RTK 2018
Naloge in rešitve

Janez Brank

1.1 Collatz++

15, 46, 23, 70, 35, 106, 53, 160, 80, 40, 20, 10, 5, 16, 8, 4, 2, 1

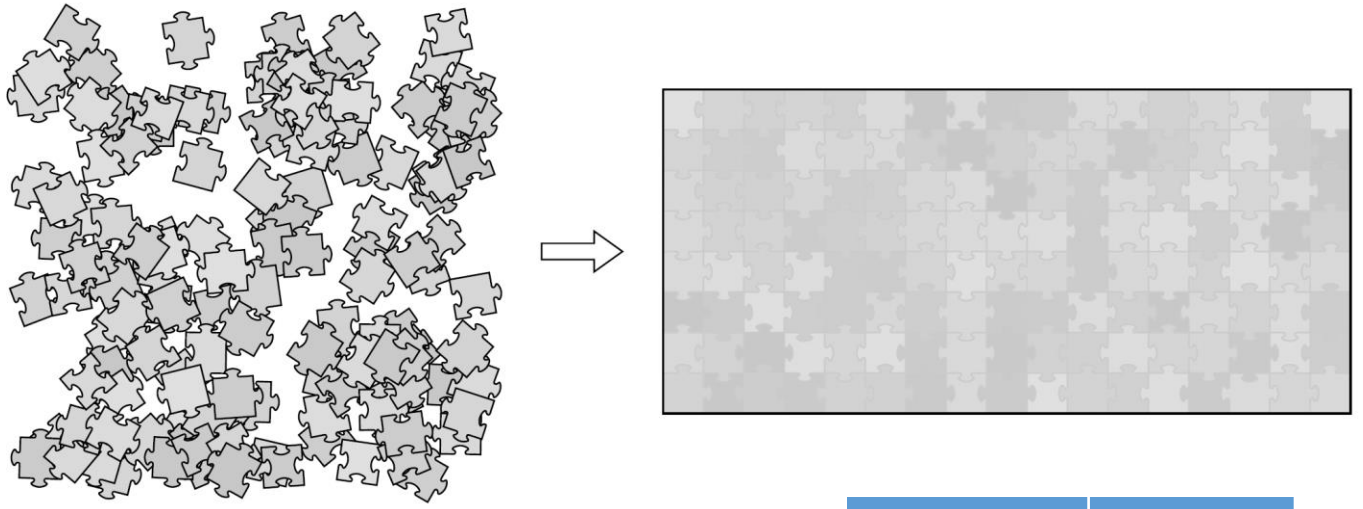
- Collatzevo zaporedje:
  - Začnemo z nekim naravnim številom *k*
  - Vsak člen izračunamo iz prejšnjega:
    - Če je *n* sod, mu sledi *n/2*
    - Če je *n* lih, mu sledi 3*n* + 1
  - Ustavimo se, ko pridemo do 1
- Naloga: pri katerih *k* z območja *a*..*b* doseže zaporedje največjo vrednost?
- Rešitev:
  - V zunanji zanki gremo po *k*-jih od *a* do *b*
  - V notranji zanki računamo člene zaporedja od k naprej, dokler ne pridemo do 1
    - Pri tem si zapomnimo največji člen zaporedja
  - Vzdržujemo seznam rešitev
    - Če je največji člen pri *k* enak največjemu členu nasploh, dodamo *k* med rešitve
    - Če je večji, dosedanje rešitve zavržemo in si zapomnimo *k*

1.2 Alfa Bravo

Črka	Beseda	Črka	Beseda	Črka	Beseda
A	ALFA	J	JULIET	S	SIERRA
B	BRAVO	K	KILO	T	TANGO
C	CHARLIE	L	LIMA	V	VICTOR
D	DELTA	M	MIKE	W	WHISKY
E	ECHO	N	NOVEMBER	X	X-RAY
F	FOXTROT	O	OSCAR	Y	YANKEE
G	GOLF	P	PAPA	Z	ZULU
H	HOTEL	Q	QUEBEC		
I	INDIA	R	ROMEO		

- Dekodiraj besedilo, kjer je vsaka črka zakodirana v besedo
  - V besedi je lahko največ 1 črka napačna, npr. `LINA` namesto `LIMA`
- Rešitev:
  - Vhodno besedilo razbijmo na besede
  - Za vsako vhodno besedo s pojdimo v zanki po črkah
    - Primerjajmo niz s *s* kodo črke *c*
    - Če sta različno dolga, se ne ujemata
    - Sicer primerjajmo v zanki istoležne znake nizov
      - Štejmo neujemanja, če opazimo drugo neujemanje, črka *c* ni prava
    - Če pridemo uspešno do konca, je *c* prava črka in jo izpišimo

1.3 Sestavljanka



- Imamo sestavljanko iz *n* koščkov, ki tvorijo pravokotnik $w \times h$
  - *w* in *h* ne poznamo, pač pa le približno razmerje *h*/*w*
  - Poišči taka *w* in *h*, da bo h/*w* čim bližje x
- Rešitev:
  - Gremo v zanki po vseh *h* (od 1 do *n*)
    - Preverimo, če *h* deli *n*
    - Če da, izračunamo w = *n*/*h* in nato razliko $|h/w - x|$
  - Zapomnimo si najmanjšo razliko doslej (in pri katerem *h* je bila dosežena)
  - To na koncu izpišemo

x = 0,4

$h \times w$	h/w
1×136	0.007
2×68	0.029
4×34	0.118
8×17	0.470
17×8	2.125
34×4	8.5
68×2	34
136×1	136

1.3 Sestavljanka

- Možna izboljšava:
  - Delitelji vedno nastopajo v parih: d in n/d
    - Eden je $\leq \sqrt{n}$, eden je $\geq \sqrt{n}$
  - Lahko gremo s h le do $\sqrt{n}$ namesto do n
  - Ko vidimo, da je h delitelj n-ja, poleg pravokotnika $h \times w$ ocenimo še $w \times h$
- Še ena izboljšava:
  - Ker je $h \times w = n$, lahko razmerje h / w zapišemo kot h / (n/h) = h^2 / n
  - Najbližje x bi torej prišli pri h^2 / $n = x$, torej $h = \sqrt{(x \cdot n)}$
  - Pregledujmo cela števila h v okolici $\sqrt{(x \cdot n)}$ (navzgor in navzdol), dokler ne naletimo na kakšen tak h, ki je delitelj n-ja

1.4 Pisalni stroj

- Natipkali bi radi vrstico, dolgo *n* znakov
  - Na mestu a_i bi radi natipkali *i* znakov
  - Po vsakem natipkanem znaku (tudi presledku) se stroj premakne za 1 mesto v desno
  - Obstaja tudi tipka za vrnitev na začetek vrstice
  - Drugih možnosti za premik v levo ni
  - Koliko pritiskov na tipke potrebujemo?

äbç de

äbç de

i	1	2	3	4	5	6
a_i	2	1	3	0	2	1

1.4 Pisalni stroj

- Rešitev:
  - Poiščimo najbolj desni i, pri katerem je $a_i > 0$
  - Enkrat bo torej treba iti do tja, da tam kaj natipkamo
    - Spotoma lahko natipkamo en znak še na mestih $1..(i - 1)$ (če je treba)
    - Ni razloga, da bi šli kaj dlje kot do i
  - Če ni noben $a_i > 1$, smo končali
  - Sicer poiščimo najbolj desni i, pri katerem je $a_i > 1$
    - Pojdimo na začetek vrstice in od tam tipkajmo do i
    - Spotoma na vsakem mestu natipkajmo še drugi znak (če je treba)
  - Itd.
  - Naj bo b_k = najbolj desni i, pri katerem je $a_i > k$
  - Potem potrebujemo $b_1 + 1 + b_2 + 1 + \ldots + b_{k-1} + 1 + b_k$ pritiskov na tipke
  - Kako to učinkovito računati?

