Duomenų struktūros ir algoritmai

2 paskaita

2019-02-13

Algoritmo sąvoka

- Algoritmas – tai tam tikra veiksmų seka, kurią reikia atlikti norint gauti rezultatą.

Algoritmo užrašymo būdai

- Blokinė schema (Flowchart)
- Pseudokodas (Pseudocode)
- Paprasta kalba
- Algoritmo žingsnių vizualizacija

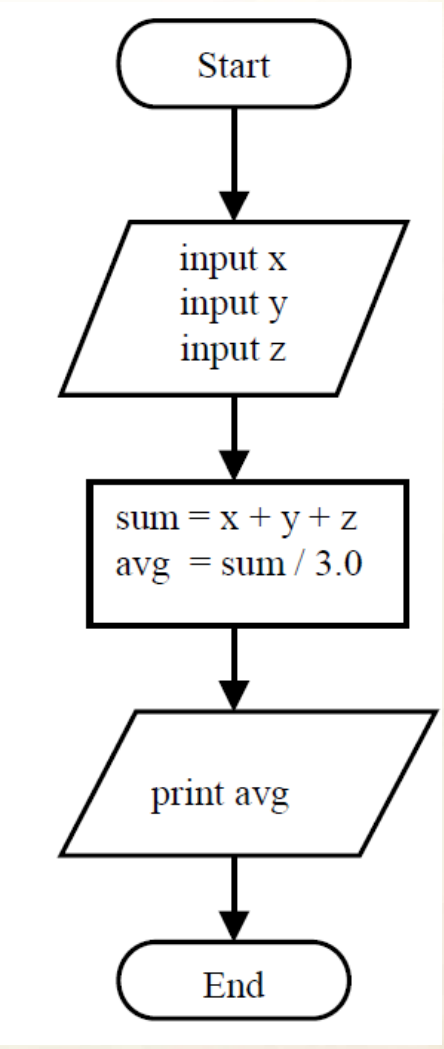
Pseudokodas ir blokinė schema (1)

Average(x, y, z)

sum ← x + y + z

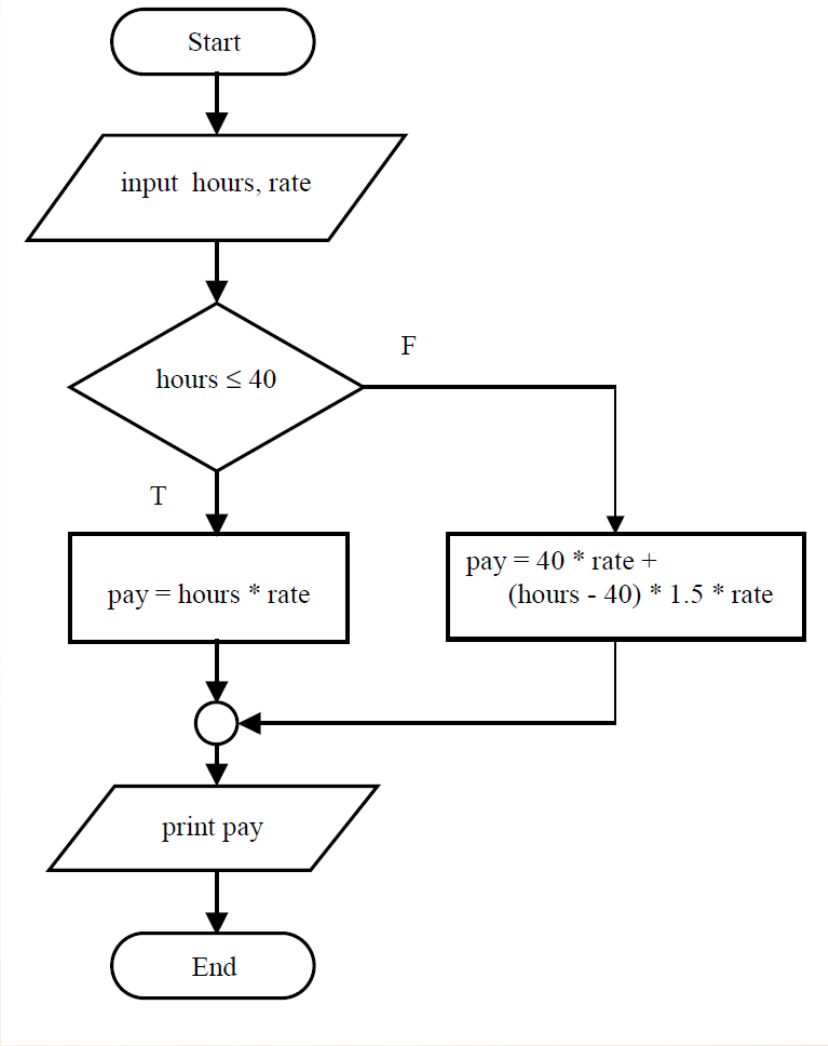
avg ← sum / 3

return(print(avg))



Pseudokodas ir blokinė schema (2)

OvertimePay(hours, rate)
    if hours ≤ 40 **then**
        pay ← hours * rate
    else
        pay ← 40 * rate +
        (hours – 40) * rate *1,5
    return(print(pay))



Klasikiniai algoritmai

Seniausias žinomas algoritmas –
Euklido didžiausio bendro daliklio algoritmas.

