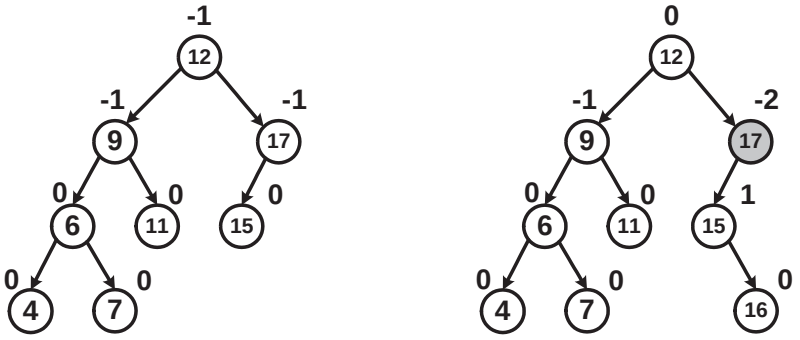
2.4. Sudėtingesni dvejetainiai medžiai

Šiame skirsnyje nagrinėsime sudėtingesnius dvejetainius medžius. Pagrindinis dvejetainio paieškos medžio trūkumas yra tas, kad blogiausiu atveju jis gali virsti tiesiniu sąrašu. Tada duomenų įterpimo, šalinimo ir paieškos veiksmai tampa labai neefektyviais, jų sudėtingumas yra proporcingas medžio viršūnių skaičiui. Parodysime, kaip ši problema sprendžiama naudojant AVL medžius ir piramidės duomenų struktūrą.

2.4.1. AVL paieškos medis

Tai iš dalies subalansuotas dvejetainis paieškos medis, kurio bet kurios viršūnės kairiojo ir dešiniojo pomedžių aukščiai gali skirtis ne daugiau nei vienetu. Jis vadinamas AVL medžiu pagal jį sukūrusių matematikų G. Adelson–Velskio ir E. Landi pavardes. Tokiame medyje elemento paieškos, naujos viršūnės įterpimo ar pašalinimo sudėtingumas yra tik $\mathcal{O}(\log N)$ veiksmų. AVL medžių pavyzdžiai yra pateikti 2.21 paveiksle, prie kiekvienos viršūnės nurodytas jos subalansuotumo faktorius.

Šis faktorius gali būti lygus $-1, 0$ arba 1. Reikšmė -1 yra suteikiama tada, kai kairiojo pomedžio aukštis didesnis vienetu už dešiniojo pomedžio aukštį, ji lygi 0, kai pomedžiai yra subalansuoti, ir lygi 1, kai dešiniojo pomedžio aukštis yra vienetu didesnis už kairiojo pomedžio aukštį. Jeigu po naujos viršūnės įterpimo ar pašalinimo šis faktorius tampa lygus -2 arba 2, tada papildomai balansuojame dvejetainį medį.



2.21 pav. Dvejetainiai paieškos medžiai: a) AVL medis, b) tai nėra AVL medis, nes 17 viršūnės kairiojo pomedžio aukštis yra lygus 2, o dešiniojo pomedžio aukštis yra 0, taigi jų skirtumas yra -2.

Realizuodami AVL medį modifikuojame dvejetainio medžio *elementą*, jame papildomai saugome informaciją apie viršūnės subalansuotumo faktorių:

```
struct nodeAVL{
  T data;
  node* left;
  node* right;
  int balance;
}
```

Paieškos algoritmo sudėtingumas

Įvertinsime paieškos algoritmo AVL medyje sudėtingumą. Tarkime, kad saugome N elementų. Aišku, kad AVL medžio aukštis yra nedidesnis už nesubalansuoto dvejetainio paieškos medžio aukštį, todėl vidutinis paieškos sudėtingumas yra $\mathcal{O}(\log N)$.

Dabar nagrinėkime blogiausiojo atvejo AVL medžio aukštį, kurį žymėsime $h_b(N)$. Kadangi AVL medžio aukštis yra nemažesnis už pilnai subalansuoto medžio aukštį, tai teisingas toks įvertis

$$h_b(N) \geq \log N\,.$$

Labiausiai išbalansuoto AVL medžio kiekvienos viršūnės kairiojo (arba dešiniojo) pomedžio aukštis yra vienetu didesnis už dešiniojo (atitinkamai, kairiojo) pomedžio aukštį. Tokio medžio T_h schema ir AVL medžio pavyzdys yra pateikti 2.22 paveiksle. Pažymėkime N_h labiausiai išbalansuoto AVL medžio viršūnių skaičių. Iš pateiktosios medžio schemos gauname lygtį