1.4 Pisalni stroj

- Koristno je iti po zaporedju *a* od desne proti levi
  - Ko prvič naletimo na nek a_i , ki je $\geq k$, je trenutni *i* ravno b_k
  - Tako imamo naslednji postopek:

K = 0; *vsota* = 0;
for *i* = *n*, *n* – 1, …, 2, 1:
    if $a_i > K$:
        vsota = *vsota* + (a_i – *K*) * *i* ;
        K = a_i
if *K* > 0: *vsota* = *vsota* + *K* – 1;

1.5 Brzinomer

- Naloga: krmiliti moramo kazalec brzinomera
  - Sistem nas vsake toliko časa pokliče in nam sporoči trenutno hitrost (0..300)
  - Mi povemo, kako naj se kazalec premakne (+1, –1 ali 0)
  - Kazalec gre lahko od 0 do 250, sicer se premik ignorira
  - Začetni položaj kazalca je neznan
    - Zato naj se prvih 250 korakov premika le navzdol, po tistem vemo, da je na 0
- Rešitev:
  - V globalni spremenljivki hranimo položaj kazalca
  - Ko nas sistem pokliče, pogledamo, ali je nova hitrost večja, manjša ali enaka trenutnemu položaju
    - V resnici namesto hitrosti gledamo min(*hitrost*, 250)
  - Še ena spremenljivka pove, ali smo še v začetni fazi, ko kazalec ves čas premikamo navzdol
    - Oz. pove, koliko korakov bomo še v tej fazi

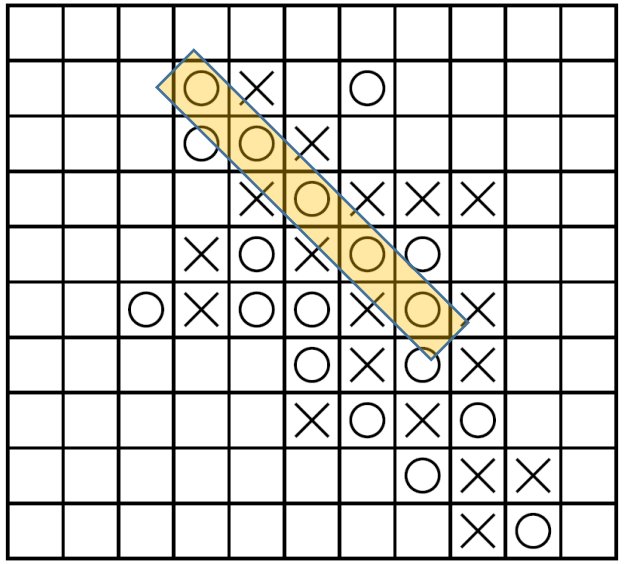
1.5 Brzinomer

- Rešitev:

```
int kazalec = 0, zacetek = 250;

int Premik(int hitrost)
{
   if (zacetek > 0) { zacetek –= 1; return –1; }
   if (hitrost > 250) hitrost = 250;
   int premik = 0;
   if (hitrost > kazalec) premik = 1;
   else if (hitrost < kazalec) premik = –1;
   kazalec += premik; return premik;
}
```

2.1 Križci in krožci



- Naloga:
  - Dva igralca izmenično rišeta križce in krožce na karirasto mrežo
  - Stanje mreže dobimo kot 2-d tabelo znakov `'x'`, `'o'` in `' '` (presledek)
  - Ugotovi, kdo (če sploh kdo) ima 5 enakih znakov skupaj (v isti vrstici, stolpcu ali diagonali)
- Rešitev:
  - Za vsak možni začetni položaj (*x*, *y*) in za vsako smer (*dx*, *dy*):
    - Naj bo *c* znak na položaju (*x*, *y*) – če je presledek, ga preskočimo
    - V notranji zanki se premikajmo od (*x*, *y*) v tej smeri, dokler še vidimo enake znake
    - Čim vidimo drugačen znak ali pademo čez rob mreže, se ustavimo
    - Če naštejemo 5 znakov, je *c* zmagovalec

2.1 Križci in krožci

- Rešitev:

```
for (x = 0; x < w; x++) for (y = 0; y < h; y++) { // za vsak začetni položaj
    c = a[y][x];  if (c == '   ') continue;
    for (s = 0; s < 4; s++) { // za vsako možno smer
        xx = x; yy = y; d = 1;
        while (d < 5) { // ali je v to smer 5 enakih znakov?
            xx += DX[s]; yy += DY[s];
            if (xx < 0 || yy < 0 || xx >= w || yy >= h) break; // čez rob
            if (a[yy][xx] != c) break; // napačen znak
            d += 1; }
        if (d == 5) return c; /* našli smo zmagovalca */  }}
return '   '; // ni zmagovalca
```

2.2 Popravljanje testov

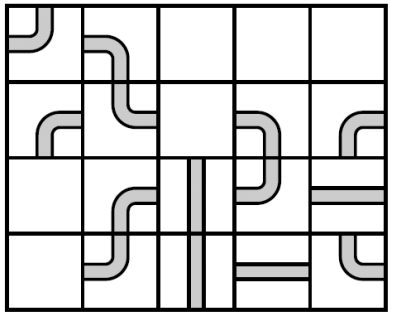
- Operacije na skladu:
  - Učenci (predstavljeni vsak s po eno črko) so odlagali teste na vrh sklada, profesor jih je pobiral z vrha sklada
  - Te operacije se med seboj prepletajo
  - Dano je zaporedje znakov, ki povedo:
    - `'?'` = nekdo je odložil test na sklad (pa ne vemo, kdo)
    - `'A'..'Z'` = profesor je pobral test s to črko z vrha sklada
  - Naloga: za vsak `'?'` ugotovi, kdo je takrat odložil test na vrh sklada
    - Ali pa povej, da je to nemogoče ugotoviti

Vhod:	`?A`
Izhod:	`AX`
Vhod:	`???`
Izhod:	*neveljaven vhod*
Vhod:	`???C?DB?RA?H?Q`
Izhod:	`ABCXDXXRXXHXQX`
Vhod:	`??QWE?`
Izhod:	*neveljaven vhod*
Vhod:	`??????DC??GBAA??WQBA`
Izhod:	`ABAACDXXBGXXXXQWXXXX`


2.2 Popravljanje testov

- Rešitev:
  - Najlažje je iti od desne proti levi in sproti vzdrževati stanje sklada
  - Ko naletimo na črko, jo dodamo na sklad
  - Ko naletimo na `'?'`, vemo, da je ta test oddal tisti učenec, ki je trenutno na vrhu sklada
    - Takrat ga tudi pobrišemo od tam
  - Če smo pri `'?'` in je sklad prazen, to pomeni, da ne moremo ugotoviti, kdo je ta test oddal
    - Ker ga profesor nikoli ni pobral s sklada
  - Če sklad ni prazen, ko pridemo na levi konec niza, je vhod tudi neveljaven
    - Profesor je pobiral teste, ki jih ni nihče oddal (???)

