Datu struktūras

Datu struktūra ir klase, kura norāda ar kādām metodēm tiek apvienoti datu vienumi, nodrošinot piekļuvi to apstrādei, uzglabāšanai.

Algoritms + Datu struktūra = Programma (N.Virts).

Izvēloties datu struktūru iegūst efektīvu vai neefektīvu programmu.

Datu tipi:

- Lineāri : elementi veido lineāru sarakstu. Piemēram, saraksts, steki un rindas.
- Nelineāri, elementu saraksts sazarojas. Piemēram: grafs un koks.

Noteikta uzdevuma veikšanai Python ir iebūvētas 4 datu struktūras:

Saraksts [List]: [sakārtotu, maināmu, dublējamu, viena veida indeksētu elementu kopa (masīvam līdzīgas struktūras)**]**.

Kortežs (Tuple): (sakārtotu, nemaināmu, dublējamu viena veida neindeksētu elementu kopa (nemaināms saraksts)**)**

Kopa {Set}: {nesakārtotu, nemaināmu, nedublējamu neindeksētu objektu kopa, var pielikt/noņemt vienumus**}**

Vārdnīca {Dictionary}: {nesakārtotu (Līdz Python 3.6 versijai, no 3.7 sakārtotu), maināmu, nedublējamu atslēgu/vērtību pāru kolekcija**}**

Saraksts [List]

[sakārtotu, maināmu, dublējamu, viena veida indeksētu elementu kopa (masīvam līdzīgas struktūras)].

Masīvam līdzīgas struktūras, savstarpēji saistītu objektu uzglabāšanai, kuru bieži nav jāpārskata, pievienot var tikai saraksta beigās,

```
x=[3,5]
print (type(x))
```

```
<class 'list'>
```

Populārākās metodes:
len() – nosaka saraksta garumu
append() – pielikt elementu, dinamiski palielinot sarakstu
pop() – noņemt elementu , dinamiski samazinot elementu
Vingrinājumi:

1. Izveidot sarakstu no 3 elementiem, pievienot pēdējo elementu, nodzēst otro elementu. Pēc katras darbības izdrukāt sarakstu funkcijā

```
def drukaSarakstu(saraksts):
    print("saraksts garums = ",len(saraksts))
    for x in saraksts:
        print(x)
    print()

saraksts = ["Vārna","Gulbis", "Žagata"]
drukaSarakstu(saraksts)
# pievienot sarakstam vienibu
saraksts.append("Kaija")
drukaSarakstu(saraksts)
#no saraksta izņem 3.elementu
saraksts.pop(2)
drukaSarakstu(saraksts)

skaitli = [1,2,3]
drukaSarakstu(skaitli)
```

```
saraksts garums =  3
Vārna
Gulbis
Žagata

saraksts garums =  4
Vārna
Gulbis
Žagata
Kaija

saraksts garums =  3
Vārna
Gulbis
Kaija

saraksts garums =  3
1
2
3
```

Uzdevumi:

1. Izveidojiet sarakstu ar 10 blakus stāvošiem veseliem skaitļiem. Tiem, kuriem žurnālā pāra skaitlis – atstājiet tikai pāra skaitli, tiem, kuriem nepāra skaitlis atstājiet tikai nepāra skaitli.
2. Izdrukājiet lielāko un mazāko skaitli.

Padziļināti

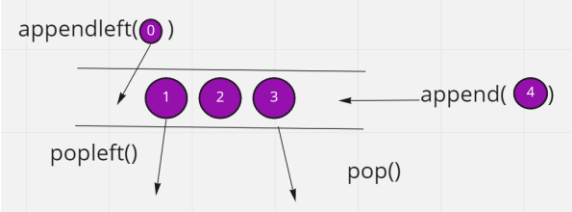
3. Izveidojiet divus viena tipa sarakstus un apvienojiet tos.

