NAPOJ

UČNA PRIPRAVA

Poraba časa in časovna zahtevnost

DATUM NAPOJ

OSNOVNI PODATKI

Šola:

Letnik: 1. Letnik

Datum:

Predmet: Informatika

Učna tema: poraba časa, časovna zahtevnost

Učna enota: poraba časa algoritma, štetje operacij, časovna zahtevnost, redi časovne zahtevnost

Učne oblike:

- frontalno, individualno

Učne metode:

- razlaga, pogovor, demonstracija, reševanje problemov

Predznanje:

- poznajo pojme algoritem, funkcija, tabela

Operativni učni cilji

Ob koncu učne ure učenec:

- Razume kaj je ter pomen časovne zahtevnosti algoritma.
- Zna prešteti koliko operacij porabi algoritem.
- Razume in zna našteti rede časovnih zahtevnosti.
- Zna zapisati algoritem na različne načine.

Učna sredstva:

- Učila: delovni list, računalnik
- Učni pripomočki: računalnik, Python, zvezek, tabla

Didaktične etape učnega procesa:

1. Pripravljanje ali uvajanje
2. Obravnava nove učne snovi ali usvajanje
3. Urjenje
4. Ponavljanje
5. Preverjanje

Medpredmetne povezave: slovenščina, angleščina

Literatura:

- J. Gal-Ezer, T. Vilner, E. Zur, Teaching algorithm efficiency at CS1 level: A different approach, Computer Science Education 14 (3) (2004) 235–248.
- J. Gal-Ezer, E. Zur, The concept of “algorithm efficiency” in the high school CS curriculum, in: Proceedings of the 32nd ASEE/IEEE Frontiers in Education Conference, 2002.
- C. A. Shaffer, A practical introduction to data structures and algorithm analysis,
- Citeseer, 1997.
- M. A. Weiss, Data Structures and Algorithm Analysis (2nd Edition), Addison Wesley, 1994.
- M. T. Goodrich, R. Tamassia, Data structures and algorithms in Java, 2001.
- G. Anželj, J. Brank, A. Brodnik, P. Bulić, M. Ciglarič, M. Đukić, L. Furst, M. Kikelj, A. Krapež, H. Medvešek, N. Mori, M. Pančur, P. Sterle, Računalništvo in informatika 1, Založba Univerze na Primorskem and Založba Fakultete za računalništvo in informatiko and Založba Fakultete za

elektrotehniko and računalništvo in informatiko, 2015. URL https://lusy.fri.uni-lj.si/ucbenik/book/index.html. - Desmos - graphing calculator, Dostopno na: https://www.desmos.com/calculator, (pridobljeno 12. 7. 2018).
Novi pojmi: - poraba časa algoritma, časovna zahtevnost algoritma, redi časovne zahtevnosti.
Priloga: - Delovni list - Delovni list - rešitve

POTEK UČNE URE

UVODNI DEL: UVAJANJE

ČAS	UČITELJ	UČENEC	UČNE OBLIKE, METODE, TEHNIKE, UČNI PRIPOMOČKI
Uvod 5 min	*Pozdravi učence, zapiše manjkajoče učence in po potrebi prosi dežurnega učenca, da pobriše tablo.*	Odzdravijo, naštejejo manjkajoče učence, dežurni učenec pobriše tablo.	Frontalno.
Motivacija 5 min	Danes se bomo naučili oceniti časovno zahtevnost algoritma, to je koliko časa bo algoritem potreboval, da se bo izvedel. Logično je, da si želimo, da so naši programi čim hitrejši. Poglejmo nekaj primerov, zakaj je temu tako: Koliko časa potrebujemo za avtomatično ocenjevanje esejev pri slovenščini? Za iskanje zadetkov v spletnem iskalniku? Za samodejno zaviranje avtomobila, če le ta oceni, da se lahko zgodi nesreča? Ali lahko še vi podate nekaj primerov, kjer se splača uporabiti hitrejše algoritme? Zaradi teh vprašanj so se bistre glave spomnile, da bi lahko analizirali zahtevnost algoritma – koliko časa in koliko prostora porabi, da se izvede. Mi se bomo osredotočili le na porabo časa.	Poslušajo. Odgovorijo, naštejejo primere.	Frontalno, razlaga, pogovor.

