NETIESINĖS DUOMENŲ STRUKTŪROS

Raimondas Čiegis

Matematinio modeliavimo katedra, e-paštas: rc@vgtu.lt

Rugsėjo 1 d., 2023

Dvejetainis medis

Pradedame nagrinėti sudėtingesnes dinamines struktūras, kurios apibrėžia daugiamačius sąryšius.

Dvejetainis medis

Pradedame nagrinėti sudėtingesnes dinamines struktūras, kurios apibrėžia daugiamačius sąryšius.

Pateiksime nepriklausomą dvejetainio medžio apibrėžimą.

Tarkime, turime elementų aibę D.

Dvejetainių medžių (angl. *binary tree*) aibei T priklauso:

- tuščioji aibė;

Dvejetainis medis

Pradedame nagrinėti sudėtingesnes dinamines struktūras, kurios apibrėžia daugiamačius sąryšius.

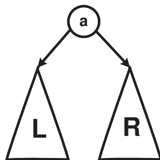
Pateiksime nepriklausomą dvejetainio medžio apibrėžimą.

Tarkime, turime elementų aibę D.

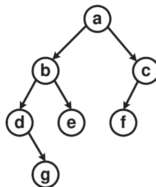
Dvejetainių medžių (angl. *binary tree*) aibei T priklauso:

- tuščioji aibė;
- viena viršūnė $a \in D$;

- visos aibės, sudarytos iš viršūnės $a \in D$, sujungtos su rūšiuota pora (L, R), kur L ir R yra dvejetainiai medžiai (žr. *a* paveikslą).
  Tada *a* vadinama medžio šaknimi, o L ir R – medžio T kairiuoju ir dešiniuoju pomedžiais.



a)



b)

Dvejetainio medžio pavyzdys pateiktas *b* paveiksle.

Medžio viršūnes žymėsime $v_j \in V$.

Medžio viršūnes žymėsime $v_j \in V$.

Jei viršūnė v_j yra sujungta su kita viršūne v_k briauna $e_{jk} = (v_j, v_k) \in E$, tai v_k vadinama viršūnės v_j *vaiku*, o pati v_j – viršūnės v_k *tėvu*.

Medžio viršūnes žymėsime $v_j \in V$.

Jei viršūnė v_j yra sujungta su kita viršūne v_k briauna $e_{jk} = (v_j, v_k) \in E$, tai v_k vadinama viršūnės v_j *vaiku*, o pati v_j – viršūnės v_k *tėvu*.

Viršūnės, kurios neturi vaikų, vadinamos lapais.

Medžio viršūnes žymėsime $v_j \in V$.

Jei viršūnė v_j yra sujungta su kita viršūne v_k briauna $e_{jk} = (v_j, v_k) \in E$, tai v_k vadinama viršūnės v_j *vaiku*, o pati v_j – viršūnės v_k *tėvu*.

Viršūnės, kurios neturi vaikų, vadinamos lapais.

Medžio šaknis yra nulinio lygmens viršūnė. Tada k-tojo lygmens viršūnės vaiko lygmuo yra $(k+1)$-tasis.

Medžio viršūnes žymėsime $v_j \in V$.

Jei viršūnė v_j yra sujungta su kita viršūne v_k briauna $e_{jk} = (v_j, v_k) \in E$, tai v_k vadinama viršūnės v_j *vaiku*, o pati v_j – viršūnės v_k *tėvu*.

Viršūnės, kurios neturi vaikų, vadinamos lapais.

Medžio šaknis yra nulinio lygmens viršūnė. Tada k-tojo lygmens viršūnės vaiko lygmuo yra $(k+1)$-tasis.

Medžio aukštis yra lygus didžiausiam viršūnės lygmeniui.

Medžio viršūnes žymėsime $v_j \in V$.

Jei viršūnė v_j yra sujungta su kita viršūne v_k briauna $e_{jk} = (v_j, v_k) \in E$, tai v_k vadinama viršūnės v_j *vaiku*, o pati v_j – viršūnės v_k *tėvu*.

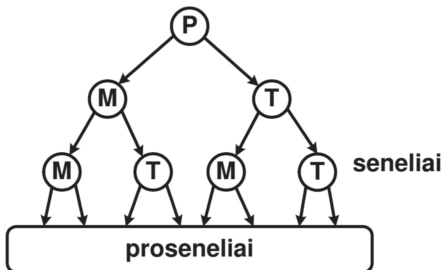
Viršūnės, kurios neturi vaikų, vadinamos lapais.

Medžio šaknis yra nulinio lygmens viršūnė. Tada k-tojo lygmens viršūnės vaiko lygmuo yra $(k+1)$-tasis.

Medžio aukštis yra lygus didžiausiam viršūnės lygmeniui.

