RTK 2019
Naloge in rešitve

Janez Brank

1.1 Smučarski užitki

Primer: 3 proge
- Začetne ocene: [7, 3.3, 6]
- Vožnje: 1, 3, 2, 1, 1, 2
→ Rezultat: 7 + 6 + 3.3 + 6.3 + 5.67 + 2.97 = 31.24

- Danih je p smučarskih prog
  - Za vsako je znana začetna ocena
  - Ko jo prevozimo, se ji ocena zmanjša na 90% dosedanje
  - Dano je zaporedje voženj
  - Izračunaj vsoto ocen vseh voženj
- Rešitev:
  - Ocene hranimo v tabeli, poleg tega imamo še spremenljivko za vsoto
  - V zanki gremo po vožnjah, prištejemo oceno trenutne proge vsoti in jo zmanjšamo

```
vsota = 0; ocene = […]
for stProge in voznje:
    vsota += ocene[stProge]
    ocene[stProge] *= 0.9
print(vsota)
```

1.2 Razmazani seznam

- Vhod: števili m, v in množica A
  - Njeni elementi so naravna števila, dobivamo jih eno po eno v naraščajočem vrstnem redu
  - Sestavimo novo množico B, ki poleg vsakega elementa iz A vsebuje še m prejšnjih in v naslednjih števil
  - Elemente B-ja moramo izpisovati sproti, vsakega le enkrat, v naraščajočem vrstnem redu
  - Na primer: $A = \{1, 3, 4, 9\}$, $m = 2$, $v = 1 \rightarrow B = \{-1, 0, 1, 2, 3, 4, 5, 7, 8, 9, 10\}$
- Rešitev:
  - Zapomnimo si zadnje že izpisano število, recimo z
  - Elemente A-ja beremo v zanki
    - Ko preberemo $x \in A$, moramo izpisati števila od $\max\{z + 1, x - m\}$ do $x + v$
    - Nato postavimo z na $x + v$

1.3 Veriga

Ne prav dolgo potem se je pot na Čatež zopet krebri obrnil. Mrtoláz pa se je menil po nemško in zmerom od pijače, kar je Dolenjčev najljubši pogovor, ki ga ne konča tako hitro, ako se ga je polotil. Nagovarjal je naju, da naj zložíva imenitno pesem letošnjemu vincu na čast. »Že sam sem se prtil ž njo,« pravi, »pa mi ni po volji, kar sem zverižil. Časi je bil Kančnik, tudi šmarski šomašter je zaokrožil katero. Zdaj ga pa ni, da bi kaj znal. Kar sta le-ta dva pomrla, nimamo kaj peti, stare so se pozabile, novih ni!«

- Fran Levstik, Popotovanje iz Litije do Čateža

- Vhod: besedilo v več vrsticah, vse črke so enako široke
  - Veriga = več enakih stavkov v istem stolpcu v več zaporednih vrsticah
  - Poišči najdaljšo verigo
  - Besedilo dobimo po vrsticah, ne vsega naenkrat
- Rešitev:
  - Vzdržujmo tabelo, v kateri $v[x]$ pove trenutno dolžino verige v stolpcu x
  - Poleg trenutne vrstice si zapomnimo še prejšnjo
  - Za vsak x:
    - Če je *trenutna*$[x]$ == *prejšnja*$[x]$, povečamo $v[x]$ za 1, sicer ga postavimo na 1
    - Po vsakem povečanju pogledamo, če je to najdaljša veriga doslej, in si jo zapomnimo

1.4 Jezero

- Imamo jezero z zapornico in merilnikom globine (1..100)
  - *GlobinaVode*() počaka na naslednjo meritev (1x na uro) in jo vrne
  - *PremakniZapornico*(**bool** *odpri*) odpre ali zapre zapornico
  - Napiši program, ki teče v zanki in upravlja zapornico:
    - Če je globina < 33, mora biti zapornica zaprta
    - Če je globina > 66, mora biti zapornica odprta
    - Če zadnjih 12 meritev gladina narašča, moramo zapornico zapreti
    - Če zadnjih 12 meritev gladina pada, moramo zapornico odpreti

1.4 Jezero

- Rešitev:
  - Vzdržujmo podatek o smeri gibanja (1 = narašča, –1 = pada, 0 = se ne spreminja) in kako dolgo ta smer že traja
  - Postopek teče v neskončni zanki:
    - Preberi novo gladino
    - Izračunajmo novo smer iz nove gladine in prejšnje gladine
    - Če je nova smer enaka dosedanji, povečamo trajanje za 1, sicer ga postavimo na 1
    - Novo smer in novo gladino si zapomnimo kot prejšnjo (za naslednjo iteracijo)
    - Če je nova gladina < 33, zapremo zapornico; če > 66, jo odpremo
    - Sicer, če je trajanje $\geq$ 11 in smer = –1, zapremo zapornico;
      če je trajanje $\geq$ 11 in smer = 1, jo odpremo.

1.5 Stolpci in vrstice

- Položaj celice v tabeli (razpredelnici) opišemo s številko stolpca in vrstice
  - Stolpce predstavimo z nizi črk: A, B, …, Z, AA, AB, …, AZ, BA, …, ZY, ZZ, AAA, AAB, …
  - Nizi so torej urejeni po dolžini, enako dolgi pa po abecedi
- Dobimo niz, v katerem je več takih parov (stolpec, vrstica) staknjenih skupaj brez presledkov
  - Primer: A1A3AA3457BB54NTL1
  - Tak opis moramo predelati v pare številskih koordinat (x, y):
    (1, 1), (1, 3), (27, 3457), (54, 54), (9996, 1)

