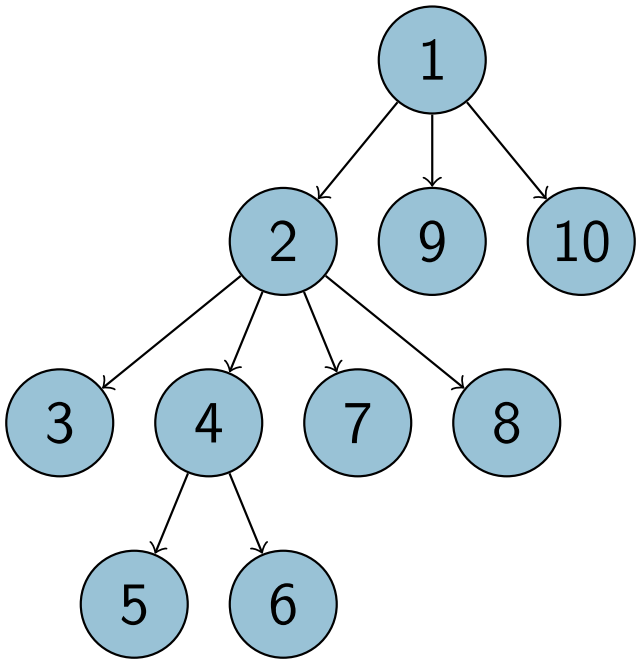
Дървета

Трифон Трифонов

Структури от данни и програмиране,
спец. Компютърни науки, 2 поток, 2015/16 г.

27 ноември 2015 г.

Пример: кореново дърво



АТД: Дърво

Йерархична структура, в която на всеки елемент е съпоставено множество от подчинени елементи.

Дефиниция

Кореново дърво е списък $(X, T_1, \ldots, T_n)$, където

- X е данна (корен)
- $T_1, T_2, \ldots, T_n$ са коренови дървета (поддървета)

АТД: Дърво

Йерархична структура, в която на всеки елемент е съпоставено множество от подчинени елементи.

Дефиниция

Кореново дърво е списък $(X, T_1, \ldots, T_n)$, където

- X е данна (корен)
- $T_1, T_2, \ldots, T_n$ са коренови дървета (поддървета), $n \geq 0$

АТД: Дърво

Йерархична структура, в която на всеки елемент е съпоставено множество от подчинени елементи.

Дефиниция

Кореново дърво е списък $(X, T_1, \ldots, T_n)$, където

- X е данна (корен)
- $T_1, T_2, \ldots, T_n$ са коренови дървета (поддървета), $n \geq 0$