Medžio briaunoms gali būti suteiktas svoris, toks medis vadinamas svertiniu.

Genealoginis medis



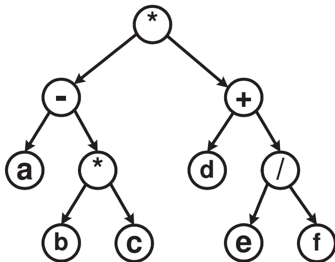
Infix aritmetinės išraiškos užrašymas dvejetainiame medyje

Nagrinėkime aritmetinę išraišką $(a - b * c) * (d + e/f)$.

Infix aritmetinės išraiškos užrašymas dvejetainiame medyje

Nagrinėkime aritmetinę išraišką $(a - b * c) * (d + e/f)$.

Ją užrašome dvejetainiame medyje (nereikia naudoti skliaustų)

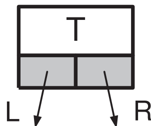


Pirmiausia apibrėžiame vieną atskirą *elementą* (angl. *node*), kurį sudaro informacinė dalis T ir dvi rodyklės, rodančios į du kitus *elementus*

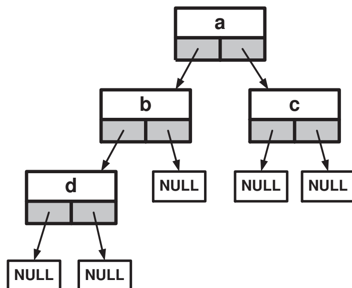
$$
\begin{aligned}
&struct\ node\, \{ \\
&\quad T\ data; \\
&\quad node *\ left; \\
&\quad node *\ right; \\
&\}
\end{aligned}
$$

Pirmiausia apibrėžiame vieną atskirą *elementą* (angl. *node*), kurį sudaro informacinė dalis T ir dvi rodyklės, rodančios į du kitus *elementus*

$$
\begin{aligned}
& struct\ node\,\{ \\
& \quad T\ data; \\
& \quad node *\ left; \\
& \quad node *\ right; \\
& \}
\end{aligned}
$$



Sujungdami šiuos elementus, sudarome dvejetainį medį.



Dažnai yra naudinga medžio elementą papildyti dar dviem atributais – raktu *key*, charakterizuojančiu saugomą informaciją, ir rodykle, rodančia į elemento tėvą:

$$
\begin{aligned}
& struct\ node\,\{ \\
& \quad T\ data; \\
& \quad int\ key; \\
& \quad node *\ left; \\
& \quad node *\ right; \\
& \quad node *\ p; \\
& \}
\end{aligned}
$$

Tokia informacija pagelbėja realizuojant įvairias operacijas su medžio elementais (panašią pastabą padarėme ir nagrinėdami vienakrypčius tiesinius sąrašus) .

Kaip ir kiekvienos duomenų struktūros atveju apibrėžiame pagrindines operacijas

a) įterpti naują viršūnę;

b) pašalinti viršūnę iš medžio;

c) patikrinti ar tokia viršūnė yra saugoma medyje;

Kaip ir kiekvienos duomenų struktūros atveju apibrėžiame pagrindines operacijas

a) įterpti naują viršūnę;

b) pašalinti viršūnę iš medžio;

c) patikrinti ar tokia viršūnė yra saugoma medyje;

Šį sąrašą papildysime keliomis naudingomis operacijomis, kurios ypač svarbios nagrinėjant paieškos medžius:

d) MINIMUM – surasti mažiausią elementą;

e) SUCCESSOR – surasti sekantį elementą, surūšiuotoje aibėje.

Visiškai subalansuotas dvejetainis medis

Daugelio algoritmų, realizuojančių dvejetainių medžių veiksmus, sudėtingumas priklauso nuo medžio aukščio. Todėl siekiame, kad didėjant medžio viršūnių skaičiui, jo aukštis didėtų kuo lėčiau.

Visiškai subalansuotas dvejetainis medis

Daugelio algoritmų, realizuojančių dvejetainių medžių veiksmus, sudėtingumas priklauso nuo medžio aukščio. Todėl siekiame, kad didėjant medžio viršūnių skaičiui, jo aukštis didėtų kuo lėčiau.

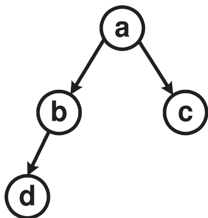
Tada reikia derinti kiekvieno elemento kairiojo ir dešiniojo pomedžių sudarymą.