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

(GCD = Greatest Common Divisor)

Pastaba: operatorius % žymi dalybos liekaną:
a % b = a **mod** b (pavyzdžiui, 20 % 7 = 6).

Euklido didžiausio bendro daliklio algoritmo taikymai

GCD(a, b, c) = **GCD**(**GCD**(a, b) , c),

GCD(a, b, c, d) = **GCD**(**GCD**(**GCD**(a, b) , c) , d) ir t. t.

1 užduotis: realizuokite algoritmą:

$$\mathbf{GCD}(a_1, a_2, ..., a_n).$$

Pažymėkime **LCM**(a, b) – mažiausią bendrą kartotinį skaičių a ir b, tada teisinga lygybė:

$$\mathbf{LCM}(a, b) = a * b * \mathbf{GCD}(a, b).$$

2 užduotis: realizuokite algoritmą:

$$\mathbf{LCM}(a_1, a_2, ..., a_n).$$

(LCM = Least Common Multiple)

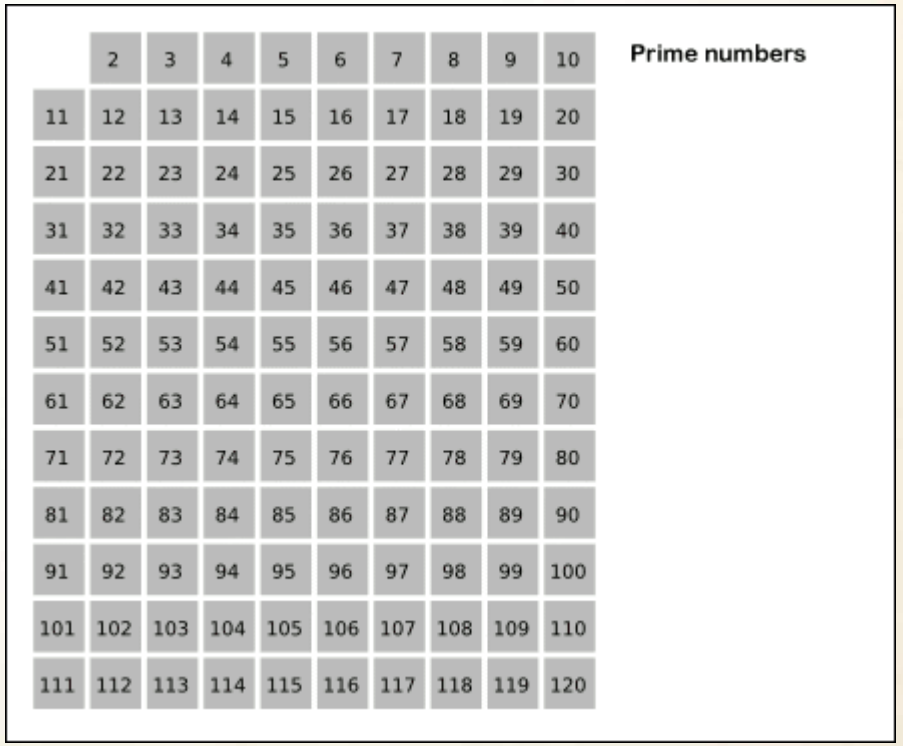
Pirminių skaičių paieškos algoritmai

Algoritmo principas:
Jei sąraše (2, 3, …, m) skaičius n neužbrauktas, užbraukiami skaičiaus n kartotiniai paliekant n neužbrauktą.
Ciklas kartojamas visiems n = 2, 3, …, $[\sqrt{m}]$.

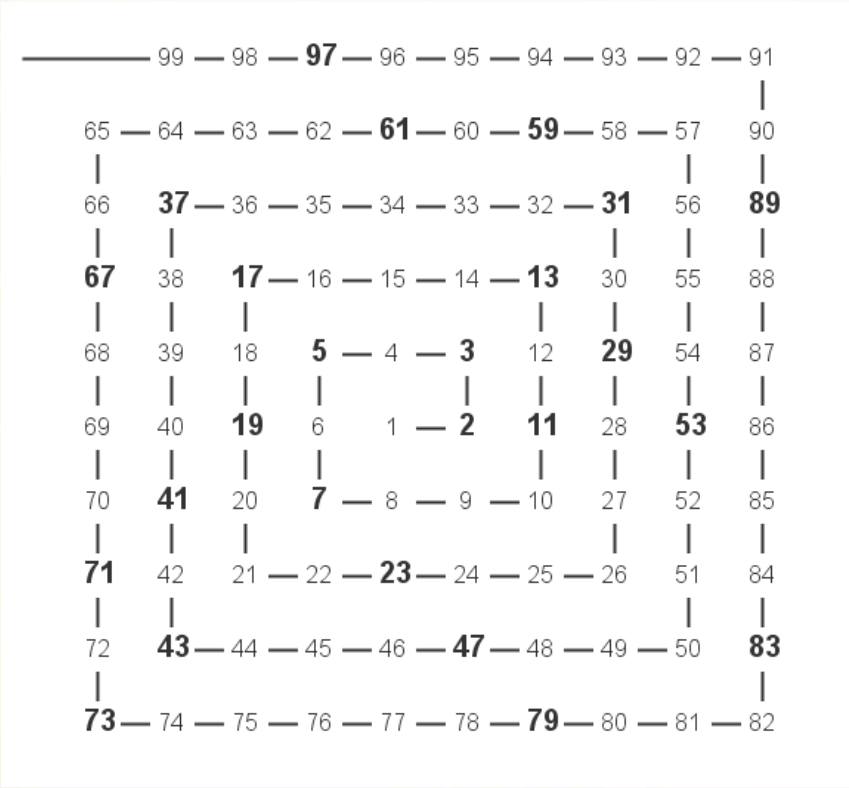
Nė karto neužbrauktieji skaičiai – **pirminiai**.

