Algoritmai ir duomenų struktūros

3 paskaita

2020-02-19

Informacija

Kovo 4 d. (už dviejų savaičių) ADS teorijos paskaitoje svečiuosis prof. dr. Dmitrij Šešok (VGTU) ir skaitys pranešimą tema „Olimpiadinis programavimas“:

- Olimpiadinis (sportinis) programavimas,
- Šio programavimo stipriosios bei silpnosios vietos,
- Olimpiadinio programavimo metodai,
- Šių metodų taikymas klasikinių studijų metu,
- Patirtis ruošiant studentus programavimo olimpiadoms (ICPC).

3 paskaitos tikslas

- Euklido algoritmo įrodymas.
- Tiesinio sąrašo realizacija, modifikacijos, *C++* ir *Python* bibliotekos.
- Tiesinių duomenų struktūrų privalumai ir trūkumai.
- Abstraktus duomenų tipas:
  - Dinaminė aibė,
  - Medžiai,
  - Dvejetainiai medžiai,
  - Dvejetainis paieškos medis,
  - Duomenų struktūra heap,
  - Prioritetinė eilutė,
  - Galimos operacijos ir pavyzdžiai.

Euklido algoritmo įrodymas (1)

- Teorema. Realizavus algoritmą **Euklidas**(a, b), gaunamas rezultatas yra skaičių a ir b didžiausias bendras daliklis.

```
Euklidas(a, b)
    while b > 0
        c ← b
        b ← a % b
        a ← c
    return(a)
```

$a = bq_1+r_1$
$b = r_1q_2+r_2$
$r_1 = r_2q_3+r_3$
…
$r_{n-3} = r_{n-2}q_{n-1}+r_{n-1}$
$r_{n-2} = r_{n-1}q_n+0$

tada r_{n-1} = DBD(a,b).

Euklido algoritmo įrodymas (2)

- Įrodymas.

$a = bq_1 + r_1$

$b = r_1q_2 + r_2$

$r_1 = r_2q_3 + r_3$

2. …

3.

$r_{n-3} = r_{n-2}q_{n-1} + r_{n-1}$

$r_{n-2} = r_{n-1}q_n + 0$

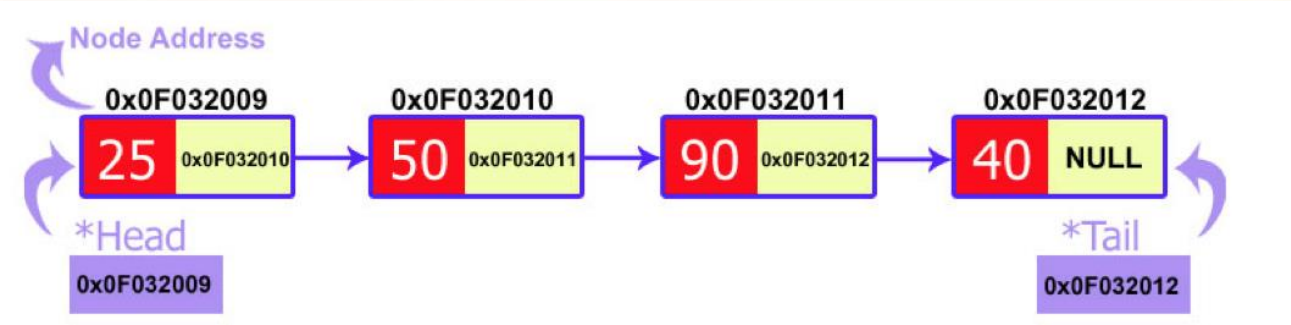
Pvz.: 9 yra dalus iš 3.

1. Kadangi seka $r_1, r_2, \ldots, r_{n-1}, r_n$ yra mažėjanti ir jos nariai priklauso neneigiamų sveikųjų skaičių aibei, todėl $r_n = 0$.

2. $r_{n-2} = r_{n-1}q_n + 0$ iš čia seka, kad r_{n-2} yra dalus iš r_{n-1}; $r_{n-3} = r_{n-2}q_{n-1} + r_{n-1}$ iš čia seka, kad r_{n-3} yra dalus iš r_{n-1}; kylant aukštyn įrodome, kad a yra dalus iš r_{n-1} ir b yra dalus iš r_{n-1} , t. y. r_{n-1} yra skaičių a ir b bendras daliklis.

3. Įrodysime, kad jis r_{n-1} yra didžiausias bendras skaičių a ir b daliklis. Tarkime priešingai, kad egzistuoja toks skaičius $R = DBD(a,b) > r_{n-1}$.
$a = bq_1 + r_1$ iš čia seka, kad r_1 yra dalus iš R,
$b = r_1q_2 + r_2$ iš čia seka, kad r_2 yra dalus iš R,
Leidžiantis į apačią įrodome, kad r_{n-1} yra dalus iš R, bet tai yra prieštara teiginiui $R > r_{n-1}$.
Išvada: $r_{n-1} = R = DBD(a,b)$.