GLAVNI DEL: OBRAVNAVANJE UČNE SNOVI

VSEBINSKI POUDARKI	UČITELJ	UČENEC	UČNE OBLIKE, METODE, TEHNIKE, UČNI PRIPOMOČKI

Merjenje časa izvajanja programa v Pythonu
 	
20 min
	Snov bomo spoznali skozi 3 načine ocenitve časovne zahtevnosti:
	
	1.) Z merjenjem časa izvajanja programa v Pythonu.
	
	2.) S Štetjem števila operacij, ki jih program opravi.
	
	3.) Z vpeljavo reda časovne zahtevnosti.
	
	Najprej se naučimo, kako merimo čas izvajanja programa. Vaša naloga seda je, da napišete funkcijo poisciMin(sez), ki sprejme seznam števil ter vrne najmanjšega med njimi.
	
	`def poisciMin(sez):`
	`  min = 999 [1]`
	`  i = 0 [2]`
	`  l = len(sez) [3]`
	`  while (i < l): [4]`
	`  if sez[i] < min: [5]`
	`  min = sez[i] [6]`
	`  i = i + 1 [7]`
	`  return min [8]`
	
	Sedaj poglejmo, kako izmerimo čas trajanja te funkcije.
	Učitelj napiše spodnjo kodo v svoj programski vmesnik in jo pojasni.
	
	`import time`
	`sez = [1] * 1000000`
	`zacetek= time.time()`
	`poisciMin(sez)`
	`konec = time.time()`
	`cas = konec - zacetek`
	`print("Poraba casa:", cas, "milisekund.")`
	
	Za »igračkanje« je merjenje časa programa vse lepo in prav, na bolj »profesionalni« ravni (npr. v podjetjih) pa se vmešajo malenkosti, ki so na koncu zelo pomembne. Kaj mislite, ali bo hitrost programa na vseh računalnikih vedno ista? No, nekatere druge malenkosti so še:
	
	-Kako hiter je procesor računalnika, na katerem merimo čas.
		Spremljajo razlago in si zapišejo v zvezek.
		
		Rešijo nalogo.
		
		Dopolnijo svojo nalogo.
		
		Odgovorijo: ne.
			Frontalno, razlaga, pogovor, individualno ali skupinsko delo, reševanje problemov, demonstracija.

	-Kako hitri so ostali deli računalnika (hitrost prenosa podatkov med glavnim pomnilnikom in procesorjem). -Ali imamo med merjenjem odprte tudi druge programe, ki trošijo procesorski čas. -V katerem programskem jeziku smo napisali program. -Ali je program napisan optimalno. Tudi na našem programu lahko vidimo, da ne bo vedno izmerilo istega časa. Pa poskusimo. *Učitelj večkrat požene program, da učenci opazijo razliko v izpisu časa.* Zato se je začelo uporabljati drug sistem – štetje osnovnih operacij. Poglejmo si ali je ta sistem kaj boljši.		
Štetje osnovnih operacij 20 minut	Štetje osnovnih operacij pomeni prav to – preštejemo, koliko operacij naš program izvede. Med osnovne operacije štejemo: -določanje vrednosti spremenljivki (a=1) -klicanje metode (a = sestej(1,3)) -aritmetične operacije (2*2, 2+2,...) -primerjanje dveh vrednosti (1 <= 2) -iskanje elementa v seznamu (sez[i]) -izhod iz funkcije (return) Ker je teorijo vedno dobro podpreti s prakso, preštejmo število osnovnih operacij v našem programu: V vrsticah 1, 2 in 3 se spremenljivkam min, i in l dodeli vrednosti, kar se šteje kot 1 operacijo, torej skupno 3. V 4. vrstici imamo primerjavo dveh vrednosti, kar se šteje za 1 operacijo, a ker se ta primerjava izvede n-krat (zaradi same zanke), se to šteje kot n operacij. V 5. vrstici imamo iskanje elementa v seznamu (1	Spremljajo razlago in si zapišejo v zvezek.	Frontalno, razlaga, pogovor.

