REŠITVE NALOG ZA TRETJO SKUPINO

1. PINi

Pri tej nalogi so PINi zaporedja n števk, torej jih lahko predstavimo s celimi števili od 0 in $10^n - 1$. Naj bo f funkcija, ki nam za dani PIN izračuna novega po postopku iz naloge; na primer, pri $n = 4$ je $f(1979) = 9796$. V spodnji rešitvi to funkcijo računa podprogram Nasl. Do posameznih števk PINa a lahko pridemo tako, da upoštevamo, da je $a \bmod 10$ zadnja števka števila a, število $a \operatorname{div} 10$ pa je število a brez zadnje števke. V zanki lahko izluščimo posamezne števke a-ja in jih seštevamo. Na koncu moramo vzeti a brez prve števke (to je $a \operatorname{div} 10^{n-1}$) in mu na desni pritakniti novo števko (torej ga pomnožimo z 10 in mu novo števko prištejemo).

Če zdaj začnemo z nekim začetnim PINom a in opazujemo, kam nas pripelje funkcija f, dobimo po vrsti $a, f(a), f(f(a)), f(f(f(a))), \ldots$ Označimo $a_0 = a$ in $a_{k+1} = f(a_k)$. Ker obstaja le 10^n različnih PINov, se začnejo PINi v tem zaporedju prej ali slej ponavljati. Naj bo j najmanjši tak indeks, pri katerem je $a_j = a_i$ za nek $i < j$. Ali je mogoče, da je $i > 0$? To bi pomenilo, da je isti PIN ($a_i = a_j$) nastal tako iz a_{i-1} kot iz a_{j-1}; ta dva PINa pa sta različna. Toda ker sta a_i in a_j enaka, se ujemata v prvih $n - 1$ števkah, torej se morata a_{i-1} in a_{j-1} ujemati v zadnjih $n - 1$ števkah; če pa bi se pri tem razlikovala v prvi števki, bi imela različno vsoto števk, vendar bi se vsoti razlikovali za največ 9; torej bi bil ostanek po deljenju teh vsot z 10 različen, torej bi se morala a_i in a_j razlikovati v zadnji števki. Ker pa se ne, smo prišli v protislovje. Možnost $i > 0$ torej odpade, kar pomeni, da bomo prvo ponavljanje v našem zaporedju opazili takrat, ko bo nek a_j enak a_0, torej se bo ponovil kar naš začetni PIN a. Takrat torej vemo, da se tako a kot vsi drugi PINi v tem zaporedju začnejo ponavljati po j dnevih.

