# CS60003 Algorithm Design and Analysis, Autumn 2010–11

## Class test 1

Maximum marks: 40 | Time: 07–08–09–10 | Duration: $1 + \epsilon$ hour

---

**Roll no:** ____________ **Name:** ______________________________

[ *Write your answers in the question paper itself. Be brief and precise. Answer all questions.* ]

In both the following exercises, you solve the same computational problem which goes like this. You are given $n$ probabilities $p_1, p_2, \ldots, p_n$ (so each $p_i \in [0,1]$). You are also given an integer $k$ in the range $0 \leqslant k \leqslant n$ (so $k = O(n)$). Assume that $p_i$ is the probability of obtaining a head in a random toss of a coin $C_i$. One toss is made of each of the coins $C_1, C_2, \ldots, C_n$ in that order. Your task is to propose efficient algorithms to compute the probability $P(n,k)$ of obtaining exactly $k$ heads in these $n$ tosses. Of course, in addition to $n$ and $k$, the value of $P(n,k)$ depends also on the probabilities $p_1, p_2, \ldots, p_n$. For simplicity, we use the simplified notation $P(n,k)$ to actually stand for $P(n,k,p_1,p_2,\ldots,p_n)$.

**Example:** Let $n = 3$, $p_1 = \frac{1}{3}$, $p_2 = \frac{1}{2}$, $p_3 = \frac{3}{4}$, and $k = 2$. Denote a head by $H$ and a tail by $T$. All possible outcomes of three tosses with exactly two heads are $HHT$, $HTH$ and $THH$. The probability to be calculated is, therefore, $P(3,2) = \frac{1}{3}\times\frac{1}{2}\times(1-\frac{3}{4})+\frac{1}{3}\times(1-\frac{1}{2})\times\frac{3}{4}+(1-\frac{1}{3})\times\frac{1}{2}\times\frac{3}{4} = \frac{1}{24}+\frac{3}{24}+\frac{6}{24} = \frac{10}{24} = \frac{5}{12}$.

Suppose that we do arithmetic on floating-point numbers of a fixed size (like `double` in C), so the cost of adding, subtracting or multiplying two floating-point values is always $\Theta(1)$. In particular, the probabilities $p_1, p_2, \ldots, p_n$ are supplied as floating-point values (not as rational numbers as in the above example).

If $k = n/2$, there are $\binom{n}{n/2} \geqslant 2^{n/2} = (\sqrt{2})^n$ outcomes with exactly $k$ heads. Enumerating all possibilities leads to fully exponential running time. Better algorithms are needed to achieve polynomial running times.

**1.** First, design an $O(n^2)$-time dynamic-programming algorithm to compute $P(n,k)$. Use the values $P(i,j)$ to stand for the probability of obtaining exactly $j$ heads in the tosses of $C_1, C_2, \ldots, C_i$.

**(a)** For $i \geqslant 1$, express $P(i,j)$ in terms of $P(i-1,j-1)$ and $P(i-1,j)$. Give brief justification. **(5)**

*Solution* There are two possibilities in the $i$-th toss: $H$ comes with probability $p_i$, and $T$ with probability $1-p_i$. In the first case, we require exactly $j-1$ heads in tosses $1$ through $i-1$, whereas in the second case, we require exactly $j$ heads in tosses $1$ through $i-1$. Therefore,

$$P(i,j) = \begin{cases} p_i P(i-1,j-1) + (1-p_i)P(i-1,j) & \text{if } j \geqslant 1, \\ (1-p_i)P(i-1,j) & \text{if } j = 0. \end{cases}$$

**(b)** Supply conditions to terminate the recursive definition of $P(i,j)$. Give brief justification. **(5)**

*Solution* Basis case $i = 0$: We have $P(0,j) = \begin{cases} 1 & \text{if } j = 0, \\ 0 & \text{otherwise.} \end{cases}$ If you toss zero coins, the only possible outcome for the number of heads is zero. Since $j \leqslant i$, you may only supply the condition $P(0,0) = 1$.

You may also choose to start the definition from $i = 1$, and use the inductive formula of Part 1(a) for $i \geqslant 2$. If so, we have $P(1,j) = \begin{cases} 1-p_1 & \text{if } j = 0, \\ p_1 & \text{if } j = 1. \end{cases}$ Here again, we assumed that $j \leqslant i$.

Whatever your basis case is, you should also supply another terminating condition: $P(i,j) = 0$ for $i < j$.

**(c)** Convert the above formulas to an $O(n^2)$-time dynamic-programming algorithm to calculate $P(n, k)$. **(5)**

*Solution* In the following algorithm, we use a two-dimensional array $P[i, j]$ with $i$ ranging from $0$ to $n$, and $j$ ranging from $0$ to $k$.

