

# Партиције природног броја

- Партиција (подела/разбијање) природног броја  $n$  је низ позитивних природних бројева  $a_1 \geq a_2 \geq \dots \geq a_m \geq 1$  тако да је  $n = a_1 + a_2 + \dots + a_m$
- При томе је:  $1 \leq m \leq n$  и  $n \geq 1$
- Лексикографски поредак
  - прва партиција је  $a_1 = n$
  - последња партиција је  $1 + 1 + \dots + 1 = n$

Све партиције  $n = 7$

| ----- |   |   |   |   |   |   |
|-------|---|---|---|---|---|---|
| 7     |   |   |   |   |   |   |
| 6     | 1 |   |   |   |   |   |
| 5     | 2 |   |   |   |   |   |
| 5     | 1 | 1 |   |   |   |   |
| 4     | 3 |   |   |   |   |   |
| 4     | 2 | 1 |   |   |   |   |
| 4     | 1 | 1 | 1 |   |   |   |
| 3     | 3 | 1 |   |   |   |   |
| 3     | 2 | 2 |   |   |   |   |
| 3     | 2 | 1 | 1 |   |   |   |
| 3     | 1 | 1 | 1 | 1 |   |   |
| 2     | 2 | 2 | 1 |   |   |   |
| 2     | 2 | 1 | 1 | 1 |   |   |
| 2     | 1 | 1 | 1 | 1 | 1 |   |
| 1     | 1 | 1 | 1 | 1 | 1 | 1 |

# C/C++ код за генерисање свих партиција природног броја

- Партиција се чува у низу  $p[0], p[1], \dots, p[n-1]$
- Уколико је  $p[k] > 0$  тада је  $p[k]$  члан партиције
- Пронаћи највеће  $k$  тако да је  $p[k] > 1$  и наћи број јединица  $q$
- $p[k] = p[k] - 1, q = q + 1$ 
  - Ако је  $p[k] \geq q$ :  $p[k+1] = q$
  - Ако је  $q > p[k]$ :  
 $p[k+1] = p[k]$   
 $q = q - p[k], k = k + 1$   
поновити корак докле год може

```
1 #include <stdio.h>
2
3 bool next_partition(int* p, int n)
4 {
5     int k, i;
6     int q=0;
7
8     if(p[0]==0) // the first one
9     {
10         p[0]=n;
11         return true;
12     }
13     else // finding k
14     {
15         for(i=0; i<n; i++)
16         {
17             if(p[i]!=0) k=i;
18             else break;
19         }
20     }
21
22     while (k>=0 && p[k]==1)
23     {
24         q++;
25         k--;
26     }
27
28     if (k<0) return false;
29
30     p[k]--;
31     q++;
32
33     while(q > p[k])
34     {
35         p[k+1] = p[k];
36         q = q - p[k];
37         k++;
38     }
39
40     p[k+1] = q;
41     k++;
42
43     for(i=k+1; i<n; i++)
44         p[i]=0;
45
46     return true;
47 }
48
49 void driver_next_partition(void)
50 {
51     int n=7;
52     int* p=new int [n];
53
54     for(int i=0; i<n; i++)
55         p[i]=0;
56
57     while(next_partition(p, n))
58     {
59         //print current partition
60         for(int i=0; i<n && p[i]>0; i++)
61             printf("%2d ", p[i]);
62         printf("\n");
63     }
64
65     delete [] p;
66 }
```

# Илустрација алгоритма

Све партиције  $n = 7$

-----

|   |   |   |   |   |   |   |                                                |
|---|---|---|---|---|---|---|------------------------------------------------|
| 7 |   |   |   |   |   |   | $k=0, q=0+1: p[0]=7-1, p[1]=q=1$               |
| 6 | 1 |   |   |   |   |   | $k=0, q=1+1: p[0]=6-1, p[1]=q=2$               |
| 5 | 2 |   |   |   |   |   | $k=1, q=0+1: p[1]=2-1, p[2]=q=1$               |
| 5 | 1 | 1 |   |   |   |   | $k=0, q=2+1: p[0]=5-1, p[1]=q=3$               |
| 4 | 3 |   |   |   |   |   | $k=1, q=0+1: p[1]=3-1, p[2]=q=1$               |
| 4 | 2 | 1 |   |   |   |   | $k=1, q=1+1: p[1]=2-1, p[2]=q=1$               |
| 4 | 1 | 1 | 1 |   |   |   | $k=0, q=3+1: p[0]=4-1, p[1]=p[0], p[2]=q-p[1]$ |
| 3 | 3 | 1 |   |   |   |   | ...                                            |
| 3 | 2 | 2 |   |   |   |   |                                                |
| 3 | 2 | 1 | 1 |   |   |   |                                                |
| 3 | 1 | 1 | 1 | 1 |   |   | $k=0, q=3+1: p[0]=3-1, p[1]=p[0], p[2]=q-p[1]$ |
| 2 | 2 | 2 | 1 |   |   |   | ...                                            |
| 2 | 2 | 1 | 1 | 1 |   |   |                                                |
| 2 | 1 | 1 | 1 | 1 | 1 |   |                                                |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | $k=-1, q=7: \text{нема наредне партиције}$     |

# Укупан број партиција: не постоји аналитички израз

The man who  
knew infinity

$$P(n) \approx \frac{1}{4n\sqrt{3}} e^{\pi\sqrt{\frac{2n}{3}}}$$

|          |     |          |       |          |        |          |          |
|----------|-----|----------|-------|----------|--------|----------|----------|
| P ( 1) = | 1   | P (21) = | 792   | P (41) = | 44583  | P (61) = | 1121505  |
| P ( 2) = | 2   | P (22) = | 1002  | P (42) = | 53174  | P (62) = | 1300156  |
| P ( 3) = | 3   | P (23) = | 1255  | P (43) = | 63261  | P (63) = | 1505499  |
| P ( 4) = | 5   | P (24) = | 1575  | P (44) = | 75175  | P (64) = | 1741630  |
| P ( 5) = | 7   | P (25) = | 1958  | P (45) = | 89134  | P (65) = | 2012558  |
| P ( 6) = | 11  | P (26) = | 2436  | P (46) = | 105558 | P (66) = | 2323520  |
| P ( 7) = | 15  | P (27) = | 3010  | P (47) = | 124754 | P (67) = | 2679689  |
| P ( 8) = | 22  | P (28) = | 3718  | P (48) = | 147273 | P (68) = | 3087735  |
| P ( 9) = | 30  | P (29) = | 4565  | P (49) = | 173525 | P (69) = | 3554345  |
| P (10) = | 42  | P (30) = | 5604  | P (50) = | 204226 | P (70) = | 4087968  |
| P (11) = | 56  | P (31) = | 6842  | P (51) = | 239943 | P (71) = | 4697205  |
| P (12) = | 77  | P (32) = | 8349  | P (52) = | 281589 | P (72) = | 5392783  |
| P (13) = | 101 | P (33) = | 10143 | P (53) = | 329931 | P (73) = | 6185689  |
| P (14) = | 135 | P (34) = | 12310 | P (54) = | 386155 | P (74) = | 7089500  |
| P (15) = | 176 | P (35) = | 14883 | P (55) = | 451276 | P (75) = | 8118264  |
| P (16) = | 231 | P (36) = | 17977 | P (56) = | 526823 | P (76) = | 9289091  |
| P (17) = | 297 | P (37) = | 21637 | P (57) = | 614154 | P (77) = | 10619863 |
| P (18) = | 385 | P (38) = | 26015 | P (58) = | 715220 | P (78) = | 12132164 |
| P (19) = | 490 | P (39) = | 31185 | P (59) = | 831820 | P (79) = | 13848650 |
| P (20) = | 627 | P (40) = | 37338 | P (60) = | 966467 | P (80) = | 15796476 |