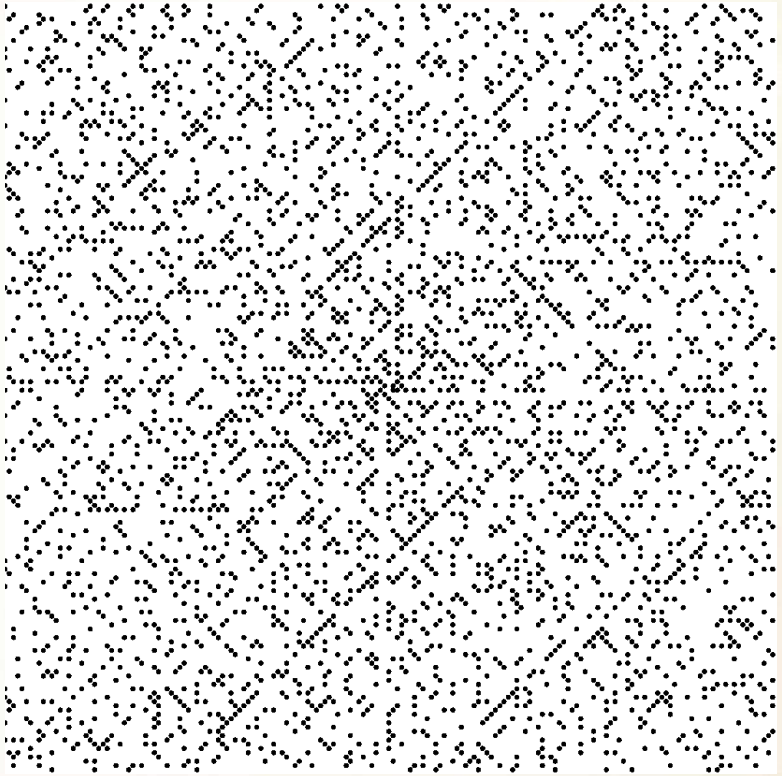
[*] žymi sveikąją skaičiaus dalį, pvz: [8,2]=8

Eratosteno rėčio algoritmo žingsnių vizualizacija



Ulamo spiralė





According to Gardner, Ulam discovered the spiral in 1963 while doodling during the presentation of "a long and very boring paper" at a scientific meeting.

Skaičiavimo sistemos keitimo algoritmas

Skaičius $a_k a_{k-1} \dots a_1$ n-tainėje skaičiavimo sistemoje užrašomas pavidalu

$$\sum_{i=1}^{k} n^{i-1} a_i$$

Pavyzdžiui, $123_{10} = 3*10^0+2*10^1+1*10^2$.

Analogiškai, $123_{10} = 1*2^0+1*2^1+0*2^2+1*2^3+1*2^4+1*2^5+1*2^6 = 1111011_2$,

$123_{10} = 4*7^0+3*7^1+2*7^2 = 234_7$, $123_{10} = 11*16^0+7*16^1 = 7(11)_{16} = 7B_{16}$.

Pseudokodas, skirtas skaičiaus $a_k a_{k-1} \dots a_1$ konvertavimui iš n-tainės į m-tainę skaičiavimo sistemą.

Atkreipkime dėmesį, kad kai $10 < n$, tai n-tainėje skaičiavimo sistemoje įvedami nauji žymėjimai A=10, B=11, C=12 ir t. t.

