Дискретна математика

проф. д-р Тодорка Глушкова,
Катедра „Компютърни технологии", ФМИ

4

Теория на дърветата

Съдържание:

- Дефиниции
- Свойства
- Бинарни дървета
- Генериране списък на възлите
- И-Или дървета

Въведение

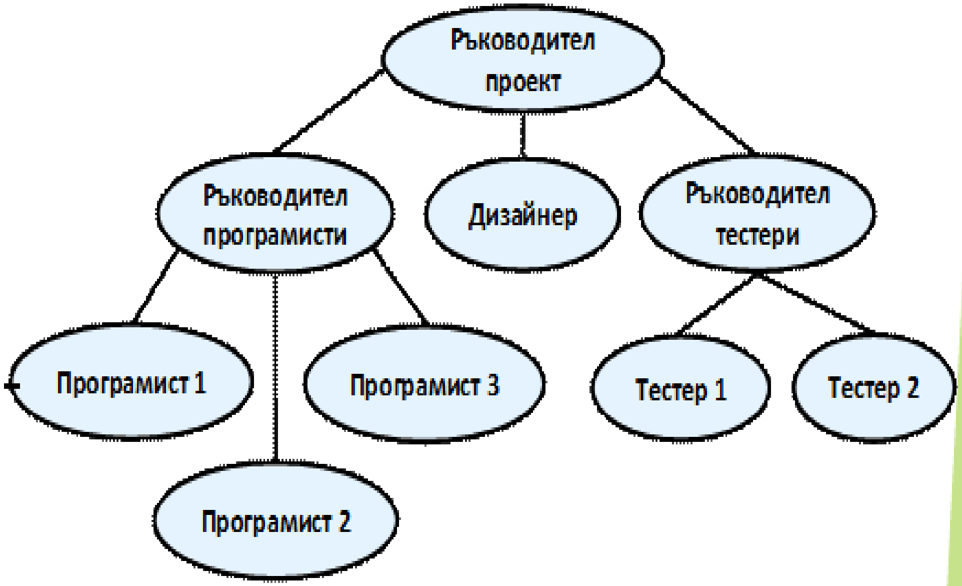
- Ще разгледаме т. нар. дървовидни структури от данни.
- Те се използва за моделирането на проблеми от реалността, които се решават ефективно с тяхна помощ.
- Ще разгледаме в детайли какво представляват дървовидните структури от данни и ще покажем техните основни предимства и недостатъци.
- Ще се спрем по-подробно на двоичните дървета.

Въведение

- В програмирането дърветата са изключително често използвана структура от данни, защото те моделират по естествен начин всякакви йерархии от обекти, които постоянно ни заобикалят в реалния свят.
- Да разгледаме един пример, преди да изложим терминологията, свързана с дърветата.

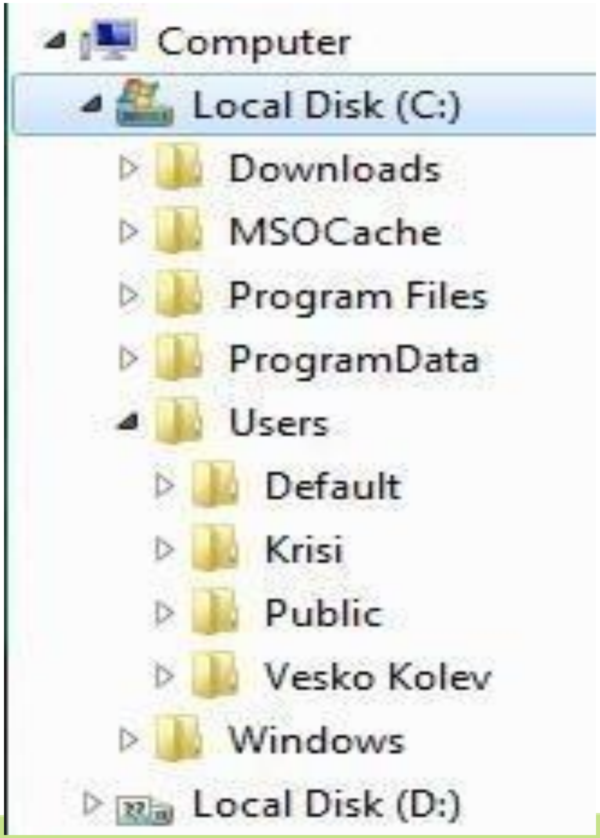
Въвеждащ пример 1

- Екип, отговорен за изработването на даден софтуерен проект.
- Участниците в него са взаимно свързани с връзката ръководител-подчинен.
- В конкретната ситуация имаме екип от 9 души:



