Дискретна математика

доц. д-р Тодорка Глушкова,
Катедра „Компютърни технологии“, ФМИ

3

Математически основи

Съдържание:

- Теория на графите
- Дървета

Въведение

- През 19 век графите се използват при построяването на схеми на електрични вериги и сложни молекули.
- Постепенно тази теория навлиза в психологията, статистическата механика, теорията на вероятностите, математическата логика, генетиката, социологията и т.н.
- Днес тя е основа в развитието на всички тези дисциплини и най-вече на информатиката.

Основни дефиниции

- <u>Дефиниция:</u> ***Граф*** G=(V,E), където

  \- V$\neq \emptyset$ множество на възлите;

  \- E – множество на връзките между възлите, наречени ребра. Ребрата са неподредена двойка от по два възела e={u, v}. Казваме, че

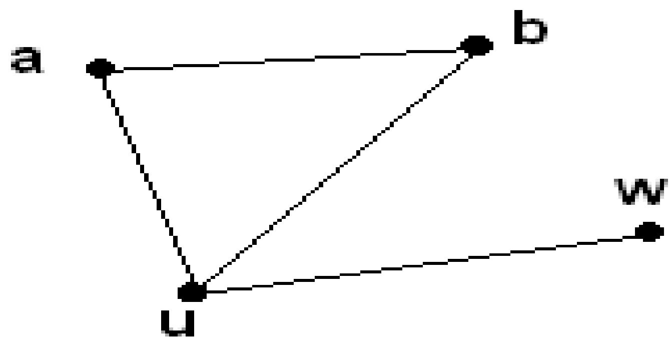
  1) "e свързва u и v"

  2)" u , v-крайни точки на e " или

  3)" u , v-са съседни възли "

Основни дефиниции

- *Пример: V={a,b,u,w}, E={au,uw,bu,ab}*



- Когато ребрата са ненасочени, т.е. няма значение кой е първия и кой – втория възел, казваме че графите са ***прости или ненасочени.***

Насочен граф

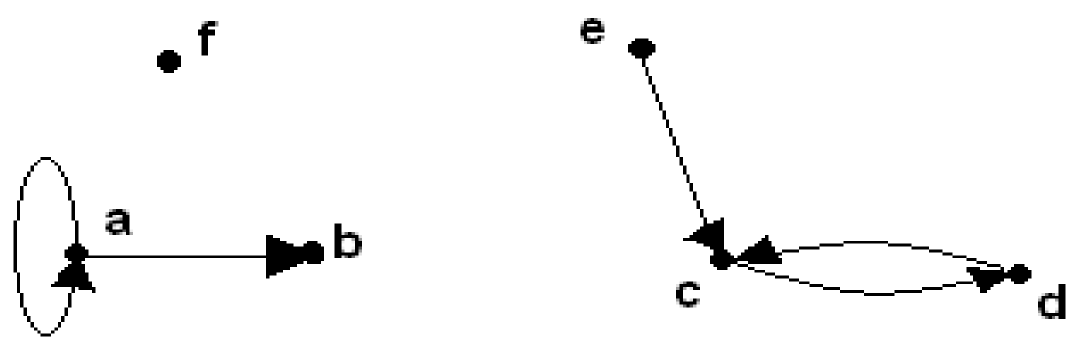
- Дефиниция: ***G***=(V,E) е ***насочен граф***, ако:
  - G е граф;
  - E – множество от наредени двойки върху V.

  Ребрата в насочения граф се наричат “клони”.
- Ако e=(u,v), тогава:

  1) “е свързва u към v”;
  2) ” u -опашка, v-глава ”;
  3)” е излиза от u и влиза във v”

Насочен граф

- *<u>Пример:</u> V = {a,b,c,d,e,f}, E={aa,ab,cd,dc,ec}*



- Когато u=v (aa), получаваме ***цикъл***.

Видове графи

- <u>Дефиниция:</u>G=(V,E,i) е ***псевдограф***, ако:

  - $V \neq \emptyset$ е крайно множество на възлите;

  - E е крайно множество на ребрата;

  - $i: E \to \{P \subseteq V, |P| = 1$ или $2\}$ е инсидентна функция.
- Ако **r** е ребро **i(r)** са крайните му точки.
- Ако |**i(r)|** =1, то реброто свързва една точка със себе си (цикъл).
- Ако за **r** и **q**, **i(r)=i(q),** казваме че **r** и **q** са паралелни ребра.

Видове графи

- <u>Дефниция: </u>Ако графът G има изображение в равнината, при което ребрата се пресичат само във върховете, казваме, че той е ***планарен*** (равнинен), в противен случай е неравнинен или ***пространствен.***

Движение през един граф

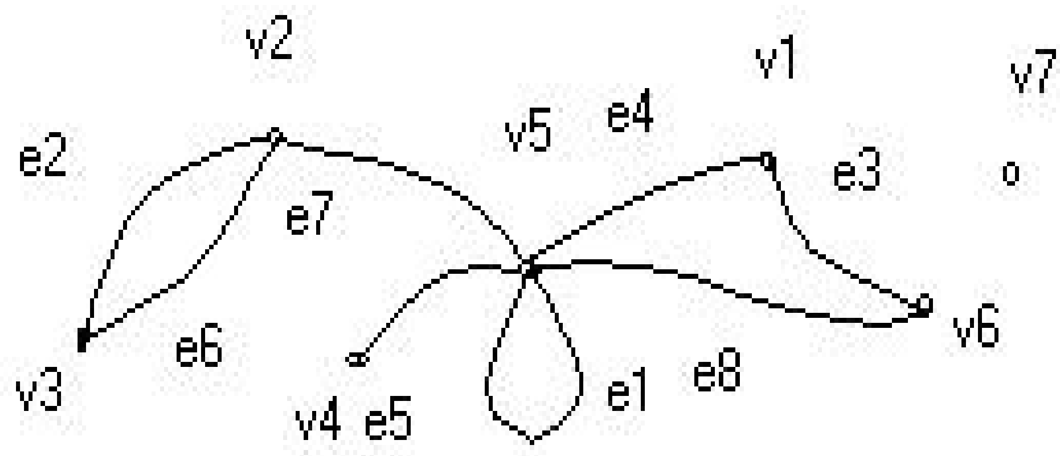
- Дефиниция: Нека G=(V,E) е граф. Тогава един ***маршрут (walk)*** W с дължина n>0 в G е редицата:

  $v_0,e_1,v_1,e_2,v_2,\ldots,v_{n-1},e_n,v_n$, така че $v_k \in V$, а $e_k \in E$ за всяко k=1..n, като e_k свързва v_{k-1} и v_k. W свързва v_0 и v_n от v_0 към v_n .
- Всяко ребро може да се разглежда като маршрут с дължина 1.
- Когато за W са изпълнени и други допълнителни условия, то получава различни наименования:
- Ако $v_0 = v_n$ и $n \geq 1$,то казваме че маршрута W е ***затворен.*** В противен случай е незатворен.

Пример

$m = [v_2, e_7, v_5, e_1, v_5, e_8, v_6, e_3, v_1, e_4, v_5, e_5, v_4]$ е незатворен маршрут, съединяващ v_2 и v_4 с дължина 6.

Заменяйки e_5 с e_7 в m, получаваме затворен маршрут с дължина 6.



Движение през един граф

1. Ако всички ребра в W са различни- се нарича ***верига (trail), (***ако не е затворен)***.***

2. Ако всички възли му са различни се нарича ***елементарен маршрут*** или ***път.***

3. Ако W е затворен и е верига като всичките му възли са различни, казваме че W е ***цикъл.***

4. Забелязваме, че в геометрична реализация елементарната верига образува проста незатворена линия, а елементарния цикъл - проста затворена линия.