ConvertNumber$(a_k a_{k-1} \dots a_1,\ n,\ m)$

$S \leftarrow \sum_{i=1}^{k} n^{i-1} a_i$

$i \leftarrow 1$

while $S > 0$

$b_i \leftarrow S \,\%\, m$

$S \leftarrow (S - b_i)/m$

$i++$

return$(b_q b_{q-1} \dots b_1)$

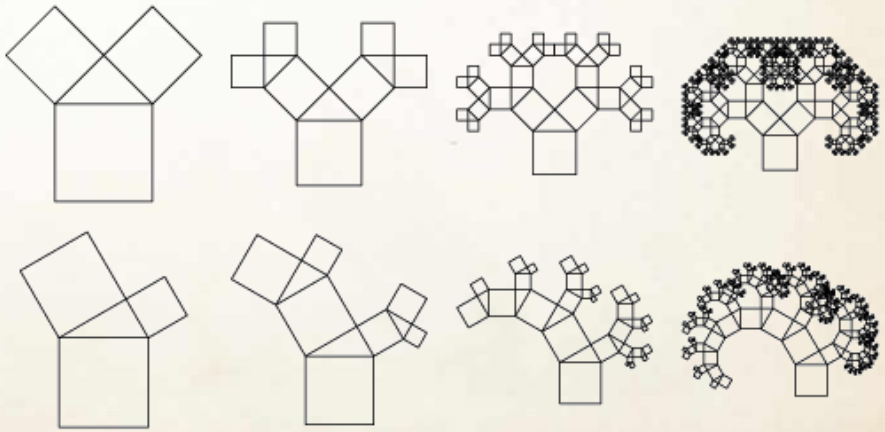
Rekursija

Rekursija – tai savybė nusakyti objektą remiantis juo pačiu. Programavime rekursija vadinamas paprogramės (procedūros arba funkcijos) kreipimasis į save pačią.

Rekursijos taikymas skaičiuojant n!:

```
factorial(n)
    if n < 1 then
        return(1)
    else
        return(n * factorial(n-1))
```

Rekursijos taikymas sudarant Pitagoro medį:



Indukcija

Indukcija[1] – loginis samprotavimas, kuriame nuo atskirų faktų, žinių einama prie bendresnių (buvo per pirmą paskaitą).
Indukcija[2] – sistemingai pasikartojanti veiksmų seka, kurioje rekurentiniai sąryšiai nekinta.

Pavyzdys.

Fibonači sekos: 1, 1, 2, 3, 5, 8, 13, 21, 34, ...

n-tasis narys (išskyrus pirmus 2 narius) rekurenčiai apskaičiuojamas pagal formulę:

$$F[n] = F[n-1] + F[n-2].$$

Algoritmų taikymo pavyzdžiai

- Nuoseklioji paieška
- Dvejetainė paieška
- „Skaldyk ir valdyk“ principas

Nuoseklioji paieška

- Programa patikrina, ar skaičius v yra duotoje aibėje a=[a[q],a[q+1],..., a[r]]. Jei taip, grąžinamas ieškomo elemento indeksas masyve, jei ne, grąžinama reikšmė -1.

```
IndexSearch(a, v, q, r)
    for i ← q to r do
        if v = a[i] then
            return(i)
    return(-1)
```

Dvejetainė paieška

- Programa patikrina, ar skaičius v yra duotoje aibėje a=[a[q],a[q+1],..., a[r]], kurios elementai surūšiuoti. Jei taip, grąžinamas ieškomo elemento indeksas masyve, jei ne, grąžinama reikšmė -1.

```
BinarySearch(a, v, q, r)
    if r ≥ q then
        m ← [(q + r) / 2]
        if v = a[m] then
                return(m)
        else if v < a[m] then
                return(BinarySearch(a, v, q, m-1)
        else
                return(BinarySearch(a, v, m+1, r)
    else
        return(-1)
```

„Skaldyk ir valdyk“ principas

„Skaldyk ir valdyk“ principas skirtas sunkesnę užduotį skaidyti į lengvesnes ir greičiau išsprendžiamas užduotis.

Galutinis atsakymas gaunamas apibendrinus lengvesnių užduočių atsakymus.

Pavyzdžiai:

- *Great Internet Mersenne Prime Search* projektas *(Merseno pirminių skaičių* $M_n = 2^n - 1$ *paieška*).
- Kompiuterinių žaidimų kūrimas.
- Lygiagretieji skaičiavimai (pvz: C/C++ OpenMP)
- Politinių rinkimų balsų skaičiavimas ☺.

Algoritmo sudėtingumas

Algoritmo sudėtingumas – algoritmo darbo laiko priklausomybė nuo pradinių duomenų (input) dydžio.

Algoritmo sudėtingumas priklauso nuo:

- Sprendžiamos problemos uždavinių sudėtingumo.
- Efektyvaus algoritmo parinkimo.
- Programavimo kokybės.

Algoritmo sudėtingumas gali būti matuojamas ne tik **laiku**, bet ir

- Naudojamos atminties dydžiu (Space Complexity).
- Techninės įrangos parametrais.
- Gaunamos informacijos kiekiu (Information Complexity).
- Sunaudojamos elektros energijos kiekiu (pvz kriptovaliutos).

Trys pagrindinės algoritmo sudėtingumo klasės

Tiesinio sudėtingumo algoritmai

$O(N)$

Polinominio sudėtingumo algoritmai

$O(N^C)$

Eksponentinio sudėtingumo algoritmai

$O(q^N)$

Algoritmo sudėtingumo žymėjimai

Dažniausiai sutinkami algoritmo sudėtingumo įverčiai yra O(n), o(n), θ(n), kurie nežymiai skiriasi matematiniais apibrėžimais, tačiau visi jie apibrėžia funkcijos g(N) = O(f(N)) asimptotiką argumentui N artėjant į begalybę.