Visiškai subalansuotas dvejetainis medis

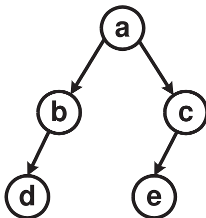
Daugelio algoritmų, realizuojančių dvejetainių medžių veiksmus, sudėtingumas priklauso nuo medžio aukščio. Todėl siekiame, kad didėjant medžio viršūnių skaičiui, jo aukštis didėtų kuo lėčiau.

Tada reikia derinti kiekvieno elemento kairiojo ir dešiniojo pomedžių sudarymą.

Apibrėžimas.
Dvejetainis medis, kurio kiekvienos viršūnės kairiojo ir dešiniojo pomedžių viršūnių skaičiai skiriasi ne daugiau kaip viena, vadinamas *visiškai subalansuotu* medžiu.



a)



b)

Visiškai subalansuoti medžiai: a) $N = 4$, b) $N = 5$

Pateiksime algoritmą, kuris perkelia duomenis iš masyvo ar rinkmenos į visiškai subalansuotą dvejetainį medį. Tai rekursinis algoritmas.

$$
\begin{aligned}
& node * \ balancedTree(int\ N)\{ \\
& \quad y = NIL; \\
& \quad if\ (N == 0)\ return\ (NIL); \\
& \quad nL = N/2;\ nR = N - nL - 1; \\
& \quad x = read(); \\
& \quad node * \ Node = new(node); \\
& \quad Node.data = x; \quad Node.key = key; \\
& \quad Node.left = balancedTree(nL); \\
& \quad Node.right = balancedTree(nR); \\
& \quad Node.p = y, \quad y = Node; \\
& \quad return\ (Node); \\
& \}
\end{aligned}
$$

Dvejetainio medžio viršūnių apėjimo algoritmai

Susipažinsime su trimis svarbiais dvejetainio medžio viršūnių aplankymo algoritmais. Visi jie rekursiniai, o skiriasi tik pomedžių aplankymo tvarka.

Dvejetainio medžio viršūnių apėjimo algoritmai

Susipažinsime su trimis svarbiais dvejetainio medžio viršūnių aplankymo algoritmais. Visi jie rekursiniai, o skiriasi tik pomedžių aplankymo tvarka.

Jei medyje saugome aritmetinę išraišką, tai šiais algoritmais randame tris pagrindines aritmetinės išraiškos formas: *prefix*, *infix* ir *postfix*, todėl taip vadinsime ir atitinkamus medžio apėjimo algoritmus.

Prefix algoritmas. Pirmiausia aplankome viršūnę-šaknį, paskui jos kairįjį pomedį ir vėliausiai dešinįjį pomedį.

```
preOrder (node* v)
begin
  (1)  if  (v ≠ NIL ) then
  (2)        P(v);
  (3)        preOrder(v.left);
  (4)        preOrder(v.right);
       end if
end preOrder
```

Infix algoritmas. Pirmiausia aplankome kairįjį pomedį, paskui viršūnę-šaknį ir vėliausiai dešinįjį pomedį

inOrder (node* v)
begin
  (1) **if** (v $\neq$ NIL) **then**
  (2)   inOrder(v.left);
  (3)   P(v);
  (4)   inOrder(v.right);
    end if
end inOrder

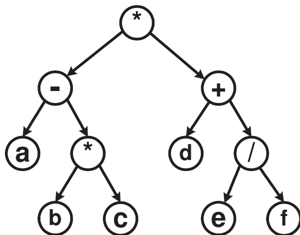
Postfix algoritmas. Pirmiausia aplankome kairįjį pomedį, paskui dešinįjį pomedį ir vėliausiai viršūnę-šaknį.

postOrder (node* v)
begin
(1) **if** ($v \neq$ NIL) **then**
(2) postOrder(v.left);
(3) postOrder(v.right);
(4) P(v);
end if
end postOrder

Pritaikykime šiuos algoritmus ir atspausdinkime aritmetinės išraiškos $(a - b * c) * (d + e/f)$ tris skirtingas užrašymo formas:

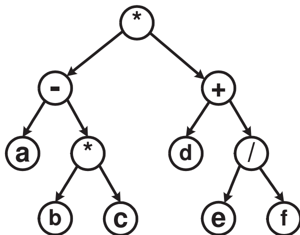
Pritaikykime šiuos algoritmus ir atspausdinkime aritmetinės išraiškos $(a - b * c) * (d + e/f)$ tris skirtingas užrašymo formas:

Informaciją keliame į dvejetainį medį:



Pritaikykime šiuos algoritmus ir atspausdinkime aritmetinės išraiškos $(a - b * c) * (d + e/f)$ tris skirtingas užrašymo formas:

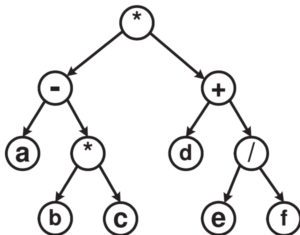
Informaciją keliame į dvejetainį medį:



