SVEUČILIŠTE U ZAGREBU
PRIRODOSLOVNO–MATEMATIČKI FAKULTET
MATEMATIČKI ODSJEK

Filip Orešić, Nina Salaj

GENLANG

Zagreb, lipanj, 2023.

Sadržaj

Sadržaj **iii**

Uvod **1**

1 Jezik **2**
1.1 Implementacija 2
1.2 Osnovna sintaksa 2
1.3 Učitavanje i ispis varijabli 3
1.4 Aritmetički i logički operatori 3
1.5 Grananje 3
1.6 Petlje 4
1.7 Funkcije 4
1.8 Liste 5
1.9 Rad s datotekama 6
1.10 Tip podataka Variant 7
1.11 Tip podataka Gen 7

2 Varijante gena **9**

3 Smith-Watermanov algoritam **11**

4 DODATAK **17**

Bibliografija **20**

Uvod

Genetika je znanstvena disciplina koja proučava naslijeđivanje, strukturu i funkciju gena te njihovu ulogu u razvoju organizama. Geni su osnovne jedinice naslijeđivanja i sastoje se od sekvenci nukleotida koji nose genetičku informaciju. No, uloga gena ne završava samo kod prenošenja naslijeđenih osobina. Geni igraju ključnu ulogu u kodiranju proteina, molekula od vitalne važnosti za sve žive organizme.

Proteini se sastoje od aminokiselinskih lanaca koji su kodirani genom. Svaka aminokiselina je definirana specifičnom sekvencom nukleotida u genu. Ta se sekvenca prevodi u lanac aminokiselina koji tvori specifičnu strukturu proteina. Poznavanje veze između gena i aminokiselina ključno je za razumijevanje funkcije proteina te kako genetske varijacije mogu utjecati na njihovu strukturu i funkciju.

U ovom ćemo radu ćemo objasniti implementaciju novog programskog jezika te pomoću njega istražiti važnost gena i aminokiselina te njihovu povezanost. Analizirat ćemo optimalno poravnanje niza aminokiselina i objasniti njegovu primjenu u istraživanju evolucije proteina. Također, fokusirat ćemo se na genetske varijante i njihovu ulogu u naslijeđivanju bolesti.

Poglavlje 1

Sadržaj i sintaksa jezika

1.1 Implementacija

Jezik je implementiran u programskom jeziku Python koristeći programski paket Vepar[1]. Iako je implementacija ostvarena u Pythonu, jezik nije interpretiran, nego se stvara skup naredbi koji se izvršava nakon što su sve naredbe spremljene, ali podržava interaktivni način unosa linije po liniju, te je dinamično tipkan, proceduralan i ne podržava objektno orijentirano programiranje, uključujući i nepostojanje struktura.

1.2 Osnovna sintaksa

Kod se piše unutar programa, nije potrebno pisati main funkciju. Svaka linija mora završiti s točkazarez ”;”, osim specijalnih slučaja. Nije potrebno navoditi tipove podataka, nego ih program sam deducira. U retku 2 podatak *x* je string, dok je u retku 3 broj. Komentari su jednolinijski, i pišu se kao ”—jednolinijski komentar”.

```
1 ---ovo je komentar
2 x="1";
3 x=1;
4 input(x);
```

1.3 Učitavanje i ispis varijabli

U varijable učitavamo preko konzole koristeći ključnu riječ input, dok ispisujemo u konzolu koristeći ključnu riječ output, te u zagradi broj, string, ili varijablu koju želimo ispisati. Prijelaz u novi red u ispisu se ostvaruje ključnom rijeći "NEWLINE".

```
input(x);
output(x);
```

1.4 Aritmetički i logički operatori

Višemjesni aritmetički operatori "+, −, ∗, /" i dvomjesni logički operatori "<,<=, >, >=, ==, !=, &, |" rade na standardan način, ali vrijednosti logičkih operacija ne možemo spremiti u varijablu. Operatori inkrementa "++" i "+ =" također rade na standardan način, ali se mogu koristiti samo u petljama.

```
---korektno, x=-9
x=-1+2/2-3*3;
---nije korektno
x=1<2;
```

1.5 Grananje

Grananje se izvršava ukoliko je logički uvjet istinit, oblika je

```
if(uvjet){
    naredba_1;
    .
    .
    .
    naredba_n;
}
```

Nakon znaka "}" koji označava kraj tijela grananja se ne piše ";" za kraj linije.

