UNIVERSITATEA NAȚIONALĂ DE ȘTIINȚĂ ȘI TEHNOLOGIE
POLITEHNICA BUCUREȘTI

Facultatea _____
Martie 2024

Numărul legitimației de bancă _____

Numele _____

Prenumele tatălui _____

Prenumele _____

CHESTIONAR DE CONCURS

DISCIPLINA: Informatică **I**
VARIANTA **Test de antrenament 2** (Subiect realizat de Mihai Nan)

1. Fie `a`, `b` și `c` trei variabile întregi care pot lua valori 0 sau 1. Care dintre următoarele expresii de mai jos are tot timpul valoarea 1, indiferent de valorile pe care le vor lua `a`, `b` și `c`.

   A. `(b && ((a && b) || !(a && b)))`

   B. `(a && (b && c)) || (b && a)`

   C. `(a && (b && c)) || !(b && a)`

   D. `!(a && b) || (((c && a) || (c && !a)) && b)`

   E. `(b && ((a && b) || !(a && b))) || !(c && b)`

   F. `!(a && b) || ((c && a) || (c && !a))`

2. Fie tipul înregistrare **punct** care memorează coordonatele carteziene ale unui punct din planul xOy. Pornind de la acest tip, definim tipul înregistrare **poligon** care memorează numărul de vârfuri și coordonatele acestora pentru un poligon ce conține maxim 100 de vârfuri. Cum putem să determinăm lungimea laturii dintre punctele aflate pe pozițiile i și j într-un poligon reprezentat prin variabila **pol** de tip înregistrare **poligon**?

   Vom presupune că avem definite următoarele două subprograme: **sqrt** pentru a calcula radicalul unui număr real și **sqr** pentru a ridica un număr real la pătrat.

Limbajul C/C++

```c
typedef struct punct {
  float x, y;
} punct;
typedef struct poligon {
  punct v[100];
  int numar;
} poligon;
poligon pol;
```

Limbajul Pascal

```pascal
type punct = record
  x, y : real;
end;
type poligon = record
  v : array [1..100] of punct;
  numar : integer;
end;
var pol: poligon;
```

   A. `sqrt(sqr(poligon.v[i].x - poligon.v[j].x) + sqr(poligon.v[i].y - poligon.v[j].y))`

   B. `sqrt(sqr(pol.x.v[i] - pol.x.v[j]) + sqr(pol.x.v[i] - pol.y.v[j]))`

   C. `sqrt(sqr(pol.v[i].x - pol.v[j].x) + sqr(pol.v[i].y - pol.v[j].y))`

   D. `sqrt(sqr(pol.punct.v[i] - pol.punct.v[j]) + sqr(pol.punct.v[i] - pol.punct.v[j]))`

   E. `sqrt(sqr(v[i].x - v[j].x) + sqr(v[i].y - v[j].y))`

   F. `sqrt(sqr(v[i].pol.x - v[j].pol.x) + sqr(v[i].pol.y - v[j].pol.y))`

3. Considerăm că avem definite următoarele subprograme:

- `int search(char s1[], char s2[])` – caută șirul de caractere `s2` în șirul de caractere `s1` și întoarce indexul primei apariții;
- `void removeChars(char s[], int start, int end)` – elimină din șirul de caractere s toate caracterele care încep de la poziția `start` și se termină la poziția `end`;
- `void copy(char src[], char dest[])` – copiază șirul de caractere `src` în șirul de caractere `dest`;
- `int length(char s[])` – determină lungimea șirului de caractere `s`.

Pornind de la aceste subprograme și de la 3 șiruri de caractere care au fost declarate pentru a putea reține maximum 100 de caractere, specificați ce secvență de instrucțiuni putem folosi pentru a șterge din șirul de caractere `s1` a doua apariție a șirului de caractere `s2`. Este garantat că avem setat conținutul pentru ambele șiruri de caractere (`s1` și `s2`) astfel încât șirul `s2` apare de cel puțin două ori în `s1`.