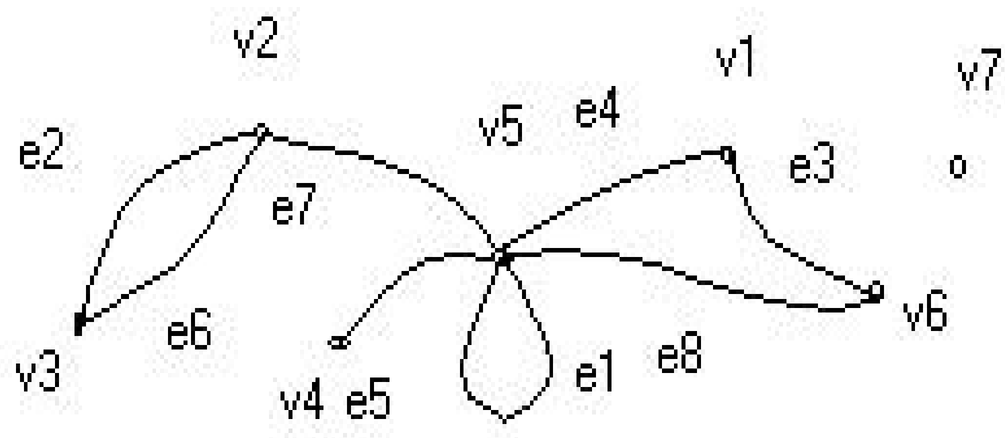
Задача

Определете всички върхове на графа по-горе, до които съществува верига от v_5:

а) с дължина 1

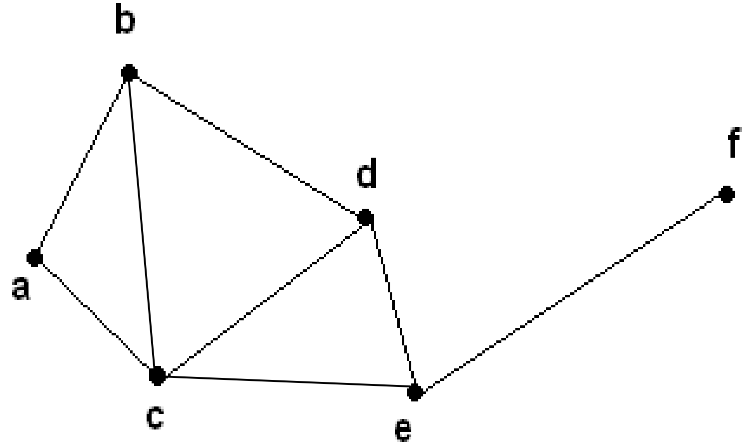
б) с дължина 2

в) с дължина 4



Примери

- a,b,e,d не е маршрут, защото be не е ребро
- b,d,e,d - W е маршрут с дължина 3 от b към d, но не е верига.
- f,e,f – затворен W с дължина 2, но не е верига.
- a,b,d,c – път с дължина 3.
- a,b,c,e,d,c,a –затворена верига с дължина 5, но не е цикъл
- b,c,d,b – цикъл с дължина 3.



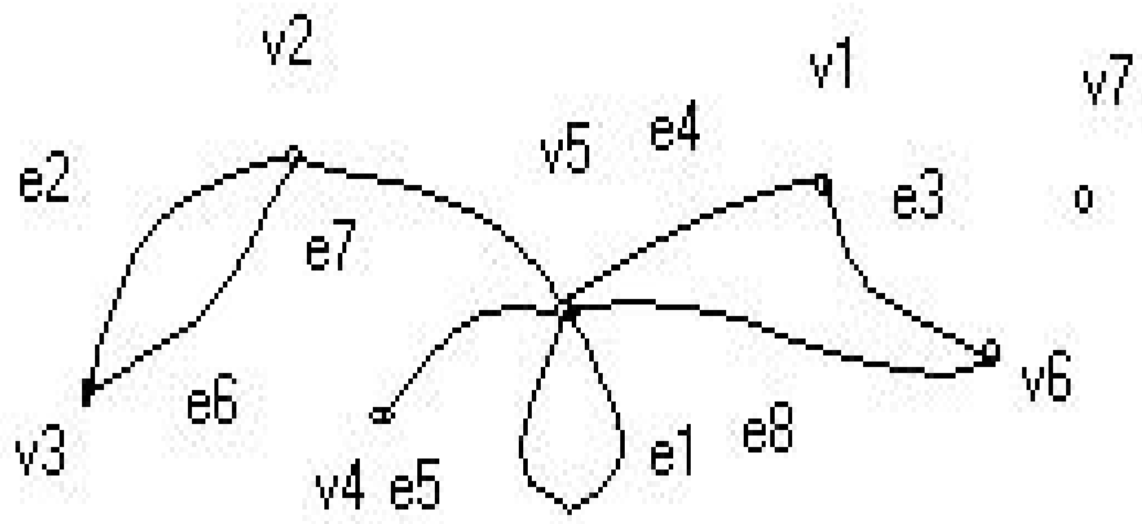
Движение през един граф

- Да отбележем, че ако съществува в *G* маршрут с дължина *n* между върховете *x* и *y*, то той може да бъде продължена до нов маршрут с дължина *n*+2

- Тогава между два различни върха *x* и *y* на граф *G* или не съществува никакъв маршрут, или съществуват безброй много. Интересуваме се от маршрута с най-малка дължина. Такава верига винаги съществува. Дължината на ***минималната верига*** означаваме с r(*x*, *y*).

Задача

- Определете r(v_1, v_3) за графа от по-горния пример. Колко вериги с тази дължина



Твърдения

- Ако между два различни върха съществува верига, то числото r е еднозначно определено, но може няколко вериги между тези върхове да са с дължина r.
- **Теорема.** Ако в графа съществува верига между върховете *x* и *y*, то съществува и поне една елементарна верига между *x* и *y*.
- **<u>Теорема:</u>** Ако G съдържа един маршрут от **u** към **v**, то G съдържа път от **u** към **v**.
- **<u>Теорема:</u>** Ако G съдържа една затворена верига, започваща и завършваща във **v**, тогава G съдържа един цикъл с начало и край във **v**.

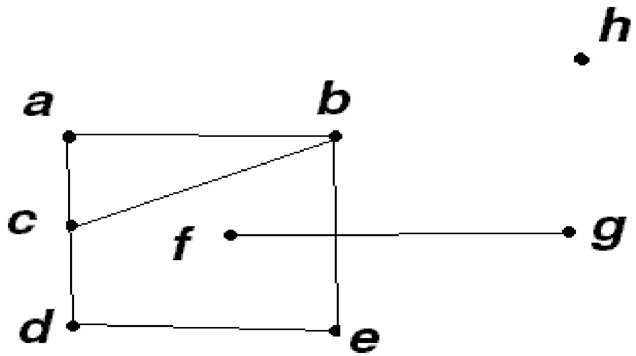
Свързани графи

- <u>Свързани графи</u> – За едно приложение в комуникационната мрежа, например, е важно да се знае дали всеки обект от мрежата има връзка с всеки друг, било директно, било през някои междинни възли.
- <u>Дефиниция:</u> G е ***свързан граф***, ако за всеки два възела съществува маршрут от единия към втория.
- **Компонент на G** е един максимален свързан подграф на G, (т.е. един свързан подграф на G), който не е подграф на никой друг свързан подграф на G.

Пример

Графът не е свързан и има 3 компонента:

- *h;*
- *f, g и реброто, което ги свързва;*
- *петте останали възли и ребрата, които ги свързват*



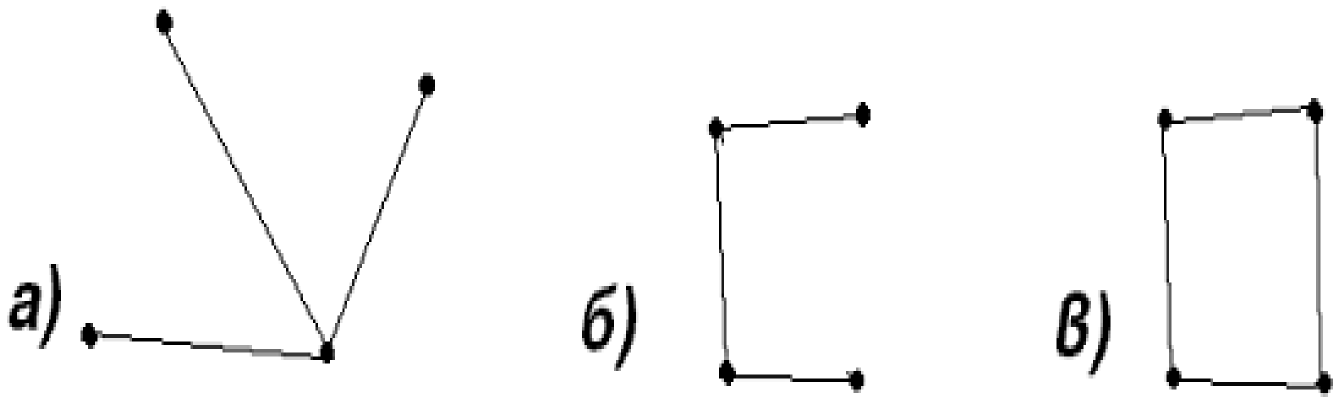
Ойлерови цикли (или турове)

- Дефиниция: ***Цикъл(или тур) на Ойлер*** в G е затворена верига, която съдържа всяко ребро само веднъж.
- **Теорема:** Един свързан граф G с най-малко едно ребро има един тур на Ойлер, тогава и само тогава, когато степента на всеки възел в G е четна, т.е. всеки възел участвува четен брой пъти.
- Забележка:Ойлеровите цикли или турове решават прословутата задача за Кьонингсбергските мостове, според която всеки мост трябва да се премине само веднъж.
- Всеки граф, който притежава Ойлеров цикъл се нарича ***Ойлеров граф***.