	operacija) in primerjava dveh vrednosti (1 operacija), skupno 2 operaciji. Ker se izvedeta v zanki, moramo pomnožiti z n, torej dobimo 2n. V 6. vrstici imamo **pogojno** 2 operaciji – iskanje elementa v seznamu in prirejanje vrednosti spremenljivki. Ker se izvedeta v zanki, moramo pomnožiti z n, torej dobimo 2n. V 7. vrstici imamo aritmetično operacijo seštevanja ter prirejanje vrednosti, torej 2 operaciji. Nazadnje imamo v 8. vrstici izhod iz funkcije, kar se šteje kot 1 operacijo. Ker se 6. vrstica izvede pogojno, je število operacij največ: t(n) = 3+n+2n+2n+2+1 = 5n+6, najmanj pa: t(n) = 3+n+2n+1(nujno se zgodi vsaj 2 zamenjava)+2+1 = 3n+7.		
	Kdaj se zgodi najboljši primer? Na katerem mestu v seznamu mora biti najmanjši element? Ali je bolj praktično vedeti število operacij (in posledično čas izvajanja) za najboljši ali najslabši primer? Primeri: -Ponudniki internetnih storitev morajo vedeti največ koliko strank lahko povežejo v svoje omrežje, da le-to ne bo preobremenjeno (usmerjati promet, itd.). -Za program za analiziranje letal v kontrolnem stolpu moramo natančno vedeti največ koliko letal lahko usmerja. Za to bomo od sedaj časovno analizo algoritmov poenostavili ter šteli število operacij le za najslabši primer. Tako bomo bili prepričani, da število operacij ne bo nikoli večje kot toliko, kolikor smo jih prešteli. Sedaj pa premislimo – ali je res potrebno vedeti natančno število operacij? Ali bo hitrost programa	Odgovorijo: ko je najmanjši element na prvem mestu. Odgovorijo: najboljši primer se zgodi le malokrat, medtem ko je ena od prednosti najslabšega primera ta, da program nikoli ne bo porabil več kot toliko časa, in je zato bolj varen.	

	bistveno počasnejša, če dodamo 1 ali 2 osnovni operaciji? V pomoč: kako hitro računalnik sešteje 2 števili?	Odgovorijo: ne, ker ni bistvene razlike.	

Red časovne zahtevnosti
 	
20 min
	Ker so današnji računalniki zelo hitri, nas načeloma ne zanima, ali je število operacij 2n+1 ali 2n+2 (oba primera se bosta izvedla skoraj enako hitro). Zato se raje osredotočimo na veliko sliko, to je, kako hitro čas izvajanja narašča glede na velikost primerka problema. Temu pravimo red časovne zahtevnosti in ga lahko kar označimo z velikim O. Za našo funkcijo poisciMin() lahko rečemo, da ima časovno zahtevnost reda O(n), ker čas izvajanja narašča sorazmerno z n-jem samim. Števila iz funkcije t(n) zanemarimo, zanemarimo pa tudi vse člene razen tistega, ki raste najhitreje.
	
	Podajmo nekaj primerov in jih pojasnimo:
	
	5n+2 spada v red časovne zahtevnosti O(n)
	
	5n^2+2 spada v red časovne zahtevnosti O(n^2)
	
	5n^2 + n + 2 spada v red časovne zahtevnosti O(n^2)
	
	2^n + n^3 + 1000 spada v red časovne zahtevnosti O(2^n)
	
	Ker nas torej ne zanima natančno število vseh operacij, ampak zgolj red časovne zahtevnosti, lahko pri analizi uvedemo nekaj enostavnih pravil:
	
	1.) Zanka: časovna zahtevnost zanke (bodisi while bodisi for), je enaka število iteracij krat časovna zahtevnost operacij v tej zanki. Primer:
	
	`for i in range(n):`
	`  if i < n:`
	`  print(i)`
	
	For zanka se izvede n-krat. Časovna zahtevnost operacij v zanki je t(n) = 1+1 € O(1). Torej je časovna zahtevnost celotne zanke O(n).
	
