Dalyko „Algoritmai ir duomenų struktūros“ 2020 m. egzamino užduočių sprendimai

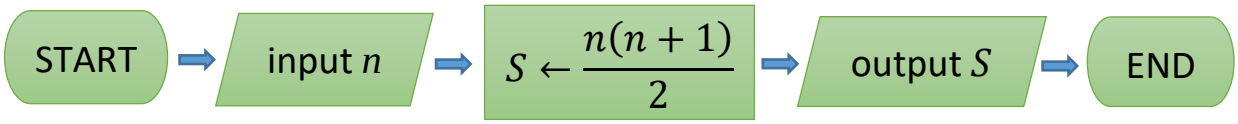
1. Tiesiniu, konstantiniu ir kvadratiniu sudėtingumu užduotyje nurodytą sumą galima apskaičiuoti atitinkamai šiais būdais:

$$1+2+3+\cdots+n=\frac{n(n+1)}{2}=1+\underbrace{1+1}_{2}+\underbrace{1+1+1}_{3}+\cdots+\underbrace{1+\cdots+1}_{n}$$

a) Rekursyvus tiesinio sudėtingumo algoritmas pavaizduotas pseudokodu:

```
1. suma(n):
2.     if n = 1 then
3.         return(n)
4.     else
5.         return(n + suma(n − 1))
```

b) Konstantinio sudėtingumo algoritmas pavaizduotas blokine schema:



c) Kvadratinio sudėtingumo algoritmas pavaizduotas pseudokodu:

```
1. suma2(n):
2.     S ← 0
3.     for i ← 1 to n
4.         for j ← 1 to i
5.             S ← S + 1
6.     return(S)
```

2. Skaičiaus 1000_n reikšmė lygi

$$1000_n = 1\cdot n^3+0\cdot n^2+0\cdot n^1+0\cdot n^0=n^3.$$

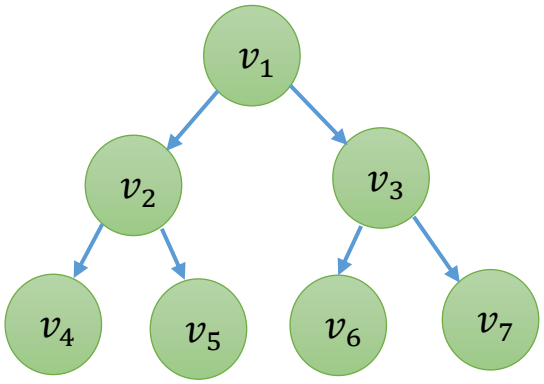
a) $\sqrt{1000_n}=\sqrt{n^3}=\sqrt{n^2\cdot n}=n\sqrt{n}$. Vadinasi, $n\sqrt{n}\in\mathbb{N}$, kai $\sqrt{n}\in\mathbb{N}$, todėl $n=4,9,16,25,\ldots$ (netinka $n=1$, nes nėra tokios skaičiavimo sistemos). **Atsakymas**: $n=(k+1)^2$, kai $k\in\mathbb{N}$.

b) Kai $n=4$: $1000_4=64_{10}$; $\sqrt{64}=8$; $8_{10}=\mathbf{20_4}$,
kai $n=9$: $1000_9=729_{10}$; $\sqrt{729}=27$; $27_{10}=\mathbf{30_9}$,
kai $n=16$: $1000_{16}=4096_{10}$; $\sqrt{4096}=64$; $64_{10}=\mathbf{40_{16}}$...

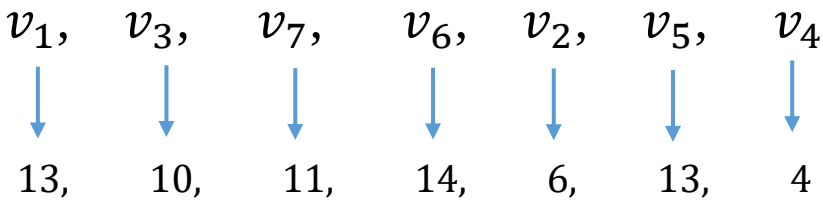
Nesunku įžvelgti, kad bendruoju atveju teisingas **atsakymas** turėtų būti $\overline{(\boldsymbol{k+1})\boldsymbol{0}}_{(\boldsymbol{k+1})^2}$, kai $k \in \mathbb{N}$. Tai nesunkiai parodoma: $(n\sqrt{n})_n = \left((k+1)^2\sqrt{(k+1)^2}\right)_{(k+1)^2} = (k+1)^2{}_{(k+1)^2} \cdot (k+1)_{(k+1)^2} = 10_{(k+1)^2} \cdot (k+1)_{(k+1)^2} = \overline{(k+1)0}_{(k+1)^2}$, kai $k \in \mathbb{N}$.