Цикли на Хамилтън

- **Дефиниция:** Нека G е граф. ***Цикъл на Хамилтън*** в G е цикъл, който съдържа всеки възел на графа. Граф, притежаващ Хамилтънов цикъл се нарича ***Хамилтънов граф.***
- С други думи - Хамилтънов цикъл в G е свързан подграф на G, съдържащ всичките му възли, в който всеки възел има степен 2.
- Всяка проста верига, съдържаща всички възли на един граф G е ***Хамилтънова верига***. Ако един граф е Хамилтънова верига, но не е Хамилтънов цикъл, то той е ***полухамилтънов граф***.

Пример



а) нито Хамилтънов, нито полухамилтънов.

б) полухамилтънов

в)Хамилтънов граф.

Забележка

- Типичен пример за Хамилтънов граф е задачата: “Може ли да се обходят всички полета на шахматната дъска с шахматен кон, като на всяко поле се стъпи само веднъж?”
- Трудността на задачата не е в намирането на желания път на коня, а в намирането на всички възможни пътища с разглежданото свойство и определянето на техния брой.
- Крайчик е доказал, че задачата има повече от 30 млн. решения.

Представяне на графи и изоморфизъм

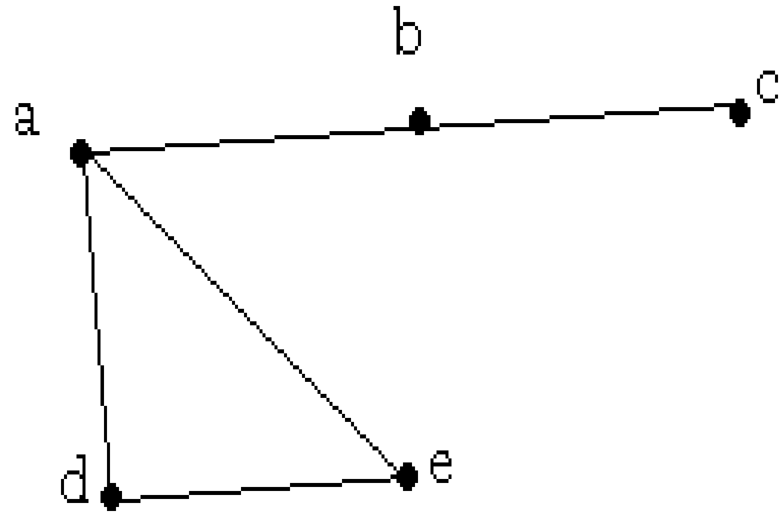
- Ние дефинирахме графите като абстрактен математически обект. Въпросът ни е как да ги представим така, че хората и компютрите да могат да работят с тях?

<u>Матрица за съседство.</u>

- Нека G(V,E) е граф. Тогава матрицата за съседство за G Ag е матрица с размерност (n x n), така че:
- aij = 1,ако $v_i v_j \in E$ или
- aij = 0,ako vivj $\notin$E,
- т.е. 1, ако има ребро между възлите v_i и v_j и 0, ако няма такова ребро.

Пример

$$\begin{bmatrix} 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 \end{bmatrix}$$

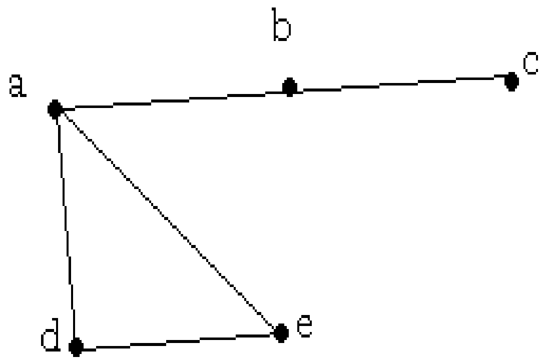


Забележка: Матрицата е симетрична, т.е. съвпада с транспонираната й матрица, тъй като графът е ненасочен.

Представяне на графи и изоморфизъм

<u>Списъци за съседство.</u>

- Това са списъци на възлите, с които всеки възел е свързан.
- Например за горния граф:
  - a: b,d,e;
  - b: a,c;
  - d: a,e;
  - e: a,d



Представяне на графи и изоморфизъм

<u>Изоморфизъм</u>

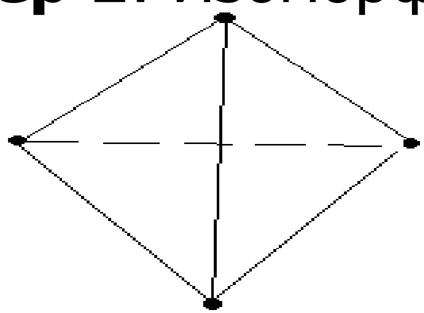
- Да отбележим, че един и същи граф можем да изобразяваме различно.
- Ребрата може да са отсечки или дъги;
- върховете може да са разположени произволно върху равнината или пространството.
- Понякога чрез разместване на върховете един граф може да бъде трансформиран в друг.

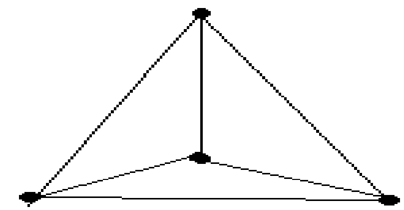
Представяне на графи и изоморфизъм

- Дефиниция: Нека G =(V,E) и H=(W,F) са графи. Казваме, че са ***изоморфни***, ако съществува биекция φ: V $\rightarrow$W, така че за всяко u , v от V:(uv $\in$E$\leftrightarrow\varphi$(u)φ(v) $\in$ F). Тази биекция φ наричаме изоморфизъм между двата графа.
- Ако два графа са изоморфни, свойствата на единия могат да се пренесат и върху другия, което е мощен метод за обработка на графи.

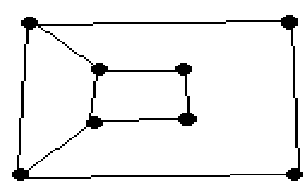
Примери

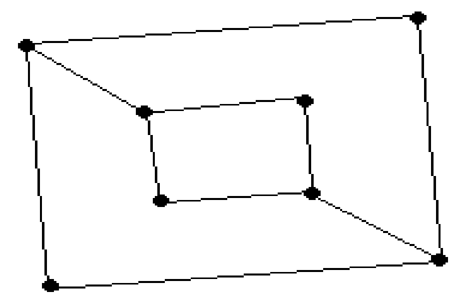
Пример 1: Изоморфни графи





Пример 2: Не изоморфни графи

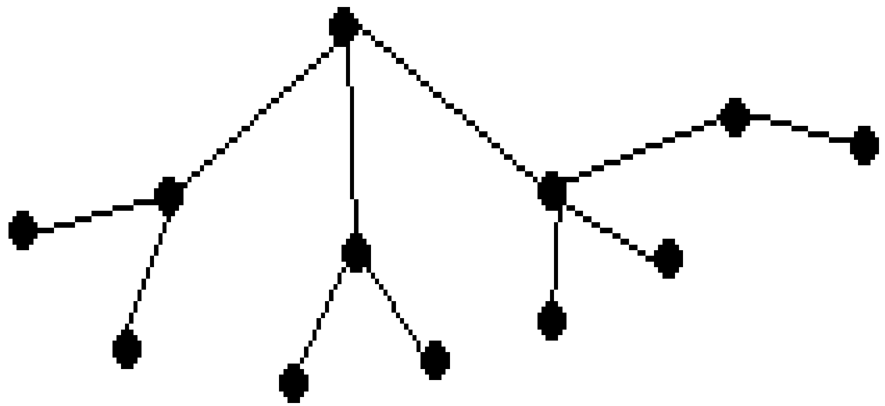




Дърво. Дефиниции

- <u>**Дефиниция:**</u> ***Дървото*** е свързан, ориентиран граф, който не съдържа затворени вериги. Обикновено се бележи с Т.
- Дървото има един специален възел - ***корен***.
- Дърво с корен бележим - (Т,r).
- Възли, от които не излиза ребро се наричат ***листа***.
- Останалите - ***вътрешни възли***.
- Дължината на пътя от корена до най-далечното листо се нарича ***височина*** на дървото.