# Партиције природног броја са тачно $m$ чланова

- $a_1 \geq a_2 \geq \dots \geq a_m \geq 1$  тако да је  
 $n = a_1 + a_2 + \dots + a_m$
- При томе је:  $2 \leq m \leq n$  задато
- Једноставна имплементација:  
прескочити партиције  $P(n)$  чији  
је број чланова различит од  $m$ 
  - Постоје и ефикасније  
имплементације

Све партиције  
 $n = 11 \quad m = 4$

| ----- |   |   |   |
|-------|---|---|---|
| 8     | 1 | 1 | 1 |
| 7     | 2 | 1 | 1 |
| 6     | 3 | 1 | 1 |
| 6     | 2 | 2 | 1 |
| 5     | 4 | 1 | 1 |
| 5     | 3 | 2 | 1 |
| 5     | 2 | 2 | 2 |
| 4     | 4 | 2 | 1 |
| 4     | 3 | 3 | 1 |
| 4     | 3 | 2 | 2 |
| 3     | 3 | 3 | 2 |

# Партиције скупа

- Партиција скупа  $S$  је подела  $S$  на подскупове  $S_m$ ,  $1 \leq m \leq |S|$  тако да важи:  
$$S = S_1 \cup S_2 \cup \dots \cup S_m$$
$$S_i \neq \{\emptyset\}, \quad 1 \leq i \leq m$$
$$i \neq j \Rightarrow S_i \cap S_j = \{\emptyset\}, \quad 1 \leq i, j \leq m$$
- $|S|$  је број елемената скупа  $S$

Све партиције

$S = \{1, 2, 3, 4\}$

-----  
 $\{1, 2, 3, 4\}$   
 $\{1, 2, 3\}, \{4\}$   
 $\{1, 2, 4\}, \{3\}$   
 $\{1, 2\}, \{3, 4\}$   
 $\{1, 2\}, \{3\}, \{4\}$   
 $\{1, 3, 4\}, \{2\}$   
 $\{1, 3\}, \{2, 4\}$   
 $\{1, 3\}, \{2\}, \{4\}$   
 $\{1, 4\}, \{2, 3\}$   
 $\{1\}, \{2, 3, 4\}$   
 $\{1\}, \{2, 3\}, \{4\}$   
 $\{1, 4\}, \{2\}, \{3\}$   
 $\{1\}, \{2, 4\}, \{3\}$   
 $\{1\}, \{2\}, \{3, 4\}$   
 $\{1\}, \{2\}, \{3\}, \{4\}$

# Пресликавање партиције скупа

Сви могући | Све партиције  
низови  $n=4$  |  $S = \{1, 2, 3, 4\}$

- Посматрајмо низ  $a_1, a_2, \dots, a_n$  који задовољава следеће услове:  
 $a_1 = 0$  и  
 $a_{j+1} \leq 1 + \max(a_1, \dots, a_j)$ ,  
 $1 \leq j < n$
- Сваки следећи елемент низа може највише за 1 да буде већи од највећег елемента пре њега у низу  
(енг: restricted growth string)
- Једнозначно пресликавање RGS на партицију скупа!
- Број партиција:  $\max(a_i) + 1$

| ----- |   |   |   |                       |
|-------|---|---|---|-----------------------|
| 0     | 0 | 0 | 0 | -> {1, 2, 3, 4}       |
| 0     | 0 | 0 | 1 | -> {1, 2, 3}, {4}     |
| 0     | 0 | 1 | 0 | -> {1, 2, 4}, {3}     |
| 0     | 0 | 1 | 1 | -> {1, 2}, {3, 4}     |
| 0     | 0 | 1 | 2 | -> {1, 2}, {3}, {4}   |
| 0     | 1 | 0 | 0 | -> {1, 3, 4}, {2}     |
| 0     | 1 | 0 | 1 | -> {1, 3}, {2, 4}     |
| 0     | 1 | 0 | 2 | -> {1, 3}, {2}, {4}   |
| 0     | 1 | 1 | 0 | -> {1, 4}, {2, 3}     |
| 0     | 1 | 1 | 1 | -> {1}, {2, 3, 4}     |
| 0     | 1 | 1 | 2 | -> {1}, {2, 3}, {4}   |
| 0     | 1 | 2 | 0 | -> {1, 4}, {2}, {3}   |
| 0     | 1 | 2 | 1 | -> {1}, {2, 4}, {3}   |
| 0     | 1 | 2 | 2 | -> {1}, {2}, {3, 4}   |
| 0     | 1 | 2 | 3 | -> {1}, {2}, {3}, {4} |

# C/C++ код за генерисање свих партиција скупа

- RGS:  $a_i = 0, 0 \leq i \leq n - 1$
- Максималне вредности елемената  $b_0 = 0$ ,  
 $b_i = \max(b_{i-1}, a_{i-1} + 1)$   
 $1 \leq i \leq n - 1$
- Полазећи од последњег елемента  $a_i$  проверити да ли  $a_i \leq b_i$ 
  - може:  $a_i = a_i + 1$  и одредити нове вредности за све  $b_i$
  - не може:  $a_i = a_i + 1$  и поновити корак за елемент  $i - 1$
- Крај је када нема елемента који се може повећати за један

```
45 bool next_partition_of_set(int* a, int n)
46 {
47     int j;
48     int* b = new int [n];
49
50     for(j=0; j<n; j++)
51         b[j]=j==0 ? 0 : std::max(b[j-1],a[j-1]+1);
52
53     for(j=n-1; j>=0; j--)
54     {
55         if(a[j]<b[j])
56         {
57             a[j]++;
58             break;
59         }
60         else
61             a[j]=0;
62     }
63     if(j==-1)
64         return false;
65
66     delete [] b;
67     return true;
68 }

70 void driver_next_partition_of_set(void)
71 {
72     int n=4;
73     int* a = new int [n];
74     int i;
75
76     for(i=0; i<n; i++)
77         a[i]=0;
78
79     do
80     {
81         for(i=0; i<n; i++)
82             printf("%2d ",a[i]);
83         printf("\n");
84     }while(next_partition_of_set(a, n));
85
86     delete [] a;
87 }
```



# Партиција скупа на тачно $m$ подскупова

- Важи све што и за партицију скупа једино је  $m$  задато
- Једноставна имплементација: узети само RGS које задовољавају  $\max(a_i) + 1 = m$
- Постоје ефикасније имплементације које су алгоритамски сложеније

Све партиције

$S = \{1, 2, 3, 4\}$

$m = 2$

-----

$\{1, 2, 3\}, \{4\}$

$\{1, 2, 4\}, \{3\}$

$\{1, 2\}, \{3, 4\}$

$\{1, 3, 4\}, \{2\}$

$\{1, 3\}, \{2, 4\}$

$\{1, 4\}, \{2, 3\}$

$\{1\}, \{2, 3, 4\}$

# Број партиција скупа

- Точно  $k$  подскупова

– Стирлингови бројеви друге врсте

$$S(n, k) = S_n^{(k)} = \left\{ \begin{matrix} n \\ k \end{matrix} \right\}$$

$S(1, 1) = 1$   
 $S(2, 1) = 1$   $S(2, 2) = 1$   
 $S(3, 1) = 1$   $S(3, 2) = 3$   $S(3, 3) = 1$   
 $S(4, 1) = 1$   $S(4, 2) = 7$   $S(4, 3) = 6$   $S(4, 4) = 1$   
 $S(5, 1) = 1$   $S(5, 2) = 15$   $S(5, 3) = 25$   $S(5, 4) = 10$   $S(5, 5) = 1$   
 $S(6, 1) = 1$   $S(6, 2) = 31$   $S(6, 3) = 90$   $S(6, 4) = 65$   $S(6, 5) = 15$   $S(6, 6) = 1$   
 $S(7, 1) = 1$   $S(7, 2) = 63$   $S(7, 3) = 301$   $S(7, 4) = 350$   $S(7, 5) = 140$   $S(7, 6) = 21$   $S(7, 7) = 1$   
 $S(8, 1) = 1$   $S(8, 2) = 127$   $S(8, 3) = 966$   $S(8, 4) = 1701$   $S(8, 5) = 1050$   $S(8, 6) = 266$   $S(8, 7) = 28$   $S(8, 8) = 1$   
 $S(9, 1) = 1$   $S(9, 2) = 255$   $S(9, 3) = 3025$   $S(9, 4) = 7770$   $S(9, 5) = 6951$   $S(9, 6) = 2646$   $S(9, 7) = 462$   $S(9, 8) = 36$   $S(9, 9) = 1$   
 $S(10, 1) = 1$   $S(10, 2) = 511$   $S(10, 3) = 9330$   $S(10, 4) = 34105$   $S(10, 5) = 42525$   $S(10, 6) = 22827$   $S(10, 7) = 5880$   $S(10, 8) = 750$   $S(10, 9) = 45$   $S(10, 10) = 1$