1.6 Petlje

Petlja se izvršava za dani broj iteracija, oblika je

```
for(var=pocetak; var<kraj; inkrement){
    naredba_1;
    .
    .
    .
    naredba_n;
}
```

Nakon znaka ”}” koji označava kraj tijela petlje se ne piše ”;” za kraj linije. Inkrement može biti tipa ”var++” ili ”var+=broj”.

1.7 Funkcije

Funkcije se deklariraju na način

```
foo=function(arg_1, ...., arg_n){
    naredba_1;
    .
    .
    .
    naredba_n;
};
```

Nakon znaka ”}” koji označava kraj tijela petlje se piše ”;” za kraj linije, dok return nije potreban osim ako želimo povratni tip. Funkcija se poziva na sljedeći način, te se taj način pozivanja koristi i za ugrađene funkcije poput append, sort, remove, itd.:

```
foo=function(arg_1, ...., arg_n){
    naredba_1;
    .
    .
    .
    naredba_n;
};
---zelimo spremiti povratan tip
x=call.foo(arg_1, ...., arg_n);
```

```
---ne zelimo spremiti povratan tip
call.foo(arg_1, ...., arg_n);
```

Za izvršavanje funkcije koristi se zasebna lokalna memorija zbog čega se argumenti funkcije prenose po vrijednosti (osim lista koje se prenose po referenci). Dopušteno je pozivati funkciju u funkciji.

```
x=0;
output(x);
f=function(x){
    x=1;
    output(x);
};
g=function(x){
    x=2;
    output(x);
    call.f(x);
};
output(x);
call.g(x);
output(x);
--- ispis je redom 0, 0, 2, 1, 0
```

1.8 Liste

Liste mogu biti jednodimenzionalne ili dvodimenzionalne. Prazna liste se inicijalizira na način da pridružimo nekoj varijabli funkciju ”collection()”, na kraj liste se dodaje koristeći ”append”, te je moguće pristupiti i mijenjati element liste na indeksu ”i” koristeći operator ”[]”:

```
---jednodimenzionalna lista
x=collection();
---na indeksu 1 je sada druga lista
append.x(1, collection());
---ispis broja 1
output(x[0]);
append.x[1](1,2,3);
---ispis [1,2,3]
output(x[1]);
```

```
---ispis [1,[1,2,3]]
output(x);
```

Na listima su definirane funkcije append, isEmpty, size, sort, reverse, clear, koje rade na standardan način, remove koja uklanja dani element, removeIndeks koja uklanja element na danom indeksu, te funkcija split na stringovima koja vraća listu kao povratni tip. Size i isClear imaju povratan tip, te je zato potrebno poziv funkcije koristiti samo s desne strane pridruživanja. Pridruživanje postojeće liste nekoj varijabli ne stvara novu listu, nego koristi referencu na postojeću listu.

1.9 Rad s datotekama

Jezik podržava rad s teksualnim datotekama. Prvo je datoteku potrebno otvoriti za određeni način, ”r” za čitanje, ”w” za pisanje koje će prebrisati stari sadržaj, te ”a” za pisanje koje će dodati novi sadržaj na kraj staroga. Na kraju korištenja je potrebno zatvoriti datoteku. Nakon toga, iz datoteke čitamo koristeći funkciju ”READOPTION”, kojoj je potrebno kao parametre poslati ime otvorene datoteke, te opcionalan drugi parametar. Ukoliko nije poslan drugi parametar, tada se čita cijeli sadržaj datoteke, ukoliko je parametar broj n, čita se n znakova, a ukoliko je parametar ”line”, čita se linija.

```
f = open("test.txt", "r");
---cita prvi znak
x = READOPTION(f, 1);
---cita ostatak prve linije
x = READOPTION(f, line);
---cita cijeli sadrzaj datoteke
x = READOPTION(f);
close(f);
```