Rindas

saraksta neindeksēts atvasinājums
Ērti apstrādāt tiešsaistes pasūtījumus, uzglabāt balss pastu, koplietot resursus, piemēram, printeri vai CPU kodolu, jo elementi sakārtoti hronoloģiskā secībā.
Rindas queue (First In First Out) - metodes append() un pop() neefektīvas, jo jāpārbīda visa rinda, tāpēc izmanto divvirzienu rindu deque ar metodēm:

- **append()** pielikt elementus rindas labajā pusē
- **appendleft()** pielikt elementus rindas kreisajā pusē

- **pop()** noņemt rindas labajā pusē
- **popleft()** noņemt rindas kreisajā pusē
- **rotate()** apgriež rindas secību



Vingrinājumi:

1. Divvirzienu rindā ielikt sarakstu ar 3 elementiem, pielikt rindas beigās vienu elementu, rindas sākumā noņemt vienu elementu, pakāpeniski pielikt rindas sākumā 3 elementus vienu pēc otra, pēc katras darbības ar rindu izdrukāt to

```python
from collections import deque
q = deque() #izveido divvirzienu rindu

q.append(["Vārna","Gulbis", "Žagata"])
print("Rinda= ",q, "\n")

# pievienot sarakstam elementu
q.append("Kaija")
print('append("Kaija") =',q)

# dzeest no kreisas
q.popleft()
print("\nq.popleft() = ",q)

# pievieno kreisaja
q.appendleft([1,2])
print("\nq.appendleft([1,2]) = ",q)
q.appendleft(3)
print("\nq.appendleft(3) = ",q)
q.appendleft(4)
print("\nq.appendleft(4) = ",q)
```

```
Rinda=  deque([['Vārna', 'Gulbis', 'Žagata']])

append("Kaija") = deque([['Vārna', 'Gulbis', 'Žagata'], 'Kaija'])

q.popleft() =  deque(['Kaija'])

q.appendleft([1,2]) =  deque([[1, 2], 'Kaija'])

q.appendleft(3) =  deque([3, [1, 2], 'Kaija'])

q.appendleft(4) =  deque([4, 3, [1, 2], 'Kaija'])
```

2. Turpināt ar to pašu sarakstu: izmest rindas pēdējo elementu, izmest rindas pirmo elementu, apgriez rindas elementus pretējā virzienā, pēc katras darbības ar rindu izdrukāt to

```python
print("\nPirms dzēšanas = ",q)

# Dzesh
q.pop()
print("\nq.pop() = ",q)
q.popleft()
print("\nq.popleft() = ",q)

q.rotate()
print("\nq.rotate() = ",q)
```

```
Pirms dzēšanas =  deque([4, 3, [1, 2], 'Kaija'])

q.pop() =  deque([4, 3, [1, 2]])

q.popleft() =  deque([3, [1, 2]])

q.rotate() =  deque([[1, 2], 3])
```

Uzdevumi:

1. Izveidojiet rindu ar 10 blakus stāvošiem veseliem skaitļiem. Tiem, kuriem žurnālā pāra skaitlis – atstājiet tikai pāra skaitli, tiem, kuriem nepāra skaitlis atstājiet tikai nepāra skaitli.
2. Izdrukājiet lielāko un mazāko skaitli.

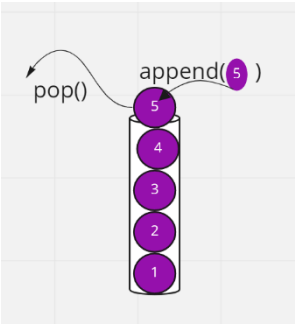
Padziļināti

- https://clevercode.lv/task/show/kartupelis
- https://clevercode.lv/task/show/histogram

Steks

saraksta neindeksēts atvasinājums

Piedāvā ‘Last In First Out’ datu pārvaldību, tas kurš pēdējais ienācis – pirmais izies.



Automātiska objekta mērogošana un tīrīšana, uzticama datu uzglabāšanas sistēma, bet ar ierobežotu atmiņu, kura var radīt steka pārpildes kļūdu. Izmanto, veidojot ļoti reaktīvas sistēmas, lai vispirms apstrādātu jaunākos pieprasījumus, iekavu saskaņošanai, Ctrl+Z funkcijas izpilde redaktoros.