2.3 Cevi

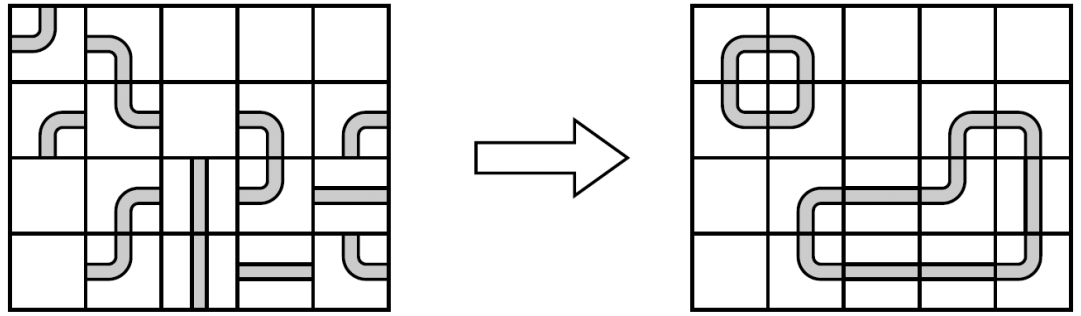


- Naloga:
  - Karirasta mreža, v vsaki celici je cev tipa I, tipa L ali pa je prazna
  - Cevi lahko vrtimo po 90° levo ali desno
  - Koliko potez potrebujemo, da bodo vse cevi sklenjene?
    - Ali pa ugotovi, da je to nemogoče

2.3 Cevi

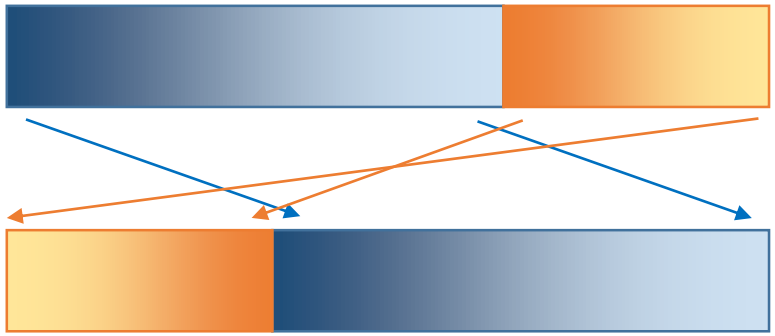
Tip cevi v trenutni celici	Naj se cev navezuje na zgornji oz. levi rob? na nobenega	le zgornjega	le levega	na oba
prazna	□	×	×	×
tip I	×			×
tip L				

- Rešitev:
  - Če za neko celico vemo, ali mora iti cev do njene zgornje in leve stranice ali ne, je položaj cevi v njej že enolično določen
  - Pojdimo torej sistematično po vrsticah od zgoraj navzdol, v vsaki vrstici od leve proti desni
    - Ko pridemo do celice (x, y), za njeno zgornjo in levo sosedo že poznamo položaj cevi
      - Če tiste sosede ni, je to enako, kot da bi bila prazna
    - Torej vemo, ali mora iti cev do zgornje in leve stranice trenutne celice ali ne
    - Torej lahko določimo položaj cevi v trenutni celici



2.4 Palačinke

- Naloga:
  - Na skladu so sprva po vrsti števila od 1 (na dnu) do *n* (na vrhu)
  - Osnovna operacija: zgornjih *k* elementov poberemo s sklada in jih v nasprotnem vrstnem redu vrinemo na dno
  - Naloga: po več takih operacijah (za različne *k*) izpiši novo stanje sklada

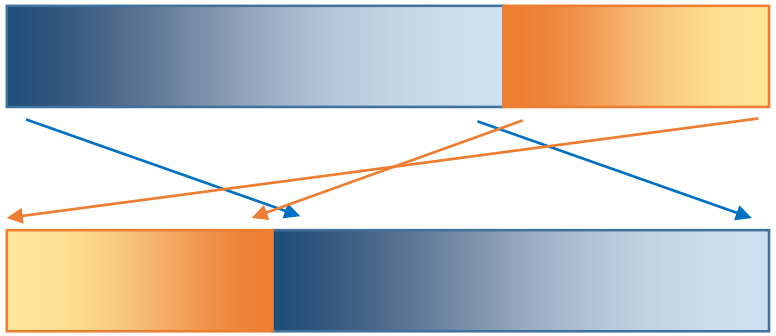


2.4 Palačinke

- Naivna rešitev:
  - Sklad hranimo v tabeli [0..n – 1]
  - Po vsakem obračanju v celoti na novo izračunamo stanje sklada

```
for (i = 0; i < k; i++) nova[i] = stara[n – 1 – i];
for (i = 0; i < n – k; i++) nova[i + k] = stara[i];
```