Tiesinio sąrašo realizacija (1)



Sąrašo elemento struktūros aprašymas

```
//C++:
#include<iostream>
using namespace std;
struct node
{
    int data;
    node *next;
};
```

```
#Python:
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None
```

Tiesinio sąrašo realizacija (2)

Sąrašo funkcijų aprašymas

```
//C++:
class list
{
    private:
    node *head, *tail;
    public:
    list()
    {
        head=NULL;
        tail=NULL;
    }
    //void append(int value)
    //void push(int value)
    //void insertAfter(int pos, int value)
    //void delete_first()
    //void delete_last()
    //void deleteNode (int pos)
    //void display()
};
```

```
#Python:
class LinkedList:
    def __init__(self):
        self.head = None
    #def append(self, new_data):
    #def push(self, new_data):
    #def insertAfter(self, prev_node, new_data):
    #def deleteNode(self, key):
    #def display(self):
```

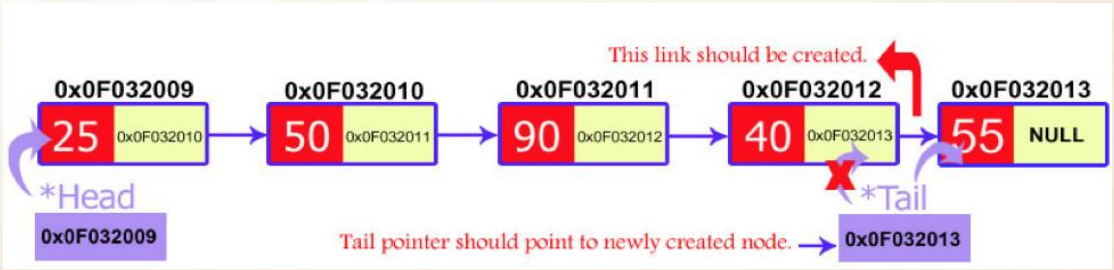
Tiesinio sąrašo realizacija (3)

Elemento pridėjimas sąrašo gale

```
//C++:
void append(int value)
      {
         node *temp=new node;
         temp->data=value;
         temp->next=NULL;
         if(head==NULL)
         {
            head=temp;
            tail=temp;
            temp=NULL;
         }
         else
         {
            tail->next=temp;
            tail=temp;
         }
      }
```

```
#Python:

 def append(self, new_data):
       new_node = Node(new_data)
       if self.head is None:
          self.head = new_node
          return
       last = self.head
       while (last.next):
          last = last.next
       last.next =  new_node
```



Tiesinio sąrašo realizacija (4)

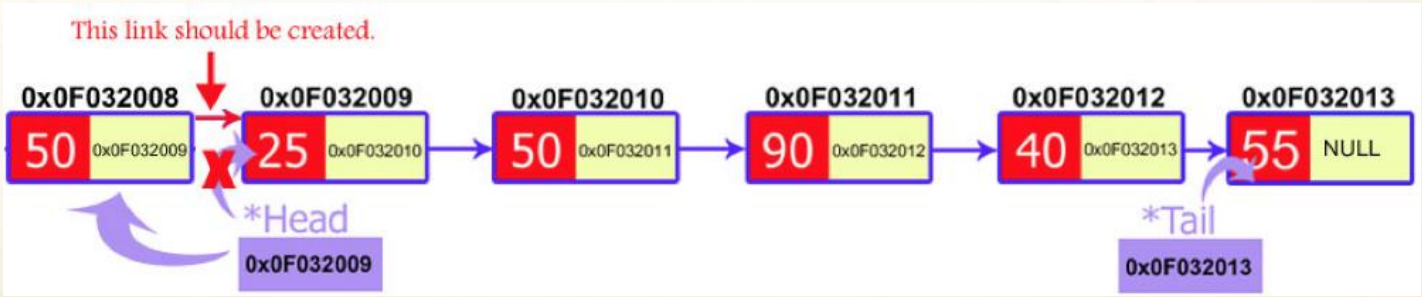
Elemento pridėjimas sąrašo priekyje

//C++:

```
void push(int value)
    {
        node *temp=new node;
        temp->data=value;
        temp->next=head;
        head=temp;
    }
```

#Python:

```
def push(self, new_data):
        new_node = Node(new_data)
        new_node.next = self.head
        self.head = new_node
```



