Randomised Algorithms

Winter term 2022/2023, Exercise Sheet No. 7

Hand-out: Mon, 28. Nov.
Hand-in: Sun, 4. Dec.

UNIVERSITY OF PASSAU
Faculty of Computer Science and Mathematics

Dr. Duc-Cuong Dang
Prof. Dr. Dirk Sudholt

Exercise 1 [6 points]

In classic roulette there are 37 numbered pockets: 18 red pockets, 18 black pockets and one uncoloured pocket for the bank. In each round, a ball is thrown into the rotating roulette and will land on one of the pockets uniformly at random. If a player bets an amount on a colour and the ball lands in a pocket of such colour, the bet is doubled; otherwise it falls to the bank. We assume that only whole euro amounts are bet and that the casino has unlimited capital. Consider a player with initial capital X_0 euros bets half of his/her capital (rounded up) in each round.

(a) Use drift analysis to find an upper bound on the expected number of rounds until the player goes broke regardless of his/her strategy to choose the colour.

(b) Let $X_0 = 10^6$, use the multiplicative drift theorem with tail bounds to estimate the probability that the player is *not yet* broke after 2000 rounds.

Exercise 2 [6 points]

Consider a collection of cards that can be totally ordered, eg. in some game Aces are preferred over Kings and so on, then Diamonds over Hearts, etc. Suppose we are given an initial deck S_0 of n cards in a random order and we want to put them in a preferred order. We would like analyse the following algorithm: at step t, select a pair of cards, denoted (a_i, a_j) with $i < j$, uniformly at random from the current deck $S_t = (a_1, \ldots, a_n)$. If their order is wrong then exchange their places, otherwise do nothing, then we go to the next step $t + 1$.

(a) Let $\mathrm{INV}(S_t)$ be the number of pairs of S_t in the wrong order. How large can $\mathrm{INV}(S_0)$ be in the worst case?

(b) In the event that an exchange (a_i, a_j) occurs at step t, prove that

$$\mathrm{INV}(S_t) - \mathrm{INV}(S_{t+1}) \geq 1.$$

This essentially means aside from correcting the order between a_i and a_j, which corresponds to the value 1 above, the number of other wrong-order pairs $(a_{i'}, a_{j'})$ with $i' \neq i$ or $j' \neq j$ does not increase.

Hints: Since $i < j$ and a_i and a_j are in the wrong order, in the current deck a_i comes before a_j, but in the preferred order a_j comes before a_i. Examine the effect of exchanging a_i and a_j on pairs of cards involving precisely one other card of the deck, that is, pairs with one element from $\{a_i, a_j\}$ and one element a_k from $R := S_t \setminus \{a_i, a_j\}$. You may also want to partition R into subsets according to whether *in the current deck* a_k is placed before a_i, after a_j or between a_i and a_j. For the latter case you might need to distinguish further cases according to how a_k relates to a_i and a_j *in the preferred order*.

(c) Estimate the probability of an exchange occurring and then use drift analysis to give an asymptotic upper bound on the expected number of steps until the algorithm has sorted the deck.

Exercise 3 [4 points]

Consider the following (naive) algorithm that creates a uniform permutation of n objects from array A in another array B.

NaiveRandomPermutation(A, B)

1: **for** $i = 1, \ldots, n$ **do**
2: $\quad B[i] \coloneqq$ `null`.
3: **for** $i = 1, \ldots, n$ **do**
4: $\quad$ **repeat**
5: $\quad\quad$ Choose j uniformly at random from $\{1, 2, \ldots, n\}$.
6: $\quad$ **until** $B[j] =$ `null`
7: $\quad B[j] \coloneqq A[i]$.

Analyse the expected running time of the algorithm NaiveRandomPermutation.

Exercise 4 [4 points]

The following algorithm computes a uniform permutation *in situ* (ie. with only $O(1)$ extra space) in time $O(n)$:

FastRandomPermutation(A)

1: **for** $i = 1, \ldots, n$ **do**
2: Choose j uniformly at random from $\{i, i+1, \ldots, n\}$.
3: Swap $A[i]$ and $A[j]$.

Show that the algorithm is correct, ie. each permutation is output with the same probability $1/(n!)$.