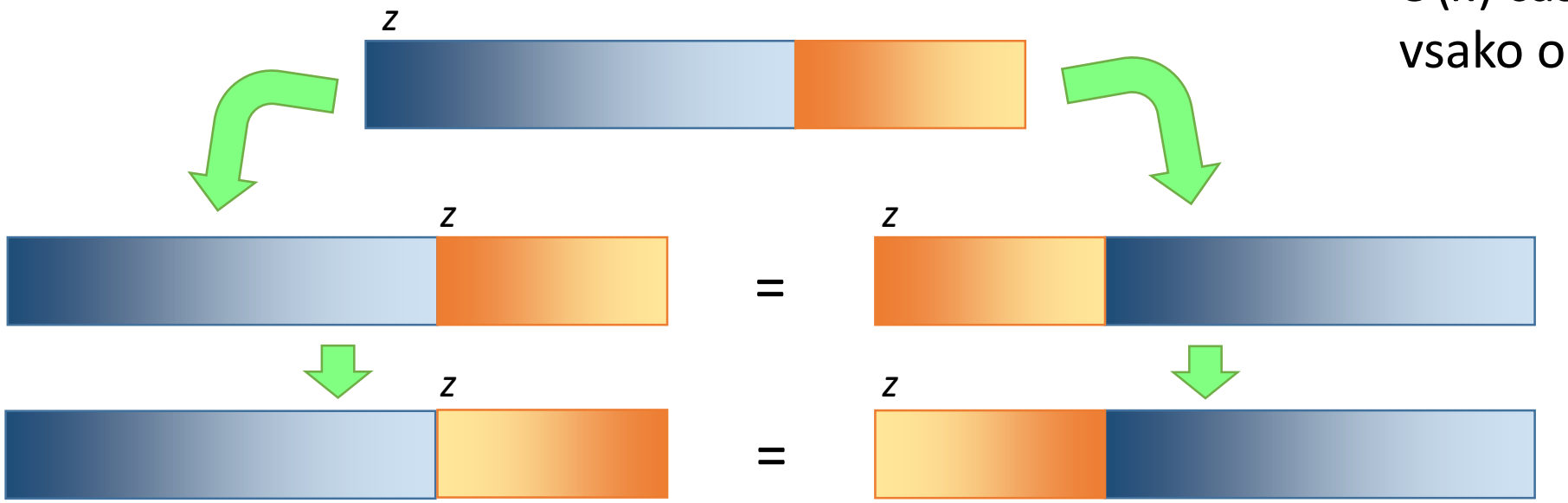
$O(n)$ časa za vsako obračanje



2.4 Palačinke

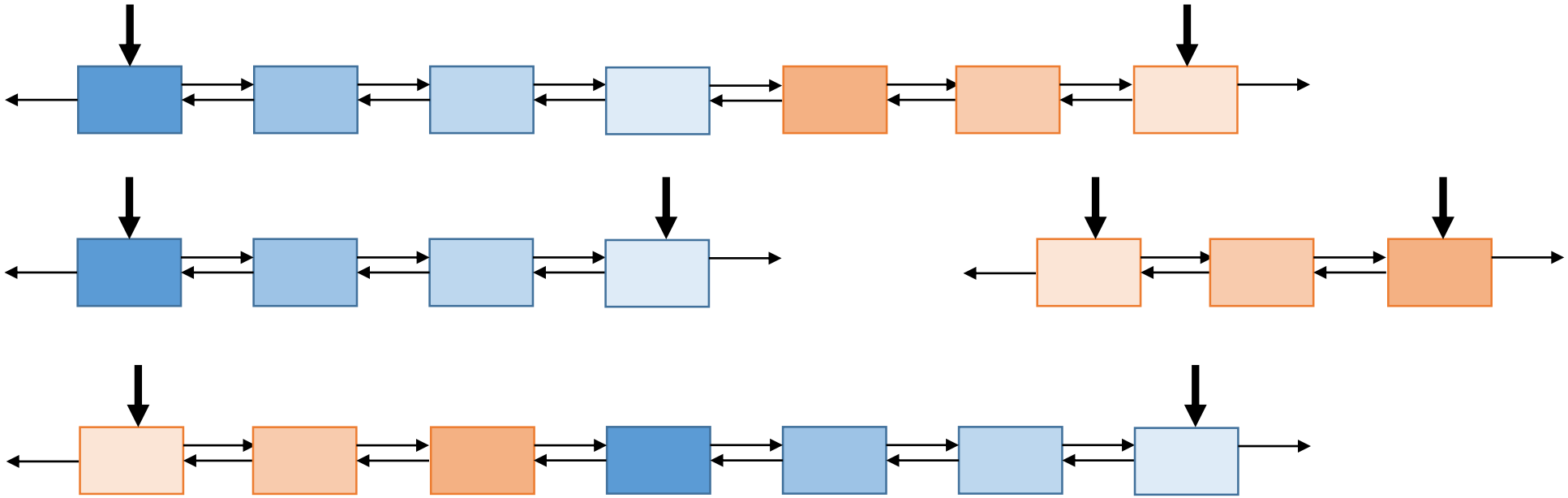
- Boljša rešitev:
  - Tabelo si predstavljamo kot ciklično, začne se na indeksu z
    - Taka tabela predstavlja zaporedje $T[z], T[z+1], \ldots, T[n-1], T[0], T[1], \ldots, T[z-1]$
  - Če z povečamo na $(z + n - k) \% n$, je to enako, kot da bi celo tabelo ciklično zamaknili za $n - k$ mest
  - Nato le še obrnemo vrstni red spodnjih k elementov

$O(k)$ časa za vsako obračanje

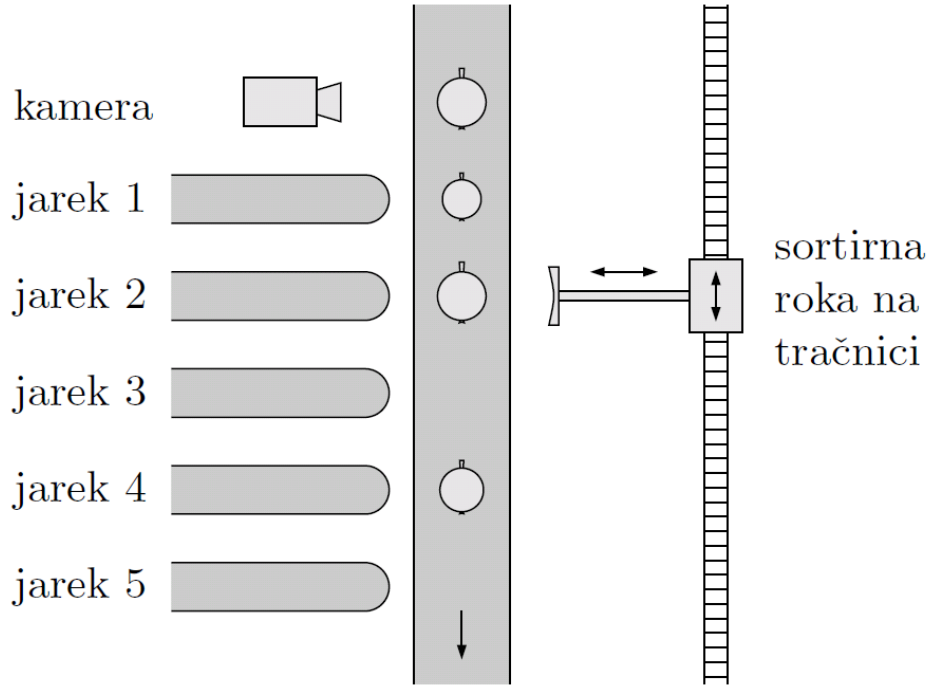


2.4 Palačinke

- Še ena možnost: dvojno povezan seznam (*doubly-linked list*)



2.5 Jabolka



- Naloga:
  - Po tekočem traku prihajajo jabolka
  - Ob traku je 5 jarkov in kamera, ki vsako jabolko oceni od 1 do 5
  - Imamo sortirno roko, ki se lahko postavi pred jarek in odrine jabolko vanj
  - Napiši program, ki skrbi za to, da vsako jabolko pride v pravi jarek
    - Na voljo so podprogrami:

```
int PremakniTrak();
void PremakniRoko(int k);
void SproziRoko();
```

2.5 Jabolka

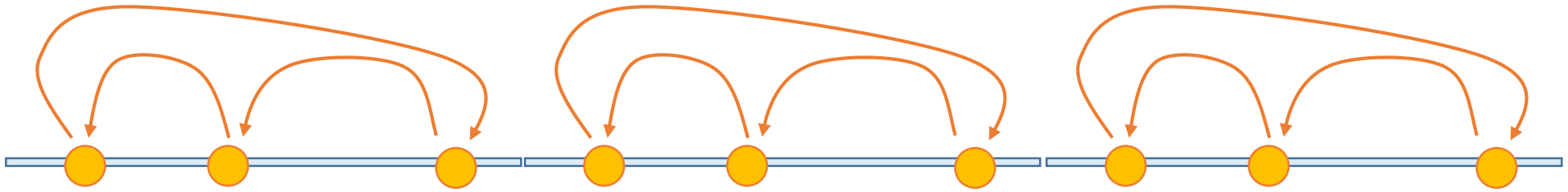
- Rešitev:
  - V tabeli *T*[0..5] vzdržujemo ocene jabolk pred vsakim jarkom in pred kamero
    - *T*[*k*] = 0 pomeni, da tam jabolka ni
  - Program teče v neskončni zanki
    - Zamaknemo *T*[0..4] v [1..5], da bomo pripravljeni na premik
    - Premaknemo trak, novo jabolko vpišemo v *T*[0]
    - Gremo po *T*[1..5], če je *T*[*k*] = *k*, premaknemo roko pred jarek *k* in jo sprožimo

```
int T[6] = { 0, 0, 0, 0, 0, 0 };

while (true) {
   for (k = 5; k >= 1; k– –) T[k] = T[k – 1];
   T[0] = PremakniTrak();
   for (k = 1; k <= 5; k ++) if (T[k] == k) { PremakniRoko(k); SproziRoko(); } }
```

3.1 Buteljke

- Naloga:
  - Imamo n ljudi, znani so njihovi rojstni dnevi r_i (vsi različni)
  - Za nekatere pare (u, v) je znano, da u obdaruje v-ja za rojstni dan
    - Darilo = vedno 1 buteljka
  - Koliko buteljk morajo kupiti, da si jih bodo lahko v nedogled podajali med sabo? (Če je to sploh mogoče)

