Advanced algorithms
Randomized Algorithms Notes

Søren Dahlgaard

June 15, 2011

1 Disposition

1. Motivation - Quicksort example
2. Average-case analysis
3. Randomized algorithms. Ensuring average-case
4. Indicator random variables.
   (a) $E[X_H] = \Pr\{H\}$.
   (b) Linearity of expectation
5. Shuffling arrays to randomize things
   (a) Slow shuffle - easy proof
   (b) Knuth shuffle. Loop invariant: $(n - i + 1)!/n!$
6. Examples
   (a) Quicksort - Amount of comparisons. Z_{ij}.

2 Summary

Usually when analyzing algorithms we are only interested in the worst-case time complexity. However in some cases we might want info about the average-case. For instance we know that all splits on the form $(\alpha, 1-\alpha)$ of quicksort result in logarithmic time. What is the average-case?

2.1 Probabilistic analysis

When analyzing the average-case we want to make some assumptions about the distribution of input. For instance assume that each ordering of candidates is equally likely in the hiring problem (see below).

2.2 Randomized algorithms

In order to make assumptions about the distribution we might sometimes have to randomize the input. For instance a very common case for sorting is already sorted sequences or nearly sorted sequences. In these cases quicksort performs bad, so what we do is that we randomize the algorithm such that no specific input will ellicit worst-case behaviour.

Company vs company analogy.

2.3 Indicator random variables

In our probabilistic analysis we use variables known as indicator random variables. For an event A we define

$$I\{A\} = \begin{cases} 1 & \text{if } A \text{ occurs} \\ 0 & \text{if } A \text{ doesn't occur} \end{cases}$$

We also talk about the expected value of a such variable. If X_H is the indicator variable that a coin flip comes up heads $I\{H\}$ then we have $\Pr\{H\} = 1/2$, thus the expected value

$$E[X_H] = 1 \cdot \Pr\{H\} + 0 \cdot \Pr\{\overline{H}\} = 1/2$$

In general we have $E[X_A] = \Pr\{A\}$ (by definition basically).

If we want to compute how many heads we expect from a series of n coin flips we could use the variables $X_I = I\{\text{the } i\text{th flip is heads}\}$. We can then sum all these n variables together:

$$X = \sum_{i=1}^{n} X_i \Leftrightarrow E[X] = E\left[\sum_{i=1}^{n} X_i\right] = \sum_{i=1}^{n} E[X_i]$$

because of linearity of expectation. This is just $n/2$.

2.4 Shuffling arrays

The book presents two ways of doing this. One is to assign each number a priority $\textsc{Random}(1, n^3)$ and sort using this priority. If we disregard that some indices may get the same priority we can prove this by letting E_i be the event that $A[i]$ gets the ith smallest priority. Then the probability that all E_i occurs are:

$$\Pr\{E_1 \cap E_2 \cap \ldots \cap E_n\}$$

This is the same as

$$\Pr\{E_1\} \cdot \Pr\{E_2 | E_1\} \cdot \Pr\{E_3 | E_2 \cap E_1\} \ldots$$

That is. The chance that E_2 happens IF E_1 has already happened. Thus The chance that $A[2]$ is the lowest of the remaining $n - 1$ numbers. This is just

$$1/n \cdot 1/(n-1) \cdot \ldots \cdot 1/2 \cdot 1 = 1/n!$$

Thus proving the shuffling.

The other way of shuffling is the Knuth Shuffle. We go through the array and at each point we pick $A[i]$ at random from the array $A[i..n]$. To prove this we use the following invariant:

Prior to the ith iteration each possible $(i-1)$-permutation has probability $(n-i+1)!/n!$ of being the one in the array $A[1..i-1]$.

Initialization: The array $A[1..0]$ is empty and has probability $(n-i+1)!/n! = n!/n! = 1$ of being the only 0-permutation.

Maintenance: Consider any i-permutation. If we know that each $i-1$-permutation has probability $(n-i+1)!/n!$ of being in $A[1..i-1]$ and let's call this event E_1 for the specific i-permutation's first $i-1$ numbers. We then have probability $n-i+1$ of picking the last element of the permutation to position $A[i]$. Thus we get:

$$\Pr\{E_2 \cap E_1\} = \Pr\{E_2|E_1\}\Pr\{E_1\}$$

is equal to

$$\frac{1}{n-i+1}\frac{(n-i+1)!}{n!} = \frac{(n-i)!}{n!}$$

Termination: At termination $i = n+1$ thus

$$\frac{(n-(n+1)+1}{n!} = \frac{0!}{n!} = \frac{1}{n!}$$

2.5 Examples

2.5.1 Hiring problem

We want to hire some people for a project. Each person has a "quality", so we know which are best. It costs c_h to hire people and the tactic we use is to always hire a candidate if he is better than the previously hired one. So if the candidates arrive in order $(4,1,5,3,8,2,7,6)$ we will hire $(4,5,8)$.

Let $X_i = I\{\text{candidate } i \text{ is hired}\}$. Then we have $X = X_1 + \ldots + X_n$ where X is the number of candidates hired. We have that each candidate in $1..i$ is equally likely to be the best of the first i candidates. Thus we get:

$$\begin{aligned} E[X] &= E\left[\sum_{i=1}^{n} X_i\right] && (1)\\ &= \sum_{i=1}^{n} E[X_i] && (2)\\ &= \sum_{i=1}^{n} 1/i && (3)\\ &= O(\ln n) && (4)\end{aligned}$$

Randomization: We cannot be sure about the distribution of the input thus we want to randomize it. Let's say the algorithm gets an array A of candidates, then we can start by shuffling this array.