Пример



Дървото има:

- един корен;
- 7 листа;
- височина - 3

Дефиниции и свойства

- Забележка: Дървото има само един път от всяко листо до корена. Всеки граф с такова свойство е дърво.
- Дефиниция: Две дървета са ***изоморфни*** тогава и само тогава, когато съществува биекция между множествата на възлите им, която запазва съседите, не съседите и възела.
- **Лема:** Едно дърво с повече от един възел, съдържа най-малко две листа.

Твърдения

<u>Теорема:</u> Нека G е граф с n- възела. Тогава следните твърдения са еквивалентни:

а) G е дърво, т.е. G е свързан граф и няма затворени вериги.

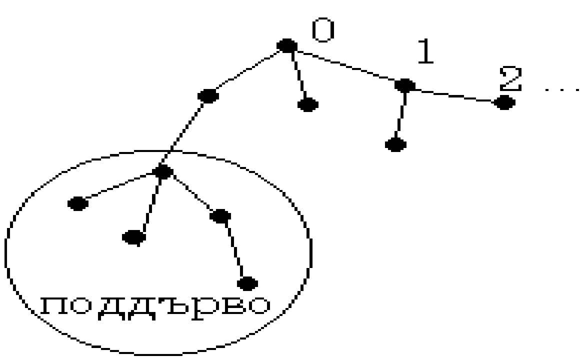
б) G има точно n-1-ребра и няма затворени вериги.

в) G е свързан граф, но ако някое ребро се отстрани от него, полученият граф няма да бъде свързан.

г) Между всеки два възела в G съществува единствен път между тях.

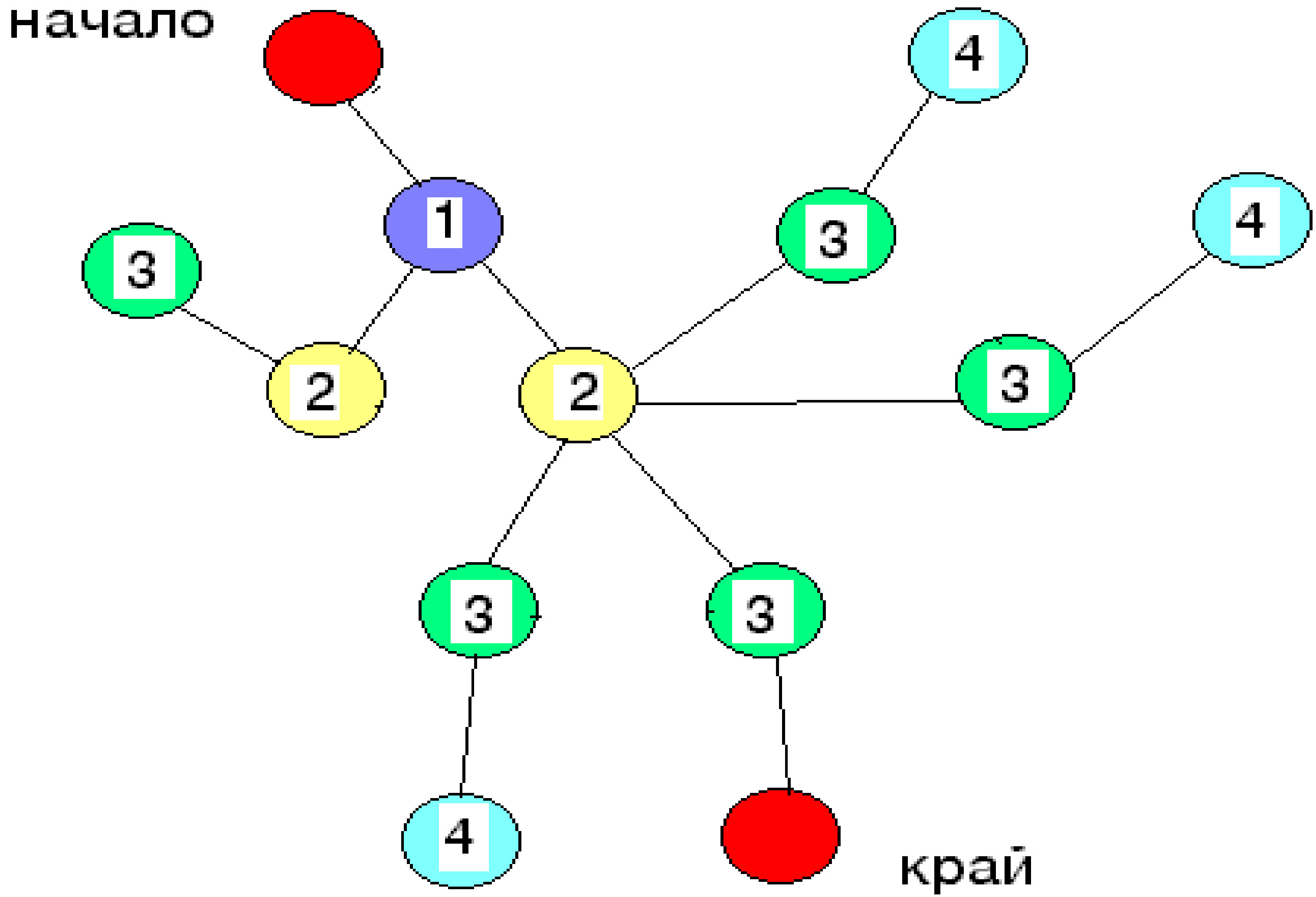
Дефиниции и свойства

- Забележка: От Теоремата следва, че съществува уникален път от корена до всеки друг възел.
- Дефиниция: **Ниво** на един възел е дължината на пътя от корена до възела. Коренът има ниво 0. Съседните му- ниво 1 и т.н.
- Дефиниция: **Поддърво** на T е дървото T1$\subseteq$T.



Дефиниции и свойства

- Дефиниция: Всяко дърво с краен брой възли е ***крайно***.
- Забележка: Когато търсим елемент в крайно поддърво го обхождаме в симулативен режим като използваме метода на препредаване на маркерите на ребрата. Коренът означаваме с 0, наследниците му - с 1, техните наследници - с 2 и т.н. Щом открием търсения възел, изграждаме път обратно като проследяваме маркерите на ребрата в низходящ ред обратно към корена.


начало
1
2
2
3
3
3
3
3
4
4
4
край

Дефиниции и свойства

Можем да дадем рекурсивна дефиниция на дърво с корен:

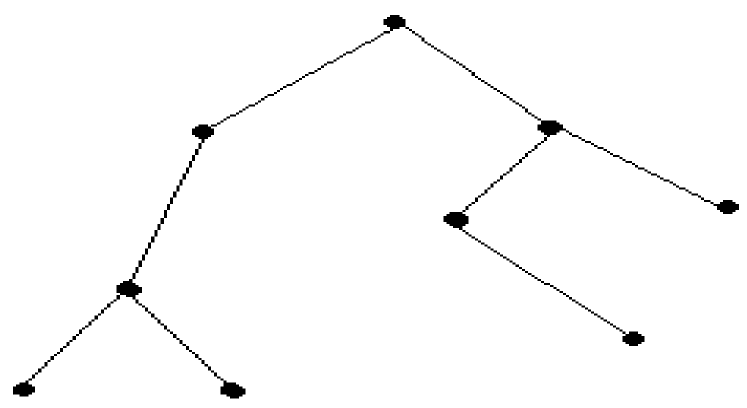
- Базов случай: Всеки отделен възел е дърво с корен- самият възелът.
- Рекурсивен случай: Ако $k \geq 1$ и $T1, T2, \ldots Tк$, $Ti \cap Tj = \varnothing$ са дървета с корени съответно $v1, v2, \ldots vk$, тогава следния граф е също дърво с корен: един нов възел v е негов корен, съвместно с $T1, T2, \ldots Tк$ с възли vi - съседи на v за всяко $i=1..k$.