Ker je možnih PINov obvladljivo mnogo (10^n, pri čemer je $n \leq 6$, torej največ milijon PINov), si lahko v neki tabeli (v spodnjem programu se imenuje ZeVidel) za vsak PIN zapomnimo, ali smo ga v našem zaporedju že videli ali ne. Sproti štejemo korake in ko se nam ponovi začetni PIN, nam število korakov pove ravno odgovor za podnalogo (a). Enak postopek lahko potem ponovimo še za druge začetne PINe, dokler ne pregledamo vseh možnih PINov. Ko pri nekem začetnem PINu u opazimo, da se nam je le-ta ponovil po recimo d korakih, lahko povečamo števec ciklov (to bo odgovor na podnalogo (c)), obenem pa tudi vemo, da za vseh d PINov na trenutnem ciklu velja, da se začnejo ponavljati po d korakih. Tako lahko tudi sproti štejemo, pri koliko PINih pride do ponavljanja prej kot pri začetnem PINu z iz vhodne datoteke (to pa bo na koncu odgovor za podnalogo (b)).

```
program PINi;
const MaxN = 6; MaxM = 1000000;
var n, b, m, mm: integer;

  function Nasl(Pin: integer): integer;
  var i, Vsota, Novi: integer;
  begin
    Novi := (Pin mod mm) * b; Vsota := 0;
    for i := 1 to n do begin
      Vsota := Vsota + Pin mod b;
      Pin := Pin div b;
    end; {for i}
```

```
    Nasl := Novi + Vsota mod b;
  end; {Nasl}

var z, u, v, StCiklov, d, DolzinaZ, StKrajsih: integer; F: text;
    ZeVidel: packed array [0..MaxM − 1] of boolean;
begin {PINi}
  { Preberimo vhodne podatke. }
  Assign(F, 'pini.in');
  Reset(F); ReadLn(F, n); b := 10;
  m := 1; for u := 1 to n do m := m * b;    { m = 10^n }
  mm := m div b;                            { mm = 10^(n−1) }
  ReadLn(F, z); Close(F);

  for u := 0 to m − 1 do ZeVidel[u] := false;
  { Poglejmo, po koliko korakih se začne ponavljati PIN z. }
  v := z; d := 0;
  while not ZeVidel[v] do begin ZeVidel[v] := true; d := d + 1; v := Nasl(v) end;
  StCiklov := 1; DolzinaZ := d; StKrajsih := 0;
  { Preglejmo zdaj še ostale PINe. }
  for u := 0 to m − 1 do if not ZeVidel[u] then begin
    { Računajmo naslednike u-ja, dokler ne pridemo spet do u. }
    d := 0; v := u;
    while not ZeVidel[v] do begin ZeVidel[v] := true; d := d + 1; v := Nasl(v) end;
    StCiklov := StCiklov + 1;
    { Na tem ciklu je d PINov, ki se začnejo ponavljati po d korakih.
      Je to manj kot pri PINu z, ki se začne ponavljati po DolzinaZ korakih? }
    if d < DolzinaZ then StKrajsih := StKrajsih + d;
  end; {for u}

  Assign(F, 'pini.out'); Rewrite(F);
  WriteLn(F, DolzinaZ); WriteLn(F, StKrajsih); WriteLn(F, StCiklov);
  Close(F);
end. {PINi}
```

Še rešitev v C-ju:

```
#include <stdio.h>
#include <stdbool.h>

#define MaxN 6
#define MaxM 1000000

int n, b, m, mm;
bool zeVidel[MaxM];

int Nasl(int pin)
{
  int i, vsota = 0, novi = (pin % mm) * b;
  for (i = 0; i < n; i++) { vsota += pin % b; pin /= b; }
  return novi + vsota % b;
}

int main()
{
  int z, u, v, d, dolzinaZ, stKrajsih, stCiklov;

  /* Preberimo vhodne podatke. */
  FILE *f = fopen("pini.in", "rt");
  fscanf(f, "%d", &n); b = 10;
```

```
  m = 1; for (u = 0; u < n; u++) m *= b; /* m = 10^n */
  mm = m / b;                            /* mm = 10^n - 1 */
  fscanf(f, "%d", &z); fclose(f);

  for (u = 0; u < m; u++) zeVidel[u] = false;
  /* Poglejmo, po koliko korakih se začne ponavljati PIN z. */
  for (v = z, d = 0; ! zeVidel[v]; v = Nasl(v), d++) zeVidel[v] = true;
  dolzinaZ = d; stCiklov = 1; stKrajsih = 0;
  /* Preglejmo zdaj še ostale PINe. */
  for (u = 0; u < m; u++) if (! zeVidel[u])
  {
    /* Računajmo naslednike u-ja, dokler ne pridemo spet do u. */
    for (v = u, d = 0; ! zeVidel[v]; v = Nasl(v), d++) zeVidel[v] = true;
    /* Na tem ciklu je d PINov, ki se začnejo ponavljati po d korakih.
       Je to manj kot pri PINu z, ki se začne ponavljati po DolzinaZ korakih? */
    stCiklov++;
    if (d < dolzinaZ) stKrajsih += d;
  }

  /* Izpišimo rezultate. */
  f = fopen("pini.out", "wt");
  fprintf(f, "%d\n%d\n%d\n", dolzinaZ, stKrajsih, stCiklov);
  fclose(f); return 0;
}
```

Boljša rešitev s pomočjo teorije števil. Besedilo naše naloge zagotavlja, da bo pri vseh testnih primerih veljalo $n \leq 6$ in za take majhne n je zgoraj opisana rešitev povsem primerna. V nadaljevanju pa si oglejmo še, kako lahko rešitev s pomočjo nekaj ugotovitev iz teorije števil močno izboljšamo, tako da bo zmogla obdelati tudi malo večje n.