Biežāk izmantojamās metodēs:

- **len()** nosaka elementu skaitu stekā
- **append ()** pieliks steka beigās elementu, kurš iekavās (*nevis push()*)
- **pop()** izmetīs augšējo elementu

Vingrinājumi:

1. Stekā ielikt sarakstu no 2 elementiem, pievienot atverošo iekavu, pievienot aizverošo iekavu, izdrukāt visu steku. Pa vienam elementam izmest no steka, to izdrukājot bez cikliskā operatora.

main.py

```
1  steks = [7,9]  # pinicialize
2  print(steks)
3  steks.append('(')
4  steks.append(')')
5
6  print(steks)
7  print(" stekā iekavu '(' = ",steks.count('('))
8
9  print ("\nStekā ir:")
10 print(steks.pop())
11 print(steks.pop())
12 print(steks.pop())
13 print(steks.pop())
```

Console Shell

```
[7, 9]
[7, 9, '(', ')']
 stekā iekavu '(' =  1

Stekā ir:
)
(
9
7
```

2. Pārbaudīt vai dotajai simbolu virknei ir pareizi saliktas apaļās iekavas:

Uzdevumi:

1. Izmantojot steku, izveidojiet funkciju min().
2. Izveidojiet virkni ar 2 veida iekavām, izmantojot steku, noteikt vai visas iekavas saliktas ievērojot matemātikas likumu.

Padziļināti

- https://clevercode.lv/task/show/histogram
- Aplūkot Postfix izmantošanas iespējas.

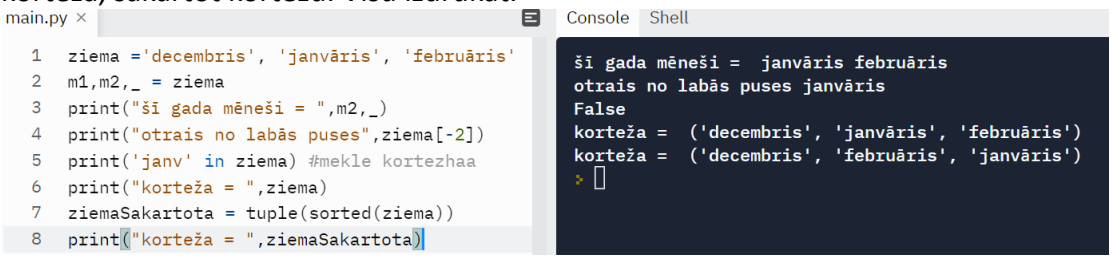
Kortežs (Tuple)

(sakārtotu, nemaināmu, dublējamu viena veida neindeksētu elementu kopa (nemaināms saraksts))

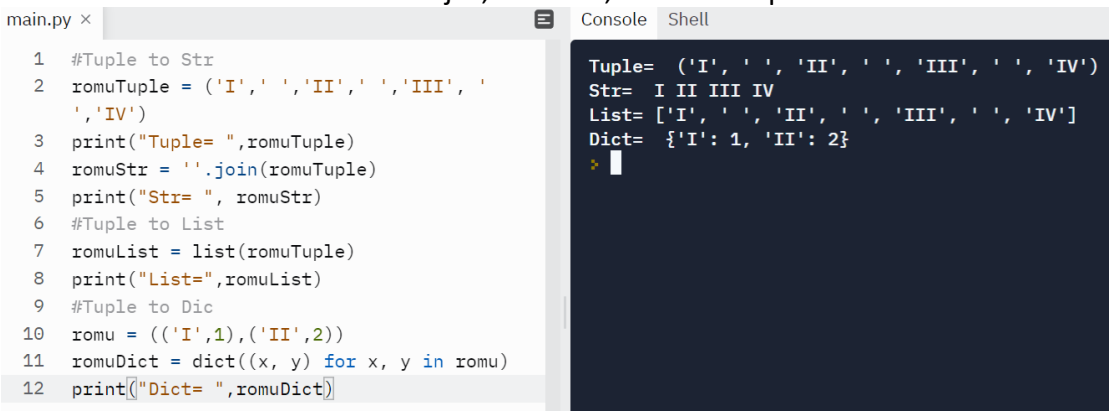
Korteži strādā ātrāk un drošāk par sarakstiem, jo to datus nevar mainīt, korteža datus ieliek **apaļās iekavās** vai arī var nelikt iekavās.. Izmanto kā atslēgu vārdnīcām.

