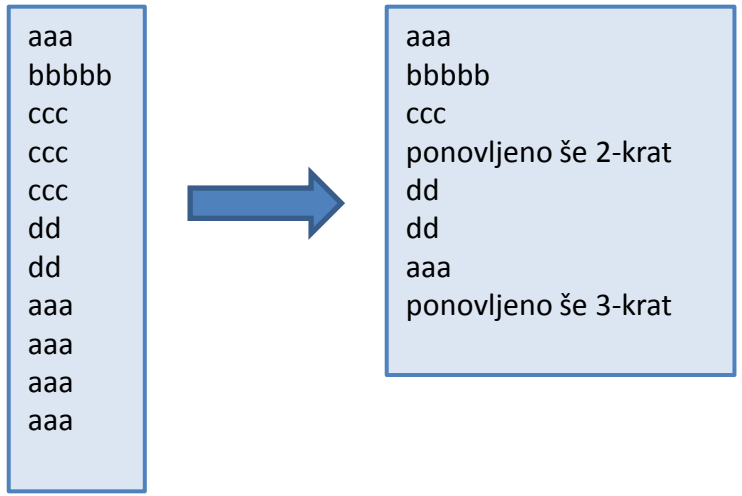
Naloge in rešitve RTK 2014

Janez Brank

1.1 Dnevnik



- Naloga:
  - Brati moramo vhod po vrsticah in jih *sproti* izpisovati
  - Kjer se pojavi ≥ 3 zaporednih enakih vrstic, izpišemo le prvo vrstico + število preostalih
- Rešitev:
  - Zapomnimo si prejšnjo vrstico in število pojavitev
  - Če je naslednja vrstica enaka prejšnji, le povečamo števec pojavitev
  - Sicer končamo izpis prejšnje, izpišemo naslednjo in postavimo števec na 1

1.1 Dnevnik

- Rešitev, malo podrobneje:

```
prejsnja = ""; n = 0;
while True:
    nova = PreberiDogodek()
    if n > 0 and nova == prejsnja:
        n = n + 1; continue // Nadaljuje se dosedanja skupina enakih vrstic.
    // Sicer pa zaključimo izpis za prejšnjo skupino enakih vrstic…
    if n == 2: print prejsnja
    elif n > 2: print "ponovljeno še (n – 1)-krat"
    // …in začnimo novo skupino.
    if nova == "": break
    print nova; prejsnja = nova; n = 1
```

1.2 Proizvodnja čopičev

- Naloga:
  - Imamo n lesenih palic z dolžinami L_1, L_2, …, L_n
  - Za izdelavo čopiča potrebujemo:
    - Ročaj dolžine r (ki mora biti v celoti dobljen iz ene same palice)
    - Za konico pa toliko lesa, kolikor se ga dobi iz palice dolžine k, vendar je lahko pridobljen iz več kosov več različnih palic
  - Koliko čopičev lahko izdelamo?
- Rešitev:
  - Iz palice L_i lahko naredimo največ $\lfloor L_i / r \rfloor$ ročajev
  - To seštejemo po vseh palicah in tako dobimo število ročajev, ki bi se jih dalo narediti; to je zgornja meja za število čopičev
  - Po drugi strani, če hočemo x čopičev, nam po tistem, ko naredimo x ročajev, ostane še $(L_1 + \ldots + L_n - x \cdot r)$ lesa za konice, potrebujemo pa ga $x \cdot k$
  - Tako imamo še omejitev $x \leq (L_1 + \ldots + L_n) / (r + k)$
  - Med obema omejitvama vzamemo nižjo

1.3 Generali

- Naloga:
  - Imamo k ključev $\{1, 2, \ldots, k\}$ za sprožitev atomske bombe
  - Imamo n generalov, vsak od njih ima nekaj ključev
    - General i ima ključe $G_i \subseteq \{1, 2, \ldots, k\}$
  - Skupina generalov lahko sproži bombo, če imajo vsi skupaj ravno vse ključe (torej če je unija njihovih G_i enaka $\{1, 2, \ldots, k\}$
  - Sistem je r-odporen, če ne obstaja nobena taka skupina r generalov, ki lahko drugim prepreči sprožitev
  - Opiši postopek, ki preveri, ali je dani sistem r-odporen
- Primer: $k = 3$, $n = 3$, $G_1 = \{1, 2\}$, $G_2 = \{2, 3\}$, $G_3 = \{1, 3\}$ je 1-odporen
- Rešitev:
  - Skupina lahko sproži bombo, če za vsak ključ (od 1 do k) velja, da ima ta ključ vsaj eden od članov skupine
  - Torej skupina lahko drugim prepreči sprožitev, če za vsaj en ključ velja, da ga nima nihče razen članov te skupine
  - Torej za vsak ključ preštejmo, koliko generalov ga ima
  - Če ima kakšen ključ r ali manj generalov, potem sistem ni r-odporen, sicer pa je