V zaporedju PINov $z, f(z), f(f(z)), \ldots$ je vsak naslednji PIN dobljen tako, da prejšnjemu pobrišemo eno števko na začetku in dodamo eno števko na koncu. Namesto zaporedja PINov lahko torej opazujemo kar zaporedje števk: na primer, pri $z = 1979$ imamo PINe $\underline{1979}, 979\underline{6}, 796\underline{1}, 961\underline{3}, \ldots$, kar lahko krajše predstavimo tako, da zapišemo le podčrtane števke: $1979613992\ldots$. Označimo števke v tem zaporedju z $a_1, a_2, \ldots$; zaradi pravila, s katerim računamo nove PINe, vidimo, da je vsaka števka (razen prvih n števk, ki smo jih dobili iz začetnega PINa) pridobljena iz prejšnjih n števk po formuli $a_k = (a_{k-1} + a_{k-2} + \ldots + a_{k-n}) \bmod 10$. Zaradi te formule je celotno zaporedje a_k-jev enolično določeno že s tem, ko smo si izbrali $a_1, \ldots, a_n$, torej števke začetnega PINa. To, da se začnejo PINi prej ali slej ponavljati, pa se na zaporedju a odrazi v tem, da se prej ali slej enkrat pojavi skupina n zaporednih števk, ki so enake prvim n števkam: naj bo torej t najmanjši tak indeks ($t > 0$), za katerega je $a_{t+1} = a_1, \ldots, a_{t+n} = a_n$. To je znak, da bi po t korakih iz začetnega PINa z spet prišli nazaj do istega PINa in odtlej se ponavljata tako zaporedje PINov kot zaporedje števk.

Definirajmo zdaj še dve zaporedji b in c s členi $b_k := a_k \bmod 2$ in $c_k := a_k \bmod 5$. Potem velja $b_k = ((a_{k-1} + \ldots + a_{k-n}) \bmod 10) \bmod 2 = (a_{k-1} + \ldots a_{k-n}) \bmod 2 = ((a_{k-1} \bmod 2) + \ldots + (a_{k-n} \bmod 2)) \bmod 2 = (b_{k-1} + \ldots b_{k-n}) \bmod 2$. Podobno bi se lahko prepričali tudi, da je $c_k = (c_{k-1} + \ldots + c_{k-n}) \bmod 5$. Zaporedji števk b_k in c_k lahko torej računamo po enakem postopku kot a_k, le da gledamo ostanke po deljenju z 2 oz. 5 namesto z 10. Za njiju velja podobno kot za zaporedje a: možnih

skupin n zaporednih števk je le končno mnogo (2^n ali 5^n), zato se v zaporedjih b in c prej ali slej pojavi ista n-terica števk kot na začetku in odtlej se zaporedji periodično ponavljata. Recimo, da se v b začetna n-terica prvič ponovi po u korakih, v c pa po v korakih, torej da je $b_{u+1} = b_1, \ldots, b_{u+n} = b_n$ in $c_{v+1} = c_1, \ldots, c_{v+n} = c_n$.

Ni se težko prepričati, da je vsaka od števk $0, \ldots, 9$ enolično določena s svojima ostankoma po deljenju z 2 in 5. Obstaja celo eksplicitna formula, s katero iz obeh ostankov izračunamo nazaj prvotno števko: $d = [5 \cdot (d \bmod 2) + 6 \cdot (d \bmod 5)] \bmod 10$. To je poseben primer ugotovitve, ki je v matematiki znana kot „kitajski izrek o ostankih“.[6] Če ta razmislek uporabimo pri naših zaporedjih, vidimo, da pri vsakem i velja $a_i = (5b_i + 6c_i) \bmod 10$.

