Sisukord

1. Pakirobot Manhattanis (`robot`)

1 sek / 2 sek 20 punkti

Idee: Ago Luberg, teostus: Peeter Aleksander Randla, lahenduse selgitus: Tähvend Uustalu

> Tasandil liigub robot. Robot saab liikumiseks teha ühe ühiku pikkuseid samme põhja, lõunasse, läände ja itta. Robot alustas oma laost ja tegi N sammu; sisendis on antud nende nimekiri. Mitu sammu on minimaalselt vaja teha, et robot lattu tagasi jõuaks?
>
> $1 \leqslant N \leqslant 1\,000$.

See on peamiselt implementeerimisülesanne, seega kirjeldame lahendust paralleelselt implementatsiooniga. Kõigepealt loeme sisse sisendi:

```
n = int(input())
steps = input()
```

Olgu roboti alguspunktiks $(0; 0)$. Käime läbi kõik roboti sammud ja peame kogu aja meeles roboti asukohta. Näiteks kui robot teeb sammu "`N`", suurendame y-koordinaati ühe võrra jne.

```
x = 0
y = 0
for step in steps:
    if step == 'N':
        y += 1
    elif step == 'E':
        x += 1
    elif step == 'S':
        y -= 1
    elif step == 'W':
        x -= 1
```

Nüüd oleme punktis $(x; y)$. Kui mitu sammu on vaja, et sealt tagasi alguspunkti jõuda? Iga samm saab muuta ainult üht koordinaati, ja täpselt ühe võrra. On selge, et meil on mõistlik teha $|x|$ sammu x-telje suunas ja $|y|$ sammu y-telje suunas, kus $|\cdot|$ tähistab absoluutväärtust. Näiteks kui oleme punktis $(5; 1)$ peame tegema 5 sammu vasakule ja 1 sammu alla. Kui aga oleme punktis $(-1; 3)$ peame tegema 1 sammu paremale ja 3 sammu alla. Niisiis peame kokku tegema $|x| + |y|$ sammu.

```
print(abs(x) + abs(y))
```

2. Pliiatsite pööramine (`pliiatsid`)

1 sek / 2 sek 30 punkti

Idee: Kobras ja Rein Prank, teostus ja lahenduse selgitus: Peeter Aleksander Randla

> On antud N pliiatsist kosnev rida. Igal pliiatsil on üleval kas terav (`t`) või nüri (`n`) ots. Üks pööre vahetab mingis lõigus kõigi pliiatsite suunad (iga `t` muutub `n`-iks ja vastupidi). Leia lühim pöörete jada, millega saab kõik pliiatsid samapidi.
>
> $1 \leqslant N \leqslant 1\,000$.

Selles ülesandes on optimaalne tõenäoliselt kõige esimene idee, mis pähe tulla võiks: leia mingisugune rida kõrvuti olevad pliiatseid, mis kõik on valepidi, pööra need ümber, ja korda seda seni, kuni kõik pliiatsid on samapidi.

Selle optimaalsuse tõestamiseks uurima, kui mitu korda on kaks real kõrvuti olevat pliiatsit eripidi. Olgu see arv F. Kui kõik pliiatsid on samapidi, siis $F = 0$, kui mitte, siis $F > 0$. Seega vaja on F minimeerida. Üks pööre saab vähendada F-i mitte rohkem kui kahe võrra: pöörde sees jäävad erinevused alles, seega F saab muutuda ainult pöörde otspunktide arvelt, ja ühel pöördel on kaks otspunkti. Samas aga varem mainitud strateegia vähendabki iga kord F-i kahe võrra: nii algus- kui ka lõpp-punkt on pärast pööret samapidi kui nende naabrid, seega erinevuste arv väheneb 2 võrra. Erandjuhuks on viimane pööre, kus võib alles jääda ainult üks erinevus. Siis tuleb pöörata kogu lõik erinevusest kuni ühe otspunktini.