1.4 Uniforme

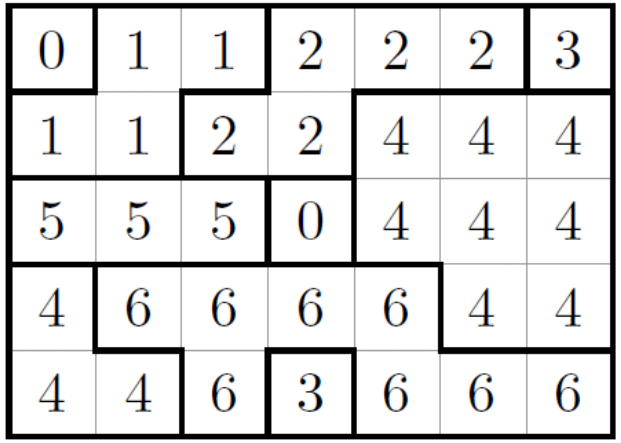
- Naloga:
  - Imamo seznam parov (*kos*, *velikost*)
    - *kos* $\in$ {1, 2, 3} (hlače, jopič, rokavice)
    - *velikost* $\in$ {1, 2, …, 100}
  - Koliko popolnih uniform lahko sestavimo (takih, pri katerih so vsi trije kosi enake velikosti)?
- Rešitev:
  - Imejmo 2-d tabelo, v kateri štejemo, koliko kosov posamezne velikosti imamo
    - Na začetku inicializiramo vse elemente na 0
    - Za vsak par (kos, velikost), ki ga preberemo, povečamo ustrezni element tabele za 1
  - Nato za vsako velikost pogledamo, katerega kosa imamo pri tej velikosti najmanj
    - Ta minimum nam tudi pove, koliko popolnih uniform te velikosti lahko sestavimo
    - To seštejemo po vseh velikostih

1.4 Uniforme

- Rešitev, malo podrobneje:

```
// Inicializacija.
for v = 1 to 100 do for k = 1 to 3 do zaloga[v][k] = 0
// Preberimo vhodne podatke.
za vsak vhodni podatek (v, k):
    zaloga[v][k] += 1
rezultat = 0
for v = 1 to 100:
    // Koliko popolnih uniform velikosti v lahko sestavimo?
    n = zaloga[v][1]
    for k = 2 to 3
        if zaloga[v][k] < n then n = zaloga[v][k]
    rezultat += n
```

1.5 Davek na ograjo



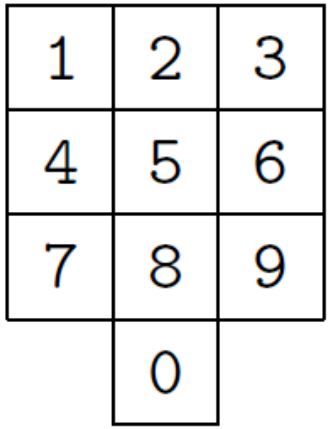
- Naloga:
  - Dana je karirasta mreža $w \times h$ kvadratkov
  - Za vsak kvadratek je znana številka lastnika (od 0 do $n - 1$)
  - Če sosednja kvadratka pripadata različnima lastnikoma, je med njima ograja
  - Ograje so tudi na zunanjem robu mreže
  - Za vsakega lastnika izpiši skupno dolžino ograj okrog njegovih kvadratkov
- Rešitev:
  - Pripravimo si tabelo n elementov, v kateri bomo za vsakega lastnika šteli ograje
  - Z gnezdenima zankama gremo po vseh kvadratkih
    - Za vsak kvadratek (x, y) pogledamo njegove štiri sosede
      - Če sosed pripada drugemu lastniku ali pa sploh leži zunaj mreže, povečajmo števec ograj lastnika kvadratka (x, y)

1.5 Davek na ograjo

- Rešitev, malo podrobneje:

```
int DX[] = { 1, –1, 0, 0 }, DY[] = { 0, 0, 1, –1 };
for a = 0 to n – 1 do dolžina[a] = 0;
for x = 0 to w – 1 do for y = 0 to h – 1 do
   for smer = 0 to 3 do // preglejmo vse štiri sosednje kvadratke
      xx = x + DX[smer]; yy = y + DY[smer];
      // če je (xx, yy) že zunaj mreže, ograja mora biti
      if xx < 0 or yy < 0 or xx >= w or yy >= h then ograja = true
      // sicer je ograja le, če imata kvadratka različna lastnika
      else ograja = (Lastnik(xx, yy) == Lastnik(x, y))
      if ograja then dolžina[Lastnik(x, y)] += 1
```