U datoteku pišemo koristeći funkciju WRITEOPTION:

```
f = open("test.txt", "w");
WRITEOPTION(f, "test1");
close(f);
f = open("test.txt", "a");
```

```
WRITEOPTION(f, "test2");
---sadrzaj datoteke je sada test1test2
close(f);
```

1.10 Tip podataka Variant

Tip podataka Variant predstavljava varijante gena(vidi 2). Variant je potrebno stvoriti koristeći .csv datoteku koja u sebi sadrži podatke određenih varijanta za neki gen. Tip Variant ima funkcije consequencePercentage i diseasePercentage, koje računaju redom postotak ucinka i postotak bolest u danim podacima.

```
x = variant("ime_datoteke.csv");
y = diseasePercentage.x("bolest");
z = consequencePercentage.x("ucinak");
```

1.11 Tip podataka Gen

Tip podataka Gen predstavlja gen, te svaka njegova instanca predstavlja jedan gen. Gen je potrebno stvoriti koristeći string koji predstavlja aminokiseline, dok je ispis preopterećen tako da ispisuje aminokiseline tog gena.

```
x = gen("ATC");
---ispise "ATC"
output(x);
```

Na tipu Gen postoje tri operatora, višemjesni operator ”+” koji stvara novi gen koristeći aminokiseline danih gena:

```
---neka gen_1,...,gen_n vec postoje
gen=gen_1+...+gen_n;
---alternativni nacin pisanja
gen=[gen_1,...,gen_n];
```

Dvomjesni operator ”%” računa postotak podudaranja dva gena:

```
---neka gen_1 i gen_2 vec postoje
postotak = gen_1%gen_2;
```

```
---alternativni nacin pisanja
postotak=similarity(gen1, gen2);
```

Unarni operator ”∼” mutira dani gen, tako da nasumično promjeni jednu aminokiselinu u neku drugu, moguće u istu.

```
---neka gen1 vec postoji
---mutira gen1 i sprema novi gen u gen1
~gen1;
---alternativni nacin pisanja
gen1=mutation(gen1);
---mutira gen1 i sprema rezultat u gen2, gen1 se ne mijenja
gen2=mutation(gen1);
```

Poglavlje 2

Varijante gena

Varijante gena su promjene u sekvenci gena koje se mogu javiti kao prirodne varijacije ili mutacije. Rezultiraju različitim oblicima gena koji se razlikuju od referentnog ili "normalnog" oblika. Postoji nekoliko vrsta varijanti gena, uključujući jednonukleotidne polimorfizme, insercije, delecije, dupliciranja i translokacije. Varijante se gena mogu javiti u kodirajućim regijama gena, što može rezultirati promjenama u aminokiselinskom slijedu proteina, ili u nekodirajućim regijama, što može utjecati na regulaciju gena. Varijante gena mogu biti prisutne u populaciji ili mogu biti specifične za određenu jedinku. Ključne su u istraživanju naslijeđivanja bolesti i razumijevanju njihovih uzroka.

Podaci o varijantama gena prikupljaju se kroz genomske studije, kao što su genomsko sekvenciranje ili genotipizacija. Ove studije omogućuju identifikaciju varijanti gena u velikim populacijama i usporedbu genoma bolesnih i zdravih jedinki. Podaci o varijantama gena dostupni su u javnim bazama podataka, a mi smo podatke preuzeli iz gnomAD baze podataka za gen BRCA1. Svaki je čovjek rođen sa genima BRCA1 (BReast CAncer 1) i BRCA2 (BReast CAncer 2) koji, dok nisu mutirani, ne stvaraju zdravstvene probleme. Međutim, kod žena koje naslijede mutaciju jednog ili oba BRCA gena postoji veći rizik raka dojke. Kod njih se u pravilu javlja u ranijoj životnoj dobi od žena koje te promjene u genima nemaju. Također postoji veća šansa za nastanak raka jajnika. Rak dojke je, na sreću, izlječiv.