Üks probleem, mis selle strateegiaga tekkida võib, on see, kummale poole pliiatseid pöörama peab, s.t. kumb suund on õiget- ja kumb valepidi. Selgub, et piisab, kui õigeks suunaks võtta esimese pliiatsi suund: kui esimene ja viimane pliiats on samapidi, siis peab kindlasti pöörama teised pliiatsid ka nendega samapidi, teisele poole pööramine võtaks ühe lisapöörde; kui esimene ja viimane pliiats on eripidi, siis võib pöörata ükskõik kummale poole, kusjuures tuleb pöörata ka üks jada otspunktidest. Mõlemal juhul on esimese pliiatsi suund lubatud.

Võib tunduda, et suunaks tuleks võtta see, mispidi pliiatseid on algseisus rohkem. See on aga vale: pööramisel ei ole oluline, kui palju pliiatseid järjest samapidi on, neid saab ikka ühe pöördega pöörata.

Kokkuvõttes on ülesande lahendav algoritm selline: leia esimene pliiats, mis ei ole kõige esimesega samapidi. Leia, kui mitu pliiatsit sellest edasi sellega samapidi on. Väljasta sellele lõigule vastav pööre ja vaata sellest punktist alates edasi.

3. Tsirkus (`tsirkus`)

1 sek / 2 sek 40 punkti

Idee: Kobras ja Rein Prank, teostus ja lahenduse selgitus: Marko Tsengov

> On antud N (või ka $N + 5$) tipuga graaf. M tipu puhul on suunatud serv kaaluga 0 mõnda teise tippu, ülejäänud tippude $i < N$ puhul serv kaaluga 1 tippudesse $i + 1$, $i + 2$, ..., $i + 6$. Leia lühim teekond tipust 1 tippu $x \geqslant N$.

Tähelepanek 1. *Kui mingi tee lõppeb ruudul $x > N$, siis leidub sama pikk tee, mis lõppeb ruudul N.*

See tähelepanek pole rangelt vajalik ülesande lahendamiseks, kuid lihtsustab pisut lõpuloogikat, kuna siis tuleb vaadelda ainult ruudule N jõudmist.

Alamülesanne 1

Selles alamülesandes on kõik ruudud samasugused. Lühim tee on võimalik saavutada, veeretades 6 kuni nupp on ruudul N või ületaks seda. Selleks kulub $\lceil \frac{N-1}{6} \rceil$ täringuviset. Nii täringuvisete arvu kui ka teekonna saab leida simuleerimise teel (teha käike, kuni nupp lõppu jõuab).

Alamülesanne 2

Selles alamülesandes on täpselt üks madu või redel. Kui sisendis on antud madu, leidub kindlasti lühim teekond, mis ei pea mao teise otsa liikuma. Seega saab kasutada eelmise alapunkti lahendust, veeretades vajadusel ühel korral 5, et vältida mao algusruudule jäämist.

Kui sisendis on antud redel, võib redeli läbimine olla nii kasulik kui ka kahjulik, sest redeli algusruudule jõudmise asemel võib mõnel juhul sama täringuviskega jõuda redeli lõpuruudust kaugemale. Seega saab üritada ülesannet lahendada kaks korda: üks kord redelit ignoreerides ning teisel korral kindlasti redelit kasutades, seejärel valides neist lühima tee. Kui redelit ignoreerides satub nupp endiselt redeli algusruudule ning selline teekond on lühim, on lahendus endiselt sobiv, isegi redelit arvesse võtmata.

Alamülesanne 3

Selles alamülesandes esinevad ainult maod. Seega on endiselt optimaalne liikuda võimalikult kaugele edasi, kuid mitte maanduda mao algusruudule. Kui järjest on 6 või enama mao algusruudud, pole võimalik nendest mööda pääseda.

Alamülesanne 4

Selles alamülesandes esinevad ainult mittelõikuvad redelid. Optimaalne on liikuda edasi täringuvisetega 6, kuid peatudes varem selliste redelite algusruutudel, mis viivad nupu kaugemale, kui täringuvise 6 nupu algselt ruudult.

