Tema 11

Subiectul I

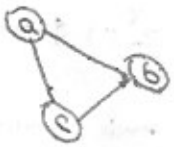
1.

$\Rightarrow \delta \leq c$ sau $a \geq d$

c)

2. Graful dat are 6 valori de o în matricea de adiacență, ce corespund elementelor de pe diagonala principală. Rezultă că celelalte valori sunt 1, deci graful este complet (K_6).

Orice subgraf cu 3 vârfuri al K_6 este un K_3, deoarece în K_6 există muchie între orice două noduri.



K_3

Cum K_3 este euleriean, răspunsul este egal cu numărul de subgrafuri ce conțin 3 vârfuri $\Rightarrow C_6^3 = 20$

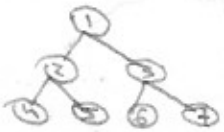
d)

3. a)

4. Considerăm un arbore cu rădăcină în care orice nod, în afară de frunze, are cel mult doi fii. Un astfel de arbore este binar, deoarece fiecare dintre noduri are cel mult doi descendenți.

Un arbore binar complet este un arbore binar în care nodul rădăcină este nod care nu este frunză are exact doi descendenți

Exemplu de arbore binar:



Considerăm p ca fiind „numărul de niveluri" al arborelui și îl notăm cu p. Se observă că p este egal cu 1+ înălțimea arborelui, iar primul nod (rădăcina) se află pe nivelul 1, următoarele două noduri se află pe nivelul 2, ș.a.m.d.

Se observă că pt. fiecare nivel k, numărul de noduri de pe acel nivel este egal cu 2^{k}.

Deci, numărul total de noduri dintr-un arbore binar complet cu l nivele este egal cu $\sum_{k=1}^{l} 2^{k-1} = 2^{l}-1$

Pentru ca un arbore să aibă un număr cât mai mare de frunze, nodurile acestuia trebuie să aibă numărul maxim permis de descendenți.

Pentru arborele din această problemă, vom încerca să construim un arbore binar complet, în funcție de nodurile disponibile.

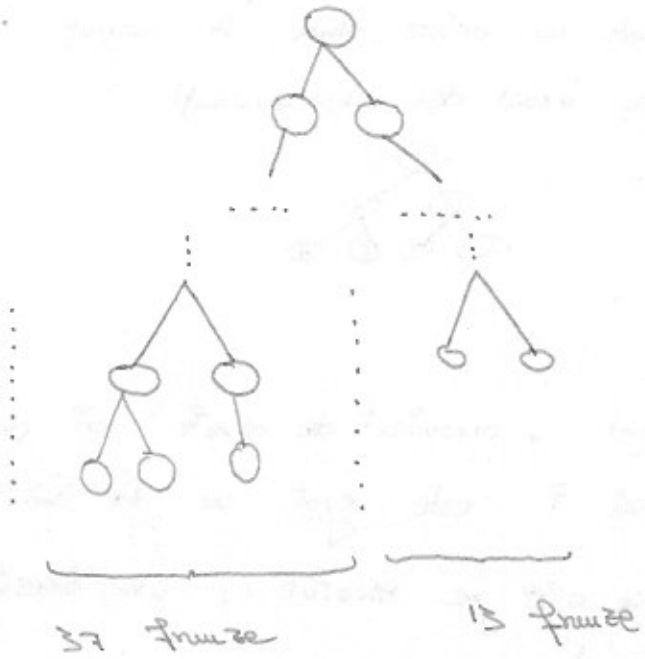
Rezolvăm inecuația nr. noduri $= 2^{l}-1 \leq 100$ (nu avem decât 100 de noduri)

$\Rightarrow 2^{l}-1 \leq 100 \Rightarrow 2^{l} \leq 101 \Rightarrow l \leq \lfloor \log_2 101 \rfloor$.

Cel mai mare l pt care inegalitatea este adevărată este $l=6$.

Deci putem obține un arbore binar complet cu 63 de noduri, având $2^{l-1} = 32$ de frunze.

Pentru a folosi toate nodurile, vom distribui cele 37 de noduri rămase frunzelor existente. Numărul minim de frunze pe care putem distribui acele 37 de noduri este 19, deci arborele obținut va avea un număr total de $37+(32-19)=50$



5.

b)

Subiectul II

1. Algoritmul descris în pseudocod determină, pentru un nr. natural nenul dat n, câte dintre perechile de divizori (k, n/k) conțin elemente cu aceeași paritate.

ex: pt. n=36, perechile sunt (1, 36), (2, 18), (3, 12), (4, 9), (6, 6) și numărul de perechi cu elemente de aceeași paritate este egal cu 2

a) 21

b) 101

c)

```
# include <iostream>
using namespace std;
int main()
{
  int k, n, a, b;
  k=0; a=1;
  cin>>n;
while( a*a<=n)
  {
    if(n%a==0)
    {
      b=n/a;
      if(a%2==b%2)
        k++;
    }
  a++;
  }
cout<<k;
return 0;
}
```

d)

citește n
$fn \leftarrow 0$

pentru $a \leftarrow \overline{1, [\sqrt{n}]}$ execută
  dacă $n \% a =$ atunci
    $b \leftarrow [n/a]$
    dacă $a \% 2 = b \% 2$ atunci
      $fn \leftarrow fn+1$

sfârșit pt

2.

```
for( i=0 ; i< D.nrFisiere ; i++)
  if(D.F[ i].numeane [0] ==='A')
     suma += D.F[i].dimensiune ;
```

3.

bac la informatica

Subiectul III.