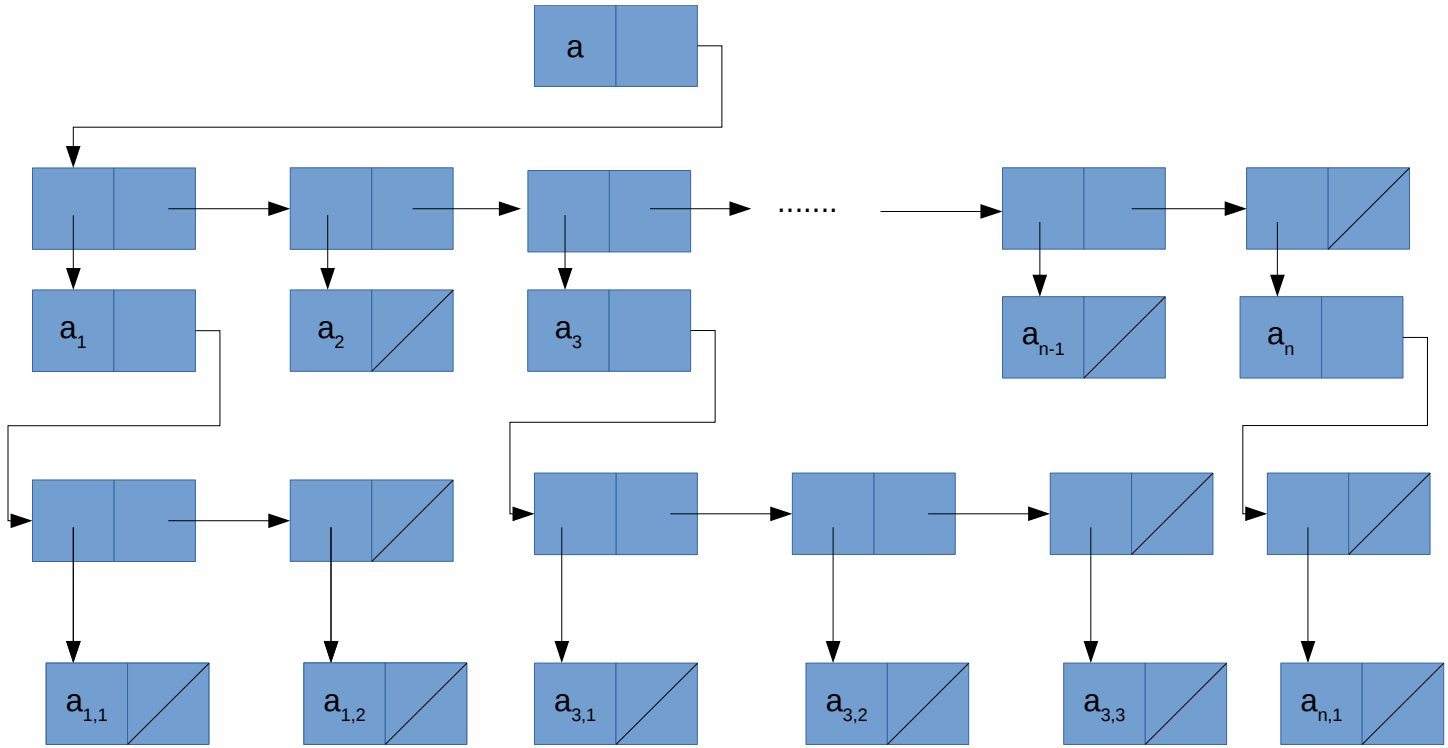
Операции

- `create(x)` — създаване на дърво с корен x
- `addChild(t)` — добавяне на поддърво t
- `root()` — достъп до корена
- `subtrees()` — достъп до поддърветата

Свързано представяне

Свързана структура от **възли**, където всеки възел се състои от:

- стойността на корена
- свързан списък от възли, представящи поддърветата (деца)



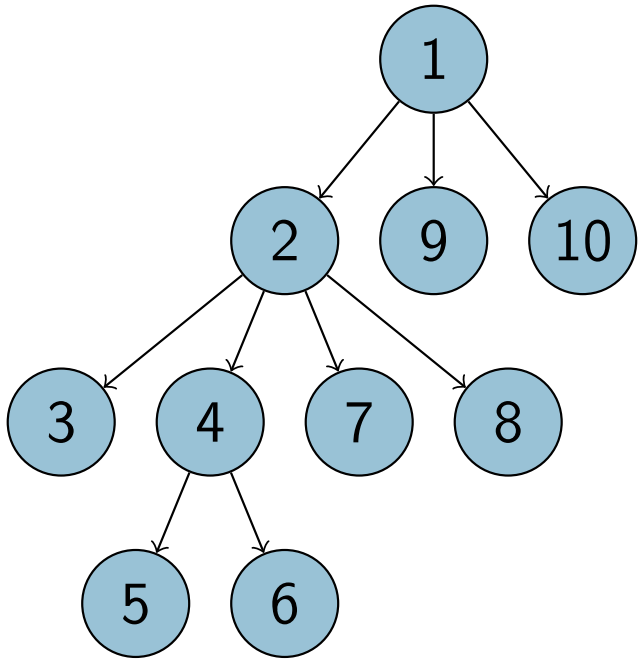
Последователно представяне

Последователност от тройки (корен, най-ляво дете, десен брат)

	0	1	2	3	4	5	6	7	8	...
корен	a	a_1	$a_{1,1}$	a_2	$a_{1,2}$	a_3	$a_{3,1}$	$a_{3,2}$	$a_{3,3}$	...
най-ляво дете	1	2	-1	-1	-1	6	-1	-1	-1	...
десен брат	-1	3	4	5	-1	9	7	8	-1	...