Въвеждащ пример 2

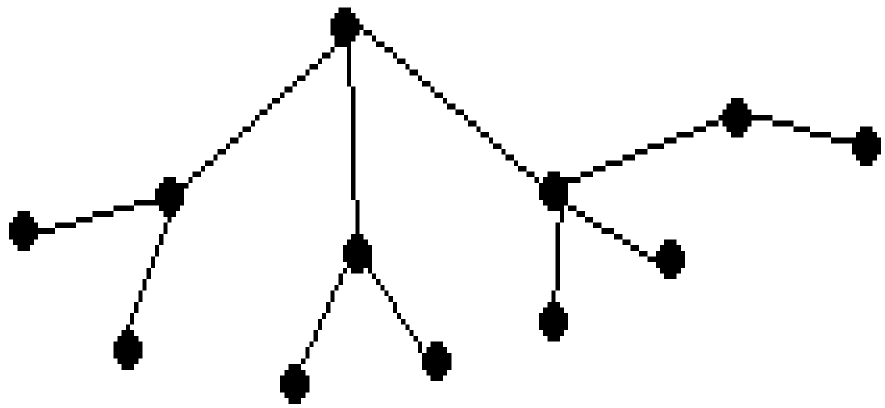
- Нека разгледаме файловата система.
- Директориите върху твърдия диск образуват йерархична структура, която е дърво.



Дърво. Дефиниции

- **<u>Дефиниция:</u>** ***Дървото*** е свързан, ориентиран граф, който не съдържа затворени вериги. Обикновено се бележи с T.
- Дървото има един специален възел - ***корен***.
- Дърво с корен бележим - (T,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.

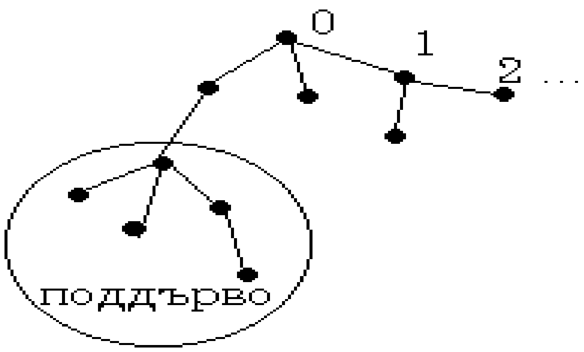
б) G има точно n-1-ребра и няма затворени вериги.

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.