Genetičko testiranje može biti važna stepenica u borbi protiv te opake bolesti.

```
csvFile = "gnomAD.csv"; ---CSV datoteka s varijantama gena BRCA1
varijante = variant(csvFile);

imeBolesti = "Benign";
postotakBolesti = diseasePercentage.varijante(imeBolesti);
output(postotakBolesti); ---output: 8.377742504409171

imeUcinka = "missense_variant";
postotakUcinka = consequencePercentage.varijante(imeUcinka);
output(postotakUcinka); ---output: 42.592592592592595
```

Proučavanje varijanti gena pruža ključne uvide u genetsku raznolikost, evoluciju, nasljedne bolesti i farmakogenetiku. Razumijevanje varijacija gena omogućuje bolje razumijevanje bioloških procesa, mehanizama bolesti i individualne varijabilnosti odgovora na terapije. To je ključno za razvoj personalizirane medicine i poboljšanje zdravlja i liječenja.

Poglavlje 3

Smith-Watermanov algoritam

Aminokiseline igraju ključnu ulogu u stvaranju proteina i prenošenju genetske informacije. DNA je molekula koja nosi genetsku informaciju u organizmima. Ona je sastavljena od četiri različite nukleinske baze: adenina (A), citozina (C), timina (T) i guanina (G). Ove nukleinske baze formiraju parove: adenin se uparuje s timinom putem dva vodikova mosta, dok se citozin uparuje s guaninom putem tri vodikova mosta. Ova specifična komplementarnost parova baza omogućuje DNA da se precizno replicira i prenosi genetsku informaciju. Međutim, kako bi se ta genetska informacija prenijela u obliku proteina, potrebna je translacija. Tokom procesa translacije, sekvenca baza u mRNA prenosi se u sekvencu aminokiselina.

Smith-Watermanov algoritam je algoritam za poravnanje niza koji se koristi za pronalaženje sličnosti između sekvenci aminokiselina. Omogućuje identifikaciju lokalnih podudaranja i razlika između sekvenci, što je važno u identifikaciji funkcionalno važnih regija proteina i otkrivanju mutacija ili varijacija. Uzima u obzir skorove podudaranja i nepodudaranja između aminokiselina, kao i kazne za praznine u sekvencama, kako bi pronašao najbolje podudaranje između sekvenci. Koristili smo BLOSUM (BLOcks SUbstitution Matrix) matrice za mjeru sličnosti između aminokiselina, odnosno za dodjeljivanje bodova za poklapanje ili nepoklapanje aminokiselina u lokalnom poravnanju što omogućuje precizno poravnanje proteina uzimajući u obzir evolucijske informacije. Matrica se može prilagoditi različitim sku-

povima proteina, ovisno o njihovoj evolucijskoj udaljenosti i sličnosti.