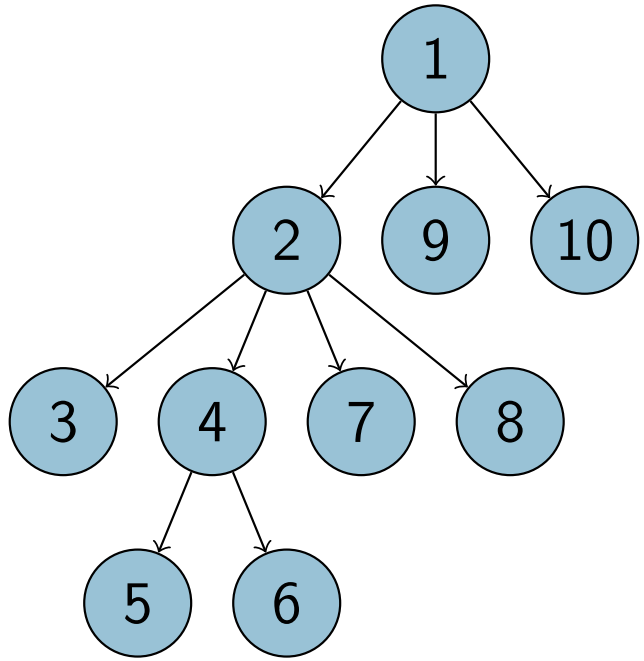
Дефиниции за дърво

- **родител** — възел, сред чиито деца е даден възел
- **листо** — възел без деца
- **ниво** — множество от възлите на еднакво разстояние от кореновия възел
- **път** — редица от възли, в която всеки следващ е сред децата на предходния
- **височина (дълбочина)** — броят на възлите по най-дългия път от кореновия възел до листо
- **широчина (разклоненост)** — максималният брой деца на възел