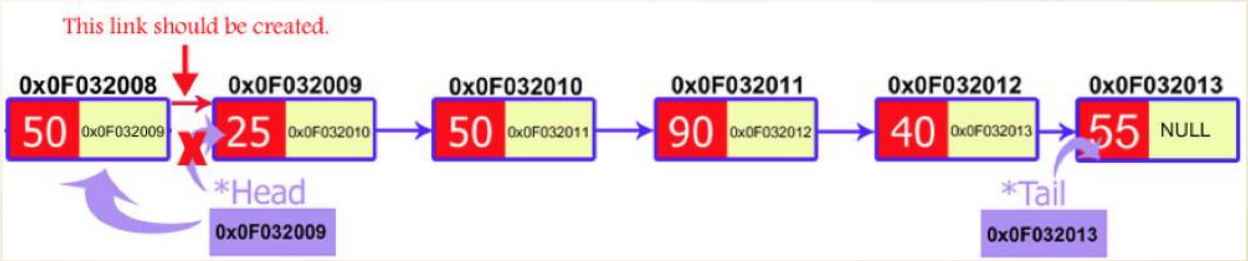
Tiesinio sąrašo realizacija (5)

Elemento pridėjimas sąrašo viduryje

```
//C++:
 void insertAfter(int pos, int value)
      {
          node *pre=new node;
          node *cur=new node;
          node *temp=new node;
          cur=head;
          for(int i=1;i<=pos;i++)
          {
              pre=cur;
              cur=cur->next;
          }
          temp->data=value;
          pre->next=temp;
          temp->next=cur;
      }
```

```
#Python:

 def insertAfter(self, prev_node, new_data):
        if prev_node is None:
            print ("The given previous node must inLinkedList.")
            return
        new_node = Node(new_data)
        new_node.next = prev_node.next
        prev_node.next = new_node
```

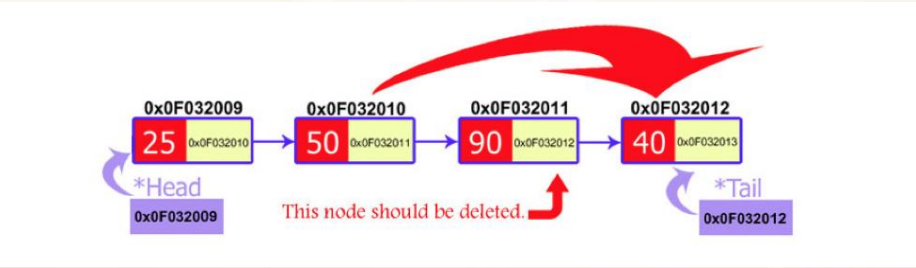


Tiesinio sąrašo realizacija (6)

Elemento šalinimas sąrašo viduryje

//C++:

```
void delete_position(int pos)
    {
        node *current=new node;
        node *previous=new node;
        current=head;
        for(int i=1;i<pos;i++)
        {
            previous=current;
            current=current->next;
        }
        previous->next=current->next;
    }
```



#Python:

```
def deleteNode(self, position):

    if self.head == None:
        return
    temp = self.head
    if position == 0:
        self.head = temp.next
        temp = None
        return
    for i in range(position -1 ):
        temp = temp.next
        if temp is None:
            break
    if temp is None:
        return
    if temp.next is None:
        return
    next = temp.next.next
    temp.next = None
    temp.next = next
```

Tiesinio sąrašo realizacija (7)

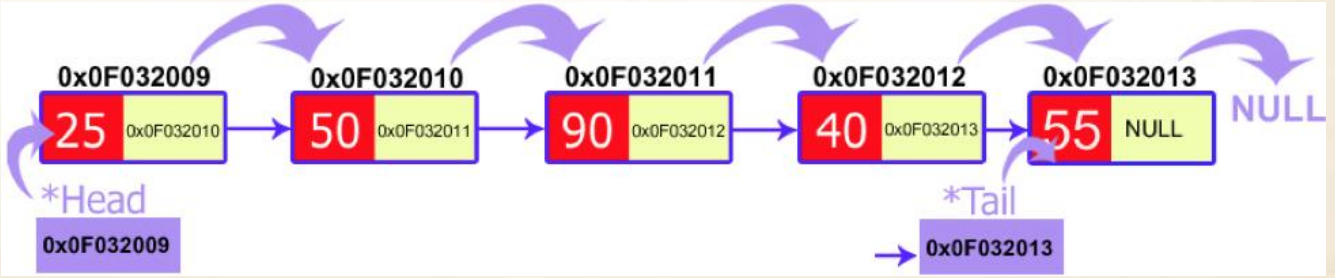
Sąrašo elementų spausdinimas

```
//C++:

 void display()
      {
         node *temp=new node;
         temp=head;
         while(temp!=NULL)
         {
            cout<<temp->data<<"\t";
            temp=temp->next;
         }
      }
```

```
#Python:

 def display(self):
        temp = self.head
        while(temp):
            print (" %d" %(temp.data)),
            temp = temp.next
```



Tiesinio sąrašo realizacija (8)