Naj bo τ najmanjši skupni večkratnik števil u in v. Ker se b ponavlja s periodo u, je $b_{\tau+i} = b_i$; podobno se c ponavlja s periodo v, zato je $c_{\tau+i} = c_i$. Potemtakem pa je $a_{\tau+i} = (5b_{\tau+i} + 6c_{\tau+i}) \bmod 10 = (5b_i + 6c_i) \bmod 10 = a_i$. Torej so števke $a_{\tau+1}, \ldots, a_{\tau+k}$ enake prvim k števkam $a_1, \ldots, a_k$. Ker smo prej rekli, da se a ponavlja s periodo t, sledi, da je τ večkratnik t-ja.

Po drugi strani, ker se a ponavlja s periodo t, je $a_{t+1} = a_1, \ldots, a_{t+k} = a_k$. Torej je tudi $b_{t+1} = b_1, \ldots, b_{t+n} = b_n$; če pišemo $t = r \cdot u + o$ za nek $0 \le o < u$, je $b_{t+j} = b_{r \cdot u + o + j} = b_{o+j}$, torej iz $b_{t+1} = b_1, \ldots, b_{t+n} = b_n$ sledi $b_{o+1} = b_1, \ldots, b_{o+n} = b_n$. Torej je nemogoče, da bi bil $o > 0$, ker bi to pomenilo, da se začetna n-terica števk zaporedja b ponovi že po o korakih, ne pa šele po u korakih (mi pa smo za u vzeli najmanjši indeks, pri katerem se začetna n-terica ponovi). Torej je $o = 0$, kar pomeni, da je t večkratnik števila u. Podoben razmislek bi lahko opravili pri zaporedju c in videli, da je t tudi večkratnik števila v. Torej je t skupni večkratnik števil u in v, kar pomeni, da je t tudi večkratnik τ-ja.

Tako torej vidimo, da t in τ delita drug drugega, kar pomeni, da sta enaka. Perioda zaporedja a je torej ravno najmanjši skupni večkratnik period zaporedij b in c.

S temi ugotovitvami lahko pridemo do cenejšega postopka za reševanje naše naloge. Oglejmo si za začetek podvprašanje (a). Naša prvotna rešitev si je v tabeli velikosti 10^n označevala, kateri PINi (torej: katere skupine n zaporednih števk v zaporedju a) so se že pojavili, in računala nove in nove člene zaporedja, dokler ni prišla nazaj do prvotnega PINa. Iz števk $a_1, \ldots, a_n$ (ki smo jih dobili iz začetnega PINa) lahko izračunamo $b_1, \ldots, b_n$ ter $c_1, \ldots, c_n$ in nato po enakem postopku kot prej za a poiščemo periodo zaporedij b in c, najmanjši skupni večkratnik teh dveh period pa je perioda zaporedja a — to pa je ravno tisto, po čemer sprašuje podnaloga (a). Pri delu z zaporedjem b lahko uporabimo tabelo 2^n elementov, pri zaporedju c pa tabelo 5^n elementov, torej prihranimo ogromno pomnilnika v primerjavi s prvotno rešitvijo.

Do odgovorov za podnalogi (b) in (c) ni težko priti, če uspemo prešteti, koliko zaporedij s kakšno periodo obstaja. To lahko spet naredimo posebej za dvojiška zaporedja in posebej za petiška zaporedja po enakem postopku, kot ga je naša prvotna rešitev uporabila za desetiška zaporedja. Pri tem spet porabimo $O(2^n + 5^n)$ pomnil-