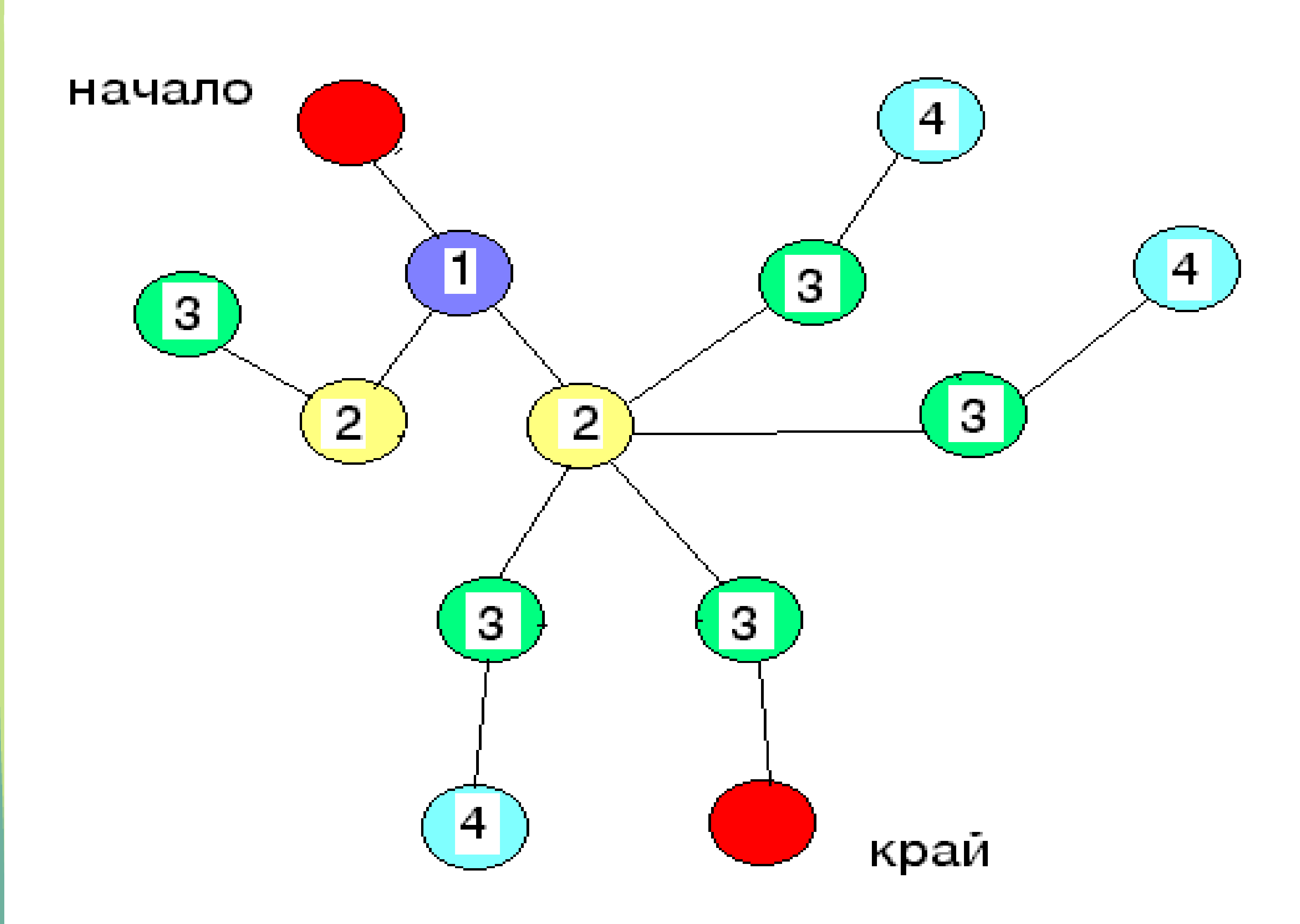
Дефиниции и свойства

- <u>Забележка:</u> От Теоремата следва, че съществува уникален път от корена до всеки друг възел.
- <u>Дефиниция</u>: **Ниво** на един възел е дължината на пътя от корена до възела. Коренът има ниво 0. Съседните му- ниво 1 и т.н.
- <u>Дефиниция:</u> **Поддърво** на T е дървото T1⊆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
4
1
3
3
4
2
2
3
3
3
4
край

Дефиниции и свойства

Можем да дадем рекурсивна дефиниция на дърво с корен:

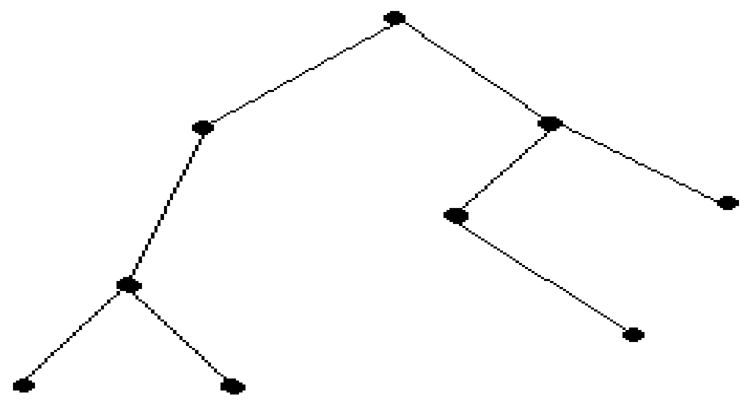
- Базов случай: Всеки отделен възел е дърво с корен- самият възелът.
- Рекурсивен случай: Ако $k \geq 1$ и $T1,T2,\ldots Tк$, $Ti \cap Tj = \emptyset$ са дървета с корени съответно $v1,v2,\ldots vk$, тогава следния граф е също дърво с корен: един нов възел v е негов корен, съвместно с $T1,T2,\ldots Tк$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