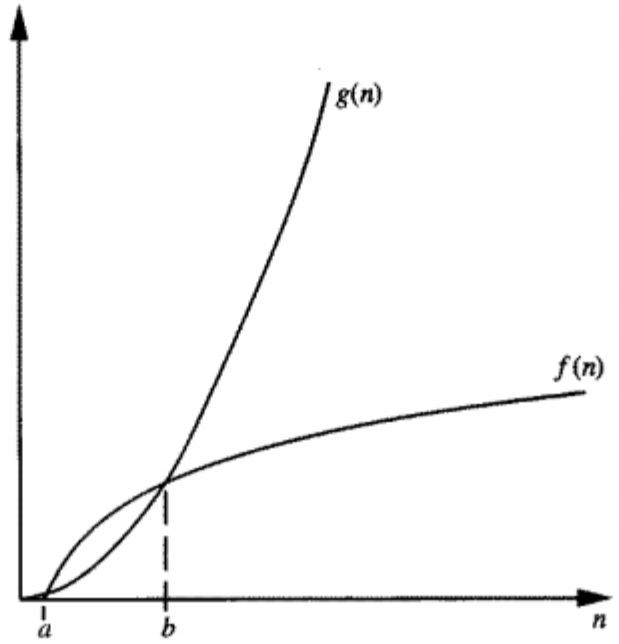
Funkcijos asimptotika – tai funkcijos f(x) reikšmės kitimo tendencija argumentui x artėjant į begalybę.

(Galima panaudoti *GeoGebra* programą funkcijos asimptotikai vizualizuoti.)

O-žymuo

O-žymuo: Funkcija *g(N)* yra vadinama *O(f(N))*, jei egzistuoja konstantos c_0 ir N_0 , tokios , kad $0 < g(N) < c_0 f(N)$ visiems $N > N_0$.

Pvz., *O(ln x) ir* $O(x^4)$:



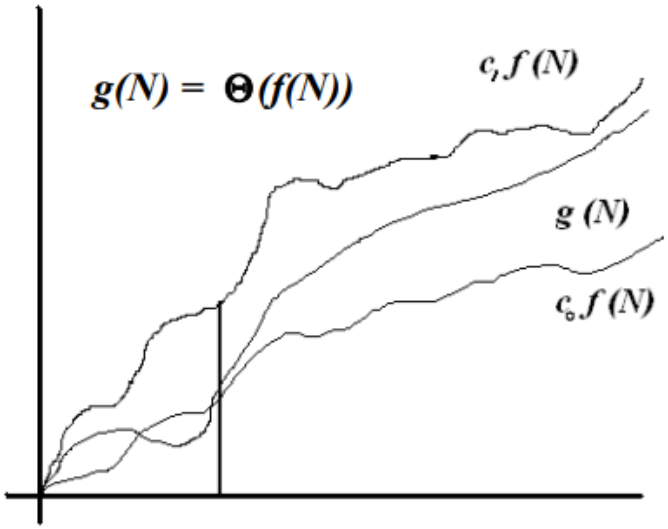
O ir θ žymenys

o-žymuo: Funkcija $g(N)$ yra vadinama $o(f(N))$, jei bet kuriai teigiamai konstantei c_0 egzistuoja N_0, tokia , kad $0 < g(N) < c_0 f(N)$ visiems $N > N_0$.

Intuityviai o-žymuo reiškia, kad

$$\lim_{N \to \infty} \frac{g(N)}{f(N)} = 0$$

Θ-žymuo: Funkcija $g(N)$ yra vadinama $\Theta(f(N))$, jei egzistuoja teigiamos konstantos c_0 ir c_1 ir egzistuoja N_0, tokios, kad $0 < c_0 f(N) < g(N) < c_1 f(N)$ visiems $N > N_0$.



Algoritmų sudėtingumo klasės

O(1)	Programos vykdymo laikas yra nekintamas (proprocingas konstantai)
O(logN)	Programa vykdoma truputį lėčiau, kai N didėja, šis laikas paprastai būna programose, kurios sprendžia didelį uždavinį, transformuodamos jį į eilę mažesnių uždavinių, išskaidydamos į fiksuotas dalis kiekviename etape, galima teigti, kad vykdymo laikas yra mažesnis nei didelė konstantos reikšmė.
O(N)	Programos vykdymo laikas yra tiesinis, jis didėja proporcingai įvedimo duomenų kiekiui.
O(NlogN)	Programos vykdymo laikas ilgėja proporcingai N log N, dažniausiai tai atsitinka, kai programa sprendžia uždavinį, išskaidydama jį į mažesnius uždavinius, kuriuos sprendžia atskirai, ir galiausiai sprendimus sujungia.
$O(N^2)$	Programos vykdymo laikas yra kvadratinis, programa praktiškai tinkama tik palyginti mažų uždavinių sprendimui, tai būdinga programoms, kurios apdoroja visas duomenų poras (galbūt dvigubo ciklo metu).
$O(N^3)$	Programos (galbūt turinčios trigubą ciklą) vykdymo laikas yra kubinis ir praktiškai skirtas mažiems uždaviniams spręsti.
$O(2^N)$, $O(N!)$	Praktiškai yra tik kelios programos su rodikliniu vykdymo laiku (eksponentinio, faktorialinio sudėtingumo), kurios tinkamos praktiniam naudojimui, pvz. tokios kaip brutalios jėgos (brute-force) uždavinių sprendimai.

