete problems;
- a few approximation algorithms for NP complete problems.

Since probability can be a tricky thing teasing our common sense, we will start with some probability puzzles as a warm up exercise.

Probability Puzzles

Problem: (Monty Hall Problem) A prize is placed behind one of three closed doors. You are allowed to chose one door. After you make your choice, the Quiz Host Monty Hall opens one of the remaining two doors and shows you that the prize is not behind that door (Monty knows where the prize is and always chooses the empty door). You are now given the following choice: you can either stick with the door you have chosen or you can switch to the other door which is still closed. What should you do?

Probability Puzzles

Problem: (Three Cards Problem) There are three cards: one is red on both sides, one is blue on both sides and one is red on one side and blue on the other. You pick a card a random and then choose the side of the car also at random and look at it and it turns out that it is red. What is the probability that the other side is also red?

Probability Puzzles

Problem: (Two Pubs) After work you go to the nearest train station. There is only one train stopping there, going East-West. At the Eastern Platform the train goes to the East; at the last stop there is the Eastern City Pub; at that stop the train turns around and goes back to the West. At the Western platform the train goes to the West, at the last stop is the Western City Pub; at that stop the train also turns around and goes back East. The trains go at perfect 5 minute intervals appart. You always choose the first train to arrive and go to the corresponding Pub. After a year you noticed that you were 4 times as often at the Eastern Pub than at the Western Pub. How is that possible?

Probability Puzzles

Problem: (Unfair Coin) Assume that you have a biased coin, but you do not know the probability of getting a head. How could you get the equivalent of a fair coin with several tosses of the unfair coin?

Probability Puzzles

Problem: (Money in Boxes) Someone shows you two boxes and he tells you that one of these boxes contains two times as much as the other one, but he does not tell you which one. He lets you choose one of these boxes, and opens it. It turns out to have \$10. Now he gives you the opportunity to change your mind and take only the money from the other box.

You reason as follows: *"The second box contains either half of the money of the first box or twice as much as the first box. Thus, it either contains \$20 with chance one half, or it contains only \$5 with chance also one half. Thus the expected value is 1/2 * \$20 + 1/2 * \$5 = \$12.5. Consequently, since the expected value is larger than the value of the first box, I should switch."*

Question: Is your reasoning correct?

Back to Algorithms: Order Statistics

- **Problem:** Given n elements, select the i^{th} smallest element;
    - for $i = 1$ we get the **minimum**;
    - for $i = n$ we get the **maximum**;
    - for $i = \lfloor \frac{n+1}{2} \rfloor$ we get the **median**.
- We can find both the minimum and the maximum in $O(n)$ many steps (linear time).
- Can we find the median also in linear time?
- Clearly, we can do it in time $n \log n$, just MergeSort the array and find the middle element(s) of the sorted array.
- Can we do it faster???

Fast algorithms for finding the median

- We will show that this can be done in linear time, by both a deterministic and by a randomised algorithm.
- Why bother with a randomised algorithm if it can be done in linear time with just a standard, deterministic algorithm?
- Because in practice the randomised algorithm runs much faster, having much smaller constant c in the bound for the run time $T(n) \leq c \cdot n$.
- It turns out that it is easier to solve (both deterministically and with randomisation) the more general problem of finding the i^{th} smallest element for any i than for finding just the median.
- Such “simplifying generalisations” will be used a lot in this course, for example in the Dynamic Programming (DP) technique.

Fast randomised algorithm for finding the median

- **Problem:** Given n elements, select the i^{th} smallest element.
- **Idea:** Divide-and-conquer; "one half" of the randomised QuickSort, operating always on one side of the partition only;

```
RAND-SELECT$(A, p, r, i)$      *choose the $i^{th}$ smallest elt of $A[p..r]$*

1. if $p = r$ & $i = 1$ then return $A[p]$; else
2. choose a random pivot from $A[p..r]$;
3. reorder the array $A[p..r]$ such that $A[p..(q-1)] \leq A[q]$ and
   $A[(q+1)..r] > A[q]$ where $A[q]$ is the randomly chosen pivot;
   * fill the details how we do this in place, just as it is done in an
   implementation of QuickSort *
4. $k \leftarrow q - p + 1$    *$k$ is the number of elements $\leq A[q]$*
5. if $k = i$ then return $A[q]$; else
6. if $i < k$ then return RAND-SELECT$(A, p, q-1, i)$;
7. else return RAND-SELECT$(A, q+1, r, i-k)$.
```