Vingrinājumi:

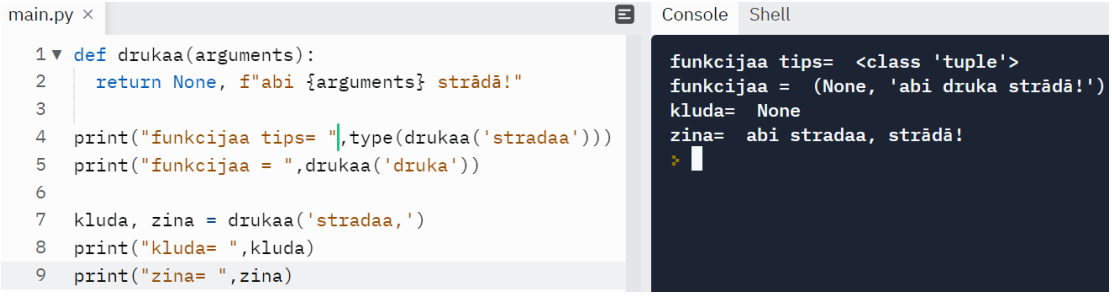
1. Izveidot kortežu no 3 elementiem, ielikt divos mainīgajos pirmos 2 mēnešus, trešo var nenorādīt, bet tad jābūt pasvītrojuma zīmei, atrast savu mēnesi kortežā, sakārtot kortežu. Visu izdrukāt.



2. Pārvērst korteža datus: simboliskajos, saraksta, vārdnīcas tipā:



3. Izveidot funkciju, kura atgriež divas vērtības:



Uzdevums

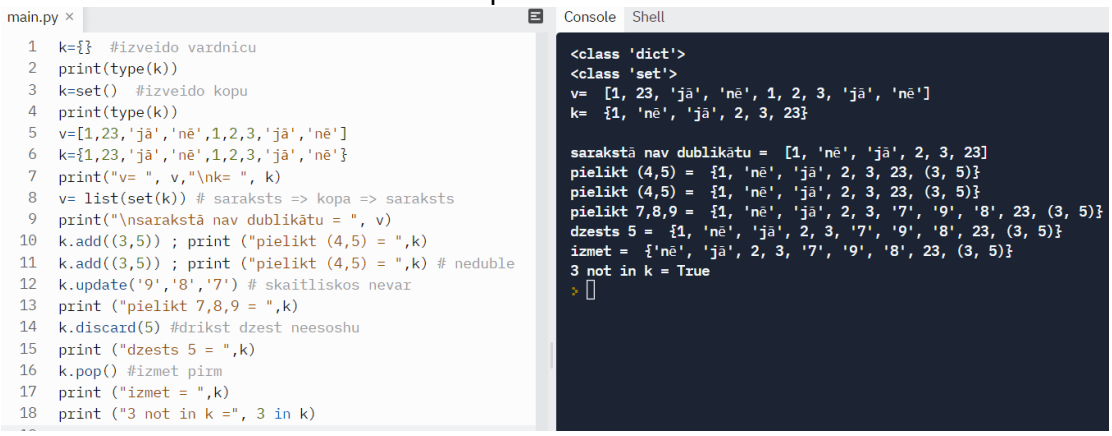
1. Izveidot kortežu mēneši un atrodot savu dzimšanas mēnesi tajā izdrukāt to un mēneša kārtas numuru, izmantojot metodi index()

Kopa {Set}:

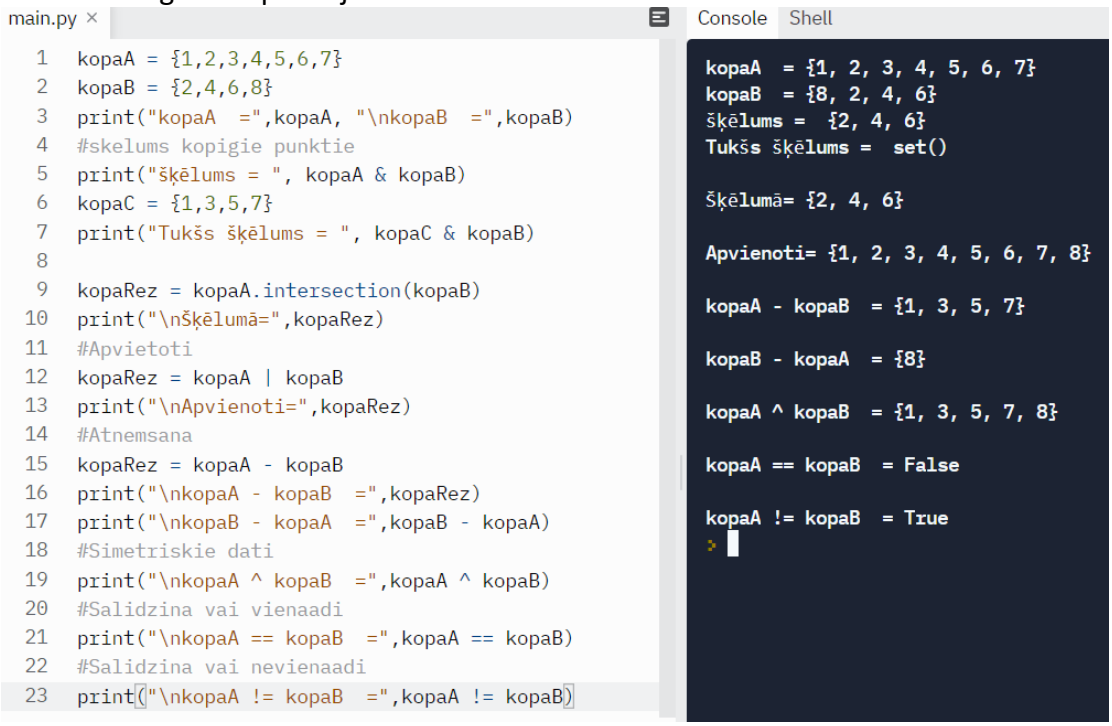
{ unikālu, nesakārtotu elementu kolekcija, ielikta figūriekavās }
Par kopas elementu var būt jebkurš nemainīgs datu tips (skaitļi, virknes, korteži). Ar kopām var meklēt apvienojumus, šķēlumus,

Vingrinājumi

1. Pārbaudīt elementārās darbības ar kopām



2. Pārbaudīt loģiskās operācijas



Uzdevums

1. Komentēt 2.vingrinājuma kopu apvienošanas īpatnības, salīdzinot ar izņemtajām datu struktūrām.

Vārdnīca {Dictionary}: -

{unikālu, nesakārtotu elementu kopa, kurā piekļuve vērtībām caur atslēgu}

Vingrinājumi

Izpētīt darbu ar vārdnīcu

main.py

```
1  vardnica = {}
2  vardnica = dict.fromkeys([1,2,3])
3  print(vardnica)
4  vardnica = dict.fromkeys([1,2,3],"x")
5  print("\nvienas vērtības",vardnica)
6  tulko = {
7    'list':  'saraksts',
8    'tuple': 'kortežs',
9    'set':   'kopa',
10   'dict':  'vārdnīca'
11   }
12 print("\nvisa vārdnīca= ",tulko)
13 print("\nizdrukā vienu vērtību = ",tulko['set'])
14 tulko['stack'] = 'steks' #pieliek vardnicai
15
16 print("\nvisa vārdnīca ar for: ")
17 for x in tulko:
18   print(x," = ",tulko[x])
19 print("\nvārdnīcā ",len(tulko), "ieraksti")
20 print("\nvai ir sarakstā = ",'list' in tulko)
```

Console Shell

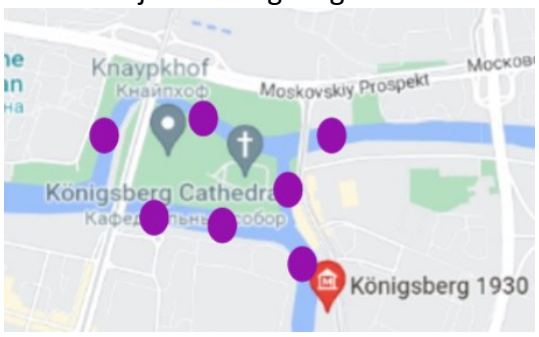
```
{1: None, 2: None, 3: None}

vienas vērtības {1: 'x', 2: 'x', 3: 'x'}

visa vārdnīca=  {'list': 'saraksts', 'tuple': 'kortežs
', 'set': 'kopa', 'dict': 'vārdnīca'}

izdrukā vienu vērtību =  kopa

visa vārdnīca ar for:
list  =  saraksts
tuple  =  kortežs
set  =  kopa
dict  =  vārdnīca
stack  =  steks

vārdnīcā  5 ieraksti

vai ir sarakstā =  True
```

Uzdevums:

Izveidot vārdnīcu: lai uzrakstot atzīmi iegūtu skaidrojumu: 10-izcili, 9-teicami, 8 – ļoti labi, 7 – labi,6 – gandrīz labi, 5 – viduvēji, 4- gandrīz viduvēji,…