2.1 Vnos šifre



- Naloga:
  - Imamo številčno tipkovnico, kakršno kaže slika
  - Izpiši vse take *n*-mestne šifre, pri katerih velja, da je vsaka naslednja števka bodisi enaka prejšnji bodisi leži v tipkovnici na eni od sosednjih tipk
  - Primer: 414558
- Rešitev:
  - Nalogo lahko rešujemo z rekurzijo
  - Prvo števko si lahko izberemo poljubno (od 0 do 9)
  - Za vsako naslednjo števko preizkusimo vse tiste števke, ki so na tipkovnici sosednje (ali enake) prejšnji; za vsako izvedemo rekurziven klic

2.1 Vnos šifre

- Rešitev malo podrobneje:

```
IzpišiŠifre(s, n)
    naj bo d dolžina niza s
    if d == n then izpiši s; return
    naj bo c zadnja števka niza s
    for (za vsako števko c', ki je soseda števke c)
        IzpišiŠifre(s + c', n)
```

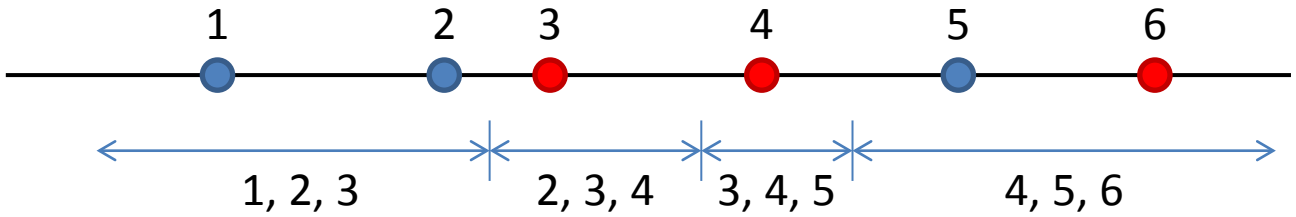
- Glavni del programa:

```
for c = '0' to '9' do IzpišiŠifre(c, n)
```

2.2 Prenova ceste

- Naloga:
  - Imamo cesto dolžine D
  - Ob njej živi n = 1 000 000 prebivalcev;
    znane so njihove koordinate $x_1 < x_2 < \ldots < x_n$
  - Cesto bosta obnavljali dve podjetji; prebivalec i podpira podjetje $p_i \in \{0, 1\}$
  - Za vsako točko t pogledamo najbližjih k prebivalcev;
    to točko bo obnovilo tisto podjetje, ki ga podpira večina od teh k prebivalcev
  - Koliko km ceste bo obnovilo prvo podjetje in koliko drugo?
- Rešitev:
  - Na prvi pogled je stvar neugodna, ker je možnih t-jev neskončno
  - Toda če sta prebivalca i in j (za $i < j$) med najbližjimi k sosedi točke t, potem so tudi $i + 1, \ldots, j - 1$ med najbližjimi k sosedi točke t
  - Najbližji sosedje točke t so torej vedno $\{z, z + 1, z + 2, \ldots, z + k - 1\}$ za
  - Vprašanje je le, kje preklopimo iz enega z na naslednjega

2.2 Prenova ceste



- Rešitev:
  - Če gledamo neko t, ki je levo od vseh prebivalcev ($t < x_1$), je najbližjih k sosedov kar prvih k prebivalcev ($x_1, x_2, \ldots, x_k$)
  - Če se zdaj počasi premikamo s t proti desni, nastopi prva sprememba takrat, ko x_1 izpade iz te soseščine, vanjo pa pride x_{k+1}
    - To se zgodi takrat, ko x_{k+1} postane točki t bližji kot x_1, torej na pol poti med njima, pri $t = (x_1 + x_{k+1}) / 2$
  - Podobno je naslednja sprememba pri $(x_2 + x_{k+2}) / 2$, ko iz soseščine izpade x_2, vanjo pa pride x_{k+2}
  - Sproti lahko tudi popravljamo spremenljivko (recimo k_1), ki nam pove, koliko izmed trenutnih k sosedov podpira podjetje št. 1
- Psevdokoda:

```
dolžina[0] = 0; dolžina[1] = 0;
k1 = 0; for i = 1 to k do k1 = k1 + pi
for z = 1 to n – k + 1:
    if z == 1 then        xOd = 0 else xOd = (x_{z–1} + x_{z–1+k}) / 2
    if z == n – k + 1 then xDo = D else xDo = (x_z + x_{z+k}) / 2
    // za vse t iz [xOd, xDo] velja soseščina z…z+k–1
    dolžina[k1 > k – k1 ? 1 : 0] += xDo – xOd;
    k1 = k1 – pz + p_{z+k}
```