- Све могуће партиције

— Белови бројеви

$$B_n = \sum_{k=1}^n S(n, k)$$

|        |     |        |           |
|--------|-----|--------|-----------|
| B( 1)= | 1   | B( 8)= | 4140      |
| B( 2)= | 2   | B( 9)= | 21147     |
| B( 3)= | 5   | B(10)= | 115975    |
| B( 4)= | 15  | B(11)= | 678570    |
| B( 5)= | 52  | B(12)= | 4213597   |
| B( 6)= | 203 | B(13)= | 27644437  |
| B( 7)= | 877 | B(14)= | 190899322 |

# Партиције код којих је редослед битан

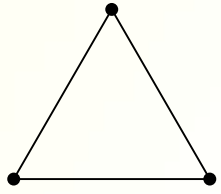
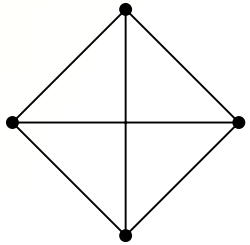
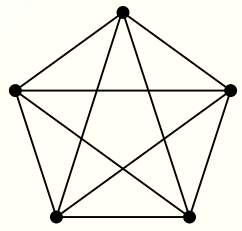
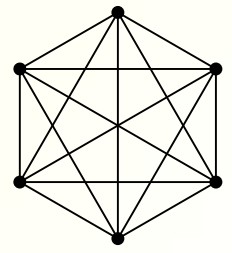
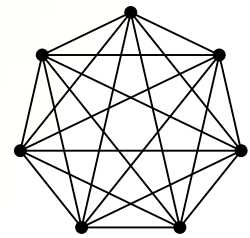
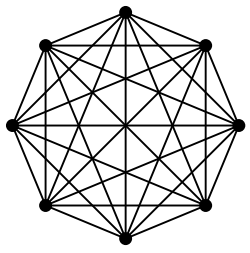
- Композиција природног броја је партиција код које је редослед сабирака битан
- Пример  $n = 7$ :  $3+2+1+1$  или  $1+1+2+3$
- Број композиција броја  $n$  је  $2^{n-1}$   
(једнозначно пресликавање  $\{0,1\}^{n-1}$ )  
 $(\overbrace{1 * 1 * 1 * \dots * 1}^n), * \in \{+, |\}$
- Партиција скупа са редоследом подскупова:  
генерисати партицију скупа +  
пермутација свих подскупова

# Претраживање по стаблима графа: комплетан граф

- Комплетан неусмерен граф са  $n$  чворова ( $K_n$ ): између сваког пара чворова постоји грана

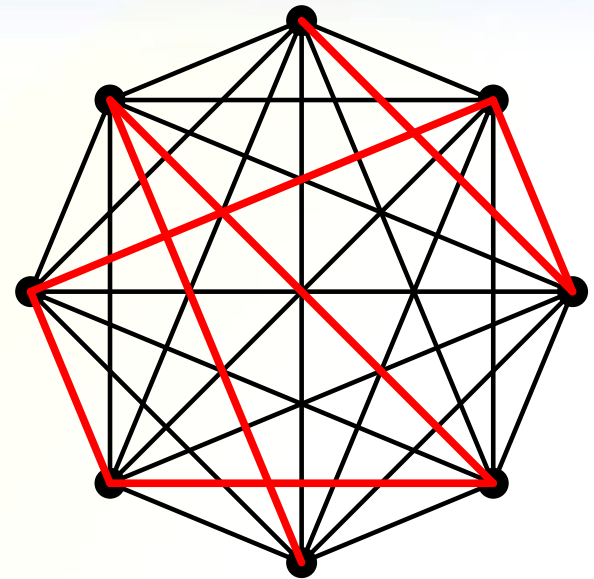
- Укупан број грана је

$$\binom{n}{2} = \frac{n(n-1)}{2}$$

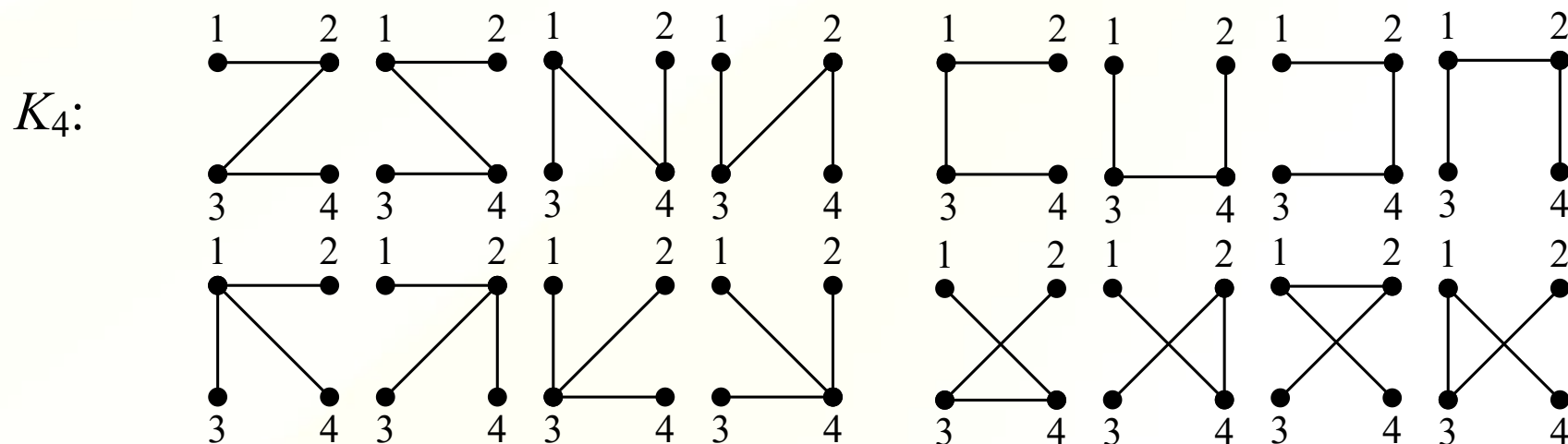
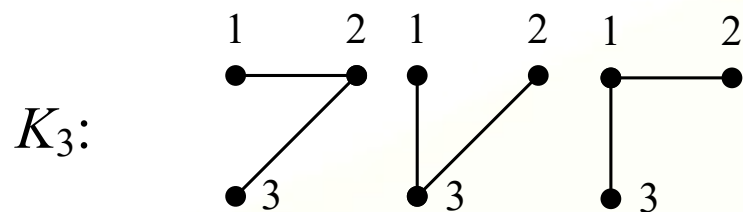
| $K_1$                                                                              | $K_2$                                                                               | $K_3$                                                                                | $K_4$                                                                                |
|------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------|
|   |   |   |   |
| $K_5$                                                                              | $K_6$                                                                               | $K_7$                                                                                | $K_8$                                                                                |
|  |  |  |  |