Analysis of $\textsc{Rand-Select}(A, p, r, i)$

- Clearly, the worst case run time is $\Theta(n^2)$;
- this happens, for example, in a very unlikely event that you always pick either the smallest element of the array or the largest element;
- in such a case during each call of Rand-Select the size of the array drops only by 1;
- due to reshuffling of elements around the pivot, each iteration of Rand-Select costs the length of the array, and you get

$$T(n) = c(n + (n-1) + (n-2) + \ldots + 1) = \Theta(n^2)$$

- But this is very unlikely to happen; in fact, most of the time the partitions will be reasonably well balanced!

Analysis of $\textsc{Rand-Select}(A, p, r, i)$

- Let us first assume that all the elements in the array are **distinct**.
- Let us call a partition *a balanced partition* if the ratio between the number of elements in the smaller piece and the number of elements in the larger piece is at least $1 : 9$.
- What is the probability that we get a balanced partition after choosing the pivot?
- Clearly, this happens if we chose an element which is neither among the smallest $1/10$ nor among the largest $1/10$ of all elements;
- Thus, thus, the probability to end up with a balanced partition is $1 - 2/10 = 8/10$.
- Let us find the expected number of partitions between two consecutive balanced partitions.

Analysis of $\textsc{Rand-Select}(A, p, r, i)$

- The probability to get another balanced partition immediately after a balanced partition is $\frac{8}{10}$; probability to need two partitions to get another balanced partition is $\frac{2}{10}\frac{8}{10}$; in general, the probability that you will need k partitions to end up with another balanced partition is $\left(\frac{2}{10}\right)^{k-1} \cdot \frac{8}{10}$.
- Thus, the expected number of partitions between two balanced partitions is

$$
\begin{aligned}
E &= 1 \cdot \frac{8}{10} + 2 \cdot \frac{2}{10}\frac{8}{10} + 3 \cdot \left(\frac{2}{10}\right)^2 \cdot \frac{8}{10} + \ldots \\
&= \frac{8}{10} \cdot \sum_{k=0}^{\infty} (k+1) \left(\frac{2}{10}\right)^k = \frac{8}{10} S
\end{aligned}
$$

where

$$
S = 1 + 2 \cdot \frac{2}{10} + 3 \cdot \left(\frac{2}{10}\right)^2 + 4 \cdot \left(\frac{2}{10}\right)^3 + 5 \cdot \left(\frac{2}{10}\right)^4 + \ldots
$$

Evaluating $S = \sum_{k=0}^{\infty}(k+1)\left(\frac{2}{10}\right)^k$

- How do we evaluate such a sum S??
- Trick # 1:

$$\begin{aligned}
S &= 1 + 2\cdot\frac{2}{10} + 3\cdot\left(\frac{2}{10}\right)^2 + 4\cdot\left(\frac{2}{10}\right)^3 + 5\cdot\left(\frac{2}{10}\right)^4 + \ldots \\
&= 1 + \frac{2}{10} + \left(\frac{2}{10}\right)^2 + \left(\frac{2}{10}\right)^3 + \left(\frac{2}{10}\right)^4 + \ldots \\
&\quad + \frac{2}{10} + \left(\frac{2}{10}\right)^2 + \left(\frac{2}{10}\right)^3 + \left(\frac{2}{10}\right)^4 + \ldots \\
&\quad + \left(\frac{2}{10}\right)^2 + \left(\frac{2}{10}\right)^3 + \left(\frac{2}{10}\right)^4 + \ldots \\
&\quad + \left(\frac{2}{10}\right)^3 + \left(\frac{2}{10}\right)^4 + \ldots
\end{aligned}$$