Kroz primjenu Smith-Watermanovog algoritma, može se odrediti optimalno poravnanje sekvenci aminokiselina, što pomaže u razumijevanju evolucijskih veza između proteina, identifikaciji funkcionalno važnih domena i otkrivanju mutacija ili varijacija koje mogu biti povezane s bolestima ili drugim fenotipima. Ovaj algoritam ima ključnu ulogu u genetici jer omogućuju proučavanje sličnosti, evolucije i funkcionalnih karakteristika proteina na temelju sekvenci aminokiselina. Koristeći naš programski jezik, implementirajmo Smith-Watermanov algoritam na neka dva gena:

```
    ---popis dozvoljenih kiselina
        file = open("acids.txt", "r");
        ak1 = READOPTION(file);
        ak = split(ak1);
        close(file);
        ---tablica vrijednosti za poravnanja
        file = open("blosum50.txt", "r");
        bm=collection();
        for(i=0; i<20; i++)
        {
            redak = READOPTION(file, line);
            vc = split(redak);
            append.bm(vc);
        }
        x1 = gen("P E W");
        x = split(x1);
        y1 = gen("G A L A P A W A L A");
        y = split(y1);
        m = 3;
        n = 10;
        sm = collection();
        maks = 0;
        ii = 0;
        jj = 0;

        m1 = m+1;
        n1 = n+1;
        for(i=0; i<n1; i++)
        {
            tmp = collection();
```

```
31              for(j=0; j<m1; j++)
32              {
33                  append.tmp(0);
34              }
35              append.sm(tmp);
36          }
37
38          ---score matrica
39          for(i=1; i<n1; i++)
40          {
41              for(j=1; j<m1; j++)
42              {
43                  tmp1 = collection();
44                  i1=i-1;
45                  j1=j-1;
46                  vrijednost = sm[i1][j];
47                  vrijednost2 = vrijednost-8;
48                  append.tmp1(vrijednost2);
49                  vrijednost = sm[i][j1];
50                  vrijednost2 = vrijednost-8;
51                  append.tmp1(vrijednost2);
52                  kiselina1 = y[i1];
53                  kiselina2 = x[j1];
54                  for(k=0; k<20; k++)
55                  {
56                      if(ak[k]==kiselina1)
57                      {
58                          ind1 = k;
59                      }
60                      if(ak[k]==kiselina2)
61                      {
62                          ind2 = k;
63                      }
64                  }
65                  bb=bm[ind1][ind2];
66                  vrijednost3 = sm[i1][j1]+bb;
67                  append.tmp1(vrijednost3);
68                  append.tmp1(0);
69                  sort.tmp1();
70                  sm[i][j] = tmp1[3];
71                  if(sm[i][j]>maks)
```

```
72                  {
73                      maks = sm[i][j];
74                      ii=i;
75                      jj=j;
76                  }
77              }
78          }
79          str1 = collection();
80          str2 = collection();
81
82          output("Score matrica izgleda ovako:");
83          for(i=0; i<n1; i++){
84              output(sm[i]);
85          }
86
87          ---traceback, vracamo se od kraja i provjeravamo
88          ---vrijednosti kako bi nasli optimalno poravnanje
89          i=ii;
90          j=jj;
91          for(l=0; l<10;l++)
92          {
93              l = l-1;
94              if(i<=0)
95              {
96                  break;
97              }
98              if(j<=0)
99              {
100                 break;
101             }
102             if(sm[i][j]==0)
103             {
104                 break;
105             }
106             n1=sm[i][j]+8;
107             vrijednost1=i-1;
108             if(sm[vrijednost1][j]==n1)
109             {
110                 if(i>0)
111                 {
112                     vrijednost2=j-1;
```

```
113                     append.str1(y[vrijednost2]);
114                     vrijednost3 = 0-100;
115                     append.str2(vrijednost3);
116                     i=i-1;
117                 }
118             }
119             vrijednost1=j-1;
120             if(sm[i][vrijednost1]==n1)
121             {
122                 if(j>0)
123                 {
124                     vrijednost2=j-1;
125                     append.str1(x[vrijednost2]);
126                     vrijednost3 = 0-100;
127                     append.str2(vrijednost3);
128                     i=i-1;
129                 }
130             }
131             vrijednost1=j-1;
132             if(sm[i][vrijednost1]!=n1)
133             {
134                 vrijednost1=i-1;
135                 vrijednost1 = i-1;
136                 vrijednost2 = y[vrijednost1];
137                 append.str2(vrijednost2);
138                 vrijednost1 = j-1;
139                 vrijednost2 = x[vrijednost1];
140                 append.str1(vrijednost2);
141                 i=i-1;
142                 j=j-1;
143             }
144             if(i<=0)
145             {
146                 vrijednost1=i-1;
147                 vrijednost1 = i-1;
148                 vrijednost2 = y[vrijednost1];
149                 append.str2(vrijednost2);
150                 vrijednost1 = j-1;
151                 vrijednost2 = x[vrijednost1];
152                 append.str1(vrijednost2);
153                 i=i-1;
```

```
154                 j=j-1;
155             }
156             if(sm[vrijednost1][j]!=n1)
157             {
158                 vrijednost1=i-1;
159                 vrijednost1 = i-1;
160                 vrijednost2 = y[vrijednost1];
161                 append.str2(vrijednost2);
162                 vrijednost1 = j-1;
163                 vrijednost2 = x[vrijednost1];
164                 append.str1(vrijednost2);
165                 i=i-1;
166                 j=j-1;
167             }
168             if(j<=0)
169             {
170                 vrijednost1=i-1;
171                 vrijednost1 = i-1;
172                 vrijednost2 = y[vrijednost1];
173                 append.str2(vrijednost2);
174                 vrijednost1 = j-1;
175                 vrijednost2 = x[vrijednost1];
176                 append.str1(vrijednost2);
177                 i=i-1;
178                 j=j-1;
179             }
180         }
181         reverse.str1();
182         reverse.str2();
183         duljina1 = size.str1();
184         duljina = duljina1 - m;
185         for(i=0; i<duljina; i++)
186         {
187             removeIndex.str1(0);
188             removeIndex.str2(0);
189         }
190         output("Optimalno poravnanje");
191         output(str1); output(str2);
```