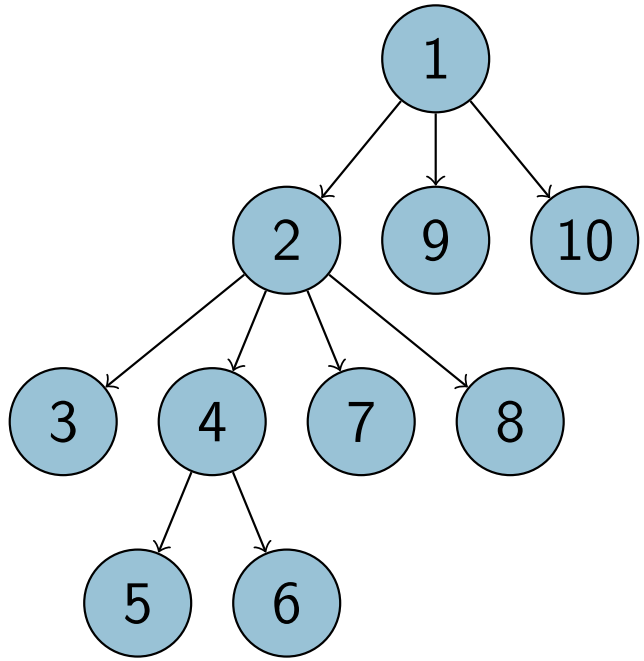
Схеми за обхождане

- префиксно обхождане
  - първо посещаваме корена
  - след това обхождаме децата



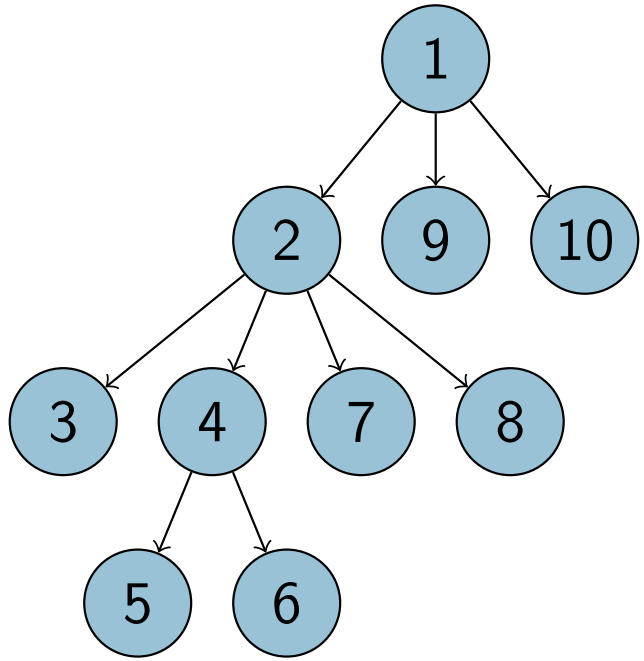
Схеми за обхождане

- префиксно обхождане
  - първо посещаваме корена
  - след това обхождаме децата
  - 1, 2, 3, 4, 5, 6, 7, 8, 9, 10



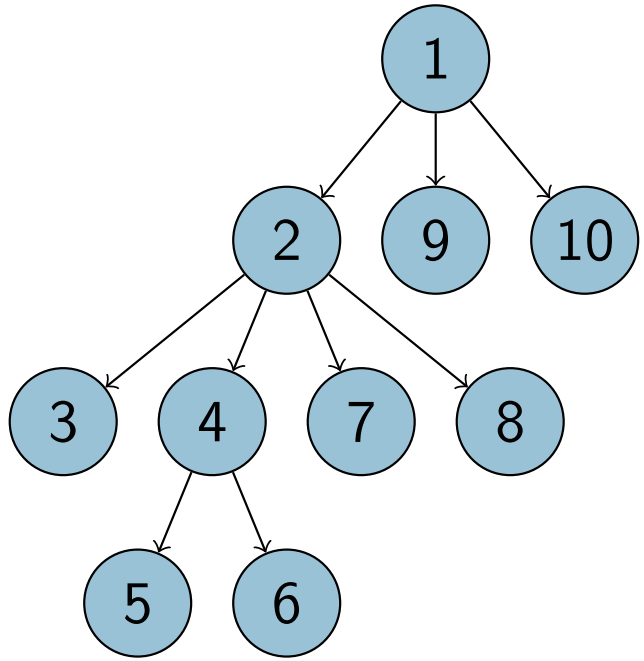
Схеми за обхождане

- префиксно обхождане
  - първо посещаваме корена
  - след това обхождаме децата
  - 1, 2, 3, 4, 5, 6, 7, 8, 9, 10
- постфиксно обхождане
  - първо обхождаме децата
  - накрая посещаваме корена



Схеми за обхождане

- префиксно обхождане
  - първо посещаваме корена
  - след това обхождаме децата
  - 1, 2, 3, 4, 5, 6, 7, 8, 9, 10
- постфиксно обхождане
  - първо обхождаме децата
  - накрая посещаваме корена
  - 3, 5, 6, 4, 7, 8, 2, 9, 10, 1



Дървета с фиксиран брой поддървета

Ако искаме всеки възел на дървото да има един и същ (максимален) брой деца, използваме алтернативна дефиниция

Дървета с фиксиран брой поддървета

Ако искаме всеки възел на дървото да има един и същ (максимален) брой деца, използваме алтернативна дефиниция

Дефиниция (n-арно дърво)

- $\perp$ (празното дърво)
- наредена $n+1$-торка $(X, T_1, \ldots, T_n)$, където
  - X е данна (корен)
  - $T_1, T_2, \ldots, T_n$ са n-арни дървета (поддървета)

Дървета с фиксиран брой поддървета

Ако искаме всеки възел на дървото да има един и същ (максимален) брой деца, използваме алтернативна дефиниция

Дефиниция (n-арно дърво)