https://en.wikipedia.org/wiki/Time_complexity

Duomenų struktūros

- Abstraktus duomenų tipas (ADS) – tai duomenų arba jų aibių, operacijų su duomenimis ir loginių ryšių tarp operacijų ir duomenų apibrėžimas (specifikacija).

- Duomenų struktūros:
  - Statinės duomenų struktūros (pavyzdžiui, masyvai).
  - Dinaminės duomenų struktūros (realizacija panaudojant rodyklės tipo kintamuosius: tiesinis sąrašas, stekas, eilė, dekas ir kt).

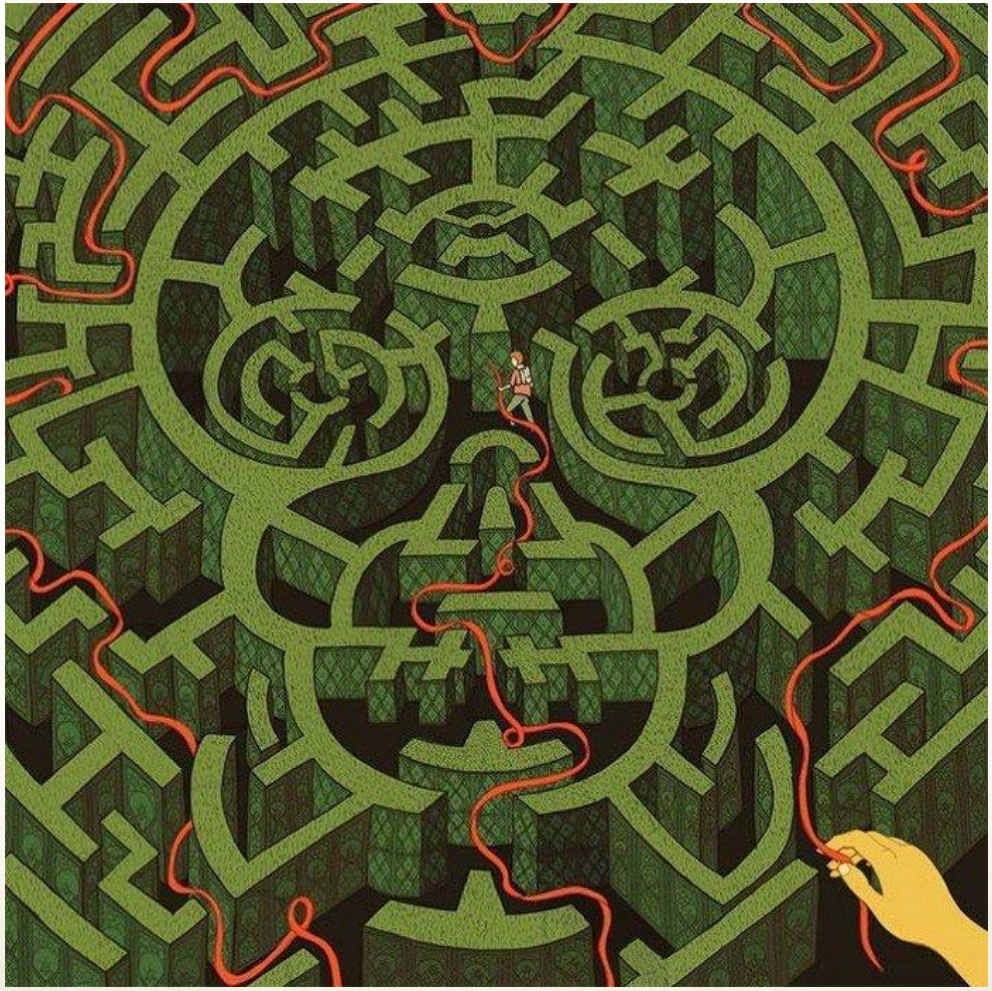
- Rodyklė (pointer) – tai atminties ląstelė, kurioje saugomas kitos ląstelės adresas. Naudojant rodykles, netiesiogiai pasiekiami duomenys, esantys ląstelėje, kurios adresą žino rodyklė.