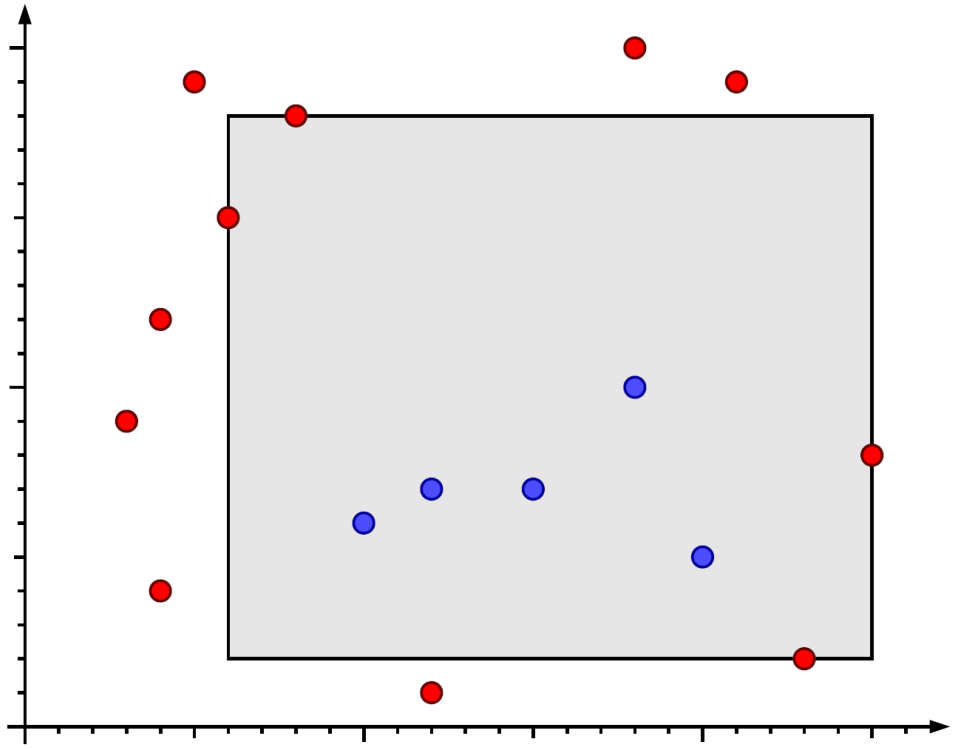


3.1 Buteljke

- Rešitev:
  - Če kdo podari v enem letu več (ali manj) buteljk,
    kot jih dobi na svoj rojstni dan, sistem dolgoročno ne more delovati
    - Ker bo moral vsako leto dokupovati (ali pa se mu bodo nabirale)
  - Sicer sistem deluje, buteljke je treba kupovati le v prvem letu
    - Kajti vsak u na svoj prvi rojstni dan dobi toliko buteljk, kot jih bo potem podaril v času do svojega naslednjega rojstnega dneva
  - Za prvo leto: preštejmo, pri koliko parih (u, v), kjer u obdaruje v-ja,
    velja $r_v < r_u$
    - Na v-jev rojstni dan nima u še nobene buteljke (ker še ni imel svojega rojstnega dne)
    - Zato jo mora kupiti, da jo lahko podari v-ju
  - Kaj če sistema ne uvedejo ravno na začetku leta?
    - Naj bo u tisti, ki ima prvi rojstni dan; premaknimo uvedbo sistema tik za r_u
    - Prej u ni kupil nobene buteljke, zdaj jih kupi toliko, kolikor jih mora podariti
    - Prej so vsi, ki obdarujejo u-ja, kupili buteljko zanj, zdaj je nihče
    - To dvoje se ravno odšteje $\rightarrow$ skupno število kupljenih buteljk ostane enako

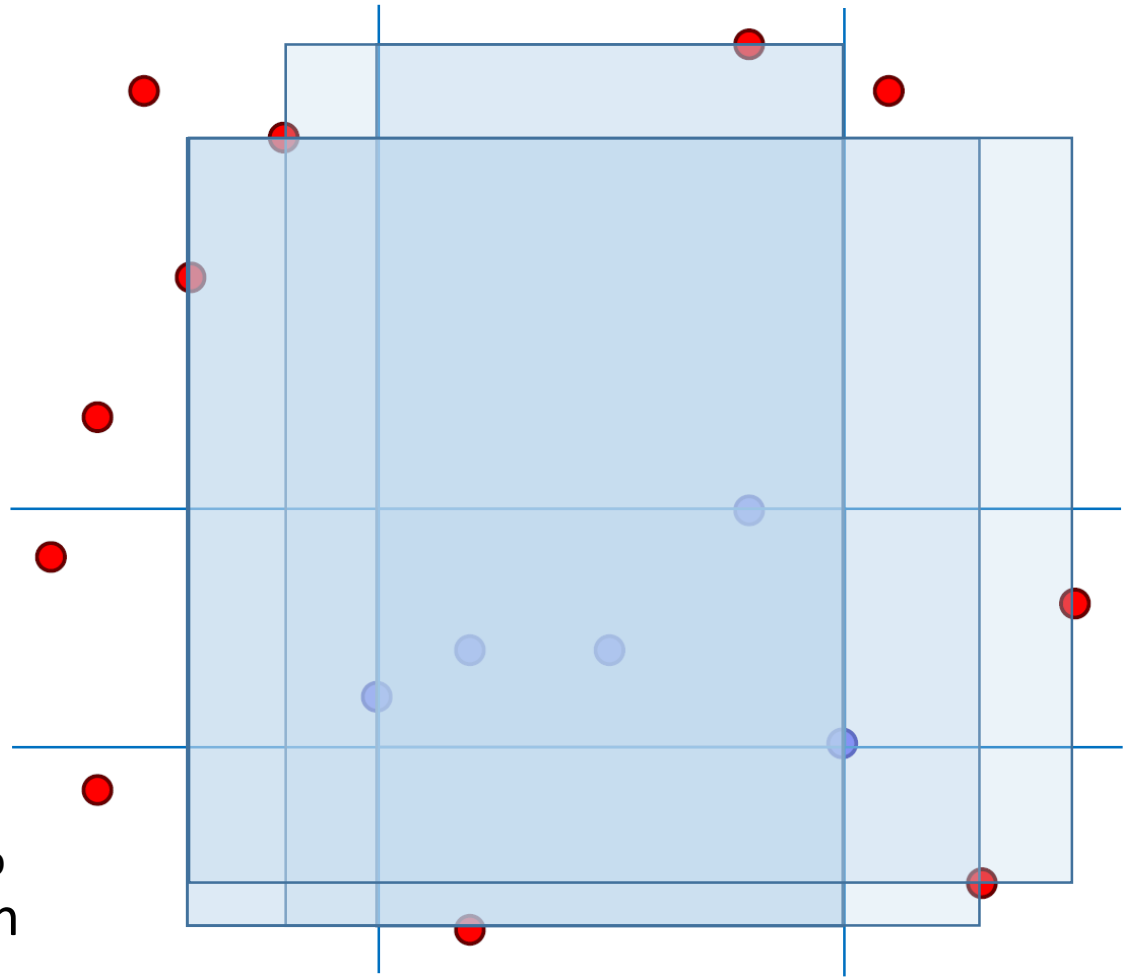
3.2 Pravokotnik

- Naloga:
  - Dane so modre in rdeče točke v ravnini
  - Iščemo največji pravokotnik (največji po ploščini), za katerega velja:
    - Stranice so vodoravne in navpične
    - Vse modre točke ležijo v njem (ali na robu)
    - Vse rdeče točke ležijo zunaj njega (ali na robu)

