

Seminar 4

Complexitatea medie.

Ștefan Ciobâcă, Dorel Lucanu
Universitatea “Alexandru Ioan Cuza”, Iași

Săptămâna 7 Martie - 11 Martie 2016

Hint: multe formule utile (e.g. pentru serii, integrale, combinări, aranjamente, limite, probabilități) în informatica teoretică sunt disponibile aici: <https://www.tug.org/texshowcase/cheat.pdf>.

1. Fie următorul algoritm probabilist (care nu primește nimic la intrare):

```
ok = 1;
while (ok) {
    if (random(2) == 0) {
        ok = 0;
    }
}
return 0;
```

Care este probabilitatea ca algoritmul să se oprească? Hint: calculați probabilitatea ca algoritmul să se oprească după exact i iterații ale buclei **while**, pentru fiecare $i \in \{1, 2, 3, \dots\}$.

2. Fie următorul algoritm probabilist, care primește la intrare un număr natural n și întoarce un număr natural:

```
sum = 0;
for (i = 0; i < n; ++i) {
    sum = sum + random(2);
}
return sum;
```

Care este media valorilor întoarse de algoritm pentru un input $n \in \mathbb{N}$ arbitrar?

3. Fie o funcție probabilistă **rand2** care întoarce cu probabilitate 0.5 valoarea 0 și cu probabilitate 0.5 valoarea 1.

Scrieți o funcție probabilistă **zar** care nu primește niciun argument și întoarce un număr natural între 0 și 5, fiecare cu aceeași probabilitate. Puteți folosi în funcția **zar** funcția **rand2** dar nu aveți voie să folosiți altă funcție probabilistă (cum ar fi **random**).

4. Fie o funcție probabilistă **rand2p** care nu primește niciun argument și întoarce 0 cu probabilitate p și 1 cu probabilitate $1 - p$. Numărul p este un număr real $p \in (0, 1)$, dar valoarea lui nu este cunoscută.

Scrieți o funcție **rand2corect** care nu primește niciun argument și întoarce 0 cu probabilitate 0.5 și 1 cu probabilitate 0.5. În implementarea funcției **rand2corect** puteți folosi funcția **rand2p**, dar nicio altă funcție probabilistă.

5. Folosindu-vă în continuare (doar) de funcția probabilistă **rand2**, scrieți funcția **random**, care primește ca argument un număr natural n și întoarce un număr natural din mulțimea $\{0, 1, \dots, n-1\}$, fiecare cu probabilitatea $1/n$.
6. Fie problemele **SSD1**, **SSD2**, **SSD3** din cursul 3 (slide 46). Scrieți câte un algoritm nedeterminist care rezolvă fiecare din aceste probleme.
7. O formulă **3-CNF** din logica propozițională (e.g. $(x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_5 \vee \neg x_2) \wedge (x_2 \vee x_2 \vee \neg x_2) \wedge (x_3 \vee x_1 \vee \neg x_2)$) poate fi reprezentată printr-o matrice v (pentru formula, precedentă, $v[n][3] = \{\{1, -2, 3\}, \{-1, 5, -2\}, \{2, 2, -2\}, \{3, 1, -2\}\}$).
 - (a) Problema 3-satisfiabilității este următoarea:
Input: O matrice $v[n][3]$ care reprezintă o formulă **3-CNF** ca mai sus și n – numărul de clauze.
Output: *da*, dacă formula este satisfiabilă; *nu*, altfel.
 Scrieți în Alk un algoritm nedeterminist care rezolvă problema 3-satisfiabilității.
 - (b) Descrieți ca pereche input-output problema validității unei formule **3-DNF**. Puteți scrie un algoritm nedeterminist pentru problema validității?
 - (c) Problema 3-non-echivalenței este următoarea:
Input: Două matrici $v[n][3]$ și $w[m][3]$ care reprezintă două formule **3-CNF** ca mai sus (n este numărul de clauze din prima formulă și m numărul de clauze din cea de-a doua formulă).
Output: *nu*, dacă cele două formule sunt echivalente (au aceeași valoare de adevăr pentru orice asignare); *da*, altfel.
 Scrieți în Alk un algoritm nedeterminist care rezolvă problema 3-non-echivalenței. Puteți scrie un algoritm nedeterminist pentru problema 3-echivalenței?
 - (d) Scrieți un algoritm *determinist* pentru problema 3-satisfiabilității. Ce complexitate-timp are algoritmul în cazul cel mai nefavorabil?
8. Fie următorul algoritm probabilist, care primește la intrare un număr natural n și un vector v de dimensiune n care conține o permutare a numerelor de la 0 la $n-1$:

```

for (i = 0; i < n; ++i) {
    if (v[i] % 2 == 0) {
        break;
    }
}
return 0;

```

Presupunând că fiecare permutare este echiprobabilă, care este timpul mediu de execuție al algoritmului pentru date de intrare de dimensiune n ?