Rodyklės C++

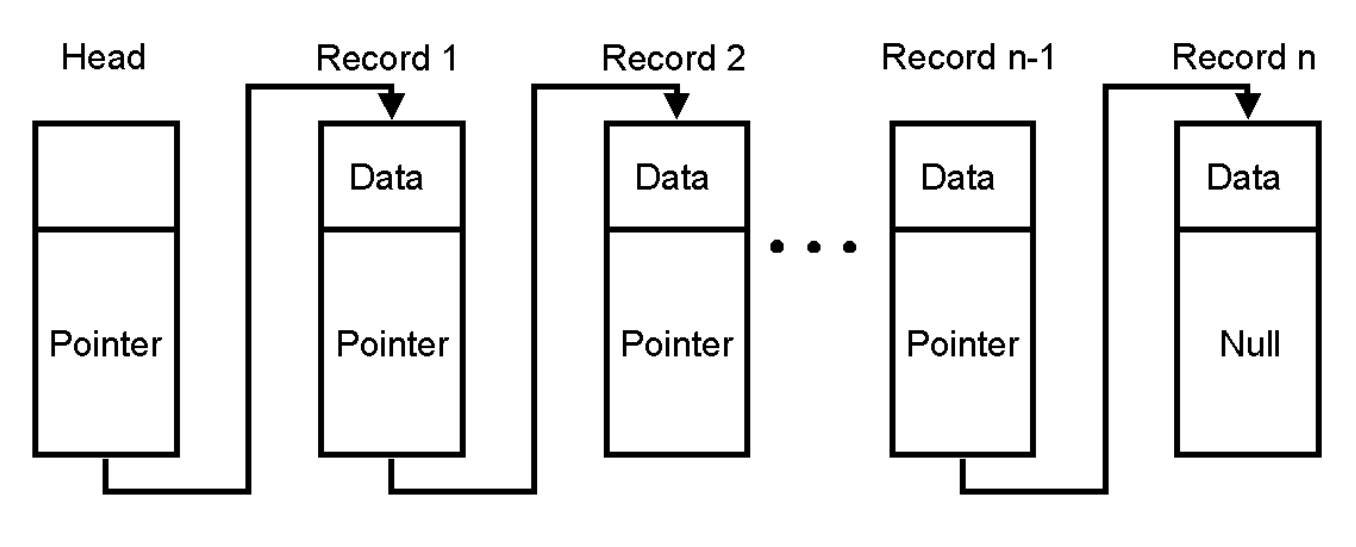
```
#include <iostream>

using namespace std;

int main () {
   int  var = 20;   // actual variable declaration.
   int  *ip;        // pointer variable

   ip = &var;       // store address of var in pointer variable

   cout << "Value of var variable: ";
   cout << var << endl;

   // print the address stored in ip pointer variable
   cout << "Address stored in ip variable: ";
   cout << ip << endl;

   // access the value at the address available in pointer
   cout << "Value of *ip variable: ";
   cout << *ip << endl;

   return 0;
}
```

Plačiau: https://www.tutorialspoint.com/cplusplus/cpp_pointers.htm

Tiesinis sąrašas (realizacija naudojant rodykles)



Sąrašo pavyzdys



Sąrašo pradžios rodyklė turi pirmojo sąrašo elemento adresą, kuriuo pradedant galima nuosekliai pereiti per visus sąrašo elementus.

Paskutinio sąrašo elemento tuščia rodyklė rodo, kad pasiektas paskutinis sąrašo elementas.

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

Tiesinis sąrašas

Tiesinis sąrašas – tai duomenų aibė, kuriai apibrėžtos tokios operacijos:

- *Sukurti tuščią sąrašą.*
- *Patikrinti, ar sąrašas tuščias.*
- *Patikrinti, ar sąrašas pilnas.*
- *Suskaičiuoti sąrašo elementus.*
- *Gauti n-tojo sąrašo elemento duomenis.*
- *Įterpti naujus duomenis (naują elementą) prieš n-tąjį elementą.*
- *Panaikinti n-tąjį sąrašo elementą.*
- *Rasti sąrašo elemento numerį su nurodytais duomenimis.*
- *Išvesti sąrašo elementus.*

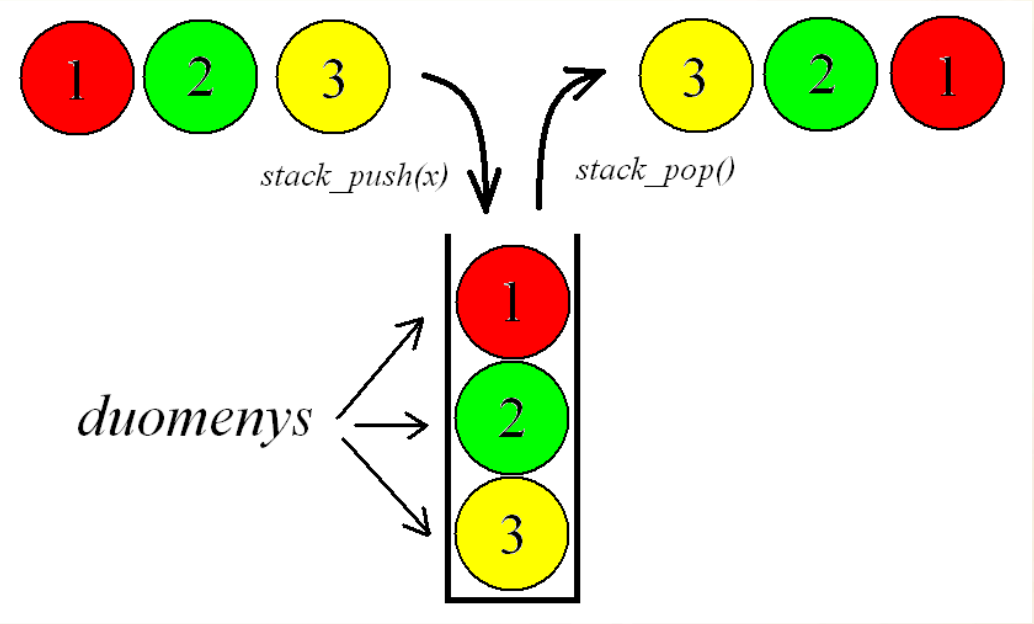
Dėklas (stack)