[6] V splošnem ta izrek pravi, da če imamo neka paroma tuja si števila $m_1, \ldots, m_r$, je potem vsako celo število od 0 do $M - 1$ (za $M = \prod_{i=1}^{r} m_i$) enolično določeno s svojimi ostanki po deljenju z $m_1, \ldots, m_r$. Z drugimi besedami, vsakemu naboru ostankov $o_1, \ldots, o_r$ (pri čemer je $0 \le o_i < m_i$ za vse $i = 1, \ldots, r$) pripada natanko tako eno število $x \in \{0, \ldots, M-1\}$, za katero je $o_i = x \bmod m_i$ za vse i od 1 do r. Pri naši nalogi delamo v desetiškem sestavu, torej nas zanima $M = 10$ in smo zato vzeli $m_1 = 2$ in $m_2 = 5$.

nika in $O(n \cdot (2^n + 5^n))$ časa. Pri $n = 4$ na primer za dvojiška zaporedja ugotovimo, da imamo en cikel dolžine 1 in tri cikle dolžine 5 (torej 15 zaporedij s periodo 5); za petiška zaporedja pa ugotovimo, da imamo en cikel dolžine 1 in dva cikla dolžine 312 (torej 624 zaporedij s periodo 312). Potem lahko razmišljamo takole. Vsako dvojiško zaporedje b s periodo u je enolično določeno s svojimi prvimi n števkami $b_1, \ldots, b_n$; podobno je tudi vsako petiško zaporedje c s periodo v enolično določeno s svojimi prvimi n števkami $c_1, \ldots, c_n$. Iz njiju lahko zdaj po zgoraj omenjeni formuli $a_i = (5b_i + 6c_i) \bmod 10$) izračunamo desetiško zaporedje, čigar perioda je (kot smo videli zgoraj) ravno najmanjši skupni večkratnik števil u in v. Če bi namesto b-ja ali c-ja vzeli kakšni drugi dve zaporedji, recimo b' in c', in iz njiju spet izračunali desetiško zaporedje a' po formuli $a'_i = (5b'_i + 6c'_i) \bmod 10$), bi bilo to zaporedje zagotovo drugačno od a: formula, s katero računamo a_i iz b_i in c_i (ali pa a'_i iz b'_i in c'_i) nam namreč zagotavlja, da je $b_i = a_i \bmod 2$ in $c_i = a_i \bmod 5$ (in podobno $b'_i = a'_i \bmod 2$ in $c'_i = a'_i \bmod 5$); če bi torej bilo $a_i = a'_i$, bi sledilo $b_i = (a_i \bmod 2) = (a'_i \bmod 2) = b'_i$ in podobno $c_i = c'_i$, torej bi prišli v protislovje s predpostavko, da smo za b' in c' vzeli drugi dve zaporedji kot za b in c.

Iz tega torej vidimo, da če imamo npr. r_u različnih dvojiških zaporedij s periodo u in s_v različnih petiških zaporedij s periodo v, potem imamo tudi $r_u \cdot s_v$ različnih desetiških zaporedij s periodo $\operatorname{lcm}\{u, v\}$ (torej najmanjši skupni večkratnik u in v). Pri $n = 4$ imamo npr. eno dvojiško zaporedje s periodo 1 in petnajst dvojiških zaporedij s periodo 5; pri petiških zaporedjih pa eno s periodo 1 ter 624 zaporedij s periodo 312. Iz tega lahko sklepamo, da imamo pri desetiških zaporedjih (za $n = 4$) eno zaporedje s periodo 1; petnajst s periodo 5; 624 s periodo 312; in 9360 ($= 15 \cdot 624$) zaporedij s periodo $5 \cdot 312 = 1560$. (Za preizkus lahko preverimo, če smo res prešteli vseh 10^n zaporedij: $1 + 15 + 624 + 9360$ je res enako 10000.)

Z opisano izboljšavo lahko nalogo dovolj učinkovito rešimo tudi za malo večje n. Pri $n = 15$ potrebujemo na primer za petiška zaporedja tabelo 5^{15} bitov, kar je približno 3,5 GB; tabela velikosti 10^{15} bitov, kakršno bi tu zahtevala naša prvotna rešitev, pa bi bila že neobvladljivo velika. Podobno dobrodošla je tu tudi izboljšava v porabi časa.