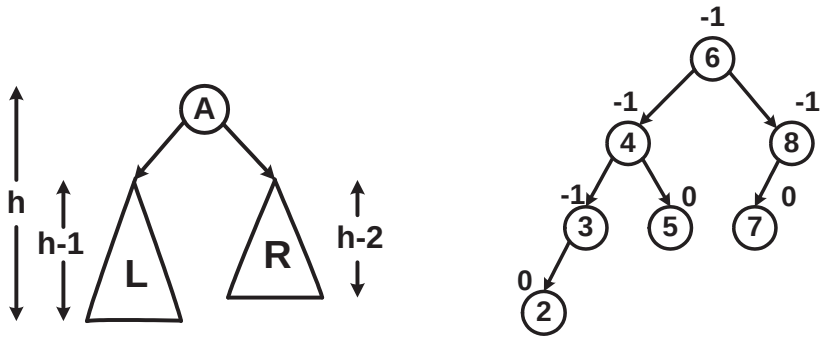
$$N_h = N_{h-1} + N_{h-2} + 1\,.$$

Tokios lygties sprendinys bus vienintelis, kai papildomai suformuluosime dvi pradines sąlygas. Jas gauname iš AVL medžio apibrėžimo:

$$N_0 = 0, \quad N_1 = 1\,.$$

Gavome tiesinę nehomogeninę antrosios eilės skirtumų lygtį. Ji sprendžiama panašiai kaip ir tiesinės diferencialinės lygtys su pastoviais koeficientais. Pažymėkime naują funkciją $M_h = N_h + 1$, ji yra tokio uždavinio sprendinys:

$$\begin{cases} M_h = M_{h-1} + M_{h-2}, & h > 1, \\ M_0 = 1, \quad M_1 = 2\,. \end{cases}$$



a) b)

2.22 pav. Maksimaliai išbalansuotas AVL paieškos medis: a) bendroji medžio T_h schema, b) AVL medžio pavyzdys, kai kairiojo pomedžio aukštis visada didesnis už dešiniojo pomedžio aukštį.

Ieškosime atskirojo lygties sprendinio $M_h = q^h$, kurį įrašę į skirtumų lygtį gauname charakteringąją lygtį

$$q^2 - q - 1 = 0\,,$$

kuri turi du sprendinius

$$q_1 = \frac{1+\sqrt{5}}{2}, \quad q_2 = \frac{1-\sqrt{5}}{2}.$$

Tada bendrasis skirtumų lygties sprendinys yra

$$M_h = c_1 q_1^h + c_2 q_2^h\,.$$

Panaudoję pradines sąlygas randame tikslų AVL medžio viršūnių skaičių

$$N_h = \frac{3\sqrt{5}+5}{2} q_1^h - \frac{3\sqrt{5}+3}{2} q_2^h - 1.$$

Kadangi $q_1 > 1$, o $|q_2| < 0.619$, tai dideliems h yra teisinga tokia lygybė

$$N_h \approx \frac{3\sqrt{5}+5}{2} q_1^h - 1,$$

todėl AVL medžio aukščio priklausomybę nuo viršūnių skaičiaus įvertiname nelygybe

$$h_b(N) \leq \frac{\log(N+1)}{\log q_1} = 1.4404 \log N\,.$$

Taigi net ir labiausiai nesubalansuoto AVL paieškos medžio aukštis tik 1.44 karto didesnis už pilnai subalansuoto medžio aukštį. Todėl duomenų paieška AVL medyje yra labai efektyvi.

Ankstesniame skirsnyje įrodėme, kad vidutinis paieškos algoritmo sudėtingumas dvejetainiame paieškos medyje yra $1.386 \log N$. Paieškos algoritmo AVL medyje vidutinio sudėtingumo teorinis įvertis nėra žinomas, tačiau išsamūs skaičiavimo eksperimentai rodo, kad jis yra $\log N + 0.25$, taigi labai artimas paieškos sudėtingumui pilnai subalansuotame medyje.

2.4.2. Piramidė

Šiame poskyryje susipažinsime su dar vienu dvejetainių medžių variantu – *piramide* (angl. *heap* data structure). Jai būdingos šios dvi pagrindinės savybės:

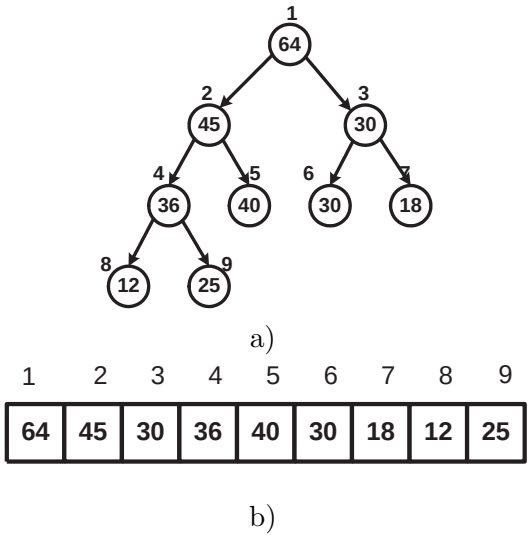
- Piramidė yra pilnas, subalansuotas medis: pirmiausia užpildoma medžio šaknis, po to pirmojo lygmens viršūnės, antrojo lygmens viršūnės ir t.t. Kiekvieną lygmenį užpildome iš kairės į dešinę.
- Medis yra sutvarkytas taip, kad kiekvienos viršūnės vaikai yra nedidesni už pačią viršūnę.