2.3 Skrivno sporočilo

- Naloga:
  - Substitucijski kod: vsaka črka se spremeni v neko drugo črko (v bistvu neka permutacija abecede)
  - Koda ne poznamo, imamo pa neko sporočilo p_1 in njegovo delno šifrirano različico c_1
    - Delno šifrirana pomeni, da so v c_1 ponekod znaki *
  - Hočemo rekonstruirati kod, kolikor je le mogoče, in z njim zašifrirati neko drugo sporočilo p_2
- Primer:

  p_1 = `tralala`

  c_1 = `q*x**yx`

  - Iz tega lahko rekonstruiramo `t` → `q`, `a` → `x`, `l` → `y`, za druge črke (npr. `r`) pa šifre ne poznamo

2.3 Skrivno sporočilo

- Rešitev:
  - Imejmo tabelo *šifra*[*a..z*], v kateri za vsako črko hranimo njeno šifro (če jo že poznamo, sicer pa znak *)
  - Pregledujmo istoležne znake nizov p_1 in c_1
    - Pri vsakem *i* vemo, da je $c_1[i]$ šifra znaka $p_1[i]$
    - Razen če je $c_1[i]$ = *
    - Če za isto črko vidimo več različnih šifer, vemo, da je v vhodnih podatkih napaka
  - Nato se samo še sprehodimo po znakih niza p_2 in jih šifriramo s pomočjo tabele *šifra*
- Psevdokoda:

```
for c = 'a' to 'z' do šifra[c] = '*'
for i = 0 to len(p1) – 1:
    if c1[i] == '*':                    continue
    else if šifra[p1[i]] == '*':        šifra[p1[i]] = c1[i]
    else if šifra[p1[i]] != c1[i]:      sporoči napako
for i = 0 to len(p2) – 1:
    izpiši znak šifra[p2[i]]
```

2.4 Potenciranje

- Naloga:
  - Dan je izmišljen jezik, podoben zbirnemu jeziku
    - Z ukazi ADD, SUB, MUL, DIV, MOD (aritmetika) in JL (pogojni skok)
  - Napiši v njem program, ki izračuna potenco a^b
  - Namig: najprej izračunaj a^2, a^4, a^8, a^{16} itd.
- Rešitev:
  - Potence iz namiga lahko računamo z zaporednim kvadriranjem:
    $a^2 = a \cdot a$, $a^4 = (a^2) \cdot (a^2)$, $a^8 = (a^4) \cdot (a^4)$ itd.
  - Število b lahko izrazimo kot vsoto potenc števila 2
    - Npr. $b = 87 = 1010111_2 = 64 + 16 + 4 + 2 + 1$
    - Zato je $a^{87} = a^{64 + 16 + 4 + 2 + 1} = a^{64} \cdot a^{16} \cdot a^4 \cdot a^2 \cdot a^1$
  - Zdaj moramo ta postopek le še zapisati v našem zbirnem jeziku

2.4 Potenciranje

- Postopek:

  $c = 1$;
  for $i = 0, 1, 2, \ldots$:
  if je bit i v dvojiškem zapisu števila b prižgan **then** $c = c \cdot a$
  $a = a \cdot a$

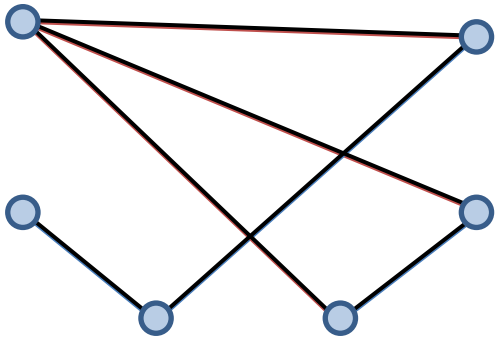
- Kako preveriti, ali je nek bit v dvojiškem zapisu b-ja prižgan?
  - Za najnižji bit je stvar preprosta: ta je prižgan $\Leftrightarrow$ b je liho število $\Leftrightarrow$ $b \bmod 2 = 1$
  - Če potem izračunamo celi del količnika $b/2$, je to ravno b brez najnižjega bita
    - Primer: $87/2 = 43{,}5$; iz $87 = 1010111_2$ smo dobili $43 = 101011_2$s
  - Torej lahko v vsaki iteraciji pogledamo najnižji bit in ga nato odrežemo
  - Ko b pade na 0, so vsi preostali biti ugasnjeni in lahko zanko končamo

2.4 Potenciranje

- Rešitev:

```
            SUB c, c        postavi c na 0
            ADD c, 1        c je zdaj 1
zanka:      JL b, 1, konec  če je b = 0, končaj
            SUB t, t        postavi t na 0
            ADD t, b        postavi t na b
            MOD t, 2        trenutni bit števila b
            JL t, 1, preskok če je ugasnjen, preskoči nasl. vrstico
            MUL c, a        pomnoži c s trenutno potenco a-ja
preskok:    MUL a, a        pripravi naslednji potenco a-ja
            DIV b, 2        odreži najnižji bit b-ja
            JL 0, 1, zanka  skoči nazaj na začetek zanke
konec:
```