Zapišimo to rešitev še v obliki programa — tokrat v pythonu, da ne bo predolg:

```
def Naslednji(pin, n, b):
  novi = pin % (b ** (n − 1))  # odrežemo prvo števko
  vsota = 0
  while pin > 0: vsota += pin % b; pin //= b
  return novi * b + (vsota % b)

def PreglejCikel(pin, n, b, zeVidel):
  dolzina = 0
  while not zeVidel[pin]:
    zeVidel[pin] = True
    pin = Naslednji(pin, n, b)
    dolzina += 1
  return dolzina

def PrestejCikle(zacetni, n, b):
  zeVidel = [False] * (b ** n)
  # Najprej poglejmo, na kako dolgem ciklu je začetni PIN.
  dolzina = PreglejCikel(zacetni, n, b, zeVidel)
  stPinovPoDolzini = dolzina: dolzina
```

```
    # Preglejmo še vse ostale cikle.
    for pin in xrange(b ** n):
        if zeVidel[pin]: continue
        d = PreglejCikel(pin, n, b, zeVidel)
        stPinovPoDolzini[d] = stPinovPoDolzini.get(d, 0) + d
    return dolzina, stPinovPoDolzini

# Evklidov algoritem za največji skupni delitelj.
def gcd(u, v):
    while v > 0: (u, v) = (v, u % v)
    return u
# Najmanjši skupni večkratnik.
def lcm(u, v): return (u // gcd(u, v)) * v

# Preberimo vhodne podatke.
f = file("pini.in", "rt")
n = int(f.readline())
pin = f.readline().strip()
f.close()

# Izračunajmo začetne števke pripadajočega dvojiškega in petiškega zaporedja.
pin2 = sum((int(pin[i]) % 2) * (2 ** (n - 1 - i)) for i in xrange(n))
pin5 = sum((int(pin[i]) % 5) * (5 ** (n - 1 - i)) for i in xrange(n))

# Preštejmo, koliko ciklov kakšne dolžine je pri dvojiških in petiških zaporedjih.
dolzina2, st2 = PrestejCikle(pin2, n, 2)
dolzina5, st5 = PrestejCikle(pin5, n, 5)

# Kako dolg je cikel našega začetnega (desetiškega) PINa?
dolzina = lcm(dolzina2, dolzina5)

# Izračunajmo zdaj, koliko je ciklov pri desetiških PINih
# in koliko PINov je na krajših ciklih od našega zaeetnega.
stNaKrajsih = 0; stCiklov = 0
for (d2, koliko2) in st2.iteritems():
    for (d5, koliko5) in st5.iteritems():
        d = lcm(d2, d5)
        koliko = koliko2 * koliko5
        # Našli smo še „koliko“ desetiških PINov, ki se začnejo
        # ponavljati po d korakih. Torej na vsakih d takih pinov pride en cikel.
        stCiklov += koliko // d
        if d < dolzina: stNaKrajsih += koliko
# Izpišimo rezultate.
f = file("pini.out", "wt")
f.write("%d\n%d\n%d\n" % (dolzina, stNaKrajsih, stCiklov))
f.close()
```

Kot zanimivost si oglejmo skupno število ciklov (torej odgovor pri podnalogi (c)) za prvih nekaj n-jev: za n od 1 do 12 dobimo 10, 6, 20, 12, 40, 850, 816, 206, 280, 66458, 267140 in 80580.

Rešitev z manjšo porabo pomnilnika. Videli smo, da pravkar opisana rešitev porabi pravzaprav le še 5^n bitov pomnilnika, če delamo z n-mestnimi PINi. To je že velika izboljšava v primerjavi z 10^n pri prvotni rešitvi, vendar pa tudi 5^n hitro postane preveč. Na primer, videli smo, da je pri $n = 15$ tabela 5^n bitov velika že dobre 3,5 GB, pri $n = 16$ pa že skoraj 18 GB. Prvo mogoče že še gre, 18 GB pomnilnika pa ima dandanes še ni preveč pogosto. Oglejmo si zdaj še rešitev, ki porabi zelo malo pomnilnika, žal pa je cena za to precej večja poraba časa.