- *Prefix* forma: $* - a * bc + d/ef$,

Pritaikykime šiuos algoritmus ir atspausdinkime aritmetinės išraiškos $(a - b * c) * (d + e/f)$ tris skirtingas užrašymo formas:

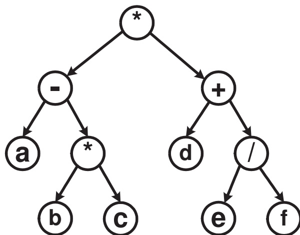
Informaciją keliame į dvejetainį medį:



- *Prefix* forma: $* - a * bc + d/ef$,
- *Infix* forma: $a - b * c * d + e/f$,

Pritaikykime šiuos algoritmus ir atspausdinkime aritmetinės išraiškos $(a - b * c) * (d + e/f)$ tris skirtingas užrašymo formas:

Informaciją keliame į dvejetainį medį:



- *Prefix* forma: $* - a * bc + d/ef$,
- *Infix* forma: $a - b * c * d + e/f$,
- *Postfix* forma: $abc * -def / + *$.

Dvejetainis paieškos medis

Labai dažnai tenka informaciją saugoti, rūšiuoti ir ieškoti. Sprendžiant šiuos uždavinius, svarbūs yra dvejetainiai paieškos medžiai (angl. *binary search tree*).

Dvejetainis paieškos medis

Labai dažnai tenka informaciją saugoti, rūšiuoti ir ieškoti. Sprendžiant šiuos uždavinius, svarbūs yra dvejetainiai paieškos medžiai (angl. *binary search tree*).

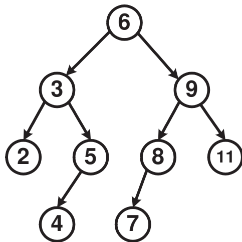
Apibrėžimas.
Paieškos medžio kiekvienoje viršūnėje esantis elementas yra didesnis už kairiojo pomedžio elementus ir nedidesnis už dešiniojo pomedžio elementus.

Dvejetainis paieškos medis

Labai dažnai tenka informaciją saugoti, rūšiuoti ir ieškoti. Sprendžiant šiuos uždavinius, svarbūs yra dvejetainiai paieškos medžiai (angl. *binary search tree*).

Apibrėžimas.

Paieškos medžio kiekvienoje viršūnėje esantis elementas yra didesnis už kairiojo pomedžio elementus ir nedidesnis už dešiniojo pomedžio elementus.



Paieškos medžiai leidžia labai efektyviai spręsti kai kuriuos dažnai sutinkamus uždavinius.

Paieškos medžiai leidžia labai efektyviai spręsti kai kuriuos dažnai sutinkamus uždavinius.

Aplankykime tokio medžio T viršūnes ir jas atspausdinkime naudodami metodą InOrder(T.root). Kokią išvadą galime padaryti?

Paieškos medžiai leidžia labai efektyviai spręsti kai kuriuos dažnai sutinkamus uždavinius.

Aplankykime tokio medžio T viršūnes ir jas atspausdinkime naudodami metodą InOrder(T.root). Kokią išvadą galime padaryti?

Taip, duomenis spausdiname didėjimo tvarka – taigi juos surūšiuojame.

Paieškos medžiai leidžia labai efektyviai spręsti kai kuriuos dažnai sutinkamus uždavinius.

Aplankykime tokio medžio T viršūnes ir jas atspausdinkime naudodami metodą InOrder(T.root). Kokią išvadą galime padaryti?

Taip, duomenis spausdiname didėjimo tvarka – taigi juos surūšiuojame.

Įrodykite, kad toks teiginys tikrai visada teisingas (rekomenduoju naudoti matematinės indukcijos metodą).

Spręskime kitą uždavinį: dvejetainiame paieškos medyje reikia rasti elementą, kurio raktas yra k. Jeigu tokio elemento nėra, tai grąžiname nuorodą NIL.

$$
\begin{aligned}
& node * \ \ Tree_Search(node * x, \ int \ k)\{ \\
& \quad if \ (x == NILL \ || \ k == x.key) \ return \ x; \\
& \quad if \ (k < x.key) \ return \ Tree_Search(x.left, \ k); \\
& \quad else \\
& \quad \quad return \ Tree_Search(x.right, \ k); \\
& \}
\end{aligned}
$$