# Стабло графа

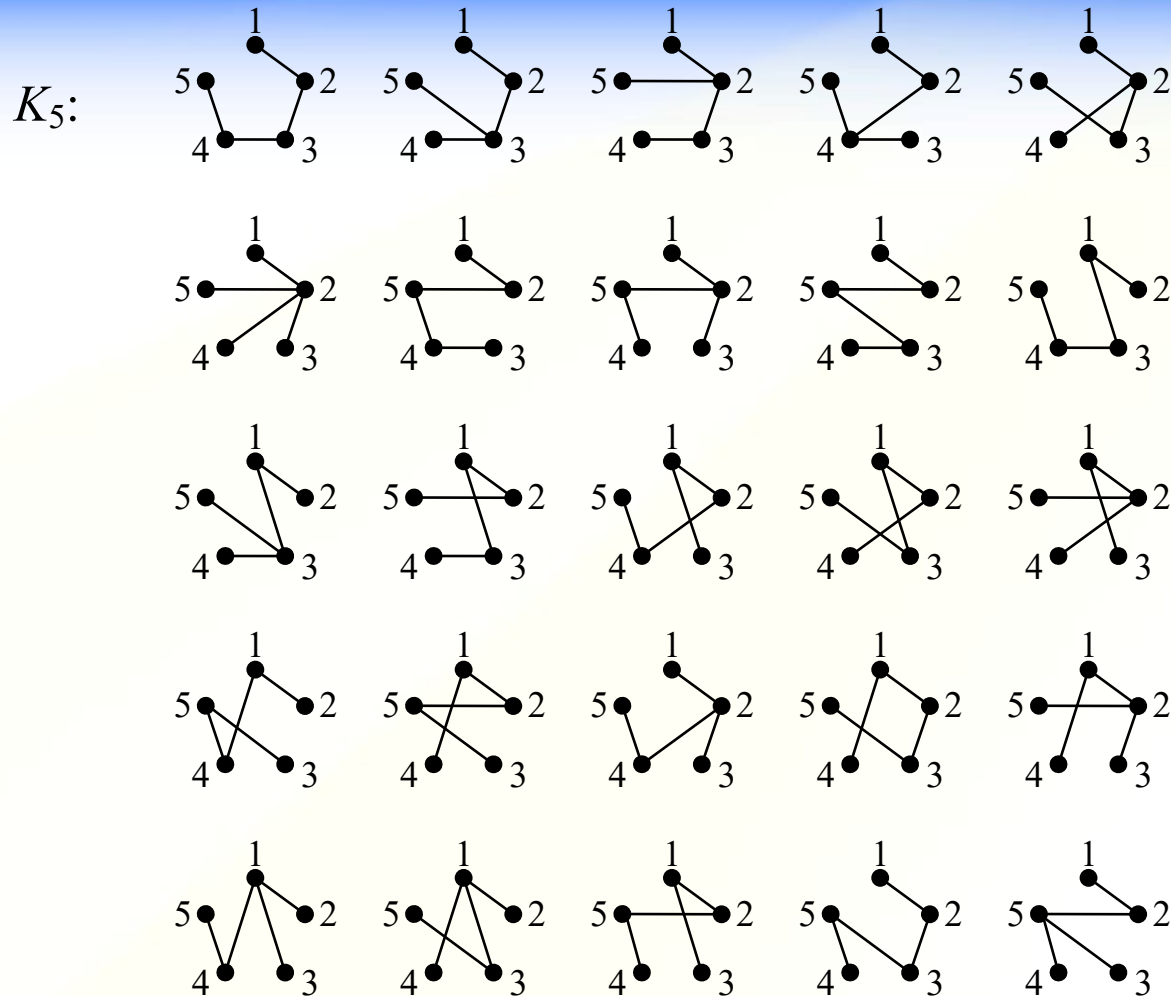
- Стабло графа (енг. *spanning tree*) је подскуп од  $n - 1$  грана које
  - повезују све чворове графа (постоји пут између произвољног пара чворова) и
  - нема петљи (до једног чвора може се доћи само на један начин)
- Колико има различитих стабала комплетног графа  $K_n$ ?



# Примери стабла комплетних графова $K_2$ , $K_3$ и $K_4$



# Пример стабла $K_5$ : 1/5 стабала (цикличка пермутација индекса)



# Колико има стабала графа?

## Кирхофова теорема

- Посматрамо произвољан граф
- Напишемо Лапласову матрицу

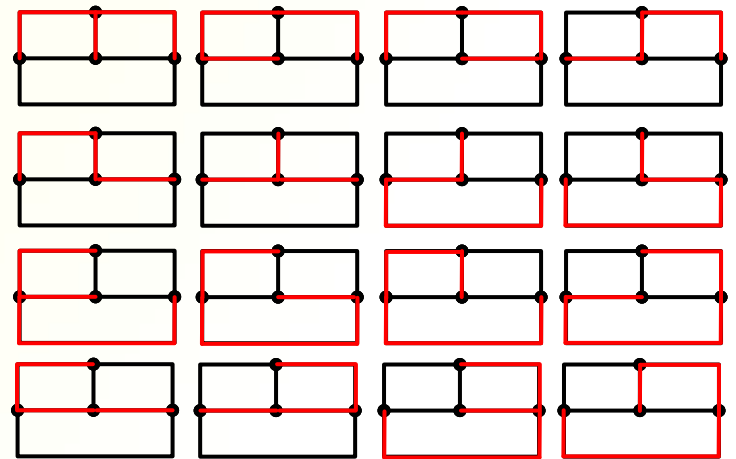
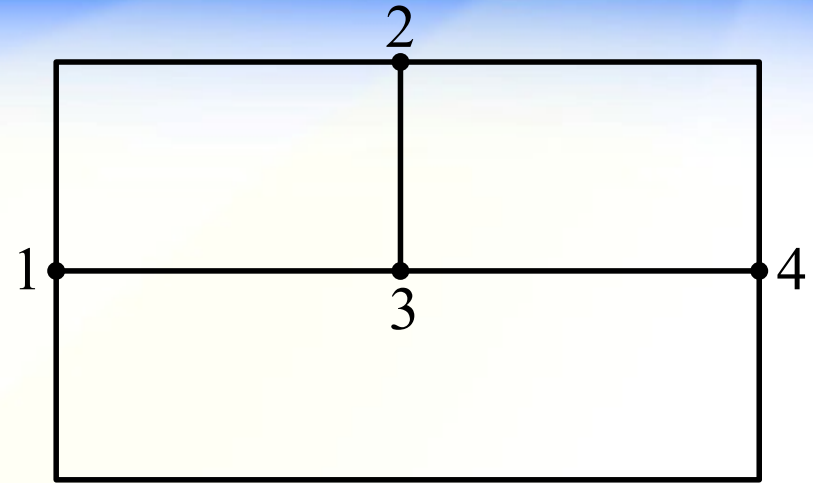
$$Q = \begin{bmatrix} +3 & -1 & -1 & -1 \\ -1 & +3 & -1 & -1 \\ -1 & -1 & +3 & -1 \\ -1 & -1 & -1 & +3 \end{bmatrix}$$

- Изоставимо један ред и колону

$$Q' = \begin{bmatrix} +3 & -1 & -1 \\ -1 & +3 & -1 \\ -1 & -1 & +3 \end{bmatrix}$$

- Детерминанта  $Q'$  је број стабала

$$\det Q' = \begin{vmatrix} +3 & -1 & -1 \\ -1 & +3 & -1 \\ -1 & -1 & +3 \end{vmatrix} = 27 - 1 - 1 - 3 - 3 - 3 = 16$$





# Колики је број стабала комплетног графа?

- Број стабала комплетног графа је  $n^{n-2}$  (Cayley's formula)
- $n^{n-2} > n!$  ако је  $n \geq 5$
- Претраживање по свим стаблима је сложенији проблем од TSP!
- Уколико граф није комплетан број стабала је мањи од  $n^{n-2}$

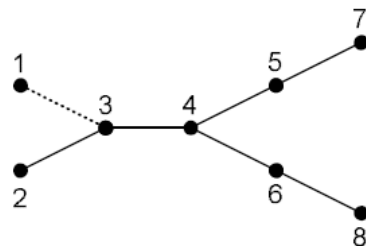
# Како генерисати сва стабла комплетног графа?