1.5 Stolpci in vrstice

- Rešitev:
  - Berimo vhod po znakih
  - Dokler smo pri črkah, računamo stolpec
  - Ko smo pri številkah, računamo vrstico
  - Ko preklopimo s številk na črke (ali na koncu vhoda), izpišemo trenutni par (x, y) in se pripravimo na naslednjega
- Pri vrsticah:
  - $y = 0$; za vsak znak c: $y = 10 * y + (c - \text{'0'})$
- Pri stolpcih:
  - $x = 0$; za vsak znak c: $x = 26 * x + (c - \text{'a'})$
  - To še ni dobro – pri vsaki dolžini niza se začnejo številke od 0 naprej
  - Torej moramo prišteti še število krajših nizov: $1 + 26 + 26^2 + 26^3 + \ldots + 26^{d-1}$
  - To lahko računamo sproti, ko računamo stolpec, na koncu prištejemo k x

    $x = 0$; prej = 1
    za vsak znak c:
    $x = 26 * x + (c - \text{'a'})$; prej = 1 + 26 * prej;
    x += prej

2.1 Anagramska razdalja

- Niz *s* bi radi predelali v poljuben anagram *t*-ja s čim manj urejevalniškimi operacijami
  - Anagram = niz, ki ga lahko dobimo iz drugega le s spremembo vrstnega reda znakov
- Urejevalniške operacije:
  - Vrivanje enega znaka: `sol` $\rightarrow$ `stol`; `tal` $\rightarrow$ `stal`; `cel` $\rightarrow$ `celo`
  - Brisanje enega znaka: `steze` $\rightarrow$ `stez`; `steze` $\rightarrow$ `teze`; `otrok` $\rightarrow$ `otok`
  - Sprememba enega znaka v drugega: `teta` $\rightarrow$ `meta`; `teta` $\rightarrow$ `trta`; `teta` $\rightarrow$ `tete`
- Primer:
  - s = `stol`, t = `volt` $\rightarrow$ razdalja 1 (`stol` $\rightarrow$ `vtol`)
  - s = `arbalest`, t = `balasta` $\rightarrow$ razdalja 2 (`arbalest` $\rightarrow$ `aabalest` $\rightarrow$ `aabalst`)

2.1 Anagramska razdalja

c	#s(c)	#t(c)	r(c)
A	2	3	–1
B	1	1	0
E	1		1
L	1	1	0
R	1		1
S	1	1	0
T	1	1	0

- Rešitev:
  - Vrstni red črk ni pomemben, le število pojavitev vsake črke
  - Naj bo $\#_s(c)$ oz. $\#_t(c)$ število pojavitev črke c v s-ju oz. v t-ju
  - Torej je $\min\{\#_s(c), \#_t(c)\}$ skupnih obema in jih lahko v s-ju pustimo pri miru, ker jih bomo potrebovali tudi v t-ju
  - Naj bo $K = \sum_c \min\{\#_s(c), \#_t(c)\}$ število vseh skupnih črk
  - V s je torej $O = |s| - K$ črk odveč in $M = |t| - K$ črk manjka
    - Od tega lahko $\min\{O, M\}$ črk spremenimo
    - Če je $O > M$, potrebujemo še $O - M$ brisanj
    - Če je $M > O$, potrebujemo še $M - O$ dodajanj
  - Rezultat je torej $\max\{M, O\} = \max\{|s|, |t|\} - K$
- Še ena pot do enakega rezultata:
  - Za vsako črko izračunajmo razliko $r(c) = \#_s(c) - \#_t(c)$
  - Potem je $O = \sum_c \max\{r(c), 0\}$ in $M = \sum_c \max\{-r(c), 0\}$

Primer: s = `arbalest`, t = `balasta`

$K = 6$, $O = 2$, $M = 1$

$\max\{M, O\} = 2$

$\max\{|s|, |t|\} - K =$

$\max\{8, 7\} - 6 = 8 - 6 = 2$

2.2 Zaboji

Operacija	Stanje po njej
(*začetno stanje*)	4, 1, 3, 5, 2
zamik v levo	1, 3, 5, 2, 4
zamik v levo	3, 5, 2, 4, 1
poberi zadnji zaboj	3, 5, 2, 4
zamik v desno	4, 3, 5, 2
poberi zadnji zaboj	4, 3, 5
zamik v desno	5, 4, 3
poberi zadnji zaboj	5, 4
poberi zadnji zaboj	5
poberi zadnji zaboj	(*prazno*)

- Imamo skladišče z *n* zaboji v vrsti (od leve proti desni)
  - Zaboji so oštevilčeni s števili od 1 do *n*, vsako se pojavlja enkrat, v nekem premešanem (in podanem) vrstnem redu
  - Sistem robotskih rok v skladišču zna izvajati naslednje tri operacije:
    - Ciklično zamakni zaboje za 1 mesto v levo (najbolj levi zaboj pride s tem na desni konec skladišča)
    - Ciklično zamakni zaboje za 1 mesto v desno (najbolj desni zaboj pride s tem na levi konec skladišča)
    - Poberi najbolj desni zaboj iz skladišča
  - Naloga: s čim manj temi operacijami poberi zaboje iz skladišča v naraščajočem vrstnem redu (od 1 do n)

2.2 Zaboji

- Rešitev:
  - Pobiranj bo treba natanko *n*, tu ni nobene izbire
  - Vprašanje je le, koliko zamikov bo treba
  - Med dvema pobiranjema nima smisla uporabljati tako levih kot desnih zamikov
  - Vprašanje je torej le, ali bomo do naslednjega zaboja, ki ga moramo pobrati, prišli prej z zamiki v levo ali v desno