Keerukus senistel juhtudel: $\mathcal{O}(N)$ (siin ja edaspidi võib-olla ka $\log N$ tegur, kui kasutada logaritmkeerukusega sõnastikku (näiteks C++ `std::map`))

Alamülesanne 5

Tähelepanek 2. *Kui mingi ruut on lühimal teekonnal ruuduni N, siis peab kasutatav teekond selle ruuduni olema lühim võimalik.*

Eelmises alamülesandes toodud naiivne ahne algoritm ei tööta, kui redelid võivad lõikuda. Selle asemel saab iga ruudu kohta hoida meeles lühimat teekonda selleni (vastavaid täringuviskeid). Kui mingilt ruudult i on võimalik ühe täringuviskega jõuda ruudule j, saab ruudu j lühimat teekonda vajadusel uuendada. Et kõik servad viivad ainult edasi, saab ruudud vaadata üle järjekorras 1 kuni $N-1$. Tippu N salvestatud lühim teekond ongi sel juhul otsitav lühim teekond.

Keerukus: teekonna pikkus $\mathcal{O}(N)$, teekond $\mathcal{O}(N^2)$.

Alamülesanne 6

Eelnevas alamülesandes toodud lahendus hoidis iga tipu jaoks mälus kogu lühimat teekonda, mis tuleb iga järgmise tipu jaoks kopeerida. Tegelikult piisab meeles hoida iga tipu kohta vaid lühima teekonna pikkust, vastavat eelmist tippu ning vastavat täringuviset. Kui kõik tipud 1 kuni $N-1$ on alamülesande 5 lahenduse põhjal külastatud, saab teekonna taastada, liikudes järjest eelmisesse tippu teekonnal ning jättes meelde vajaliku täringuviske. Lõpliku teekonna saab konstrueerida, kui saadud täringuvisete jada ümber pöörata.

Keerukus: $\mathcal{O}(N)$.

Alamülesanded 7 ja 8

Need on ainsad alamülesanded, kus sisendis võib esineda madusid ning redeleid korraga. Ehkki eelnevatel juhtudel oli alati parimas lahenduses võimalik madu vältida, on sellisel juhul mõnikord mao kasutamine oluliselt kasulikum (näiteks läbida redeliga $\frac{2}{3}$ teest, siis liikuda maoga tagasi $\frac{1}{3}$ tee peale ning sealt redeliga lõppu). Seega mitmed eelnevad ahned lähenemised enam ei tööta ning vaja on üldisemat algoritmi.

Antud juhul on sobivaks algoritmiks laiuti läbimine: jagame graafi tipud „kihtidesse“ selle põhjal, mis on lühima teekonna pikkus vastavasse tippu. Esimene kiht (teekonna pikkusega 0) on tipp 1. Iga järgmise kihi tipud saab leida, võttes kõik tipud, kuhu saab jõuda eelmise kihi tippudest ning mis pole juba mõnes teises kihis. Kui mõnesse kihti lisatakse tipp N, ongi teekond leitud, ning kihtide omaduse tõttu peab see olema lühim tee tippu N. Kihtide vahel liikudes saab alamülesannete 5 ja 6 ideede põhjal meelde jätta senise teekonna.

Keerukus on vastavalt kasutatud ideele kas sama kui alamülesandes 5 või sama kui alamülesandes 6, lahendades vastavalt alamülesanded 7 ning 8.

4. Bititehete avaldis (`avaldis`)

2 sek / 15 sek 60 punkti

Idee ja lahenduse selgitus: Tähvend Uustalu, teostus: Olivia Tennisberg

> On antud $N+1$ täisarvust ja N tehtemärgist koosnev avaldis, kus kasutusel on tehtemärgid `AND`, `OR` ja `XOR`. Kõik tehted tehakse vasakult paremale. Esitatakse Q päringut kujul "asenda antud positsioonil olev tehtemärk ja arv uue etteantud tehtemärgi ja arvuga ning arvuta avaldise väärtus".
>
> $1 \leqslant N, Q \leqslant 10^5$; arvud avaldises on ülimalt 10^9.