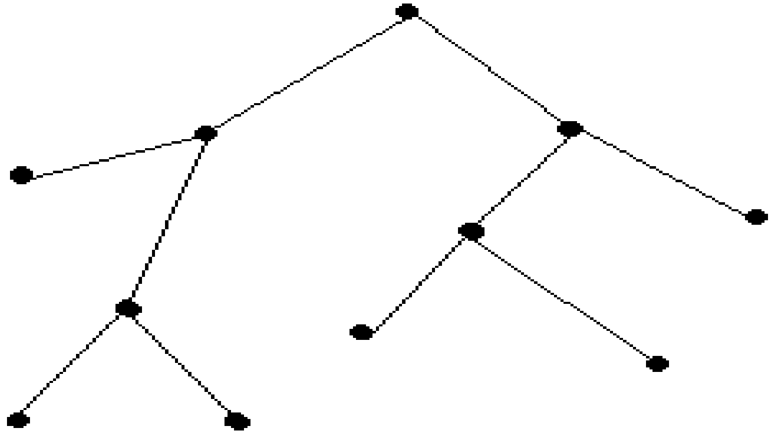
Двоично дърво

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



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

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

<u>Доказателство:</u> 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 $\Rightarrow$**<u>e=i+1</u>**.

Spanning дървета

<u>Дефиниция:</u> ***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)$\leftarrow$true {започваме от възел 1}

for i$\leftarrow$2 to n do

visited(i) $\leftarrow$false {не са посетени други възли}

T$\leftarrow$({1}, $\varnothing$) {Първоначално дървото съдържа възел 1 и няма ребра}

(T, visited) $\leftarrow$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.

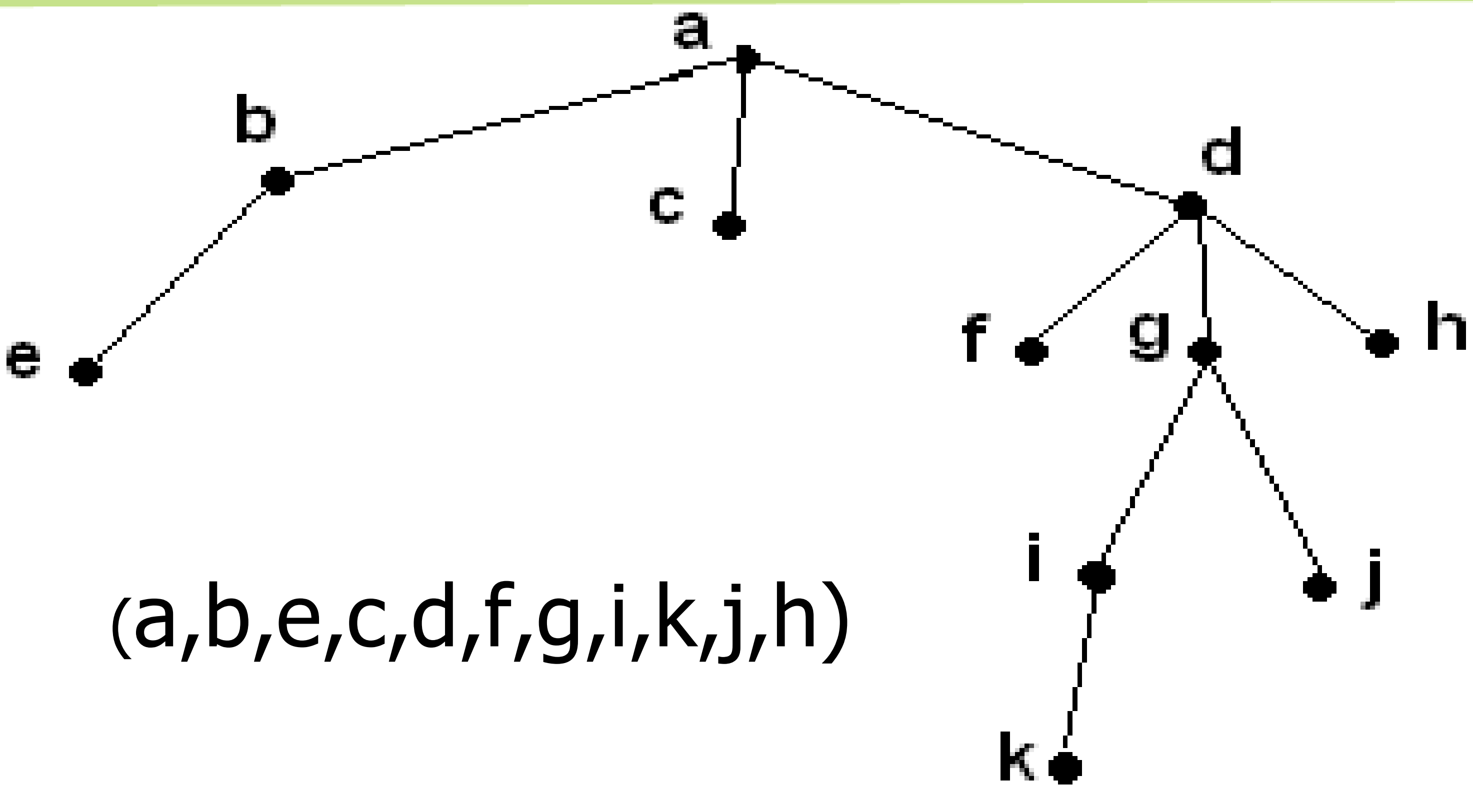
- **Проблем:** Да генерираме систематизиран списък на възлите на едно дърво.
- Съществуват различни начини за решаването на този проблем:

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)

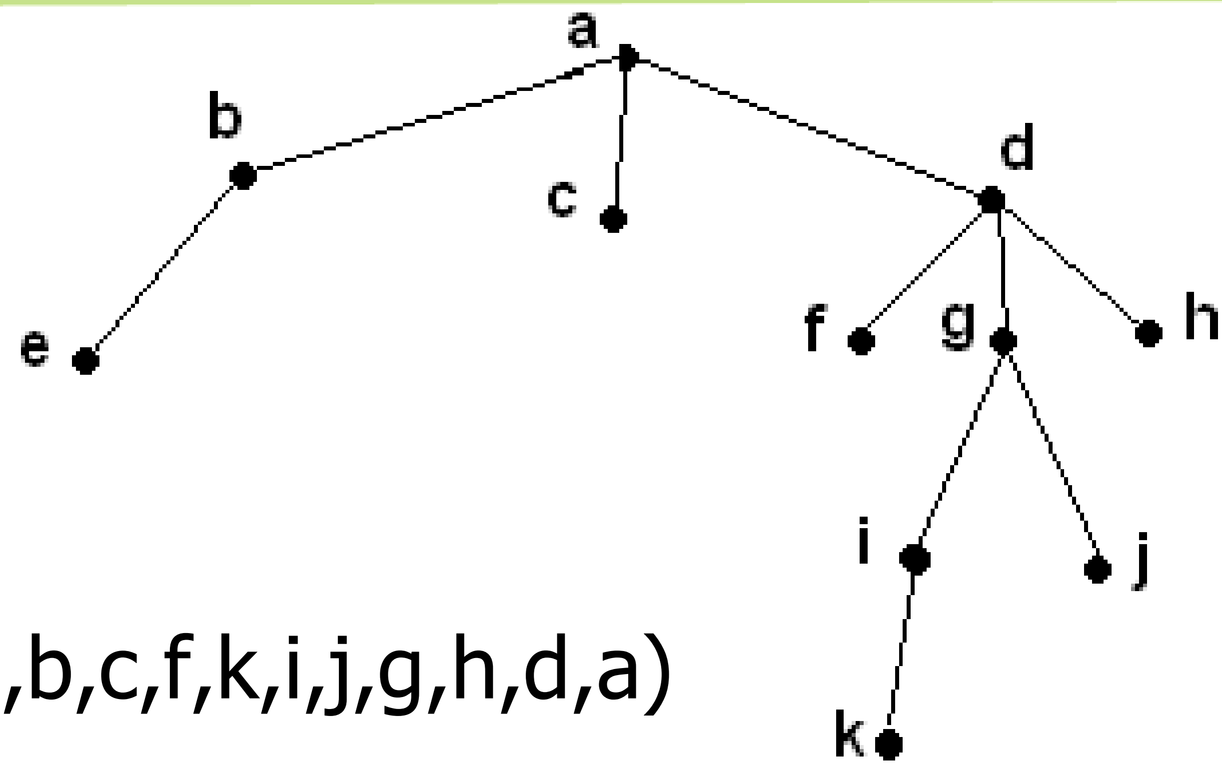
Return;