	2.) Gnezdene zanke: časovna zahtevnost je zmnožek iteracij zunanje zanke in notranje zanke. Primer:
	
	`for i in range(n):`
	`  for j in range(n):`
	`  s = i+j`
	
	Zunanja zanka potrebuje n iteracij, notranja zanka prav tako n iteracij, prirejanje vrednosti spremenljivki pa štejemo kot O(1). Tako imamo t(n)
		Sledijo razlagi in si zapisujejo.	Frontalno, razlaga, pogovor, demonstracija.

= n*n*1 = n^2 € O(n^2).

3.) Zaporedne operacije: Če imamo v funkciji zaporedne operacije, jih zgolj seštejemo.

```
for i in range(n):
    for j in range(n):
        s = i+j
for k in range(n):
    if k < n:
        print(k)
```

V zgornjem primeru imamo tako O(n) + O(n^2) = O(n^2 + n) = O(n^2).

4.) V primeru if-else stavka, upoštevamo le počasnejši del.

```
if n < 5:
    for k in range(n):
        print(k)
else:
    n = 5
```

V zgornjem primeru imamo v if delu stavka O(n), v else delu pa O(1). Upoštevali bomo počasnejši red zahtevnosti, to je O(n).

Za konec dodajmo še to, da poznamo več redov časovne zahtevnosti (tu so razporejene od najhitrejše do najpočasnejše!):

Učitelj razloži naslednje časovne rede.

Logaritmična: O(log(n))

Linearna: O(n)

Kvadratna: O(n^2)

Polinomska: O(n^k)

Eksponentna: O(a^n)

S pomočjo spletni strani https://www.desmos.com/calculator si poglejmo naraščanje porabe časa za te rede časovne zahtevnosti.

Učitelj na spletni strani izriše funkcije y=log(n), y=n, y=n^2, y=2^n in razloži, da X os predstavlja »n« (število primerkov – 1n, 2n, 3n, ...), y os pa čas. Lahko pa ne uporabi spletne strani in nariše graf na tablo.

ZAKLJUČNI DEL: ZAKLJUČNO PONAVLJANJE / PREVERJANJE

ČAS	UČITELJ	UČENEC	UČNE OBLIKE, METODE, TEHNIKE, UČNI PRIPOMOČKI
20 min	*Ponovi nove pojme in razdeli delovne liste, za katerega na koncu skupaj z učenci preveri rešitve.* Za konec vas prosim, da rešite delovni list, na koncu pa bomo pregledali vaše odgovore.	Poslušajo, rešijo delovni list, pregledajo rešitve.	Frontalno, individualno, razlaga, reševanje problemov, pogovor.
5 min	*Učitelj se zahvali učencem za sodelovanje, jim da napotke za domačo nalogo. Dežurni učenec naj po potrebi pobriše tablo. Učenci naj ugasnejo računalnike.*	Poslušajo, ugasnejo računalnike, dežurni učenec pobriše tablo.	Frontalno.

Priloge

Priloga 1: Delovni list

Delovni list – Poraba časa in časovna zahtevnost

1) Določite red časovne zahtevnosti za naslednje funkcije (v pomoč: za zadnje 3 si pomagajte s seznamom redov časovne zahtevnosti v zapiskih):

T(n) = 1000 Red čas. zahtevnosti:

T(n) = 5*n + 2 Red čas. zahtevnosti:

T(n) = 5*n^2 + 2 Red čas. zahtevnosti:

T(n) = 1*n^3 + 2*n^2 + 3 Red čas. zahtevnosti:

T(n) = 10 * log(n) Red čas. zahtevnosti:

T(n) = n^5 + log(n) Red čas. zahtevnosti:

T(n) = 2^n + n^5 + log(n) Red čas. zahtevnosti:

2) Za naslednje programe **v splošnem** opišite kaj program naredi, določite število operacij in podajte red časovne zahtevnosti.

```
def f(n):
    rez = n + 1
    return rez
```

Opis:

Št. operacij:

Ocena čas. zaht:

```
def f(n):
    rez = 0
    for i in range(n):
        rez = rez + 1
    return rez
```

Opis:

Št. operacij:

Ocena čas. zaht:

```
def f(n):
    rez = 0
    for i in range(n):
        for j in range(n):
```

Opis:

Št. operacij:

```
            rez = rez + 1
    return rez
```

Ocena čas. zaht:

```
def f(n):
    rez = 0
    for i in range(n):
        rez = rez + 1
    for j in range(n):
        rez = rez + 1
    return rez
```

Opis:

Št. operacij:

Ocena čas. zaht:

3) Za spodnjo funkcijo opišite kaj naredi, določite red časovne zahtevnosti, za konec pa še poskusite napisati svojo funkcijo, ki bo isto stvar naredila hitreje za eno stopnjo reda časovne zahtevnosti (primer: če za funkcijo velja O(n^2), napišite funkcijo, ki bo potrebovala O(n)).

```
def f(seznam):
    velikost = len(seznam)
    max = 0
    for i in range(velikost):
        for j in range(velikost):
            if ( abs(seznam[i] - seznam[j] > max) ):
                max = abs(seznam[i] - seznam[j])
    return max
```

Opis funkcije:

Red časovne zahtevnosti:

Izboljšana verzija programa:

Priloga 2: Delovni list - rešitve

Delovni list – Poraba časa in časovna zahtevnost

1) Določite red časovne zahtevnosti za naslednje funkcije (v pomoč: za zadnje 3 si pomagajte s seznamom redov časovne zahtevnosti v zapiskih):

T(n) = 1000 — Red čas. zahtevnosti: O(1)

T(n) = 5*n + 2 — Red čas. zahtevnosti: O(n)

T(n) = 5*n^2 + 2 — Red čas. zahtevnosti: O(n)

T(n) = 1*n^3 + 2*n^2 + 3 — Red čas. zahtevnosti: O(n^3)

T(n) = 10 * log(n) — Red čas. zahtevnosti: O(log(n))

T(n) = n^5 + log(n) — Red čas. zahtevnosti: O(n^5)

T(n) = 2^n + n^5 + log(n) — Red čas. zahtevnosti: O(2^n)

2) Za naslednje programe **v splošnem** opišite kaj program naredi, določite število operacij in podajte red časovne zahtevnosti.

```
def f(n):
    rez = n + 1
    return rez
```

Opis: sešteje prejeto število z 1 in ga vrne

Št. operacij: 3

Ocena čas. zaht: O(1)

```
def f(n):
    rez = 0
    for i in range(n):
        rez = rez + 1
    return rez
```

Opis: Sešteje števila od 1 do n in vrne rezultat

Št. operacij: 2*n +2

Ocena čas. zaht: O(n)

```
def f(n):
    rez = 0
    for i in range(n):
        for j in range(n):
            rez = rez + 1
    return rez
```

Št. operacij: 2n^2 + 2

Ocena čas. zaht: O(n^2)

```
def f(n):
    rez = 0
    for i in range(n):
        rez = rez + 1
    for j in range(n):
        rez = rez + 1
    return rez
```

Št. operacij: 4n + 2

Ocena čas. zaht: O(n)

3) Bonus naloga: Za spodnjo funkcijo opišite kaj naredi, določite red časovne zahtevnosti, za konec pa še poskusite napisati svojo funkcijo, ki bo isto stvar naredila hitreje za eno stopnjo reda časovne zahtevnosti (primer: če za funkcijo velja O(n^2), napišite funkcijo, ki bo potrebovala O(n)).

```
def f(seznam):
    velikost = len(seznam)
    max = 0
    for i in range(velikost):
        for j in range(velikost):
            if ( abs(seznam[i] - seznam[j] > max) ):
                max = abs(seznam[i] - seznam[j])
    return max
```

Opis funkcije: Vrne razliko največjega in najmanjšega števila iz prejetega seznama.

Red časovne zahtevnosti: O(n^2); n je velikost seznama

Izboljšana verzija programa:

```
def f(seznam):
    min = seznam[0]
    max = seznam[0]
    for i in range(1, len(seznam)):
        if seznam[i] < min:
            min = seznam[i]
        if seznam[i] > max:
            max = seznam[i]
    return max-min
```