- Секвенце: варијације са понављањем  $n \geq 2$  елемената на  $n - 2$  места
  - $n = 4$  (број стабала  $K_4$  је 16):  
(1 1) (1 2) (1 3) (1 4) (2 1) (2 2) (2 3) (2 4)  
(3 1) (3 2) (3 3) (3 4) (4 1) (4 2) (4 3) (4 4)
- Постоји **једнозначно пресликавање** између оваквих секвенци и стабала комплетног графа са  $n$  чворова (Prüfer)

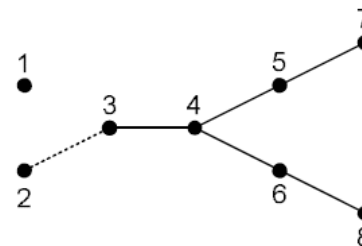
# Кодовање стабала у секвенце: алгоритам

- Лист = чвор из кога полази само једна грана
- Пронаћи лист стабла са најмањим редним бројем
- Записати редни број чвора са којим је повезан нађени лист (јединствени сусед)
- Избацити лист из стабла
- Поновити процедуру  $n - 2$  пута

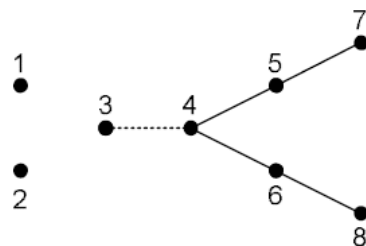
# Кодовања стабла у секвенце: пример



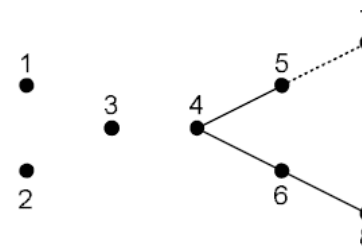
(a)  $P = (3)$



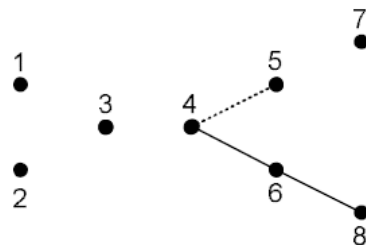
(b)  $P = (3, 3)$



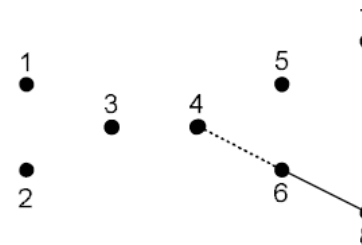
(c)  $P = (3, 3, 4)$



(d)  $P = (3, 3, 4, 5)$



(e)  $P = (3, 3, 4, 5, 4)$

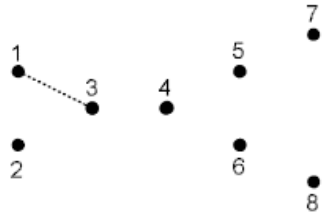


(f)  $P = (3, 3, 4, 5, 4, 6)$

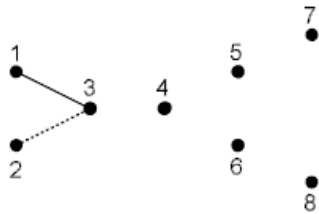
# Декодовање секвенце у стабло: алгоритам

- Задата је секвенца  $P=(p_1, p_2, \dots, p_{n-2})$
- Почети са низом чворова  
 $V = (v_1, v_2, \dots, v_n) = (1, 2, \dots, n)$
- Понављати за  $i = 1, 2, \dots, n - 2$ 
  - Пронаћи најмањи редни број чвора  $v_i$  који се не појављује у секвенци  $P$
  - Повезати  $v_i$  са  $p_i$
  - Избацити  $v_i$  из  $V$  и  $p_i$  из  $P$
- Последњи корак је повезивање преостала два елемента из  $V$

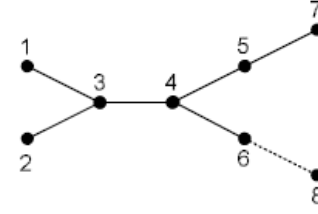
# Декодовање секвенце у стабло: пример



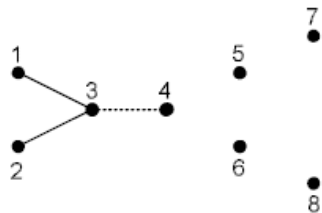
(a)  $P = (\underline{3}, 3, 4, 5, 4, 6)$ ;  $V = \{1, 2, 3, 4, 5, 6, 7, 8\}$



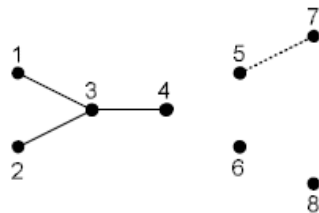
(b)  $P = (\underline{3}, 4, 5, 4, 6)$ ;  $V = \{2, 3, 4, 5, 6, 7, 8\}$



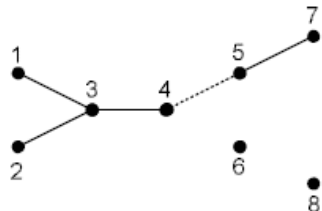
(g)  $P = ( )$ ;  $V = \{\underline{6}, \underline{8}\}$



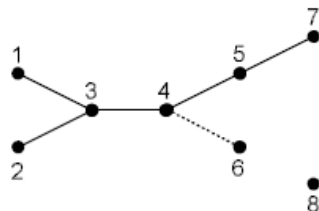
(c)  $P = (\underline{4}, 5, 4, 6)$ ;  $V = \{\underline{3}, 4, 5, 6, 7, 8\}$



(d)  $P = (\underline{5}, 4, 6)$ ;  $V = \{4, 5, 6, \underline{7}, 8\}$



(e)  $P = (\underline{4}, 6)$ ;  $V = \{4, \underline{5}, 6, 8\}$



(f)  $P = (\underline{6})$ ;  $V = \{\underline{4}, 6, 8\}$

# C/C++ код за декодовање

```
1 #include <stdio.h>
2
3 void SequenceToSpanningTree(int* P, int len, int* T)
4 {
5     int i,j,q=0;
6     int n=len+2;
7     int* V=new int [n];
8
9     for(i=0; i<n; i++)
10         V[i]=0;
11
12     for(i=0; i<len; i++)
13         V[P[i]-1] += 1;
14
15     for(i=0; i<len; i++)
16     {
17         for (j=0; j<n; j++)
18         {
19             if (V[j]==0)
20             {
21                 V[j]=-1;
22                 T[q++]=j+1;
23                 T[q++]=P[i];
24                 V[P[i]-1]--;
25                 break;
26             }
27         }
28     }
29
30     j=0;
31     for(i=0; i<n; i++)
32     {
33         if(V[i]==0 && j==0)
34         {
35             T[q++]=i+1;
36             j++;
37         }
38         else if(V[i]==0 && j==1)
39         {
40             T[q++]=i+1;
41             break;
42         }
43     }
44
45     delete [] V;
46 }
47
```

```
48 void main(void)
49 {
50     int P[] = { 1, 2, 2 };
51     int len = sizeof(P) / sizeof(P[0]);
52
53     int* T = new int [2*(len+1)];
54     SequenceToSpanningTree(P, len, T);
55
56     for(int i=0; i<2*(len+1); i++)
57     {
58         printf(" %d",T[i]);
59         if((i+1)%2==0 && i<2*len)
60             printf(" - ");
61     }
62     printf("\n");
63
64     delete [] T;
65
66 }
```