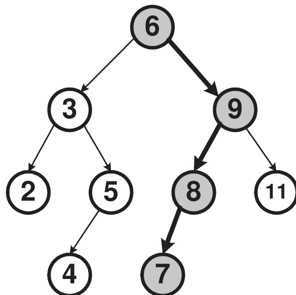
Spręskime kitą uždavinį: dvejetainiame paieškos medyje reikia rasti elementą, kurio raktas yra k. Jeigu tokio elemento nėra, tai grąžiname nuorodą NIL.

```
node * Tree_Search(node * x, int k){
  if (x == NILL || k == x.key) return x;
  if (k < x.key) return Tree_Search(x.left, k);
  else
    return Tree_Search(x.right, k);
}
```

Nors ir šio algoritmo struktūra yra panaši į viršūnių aplankymo algoritmų rekursinę struktūrą, bet paieškos atveju kiekviename žingsnyje pasirenkame tik vieną iš pomedžių.
Todėl algoritmo sudėtingumas priklauso tik nuo dvejetinio medžio aukščio $O(h)$.

Pateikiame ir iteracinę paieškos algoritmo versiją, kuri dažnai yra efektyvesnė už realizaciją, naudojančią rekursiją.

$$
\begin{array}{l}
node * \ \textit{Tree_Iterative_Search}(node * x, \ int \ k)\{ \\
\quad \text{while } (\, x \neq \ \text{NILL} \quad \text{and} \quad k \neq x.key \,) \\
\qquad \text{if } (k < x.key) \\
\qquad\quad x = x.left; \\
\qquad \text{else} \\
\qquad\quad x = x.right; \\
\quad return \ x; \\
\}
\end{array}
$$



Viršūnės, kurioje saugomas skaičius 7, paieškos kelias.

Užduotys

Sudarykite iteracinius algoritmus, realizuojančius šiuos dvejetainio paieškos medžio metodus:

Tree_Minimum_Search(node* T.root),

Tree_Maximum_Search(node* T.root).

Užduotys

Sudarykite iteracinius algoritmus, realizuojančius šiuos dvejetainio paieškos medžio metodus:

Tree_Minimum_Search(node* T.root),

Tree_Maximum_Search(node* T.root).

Ar tokiuose algoritmuose reikia lyginti elementų raktus?

Naujos viršūnės įterpimas

Norime į dvejetainį paieškos medį T įtepti viršūnę v, kurios atributų pradinės reikšmės yra tokios

$$v.key = k, \;\; v.left = NIL, \;\; v.right = NIL, \;\; v.p = NIL.$$

Naujoji viršūnė turi būti įterpta taip, kad ir atlikus šį veiksmą, medis išliktų dvejetainiu paieškos medžiu.

Naujos viršūnės įterpimas

Norime į dvejetainį paieškos medį T įtepti viršūnę v, kurios atributų pradinės reikšmės yra tokios

$$v.key = k, \quad v.left = NIL, \quad v.right = NIL, \quad v.p = NIL.$$

Naujoji viršūnė turi būti įterpta taip, kad ir atlikus šį veiksmą, medis išliktų dvejetainiu paieškos medžiu.

Vykdydami įterpimo algoritmą nebandome kontroliuoti medžio aukščio: tik surandame atitinkamą medžio šaką ir sukuriame naują viršūnę-lapą.

Naujos viršūnės įterpimas

Norime į dvejetainį paieškos medį T įtepti viršūnę v, kurios atributų pradinės reikšmės yra tokios

$$v.key = k, \quad v.left = NIL, \quad v.right = NIL, \quad v.p = NIL.$$

Naujoji viršūnė turi būti įterpta taip, kad ir atlikus šį veiksmą, medis išliktų dvejetainiu paieškos medžiu.

Vykdydami įterpimo algoritmą nebandome kontroliuoti medžio aukščio: tik surandame atitinkamą medžio šaką ir sukuriame naują viršūnę-lapą.

Jei pradinis medis T yra tuščias, tai ši viršūnė tampa jo šaknimi.

```
insert (tree T, node* v)
(1)  y = NIL,  x = T.root
(2)  while  ( x ≠ NIL )
(3)             y = x
(4)             if ( v.key < x.key )
(5)                x = x.left
(6)             else   x = x.right
(7)  v.p = y
(8)  if (y == NIL)
(9)             T.root = v
(9)  elseif (v.key < y.key)
(10)            y.left = v
(11) else  y.right = v
```

Pavyzdys

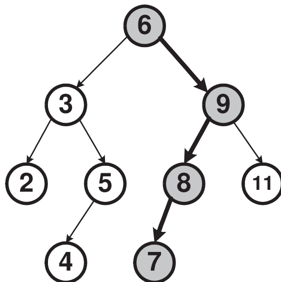
Sudarykite dvejetainį paieškos medį, kai turime tokią duomenų seką:

$$6,\ 9,\ 3,\ 5,\ 11,\ 4,\ 8,\ 2,\ 7.$$

Pavyzdys

Sudarykite dvejetainį paieškos medį, kai turime tokią duomenų seką:

6, 9, 3, 5, 11, 4, 8, 2, 7.