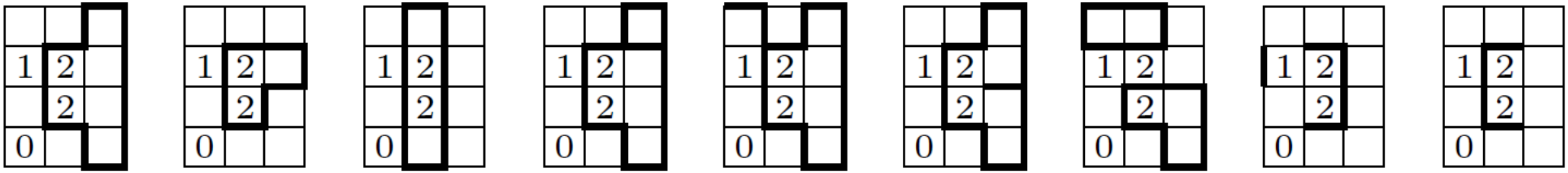
```
// vhod: začetni seznam zabojev s[1..n]
r = 0 // tu bomo računali število operacij
for (z = 1; z <= n; z++) {
    k = n – z + 1;   // število preostalih zabojev; so v s[1..k]
    i = položaj zaboja z v seznamu s[1..k];
    L = i; D = k – i; // potrebno število zamikov v levo oz. v desno
    r = r + min{i, k – i} + 1;
    if L < D then ciklično zamakni s[1..k] za L mest v levo
    else ciklično zamakni s[1..k] za D mest v desno;
    zaboj z je zdaj na koncu seznama, v s[k]; pobriši ga od tam;
return r
```

2.2 Zaboji

- Boljša rešitev:
  - Iskanje zaboja *z* v seznamu traja $O(n)$ časa, prav tako ciklični zamik
    - Pri iskanju pravzaprav ne gre toliko za iskanje kot za ugotavljanje tega, koliko zabojev je desno od z v skladišču
  - Zamik v O(1), če si le zapomnimo indeks *d* najbolj desnega zaboja
    - Toda potem brisanje ne bo s konca seznama in bo trajalo $O(n)$
    - Lahko pa v seznamu pustimo luknjo: $L[i] = 1$, če je tam luknja, in 0 sicer
    - Lepo: posamezni zaboj je zdaj vedno na istem mestu v tabeli $s[1..n]$ — ni ga treba iskati
    - Toda kako potem prešteti, koliko je lukenj med *z* in najbolj desnim zabojem?
      - Potrebujemo vsoto $L[i] + L[i + 1] + \ldots + L[d]$, če je *i* indeks *z*-ja
    - Nad tabelo $L[1..n]$ vdržujmo Fenwickovo drevo, pa bomo za vsako brisanje in izračun vsote porabili $O(\log n)$ časa – skupaj $O(n \log n)$ namesto $O(n^2)$
- Ali pa namesto seznama *s* uporabimo drevo (rdeče-črno ipd.)
  - V vsakem vozlišču vzdržujmo še število vseh elementov v njegovem poddrevesu

2.3 Ograje

- Dana je karirasta mreža z $w \times h$ polji
  - Na posameznem polju je lahko število od 0 do 3 ali pa je prazno
  - Med polji so lahko ograje
  - Če je na polju število, mora biti okrog njega točno toliko ograj
  - Vse ograje morajo tvoriti en sam sklenjen cikel
- Naloga:
  - Opiši podatkovno strukturo, s katero bi predstavil stanje mreže
  - Opiši postopek, ki preveri, ali ustreza gornjima dvema pogojema



2.3 Ograje

- Podatkovna struktura:
  - Lahko uporabimo nekaj 2-d tabel:
    int *m*[*w*][*h*]; *// števila v celicah (–1 = prazno)*
    bool *vod*[*w*][*h* + 1]; *// vodoravne ograje*
    bool *nav*[*w* + 1][*h*]; *// navpične ograje*
- Postopek:
  - Z dvema zankama po vseh poljih preštejmo ograje in preverimo, če jih je okrog vsakega polja pravo število:
    if (*m*[*x*][*y*] != (*vod*[*x*][*y*] ? 1 : 0) + (*vod*[*x*][*y* + 1] ? 1 : 0) +
    (*nav*[*x*][*y*] ? 1 : 0) + (*nav*[*x* + 1][*y*] ? 1 : 0)) **return false**;
  - Postavimo se v poljubno točko, kjer je ograja, in sledimo ograjam od tam
  - Na vsakem koraku pogledamo, v katero od 3 možnih smeri je mogoče nadaljevati
    - (ne v tisto, iz katere smo prišli)
    - Če je možnih več nadaljevanj, so ograje razvejene in mreža je neveljavna
  - Sčasoma pridemo nazaj na začetek – če smo s tem obiskali vse ograje, je mreža veljavna, sicer pa ne (več ločenih skupin ograj)

2.4 Past za žvižgače

- Dobimo število *n* (od 1 do 32) in neko besedilo
  - Nekaj pojavitev besed »*in*« in »*ter*« spremeni tako, da bo nastala ena od 32 različnih različic besedila odvisno od vrednosti *n*
  - Podrobnosti tega, katere pojavitve spremeniš in kako, niso predpisane

```
ena in dve in tri in štiri in pet in šest in sedem
ena ter dve in tri in štiri in pet in šest in sedem
ena in dve ter tri in štiri in pet in šest in sedem
ena ter dve ter tri in štiri in pet in šest in sedem
ena in dve in tri ter štiri in pet in šest in sedem
ena ter dve in tri ter štiri in pet in šest in sedem
ena in dve ter tri ter štiri in pet in šest in sedem
ena ter dve ter tri ter štiri in pet in šest in sedem
ena in dve in tri in štiri ter pet in šest in sedem
ena ter dve in tri in štiri ter pet in šest in sedem
ena in dve ter tri in štiri ter pet in šest in sedem
ena ter dve ter tri in štiri ter pet in šest in sedem
ena in dve in tri ter štiri ter pet in šest in sedem
ena ter dve in tri ter štiri ter pet in šest in sedem
ena in dve ter tri ter štiri ter pet in šest in sedem
ena ter dve ter tri ter štiri ter pet in šest in sedem
ena in dve in tri in štiri in pet ter šest in sedem
ena ter dve in tri in štiri in pet ter šest in sedem
ena in dve ter tri in štiri in pet ter šest in sedem
ena ter dve ter tri in štiri in pet ter šest in sedem
ena in dve in tri ter štiri in pet ter šest in sedem
ena ter dve in tri ter štiri in pet ter šest in sedem
ena in dve ter tri ter štiri in pet ter šest in sedem
ena ter dve ter tri ter štiri in pet ter šest in sedem
ena in dve in tri in štiri ter pet ter šest in sedem
ena ter dve in tri in štiri ter pet ter šest in sedem
ena in dve ter tri in štiri ter pet ter šest in sedem
ena ter dve ter tri in štiri ter pet ter šest in sedem
ena in dve in tri ter štiri ter pet ter šest in sedem
ena ter dve in tri ter štiri ter pet ter šest in sedem
ena in dve ter tri ter štiri ter pet ter šest in sedem
ena ter dve ter tri ter štiri ter pet ter šest in sedem
```