Programos testavimas

```
//C++:

int main()
{
        list obj;
        obj.append(6);
        obj.push(7);
        obj.push(1);
        obj.append(4);
        obj.insertAfter(2,8);
        obj.display();
        system("pause");
        return 0;
}
```

```
#Python:

llist = LinkedList()
llist.append(6)
llist.push(7);
llist.push(1);
llist.append(4)
llist.insertAfter(llist.head.next, 8)
print ("Created linked list is:"),
llist.display()
```

Rezultatas: 1 7 8 6 4

Informacija paimta iš

C++:
https://www.codementor.io/codementorteam/a-comprehensive-guide-to-implementation-of-singly-linked-list-using-c_plus_plus-ondlm5azr

Python:
https://www.geeksforgeeks.org/data-structures/linked-list/

Naudingos pamokos:
https://www.youtube.com/watch?v=o5wJkJJpKtM

Buvo teoriškai, dabar praktiškai...

```
// C++ stack::push/pop
#include <iostream>       // std::cout
#include <stack>          // std::stack

int main ()
{
  std::stack<int> mystack;

  for (int i=0; i<5; ++i) mystack.push(i);

  std::cout << "Popping out elements...";
  while (!mystack.empty())
  {
     std::cout << ' ' << mystack.top();
     mystack.pop();
  }
  std::cout << '\n';

  return 0;
}
```

// Paimta iš: http://www.cplusplus.com/reference/stack/stack/push/

Buvo teoriškai, dabar praktiškai...

```
# Python
stack = [3, 4, 5]
stack.append(6)
stack.append(7)
stack
[3, 4, 5, 6, 7]
stack.pop()
# 7
stack
# [3, 4, 5, 6]
stack.pop()
# 6
stack.pop()
# 5
stack
# [3, 4]

# Paimta iš: https://docs.python.org/2/tutorial/datastructures.html
```

C++ duomenų tipai

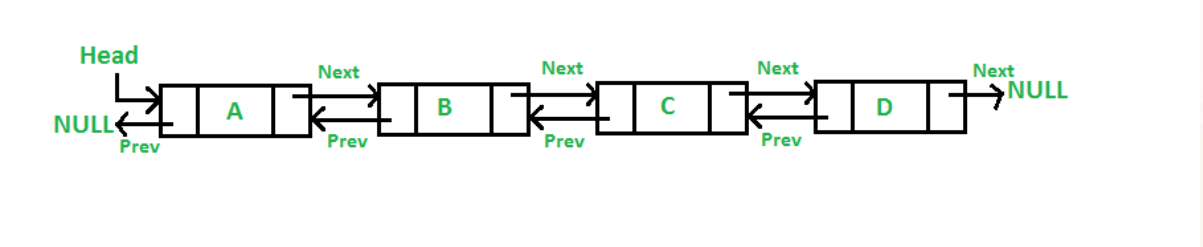
Type	Typical Bit Width	Typical Range
char	1byte	-127 to 127 or 0 to 255
unsigned char	1byte	0 to 255
signed char	1byte	-127 to 127
int	4bytes	-2147483648 to 2147483647
unsigned int	4bytes	0 to 4294967295
signed int	4bytes	-2147483648 to 2147483647
short int	2bytes	-32768 to 32767
unsigned short int	Range	0 to 65,535
signed short int	Range	-32768 to 32767
long int	4bytes	-2,147,483,648 to 2,147,483,647
signed long int	4bytes	same as long int
unsigned long int	4bytes	0 to 4,294,967,295
float	4bytes	+/- 3.4e +/- 38 (~7 digits)
double	8bytes	+/- 1.7e +/- 308 (~15 digits)
long double	8bytes	+/- 1.7e +/- 308 (~15 digits)
wchar_t	2 or 4 bytes	1 wide character

Python duomenų tipai

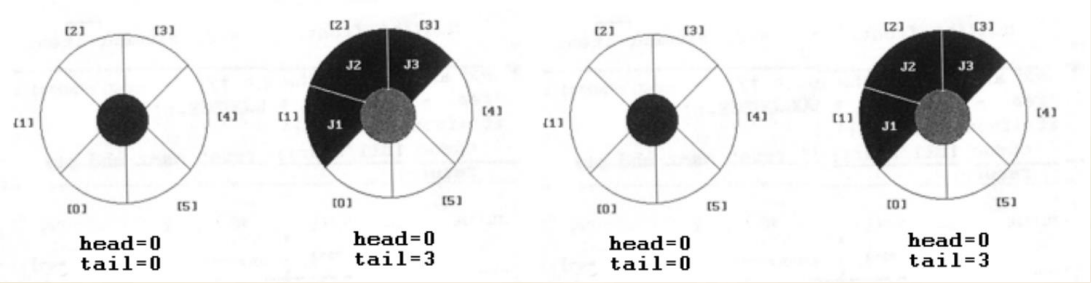
Class	Description
bool	Boolean value
int	integer (arbitrary magnitude)
float	floating-point number
list	mutable sequence of objects
tuple	immutable sequence of objects
str	character string
set	unordered set of distinct objects
frozenset	immutable form of set class
dict	associative mapping (aka dictionary)

Tiesinio sąrašo modifikacijos

- Dėklas (arba stekas, angl. *stack*)
- Eilutė (arba eilė, angl. *queue*)
- Abipusis dėklas (arba dekas, angl. *dequeue*)
- Dvipusis tiesinis sąrašas (doubly linked list)



- Ciklinis sąrašas (vienpusis arba dvipusis) – sarašas, kuriame po paskutiniojo elemento seka pirmasis.
- Eilutė „suklijuotame“ masyve ir t. t.