Kõikides kasutusel olevates tehetes on bitid sõltumatud: kui näiteks arvutada arv a `XOR` b, siis tulemuse kaheksalised ei sõltu mitte millestki muust kui ainult a kaheksalistest ja b kaheksalistest. See tähendab, et võime lahendada ülesannet "iga biti kohta eraldi". Kirjutame lahenduse, mis eeldab, et kõik arvud on nullid ja ühed ning jooksutame seda üheliste, kaheliste, neljaliste jne. kohta eraldi. Et avaldises olevates arvudes on ülimalt 30 bitti tähendab see, et lahendus läheb 30 korda aeglasemaks. Kogu järgnevas arutelus eeldame niisiis, et kõik arvud on nullid ja ühed.

Lahendus 1

Paneme tähele, et suvalise $a \in \{0, 1\}$ korral:

$$a \texttt{ AND } 0 = 0 \qquad a \texttt{ OR } 0 = a$$
$$a \texttt{ AND } 1 = a \qquad a \texttt{ OR } 1 = 1$$

See tähendab, et:

- Kui avaldises on kuskil jupp "`AND` 1" või "`OR` 0", siis seda juppi võib täielikult ignoreerida.
- Kui avaldises on kuskil jupp "`AND` 0", siis kõike enne seda võib ignoreerida ning eeldada, et avaldis algab sellest punktist ja arvuga 0.
- Kui avaldises on kuskil jupp "`OR` 1", siis kõike enne seda võib ignoreerida ning eeldada, et avaldis algab sellest punktist ja arvuga 1.

Igale päringule vastamine võiks siis näha välja midagi sellist:

- Leiame parempoolseima "`AND` 0" või "`OR` 1". Kui seda ei leidu, siis arvestame, et selleks on avaldise päris alguses olev 0.
- Leiame alates sellest punktist tulevate arvude `XOR`, kus erandlikult arvestame, et `AND` 1 loeb nullina.
  - See töötab, sest sellest punktist edasi on kõik jupid kujul "`AND` 1", "`OR` 0" ja "`XOR` a", ning a `XOR` $0 = a$.

Mõlemad operatsioonid peaksid toimima $\mathcal{O}(\log N)$ või sarnases ajas.

Parempoolseima "`AND` 0" või "`OR` 1" leidmine

Kõige lihtsam viis selleks on kasutada C++ standardteegi andmestruktuuri `std::set<int>`. See simuleerib arvuhulka (s.t. iga arv saab esineda seal ülimalt ühe korra), kusjuures toetab järgmiseid operatsioone:

- `my_set.insert(x)`: arv x lisatakse hulka `my_set` $\mathcal{O}(\log M)$ ajas.
- `my_set.erase(x)`: arv x kustutatakse hulgast `my_set` $\mathcal{O}(\log M)$ ajas.
- `*my_set.rbegin()`: leitakse ja tagastatakse hulga `my_set` suurim element $\mathcal{O}(1)$ ajas.

Siin M on hulga suurus.

Ülesande lahendamiseks haldame `set<int>`, mis peab meeles kõikide "`AND` 0" ja "`OR` 1" indekseid. Kui muudame avaldise mingit juppi ja see on nüüd "`AND` 0" või "`OR` 1", siis lisame selle hulka. Kui enne oli, aga nüüd ei ole, siis kustutame selle hulgast. Päringute vastamiseks leiame hulga suurima elemendi.

Alates mingist punktist tulevate arvude `XOR` leidmine

Siin peame sisuliselt lahendama järgmist ülesannet:

> Antud massiiv A, mis koosneb nullidest ja ühtedest, ning kaht tüüpi päringud:
>
> 1. antud i ja x, sea $A[i] \leftarrow x$;
> 2. antud i, leia $A[i]$ `XOR` $A[i+1]$ `XOR` $A[i+2]$ `XOR` $\cdots$ `XOR` $A[n]$.

Selleks on palju variante. Üks võimalus on kasutada Fenwicki puud või lõikude puud, mis lahendavad täpselt selliseid ülesandeid (vt. nt. peatükk 9.3 siit).