Подредени дървета. Двоични дървета

- Нека T е дърво с корен r. Можем да го разглеждаме като насочен граф с ребра насочени надолу. Ако uv е директно ребро на T, то казваме, че v е наследник на u, а u-предшественик на v.
- Често се налага да дадем някакъв ред на наследниците за всеки възел.
- **Подредено дърво** наричаме дърво с корен с допълнителна структура за линеен ред на наследниците за всеки вътрешен възел.
- Например: Отляво надясно.

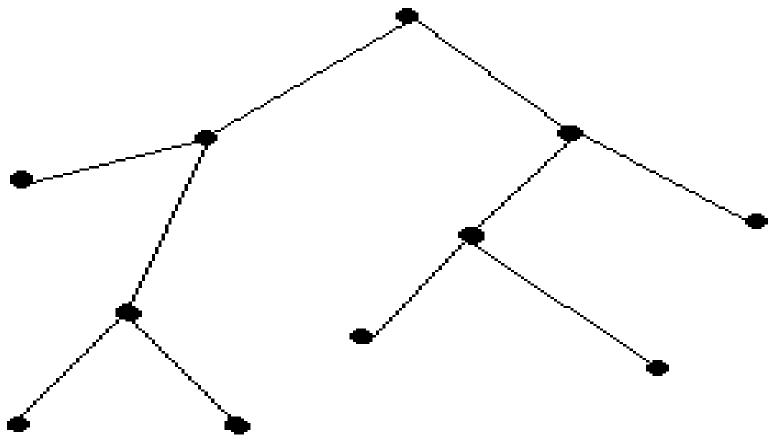
Двоично дърво

- **Двоично** или **бинарно** дърво е дърво с корен, като за всеки възел съществува най-много един ляв наследник и/или един десен наследник.
- <u>Пример:</u>



Пълно двоично дърво

- <u>Дефиниция:</u> Едно двоично дърво и ***пълно***, ако за всеки вътрешен възел съществуват и двата му наследника.
- <u>*Пример:*</u>



Пълно двоично дърво

Теорема: Нека B е пълно двоично дърво с e-листа и i-вътрешни възли. Тогава e= i+1.

Доказателство: Ще използваме индукция върху рекурсивни дефиниции:

- i=0, e=1, т.е. e=i+1;
- i=1, e=2, т.е. e=i+1;
- Нека i=k, следователно e=к+1. Тогава за i=k+1$\Rightarrow$ e=k+2 $\Rightarrow$**e=i+1**.

Spanning дървета

Дефиниция: ***Spanning*** – дърво е свързан граф G[1], който е подмножество на графа, който съдържа всички възли на G и е дърво.

- **Теорема:** Всеки свързан граф съдържа едно spanning дърво.
- Ще разгледаме два алгоритъма, целта на които е систематично да "посетят" всички възли на графа.
- Spanning дърветата се получават като допълнителен резултат.

Търсене първо в дълбочина

- Рекурсивен алгоритъм за посещаване на всичките възли на един свързан граф G. След като посетим един възел, ние го маркираме (за да не го разглеждаме повторно). Успоредно с това ние генерираме едно spanning дърво T.

Procedure depth_first_search(G: свързан граф)

{Нека възлите на G са номерирани от 1 до n visited(i)=T, когато възел i е посетен, като T е spanning дърво}

visited(1)←true {започваме от възел 1}

for i←2 to n do

visited(i) ←false {не са посетени други възли}

T←({1}, ∅) {Първоначално дървото съдържа възел 1 и няма ребра}

(T, visited) ←DFS(G,T,visited,1)

return (T)

Procedure DFS(G:граф, T: дърво, visited: масив, i: възел от G)

{T е подграф на G; visited са възлите от G, които са вече посетени. Тази рекурсивна процедура се извиква във възел i}

for j$\leftarrow$1 to n do

if (j е съседен на i)$\wedge$not visited(j)

begin

{j е нов непосетен съсед}

visited(j) $\leftarrow$true

add възел j и ребро ij към T

(T, visited) $\leftarrow$DFS(G,T,visited,j)

end;

return (T,visited)

Търсене първо в ширина

Procedure **width_first_search**(G: свързан граф)
{предполагаме, че възлите на G са номерирани 1,2,… n
visited(i)=T ⇔ възел i е вече посетен, т.е. T е spanning дърво; L е списъкът от посетени, но все още необработени възли.}
visited(1) ← true {започваме от възел 1}
for i←2 to n do
visited(i) ← false {другите възли не са посетени}
T←({1},∅) {Spanning-дървото първоначално съдържа възел 1 и несъседни ребра}
L←(1) {възел 1 очаква обработка}
while L<>empty do
begin
i← първия елемент на L
L←L с отстранен i

Търсене първо в ширина

for j←1 to n do
 if (j е съседен на i) ∧not visited(j)
then
 begin *{j е нов, но непосетен съсед}*
visited(j) ←true
add((възел j и ребро ij) към T)
add(възел j към края на L)
end;
end;
return(T)

Генериране на списък от възлите на едно дърво

Ще разгледаме основата(рамката) на много важни приложения – особено в конструирането на компилатори и езици за програмиране.

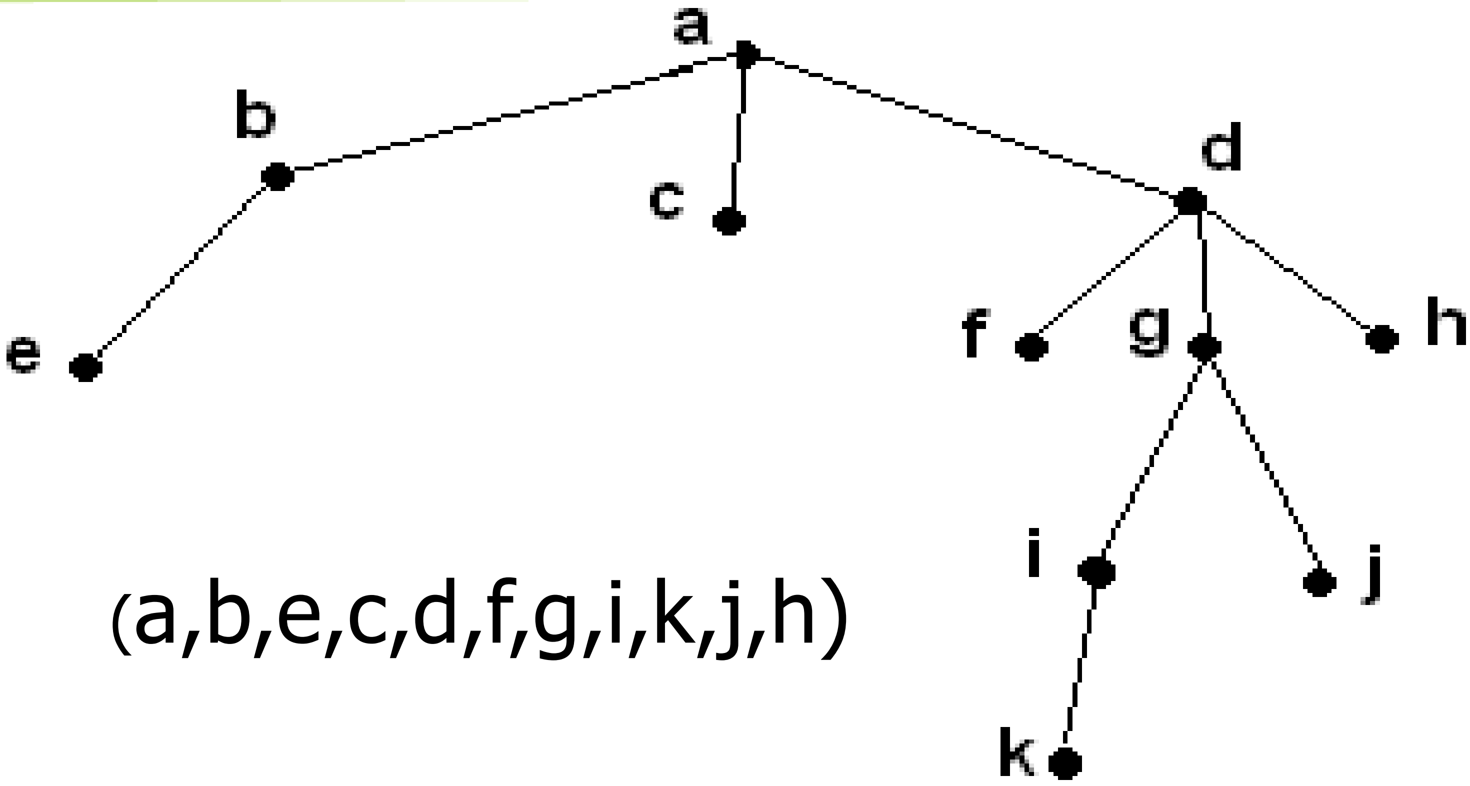
- **Проблем:** Да генерираме систематизиран списък на възлите на едно дърво.
- Съществуват различни начини за решаването на този проблем:

Преордер(от горе - надолу)

- Дефиниция:Нека T е подредено дърво с корен r. **Преордерът** на възлите на T се задава чрез следните условия:

1. Ако T съдържа точно един възел r, тогава преордер е r.
2. В противен случай преордер е r, следван от преордер на възлите в непосредствените поддървета на T.