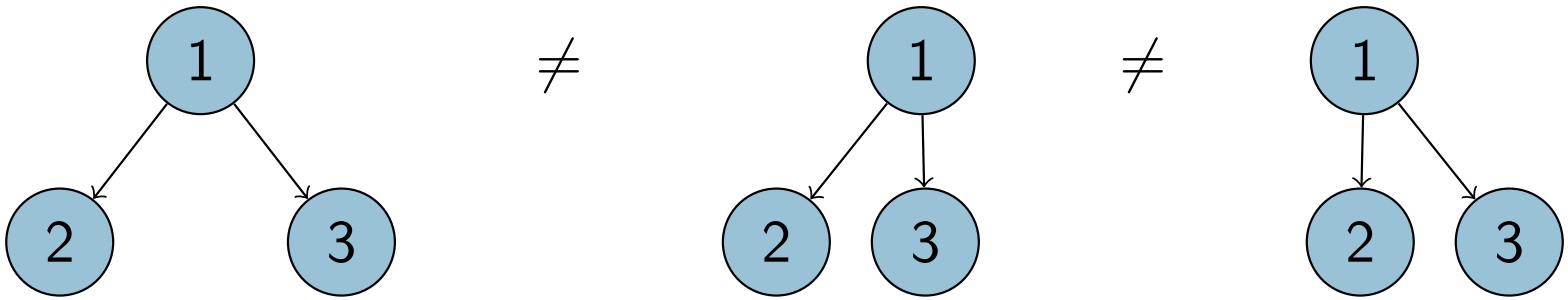
- $\perp$ (празното дърво)
- наредена $n+1$-торка $(X, T_1, \ldots, T_n)$, където
  - X е данна (корен)
  - $T_1, T_2, \ldots, T_n$ са n-арни дървета (поддървета)

Операции

- `empty_tree()` — създаване на празно дърво
- `create_tree(x,`$\mathtt{t}_1$`,...,`$\mathtt{t}_n$`)` — създаване на n-арно дърво с корен `x` и поддървета $\mathtt{t}_1, \ldots, \mathtt{t}_n$
- `root()` — достъп до корена
- `subtree(i)` — достъп до `i`-тото поддърво

Наредба на поддърветата

Дефинициите за коренови дървета и n-арни дървета не са еквивалентни!

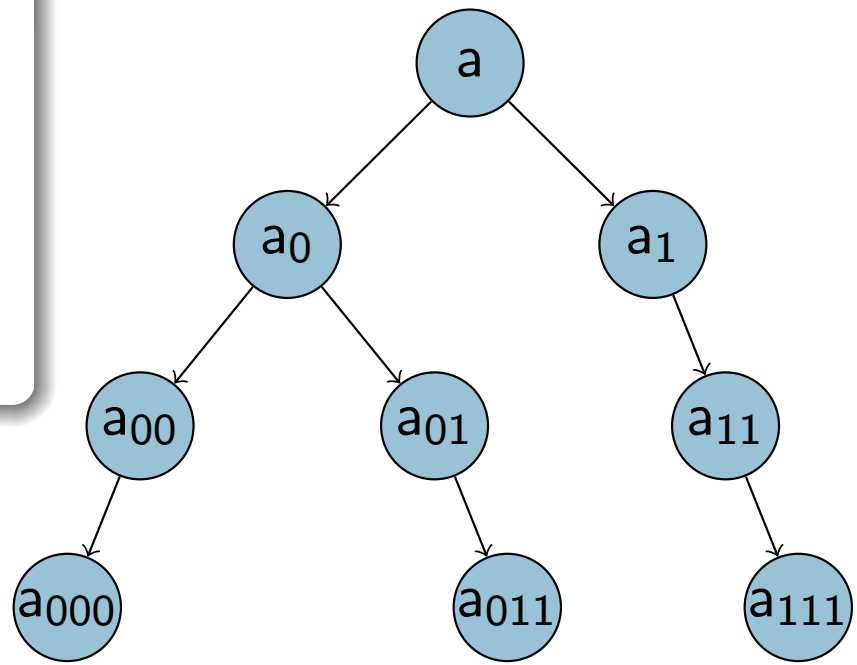


Едно и също кореново дърво може да бъде представено като n-арно дърво по няколко различни начина.