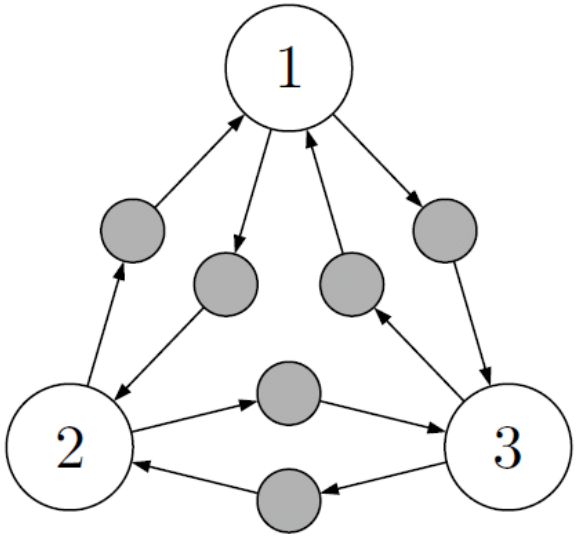
2.4 Past za žvižgače

- Rešitev:
  - Najlažje je, če uporabimo spodnjih 5 bitov števila *n*
  - Za vsako od prvih 5 pojavitev besed *in*/*ter* poglejmo ustrezni bit *n*-ja in nato izpišimo *in* ali *ter* glede na vrednost tega bita, ne glede na vhodno besedo
  - Vse ostale besede in presledke v vhodnem besedilu izpišimo brez sprememb
  - Lahko na primer beremo po znakih:

    s = prazen niz; *bit* = 0
    while je trenutni znak črka:
        dodaj jo na konec *s*-ja in se premakni na naslednjo
    if *bit* < 5 **and** (*s* == `"in"` **or** *s* == `"ter"`):
        if (*n* >> *bit*) & 1 == 1 **then** izpiši `"in"` **else** izpiši `"ter"`
        bit += 1
    else izpiši *s* brez sprememb;
    while trenutni znak ni črka:
        izpiši jo brez sprememb in se premakni na naslednjo

  - Lahko pa pač besedilo kako drugače razbijamo na besede, verjetno je v knjižnici našega programskega jezika kaj uporabnega

2.5 Pekarna



- Trije strežniki, vsak nadzoruje po enega od napajalnikov drugih dveh
  - En strežnik lahko drugemu pošlje vprašanje (celo število) in kot odgovor dobi dvakratnik tega števila
  - Možni sta dve vrsti napak:
    - Strežnik se zatakne in odtlej ves čas pošilja isti odgovor
    - Strežnik deluje počasi in njegovi odgovori vse bolj zamujajo
  - Naloga: napiši kontrolni program, ki bo v neskončni zanki tekel na vsakem strežniku
    - Vsako sekundo naj drugima dvema pošlje novo vprašanje
    - Če od drugega strežnika več kot 10 sekund ne dobi nobenega odgovora ali pa dobi odgovor na več kot 10 sekund staro vprašanje, naj mu ugasne napajalnik

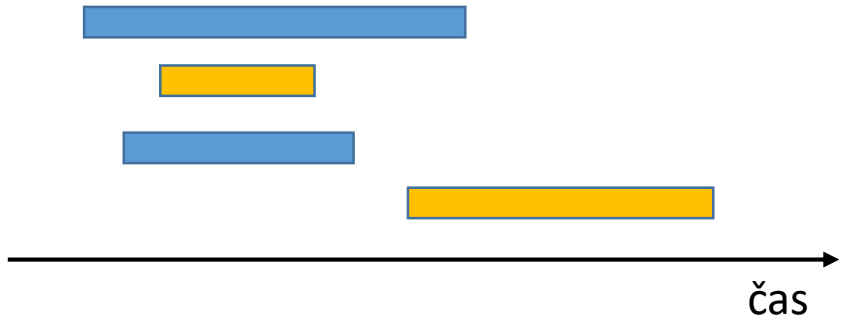
2.5 Pekarna

- Rešitev:
  - Vprašanja ne smejo biti ves čas enaka, sicer ne bomo zaznali primerov, ko se strežnik zatakne in ves čas pošilja isti odgovor
  - Vedeti moramo, kdaj smo poslali neko vprašanje (da bomo vedeli, ko dobimo odgovor na več kot 10 sekund staro vprašanje)
  - Najlažja rešitev: vprašanje naj bo kar trenutni čas v sekundah

```
while (true) {
    int zdaj = Počakaj();  // počaka na novo sekundo in vrne njeno zap. št.
    for (int x = 1; x <= 3; x++) if (x != KdoSem()) { // obdelamo druga dva strežnika
        if (zdaj – zadnji[x] > 10) UgasniNapajalnik(x); // že predolgo ni odgovora
        else Vprasanje(x, zdaj); // pošlji novo vprašanje
        while ((odg = Odgovor(x)) != 0) { // preberimo vse prispele odgovore
            zadnji[x] = zdaj;
            if (zdaj – odg / 2 > 10) UgasniNapajalnik(x); // odgovor na prestaro vprašanje
        }
    }
}
```