Tiesinių duomenų struktūrų privalumai ir trūkumai

Jei realizuota panaudojant masyvą.

Privalumai:

- paprasta struktūra;
- paprasta operacijų realizacija;
- laikomi tik patys duomenys (nereikia naudoti rodyklių).

Trūkumai:

- masyvo (tiesinio sąrašo ar jo modifikacijos) dydis turi būti nusakytas iš anksto, todėl gali būti arba naudojama tik nedidelė jo dalis, arba pritrūkti vietos.
- elemento įterpimo/naikinimo operacijos tiesinių duomenų struktūrų atveju efektyvios.

Jei realizuota panaudojant rodykles.

Privalumai:

- atmintis naudojama tik egzistuojantiems steko elementams, t.y. nerezervuojama
- vieta potencialiems elementams.

Trūkumai:

- operacijų realizacija sudėtingesnė (reikia mokėti dirbti su rodyklėmis);
- reikia daugiau atminties, nes saugomi ne tik duomenys, bet ir nuorodos (rodyklės).

ADT matricos operacijos

- *Inicializuoti matricą (M)* – kompiuterio atmintyje išskiria matricai *M* vietą,
- *Surasti elementą (i, j, M)* – surasti elementą matricos *M i*-toje eilutėje ir j-ame stulpelyje,
- *Priskirti elementą (i, j, x, M)* – matricos *M i*-tos eilutės ir *j*-ojo stulpelio elementui priskiria reikšmę *x*,
- *Priskirti (M1, M2)* – priskiria matricos *M1* elementus matricos *M2* elementams, loginė sąlyga: matricų *M1* ir *M2* matavimai turi sutapti,
- *Sudėtis (M1, M2)* – matricų *M1* ir *M2* atitinkamų elementų sudėtis, loginė sąlyga: matricų *M1* ir *M2* matavimai turi sutapti,
- *Neigimas (M)* – skaičiuoja matricos *M* neigiamą reikšmę,
- *Atimtis (M1, M2)* – iš matricos *M2* atimama matrica *M1*, loginė sąlyga: matricų *M1* ir *M2* matavimai turi sutapti,
- *Daugyba iš skaliaro (s, M)* – matrica *M* padauginama iš skaliaro *s*,
- *Daugyba (M1, M2)* – sudaugina matricas *M1* ir *M2*, loginė sąlyga: matricos *M1* stulpelių skaičius turi būti lygus matricos *M2* eilučių skaičiui, gautoji matrica turi tiek eilučių kiek *M1* ir tiek stulpelių kiek *M2*,
- *Transponuoti (M)* – transponuoja matricą *M*,
- *Determinantas (M)* – skaičiuoja matricos *M* determinantą,
- *Atvirkštinė (M)* – skaičiuoja matricos *M* atvirkštinę matricą (jei ši egzistuoja) arba apibendrintą atvirkštinę (jei matricos determinantas lygus 0),
- *Ištrinti (M)* – palaisvina atmintį, kurią buvo užėmusi matrica *M*.

Aibės duomenų struktūra

Aibės duomenų struktūrai būdingos tokios operacijos:

- *initialize(S)* – inicializuoti aibę *S* (išskirti vietą atmintyje ir parinkti dėstymo metodą joje),
- *memberof(x,S)* – funkcija, kuri patikrina ar elementas *x* priklauso aibei *S*,
- *insert(x,S)* – įterpti elementą *x* į aibę *S*,
- *delete(x,S)* – pašalinti elementą *x* iš aibės *S*,
- *join(S1,S2,S3)* – sujungti dvi aibes *S1* ir *S2* į vieną *S3*,
- *find(x)* - rasti aibę, kurioje yra elementas *x*,
- *disjoin(x,S)* – aibę *S* išskaidyti į dvi aibes, iš kurių vienoje yra visi elementai, mažesni ar lygūs *x*, o kitoje - didesni už *x* (paprastai aibė *S* yra tiesiškai sutvarkyta),
- *min(S)* – rasti minimalų aibės *S* elementą.