2.5 Vezja



- Naloga:
  - Imamo *n* točk z znanimi koordinatami (x_i, y_i), $i = 1, \ldots, n$
  - In *m* daljic, ki povezujejo razne pare točk (krajišča so tudi znana)
  - Ali lahko to vezje narišemo tako, da se noben par daljic ne seka?
  - Pri tem smemo uporabiti obe strani plošče
    - Ali pa eno stran in dve barvi, pri čemer se daljice različnih barv lahko sekajo
- Rešitev:
  - Ko izberemo barvo neke daljice, je s tem enolično določena tudi barva vseh daljic, ki se sekajo z njo
    - Iz tega pa tudi sledi barva vseh daljic, ki se sekajo z njimi in tako naprej
  - Tem omejitvam sledimo, dokler ne pridemo v protislovje ali pa pobarvamo celoten graf

2.5 Vezja

- Rešitev:

```
za vsako daljico u: barva[u] = – 1;
za vsako daljico u:
   if barva[u] != –1 then continue      // že pobarvana
   barva[u] = 0;          // Pobarvajmo u; kaj to pomeni za barve drugih daljic?
   naj bo Q vrsta z u kot edinim elementom;
   while Q ni prazna:
      vzemi naslednjo daljico v iz vrste Q
      // Pobarvali smo v; poglejmo tiste, ki se sekajo z njo.
      za vsako daljico w, ki se seka z v:
         if barva[w] == –1 then          // w mora biti druge barve kot v
            barva[w] = 1 – barva[v]; dodaj w v vrsto Q
         else if barva[w] == barva[v] then
            sporoči napako (vezja ni mogoče narisati)
```

3.1 Ljudožerci na premici

- Naloga:
  - Na premici imamo n ljudožercev z znanimi koordinatami x_1, …, x_n
    - Niso nujno različne in niso nujno urejene
  - Po vrsti padajo na premico padalci
    - Padalec j pade na točko a_j, najbližji ljudožerec se premakne do njega in ga požre
    - Če je več najbližjih ljudožercev (enako oddaljenih od a_j), vzamemo najbolj levega med njimi;
    - Če je več takih, pa tistega z najmanjšo zaporedno številko
  - Zanima nas končni položaj vseh ljudožercev v naraščajočem vrstnem redu
- Rešitev:
  - Koristno je imeti ljudožerce urejene naraščajoče po koordinati
    - Ker vhodni podatki niso taki, jih najprej uredimo
  - Zdaj lahko najbližjega poiščemo z bisekcijo
  - Če leži a_j točno na pol poti med dvema zaporednima ljudožercema, vzamem
  - Če jih je več na isti koordinati, je v resnici vseeno, katerega vzamemo, ker jih bomo morali tako ali tako izpisati v naraščajočem vrstnem redu

3.1 Ljudožerci na premici

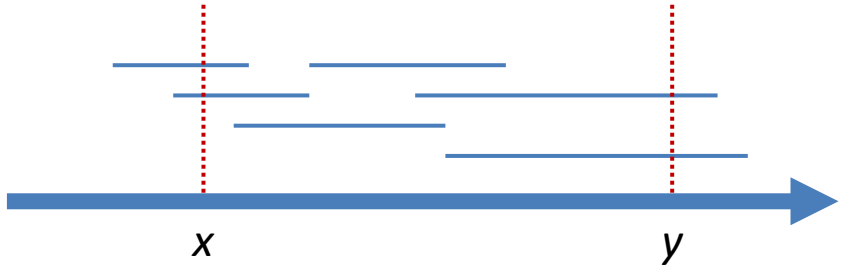
- Rešitev:

  najprej uredimo vhodne podatke naraščajoče,
  tako da je $x_1 \le x_2 \le x_3 \le \ldots \le x_n$
  za vsakega padalca a_j:
  i = *PoiščiNajbližjega*(a_j)
  $x_i = a_j$
  na koncu izpišemo zaporedje $x_1, x_2, \ldots, x_n$

- Postopek za iskanje najbližjega z bisekcijo:

```
int PoiščiNajbližjega(int a) {
  if a < x_1 then return 1 else if a ≥ x_n then return n;
  L = 1; D = n + 1;
  while D – L > 1:
    // Na tem mestu velja x_L < a ≤ x_D
    M = (L + D) / 2
    if x_M < a then L = M else D = M
  // Na tem mestu velja x_{D–1} < a ≤ x_D
  if x_D – a < a – x_{D–1} then return D else return D – 1 }
```