Viršūnės šalinimas

Pašalinti viršūnę iš paieškos medžio yra sudėtingiau – reikia garantuoti, kad ir atlikus šį veiksmą turėsime dvejetainį paieškos medį.

Viršūnės šalinimas

Pašalinti viršūnę iš paieškos medžio yra sudėtingiau – reikia garantuoti, kad ir atlikus šį veiksmą turėsime dvejetainį paieškos medį.

Sudarydami algoritmus mes vėl nesieksime subalansuoti gautojo paieškos medžio pomedžius taip, kad jo aukštis h irgi mažėtų. Tokių algoritmų sudarymas yra atskira svarbi tema.

Galimi trys atvejai:

Galimi trys atvejai:

1. Reikia pašalinti viršūnę v, kuri neturi vaikų. Tada šios višūnės tėvo $w = v.p$ atitinkamai rodyklei suteikiame reikšmę NIL:

$$w = v.p$$
$$\text{if } (T.root == v)\ T.root = NIL$$
$$\text{else}$$
$$\quad \text{if } (v.key < w.key)\ w.left = NIL$$
$$\quad \text{else } w.right = NIL$$
$$delete(v)$$

2. Reikia pašalinti viršūnę v, kuri turi vieną vaiką. Tada šios višūnės tėvo $w = v.p$ atitinkamai rodyklei suteikiame vaiko viršūnės reikšmę:

$w = v.p$
if $(v.left == NIL)$ $c = v.right$
else $c = v.left$
if $(T.root == v)$ $T.root = c$
else
  $c.p = w$
  if $(v.key < w.key)$ $w.left = c$
  else $w.right = c$
$delete(v)$

3. Reikia pašalinti viršūnę v, kuri turi abu vaikus.

3. Reikia pašalinti viršūnę v, kuri turi abu vaikus.

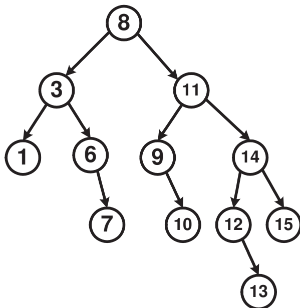
Pirmiausia sudarysime algoritmą, kuris suranda elemento v dešiniojo pomedžio mažiausią elementą:

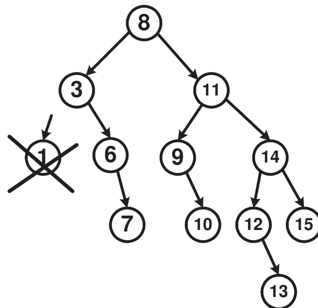
$$
\begin{aligned}
& node * \ Tree_Right_Minimum(node * x)\{ \\
& \quad y = x.right \\
& \quad s = y \\
& \quad \text{while } (y \neq \ \text{NILL} \ ) \\
& \qquad\qquad s = y \\
& \qquad\qquad y = y.left; \\
& \quad return\ s; \\
& \}
\end{aligned}
$$

Tada viršūnę v pašaliname tokiu algoritmu (patikriname, ar mažiausias elementas sutampa su v dešiniuoju vaiku) :

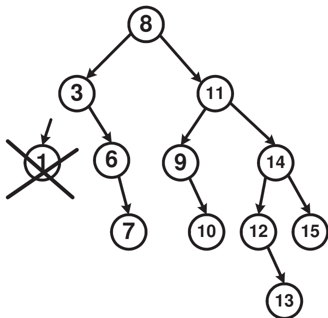
$$s = Tree_Right_Minimum(v)$$
$$w = v.p, \ z = s.p$$
$$\text{if } (z \neq v)$$
$$\quad z.left = s.right$$
$$\quad s.right = v.right$$
$$s.left = v.left$$
$$\text{if } (v.key < w.key)$$
$$\quad w.left = s$$
$$\text{else}$$
$$\quad w.right = s$$
$$delete(v)$$

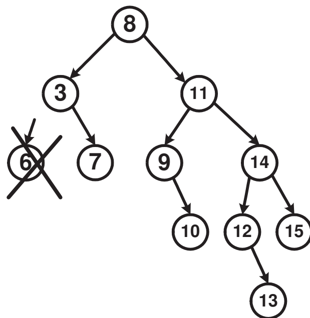
Imkime dvejetainį pieškos medį ir iš jo pašalinkime viršūnę–lapą 1.