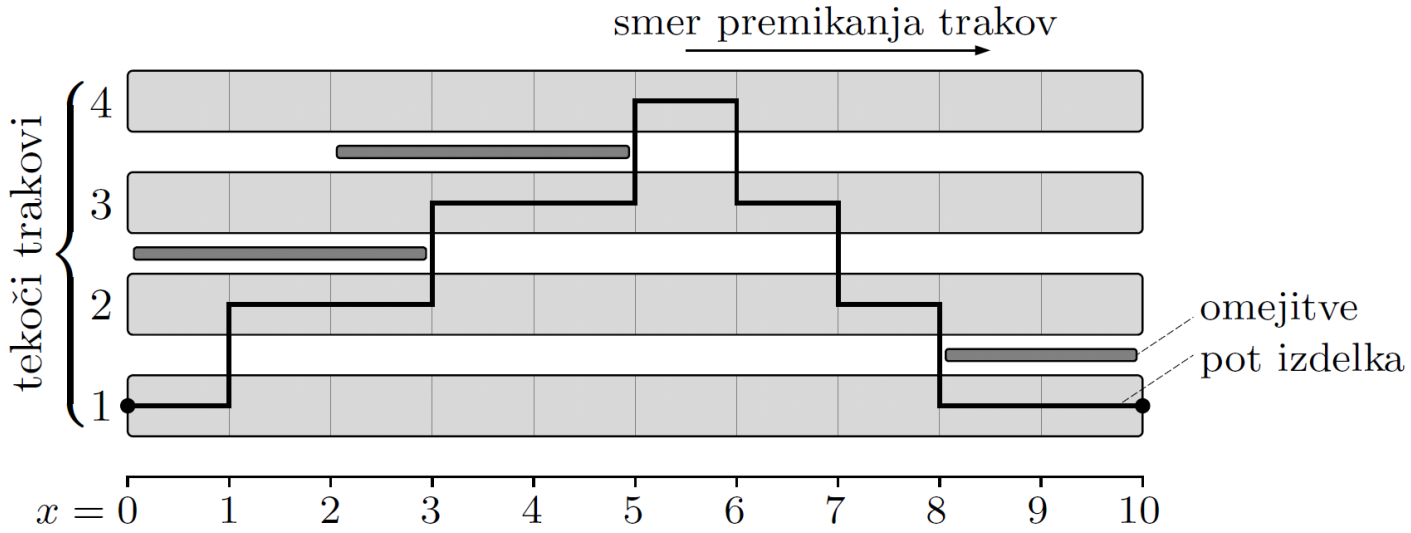


3.2 Pravokotnik

- Rešitev:
  - Poiščimo najmanjši pravokotnik P, ki pokrije vse modre točke
    - Pravokotnik, ki ga iščemo, lahko dobimo tako, da P malo raztegnemo
  - Razdelimo ravnino na 9 delov
  - Poglejmo, kako daleč se da premakniti zgornjo in spodnjo stranico
  - Počasi premikajmo levo stranico v levo
    - Po potrebi spustimo zgornjo in dvignimo spodnjo
    - Nato preizkusimo vse možne položaje desne stranice
      - Po vsakem premiku spustimo zgornjo in dvignimo spodnjo, če je treba
      - Izračunajmo ploščino, zapomnimo si največjo
  - $O(n^2)$ – dve gnezdeni zanki po rdečih točkah



3.3 Tekoči trak

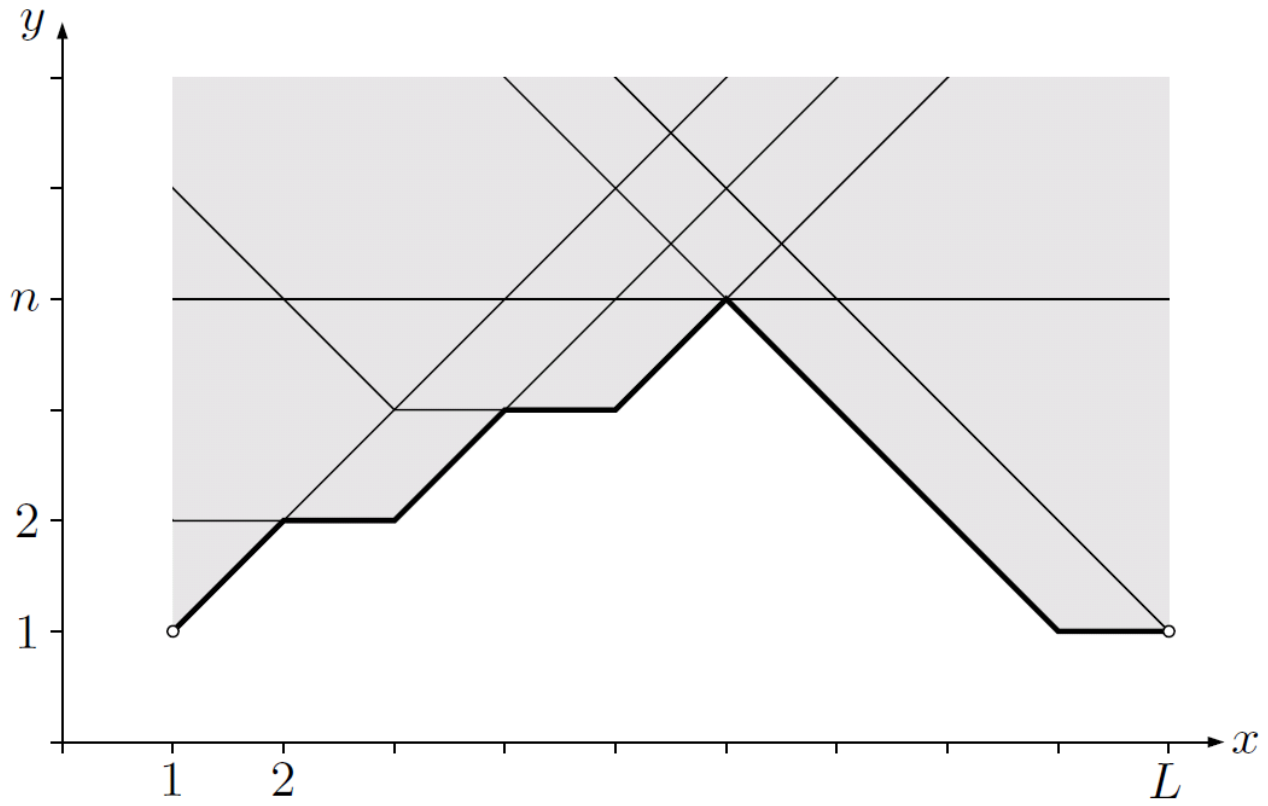


- Imamo *n* trakov dolžine *L* s hitrostmi od 1 do *n*
- Po njih potuje izdelek:
  - Prvi in zadnji meter mora prevoziti na traku 1
  - Po koncu vsakega metra se lahko premakne na sosednji trak ($t \rightarrow t \pm 1$)
  - Je nekaj omejitev oblike „od $x = s_i$ do $x = e_i$ sme biti le na trakovih $1..v_i$“
  - Kako priti čim hitreje od $x = 0$ do $x = L$?