Set $P[0, 0] = 1$.
For $j = 1, 2, \ldots, k$, set $P[0, j] = 0$.
For $i = 1, 2, \ldots, n$ {
    Set $P[i, 0] = (1 - p_i) \times P[i - 1, 0]$.
    For $j = 1, 2, \ldots, k$ {
        Compute $P[i, j] = p_i \times P[i - 1, j - 1] + (1 - p_i) \times P[i - 1, j]$.
    }
}
Return $P[n, k]$.

**(d)** Justify that your algorithm runs in $O(n^2)$ time. **(5)**

*Solution* Each element of the $(n + 1) \times (k + 1)$ matrix $P$ can be computed in $O(1)$ time. So the running time is $O(nk)$. Since $k = O(n)$, this running time is $O(n^2)$.

**2.** Now, design an $O(n \log^2 n)$-time divide-and-conquer (top-down) algorithm for computing $P(n, k)$. Denote by $Q(i, j, k)$ the probability of obtaining exactly $k$ heads in tosses of $C_i, C_{i+1}, \ldots, C_j$. (We have $P(n, k) = Q(1, n, k)$.) Also, denote by $F_{i,j}(x)$ the polynomial

$$F_{i,j}(x) = \sum_{k \geqslant 0} Q(i, j, k)x^k.$$

$Q(i, j, k) = 0$ for $k > j - i + 1$, so $F_{i,j}(x)$ is indeed a polynomial (not a non-terminating power series). If we can compute the polynomial $F_{i,j}(x)$, we output its coefficient of $x^k$ as $Q(i, j, k)$. In particular, $P(n, k)$ is the coefficient of $x^k$ in $F_{1,n}(x)$. So it suffices to compute $F_{1,n}(x)$.

**(a)** Basis case: Let $i \in \{1, 2, \ldots, n\}$. Write the expression for $F_{i,i}(x)$ with justification. **(5)**

*Solution* We have $Q(i, i, 0) =$ probability of $T$ in the $i$-th toss $= 1 - p_i$, $Q(i, i, 1) =$ probability of $H$ in the $i$-th toss $= p_i$, and $Q(i, i, k) =$ probability of $k \geqslant 2$ heads in the $i$-th toss $= 0$. Therefore, $F_{i,i}(x) = (1 - p_i) + p_i x$.

**(b)** Induction: Let $1 \leqslant i \leqslant m < j \leqslant n$. Prove that $F_{i,j}(x) = F_{i,m}(x)F_{m+1,j}(x)$. **(5)**

*Solution* The coefficient of $x^k$ in $F_{i,j}(x)$ is the probability of obtaining exactly $k$ heads in tosses $i$ through $j$, that is, $Q(i, j, k)$. This probability can be calculated in another way. Let $k_1$ be the number of heads in tosses $i$ through $m$, and $k_2$ the number of heads in tosses $m + 1$ through $j$. The probability of this event for a choice of the pair $(k_1, k_2)$ is $Q(i, m, k_1) \times Q(m + 1, j, k_2)$. Summing over all pairs $(k_1, k_2)$ with $k_1 + k_2 = k$ gives

$$Q(i, j, k) = \sum_{k_1+k_2=k} Q(i, m, k_1) \times Q(m + 1, j, k_2).$$

But the right side of this equality is the coefficient of $x^k$ in the polynomial product $F_{i,m}(x)F_{m+1,j}(x)$.

**(c)** Propose an $O(n \log^2 n)$-time divide-and-conquer algorithm to compute $F_{1,n}(x)$. **(5)**

*Solution* The pseudocode for computing $F_{i,j}(x)$ follows. The outermost call should be made with $i = 1$ and $j = n$.

If $i = j$, return $F_{i,i}(x) = (1 - p_i) + p_i x$,
else {
    Compute the middle index $m = \lfloor (i + j)/2 \rfloor$.
    Recursively compute $F_{i,m}(x)$.
    Recursively compute $F_{m+1,j}(x)$.
    Return the product $F_{i,m}(x) F_{m+1,j}(x)$ (use FFT-based polynomial multiplication).
}

**(d)** Prove that your algorithm in Part 2(c) runs in $O(n \log^2 n)$ time. (You may use without proof the result that the solution of the recurrence $T(n) = 2T(n/2) + O(n \log n)$ is $T(n) = O(n \log^2 n)$. For a proof, look at the solution of Exercise 1 in the Mid-Semester Test of Autumn 2008.) **(5)**

*Solution* The *divide* step takes only $O(1)$ time. The *combine* step can be completed in $O(n \log n)$ time using FFT-based polynomial multiplication, since all polynomials $F_{i,j}(x)$ involved in the computation are of degrees $\leqslant j - i + 1 \leqslant n$.

(**Remark:** Never ever underestimate the power of top-down programming.)

[You may also use this space for continuation of answers. Give pointers from Pages 1–4.]

[You may also use this space for continuation of answers. Give pointers from Pages 1–4.]