Преордер. Пример



(a,b,e,c,d,f,g,i,k,j,h)

Преордер.

Procedure Preorder (r: корен на подреденото поддърво)

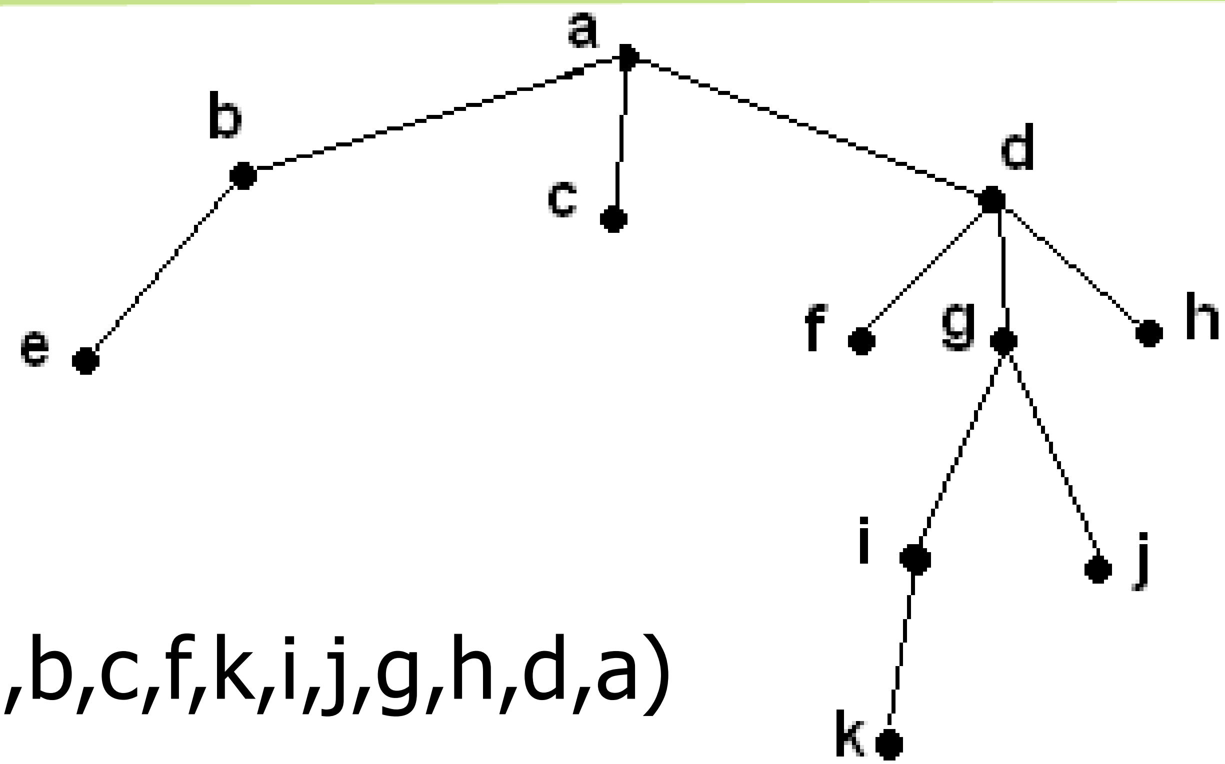
```
process  r
for всяко дете V на r поред do
call Preorder(V)
Return;
```

Постордер(postorder)

- В много приложения на дърветата с корен се налага да работим отдолу нагоре.
- Дефиниция: Нека T е подредено дърво с корен r. ***Постордер*** на възлите на T е даден чрез следните условия:

1. Ако T съдържа точно един възел r, тогава постордер е r.
2. В противен случай, постордерът е постордер на възлите в непосредствените поддървета на T поред, следван от r.

Постордер. Пример



(e,b,c,f,k,i,j,g,h,d,a)

Постордер

Procedure Postorder (r: корен на подреденото поддърво)

for всяко дете V на r поред do

call Postorder(V)

process r

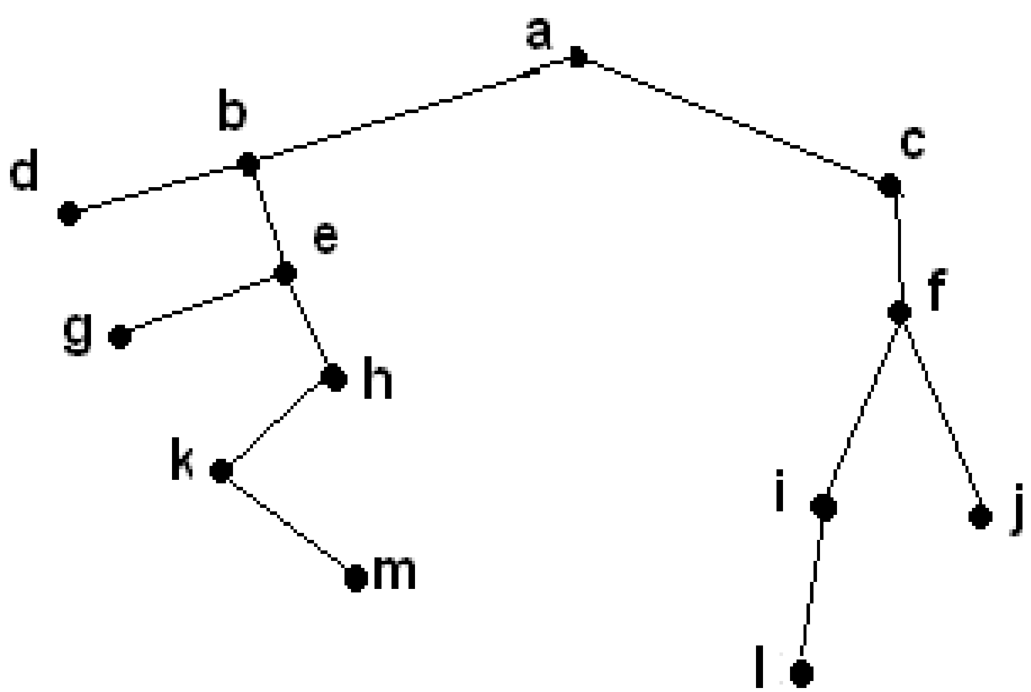
Return;

Инордер (inorder)

Дефиниция: нека Т е подредено бинарно дърво с корен r. Инордер на възлите на Т е даден чрез следните условия:

1. Ако Т съдържа точно един възел r- тогава инордер е r.
2. В противен случай, инордер е инордерът на възлите в лявото поддърво на Т(ако съществува), следван от r, следван от инордерът на възлите в дясното поддърво(ако съществува).