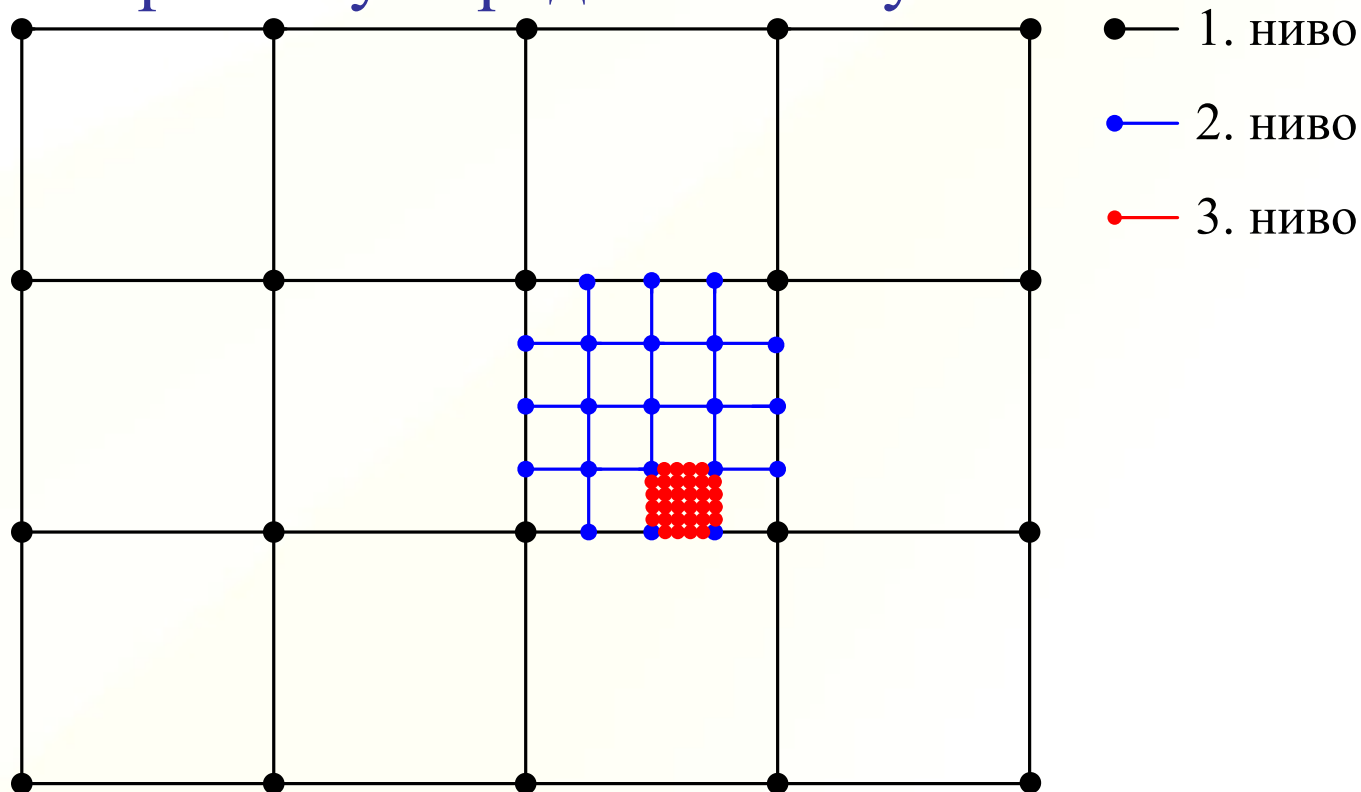
# Систематско претраживање: особине и употреба

- Уколико желимо да **докажемо** да смо нашли **глобални оптимум** (најбоље могуће решење) једини начин је да **систематски претражимо** читав оптимизациони простор у општем случају
- Једно решење (једна тачка оптимизационог простора) се једном и само једном проверава
- Континуалне променљиве (NLP):
  - систематско претраживање подразумева коначан корак за сваку димензију (тачност)
  - решење смо нашли са тачношћу која је пропорционална кораку
- Дискретне променљиве (SAT и TSP):
  - сва решења су проверена