3.3 Tekoči trak

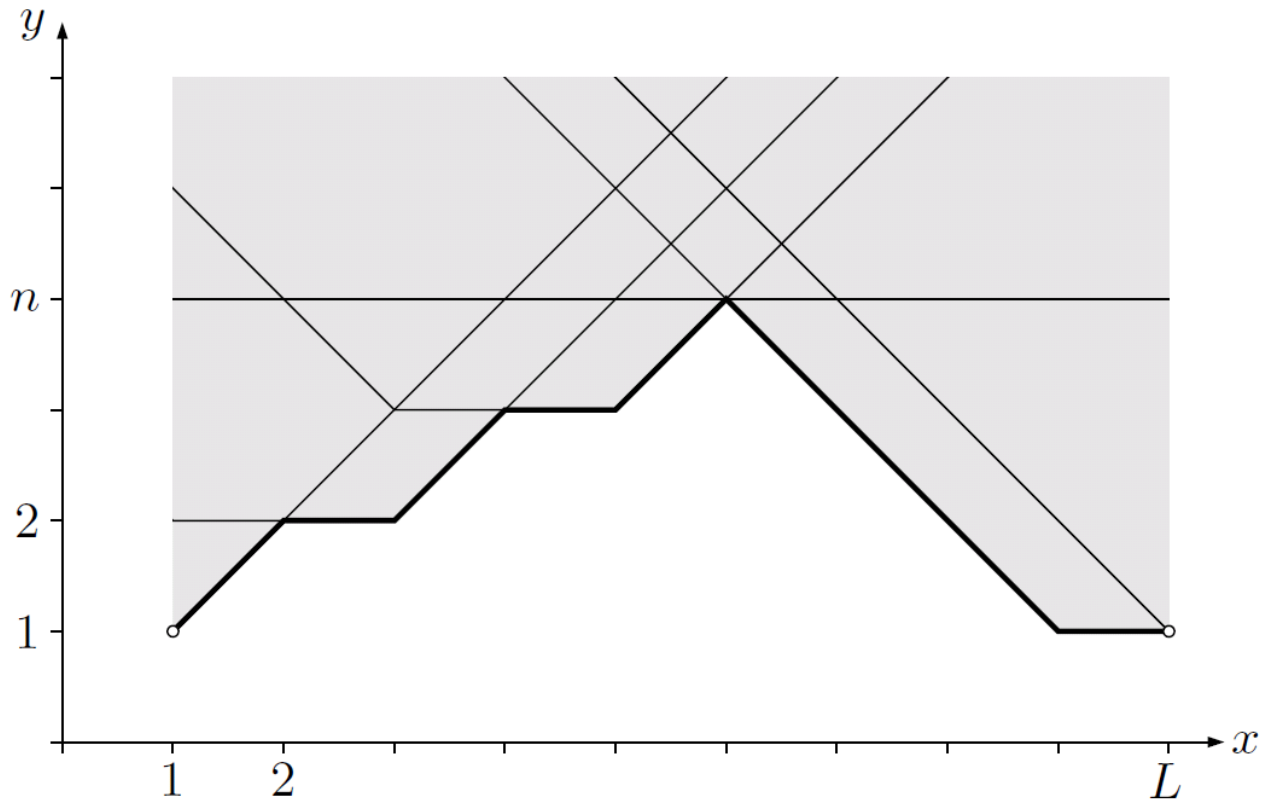
- Preprosta rešitev (če je L majhen):
  - Po vsakem prevoženem metru pogledamo, kateri je najhitrejši trak, po katerem lahko nadaljujemo
  - Recimo, da smo na (x, y). Za novi trak imamo omejitve:
    - $y_{novi} \leq y + 1$, ker ne moremo pospešiti za več kot toliko v 1 koraku
    - $y_{novi} \leq n$, ker je le n trakov
    - $y_{novi} \leq L - x$, da bomo na koncu lahko prišli nazaj na trak 1
    - Za vsako omejitev (s_i, e_i, v_i):
      - $y_{novi} \leq v_i$, če smo na $s_i < x \leq e_i$ [tu velja omejitev]
      - $y_{novi} \leq s_i - x + v_i$, če smo na $x \leq s_i$ [da začnemo pred omejitvijo pravi čas zavirati]
    - Med vsemi temi zgornjimi mejami vzamemo najnižjo
  - Časovna zahtevnost $O(L \cdot n)$, doseže 60% točk

3.3 Tekoči trak



- Boljša rešitev:
  - Pogoje, ki smo jih videli na prejšnji foliji, si lahko predstavljamo kot enačbe premic v ravnini:
    - $y \leq x$
    - $y \leq n$
    - $y \leq -x + L + 1$
    - Za omejitev (s_i, e_i, v_i):
      - [na $x \leq s_i + 1$] $y \leq -x + s_i + 1 + v_i$
      - [na $s_i + 1 \leq x \leq e_i$] $y \leq v_i$
      - [na $e_i \leq x$] $y \leq x + v_i - e_i$
  - Znotraj teh omejitev se hočemo premikati čim hitreje (= po čim višjih y)
  - Torej sledimo zgornjemu robu dopustnega območja

3.3 Tekoči trak



- Boljša rešitev (*nadaljevanje*):
  - Ta rob je lomljena črta iz $O(m)$ odsekov [m = št. omejitev]
  - Kako daleč gremo lahko po trenutnem odseku, preden se kaj spremeni?
    - Do konca daljice, na kateri smo
    - Ali pa do prvega presečišča s kakšno drugo daljico
      - $O(m)$ časa, da pregledamo vseh $O(m)$ daljic
  - Skupaj torej $O(m^2)$ časa, da prehodimo celo lomljeno črto
  - Gre tudi še hitreje, v $O(m \log m)$

3.4 Knjige

- Naloga:
  - Dano je zaporedje števil a_1, a_2, …, a_n
  - Iščemo čim daljše podzaporedje (lahko nestrnjeno),
    ki najprej nekaj časa samo narašča, odtlej pa samo pada
  - Primer:

3.4 Knjige

- Rešitev:
  - Recimo, da je največji element našega podzaporedja na indeksu *i*
  - Levo imamo torej naraščajoče podzaporedje, ki se konča pri *i*
  - Desno imamo padajoče podzaporedje, ki se konča pri *i*
  - Naj bo d_i (oz. p_i) dolžina najdaljšega naraščajočega (oz. padajočega) podzaporedja, ki se konča (oz. začne) na indeksu *i*
  - Potem je najdaljše zaporedje, kakršno zanima nas, z največjim elementom na *i*, dolgo $d_i + p_i - 1$
    - Poiščimo maksimum tega po vseh *i*
- Ostane le še problem nadaljših naraščajočih podzaporedij (da izračunamo vse d_i)
  - Za padajoča podzaporedja naredimo enako od desne proti levi

3.4 Knjige

- Najdaljše naraščajoče podzaporedje, ki se konča s členom i:
  - Ena možnost je $d_i = 1$
  - Drugače pa je pred i v tem podzaporedju nek drug element j
    - $j < i$ in $a_j < a_i$
    - Za ta j je $d_i = d_j + 1$
    - Med takimi j vzemimo torej tistega z najdaljšim d_j
  - Tega ni težko računati z dvema gnezdenima zankama:

```
for (i = 1; i <= n; i++)
    for (j = 1, d[i] = 1; j < i; j++)
        if (a[j] < a[i]) d[i] = max(d[i], d[j] + 1);
```

  - To je $O(n^2)$, kar je za velike primere že prepočasi [60% točk]

3.4 Knjige

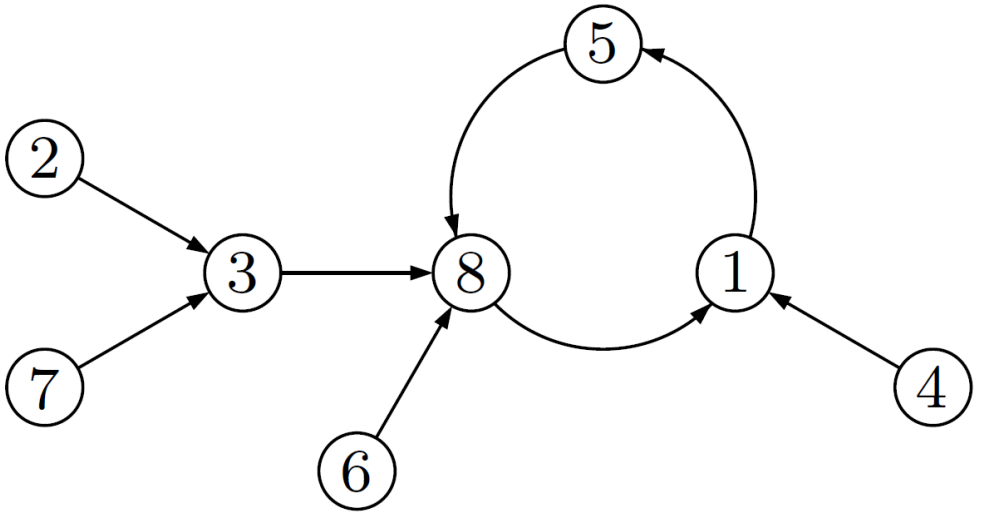
- Boljša rešitev za najdaljša naraščajoča podzaporedja (NNP):
  - NNP, ki se konča s členom *i*, dobimo tako,
    da vzamemo neko NNP, ki se konča s členom *j*, in ga podaljšamo s členom *i*
    - Pogoj za to je seveda $j < i$ in $a_j < a_i$
    - Lažje je podaljšati takšno podzaporedje, ki se konča z manjšim elementom
    - Recimo, da si za vsako dolžino *k* zapomnimo najmanjšo vrednost z_k,
      s katero se konča kakšno NNP dolžine *k*
  - Ko smo pri členu *i*, poiščimo največji *k*, za katerega je z_k še $< a_i$
    - Zanj je potem $d_i = k + 1$
    - In a_i je nova vrednost z_{k+1}
  - To lahko naredimo z bisekcijo po zaporedju *z*, ker je le-to naraščajoče
    - Tako smo pri $O(n \log n)$