3.2 Po Indiji z avtobusom



- Naloga:
  - Na premici potujemo od zahoda proti vzhodu
  - Začnemo v *x* in hočemo priti v *y*
  - Obstaja *n* avtobusov, znane so začetne in končne postaje (a_i, b_i)
  - Na avtobus lahko vstopimo/izstopimo kjerkoli na [a_i, b_i]
  - Kako s čim manj avtobusi priti iz *x* v *y*?
- Rešitev:
  - Požrešni algoritem:
    - Med vsemi avtobusi, ki peljejo mimo našega trenutnega položaja, se splača stopiti na tistega, ki ima najkasnejšo končno postajo
      - Ker če stopimo na kakšnega, ki ima zgodnejšo končno postajo, s tem ničesar ne pridobimo; vse, kar bi lahko naredili na njem, bi lahko naredili tudi na tistem s kasnejšo končno postajo
    - Podobno, ko stopimo na avtobus, se splača peljati z njim vse do končne postaje (ali do *y*)

3.2 Po Indiji z avtobusom

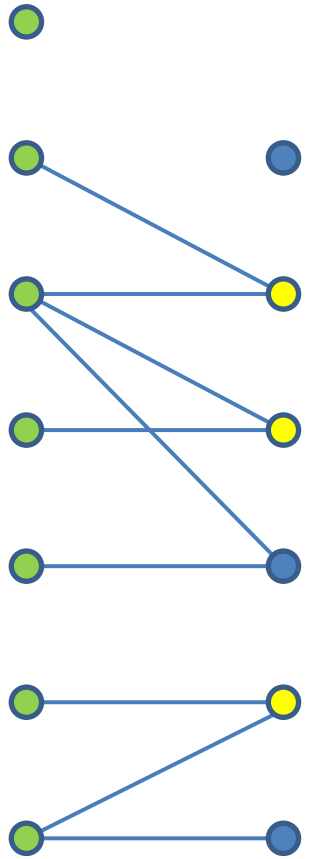
- Postopek:

  štVoženj = 0
  while $x < y$:
  med avtobusi, ki peljejo mimo točke x,
  naj bo i tisti z najbolj desno končno postajo b_i ;
  $x = b_i$; *štVoženj* = *štVoženj* + 1

- Kako učinkovito izberemo naslednji avtobus?
  - Pogoj, da pelje mimo x (torej $a_i \leq x \leq b_i$), lahko omilimo v $a_i \leq x$
    - Tisti, ki se končajo že pred x (torej $b_i < x$), bodo tako ali tako izpadli, ko bomo vzeli avtobus z najbolj desno končno postajo
  - V vsaki iteraciji zunanje zanke se x poveča in pogoju $a_i \leq x$ ustreza še malo več avtobusov kot prej
  - Zato je koristno imeti avtobuse urejene po a_i in se v vsakem koraku premakniti samo še malo naprej po tem zaporedju

    štVoženj = 0; $i = 1$; $b = x$
    while $x < y$:
    while $a_i \leq x$:
    $b = \max(b, b_i)$; $i = i + 1$
    $x = b$; *štVoženj* = *štVoženj* + 1

3.3 Luči



- Naloga:
  - Imamo *S* stikal in *L* luči; znano je začetno stanje luči
  - Vsaka luč je povezana z največ 2 stikaloma
  - Vsako stikalo lahko pritisnemo 1× ali 0×
  - S koliko kombinacijami stikal lahko ugasnemo vse luči?
- Rešitev:
  - Če je luč *x* povezana z enim samim stikalom, recimo *u*:
    - Če je *x* prižgana, potem *u* moramo pritisniti
    - Če je *x* ugasnjena, potem *u* ne smemo pritisniti
  - Če pa je *x* povezana z dvema stikaloma *u* in *v*:
    - Če je *x* prižgana, potem moramo pritisniti enega od *u* in *v*, drugega pa ne
    - Če je *x* ugasnjena, potem moramo pritisniti oba (*u* in *v*) ali nobenega
  - Torej, čim si za neko stikalo *u* izberemo, ali ga bomo pritisnili ali ne, nam to določi stanje stikal, ki so povezana na iste luči kot *u*
    - To pa nam določi stanje spet novih stikal in tako naprej po celi povezani komponenti
  - Za vsako povezano komponento dobimo 0, 1 ali 2 možnosti; to zmnožimo med sabo
  - Poseben primer: če je luč prižgana in ni povezana z nobenim stikalom

3.3 Luči

- Postopek:

```
for u = 1 to S do obdelano[u] = false, stanje[u] = –1
rezultat = 1
for u = 1 to S do if not obdelano[u]:
    // Preglejmo povezano komponento, ki ji pripada u.
    možnosti = 0
    if Poskusi(u, 0) then možnosti += 1
    if Poskusi(u, 1) then možnosti += 1
    rezultat *= možnosti; if možnosti == 0 then break
    // Če je možnosti > 0, je vsaj eden od klicev Poskusi že označil
    // vsa stikala te povezane komponente kot obdelana.
```