Lähenemine, mille jaoks ise kõige vähem koodi kirjutama peab on aga järgmine. Paneme tähele, et sufiksi `XOR` leidmine on sisuliselt sama, mis sufiksis olevate ühtede loendamine. GCC kompilaatoris on olemas üks andmestruktuur, mis ei ole küll ametlikult C++ standardi osa, aga on GCC kompilaatoris siiski kättesaadav.

Kui lisada oma koodi jupp:

```
#include <bits/extc++.h>

template <typename T>
using ordered_set = __gnu_pbds::tree<T, __gnu_pbds::null_type, std::less<T>,
  __gnu_pbds::rb_tree_tag, __gnu_pbds::tree_order_statistics_node_update>;
```

on nüüd kättesaadav andmestruktuur `ordered_set<int>`. See on nagu `std::set<int>`, aga toetab üht täiendavat kasulikku operatsiooni:

- `my_set.order_of_key(x)`: leitakse ja tagastatakse hulgas `my_set` arvust x väiksemate elementide arv $\mathcal{O}(\log M)$ ajas.

Nüüd saame hoida meeles `ordered_set<int>`, mis peab meeles kõikide "`XOR` 1" juppide indekseid. Päringule vastamiseks lahutame hulga suurusest enne etteantud alguspunkti olevate "`XOR` 1" juppide arvu.

Kõik kokku arvutades on keerukus $\mathcal{O}(N \log N \log \max X_i)$.

Lahendus 2

Nagu enne, lahendame ülesannet iga biti kohta eraldi.

Vaatleme ülesande avaldisi funktsioonidena, mille argument on 0 või 1 ja väärtus samuti 0 või 1. Näiteks võime defineerida

$$f(x) = (((x \texttt{ OR } 0) \texttt{ XOR } 0) \texttt{ XOR } 1) \texttt{ AND } 1.$$

Selle funktsiooni saame kokku võtta 2×2 tabelina:

sisse	välja
0	1
1	0

.

Kui defineerime $g(x) = (x \texttt{ OR } 0) \texttt{ XOR } 0$ ja $h(x) = (x \texttt{ XOR } 1) \texttt{ AND } 1$, siis $f(x) = h(g(x))$. Kui meil on teada funktsiooni esitus tabelina, saame ka nende kompositsiooni tabeli $\mathcal{O}(1)$ ajas välja arvutada.

Ehitame lõikude puu, mille igas tipus hoiame meeles mingi funktsiooni esitust 2×2 tabelina. Puu lehtedesse kirjutame avaldise jupid, sisetippudesse kirjutame tema laste esitused funktsioonidena. Päringutele vastamiseks võtame juurtipus oleva funktsiooni ja arvutame selle välja argumendi 0 jaoks.

Keerukus on sama, mis eelmisel.

See lahendus on abstraktsem ja ehk ka tehniliselt keerulisem, kui teine lahendus. Samas on see üldisem ja ei nõua ülesande struktuuri nii sügavat uurimist.

5. Pesunöör (pesu)

2 sek / 30 sek 100 punkti

Idee, teostus ja lahenduse selgitus: Jaagup Tamme

On antud pesunöör, mis koosneb N särgist. Iga särk on kas punane, sinine või roheline. Vasta Q päringule, millest igaüks on ühel allolevatest kujudest:

- Antud l ja r. Leia lõigus $[l, r]$ olevate etteantud värvi särkide arv.
- Antud l ja r. Tõsta lõigus $[l, r]$ olevad särgid nööri algusesse.
- Antud l ja r. Tõsta lõigus $[l, r]$ olevad särgid nööri lõppu.
- Antud l ja r. Pööra ümber pesunööri lõik $[l, r]$.

$1 \leqslant N, Q \leqslant 10^5$.

Alamülesanne 1

Esimese testigrupi saab lahendada sõne peal kõiki kirjeldatud päringuid simuleerides. Sellisel juhul on üheks implementatsiooni detailiks, et enamikus programmeerimiskeeltes algab sõnede indekseerimine nullist, aga ülesandes hakkab ühest.