3. Realizavus dvi Eratosteno rėčio algoritmo iteracijas, išbraukiami 2 ir 3 kartotiniai (išskyrus 2 ir 3). Jei parodysime, kad visi likę neišbraukti skaičiai (išskyrus 2 ir 3) išreiškiami pavidalu $6k \pm 1$, kai $k \in \mathbb{N}$, tuo pačiu parodysime, kad ir bet kuris pirminis skaičius (išskyrus 2 ir 3) yra išreiškiamas pavidalu $6k \pm 1$, kai $k \in \mathbb{N}$. Pažymėkime natūraliųjų skaičių aibę (be vieneto) šia išraiška
$$\cancel{1}, 2, 3, \underbrace{6k-2, 6k-1, 6k, 6k+1, 6k+2, 6k+3}_{k\in\mathbb{N}}.$$
Išbraukus 2 ir 3 kartotinius (išskyrus 2 ir 3) gaunama tokia aibė
$$\cancel{1}, 2, 3, \underbrace{\cancel{6k-2}, 6k-1, \cancel{6k}, 6k+1, \cancel{6k+2}, \cancel{6k+3}}_{k\in\mathbb{N}}.$$
Iš čia matome, kad bet kuris pirminis skaičius (išskyrus 2 ir 3) yra išreiškiamas pavidalu $6k+1$ arba $6k-1$, kai $k \in \mathbb{N}$.

4. Tarkime, kad V-D-K strategija apeiname šį Heap medį (viršūnių reikšmes pažymėkime v_1, v_2, v_3, ...)



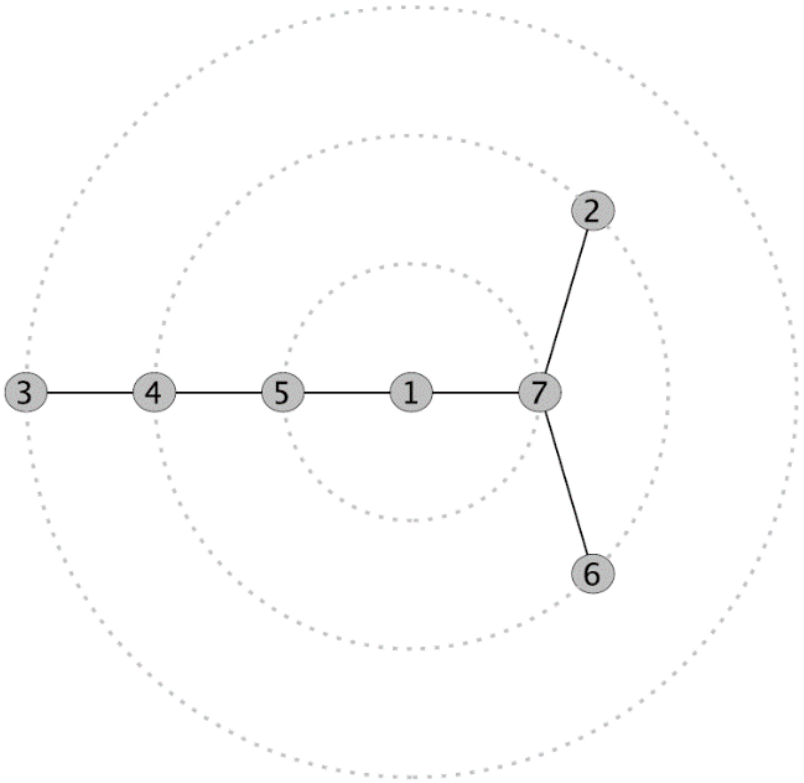
V-D-K strategija apėjus viršūnes, jų reikšmės atspausdinamos šia seka, kuri duota sąlygoje:



Iš čia matome, kad sąrašas, kuriuo apibrėžtas Heap medis, turėtų būti $[v_1, v_2, v_3, v_4, v_5, v_6, v_7]$ = [13, 6, 10, 4, 13, 14, 11], tačiau šis sąrašas netenkina nei min-Heap, nei max-Heap sąlygų, todėl jis neegzsistuoja.