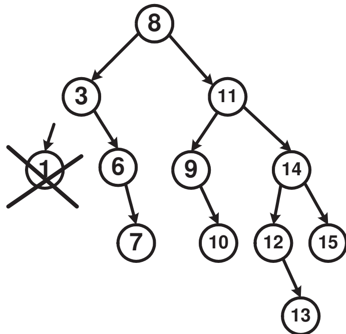


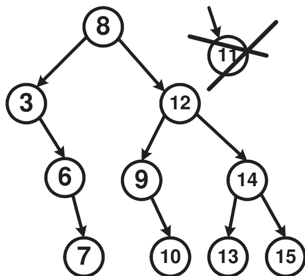
Iš gautojo dvejetainio medžio pašalinkime viršūnę, kuri turi tik vieną vaiką – 6.





Iš to pačio dvejetainio medžio pašalinkime viršūnę 11.





Paieškos algoritmo sudėtingumas

Kiek užtruks informacijos paieška dvejetainio medžio duomenų struktūroje?

Paieškos algoritmo sudėtingumas

Kiek užtruks informacijos paieška dvejetainio medžio duomenų struktūroje?

Bazine algoritmo operacija laikome dviejų raktų palyginimą.

Paieškos algoritmo sudėtingumas

Kiek užtruks informacijos paieška dvejetainio medžio duomenų struktūroje?

Bazine algoritmo operacija laikome dviejų raktų palyginimą.

Elemento paieškos sudėtingumas priklauso nuo dvejetainio medžio aukščio.

Paieškos algoritmo sudėtingumas

Kiek užtruks informacijos paieška dvejetainio medžio duomenų struktūroje?

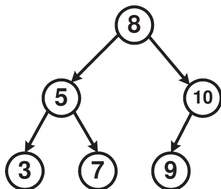
Bazine algoritmo operacija laikome dviejų raktų palyginimą.

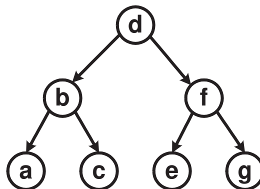
Elemento paieškos sudėtingumas priklauso nuo dvejetainio medžio aukščio.

Aišku, kad bet kuriam elementų išsidėstymui geriausio atvejo sudėtingumas yra lygus vienetui – ieškomasis elementas yra saugomas medžio šaknyje.

Palankiausias variantas, kai informacija yra saugoma idealiai subalansuotame paieškos medyje.

Palankiausias variantas, kai informacija yra saugoma idealiai subalansuotame paieškos medyje.





a) b)

Idealiai subalansuoti dvejetainiai paieškos medžiai: a) skaičių aibė $\{3, 5, 7, 8, 9, 10\}$, b) raidžių aibė $\{a, b, c, d, e, f, g\}$

Jeigu saugome N elementų, tai subalansuoto medžio aukštis yra $H = \log(N+1) - 1$.

Jeigu saugome N elementų, tai subalansuoto medžio aukštis yra $H = \log(N+1) - 1$.

Taigi net ir blogiausio atvejo paieškos sudėtingumas yra

$$W_b = \log(N+1).$$

Jeigu saugome N elementų, tai subalansuoto medžio aukštis yra $H = \log(N+1) - 1$.

Taigi net ir blogiausio atvejo paieškos sudėtingumas yra

$$W_b = \log(N+1).$$

Dažniausiai svarbu žinoti vidutinį (tikėtinąjį) paieškos algoritmo sudėtingumą, kai vienodai dažnai tikrinami visi N elementai. Gauname tokį įvertį

$$W_v = \frac{1}{N}\sum_{i=0}^{H} 2^i(i+1) = \log N + \mathcal{O}(1).$$

Nepalankiausiu atveju dvejetainis paieškos medis išsigimsta į tiesinį sąrašą. Taip bus, kai duomenis skaitome iš rinkmenos rakto didėjimo ar mažėjimo tvarka.

Nepalankiausiu atveju dvejetainis paieškos medis išsigimsta į tiesinį sąrašą. Taip bus, kai duomenis skaitome iš rinkmenos rakto didėjimo ar mažėjimo tvarka.

Tokiame paieškos medyje vidutinis paieškos algoritmo sudėtingumas yra

$$W_v = \frac{1}{N} \sum_{i=1}^{N} i = \frac{(N+1)}{2} = \frac{N}{2} + \mathcal{O}(1) .$$

Nepalankiausiu atveju dvejetainis paieškos medis išsigimsta į tiesinį sąrašą. Taip bus, kai duomenis skaitome iš rinkmenos rakto didėjimo ar mažėjimo tvarka.

Tokiame paieškos medyje vidutinis paieškos algoritmo sudėtingumas yra

$$W_v = \frac{1}{N} \sum_{i=1}^{N} i = \frac{(N+1)}{2} = \frac{N}{2} + \mathcal{O}(1) \,.$$

Blogiausiojo atvejo sudėtingumas $W_b = N$.