Üht värvi särkide loendamiseks saab nii Pythonis kui C++'st kasutada standardteegi meetotit `count()`.

Alamülesanne 2

Teise testigrupi lahendamisel on kasulik märgata, et särke on ainult kolme erinevat värvi. Sellest lähtuvalt teeme kolm uut sõne, ühe iga värvi jaoks, kus indeksil i on "1", kui algses sõnes on indeksil i vaadeldav värv, vastasel juhul on indeksil i väärtus "0".

Nüüd peame rakendama kõiki pesunööri muutvaid päringuid kõigile kolmele loodud sõnele. Loendamispäringu vastuseks on vaadeldavas sõnes vastavas vahemikus olev ühtede arv.

Kuna loodud sõnedes on ainult nullid ja ühed, siis võime neid esitada samahästi täisarvude massiividena. Muutmispäringud jäävad nii samaks, kuid loendamispäringute vastuseks võime leida vaadeldava massiivi vaadeldava osa elementide summa.

Siiski ei tohiks eelnevalt kirjeldatud lahendus teist testgruppi ajalimiidi sees ära lahendada. Lahenduse kiiremaks muutmiseks võime arvutada iga massiivi jaoks välja nn prefiksisummamassiivi, mille indeksil i on algse massiivi esimese i väärtuse summa. Et teises testigrupis pole pesunööri muutvaid päringuid, ei pea neid massiive päringutele vastamise ajal uuesti ehitama. Nüüd peame loendamispäringule piiridega $[l, r]$ vastamiseks lahutama vastava prefiksisummamassiivi indeksil r olevast väärtusest vastava prefiksisummamassiivi indeksil $l - 1$ oleva väärtuse. Märkame, et loendamispäringutele vastamine käib sellisel juhul konstantse ajaga, mistõttu selline lahendus mahub ajalimiidist läbi.

Täislahendus

Täislahendust on võimalik kirjutada mitut moodi. Oluline on vaid, et kõiki päringuid täidetaks kiiremini võrreldes naiivse lahendusega.

Lahendus 1: ruutjuureliste pikkustega alampesunöörid

Üks võimalik täislahenduse idee on jagada kogu pesunöör umbes võrdse pikkusega alampesunöörideks, kusjuures alampesunööride pikkus võiks olla umbes $\sqrt{N}$. Järjestikuseid alampesunööre võib hoida näiteks listina. Iga alampesunööri kohta hoiame meeles selle kõik särgid, selle pikkuse, iga värvi jaoks seda värvi särkide arvu ja tõeväärtuse, kas peame alampesunööri enne töötlemist ümber pöörama.

Loendamispäringu vastuse leidmiseks käime kõik alampesunöörid läbi.

- Kui alampesunöör ei lõiku päringuga, siis ei tee me midagi.
- Kui alampesunöör lõikub osaliselt päringuga, siis loeme naiivselt kokku, mitu sobivat särki on otsitavas vahemikus.
- Kui alampesunöör lõikub täielikult päringuga, liidame tulemusele juurde vastava särgi esinemiste arvu seal (salvestasime selle eraldi muutujana).

Ette ja taha liigutamise päringute korral käitume sarnaselt, vaatame endiselt kõik alampesunöörid läbi.

- Kui alampesunöör ei lõiku päringuga, siis ei tee me midagi.
- Kui alampesunöör lõikub täielikult päringuga, siis tõstame ta tervenisti õigesse kohta alampesunööride listis.
- Kui alampesunöör lõikub osaliselt päringuga, siis jagame ta omakorda osadeks nii et iga tekkinud alapesunöör, kas ei lõiku päringuga või on tervenisti päringu sees ja nende osadega siis, vastavalt ei tee midagi või tõstame õigesse kohta alampesunööride listis.

Ka ümberpööramispäringutega saame samamoodi iga alampesunööri järjest töödelda.