Duomenų struktūra stekas – tai duomenų aibė, kuriai apibrėžtos tokios operacijos:

- *inicializuoti dėklą* (išskirti vietą stekui kompiuterio atmintyje);
- *įterpti elementą* x į *dėklą* (operacija stack_push(x));
- *pašalinti elementą* iš *dėklo* (operacija stack_pop());
- *skaityti dėklą*;
- *panaikinti dėklą* (stack_clear()) – panaikinti vietą dėklui kompiuterio atmintyje.

LIFO principas

- Last-In-First-Out



- Operacijos stack_pop() metu iš dėklo bus pašalintas elementas, kuris buvo įterptas paskutinis.

Dėklas naudojamas

- kompiuterio aparatūroje,
- programose,
- operacinių sistemų architektūroje,
- kompiliatoriuose,
- loginių ir simbolinių skaičiavimų algoritmuose.

Dėklo taikymo pavyzdys

Užrašykime dėklo operacijų seką skaičiuojant aritmetinio reiškinio 9*(((5+8)+(8*7))+3) reikšmę:

- stack_push(9);
- stack_push(5);
- stack_push(8);
- stack_push(stack_pop() + stack_pop());
- stack_push(8);
- stack_push(7);
- stack_push(stack_pop() * stack_pop());
- stack_push(stack_pop() + stack_pop());
- stack_push(3);
- stack_push(stack_pop() + stack_pop());
- stack_push(stack_pop() * stack_pop()).

Dėklo turinio kitimas

					7						
		8		8	8	56		3			
	5	5	13	13	13	13	69	69	72		
9	9	9	9	9	9	9	9	9	9	648	NULL

Realizacija

Programuojant dėklo operacijas C++ kalba galima taikyti vieną iš 2 *bazinių tipų*:

- rodyklės tipą (rekomenduojama),
- masyvo tipą.

Eilutė (queue)

Duomenų struktūra eilutė – tai duomenų aibė, kuriai apibrėžtos tokios operacijos:

- *inicializuoti eilutę* (išskirti vietą eilutei kompiuterio atmintyje);
- *įterpti tam tikrą elementą x į eilutę;*
- *pašalinti elementą x iš eilutės;*
- *skaityti eilutę;*
- *panaikinti eilutę* (panaikinti vietą eilutei kompiuterio atmintyje).

Eilutė

- Duomenims eilėje taikomi *loginiai apribojimai*.
- FIFO (**First-In-First-Out**) principas.
- Eilutės struktūra taikoma:
  - algoritmams realizuoti;
  - programų loginėms schemoms;
  - operacinių ir taikomųjų sistemų architektūroje;
  - kompiliatoriams;
  - loginių ir simbolinių skaičiavimų algoritmams;
  - matematinio modeliavimo uždaviniams ir t. t.
- Programuojant eilutės operacijas taikomas vienas iš 2 bazinių tipų:
  - rodyklės tipas,
  - masyvo tipas.

Abipusis dėklas (deque, deck)

Duomenų struktūra abipusis dėklas – tai duomenų aibė, kuriai apibrėžtos tokios operacijos:

- *inicializuoti abipusį dėklą* (išskirti vietą dekui kompiuterio atmintyje);
- *įterpti elementą x į abipusio dėklo pradžią;*
- *įterpti elementą x į abipusio dėklo pabaigą;*
- *pašalinti elementą iš abipusio dėklo pradžios;*
- *pašalinti elementą iš abipusio dėklo pabaigos;*
- *skaityti abipusio dėklo pradžią;*
- *skaityti d abipusio dėklo pabaigą;*
- *panaikinti abipusį dėklą* (panaikinti vietą abipusio dėklo kompiuterio atmintyje).

Abipusis dėklas

- Duomenims saugoti abipusiame dėkle taikomi *loginiai apribojimai*.
- Sudėtingesnė realizacija.
- Programuojant abipusio dėklo operacijas taikomas vienas iš 2 bazinių tipų:
  - rodyklės tipas,
  - masyvo tipas.

Ačiū už dėmesį.

Klausimai?