2.5.2 Quicksort

We pick the pivot at random from the $r - p + 1$ elements.

The analysis looks at how many comparisons we do. Let X_{ij} be the indicator random variable for whether i and j are compared. (They will only be compared zero or one times because one of them must be the pivot). Let Z_{ij} be the sorted array $A'[i..j]$. Then i and j will only be compared if one of them is the first pivot chosen in Z_{ij}. Thus we have:

$$E[X_{ij}] = \frac{2}{j - i + 1}$$

and thus:

$$\begin{aligned}
E[X] &= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{2}{j - i + 1} && (5)\\
&= \sum_{i=1}^{n-1} \sum_{k=1}^{n-i} \frac{2}{k + 1} && (6)\\
&< \sum_{i=1}^{n-1} \sum_{k=1}^{n} \frac{2}{k} && (7)\\
&= O(n \lg n) && (8)
\end{aligned}$$

2.5.3 Selection

We use randomized-partition and do a quicksort except we only sort half the array.. basically.

Let X_k be the indicator random variable that the array $A[p..q]$ has exactly k elements. Because each pivot is equally likely to be chosen we have $E[X_k] = 1/n$. We upper bound the selection by assuming that i always falls in the largest of the two arrays $A[1..k-1]$ and $A[k+1..n]$. Thus we have:

$$\begin{aligned}
T(n) &\leq \sum_{k=1}^{n} X_k \cdot (T(\max(k-1, n-k)) + O(n)) && (9)\\
&= \sum_{k=1}^{n} X_k \cdot T(\max(k-1, n-k)) + O(n) && (10)
\end{aligned}$$

Taking expectations:

$$\begin{aligned}
E[T(n)] &= \sum_{k=1}^{n} E[X_k \cdot T(\max(k-1, n-k))] + O(n) && (11)\\
&= \sum_{k=1}^{n} E[X_k] \cdot E[T(\max(k-1, n-k))] + O(n) && (12)\\
& && (13)
\end{aligned}$$

This is because X_k and $T(\max(k-1, n-k))$ are independant. We also have $E[X_k] = 1/n$. Because we take $\max(k-1, n-k)$ we have that each term from $\lfloor n/2 \rfloor..n-1$ appears twice:

$$E[T(n)] \leq \frac{2}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} E[T(k)] + O(n)$$

This can be show n by substitution to be $O(n)$. Not really relevant for this course. It's just "simple" math.

2.5.4 Binary Search Trees

The proof that a randomly build BST has expected height $O(\lg n)$ goes as follows:

1. We have a binary tree built on n keys. Let X_n be the height of this tree.
2. Let Y_n be the exponential height, $Y_n = 2^{X_n}$.
3. Let R_n be the rank of the root. If we built the tree on the set $\{1, 2, \ldots, n\}$ then R_n is equally likely to be any of those.
4. Let $Z_{n,i}$ be the indicator random variable that $\Pr\{R_n = i\}$. We thus have $Z_{n,i} = 1/n$.
5. We have $Y_n = 2 \cdot \max(Y_{i-1}, Y_{n-i})$. Or Y_n is two times the exponential height of its biggest subtree.
6. Y_{n-i} and Y_{i-1} are independant of $Z_{n,i}$ except that it defines how many elements there are in this tree. Y_{i-1} is just like any other tree with $i-1$ nodes.
7. This all results in:

   $$Y_n = \sum_{i=1}^{n} Z_{n,i}(2 \cdot \max(Y_{i-1}, Y_{n-i}))$$

   Taking expectations and using linearity and independence we get:

   $$\begin{aligned} E[Y_n] &= \sum_{i=1}^{n} E[Z_{n,i}] E[2 \cdot \max(Y_{i-1}, Y_{n-i})] && (14) \\ &= \frac{2}{n} \sum_{i=1}^{n} E[\max(Y_{i-1}, Y_{n-i})] && (15) \\ &\leq \frac{2}{n} \sum_{i=1}^{n} (E[Y_{i-1}] + E[Y_{n-i}]) && (16) \\ &= \frac{4}{n} \sum_{i=0}^{n-1} E[Y_i] && (17) \end{aligned}$$

   Line two we move the 2 out of the second $E[..]$. Line three follows from some excercise. Line three is because each term appears twice.

If we use the substitution method with the guess:

$$E[Y_n] \leq \frac{1}{4}\binom{n+3}{3}$$

and the identity:

$$\sum_{i=0}^{n-1}\binom{i+3}{3} = \binom{n+3}{4}$$

We can see that Y_n is polynomial in n and thus $E[X_n]$ is $O(\lg n)$.

2.5.5 3-CNF

We randomly set each literal to 1 with probability 1/2 and 0 with same probability. This version of 3-CNF requires exactly 3 literals per clause. No literal and its negation in same clause. No literal more than once per clause.

Let $Y_i = I\{\text{clause } i \text{ is satisfied}\}$

A clause is only satisfied when all its literals are set to 0. This means $E[Y_i] = 7/8$. Summing all Y_is up gives $7m/8$ and thus a randomized 8/7-approximation.