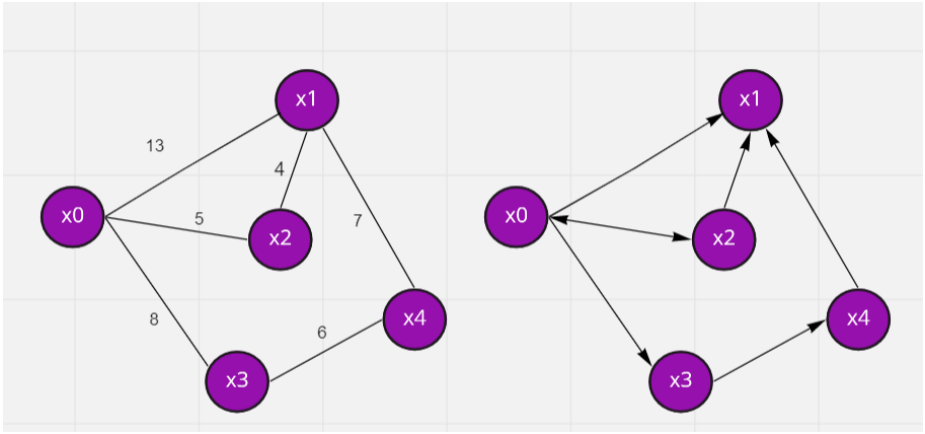
Grafi

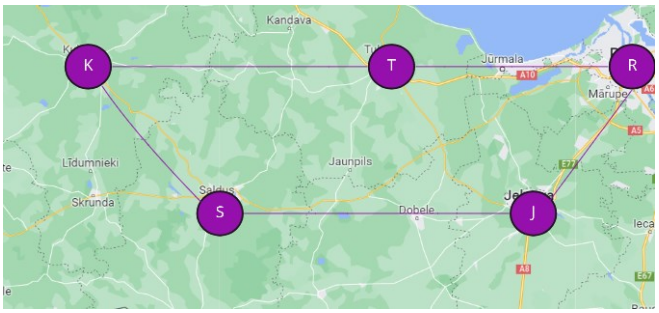
Par grafu sauc nelineāru datu struktūru, kura sastāv no objektiem un to savienojumiem. Ļoti daudzas reālās sistēmas matemātiski abstrahējot uz objektiem un to savienojumiem iegūst grafu.

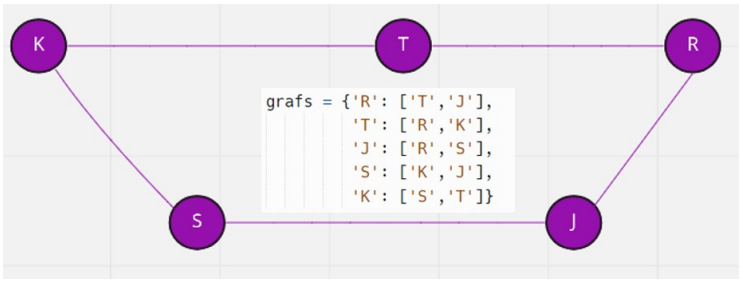


1736. gadā L. Eilers uzsāka grafu teoriju - pierādot, ka pa septiņiem Kēnigsbergas pilsētas tiltiem nevar no viena punkta izejot atgriezties tajā, ja pa katru tiltu iet tikai vienu reizi.

Grafs kā matemātisks objekts sastāv no virsotnēm (V) un malām/šķautnēm (E) veidojot sakārtotu kopu pāri G = (V, E). Mezgls ir tikai kāds objekts, un mala ir savienojums starp tiem. Kreisajā pusē ir neorientēts grafs, tas ir, bez skaidriem virzieniem, un malas šeit ir divvirzienu. Labajā pusē ir orientēts grafs, kuru malām ir noteikts virziens.