Veiksmai su aibėmis *A* ir *B*:

- ar teisybė, kad dvi aibės *A* ir *B* yra lygios (arba ne),
- ar teisybė, kad aibė *A* yra aibės *B* poaibis (tuščia aibė yra kiekvienos aibės poaibis),
- suformuoti aibių *A* ir *B* sąjungą,
- suformuoti aibių *A* ir *B* sankirtą,
- suformuoti aibių *A* ir *B* skirtumą,
- suformuoti aibių *A* ir *B* Dekarto sandaugą.

ADT dinaminė aibė

Dinaminėje aibėje kiekvienas elementas turi savo ID (raktą), kuriems aibė yra visiškai sutvarkyta.
initialize(S) – inicializuoti aibę *S* (išskirti vietą atmintyje ir parinkti dėstymo metodą joje).

- *Search(S,k)* – paieška*(S, k)* – grąžina aibės *S* elementą, kurio reikšmė yra k, arba nulinę reikšmę jei tokio elemento nėra,
- *Insert(S,x)* – *įterpti(S, x)* – įterpia elementą x į *S*, jei operacija sėkminga grąžina reikšmę true, jei ne – false,
- *Delete(S,k)* – *ištrinti(S, k)* – iš aibės *S* pašalina elementą, kurio reikšmė k, jei operacija sėkminga grąžina reikšmę true, jei ne – false,
- *Minimum(S)* – *minimumas(S)* – grąžina dinaminės aibės *S* elementą, kurio reikšmė yra mažiausia, arba nulinę reikšmę jei aibė *S* yra tuščia,
- *Maximum(S)* – *maksimumas(S)* – grąžina dinaminės aibės *S* elementą, kurio reikšmė yra didžiausia, arba nulinę reikšmę jei aibė *S* yra tuščia,
- *Predecessor(S,k)* – *pirmtakas(S, k)* – grąžina aibės *S* elementą, kurio reikšmė didžiausia, bet mažesnė už k arba nulinę reikšmę, jei tokių elementų nėra,
- *Successor(S,k)* – *įpėdinis (S, k)* – grąžina aibės *S* elementą, kurio reikšmė mažiausia, bet didesnė už *k* arba nulinę reikšmę, jei tokių elementų nėra,
- *Empty(S)* – tikrina ar aibė yra tuščia, grąžina loginę reikšmę true, jei dinaminė aibė *S* yra tuščia, arba false jei *S* yra netuščia,
- *MakeEmpty(S*) – ištrina visus aibės *S* elementus, todėl *S* tampa tuščia dinamine aibe.

Medžiai

Medis (tree) – tai duomenų aibė, kuriai apibrėžtos duomenų incidentumo ir eiliškumo operacijos:

- medžio (žymime T) elementai yra dvi aibės: viršūnių aibė V ir briaunų aibė E,
- tarp V ir E apibrėžtas incidentumo santykis: viena briauna (elementas iš E) atitinka dvi ir tik dvi viršūnes (elementų iš V porą, elementų tvarka poroje yra fiksuota),
- viena viršūnė medyje yra išskiriama ir vadinama šaknimi (šakninis medis grafų teorijoje),
- bet kurias dvi viršūnes jungia vienas ir tik vienas kelias (kelias yra briaunų seka, kurioje kiekvienos dvi gretimos briaunos turi bendrą viršūnę).

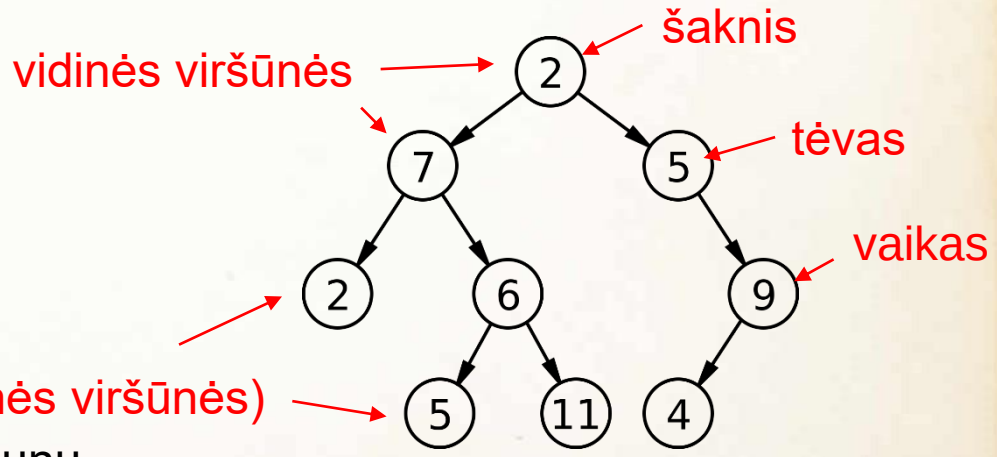
Manipuliavimo su duomenimis operacijos tokios:

- inicializuoti medį (išskirti vietą būsimai medžio struktūrai kompiuterio atmintyje)
- įterpti elementą į medį (įterpti naują viršūnę),
- išmesti elementą iš medžio (pašalinti viršūnę),
- sujungti du medžius,
- išskaidyti medį į du medžius,
- sutvarkyti medį (išdėstyti medžio viršūnes taip, kad pagal tam tikrą viršūnių
- perrinkimo tvarką, jų reikšmės atitiktų norimas sąlygas),
- panaikinti medį (išlaisvinti medžiui skirtą kompiuterio atmintyje vietą).