1.
```
# include <iostream>
using namespace std;
int matrice[50][50], fin[50], col[50],N;
int main()
{ cin>>N;
  for( int i=0; i< N; i++)
  for( int j=0; j< N; j++)
  {   cin>> matrice[i][j];
      if( matrice[i][j]==-1)
      { fin[i]++ ; col[j]++;}
  }
for( int i=0 ; i< N; i++)
for( int j=0; j< N; j++)
  if( fin[i]==0)
  matrice[i][j]= col[j];

for( int i=0 ; i< N ; i++)
{
   for( int j=0; j< N ; j++)
      cout << matrice[i][j]<<' ';
   cout <<'\n';
  }
 return 0;
}
```

Explicație pb. 1

Folosim două tablouri unidimensionale, fin și col, care vor memora numărul de valori de 1 de pe fiecare linie, respectiv coloană.
Pentru fiecare linie în parte, verificăm dacă aceasta nu conține nicio valoare de 1, caz în care atribuim elementelor de pe această linie numărul de valori de 1 de pe coloana pe care se află.

2.

```
void aPtrone( int n, int &lu, int p[1000])
{
    lu = 0;
    for( int nr=1; nr<n; nr++)
    {
        int a=nr, b=n, r;
        do {
            r=a%b;
            a=b;
            b=r;
        } while(r!=0);
        if(a==1) p[lu++]=nr;
    }
}
```

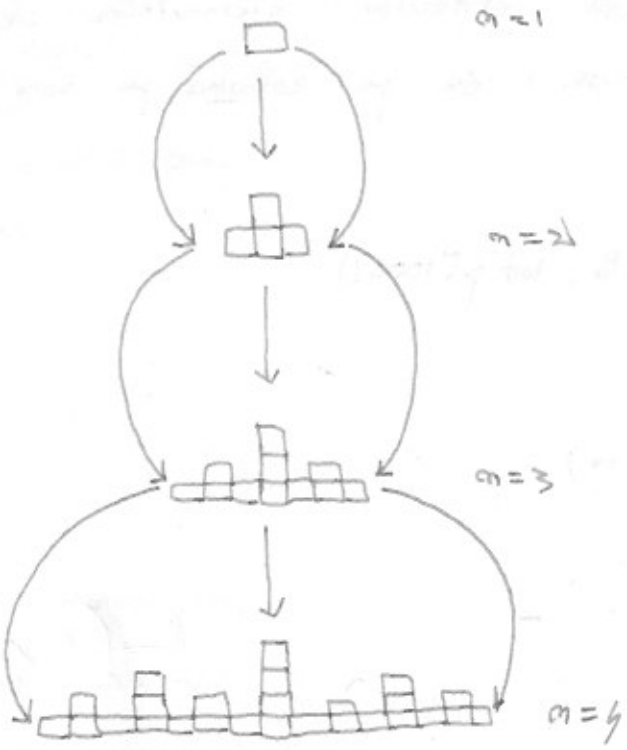
3. Configurația optimă pentru n=1 este : 

Configurația optimă pentru n=2 este

Configurația optimă pentru n=3 este 

Se observă că pentru a obține un număr maxim de stive, între orice două stive de aceeași înălțime trebuie să punem o stivă cu înălțime cât mai mică, dât timp cât sunt respectate condițiile problemei.

Astfel, o soluție optimă pentru un anumit număr n, se obține din soluția optimă pentru n-1, adăugând în mijloc configurației de stive (dacă o stivă de înălțime n) adăugăm atât la stânga, cât și la dreapta configurația optimă pentru n-1)



Se consideră două șiruri $(stive_n)_{n\geq 1}$ și $(cutii_n)_{n\geq 1}$, ce reprezintă valorile numărului maxim de stive, respectiv de cutii pentru o anumită valoare a lui n.

Cum pentru a obține configurația optimă pentru n adăugăm în stânga și în dreapta configurația optimă pt. n-1 și plasăm în mijloc încă o stivă de înălțime n, putem defini șirul $cutii_n$ cu ajutorul unei relații recurente:

$\boxed{stive_n = 2 * stive_{n-1} + 1}$, și $stive_1 = 1$

Analog, pentru șirul $cutii_n$, $\boxed{cutii_n = 2 * cutii_{n-1} + n}$, și $cutii_1 = 1$

Deci, putem determina soluția problemei folosindu-ne de două metode recursive, ceea ce va obține o complexitate de timp liniară.