Meklēšana plašuma (*Breadth-first search*)

nesvērtam grafam:

- Pareizi aprakstīt kā virsotnes saistītas.
- Importēt bibliotēkas sadaļu darbam ar rindām
- Izveidot vārdnīcu, kur virsotne ir atslēgaar kuru iegūt saistītās virsotnes, t.i. vērtību
- Funkcijā
  - sākuma virsotni ielikt rindā
  - veidot apmeklēto virsotņu vārdnīcu
  - kamēr rinda nav tukša
    - izņemt pirmo virsotni un iegūt saistītās virsotnes, ja
    - tās nav apmeklēto vārdnīcā, tad ieliek vārdnīcā
- ierakstīt info no kurienes atnāca uz norādīto virsotni

main.py ×

```python
from collections import deque

grafs = {'R': ['T','J'],
         'T': ['R','K'],
         'J': ['R','S'],
         'S': ['K','J'],
         'K': ['S','T']}
def bfs(no , uz, grafs):
  q = deque([no])
  Bija = {no : None}
  while q:
    virsotne = q.popleft() #virsotne
    if virsotne == uz:
      break
    nakamVirsotnes = grafs[virsotne]
    for v in nakamVirsotnes:
      if v not in Bija:
        q.append(v)
        Bija[v]=virsotne
  return Bija

no = 'R'
uz = 'K'
Bija = bfs(no, uz, grafs)

print(f'no {no} līdz {uz}: \n{uz} ', end=" ")
lidz = uz
while lidz != no:
  lidz = Bija[lidz]
  print(f'<--- {lidz}', end='')
```

Console Shell

```
no R līdz K:
K  <--- T<--- R>
```

Meklēšana dziļumā (Depth-first search, DFS)

Parasti izmanto neizpētītu grafu apstrādei, ar rekursijas palīdzību izpēta visas virsotnes, tad atgriežas sākumpunkta un no citas blakus virsotnes pēta rekursijā:

1. Dodieties uz kādu blakus virsotni, kas iepriekš nav apmeklēta.
2. No šīs virsotnes palaist dziļuma pirmās meklēšanas algoritmu
3. Atgriezties uz sākuma virsotni.
4. Atkārtojiet 1.-3. darbību visām iepriekš neapmeklētajām blakus virsotnēm.

Deikstra algoritms

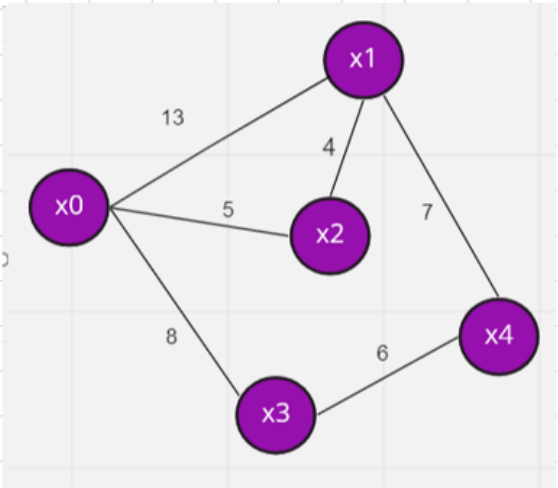
Īsāko ceļu no viena punktu līdz visiem citiem var noteikt ar Deikstra algoritmu, kuru viņš izdomāja 1959.gadā, lai ietaupītu laiku apmeklējot veikalus ar sievu. Ar grafa palīdzību rēķina informācijas paku maršrutizētāja ceļu.