Iš antrosios savybės nesunkiai seka išvada, kad piramidės šakninėje viršūnėje yra saugomas *didžiausias* aibės elementas.

Piramidės pavyzdys pavaizduotas 2.23*a* brėžinyje.

Piramidės atvaizdavimas. Kadangi piramidė yra pilnas dvejetainis medis, tai jos viršūnes labai patogu saugoti masyve. Tada i-tosios viršūnės a_i vaikai yra a_{2i}, a_{2i+1} masyvo elementai. Lengvai randame kiekvienos piramidės viršūnės a_i tėvinę viršūnę: ji saugoma $j = \lfloor i/2 \rfloor$–ajame masyvo elemente a_j. 2.23*b* brėžinyje pavaizduota piramidė–masyvas.

Piramidės formavimas. Tarkime turime seką duomenų $e_1, e_2, \ldots, e_N$ iš kurių reikia sudaryti piramidės struktūrą. Šiuos elementus paeiliui talpiname į A masyvą. Tada visos dvejetainio medžio viršūnės – lapai jau yra sutvarkyti. Piramidės lapų indeksai kinta nuo $(\frac{N}{2} + 1)$ iki N. Tada paeiliui imame viršūnes $\frac{N}{2}, \ldots, 1$ ir rekursyviai tikriname piramidės sutvarkymo sąlygą. Jei ji yra pažeista, tai sukeičiame vietomis tėvinę viršūnę su didžiausiu jos vaiku.



2.23 pav. Piramidės pavyzdys: a) dvejetainis medis, b) masyvas.

Piramidės formavimo algoritmas

MakeHeap ()
begin
(1) **for** (i=1; i $\leqslant$ N; i++) $a_i = e_i$;

(2) j = $\frac{N}{2}$;
(3) **for** (i=j; i > 0; i = i-1)
(4) HeapDownOrder (i, N);
end

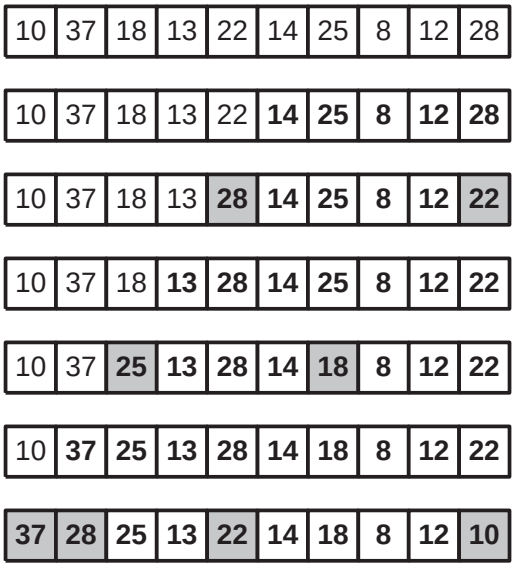
Pateiksime piramidės elementų sutvarkymo algoritmo pavyzdį.

```
HeapDownOrder ( p, N )
begin
 (1)  i=p;   j = i+i;
 (2)  while  ( j ⩽ N ) {
              (3)  k = j;
              (4)  if  ( (j+1) ⩽ N )
                      (5)  if  ( a_{j+1} > a_j ) k = j+1;
              (6)  if  ( a_i < a_k ) {
                      (7)  swap (a_i, a_k);
                      (8)  i = k;   j= i+i;
                   } else  j = N+1;
     }
end
```

Kadangi piramidė yra pilnas dvejetainis medis, tai jos aukštis yra nedidesnis už $\log N$, todėl kiekvienos elementų tvarkymo operacijos metu įvykdome nedaugiau kaip $2\log N$ elementų lyginimo veiksmų ir $\log N$ elementų sukeitimų. Taigi piramidės formavimo algoritmo sudėtingumas yra

$$L(N) = N\log N, \quad S(N) = \frac{1}{2}N\log N\,.$$

2.8 pavyzdys. Skaičių masyvo pertvarkymas į piramidę. Iš skaičių masyvo $A = (10, 37, 18, 13, 22, 14, 25, 10, 12, 28)$ suformuosime piramidę. Formavimo eiga pavaizduota 2.24 brėžinyje.



2.24 pav. Skaičių masyvo pertvarkymas į piramidės struktūrą. Riebiu šriftu pavaizduoti jau sutvarkyti elementai, pilka spalva pažymėti tie elementai, kurie buvo sukeisti vietomis, vykdant eilinį piramidės formavimo žingsnį.