Инордер. Пример

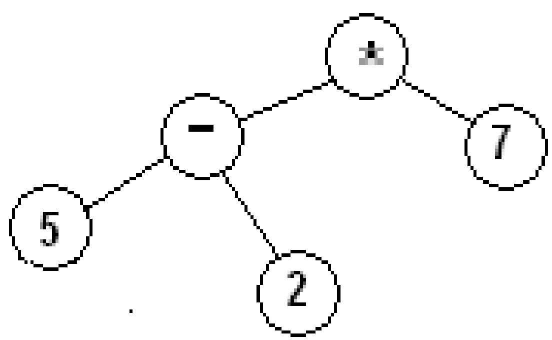


(d,b,g,e,k,m,h,a,c,l,i,f,j)

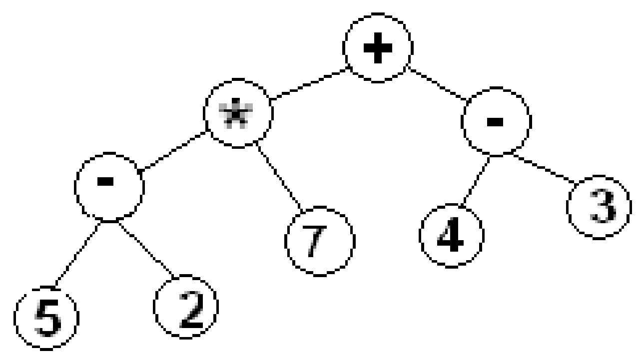
Expression дървета

- Можем да прилагаме дърветата при изчисляване на математически изрази. Във възлите могат да се поставят числа, имена на променливи, аритметични операции и други.

а) ((5-2)*7)

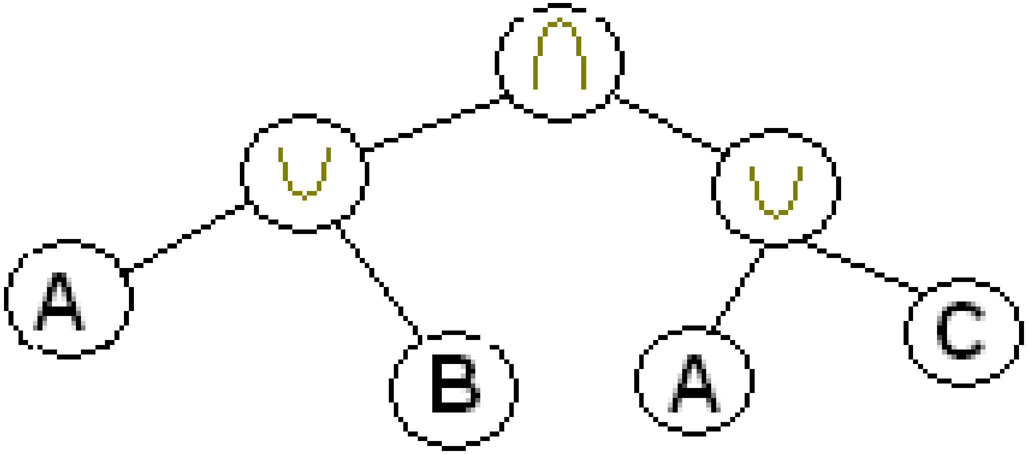


б) (5-2)*7 + (4 - 3)



Expression дървета

в) $(A \cup B) \cap (A \cup C)$



И-ИЛИ дървета

- Интересно е прилагането на дърветата и графите при търсене в пространството на състояния. В определени случаи е целесъобразно да декомпозираме трудни за достигане цели в една или повече по-малки цели. Всяка подцел от своя страна може да бъде отново декомпозирана в нови подцели на по-ниско ниво и т.н.
- Декомпозиране на проблема е итерационен процес от последователен избор на алтернативи и съответно разлаганена проблемите на подпроблеми. При всяка итерационна стъпка трябва да решаваме една алтернатива (ИЛИ-свързаност) или да решаваме последователно всички подпроблеми (И-свързаност), като отделните подпроблеми могат да бъдат взаимно зависими.

И-ИЛИ дървета

Една адекватна форма за представяне на модела са И-ИЛИ-дървета (И-ИЛИ-графи). Представянето на декомпозирането на проблема като И-ИЛИ-дърво може да се извърши по следната схема:

- **И-възли** - представят разлаганията на проблема, при което всички подпроблеми (възли-наследници) трябва да се решават
- **ИЛИ-възли** - представят алтернативи, при което един алтернативен проблем (възел-наследник) трябва да се реши

И-ИЛИ дървета

Начален възел - изходен проблем

Възли без наследници - могат да бъдат:

-***примитивни проблеми***, които са непосредствено решими (ще ги наричаме листа)

-***нерешими проблеми*** - при достигането на такива възли разлагането на проблема в тези алтернативи е бил безрезултатен

Циклите водят до "кръгови" заключения, т.е. не могат да се намерят решения.

И-ИЛИ дървета

С други думи едно И-ИЛИ-дърво е дървовидна структура с взаимно редуващи се И-разклонения и ИЛИ-разклонения. Възлите без наследници могат да бъдат терминални (листа, примит. проблеми) или нетерминални. Коренът на дървото съответства на някакъв начален (изходен)проблем.

При генериране или обхождане на едно такова дърво И-разклоненията индикират разлагане на проблем, а ИЛИ-разклоненията специфицират възможни алтернативи.

Дърво на решение - крайно поддърво, за което:

1.всичките възли са решими;

2. съдържа един начален възел;

3.И-разклоненията съдържат всичките си наследници;

4.ИЛИ-разклонения съдържат само един наследник.

Bottom-up

- RESOLV - множество на решими възли;
- UNRESOLV - множество на нерешими възли.
- *$Step_1$ (Начална стъпка)*
- *$Step_2$(Разширение)*
  if (k = = И-разклонение AND всички наследници са в RESOLV) OR (k = = ИЛИ-разклонение AND поне един наследник е в RESOLV) then RESOLV = RESOLV $\cup$ {k}

  else UNRESOLV = UNRESOLV $\cup$ {k};
- *$Step_3$(Условия за прекъсване)*

  if (коренът на дървото е в RESOLV) then EXIT(yes);

  if (коренът на дървото е в UNRESOLV) then EXIT(no);

$Step_4$(Итерация)

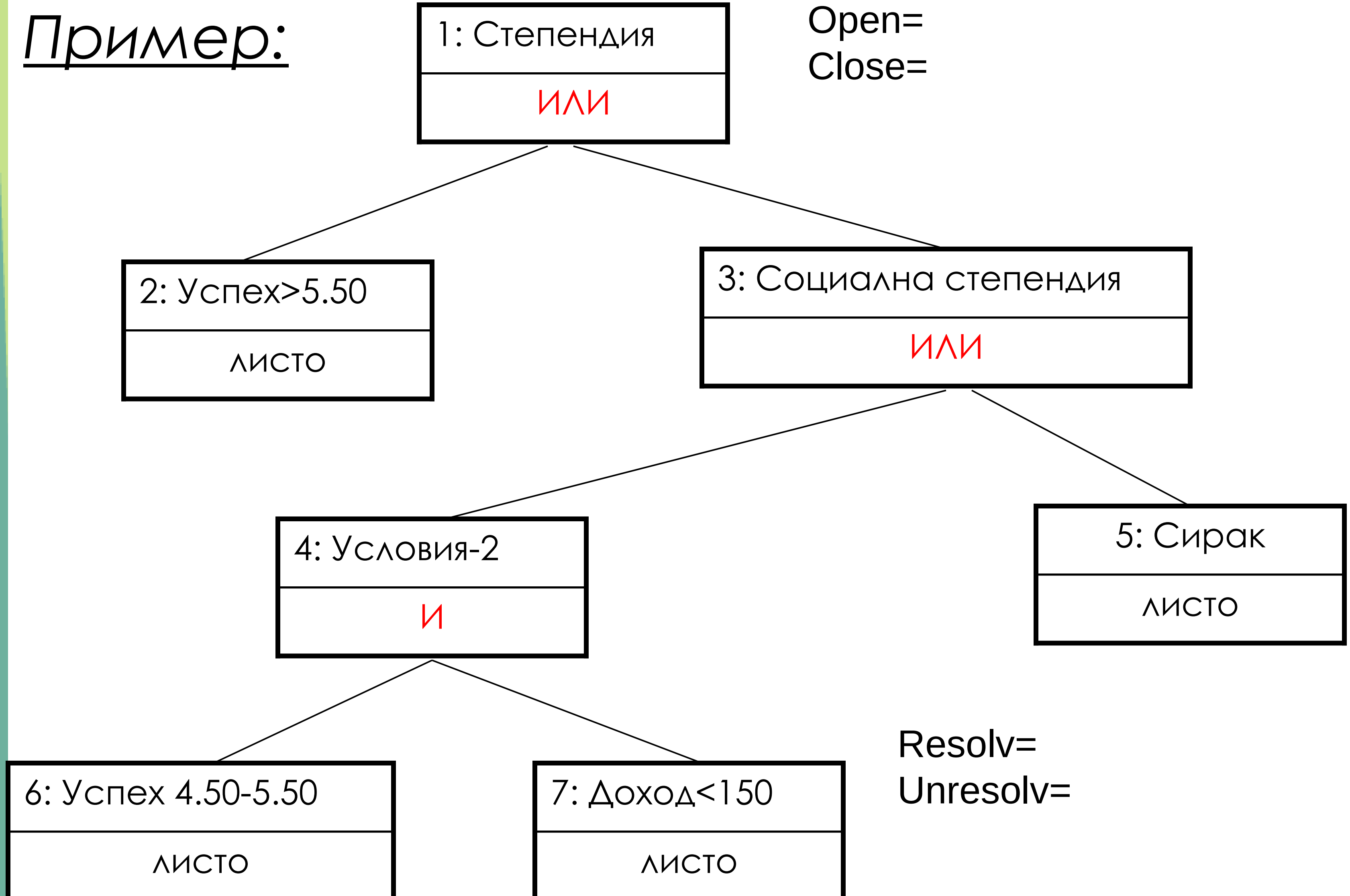
GOTO (Избор).