Izvršavajući gornji kod, dobivamo optimalno poravnanje ”PEW” i ”PAW”.

Poglavlje 4

DODATAK

U svrhu usporedbe GENLEN s Pythonom prikazan je kod koji je bio rješenje jednog zadatka s kolegija Bioinformatika koji daje isti rezultat kao i prethodno naveden kod u GENLEN za Smith-Watermanov algoritam:

```
f1=open("acids.txt", "r") ## reading acids
ak=f1.readline()
f1.close()

bm=[] ## reading matrix
f1=open("blosum50.txt", "r")
for i in range(20):
    line=f1.readline()
    vc=line.split()
    bm.append(vc[:])
for i in range(20): ## integer matrix
    for j in range(20):
        bm[i][j]=int(bm[i][j])

x='PEW'
y='GALAPAWALA'
m=len(x)
n=len(y)

sm=[] ## score matrix
tmp=[]
maks=0
ii=0
jj=0
```

```
for i in range(m+1): #popunimo score matricu nulama, ne trebaju nam po etni uvjeti
    tmp.append(0)
for i in range(n+1):
    sm.append(tmp[:])

## score matrica
for i in range(1,n+1):
    for j in range(1,m+1):
        tmp=[]
        tmp.append(sm[i-1][j]-8)
        tmp.append(sm[i][j-1]-8)
        bb=bm[ak.index(y[i-1])][ak.index(x[j-1])]
        tmp.append(sm[i-1][j-1]+bb)
        tmp.append(0)
        tmp.sort()
        sm[i][j]=tmp[3]
        if sm[i][j]>maks:
            maks=sm[i][j]
            ii=i
            jj=j
str1=[]
str2=[]

##kretanje od maksimalne vrijednosti
i=ii ##kretanje po y
j=jj ##kretanje po x

##traceback
while i>0 and j>0:
        if sm[i][j]==0: break
        n1=sm[i][j]+8
        if(i==0 and y==0): break
        if(i>0 and sm[i-1][j]==n1):
            print(1)
            str2.append(y[i-1])
            str1.append('_')
            i=i-1
        elif (j>0 and sm[i][j-1]==n1):
            str1.append(x[j-1])
            str2.append('_')
```

```
65              j=j-1
66          else:
67              str2.append(y[i-1])
68              str1.append(x[j-1])
69              i=i-1
70              j=j-1
71
72  str1.reverse()
73  str2.reverse()
74
75  duljina=len (str1)
76
77  for w in range (duljina):
78      print (str1[w], end=' ')
79  print ()
80  for q in range (duljina):
81      print (str2[q], end=' ')
```

Bibliografija

[1] https://vepar.readthedocs.io/hr/latest/

[2] https://poliklinika-harni.hr/novosti/hrvatska/brca1-i-brca-2-geni

[3] https://en.wikipedia.org/wiki/Smith%E2%80%93Waterman_algorithm

[4] https://web.math.pmf.unizg.hr/nastava/bioinformatika/?Bioinformatika_2023

[5] https://medlineplus.gov/genetics/understanding/basics/gene/

[6] https://medlineplus.gov/genetics/understanding/mutationsanddisorders/genemutation/

[7] https://openai.com/blog/chatgpt