Având în vedere modul în care sunt definite șirurile, putem încerca să găsim o definiție a șirului care să depindă doar de n, pentru a rezolva problema în $O(1)$

$$\text{stive}_n = 2^* \cancel{\text{stive}_{n-1}} + 1 \quad | \cdot 2^0$$

$$\cancel{\text{stive}_{n-1}} = 2^* \cancel{\text{stive}_{n-2}} + 1 \quad | \cdot 2^1$$

$$\cancel{\text{stive}_{n-2}} = 2^* \cancel{\text{stive}_{n-3}} + 1 \quad | \cdot 2^2$$

$$\cancel{\text{stive}_{n-3}} = 2^* \cancel{\text{stive}_{n-4}} + 1 \quad | \cdot 2^3 \quad \Rightarrow \boxed{\text{stive}_n = 2^n - 1}$$

$$\vdots$$

$$\cancel{\text{stive}_3} = 2^* \cancel{\text{stive}_2} + 1 \quad | \cdot 2^{n-3}$$

$$\cancel{\text{stive}_2} = 2^* \cancel{\text{stive}_1} + 1 \quad | \cdot 2^{n-2}$$

$$\cancel{\text{stive}_1} = 1 \quad | \cdot 2^{n-1}$$

$$\text{cutii}_n = 2^* \cancel{\text{cutii}_{n-1}} + n \quad | \cdot 2^0$$

$$\cancel{\text{cutii}_{n-1}} = 2^* \cancel{\text{cutii}_{n-2}} + (n-1) \quad | \cdot 2^1$$

$$\cancel{\text{cutii}_{n-2}} = 2^* \cancel{\text{cutii}_{n-3}} + (n-2) \quad | \cdot 2^2$$

$$\vdots$$

$$\Rightarrow \text{cutii}_n = 1 \cdot 2^{n-1} + 2 \cdot 2^{n-2} + 3 \cdot 2^{n-3} + \ldots + (n-1) \cdot 2^1 + n \cdot 2^0$$

$$\cancel{\text{cutii}_3} = 2^* \cancel{\text{cutii}_2} + 3 \quad | \cdot 2^{n-3}$$

$$\cancel{\text{cutii}_2} = 2^* \cancel{\text{cutii}_1} + 2 \quad | \cdot 2^{n-2}$$

$$\cancel{\text{cutii}_1} = 1 \quad | \cdot 2^{n-1}$$

$$\Rightarrow \text{cutii}_n = \sum_{k=1}^{n} \left(k \cdot 2^{n-k} \right)$$

$$S = \sum_{k=1}^{n} \left(k \cdot 2^{n-k} \right) = 2^n \sum_{k=1}^{n} \frac{k \cdot 2^{n-k}}{2^n} = 2^n \sum_{k=1}^{n} k \cdot 2^{-k} = 2^n \underbrace{\sum_{k=1}^{n} \frac{k}{2^k}}_{F}$$

$$\Rightarrow S = 2^n \cdot F$$

$$F = \left(\frac{1}{2} + \frac{2}{4} + \frac{3}{8} + \frac{4}{16} + \frac{5}{32} + \ldots + \frac{n-1}{2^{n-1}} + \frac{n}{2^n} \right)$$

$$2F = \left(1 + \frac{2}{2} + \frac{3}{4} + \frac{4}{8} + \frac{5}{16} + \ldots + \frac{n-1}{2^{n-2}} + \frac{n}{2^{n-1}} \right)$$

$$\Rightarrow 2F - F = F = \left(1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{16} + \ldots + \frac{1}{2^{n-1}} \right) - \frac{n}{2^n} \quad \left(\because, \ b_1 = 1, \ q = \frac{1}{2} \right)$$

$$F = b_1 \cdot \frac{1-q^n}{1-q} - \frac{n}{2^n} \Rightarrow F = \frac{1-\left(\frac{1}{2}\right)^d}{\frac{1}{2}} - \frac{n}{2^n} \Rightarrow F = 2\left(1-\frac{1}{2^n}\right) - \frac{n}{2^n}$$

$$\Rightarrow F = \overset{2^n}{2} - \frac{2}{2^n} - \frac{n}{2^d} \Rightarrow F = \frac{2^{n+1} - n - 2}{2^n}$$

$$S = F \cdot 2^n \Rightarrow S = 2^{n+1} - n - 2 \Rightarrow \boxed{cutii_n = 2^{n+1} - n - 2}$$

```
# include <iostream>
# include <fstream>

ifstream fin("boc.txt");

int main()
{
    int n; fin>>n;
    cout<< ((1<<n)-1<<1 << (1<<(n+1))-n-2;
    return 0;
}
```