3.1 Fitnes

- Vhod: podatki o obiskovalcih fitnesa
  - Zapisi oblike ⟨čas prihoda, čas odhoda, tip naprave⟩
  - Časi so v desetinkah sekunde znotraj dneva = cela števila od 0 do 863999
  - Če hoče več ljudi hkrati uporabljati napravo istega tipa, potrebujemo ustrezno število naprav tega tipa
  - Za vsak tip naprave ugotovi, koliko naprav tega tipa potrebujemo

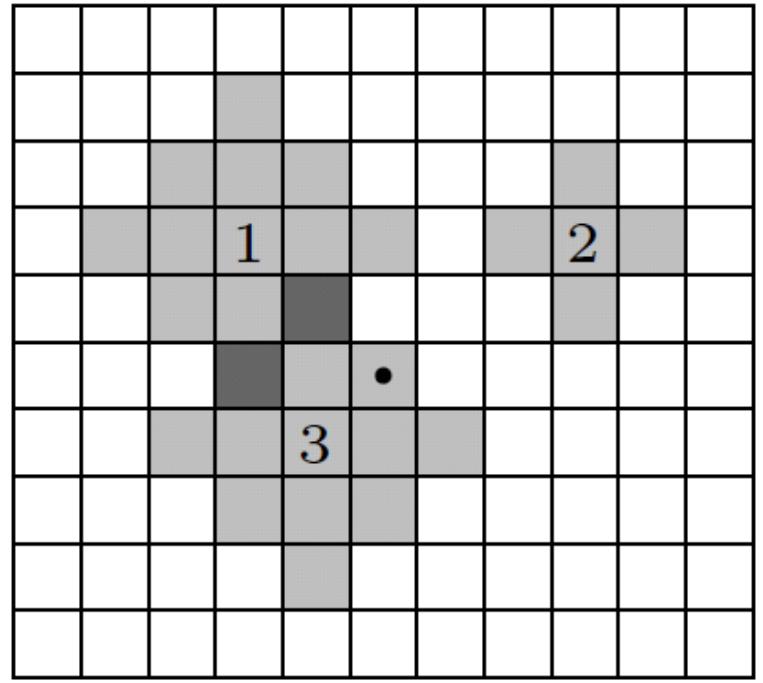


3.1 Fitnes

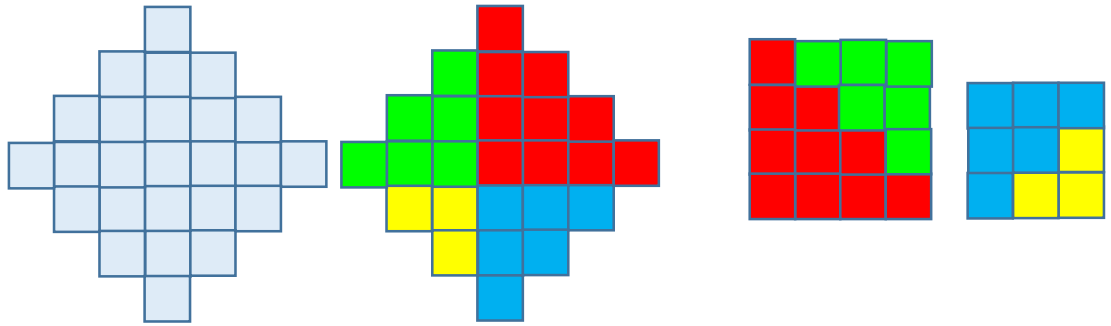
- Rešitev:
  - Obiskovalce uredimo tako, da pridejo tisti z istim tipom naprave skupaj
    - V nadaljevanju se bomo ukvarjali z vsako tako skupino posebej
  - Preprosta rešitev: imejmo tabelo 864000 elementov, ki za vsak časovni interval povedo, koliko obiskovalcev je takrat prisotnih
    - Gremo v zanki po vseh obiskovalcih, za vsakega v še eni zanki od $t_{prihoda}$ do t_{odhoda} in povečujemo števce v tabeli
    - Slabost: pri večjih testnih primerih bo to prepočasi
  - Boljša rešitev:
    - Do sprememb v številu obiskovalcev lahko pride le takrat, ko kdo pride ali gre
    - Za vsakega obiskovalca pripravimo dva para ($t_{prihoda}$, +1) in (t_{odhoda}, –1)
    - Te pare uredimo po času, pri istem času pa odhode pred prihodi
    - Sprehodimo se po tem seznamu in seštevajmo spremembe v številu obiskovalcev
    - Po vsakem povečanju preverimo, če je to največje število doslej

3.2 Telefonsko omrežje

- Na karirasti mreži stoji *n* telefonskih oddajnikov (x_i , y_i , r_i)
  - Tak oddajnik pokriva celice (x, y), za katere je $|x - x_i| + |y - y_i| \leq r_i$.
  - Mreža je dovolj velika, da noben tak karo ne gleda čez rob
  - Kari različnih oddajnikov se lahko prekrivajo
  - Koliko polj je pokritih (z vsaj enim oddajnikom)?