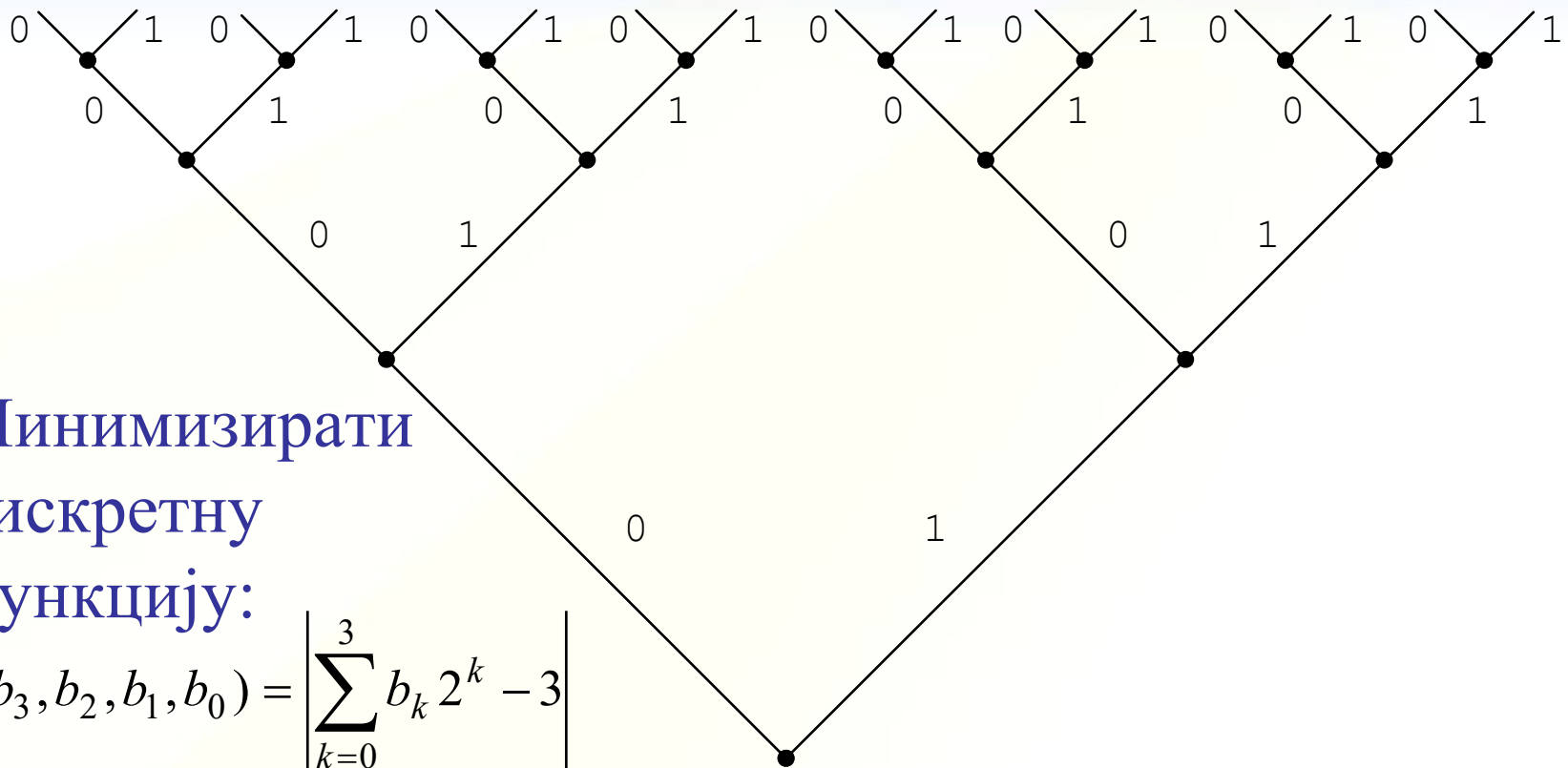


# Варијације: хијерархијско систематско претраживање

- Проценити део простора и претражити га са мањим кораком у наредном нивоу



# Варијације: гранање и одсецање (branch & cut backtracking)



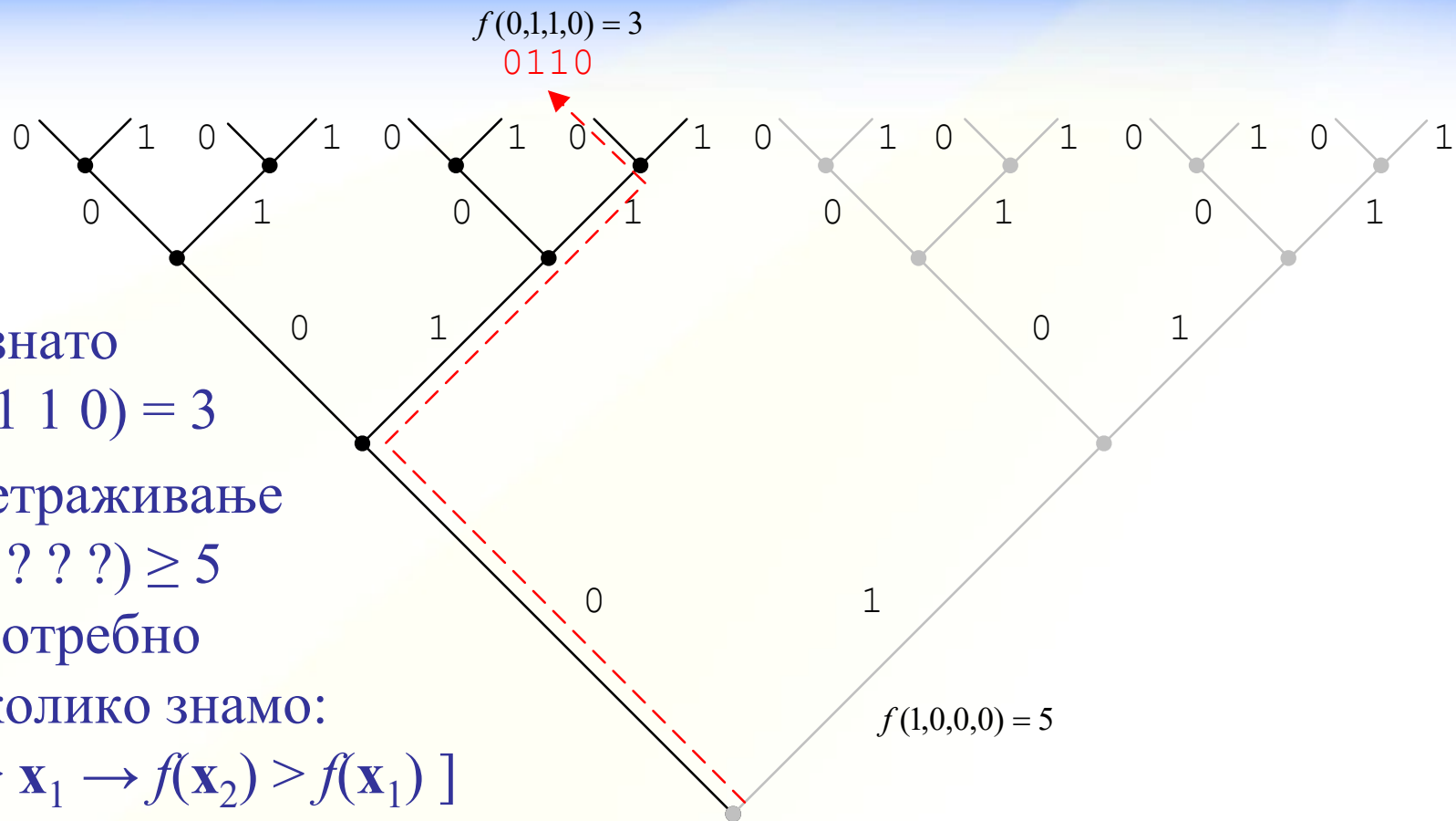
- Минимизирати  
дискретну  
функцију:

$$f(b_3, b_2, b_1, b_0) = \left| \sum_{k=0}^3 b_k 2^k - 3 \right|$$

$$b_k \in \{0,1\}$$

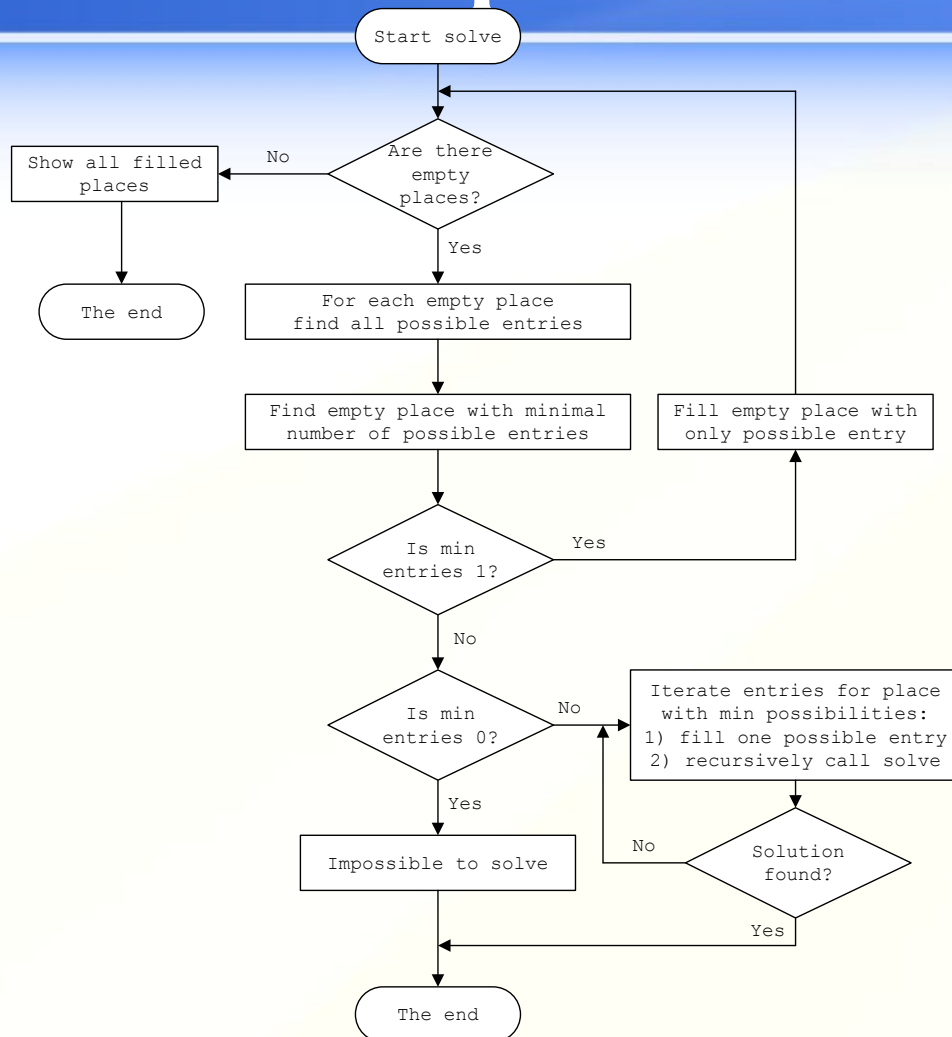
# Гранање и одсецање: прескочити део простора

- Познато  
 $f(0\ 1\ 1\ 0) = 3$
- Претраживање  
 $f(1\ ?\ ?\ ?) \geq 5$   
непотребно  
[ уколико знамо:  
 $\mathbf{x}_2 > \mathbf{x}_1 \rightarrow f(\mathbf{x}_2) > f(\mathbf{x}_1)$  ]
- Простор смањен 2 пута!