Top- down

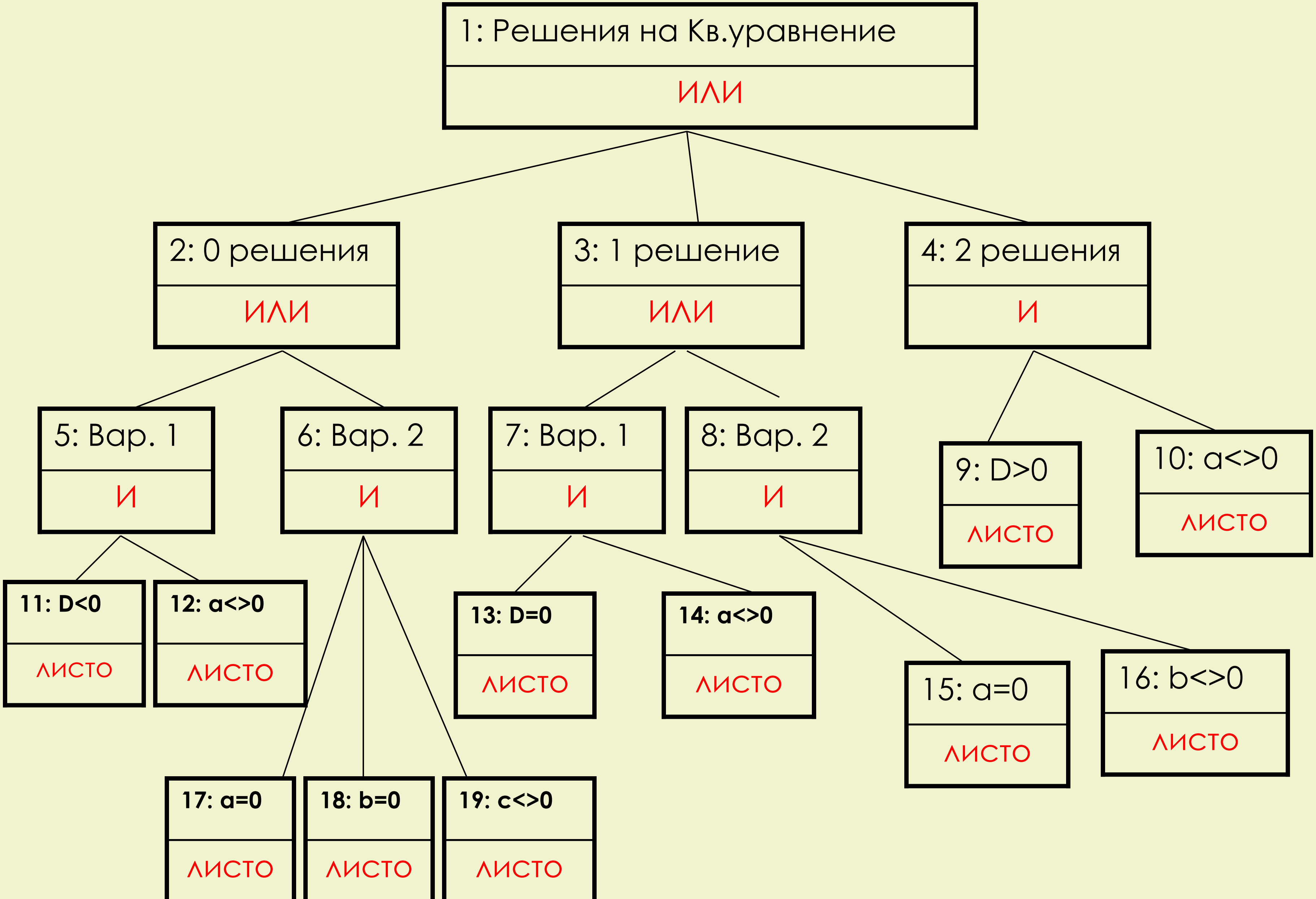
- OPEN -на отворените, но все още необработени възли
- CLOSE - на обработените възли с генерирани наследници.
- Идеята на подхода е да се започне от корена и посещавайки всеки следващ възел, да го прехвърляме от списъка OPEN в CLOSE, като генерираме наследниците му, докато стигнем до решими терминални възли (листа). После по обратен път генерираме решението.
- Интересен е проблемът къде поставяме наследниците на текущия възел - в началото на списъка или в края му. В зависимост от това търсенето ще се извърши първо в дълбочина (при по-младите инстанции) или първо в ширина (при по-старшите инстанции)

Пример:

Open=
Close=



Resolv=
Unresolv=



1: Решения на Кв.уравнение
ИЛИ
2: 0 решения
ИЛИ
3: 1 решение
ИЛИ
4: 2 решения
И
5: Вар. 1
И
6: Вар. 2
И
7: Вар. 1
И
8: Вар. 2
И
9: D>0
ЛИСТО
10: a<>0
ЛИСТО
11: D<0
ЛИСТО
12: a<>0
ЛИСТО
13: D=0
ЛИСТО
14: a<>0
ЛИСТО
15: a=0
ЛИСТО
16: b<>0
ЛИСТО
17: a=0
ЛИСТО
18: b=0
ЛИСТО
19: c<>0
ЛИСТО

Използвана литература в курса

- D. W. Hoffmann, Theoretische Informatik, Hansen Verlag, 2009
- H. P. Gumm, M. Sommer, Einfuehrung in die Informatik, Oldenbourg Wissenschaftsverlag, 2004
- J. W. Grossman, Discrete Mathematics, Macmillan Pub. Co., 1990
- К. Манев, Увод в дискретната математика, КЛМН, 2005
- Й. Денев, Р. Павлов, Я. Демирович. *Дискретна математика*. Наука и изкуство, София, 1984.

Използвана литература в курса

- Д. Байнов, С. Костадинов, Р. Павлов, Л. Луканова. *Ръководство за решаване на задачи по дискретна математика.* Университетско издателство "Паисий Хилендарски", Пловдив, 1990.
- В.А. Успенский, *Машина Поста*, Москва, Наука, 1988, ISBN 5-02-013735-9.
- L. Lovasz, J. Pelikan, K. Vesztergombi, *Discrete Mathematics – Elementary and Beyond*, Springer Verlag, New York, 2003, ISBN 0-387-95584-4.

Използвана литература в курса

- E. Bender, S. Williamson, *A Short Course in Discrete Mathematics*, Dover, 2006, ISBN 0-486-43946-1.
- P. Linz, An *Introduction to Formal Languages and Automata,* Jones and Bartlett Publishers, 6-th edition, Jones & Bartlett Publishers, ISBN-13: 9781284077247, 2016
- Kenneth H. Rosen, Kamala Krithivasan, Discrete mathematics and its application, McGraw-Hill Companies, 7-th edition, ISBN 978-0-07-338309-5, 2012

Използвана литература в курса

- Owen D. Byer, Deirdre L. Smeltzer, Kenneth L. Wantz, Journey into Discrete Mathematics, AMS, MAA Press, Providence Rhode Island, ISBN 9781470446963, 2018
- Christopher Rhoades, Introductory Discrete Mathematics, Willford Press, ISBN 1682854922, 9781682854921, 2018
- David Liben-Nowell, Discrete Mathematics for Computer Science, Wiley, 2017, ISBN 1119397197, 9781119397199, 2017.
- ***http://www.jflap.org/*** - софтуерна среда