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

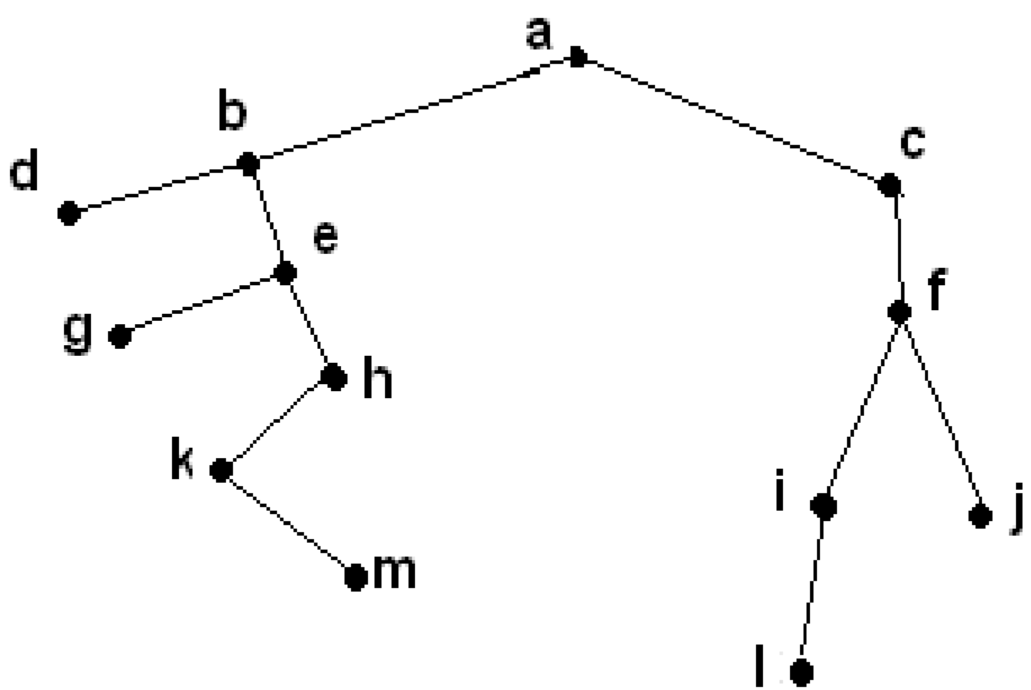
Return;

Инордер (inorder)

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

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

Инордер. Пример

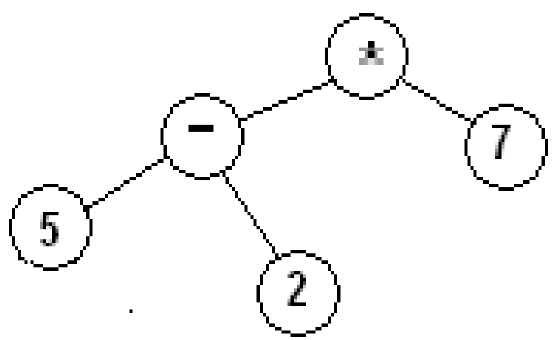


(d,b,g,e,k,m,h,a,c,l,i,f,j)

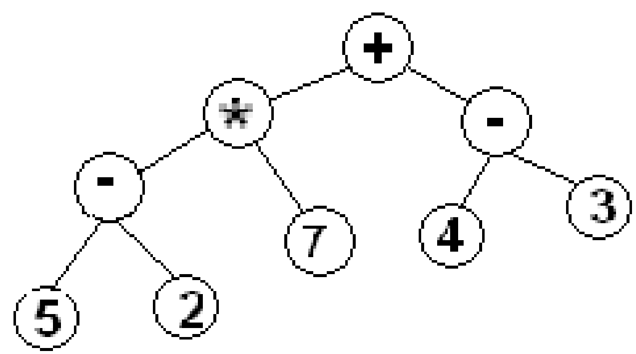
Expression дървета

- 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)

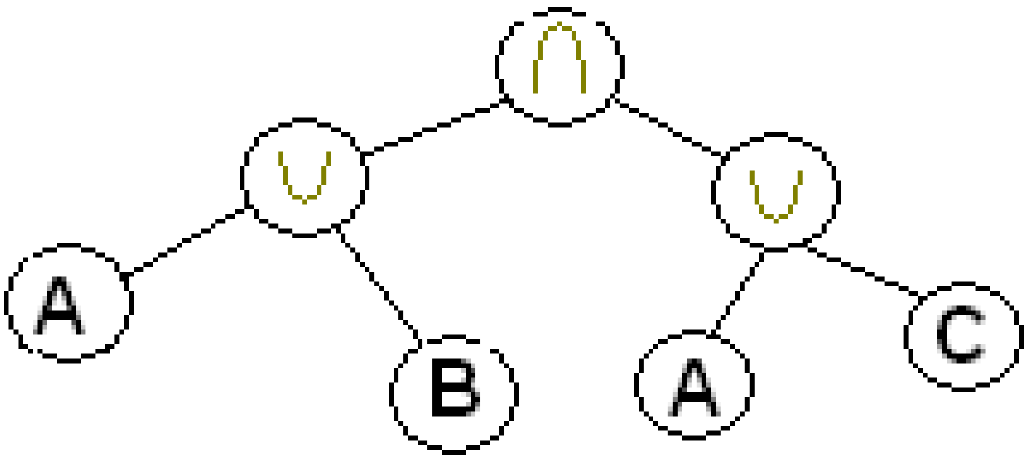


б) (5-2)*7 + (4 - 3)