# SUDOKU:

## гранање и одсецање



|   |   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|---|
|   |   |   |   |   |   |   |   |   |
|   |   | 8 |   |   | 9 |   | 1 | 4 |
|   | 2 |   |   |   |   |   |   |   |
|   |   |   |   | 2 |   | 8 |   | 5 |
| 1 |   |   |   |   |   |   | 7 |   |
|   |   |   |   |   |   |   |   |   |
|   | 7 |   |   |   |   | 9 | 3 |   |
| 6 |   |   | 1 | 4 |   |   |   |   |
|   |   |   | 2 |   |   |   |   |   |

|   |   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|---|
| 5 | 6 | 9 | 8 | 7 | 2 | 3 | 1 | 4 |
| 3 | 1 | 8 | 4 | 6 | 9 | 2 | 5 | 7 |
| 7 | 2 | 4 | 3 | 1 | 5 | 6 | 8 | 9 |
| 9 | 4 | 3 | 7 | 2 | 1 | 8 | 6 | 5 |
| 1 | 5 | 7 | 9 | 8 | 6 | 4 | 2 | 3 |
| 2 | 8 | 6 | 5 | 3 | 4 | 7 | 9 | 1 |
| 4 | 7 | 1 | 6 | 5 | 8 | 9 | 3 | 2 |
| 6 | 9 | 2 | 1 | 4 | 3 | 5 | 7 | 8 |
| 8 | 3 | 5 | 2 | 9 | 7 | 1 | 4 | 6 |

[ 33] Solving time: 0.061 [sec]

# Други називи и особине систематског претраживања

- Други називи за систематско претраживање
  - Grid search
  - Brute-force search
  - Parameter sweep
  - Exhaustive search
  - Enumeration
  - Generate & test
- Најбољи могући приступ  
уколико можемо да сачекамо да се претрага заврши
  - оптимизациони простор је мали у односу на расположиве ресурсе
- Уколико претрага траје недопустиво дуго није од користи
  - оптимизациони простор је велики у односу на рачунарске ресурсе
- Погодан за извршавање у  
паралели на рачунарима са више процесора (језгара)

# Случајно претраживање

- На случајан начин генеришу се тачке у којима се рачуна функција грешке
- Најчешће се користи генератор са униформном расподелом
- Неефикасан начин оптимизације јер су претходно и наредно израчунавање оптимизационе функције независни (иста тачка може да се испита више пута)
- Добар начин за грубу претрагу простора
- Сложеност  $O(N)$   
 $N$  број одбирака оптимизационог простора

# Генератори случајних бројева

- Рачунарско генерисање случајних бројева је нетривијалан задатак
- Генератори који постоје у библиотекама пролазе строге тестове случајности само делимично!
- Доступне функције:
  - C/C++: `rand()`, `srand()`, `boost/random...`
  - MATLAB: `rand()`, `randn()`, `randi()`...
  - Python: `random.randint`, `random.uniform...`

# Једноставна рутина за генерисање целих случајних бројева

- Опсег  $a \leq \xi \leq b$
- Ограничење:  
RAND\_MAX
- Ограничење зависи од компајлера  
(VS2017 RAND\_MAX = 32768)
- Уколико је потребан већи опсег
  - генерисати произвољан низ бита {0, 1} и претворити га у цео број
  - генерисати два цела броја (или више), надовезати бите и интерпретирати нови низ као нови (већи) цео број

```
int random_int(int a, int b)
{
    int res;
    res = a + rand() % (b + 1 - a);
    return res;
}
```



# Једноставна рутина за генерисање реалних случајних бројева

- Опсег  $a \leq \xi \leq b$
- Разлика између два суседна броја који се генеришу (грануларност) је  $1/\text{RAND\_MAX}$
- Уколико је потребна већа грануларност:
  - Опсег  $[a \ b]$  поделити на више подопсега и сабрати (суперице) бројеве из подопсега

```
double random_float(double a, double b)
{
    double res;
    res = a + (double)rand() / (double)(RAND_MAX / (b-a));
    return res;
}
```

# C++11 <random>

```
#include <random>
#include <iostream>

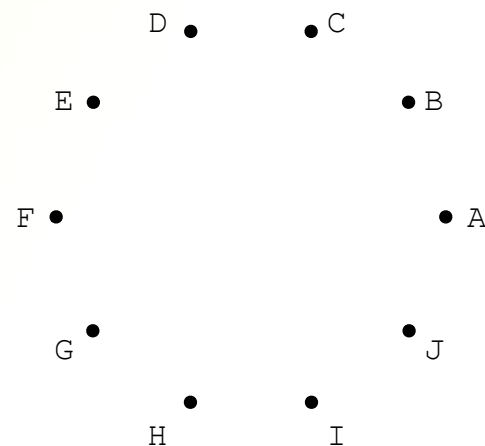
int main() {
    std::random_device rd;
    std::mt19937 mt(rd());

    std::uniform_real_distribution<double> random_float(1.0, 10.0);
    for (int i = 0; i<16; ++i)
        std::cout << random_float(mt) << "\n";

    std::uniform_int_distribution<int> random_int(1, 5);
    for (int i = 0; i<16; ++i)
        std::cout << random_int(mt) << "\n";
}
```

# Задатак

- Десет градова потребно је повезати мрежом за дистрибуцију електричне енергије
  - Сви градови морају бити повезани
  - При повезивању није дозвољено да постоје затворени путеви (петље)
  - Цена повезивања сваког пара градова дата је у табели (табела је симетрична)
  - Свако гранање од једног града ка **четири или више** градова повећава цену за  $(g - 3) \cdot 250$  јединица цене
  - Пронаћи оптималан начин повезивања градова коришћењем **потпуне претраге** (минимизирати цену повезивања)



# Цена повезивања

(табела је симетрична)

|   | A   | B   | C   | D   | E   | F   | G   | H   | I   | J   |
|---|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| A | 0   | 374 | 101 | 223 | 108 | 178 | 252 | 285 | 240 | 356 |
| B | 374 | 0   | 27  | 166 | 433 | 199 | 135 | 95  | 136 | 17  |
| C | 101 | 27  | 0   | 41  | 52  | 821 | 180 | 201 | 131 | 247 |
| D | 223 | 166 | 41  | 0   | 430 | 47  | 52  | 84  | 40  | 155 |
| E | 108 | 433 | 52  | 430 | 0   | 453 | 478 | 344 | 389 | 423 |
| F | 178 | 199 | 821 | 47  | 453 | 0   | 91  | 37  | 64  | 181 |
| G | 252 | 135 | 180 | 52  | 478 | 91  | 0   | 25  | 83  | 117 |
| H | 285 | 95  | 201 | 84  | 344 | 37  | 25  | 0   | 51  | 42  |
| I | 240 | 136 | 131 | 40  | 389 | 64  | 83  | 51  | 0   | 118 |
| J | 356 | 17  | 247 | 155 | 423 | 181 | 117 | 42  | 118 | 0   |

# Поступак решавања

- Проблем се може решити претраживањем по комплетном графу  $K_{10}$
- Потребно је претражити сва стабла овог графа
- Максималан број претрага је  $n^{n-2} = 10^8$
- Написати код за израчунавање оптимизационе функције:  
$$f(\mathbf{x}) = \sum_{k=1}^9 c_k + \sum_{n=1}^{10} \begin{cases} 0, & g(n) < 4 \\ 250(g(n) - 3), & g(n) \geq 4 \end{cases}$$

$c_k$  је “цена” гране са редним бројем  $k$   
(тј. вредност из табеле за полазни и крајњи чвор  $k$ -те гране)  
плус “пенал” за сваки чвор стабла у којем постоји гранање четири или више грана ( $g(n)$  је број грана које се стичу у чвору  $n$ )
- Написати код који генерише све секвенце које се могу једнозначно пресликати у стабла графа (варијације са понављањем)
- Искористити рутину за пресликавање секвенце у стабло графа
- Написати код који извршава потпуну претрагу
- Пронаћи и записати у ASCII фајл
  - минималну цену мреже за повезивање
  - путању за повезивање (стабло графа)  
решење треба да садржи девет грана (стабло) графа  
свака грана графа је дефинисана са два чвора  $X$  и  $Y$  (чворови су означени са A, B, C, D...)  
гране записати као два чвора раздвојена размаком (A B)  
стабло записати као низ грана раздвојених цртицом (A B – C D – A D ...)