  - Manjka še preverjanje prižganih luči brez stikal

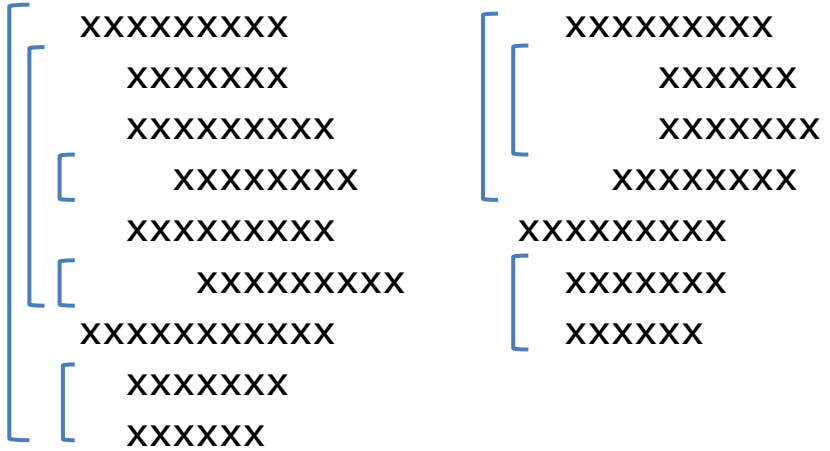
3.3 Luči

- Podprogram *Poskusi*(u_0, s) preveri, če lahko postavimo u_0 v stanje s

```
stanje[u0] = s; naj bo Q prazna vrsta
dodaj u0 v Q; obdelana[u0] = true
while Q ni prazna:
    vzemi iz Q naslednje stikalo, recimo u
    // Stanje u smo določili; kako vpliva u na druga stikala?
    za vsako luč x, ki je povezana s stikalom u:
        if je to edino stikalo te luči:
            if prižgana[x] xor stanje[u] then return false
        if ima ta luč še neko drugo stikalo v:
            if prižgana[x] xor stanje[u] xor stanje[v] then return false
            else if stanje[v] = –1 then // v-ju lahko določimo stanje
                stanje[v] = prižgana[x] xor stanje[u]
                obdelano[v] = true; dodaj v v vrsto Q
return true
```

  - Pred vrnitvijo iz podprograma (ne glede na rezultat) moramo še postaviti *stanje*[*u*] nazaj na –1 za vse *u*, ki smo jih kdaj dodali v vrsto *Q*

3.4 Bloki



- Naloga:
  - Imamo več vrstic besedila,
    v njem bi radi prepoznali bloke
  - V vrstici se začne blok, če je neprazna in
    - Ima večji zamik kot prejšnja neprazna vrstica
    - Ali pa je to prva neprazna vrstica sploh
  - Ta blok se potem nadaljuje do prve take neprazne vrstice,
    ki ima manjši zamik kot začetna vrstica bloka (tista potem že ni več del bloka)
- Rešitev:
  - Za vsako vrstico izračunajmo zamik (število presledkov na začetku);
    besedilo lahko sproti pozabljamo
  - Bloke iščemo z rekurzijo
    - Zapomnimo si zamik prejšnje neprazne vrstice
    - Če ima trenutna vrstica večji zamik, se začne nov blok
      ($\rightarrow$ rekurzivni klic; ob vrnitvi nam trenutni položaj pove, kje se ta blok konča)
    - Če ima trenutna vrstica manjši zamik kot trenutno najbolj notranji blok,
      se ta blok konča (vrnitev iz rekurzivnega klica)

3.4 Bloki

- Rešitev:
  - Naj bo *zamik*[*i*] zamik *i*-te vrstice (–1, če je prazna)
  - Naj bo *blokDo*[*i*] zadnja vrstica bloka, ki se začne v vrstici *i* (–1, če se tam ne začne blok)
  - Glavni blok programa:
    izračunaj zamike vrstic;
    postavi vse *blokDo*[*i*] na –1;
    prejZamik = –1; *i* = 0; *PoiščiBloke*(–1)
  - Rekurzivni podprogram *PoiščiBloke*(*zamikBloka*):

```
// i = trenutna vrstica (globalna spremenljivka)
while i < n:
    if zamik[i] < 0 then continue;                    // preskoči prazne vrstice
    prejPrejZamik = prejZamik; prejZamik = zamik[i]
    if zamik[i] < zamikBloka then break;              // konec bloka
    if zamik[i] <= prejPrejZamik then i +=1; continue // blok se nadaljuje
    // Sicer se v i začne nov blok.
    blokOd = i ; i += 1; PoiščiBloke(prejZamik)
    blokDo[blokOd] = i – 1
```