Expression дървета

в) (A ∪ B) ∩ (A∪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 един итерационен процес от последователен избор на алтернативи и съответно разлаганена проблемите на подпроблеми. При всяка итерационна стъпка трябва да решавамеедна алтернатива (ИЛИ-свързаност) или да решаваме последователно всички подпроблеми(И-свързаност), като отделните подпроблемите 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);

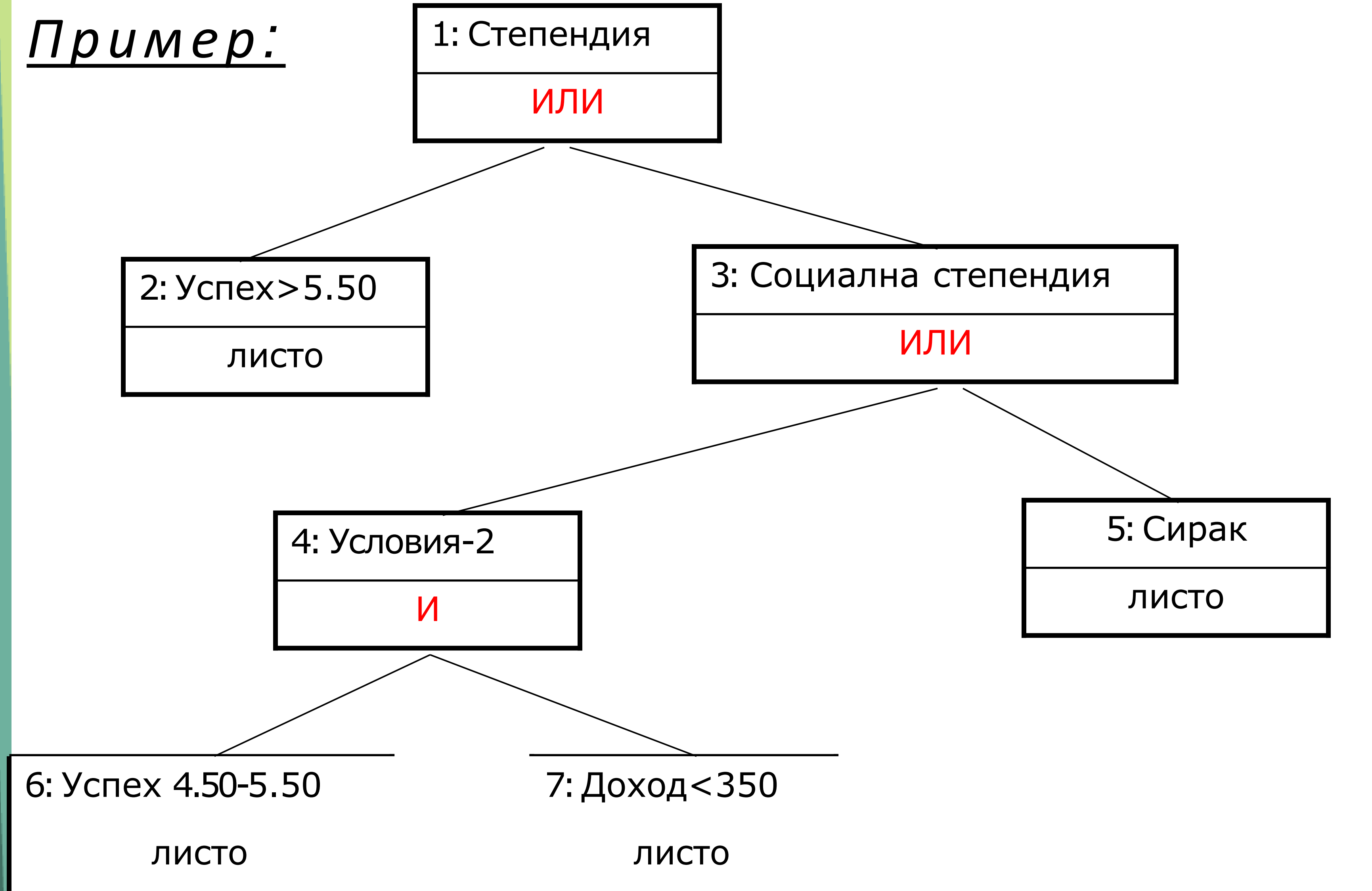
$Step_4$(Итерация)

