Universitatea Alexandru Ioan Cuza, Iaşi
Facultatea de Informatică
Proiectarea Algoritmilor - Test Scris (5 iunie 2020)
Varianta de test pentru studentul STAMATE D. VALENTIN

Redactați rezolvarea pe maxim 4 pagini A4, conform procedurii de examinare online prezentată pe site-ul cursului. După ce terminați, scanați paginile într-un singur fișier .PDF de dimensiune de maxim 4MB și încărcați fișierul la adresa `https://docs.google.com/forms/d/e/1FAIpQLSdfKHhmwOVqFhzCbqRSeI1YVxikdoABTId3dYv8ooW8q3srrA/viewform?usp=sf_link` (trebuie să vă conectați folosind un cont Google). Pe 5 iunie 2020, la ora 10h30, formularul va fi închis și nu veți putea încărca rezolvări după această oră.

Test scris

Completați de mână pe foaia de examen următoarele informații.

Nume: .....

Grupa: .....

Declarație de onestitate

Subsemnatul, ..., student al Facultății de Informatică, Universitatea Alexandru Ioan Cuza, declar că rezolvarea care urmează îmi aparține în integralitate și că cunosc regulamentul universității referitor la sancțiunile posibile (inclusiv nepromovarea disciplinei, exmatriculare) aplicate în caz de fraudă sau tentativă de fraudă.

Data: ..... **Semnătură .....**

Rezolvați următoarele exerciții. Nu copiați cerința, dar scrieți numărul itemului la care răspundeți.

Q1 Analiza și proiectarea algoritmilor - elemente de bază.
Se consideră următorul algoritm, care primește la intrare un număr natural n și un tablou multi-dimensional M:

```
s = 0;
for (i = 0; i < n; i = i + 1)
  for (j = 0; j * j < n; j = j + 1)
    for (k = 0; k * k < n; k = k + 1)
      for (x = 0; x * x < n; x = x + 2)
        for (y = 0; y < n; y = y + 2) {
          if (M[i][j][k][x][y] > 10) { // atentie la detalii
            s = 1;                     // in for-urile de mai sus
            break;
          }
        }
return s;
```

Datele de intrare se presupun corecte.

(a) (4p) Ce problemă computațională rezolvă algoritmul de mai sus?
(b) (6p) Să se scrie un algoritm nedeterminist care să rezolve aceeași problemă. Algoritmul nedeterminist nu va conține instrucțiuni repetitive și nici funcții recursive.
(c) (6p) Să se calculeze complexitatea în cazul cel mai nefavorabil.
  (1p) *Dimensiunea unei instanțe.*
  (1p) *Operații numărate.*
  (2p) *Cazul cel mai nefavorabil.*
  (2p) *Timpul pentru cazul cel mai nefavorabil.*
(d) (4p) Să se calculeze complexitatea în cazul cel mai favorabil.
  (2p) *Cazul cel mai favorabil.*
  (2p) *Timpul pentru cazul cel mai favorabil.*

Q2 Algoritmi probabiliști.
Se consideră următorul algoritm probabilist, care nu primește la intrare nicio valoare:

```
f()
{
    k = random(6); // intoarce un numar aleatoriu uniform distribuit din multimea
    switch (k) {  // { 0, 1, 2, 3, 4, 5 }
```

```
    case 0: return 0;   // instructiunea switch functioneaza ca in limbajul C
    case 1: return 2;
    case 2: return 2;
    case 3: return f();
    case 4: return 1;
    case 5: return f();
    }
}
```

(a) (6p) Care e probabilitatea ca algoritmul să se oprească după exact i apeluri ale funcției $f()$, unde $i \in \mathbb{N}$?
(b) (5p) Care e probabilitatea ca algoritmul să ruleze la infinit?
(c) (3p) Care e probabilitatea ca algoritmul să întoarcă valoarea 0?
(d) (3p) Care e probabilitatea ca algoritmul să întoarcă valoarea 1?
(e) (3p) Care e probabilitatea ca algoritmul să întoarcă valoarea 2?

Q3 Programare dinamică

Se consideră următorul algoritm (unde a, b sunt tablouri multidimensionale globale):

```
d(n,m,k)
{
  if (n <= 0 || m <= 0 || k <= 0) {
    return 0;
  }
  best = 0;
  for (i = 0; i * i < n; i = i * 2) {
    for (j = 0; j < m; j = j * 2) {
      temp = d(i, j, k - 3);
      if (temp + a[i][j] * b[j][k] > best) {
        best = temp + a[i][j] * b[j][k];
      }
    }
  }
  return best;
}
```

(a) (6p) Calculați complexitatea-timp a algoritmului în cazul cel mai nefavorabil.
(b) (8p) Proiectați un algoritm echivalent care folosește tehnica de *memoizare*.
(c) (6p) Justificați (pe scurt) faptul că algoritmul proiectat la punctul anterior rezolvă aceeași problemă și calculați complexitatea-timp a acestuia.

Q4 Backtracking, Probleme NP-complete.

Considerăm problema SAT și rezolvarea ei prin metoda backtracking prezentată la curs.

(a) (10p) Reprezentați pe foaia de examen arborele de căutare pentru formula $(x_1 \wedge x_2) \wedge \neg(x_2 \wedge \neg x_3) \wedge \neg(x_4 \vee x_1)$ (presupunem că dorim să găsim toate soluțiile, nu doar prima). Marcați nodurile unde are loc *pruning*ul (lipsă marcaj = 0 puncte).
(b) (10p) Folosind reducerea de tip Karp de la problema 3-SAT la problema 3-COL (3-colorabilitate – dându-se un graf neorientat, să se determine dacă nodurile acestuia pot fi colorate cu 3 culori astfel încât orice două noduri adiacente să aibă culori diferite) care a fost propusă pentru rezolvare la seminar, găsiți transformarea instanței $(\neg x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_2 \vee \neg x_3 \vee x_4) \wedge (\neg x_2 \vee x_3 \vee \neg x_3)$ într-o instanță corespunzătoare pentru 3-COL și explicați pe scurt reducerea.