3.5 Poravnavanje desnega roba

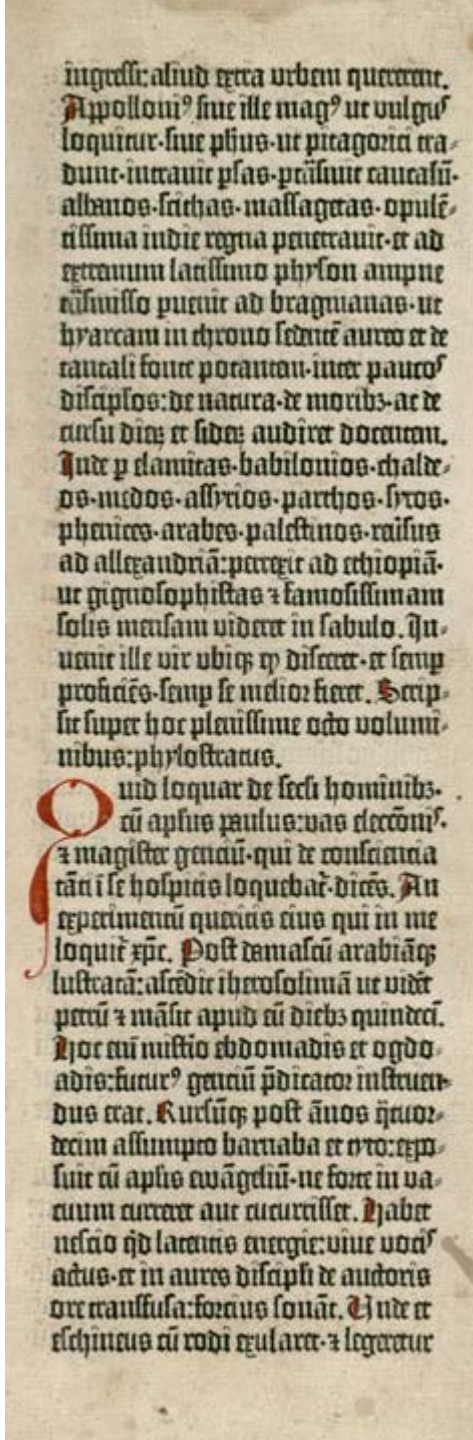
- Naloga:
  - Imamo besedilo, ki ga tvori n besed z dolžinami $w_1, w_2, \ldots, w_n$
  - Radi bi ga razbili v vrstice dolžine d
  - Presledek med dvema zaporednima besedama v vrstici mora biti vsaj s, če pa je treba, bomo presledke razširili, da bo vrstica dosegla širino d
  - Besede od w_i do w_j lahko zložimo v eno vrstico, če je
    $w_i + w_{i+1} + \ldots + w_{j-1} + w_j + (j - i) \cdot s \leq d$
  - Ocena take vrstice je
    $ocena(i, j) = [d - (w_i + w_{i+1} + \ldots + w_{j-1} + w_j + (j - i) \cdot s)]^2$
  - Zadnje vrstice pa ne poravnavamo: $ocena(i, n) = 0$
  - Razbij besedilo na vrstice tako, da bo skupna ocena vrstic najmanjša možna
- Primer: d = 30, s = 1; n = 6 besed z dolžinami 10, 10, 3, 9, 9, 30.

1. vrstica: 10 + s + 10 + s + 3 = 25, ostane 5 → ocena = 25
2. vrstica: 9 + s + 9 = 19, ostane 11 → ocena = 121
3. vrstica: 30, ostane 0 → ocena 0

Skupaj 25 + 121 + 0 = 146

1. vrstica: 10 + s + 10 = 21, ostane 9 → ocena = 81
2. vrstica: 3 + s + 9 + s + 9 = 23, ostane 7 → ocena = 49
3. vrstica: 30, ostane 0 → ocena 0

Skupaj 81 + 49 + 0 = 130



3.5 Poravnavanje desnega roba

- Rešitev:
  - Recimo, da bi radi razbili besede w_i, w_{i+1}, …, w_n
    - Oceno najboljšega takega razbitja označimo s $f(i)$
    - Prvih nekaj besed damo v prvo vrstico, recimo do w_j
    - Ostane nam problem, kako čim bolje razbiti podzaporedje w_{j+1}, w_{j+2}, …, w_n
    - Imamo torej rekurzivno zvezo
      $$f(i) = \min \{ ocena(i, j) + f(j + 1) : i \le j \le n \}$$
    - V resnici sme iti j le tako daleč, dokler še lahko spravimo vse besede w_i, w_{i+1}, …, w_{j-1}, w_j v eno vrstico
  - Da ne bomo računali istih $f(i)$ po večkrat, si rezultate zapomnimo, čim jih izračunamo
    - Zato lahko računamo kar sistematično po padajočih i:

```
f[n + 1] = 0
for i = n downto 1 do
    j = i; širina = 0; f[i] = ∞      // izračunajmo najboljše razbitje besed i..n
    while j ≤ n:
        širina += w[j]; if j > i then širina += s   // širina vrstice po dodajanju besede w_j
        if širina > d then break                    // je vrstica že preširoka?
        f[i] = min{ f[i], (d – širina)^2 + f[j + 1] } // cena razbitja, če začnemo z vrstico i..j
```