Evaluating $S = \sum_{k=0}^{\infty}(k+1)\left(\frac{2}{10}\right)^k$

- Summing each row separately we obtain

$$\begin{aligned}
1 + \frac{2}{10} + \left(\frac{2}{10}\right)^2 + \left(\frac{2}{10}\right)^3 + \left(\frac{2}{10}\right)^4 + \ldots &= \frac{1}{1-\frac{2}{10}} = \frac{10}{8} \\
+ \frac{2}{10} + \left(\frac{2}{10}\right)^2 + \left(\frac{2}{10}\right)^3 + \left(\frac{2}{10}\right)^4 + \ldots &= \frac{2}{10}\frac{10}{8} \\
+ \left(\frac{2}{10}\right)^2 + \left(\frac{2}{10}\right)^3 + \left(\frac{2}{10}\right)^4 + \ldots &= \left(\frac{2}{10}\right)^2\frac{10}{8} \\
+ \left(\frac{2}{10}\right)^3 + \left(\frac{2}{10}\right)^4 + \ldots &= \left(\frac{2}{10}\right)^3\frac{10}{8} \\
&\ldots
\end{aligned}$$

- We can now sum the right hand side column;

Evaluating $S = \sum_{k=0}^{\infty}(k+1)\left(\frac{2}{10}\right)^k$

$$\begin{aligned} S &= \frac{10}{8}\left(1 + \frac{2}{10} + \left(\frac{2}{10}\right)^2 + \left(\frac{2}{10}\right)^3 + \left(\frac{2}{10}\right)^4 + \dots\right) \\ &= \frac{10}{8}\frac{1}{1-\frac{2}{10}} = \left(\frac{10}{8}\right)^2 \end{aligned}$$

Thus, we obtain

$$E = \frac{10}{8}S = \frac{10}{8}\left(\frac{10}{8}\right)^2 = \left(\frac{10}{8}\right)^3 = \left(\frac{5}{4}\right)^3 = \frac{125}{64} < 2$$

- This means that on average there will be only one unbalanced partition between two consecutive balanced partitions.
- Consequently the total **average** run time satisfies

$$T(n) < 2n + 2\frac{8}{10}n + 2\left(\frac{8}{10}\right)^2 n + 2\left(\frac{8}{10}\right)^3 n + \dots = \frac{2n}{1-\frac{8}{10}} = 10n$$

A useful digression (Trick #2): $S = \sum_{k=0}^{\infty}(k+1)\left(\frac{2}{10}\right)^k$ evaluated another way.

$$\begin{aligned}
S &= 1 + 2\cdot\frac{2}{10} + 3\cdot\left(\frac{2}{10}\right)^2 + 4\cdot\left(\frac{2}{10}\right)^3 + 5\cdot\left(\frac{2}{10}\right)^4 + \dots \\
&= 1 + \frac{2}{10} + \left(\frac{2}{10}\right)^2 + \left(\frac{2}{10}\right)^3 + \left(\frac{2}{10}\right)^4 + \dots \\
&\qquad + \frac{2}{10} + 2\left(\frac{2}{10}\right)^2 + 3\left(\frac{2}{10}\right)^3 + 4\left(\frac{2}{10}\right)^4 + \dots \\
&= \frac{1}{1-\frac{2}{10}} + \frac{2}{10}\left(\underbrace{1 + 2\cdot\frac{2}{10} + 3\cdot\left(\frac{2}{10}\right)^2 + 4\cdot\left(\frac{2}{10}\right)^3 + 5\cdot\left(\frac{2}{10}\right)^4 + \dots}_{S}\right)
\end{aligned}$$

Thus we obtain $S = \frac{10}{8} + \frac{2}{10}S$, which yields $S = \left(\frac{10}{8}\right)^2$.

Performance of RAND-SELECT:

- Where did we tacitly assume that all elements are distinct?
- What is the probability of choosing the pivot so that the partition is balanced?
- Note that if all elements are the same RAND-SELECT would run in quadratic time no matter which elements are chosen as pivots - they are all equal.
- **Homework:** Modify RAND-SELECT so that it runs in linear time even when there are many repetitions. You might want try to modify step 3 of RAND-SELECT by slightly changing the way how the array is reordered.

- In 1972 Blum, Floyd, Pratt, Rivest and Tarjan designed a deterministic Order Statistic Selection which runs in linear time in the worst case;
- Next time: A deterministic Order Statistic Selection algorithm.