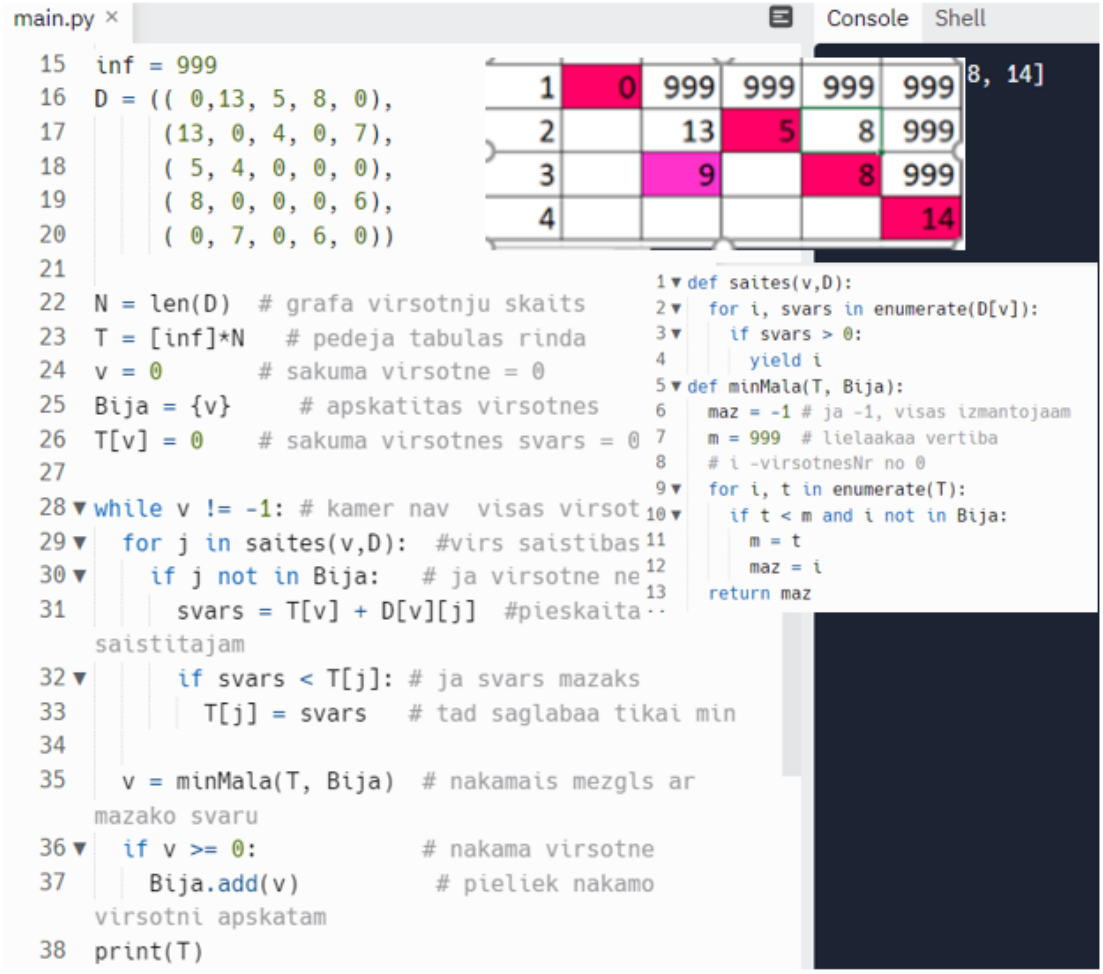
Deikstra algoritms:

1. Izvēlas vienu no sākumu virsotnēm, kurai attālums līdz sevīm 0, pārējiem bezgalība, (šeit 999), doto virsotni pievieno pie izmantotajām virsotnēm
2. Kamēr neizmanto visas virsotnes
   - atrod vismazāko iegūto attālumu, kuru pieskaita visām saistītajām virsotnēm no dotās virsotnes, ja tur lielāks skaitlis, tad to samaina ar iegūto skaitli
   - nesaistītajām virsotnēm pārraksta nemainītu attālumu
   - aplūkoto virsotni pievieno izmantoto virsotņu sarakstam

	x0	x1	x2	x3	x4
x0	0	13	5	8	0
x1	13	0	4	0	7
x2	5	4	0	0	0
x3	8	0	0	0	6
x4	0	7	0	6	0

Nr	x0	x1	x2	x3	x4
1	0	999	999	999	999
2		13	5	8	999
3		9		8	999
4					14





Uzēdums:
Ir pieci biroji piecās dažādās pilsētās. Ceļojuma izmaksas starp katru pilsētu pāri ir zināmas. Noteikt lētākais veids, kā sasniegt katru biroju no jebkura izvēlētā biroja.

Koks ir grafa zars.

Katram kokam ir saknes mezgls, no kura atzarojas visi pārējie mezgli. Saknē ir norādes uz visiem tieši zem tās esošajiem elementiem, kas ir zināmi kā tās pakārtotie mezgli. Mezglus bez pakārtotiem mezgliem sauc par lapu mezgliem.
Gan mezgli, gan malas var būt informācijas nesēji. Piemēram, ēkas ar ielām, kuras tās savieno.
Floida Voršala algoritms, lai atrisinātu visu pāru īsākā ceļa problēmu
Floida-Varšala algoritms ir algoritms īsākā ceļa atrašanai starp visiem virsotņu pāriem svērtā grafā. Šo algoritmu var izmantot gan virzītiem, gan nevirzītiem svērtiem grafikiem, bet ne ar negatīviem.