3.5 Posredne volitve

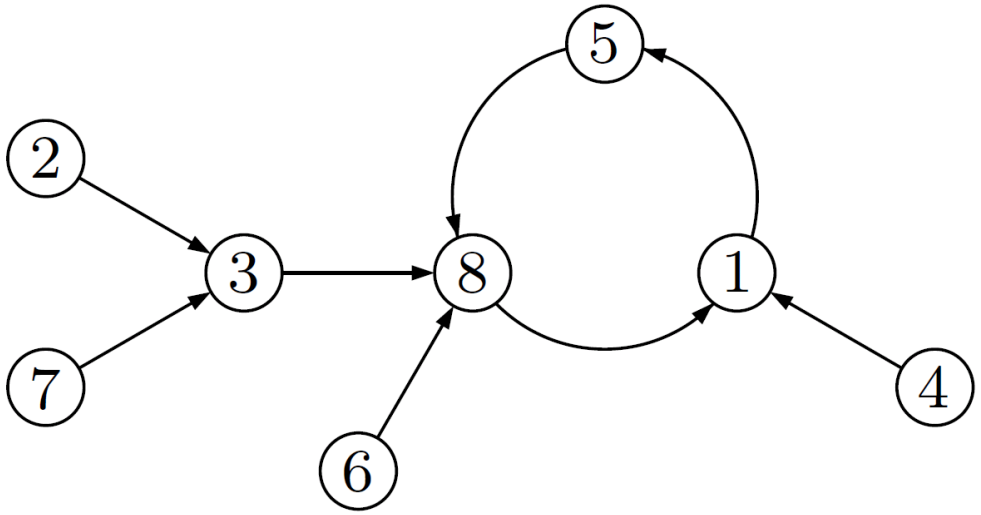
- Naloga:
  - Imamo n državljanov (1..n), vsak imenuje svojega *zastopnika*: $u \to g(u)$
  - Na začetku ima vsak 1 kroglico, nato si jih prerazporejajo v fazah:
    - V vsaki fazi odnese vsak državljan vse kroglice, ki jih je imel na začetku te faze, svojemu zastopniku
  - Naj bo $f_k(u)$ število kroglic, ki jih ima u na koncu k-te faze
    - Izkaže se, da če pade $f_k(u)$ na 0, tam tudi ostane (pri večjih k) – rečemo, da je *u izpadel*
  - V kateri fazi K se zadnjič zgodi, da kdo izpade?
    - Izračunaj tudi $f_K(u)$ za vse u

3.5 Posredne volitve

- Sledimo poti *u*-jeve kroglice:
  - V zaporedju $u \to g(u) \to g(g(u)) \to g(g(g(u))) \to \ldots$ se neka oseba prej ali slej pojavi drugič
  - Torej imamo $u \to g(u) \to g(g(u)) \to \ldots \to v \to g(v) \to g(g(v)) \ldots \to v$
  - Nadaljuje se očitno z $\to g(v) \to g(g(v))$ in tako naprej, torej se odslej ciklično ponavlja
  - Če bi začeli s kakšnim drugim *u*, bi tudi prišli do cikla, mogoče istega, mogoče kakšnega drugega
- Struktura grafa je torej takšna:
  - Ena ali več komponent
  - V vsaki komponenti je en usmerjen cikel
  - In 0 ali več dreves, pripetih na ta cikel
  - V njih so vse povezave usmerjene k ciklu

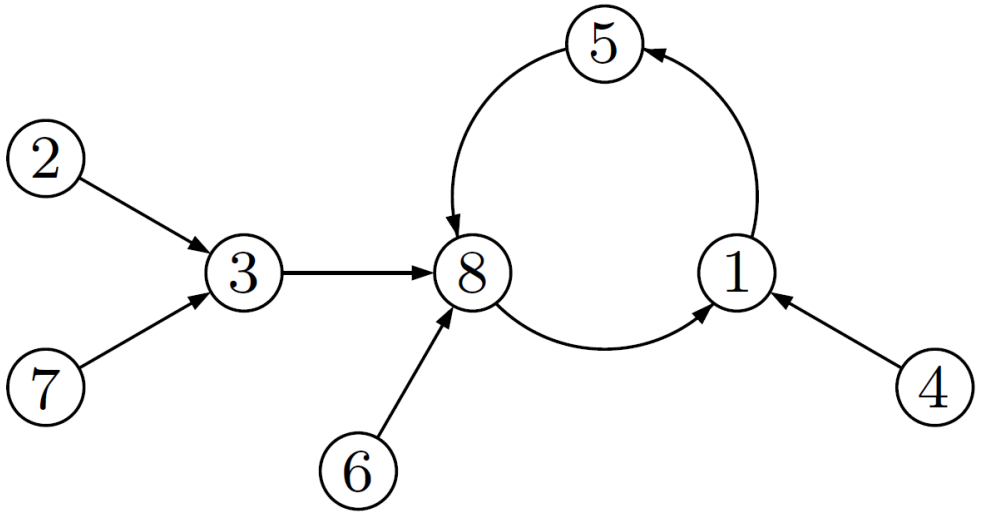


3.5 Posredne volitve



- Kako se po taki komponenti prenašajo kroglice?
  - Kroglice ljudi na ciklu ostanejo ves čas na ciklu, le v vsaki fazi se zamaknejo za 1 mesto naprej po ciklu
  - Ti ljudje torej nikoli ne izpadejo
  - Kroglice ljudi na drevesih se v vsaki fazi pomaknejo en korak bliže ciklu
  - Prej ali slej vse pridejo na cikel, ljudje v drevesih torej zagotovo izpadejo
- Kdaj je konec izpadanja?
  - Naj bo d_u razdalja od u do najbližje točke na ciklu
  - u-jeva kroglica torej potrebuje d_u faz, da pride na cikel
  - Šele po $\max_u d_u$ fazah so vse kroglice na ciklih
  - Izpadanje se neha takrat in nič prej

3.5 Posredne volitve



- Izračun $f_K(u)$:
  - Namesto da se vprašamo
    „koliko kroglic ima u (po K-ti fazi)?“
    se vprašajmo „ pri kom je (po K-ti fazi) kroglica, ki jo je imel u na začetku?“
  - u-jeva kroglica pride po d_u korakih na cikel
    - Poleg d_u si zapomnimo, katera je tista najbližja točka na ciklu, recimo c_u
  - Potem naredi še $K - d_u$ korakov naprej po ciklu
    - To je isto kot $(K - d_u)$ % *dolzina_tega_cikla*
    - Koristno je imeti za vsak cikel seznam točk na njem
      in za c_u podatek o tem, na katerem ciklu leži in na katerem indeksu v njegovem seznamu