Iš viso galime sudaryti $N!$ skirtingų paieškos medžių. Tai milžiniškas skaičius, net kai turime nedaug elementų, pvz., $N = 20$, nes remiantis Stirlingo formule

$$20! \geqslant \sqrt{40\pi} \left(\frac{20}{e} \right)^{20} > 2 \cdot 10^{18} \, .$$

Iš viso galime sudaryti $N!$ skirtingų paieškos medžių. Tai milžiniškas skaičius, net kai turime nedaug elementų, pvz., $N = 20$, nes remiantis Stirlingo formule

$$20! \geqslant \sqrt{40\pi} \left(\frac{20}{e} \right)^{20} > 2 \cdot 10^{18} \, .$$

Pažymėkime a_N paieškos algoritmo vidutinį sudėtingumą, kai nagrinėjame visus galimus paieškos medžius ir kai vienodai dažnai tikrinami visi N medžio elementai.

Iš viso galime sudaryti $N!$ skirtingų paieškos medžių. Tai milžiniškas skaičius, net kai turime nedaug elementų, pvz., $N = 20$, nes remiantis Stirlingo formule

$$20! \geqslant \sqrt{40\pi} \left(\frac{20}{e} \right)^{20} > 2 \cdot 10^{18} \, .$$

Pažymėkime a_N paieškos algoritmo vidutinį sudėtingumą, kai nagrinėjame visus galimus paieškos medžius ir kai vienodai dažnai tikrinami visi N medžio elementai.

Šiame pavyzdyje puikiai matysime teorinės analizės galimybes. Atlikę pakankamai paprastus veiksmus, galėsime įvertinti šio milžiniško variantų skaičiaus vidutinį sudėtingumą.

Nesunku įrodyti, kad teisinga tokia rekurentinė lygtis

$$a_N = 1 + \frac{2}{N^2} \sum_{j=1}^{N-1} j\, a_j, \quad a_1 = 1.$$

Ją gauname nagrinėdami visus galimus medžio šaknyje esančių elementų atvejus. Su vienoda tikimybe šaknyje bus visi elementai. Tarkime, kad toks elementas yra i. Tada žinome, kad kairiajame pomedyje yra $(i-1)$ viršūnė, o dešiniajame – $(N-i)$ viršūnių. Atlikę nesudėtingus skaičiavimus ir gauname užrašytą formulę.

Nesunku įrodyti, kad teisinga tokia rekurentinė lygtis

$$a_N = 1 + \frac{2}{N^2} \sum_{j=1}^{N-1} j\, a_j, \quad a_1 = 1.$$

Ją gauname nagrinėdami visus galimus medžio šaknyje esančių elementų atvejus. Su vienoda tikimybe šaknyje bus visi elementai. Tarkime, kad toks elementas yra i. Tada žinome, kad kairiajame pomedyje yra $(i-1)$ viršūnė, o dešiniajame – $(N-i)$ viršūnių. Atlikę nesudėtingus skaičiavimus ir gauname užrašytą formulę.

Žinant šią lygtį, galime skaičiuoti paieškos algortmo vidutinį sudėtingumą net ir labai dideliam elementų skaičiui N.

Jeigu pratęsime teorinę analizę, tai galime sprendinį užrašyti ir analizine forma.

Jeigu pratęsime teorinę analizę, tai galime sprendinį užrašyti ir analizine forma.

Apibrėžkime harmoninės eilutės dalinės sumos funkciją

$$H_N = 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{N}\,.$$

Tada nesunku patikrinti, kad lygties sprendinys yra užrašomas taip:

$$a_N = 2\frac{N+1}{N} H_N - 3\,.$$

Atsižvelgę ¿ tai, kad $H_N = \ln N + \gamma + \mathcal{O}\left(\frac{1}{N^2}\right)$, gauname, kad paieškos algoritmo vidutinis sudėtingumas yra

$$a_N = 2 \ln N + \mathcal{O}(1) .$$

Atsižvelgę į tai, kad $H_N = \ln N + \gamma + \mathcal{O}\left(\frac{1}{N^2}\right)$, gauname, kad paieškos algoritmo vidutinis sudėtingumas yra

$$a_N = 2 \ln N + \mathcal{O}(1) .$$

Priminsime, kad idealiai subalansuoto paieškos medžio atveju $W_v = \log N$. Kadangi

$$\frac{2 \ln N}{\log N} = 2 \ln 2 = 1.386,$$

tai vidutinis paieškos algoritmo sudėtingumas dvejetainiame paieškos medyje yra tik 1.386 karto didesnis nei idealiai subalansuotame medyje.