5. Pažymėkime duoto sąrašo [7, 4, 5, 1, 7, 7, 5] pasikartojančių elementų reikšmes raidėmis [7A, 4, 5A, 1, 7B, 7C, 5B].
   a) Rikiavimo išrinkimo algoritmu (Selection Sort) eiga:
      [7A, 4, 5A, 1, 7B, 7C, 5B]
      [1, 4, 5A, 7A, 7B, 7C, 5B]
      [1, 4, 5A, 7A, 7B, 7C, 5B]
      [1, 4, 5A, 7A, 7B, 7C, 5B]
      [1, 4, 5A, 5B, 7B, 7C, 7A]
      [1, 4, 5A, 5B, 7B, 7C, 7A]
      [1, 4, 5A, 5B, 7B, 7C, 7A]
      Šiuo atveju parodomas rikiavimo nestabilumas, nes sąrašo elementai su reikšmėmis 7 susikeitė vietomis.
   b) Rikiavimo Šelo metodu (Shell Sort) eiga:
      [7A, 4, 5A, 1, 7B, 7C, 5B]
      [1, 4, 5A, 5B, 7B, 7C, 7A]
      [1, 4, 5A, 5B, 7B, 7C, 7A]
      Šiuo atveju parodomas rikiavimo nestabilumas, nes sąrašo elementai su reikšmėmis 7 susikeitė vietomis.
   c) Piramidės rikiavimo (Heap Sort) eiga:
      [7A, 4, 5A, 1, 7B, 7C, 5B]
      [7A, 4, 7C, 1, 7B, 5A, 5B]
      [7A, 7B, 7C, 1, 4, 5A, 5B] (max Heap!)
      [5B, 7B, 7C, 1, 4, 5A, 7A]
      [7B, 5B, 7C, 1, 4, 5A, 7A] (max Heap!)
      [5A, 5B, 7C, 1, 4, 7B, 7A]
      [7C, 5B, 5A, 1, 4, 7B, 7A] (max Heap!)
      [4, 5B, 5A, 1, 7C, 7B, 7A]
      [5B, 4, 5A, 1, 7C, 7B, 7A] (max Heap!)
      [1, 4, 5A, 5B, 7C, 7B, 7A]
      [5A, 4, 1, 5B, 7C, 7B, 7A] (max Heap!)
      [1, 4, 5A, 5B, 7C, 7B, 7A]
      [4, 1, 5A, 5B, 7C, 7B, 7A] (max Heap!)
      [1, 4, 5A, 5B, 7C, 7B, 7A] (max Heap!)
      [1, 4, 5A, 5B, 7C, 7B, 7A]
      Šiuo atveju parodomas rikiavimo nestabilumas, nes sąrašo elementai su reikšmėmis 7 susikeitė vietomis.

6. Pagal duotą Priuferio kodą [7, 4, 5, 1, 7] pavaizduokime medį radialiniu būdu (brėžinyje aiškiai matomas medžio centras [1, 5]):



Šio medžio gretimumo matrica

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

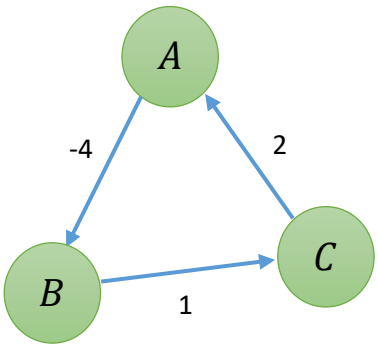
Šio medžio incidentumo matrica (matricos stulpelių eilės tvarka nesvarbi, jei neįvardintos briaunos)

$$\begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 \end{bmatrix}$$

7. Dijkstros ir Belmano–Fordo trumpiausių takų paieškos algoritmai

Algoritmas	**Dijkstros**	**Belmano–Fordo**
Briaunų relaksacija (1)	Vykdoma pasiektų viršūnių aibei pagal mažiausius briaunų svorius.	Vykdoma pasiektų viršūnių aibei visoms briaunoms.
Briaunų relaksacija (2)	Vykdoma po kartą kiekvienai briaunai.	Vykdoma po kelis kartus kiekvienai briaunai.
Asociacija su kitais grafų paieškos algoritmais	Primena paiešką į gylį.	Primena paiešką į plotį.
Neigiamų ciklų (di)grafe aptikimas	Neaptinka, programos vykdymo eigoje susidaro amžinas ciklas.	Aptinka, grąžinamas rezultatas FALSE.
Algoritmo sudėtingumas	O((|V| + |E|)log |V|)	O(|V|·|E|)
Grafas be neigiamų ciklų	Apdorojamas greičiau.	Apdorojamas lėčiau.

Žemiau pateiktas digrafas, turintis neigiamą ciklą.



Šiuo pavyzdžiu parodoma, kad, ieškant trumpiausių takų, Belmano–Fordo algoritmas veiks efektyviau, nes grąžins rezultatą FALSE, o Dijkstros algoritmas skaičiuos amžinai.