- Kui alampesunöör ei lõiku päringuga, siis ei tee me midagi.
- Kui alampesunöör lõikub täielikult päringuga, siis tõstame ta tervenisti õigesse kohta alampesunööride listis.
- Kui alampesunöör lõikub osaliselt päringuga, siis jagame ta omakorda osadeks nii et iga tekkinud alapesunöör, kas ei lõiku päringuga või on tervenisti päringu sees ja nende osadega siis, vastavalt ei tee midagi või tõstame õigesse kohta alampesunööride listis.

Lisaks peame kõigi päringusse kuuluvate alampesunööride ümberpööramisbitti vahetama.

Oluline on meeles pidada, et kui hakkame alampesunööri üksikuid elemente vaatama, siis peame vastavalt ümberpööramisbitile vajadusel särkide järjestuse ümber pöörama ja biti nulliks seadma.

Kui mingil hetkel tekib alampesunööre liiga palju, siis muutub lahendus aeglaseks. Selle vältimiseks võime mingi aja tagant uuesti kogu pesunööri oleku välja arvutada ja ta uuesti alampesunöörideks jagada.

Selle lahenduse puhul on väga oluline katseliselt leida optimaalne konstant alamesunööri pikkuse jaoks.

Lahendus 2: puhi

Teise lahenduse idee on hoida särke mingi isetasakaalustuva kahendotsingupuu tippudena. See kahendotsingupuu peaks olema selline, mis toetab $\mathcal{O}(\log N)$ ajas puu kaheks jagamist ja $\mathcal{O}(\log N)$

ajas kahe puu üheks ühendamist. Programmeerimisvõistlustel osalejate seas on selle jaoks populaarne kasutada andmestruktuuri *treap*. Nimi "*treap*" on moodustatud sõnade "*tree*" ja "*heap*" kombineerimisel. Eesti keeles on selle andmestruktuuri kohta mõnikord kasutatud sõna *puhi*, mis on samamoodi moodustatud sõnade "puu" ja "kuhi" kombineerimisel.

Puhjas on erinevalt tavalisest kahendpuust igal tipul juhuslikult genereeritud prioriteet. Puhja tippude prioriteedid peavad järgima kuhjareeglit ehk tipu vasaku ja parema alluva prioriteedid ei tohi olla suuremad tipu ende prioriteedist. Puhjadest saab täpsemalt lugeda siit.

Pesunöörist koostame sellise puhja, mille keskjärjestus on võrdne pesunööriga, seda küll ainult alguses. Lisaks jätame iga tipu kohta meelde tema alampuus olevate särkide koguarvu, igat värvi särkide arvu ja ühe biti ümberpööramise kohta. Ka siin kasutame laiska ümberpööramist nagu teises lahenduses.

Loendamispäringute vastuseks tükeldame puhja nii, et alles jäänud ossa kuuluvad täpselt need särgid, mis on päringu piirides. Seejärel väljastame nõutud värvi esinemiste arvu tekkinud puhjas. Lõpuks ühendame vahepeal tekkinud puhjad uuesti kokku.

Muutmispäringute täitmiseks samuti tükeldame puhja vastavalt päringu piiridele. Ette ja tagasi liigutamise päringute täitmiseks peame tekkinud puhjad õiges järjekorras kokku ühendama. Ümberpööramise päringu puhul vahetame juurtipu ümberpööramisbiti väärtust ja ühendame puhjad kokku.

Igal puhjade ühendamise sammul uuendame juurtipus särkide koguarvu ja iga värvi särkide arvu.

Enne puhja tipu töötlemist peame alati kontrollima ümberpööramisbitti. Kui see on sisselülitatud, siis lülitame selle välja, vahetame vasaku ja parema alampuu ning muudame alampuu juurtippude ümberpööramisbitid senistele vastupidisteks.