Двоично дърво

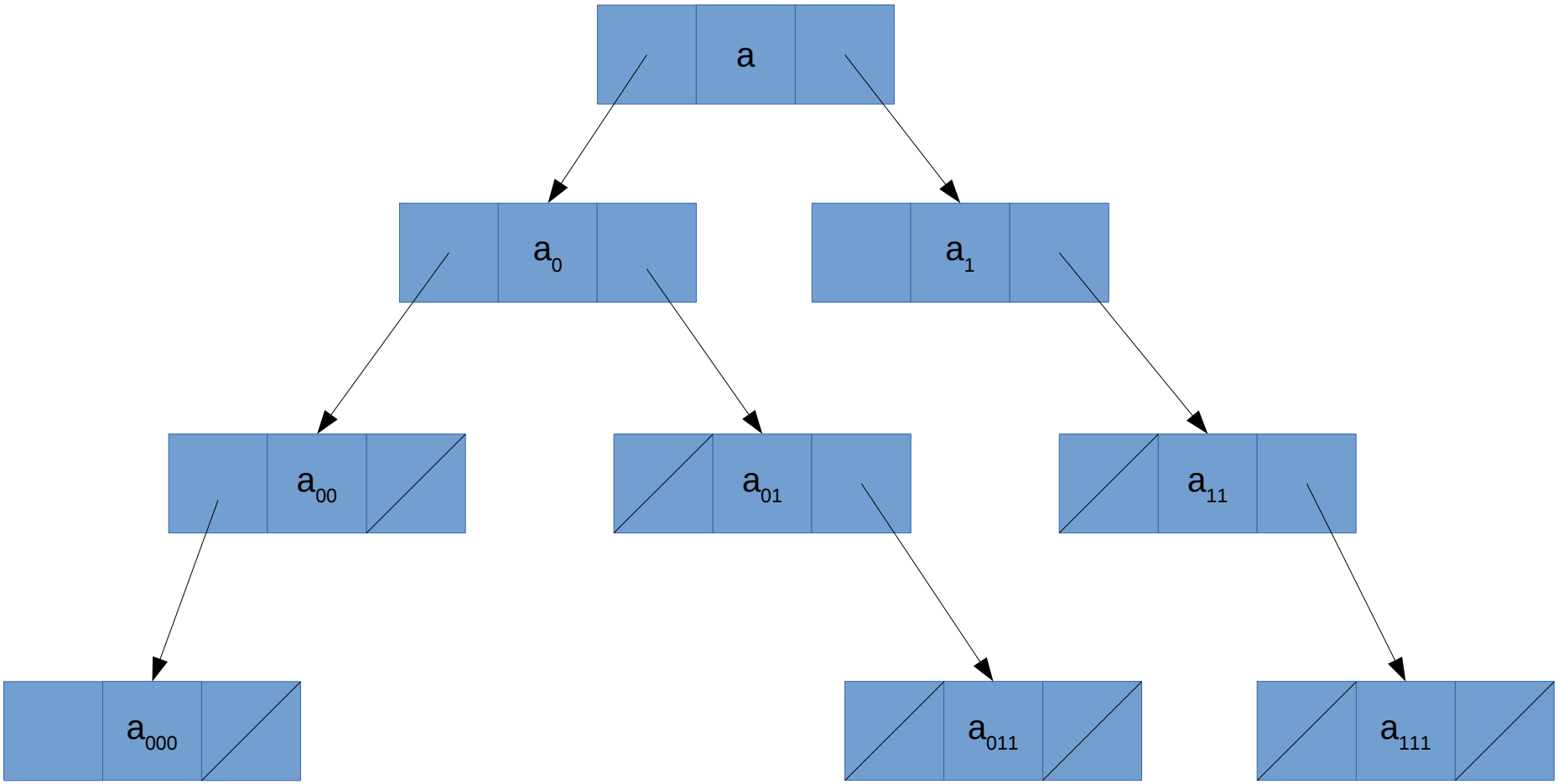
Дефиниция (двоично дърво)

- $\perp$ (празното дърво)
- наредена тройка (X, L, R), където
  - X е данна (корен)
  - L е ляво поддърво
  - R е дясно поддърво



Свързано представяне

Свързана структура от тройни кутии



Последователно представяне

Последователност от тройки (корен, ляво, дясно)

	0	1	2	3	4	5	6	7	8	...
корен	a	a_0	a_1	a_{00}	a_{01}	a_{11}	a_{000}	a_{011}	a_{111}	...
ляво дете	1	3	-1	6	-1	-1	-1	-1	-1	...
дясно дете	2	4	5	-1	7	8	-1	-1	-1	...