V naši gornji rešitvi potrebuje veliko tabelo zeVidel predvsem podprogram PrestejCikle, da z njeno pomočjo takoj vidi, če je nek PIN že pregledal (v okviru kakšnega od doslej pregledanih ciklov) in mu ga torej ni treba pregledovati še enkrat. To tabelo uporablja sicer tudi podprogram PreglejCikel, vendar zanj ni tako kritična — to, kdaj je prišel naokrog celega cikla, lahko preverja tudi tako, da si pač zapomni PIN, pri katerem je začel, in po vsake koraku preverja, če je novi PIN enak tistemu prvotnemu. Podprogramu PrestejCikle pa lahko pomagamo takole: ker po novem ne bo več imel tabele, ki bi mu povedala, kateri PINi so bili že obiskani, bo moral začeti pregledovati cikel pri čisto vsakem PINu. Vendar pa, ker pregledujemo PINe po naraščajočem vrstnem redu, vemo, da vsak cikel obiščemo že takrat, ko je bila naša spremenljivka pin enaka najmanjšemu PINu tega cikla. Če torej pri sprehodu po nekem ciklu pridemo do kakšnega PINa, ki je manjši od tistega, pri katerem smo začeli, vemo, da smo ta cikel že videli in ga lahko takoj nehamo pregledovati.

V našem gornjem pythonovskem programu lahko zdaj pobrišemo podprogram PreglejCikel, nato pa podprogram PrestejCikle zamenjamo z naslednjim:

```
def PrestejCikle(pin, n, b):
  # Najprej poglejmo, na kako dolgem ciklu je začetni PIN.
  dolzina = 1; u = Naslednji(pin, n, b)
  while u != pin: u = Naslednji(u, n, b); dolzina += 1
  # Preglejmo zdaj sistematično vse cikle.
  stPinovPoDolzini =
  for pin in xrange(b ** n):
    # Preglejmo cikel, na katerem leži ta PIN.
    u = Naslednji(pin, n, b); d = 1
    while u > pin: u = Naslednji(u, n, b); d += 1
    if u < pin: continue # Ta cikel smo nekoč prej že pregledali.
    stPinovPoDolzini[d] = stPinovPoDolzini.get(d, 0) + d
  return dolzina, stPinovPoDolzini
```

Poraba časa bo zdaj seveda večja kot prej, ker se lahko zgodi, da mora notranja zanka **while** narediti kar nekaj iteracij, preden opazimo, da smo ta cikel nekoč že videli. Vendar pa v povprečju ponavadi ni treba prav veliko iteracij, preden to opazimo; pri $n = 6$ za petiška zaporedja ($b = 5$) na primer porabimo povprečno po 5,4 iteracije; pri $n = 10$ pa po 10,9 iteracije. Če to pomeni, da bomo porabili desetkrat več časa kot pri prejšnji rešitvi, je stvar mogoče vendarle sprejemljiva, če bomo zaradi tega na koncu vsaj prišli do rezultata, pri prejšnji rešitvi pa ne bi mogli, ker nimamo 5^n bitov pomnilnika. Omeniti velja tudi, da je sedanji postopek zelo primeren za paralelno izvajanje — vsak procesor (ali pa vsak računalnik) lahko na primer pregleda nek interval možnih vrednosti spremenljivke pin, pri tem so lahko čisto ločeni (ne potrebujejo skupnega pomnilnika in ne komunicirajo med seboj), na koncu pa le združimo (torej seštejemo) dobljene tabele stPinovPoDolzini z vseh procesorjev.

2. Berberi

Zemljevid začnimo preiskovati v tistem polju na zahodnem robu, pri katerem so Berberi sploh vstopili na zemljevid. Od tam sledimo njihovim selitvam po poljih, označenih z `X`. Kjer se njihova pot razveji, obiščimo najprej eno nadaljevanje poti in kasneje še drugo. Pri tem torej vidimo, da je koristno, če si pri vsakem polju