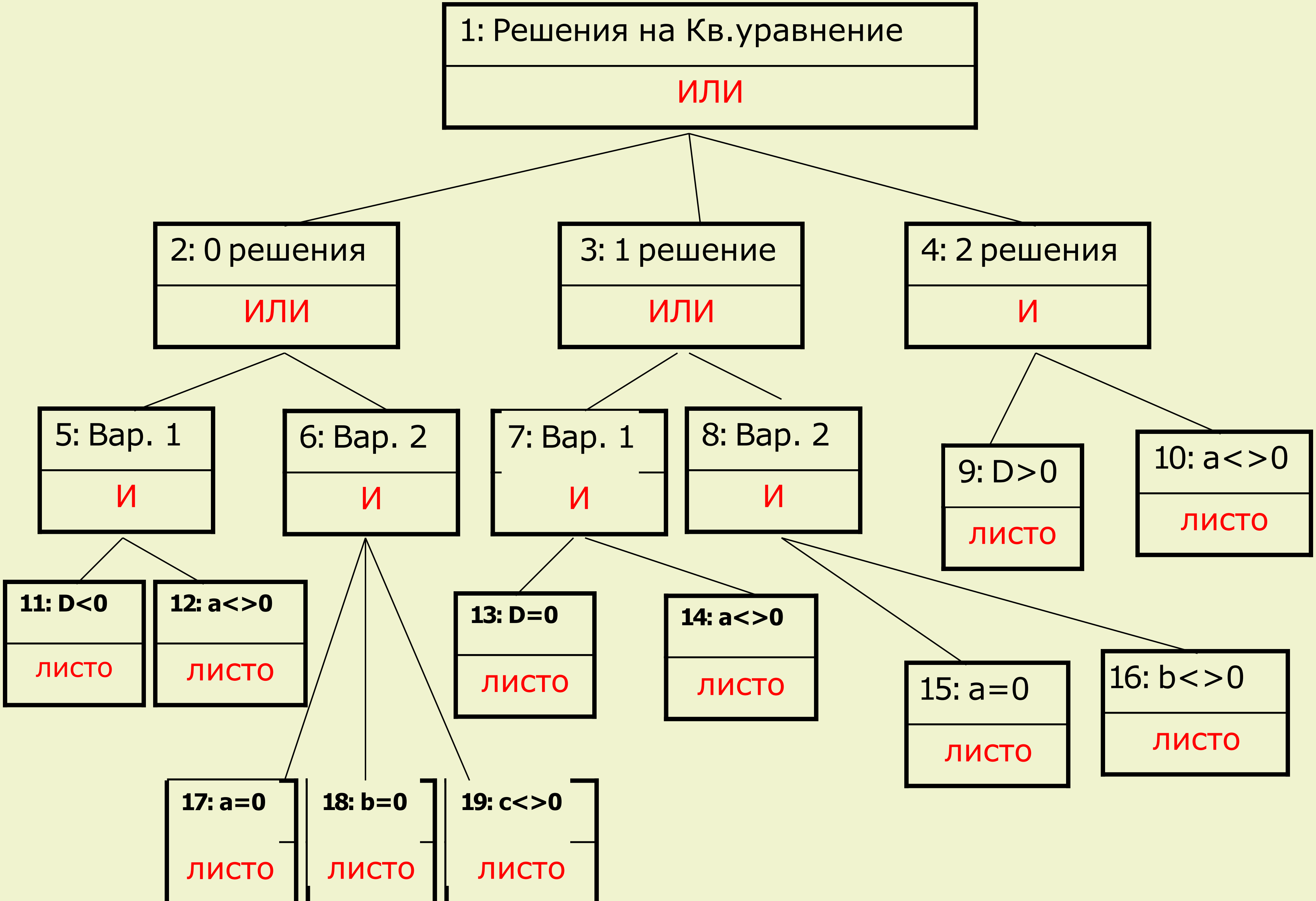
GOTO (Избор).

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:


1: Степендия
или
2: Успех>5.50
листо
3: Социална степендия
или
4: Условия-2
и
5: Сирак
листо
6: Успех 4.50-5.50
листо
7: Доход<350
листо


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