A.
```
int pos1 = search(s1, s2);
removeChars(s1, pos1, pos1 + length(s2));
```

B.
```
int pos1 = search(s1, s2);
copy(s1, s3);
removeChars(s3, pos1, pos1 + length(s2));
int pos2 = search(s3, s2);
removeChars(s1, length(s2) + pos2, length(s2) + pos2);
```

C.
```
int pos1 = search(s1, s2);
copy(s1, s3);
removeChars(s3, pos1, pos1 + length(s2));
int pos2 = search(s3, s2);
removeChars(s1, pos2, length(s2) + pos2);
```

D.
```
int pos1 = search(s1, s2);
copy(s3, s1);
removeChars(s3, pos1, pos1 + length(s2));
int pos2 = search(s3, s2);
removeChars(s1, pos2, length(s2) + pos2);
```

E.
```
int pos1 = search(s1, s2);
removeChars(s1, pos1, pos1 + length(s2));
int pos2 = search(s1, s2);
removeChars(s1, pos2, length(s2) + pos2);
```

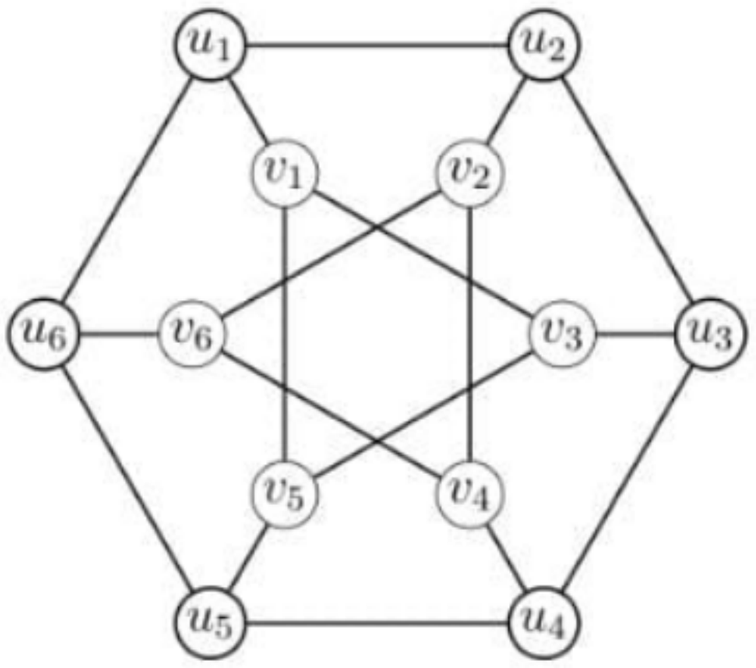
F.
```
int pos1 = search(s1, s2);
copy(s1, s3);
removeChars(s3, pos1, pos1 + length(s2));
int pos2 = search(s3, s2);
removeChars(s1, length(s2) + pos2, 2 * length(s2) + pos2);
```

4. Scriem un program care folosește metoda backtracking pentru a genera toate parolele posibile de dimensiune 7 care conțin doar caracterele a, b cu constrângerea ca parola să înceapă cu litera b, să se termine cu litera b sau să conțină exact 4 apariții ale literei b. Care este numărul total de parole generate?

A. C_5^4
B. $C_5^4 + C_7^2$
C. 120
D. 101
E. 149
F. 210

Probleme propuse de **Mihai Nan** Contact: mihai.nan@upb.ro

5. În câte moduri se poate colora următorul graf neorientat folosind 3 culori astfel încât 2 noduri adiacente să nu aibă aceeași culoare?



A. 0

B. 6

C. 90

D. 96

E. 108

F. 144

6. Se dau 4 tipuri de bancnote pentru care cunoaștem valorile $\{1, 2, 3, 5\}$. Dorim să determinăm numărul de modalități prin care putem plăti suma $S = 13$ folosindu-ne de bancnotele disponibile. Vom considera că pentru fiecare tip bancnotă avem un număr nelimitat de exemplare. În câte moduri distincte putem plăti suma 13?