3.2 Telefonsko omrežje

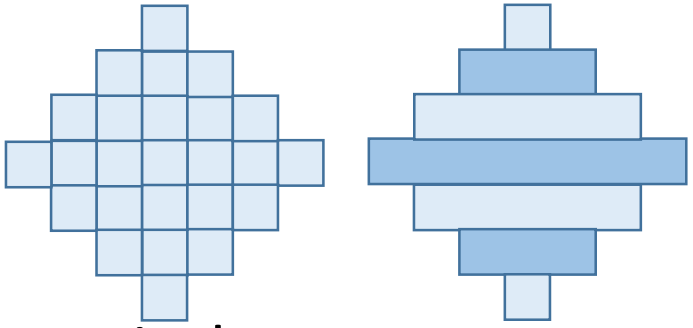


- Naivna rešitev:
  - Pojdimo v zanki po oddajnikih, pri vsakem z dvema zankama po vseh poljih, ki jih ta oddajnik pokriva, in jih dodajajmo v množico / razpršeno tabelo itd.

```
for (i = 0; i < n; i++)
    for (dy = –r[i]; dy <= r[i]; dy++)
        for (dx = –(r[i] – abs(dy)); dx <= r[i] + abs(dy); dx++)
            dodaj (x[i] + dx, y[i] + dy) v množico;
```

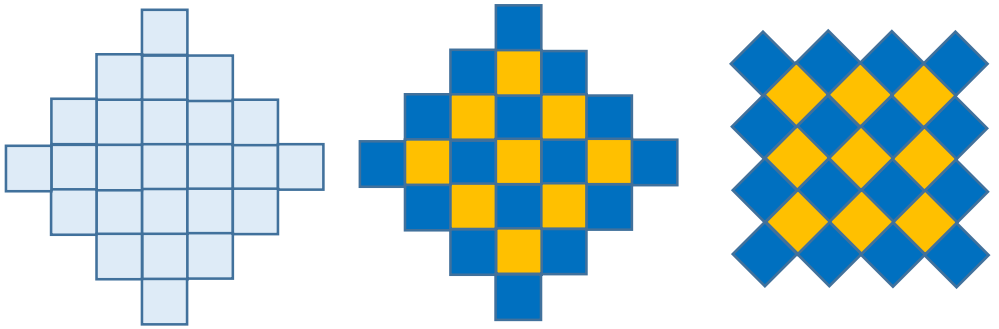
  - Oddajnik z močjo r pokriva $r^2 + (r + 1)^2$ polj
  - Skupaj je torej lahko pokritih $O(k\, r^2)$ polj
  - Ta rešitev porabi preveč pomnilnika (in tudi časa)

3.2 Telefonsko omrežje

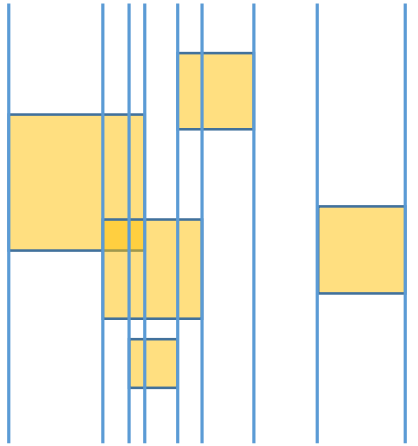


- Boljša rešitev: razrežimo karo na bloke po vrsticah
  - V isti vrstici *y* so lahko prisotni bloki od več oddajnikov
  - Lahko se tudi prekrivajo — koliko polj je pokritih?
  - Podobna ideja kot pri 1. nalogi: pripravimo pare (x_{od}, +1) in (x_{do} + 1, –1)
  - Pregledujmo jih naraščajoče po *x* in seštevajmo spremembe v pokritosti
  - Če od prejšnjega *x* do trenutnega velja pokritost > 0, ta polja prištejmo k skupnemu številu pokritih polj
- Kako vemo, katere *y* gledati in katere oddajnike gledati pri posameznem *y*?
  - Spet podobna ideja: pripravimo pare ($y_i - r_i$, +1) in ($y_i + r_i + 1$, –1)
  - Glejmo jih po naraščajočem *y*, pri vsaki spremembi izračunajmo nov seznam aktivnih oddajnikov in ga uporabljajmo do naslednje spremembe
- $O(kr \log r)$ časa in $O(k)$ prostora – dovolj dobro za naše testne primere

3.2 Telefonsko omrežje



- Še boljša rešitev:
  - Mrežo si predstavljajmo kot šahovnico
    - Bela in črna polja bomo obravnavali ločeno
  - Zasukajmo vse skupaj za 45 stopinj
    - Vsak oddajnik pokriva en črn in en bel kvadrat
    - Eden je velikosti $r \times r$, eden pa $(r + 1) \times (r + 1)$



  - Za vsako barvo moramo izračunati ploščino unije kvadratov tiste barve
  - To gre s preletom ravnine v $O(k \log k)$
    - Ravnino razdelimo na pasove pri vsaki x-koordinati, kjer se kak kvadrat začne/konča
    - Pregledujemo jih od leve proti desni in sproti vzdržujemo podatke o tem, kateri intervali y-koordinat so pokriti (drevo intervalov)
    - To je bila nekoč naloga za drugo skupino (1998.2.3)

3.3 Transakcijski računi

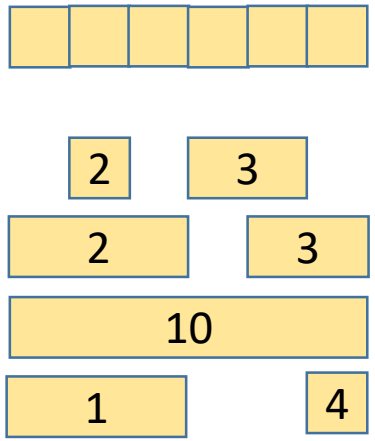
```
5 5
A6666666664
A2140319954
A1000000056
A0050050000
A0050050088
A1000000056 100000
A2140319954 750000
A6666666664 1001
A1000000056 60000
A0050050088 250000
```

- Vhodni podatki:
  - Imamo bazo številk računov, prvi od njih je račun dobrodelne organizacije
    - Zadnja števka v številki računa je kontrolna (= ostanek po deljenju vsote ostalih števk z 10)
  - Imamo seznam nakazil (številka računa + znesek)
    - Seznam ima tudi svojo kontrolno števko (= seštejemo kontrolne števke po vseh nakazilih in obdržimo ostanek po deljenju z 10)
  - Številke računov v nakazilih hočemo spremeniti tako, da bo čim več dobila dobrodelna organizacija
    - Uporabljamo lahko le številke računov iz baze
    - Kontrolna števka seznama mora ostati nespremenjena