Позиция в двоично дърво

Ще въведем абстракция за позиция в двоичното дърво.

Операции:

- `root()` — инициализация в корена на дървото
- `left()` — преместване наляво
- `right()` — преместване надясно
- `up()` — преместване нагоре
- `get()` — достъп до елемент на дадената позиция
- `valid()` — проверка за валидност

Позиция в двоично дърво

Ще въведем абстракция за позиция в двоичното дърво.

Операции:

- `root()` — инициализация в корена на дървото
- `left()` — преместване наляво
- `right()` — преместване надясно
- `up()` — преместване нагоре
- `get()` — достъп до елемент на дадената позиция
- `valid()` — проверка за валидност

Не използваме термина “итератор”, понеже не абстрахираме начина на обхождане, а само позицията.

Псевдоними на стойности в C++11

- `T&&` — псевдоним на **стойност** от тип `T`

Псевдоними на стойности в C++11

- `T&&` — псевдоним на **стойност** от тип `T`
  - rvalue reference

Псевдоними на стойности в C++11

- `T&&` — псевдоним на **стойност** от тип `T`
  - rvalue reference
- разлика с `T&`: може да се свърза със стойности (rvalue)

Псевдоними на стойности в C++11

- `T&&` — псевдоним на **стойност** от тип `T`
  - rvalue reference
- разлика с `T&`: може да се свърза със стойности (rvalue)
  - `int&  x = 3;` — грешка

Псевдоними на стойности в C++11

- `T&&` — псевдоним на **стойност** от тип `T`
  - rvalue reference
- разлика с `T&`: може да се свърза със стойности (rvalue)
  - `int&  x = 3;` — грешка
  - `int&& x = 3;` — OK

Псевдоними на стойности в C++11

- `T&&` — псевдоним на **стойност** от тип `T`
  - rvalue reference
- разлика с `T&`: може да се свърза със стойности (rvalue)
  - `int&  x = 3;` — грешка
  - `int&& x = 3;` — OK
- разлика с `T const&`: може да променя стойността

Псевдоними на стойности в C++11

- `T&&` — псевдоним на **стойност** от тип `T`
  - rvalue reference
- разлика с `T&`: може да се свърза със стойности (rvalue)
  - `int&  x = 3;` — грешка
  - `int&& x = 3;` — OK
- разлика с `T const&`: може да променя стойността
  - `Point p = Point(1,2); p.x = 3;` — променя се копието

Псевдоними на стойности в C++11

- `T&&` — псевдоним на **стойност** от тип `T`
  - rvalue reference
- разлика с `T&`: може да се свърза със стойности (rvalue)
  - `int&  x = 3;` — грешка
  - `int&& x = 3;` — OK
- разлика с `T const&`: може да променя стойността
  - `Point p = Point(1,2); p.x = 3;` — променя се копието
  - `Point const& p = Point(1,2); p.x = 3;` — грешка

Псевдоними на стойности в C++11

- `T&&` — псевдоним на **стойност** от тип `T`
  - rvalue reference
- разлика с `T&`: може да се свърза със стойности (rvalue)
  - `int&  x = 3;` — грешка
  - `int&& x = 3;` — OK
- разлика с `T const&`: може да променя стойността
  - `Point p = Point(1,2); p.x = 3;` — променя се копието
  - `Point const& p = Point(1,2); p.x = 3;` — грешка
  - `Point& p = Point(1,2); p.x = 3;` — грешка

Псевдоними на стойности в C++11

- `T&&` — псевдоним на **стойност** от тип `T`
  - rvalue reference
- разлика с `T&`: може да се свърза със стойности (rvalue)
  - `int&  x = 3;` — грешка
  - `int&& x = 3;` — OK
- разлика с `T const&`: може да променя стойността
  - `Point p = Point(1,2); p.x = 3;` — променя се копието
  - `Point const& p = Point(1,2); p.x = 3;` — грешка
  - `Point& p = Point(1,2); p.x = 3;` — грешка
  - `Point&& p = Point(1,2); p.x = 3;` — OK