9. Fie următorul algoritm probabilist, care primește la intrare un număr natural n și un vector v de dimensiune n care conține o permutare a numerelor de la 1 la n :

```

max = v[0];
sum = 0;
for (i = 0; i < n; ++i) {
    if (v[i] > max) {
        max = v[i];
        for (j = 0; j < n; ++j) { // bucla for interioara
            sum = sum + 1;
        }
    }
}

```

```

    }
}
return sum;

```

Presupunând că fiecare permutare este echiprobabilă, care este timpul mediu de execuție al algoritmului pentru date de intrare de dimensiune n ? Hint: calculați numărul mediu de execuții ale buclei `for` interioare.

10. Fie următorul algoritm probabilist, care primește la intrare un număr natural n și un vector v de dimensiune n care conține o permutare a numerelor de la 1 la n :

```

max = v[0];
sum = 0;
for (i = 0; i < n - 1; ++i) {
    for (j = i + 1; j < n; ++j) {
        if (v[i] > v[j]) {
            for (j = 0; j < n; ++j) { // bucla for interioara
                sum = sum + 1;
            }
        }
    }
}
return sum;

```

Presupunând că fiecare permutare este echiprobabilă, care este timpul mediu de execuție al algoritmului pentru date de intrare de dimensiune n ? Hint: calculați numărul mediu de execuții ale buclei `for` interioare.

11. Rezolvați cele 3 probleme precedente pentru cazul în care vectorul v conține pe poziția i un număr natural între 0 și $n - 1$, fiecare număr cu aceeași probabilitate, iar fiecare astfel de vector este echiprobabil (deci v nu mai conține neapărat o permutare).
12. În acest exercițiu vom proiecta un algoritm probabilist care primește la intrare un număr n și produce la ieșire o permutare de ordin n , fiecare permutare fiind echiprobabilă.

- (a) Arătați că algoritmul următor nu este corect:

```

for (i = 0; i < n; ++i) {
    p[i] = i;
}
for (i = 0; i < n; ++i) {
    j = random(n);
    temp = p[i];
    p[i] = p[j];
    p[j] = temp;
}

```

în sensul în care nu generează fiecare permutare cu aceeași probabilitate. Hint: considerați toate rezultatele posibile pentru $n = 3$ și calculați probabilitatea fiecăruia.

- (b) Fie următorul algoritm (Fisher-Yates):

```

for (i = 0; i < n; ++i) {
    used[i] = 0;
}

```

```

notused = n;
for (i = 0; i < n; ++i) {
    int k = random(notused);
    int j = 0;
    while (j < n) {
        if (used[j]) {
            k = k - 1;
            if (k == 0) {
                break;
            }
        }
        j = j + 1;
    }
    p[i] = j;
    used[j] = 1;
    notused = notused - 1;
}

```

Arătați că algoritmul întoarce fiecare permutare de ordin n cu aceeași probabilitate. Care este complexitatea algoritmului?

(c) Fie următorul algoritm (Fisher-Yates optimizat):

```

for (i = 0; i < n; ++i) {
    p[i] = i;
}
for (i = 0; i < n; ++i) {
    j = random(i + 1);
    temp = p[i];
    p[i] = p[j];
    p[j] = temp;
}

```

Arătați că, după a j -a iterație a algoritmului (pentru $j \in \{1, \dots, n\}$), vectorul p conține pe primele j poziții (adică pe pozițiile $0, \dots, j - 1$) fiecare permutare a numerelor $0, \dots, j - 1$ cu aceeași probabilitate. Care este complexitatea algoritmului?

13. Fie următorul algoritm probabilist pentru alegerea minimului dintr-un tablou:

```

p = fisher-yates(n);
min = a[p[0]];
sum = 0;
for (i = 1; i < n; ++i) {
    if (a[p[i]] < min) {
        min = a[p[i]];
        for (j = 0; j < n; ++j) { // bucla for interioara
            sum = sum + 1;
        }
    }
}
return 0;

```

Care este timpul mediu de execuție al algoritmului de mai sus, când tabloul a de la intrare conține n elemente distincte? Hint: de câte ori se execută bucla `for` interioară?