3.3 Transakcijski računi

- Rešitev:
  - Vprašanje je le, v *koliko* nakazil v seznamu lahko vpišemo račun dobrodelne organizacije
    - Ne pa, v *katera* nakazila — to je trivialno: na tista z največjimi zneski
  - Pomembna je le zadnja števka vsakega računa, ostalo lahko ignoriramo
    - Naj bo c zadnja števka pri računu dobrodelne organizacije
    - Naj bo $D \subseteq \{0, 1, \ldots, 9\}$ množica zadnjih števk pri ostalih računih iz baze
    - Naj bo g kontrolna števka celotnega seznama (ki mora ostati nespremenjena)
  - Recimo, da pri k od m nakazilih uporabimo račun dobrodelne organizacije
    - To h kontrolni vsoti prispeva $(k \cdot c) \bmod 10$
    - Ali je mogoče s seštevanjem $m - k$ števk iz D dobiti vsoto $(g - k \cdot c) \bmod 10$?
  - Naj bo D_t množica vsot, ki jih je mogoče dobiti s t členi iz D
    - $D_0 = \{0\}$, $D_t = \{a + b : a \in D_{t-1}, b \in D\}$
    - To pregledujemo po naraščajočem t in vsakič preverimo, če je v D_t kakšna števka, ki skupaj z $(m - t) \cdot c$ dá vsoto g (po modulu 10)

3.4 Nadzor



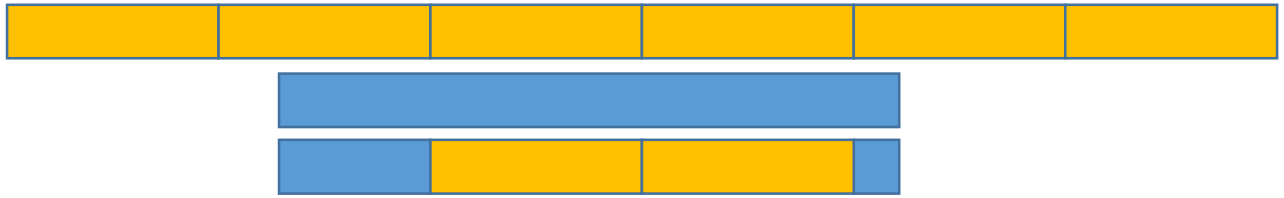
6 7
3 5 3
1 2 2
0 3 2
4 6 3
0 6 10
5 6 4
0 3 1

- Naloga:
  - Dana je ulica, ki je daljica [0, *d*]
  - Danih je *n* kamer — kamera *i* za ceno c_i pokrije interval [a_i , b_i]
  - Iščemo najcenejši nabor kamer, ki skupaj pokrijejo celotno ulico
  - Smejo se tudi prekrivati

3.4 Nadzor

- Rešitev: dinamično programiranje
  - Uredimo kamere naraščajoče po desnem krajišču (b_i)
  - Zadnja kamera v našem izboru mora biti ena od tistih, ki imajo $b_i = d$
  - Ona pokrije $[a_i, b_i] = [a_i, d]$; pred njo moramo pokriti še $[0, a_i]$
  - Če bi uporabili same take kamere, ki imajo $b_j < a_i$, se ne bi izšlo
  - Predzadnja kamera mora biti torej ena od tistih, ki imajo desno krajišče na $[a_i, b_i]$
  - Naj bo $f(i)$ = najcenejši nabor kamer, ki uporabi kamero i, ne uporabi kamer $i + 1, \ldots, n$ in v celoti pokrije $[0, b_i]$
    - Naloga potem sprašuje po $\min_i \{ f(i) : b_i = d \}$.
  - Funkcijo f lahko računamo naraščajoče po i: $f(i) = c_i + \min_j \{ f(j) : j < i, b_j \geq a_i \}$.
  - Preprosta implementacija: dve gnezdeni zanki po i in $j \rightarrow O(n^2)$

3.4 Nadzor



- Boljša implementacija: $f(i) = c_i + \min_j \{ f(j) : j < i, b_j \geq a_i \}$
  - Najmanjši primerni j poiščimo z bisekcijo
    - Iščemo, kam bi prišel a_i v urejenem seznamu desnih krajišč) $\rightarrow O(\log n)$
  - Za hitro računanje minimuma $f(j), \ldots, f(i-1)$ moramo nekako hraniti minimume po več zaporednih vrednosti funkcije f
  - Preprosta rešitev: razdelimo tabelo $f[1..n]$ na $\sqrt{n}$ blokov s po $\sqrt{n}$ elementi
    - Za vsak blok hranimo njegov minimum
    - Za računanje min $f(j), \ldots, f(i-1)$ moramo pregledati le dva bloka po elementih, za vmesne bloke uporabimo shranjeni minimum $\rightarrow O(\sqrt{n})$ pri vsakem i, skupaj $O(n \sqrt{n})$
  - Boljša rešitev: binarno drevo ali Fenwickovo drevo $\rightarrow O(\log n)$ pri vsakem i, skupaj $O(n \sqrt{n})$

3.5 Detektorji

- Naloga:
  - Danih je *n* trezorjev in *m* hodnikov (neusmerjenih povezav) med njimi
  - Danih je d detektorjev — trezor *i* pokrivajo detektorji od s_i do e_i
    - V trezorju je mogoče lopov, če vsaj polovica teh detektorjev zaznava gibanje
  - Podano je stanje detektorjev v *q* časovnih korakih
    - V bistvu so podane spremembe stanja v posameznem koraku glede na prejšnji korak
  - V posameznem koraku je lahko lopov v istem trezorju kot v prejšnjem ali pa v enem od sosednjih trezorjev
  - Če je v enem koraku v trezorju *i*, nakrade w_i enot denarja
  - Izračunaj največjo možno skupno nakradeno vsoto