Псевдоними на стойности в C++11

- `T&&` — псевдоним на **стойност** от тип `T`
  - rvalue reference
- разлика с `T&`: може да се свърза със стойности (rvalue)
  - `int&  x = 3;` — грешка
  - `int&& x = 3;` — OK
- разлика с `T const&`: може да променя стойността
  - `Point p = Point(1,2); p.x = 3;` — променя се копието
  - `Point const& p = Point(1,2); p.x = 3;` — грешка
  - `Point& p = Point(1,2); p.x = 3;` — грешка
  - `Point&& p = Point(1,2); p.x = 3;` — OK
- използва се за ефективност: вместо копиране на данните от временна стойност, може да се извърши преместване

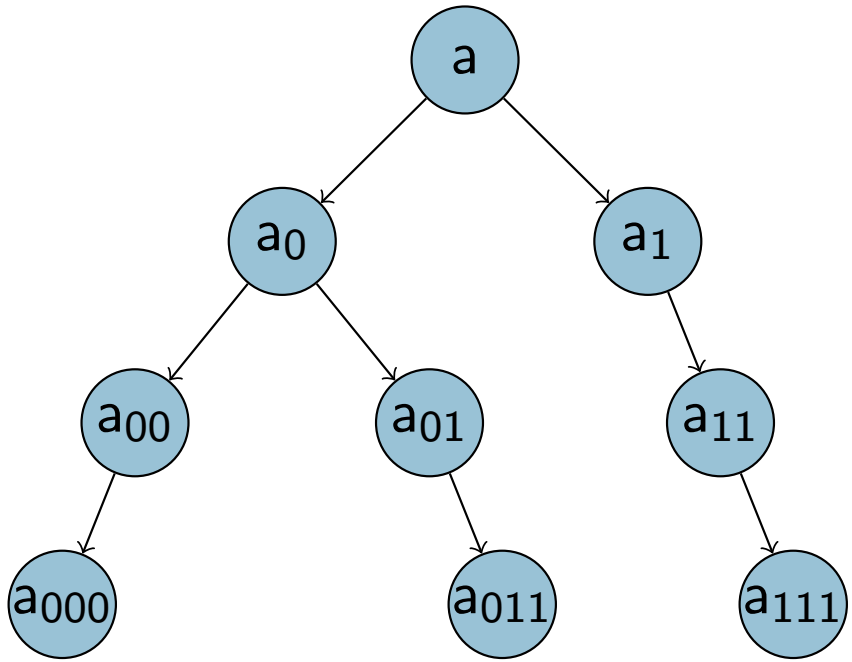
Визуализация на дървета

GraphViz (`graphviz.org`) е набор от инструменти за визуализация на графи.

- Изключително разнообразни възможности за настройка на графичния изход
- Разнообразни алгоритми за автоматично разполагане на графи
- Използва DOT формат за описание на графите
- `digraph` <име-на графа>`{` { <родител> `->` <дете>`;` } `}`
- `dot -Tpng graph.dot > graph.png`
- Можем да го използваме за бърза визуализация на дървета

Схеми за обхождане

- префиксни
  - КЛД
  - КДЛ
- инфиксни
  - ЛКД
  - ДКЛ
- постфиксни
  - ЛДК
  - ДЛК



Задачи за двоично дърво

Задача. Да се намери дълбочината на дадено дърво.

Задачи за двоично дърво

Задача. Да се намери дълбочината на дадено дърво.

Задача. Да се провери дали две дадени дървета съвпадат.

Задачи за двоично дърво

Задача. Да се намери дълбочината на дадено дърво.

Задача. Да се провери дали две дадени дървета съвпадат.

Задача. Да се построи дърво на аритметичен израз със скоби.

Задачи за двоично дърво

Задача. Да се намери дълбочината на дадено дърво.

Задача. Да се провери дали две дадени дървета съвпадат.

Задача. Да се построи дърво на аритметичен израз със скоби.

Задача. Да се премсетне аритметичен израз, записан в дърво.