A. 18 B. 21 C. 24 D. 29 E. 34 F. 40

7. Un arbore cu 9 noduri, numerotate de la 1 la 9, este memorat cu ajutorul vectorului de tați $t = (9, 3, 4, 7, 3, 9, 0, 7, 2)$. Care este numărul minim de muchii care trebuie eliminate pentru ca lungimea celui mai lung lanț, cu noduri distincte, cu o extremitate în rădăcină să fie 3 și graful obținut după eliminarea muchiilor și nodurilor care rămân izolate să fie tot arbore?

A. 0 B. 1 C. 2 D. 3 E. 4 F. 5

8. Pornim de la următorul program pentru care citim de la tastatură două numere naturale nenule. Care dintre următoarele afirmații este corectă?

```
int func(int x, int y, int z) {
    if (y == 0)
        return z / x;
    else
        return func(y, x % y, z);
}
```

```
int main() {
    int x, y;
    cin >> x >> y;
    cout << func(x, y, x * y) << "\n";
    return 0;
}
```

A. Rezultatul funcției `func` reprezintă cel mai mare divizor comun pentru `x` și `y`.

B. Rezultatul funcției `func` reprezintă câtul împărțirii lui `x * y` la restul împărțirii lui `x` la `y`.

C. Rezultatul funcției `func` reprezintă cel mai mic divizor comun pentru `x` și `y`.

D. Rezultatul funcției `func` reprezintă cel mai mare divizor comun pentru `x * y` și `y`.

E. Funcția `func` verifică dacă `x` și `y` sunt prime între ele.

F. Rezultatul funcției `func` reprezintă cel mai mic multiplu comun pentru `x` și `y`.

Probleme propuse de **Mihai Nan** Contact: mihai.nan@upb.ro

9. Care o să fie conținutul fișierului **admitere.out** după rularea următorului program?

```
ofstream fout("admitere.out");
void f(char sir[], int idx) {
  if (idx == strlen(sir)) {
    fout<<"END\n";
    return;
  }
  if (idx % 2 == 0) {
    sir[idx] = '*';
    f(sir, idx+1);
    fout<<sir<<"\n";
  }
  else {
    sir[idx] = '-';
    fout<<sir<<"\n";
    f(sir, idx+1);
  }
}
int main() {
  char sir[] = "INFO";
  f(sir, 0);
  fout.close();
  return 0;
}
```

A.
```
*-FO
*-*-
END
*-*-
*-*-
```

B.
```
*NFO
*-*O
*-*-
*-*-
END
```

C.
```
*NFO
*-FO
*-*O
*-*-
END
```

D.
```
*-*-
*-*O
*-FO
*NFO
END
```

E.
```
*-*-
*-*O
END
*-FO
*NFO
```

F.
```
*NFO
*-*O
END
*-FO
*-*-
```

10. Specificați care este complexitatea temporală pentru următorul algoritm (V_1 tablou unidimensional cu m_1 elemente și V_2 tablou unidimensional cu m_2):

Algoritm(V_1, V_2, n, m_1, m_2)
  $S_1 \leftarrow 0$; $S_2 \leftarrow 0$
  for $i \leftarrow 1$ **to** n **do**
    for $j \leftarrow 1$ **to** m_1 **do**
      if $V_1[j] \mod i == 0$ **then**
        $S_1 \leftarrow S_1 + V_1[j]$
      end if
    end for
    $k \leftarrow 1$
    while $k \leq m_2$ **do**
      $S_2 \leftarrow S_2 + V_2[k]$
      $k \leftarrow 2 * k$
    end while
  end for

A. $\Theta(n \cdot (m_1 + m_2))$
B. $\Theta(n \cdot m_1 \cdot m_2)$
C. $\Theta(n \cdot m_1 + \log(m_2^n))$
D. $\Theta(n + m_1 + m_2)$
E. $\Theta(n^{m_1+m_2})$
F. $\Theta(n)$