3.5 Detektorji

- Rešitev: dinamično programiranje
  - Naj bo $f_t(v)$ največji znesek, ki ga lahko nakrade v prvih t korakih, če korak t preživi v trezorju v
  - Potem je:
    - $f_t(v) = w_v + \max\{ f_{t-1}(u) : u = v$ ali pa obstaja hodnik med u in $v \}$
    - $f_t(v) = -\infty$, če manj kot polovica trezorjev $s_v, \ldots, e_v$ ob času t zaznava prisotnost
  - Naloga spraša po $\max_v f_q(v)$
  - To lahko računamo naraščajoče po t
    - Pri vsakem t gremo z zanko po u in če je $f_{t-1}(u) > -\infty$, pojdimo z gnezdeno zanko po u-jevih sosedih v
    - Nato gremo pri tem t še z zanko po v in preverjamo stanje detektorjev
  - Kako prešteti, koliko detektorjev izmed $s_v, \ldots, e_v$ zaznava prisotnost?
    - Z zanko po detektorjih je prepočasi
    - Bolje: pri vsakem t najprej izračunamo tabelo delnih vsot:
      $a[i] = 0$ ali 1, ki pove, ali detektor i trenutno zaznava prisotnost
      $b[i]$ = število detektorjev izmed 1, …, i, ki trenutno zaznavajo prisotnost
      $b[0] = 0$, nato $b[i] = b[i-1] + a[i]$
    - Za trezor v preverjamo, ali je $b[e_v] - b[s_{v-1}] \geq (e_v - s_v + 1) / 2$

Kolesarji

Off-line naloga

Off-line naloga: Kolesarji

- Nalogo in testne primere smo objavili konec leta 2018
- Tekmovalci so lahko do vključno včeraj oddajali svoje rezultate na naš ocenjevalni strežnik
- 12 tekmovalcev (6 študentov, 5 dijakov)

Naloga

Mesto		1.	2.	3.	4.	5.
Točke		100	80	60	40	30
Izidi posameznih dirk	Prva	5	3	2	4	1
	Druga	3	4	5	1	2
	Tretja	1	2	5	3	4

- Danih je *n* kolesarjev, nastopili so na *d* dirkah
  - Za vsako dirko je znan njihov vrstni red
  - Za vsak položaj v vrstnem redu je znano, koliko točk je vreden
- Sestaviti želimo navidezno ekipo, ki jo sestavlja *k* kolesarjev
  - Po vsaki dirki jih lahko največ m zamenjamo
  - Maksimiziramo skupno število točk, ki jih dobijo kolesarji v naši ekipi
- Primer: $k = 3$, $m = 1$
  - Ves čas {1, 2, 3}: (30 + 60 + 80) + (40 + 30 + 100) + (100 + 80 + 40) = 560
  - {1, 2, 4}, {1, 2, 4}, {2, 3, 4}: (30 + 60 + 40) + (40 + 30 + 80) + (80 + 40 + 30) = 430
  - {2, 3, 5}, {1, 3, 5}, {1, 2, 5}: (60 + 80 + 100) + (40 + 100 + 60) + (100 + 80 + 60) = 680
- 27 testnih primerov
  - Do 1000 dirk, 1000 kolesarjev, $k \leq 300$
  - Večina je bila manjših

Dinamično programiranje

- Za majhne n in k: optimalna rešitev z dinamičnim programiranjem
  - Možnih je $n(n-1)\ldots(n-k+1)/k!$ izborov k tekmovalcev izmed n tekmovalcev
  - Naj bo $f(t, A)$ = najboljša rešitev za prvih t dirk, ki v zadnji od njih uporabi ekipo A
  - To računamo po naraščajočih t:
    $f(t, A) = točke(t, A) + \max\{ f(t-1, B) : B$ in A se razlikujeta v $\leq m$ kolesarjih $\}$
    - Pri $t = 0$ namesto max{…} vzamemo 0
  - Na koncu nas zanima $\max\{ f(d, A) : A$ ima k kolesarjev $\}$.

Požrešni algoritem

- Požrešna strategija:
  - Pred vsako dirko vržemo iz ekipe tistih *m* kolesarjev, ki bodo v njej dobili najmanj točk (izmed vseh v naši ekipi)
  - Nato vzamemo v ekipo tistih *m* kolesarjev, ki bodo v tej dirki dobili največ točk (izmed vseh, ki jih zdaj ni v naši ekipi)
- Malo manj kratkovidna požrešna strategija:
  - $ocena(x, t) = točke(x, t) + c \cdot ocena(x, t + 1)$ za neko konstanto $c < 1$
  - Ko se odločamo, koga bi vrgli iz ekipe in vzeli vanjo, gledamo $ocena(x, t)$ namesto $točke(x, t)$
  - Preizkusimo različne konstante *c*
    - Mogoče različne c za različne dneve?

Celoštevilsko linearno programiranje

- Naj bo $a_{xt} \in \{0, 1\}$ število, ki pove, ali je kolesar x na dirki t v ekipi
- Maksimiziramo $\Sigma_x \Sigma_t$ *točke*(x, t) a_{xt}
- Omejitve:
  - Za vsak x in vsak t: $0 \leq a_{xt} \leq 1$
  - Za vsak t: $\Sigma_x a_{xt} = k$ (velikost ekipe)
  - Za vsak x in vsak t: $d_{xt} \geq 0$ in $d_{xt} \geq a_{xt} - a_{x,t-1}$
    - Ideja je, da bo $d_{xt} \geq 1$, če je kolesar x v dirki t vstopil v ekipo, sicer $d_{xt} = 0$
  - Za vsak t: $\Sigma_x d_{xt} \leq m$ (največ m menjav)
- V splošnem je celoštevilsko linearno programiranje NP-težak problem, vendar v praksi pogosto dobimo rešitve sprejemljivo hitro