Dvejetainiai medžiai

Dvejetainis medis (*binary tree*) – tai toks medis, kurio kiekviena viršūnė turi ne daugiau kaip 2 vaikus, kurie vadinami dešiniuoju ir kairiuoju medžio pomedžiu.

Pilnas dvejetainis medis – medis, kurio visos viršūnės arba neturi vaikų, arba jų turi 2.

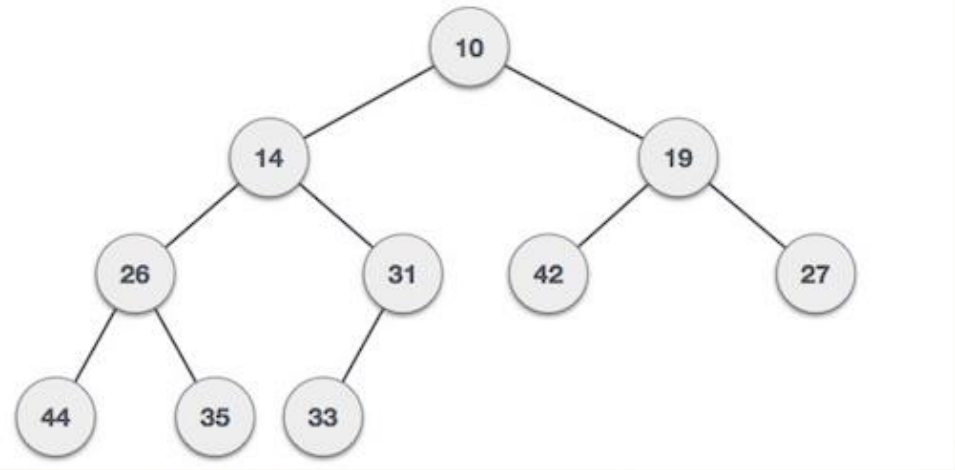


Medžio savybės:

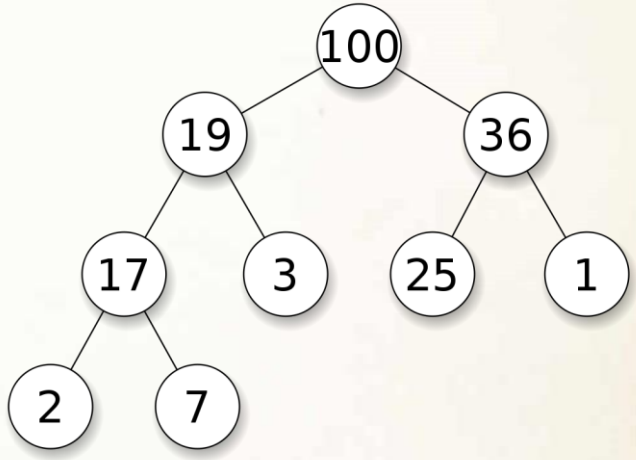
1. Medis su N viršūnių turi N-1 briaunų.
2. Pilnas dvejetainis medis su N vidinių viršūnių, turi N+1 išorinių viršūnių.
3. Pilno dvejetainio medžio aukštis n su N vidinių viršūnių gali būti įvertinamas skaičiumi $\log_2 N$:

$$2^{n-1} < N + 1 \leq 2^{n}.$$

Medžių pavyzdžiai



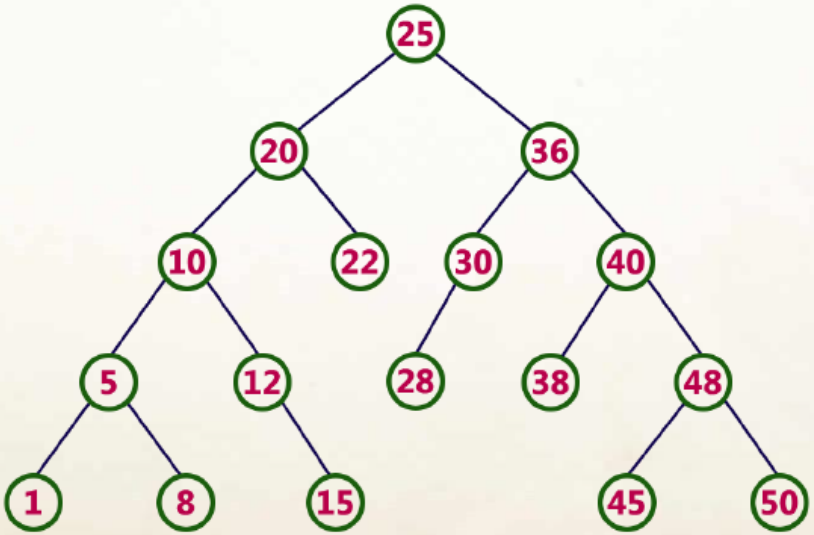
Dvejetainis medis



Pilnas dvejetainis medis

Dvejetainis paieškos medis

- Duomenų struktūra dvejetainis paieškos medis binary search tree tai toks dvejetainis medis , kurio viršūnių reikšmės išdėstytomos pagal šias taisykles:
  - Viršūnės reikšmė yra didesnė už visas reikšmes jos kairiajame pomedyje.
  - Viršūnės reikšmė yra mažesnė už visas reikšmes jos dešiniajame pomedyje.
  - Kairysis ir dešinysis pomedžiai yra dvejetainiai paieškos medžiai
  - Viršūnių reikšm ė s negali kartotis.



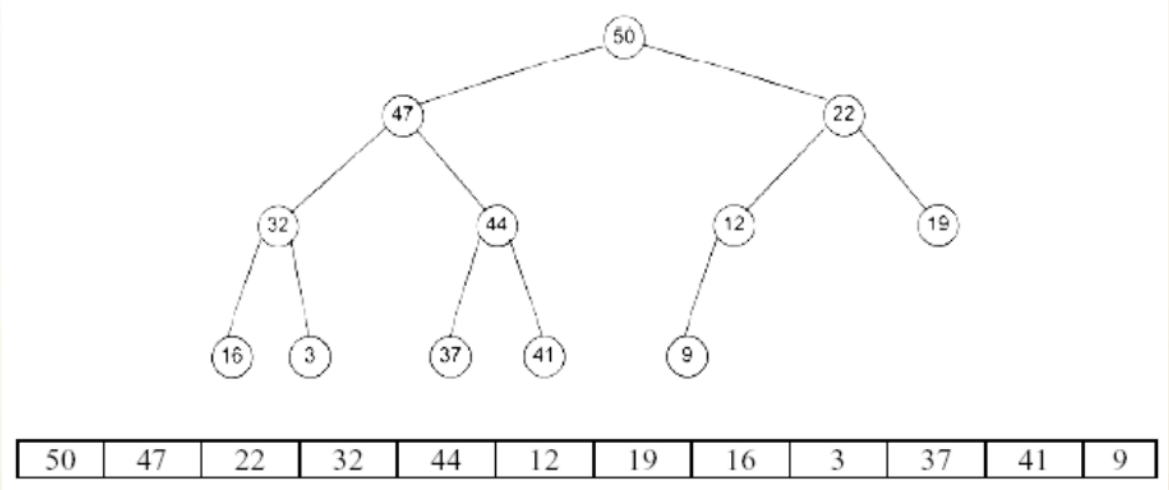
Operacijos dvejetainiame paieškos medyje

- Medžio apėjimas (visų reikšmių išvedimas)
- Reikšmės paieška
- Elemento įterpimas
- Elemento šalinimas

Duomenų struktūra heap

Heap struktūra – tai dvejetainis medis, saugomas vienmačiame masyve, kuriame nuorodos nukreiptos iš tėvo viršūnės į sūnus. Medis turi būti sutvarkytas pagal lygius, o kiekvienai viršūnei patenkinama heap sąlyga:

- **tėvo reikšmės turi būti didesnės (arba mažesnės) už jo vaikų reikšmes.**



Heap medis ir jį atitinkantis masyvas.

Prioritetinė eilutė

Duomenų struktūra, vadinama prioritetine eilute, pritaikyta operacijoms:

- inicializuoti aibę iš N elementų,
- įterpti naują elementą (insert),
- išmesti didžiausią elementą (remove),
- pakeisti didžiausią elementą nauju, jei šis nėra didžiausias (replace),
- pakeisti elemento prioritetą (change),
- išmesti bet kurį nurodytą elementą (delete),
- sujungti dvi prioritetines eiles į vieną (join).

Operacijų pavyzdžiai

operation	argument	return value	size	contents (unordered)	contents (ordered)
insert	P		1	P	P
insert	Q		2	P Q	P Q
insert	E		3	P Q E	E P Q
remove max		Q	2	P E	E P
insert	X		3	P E X	E P X
insert	A		4	P E X A	A E P X
insert	M		5	P E X A M	A E M P X
remove max		X	4	P E M A	A E M P
insert	P		5	P E M A P	A E M P P
insert	L		6	P E M A P L	A E L M P
insert	E		7	P E M A P L E	A E E L M
remove max		P	6	E E M A P L	A E E L M

A sequence of operations on a priority queue

Ačiū už dėmesį.

Klausimai?