6. Rikkis teleporter (`teleport`)

2 sek / 10 sek 100 punkti

Idee, teostus ja lahenduse selgitus: Tähvend Uustalu

> On antud N tipu ja M servaga graaf ning parameeter K. Sa saad liikuda mööda graafi servi, milleks kulub 1 tund või kasutada teleporterit, milleks kulub K tundi. Teleporteri kasutamise korral viiakse sind ühtlase jaotusega juhuslikult valitud tippu. Leia iga linna kohta, kui palju aega kulub optimaalse strateegia korral keskmiselt sellest linnast linna 1 minemiseks.
>
> $1 \leqslant N, M, K \leqslant 3 \cdot 10^5$.

Esiteks katsume paremini aru saada optimaalse strateegia struktuurist.

Tähelepanek 3. *Meil ei ole kunagi mõtet liikuda mööda mingit serva ja seejärel teleportida.*

Tõestus. Kui me selle asemel lihtsalt telepordime, kulutame vähem aega ja pärast teleportimist oleme täpselt samamoodi ühtlase jaotusega juhuslikult valitud linnas. □

Tähelepanek 4. *Kui me oleme mingis linnas, siis kõik enne toimunu on ebaoluline. Linnast u linna 1 minemiseks kulub optimaalse strateegia korral keskmiselt sama palju aega sõltumata sellest, kuidas me linna u jõudsime.*

Seega võime eeldada, et meie strateegias on iga linna korral fikseeritud, mida me seal teeme: kas liigume mööda mingit konkreetset serva või telepordime.

Tähistagu $\mathrm{dist}(u)$ linna u kaugust tipust 1. Laiuti läbimise abil on võimalik iga u jaoks arvutada $\mathrm{dist}(u)$; kokku võtab see $\mathcal{O}(N + M)$ aega.

Tähelepanek 5. *Leidub optimaalne strateegia, kus tipus u:*

- *kui* $\mathrm{dist}(u) < D$*, siis liigume mööda lühimat teekonda tippu 1.*
- *kui* $\mathrm{dist}(u) > D$*, siis telepordime.*

Siin D on mingi (hetkel veel tundmatu) parameeter.

Tõestus. Kui oleme otsustanud mingis tipus liikuda mööda servi, siis tähelepaneku 3 tõttu on kindlasti optimaalne alates sealt ainult mööda servi liikuda. On selge, et optimaalne on liikuda siis mööda lühimat teekonda. See tähendab, et kui oleme otsustanud tipust u liikuda mööda servi, siis peame liikuma mööda servi ka kõikidel tippudel, mis on lühimal teekonnal $1 \rightsquigarrow u$.

Tähistagu W nende tippude hulka, kust me ei telepordi (sh. tipp 1). Kui tipust u teleportida, siis telepordime nii kaua, kuni jõuame mingisse hulka W kuuluvasse tippu. Saab näidata, et selleks kulub keskmiselt $\frac{N}{|W|}$ katset. Seega kulub selle strateegia korral tipust u tippu 1 jõudmiseks aega keskmiselt

$$\begin{cases} \frac{1}{|W|}\left(KN + \sum_{w \in W} \mathrm{dist}(w)\right), & \text{kui } u \notin W; \\ \mathrm{dist}(u), & \text{kui } u \in W. \end{cases}$$

Kui meie strateegia korral on võimalik vahetada mingite tippude staatust (kas sealt telepordime või mitte) nii, et mingist tipust läheb vastus väiksemaks, siis ei ole strateegia kindlasti optimaalne. Tähistagu Φ kõikide tippude vastuste summat. Kui seda summat on võimalik vähendada mingite

tippude staatust muutes, siis strateegia ei ole optimaalne, sest kui Φ väheneb, siis peab vähenema ka vähemalt ühe tipu vastus.

Paneme tähele, et

$$\Phi = \frac{N - |W|}{|W|} \left(KN + \sum_{w \in W} \operatorname{dist}(w) \right) + \left(\sum_{w \in W} \operatorname{dist}(w) \right).$$

See summa sõltub ainult hulgast W ja sellesse kuuluvate tippude kauguste summast. Kui $|W|$ on fikseeritud, saab see summa olla optimaalne ainult juhul, kui W koosneb mingist $|W|$ tipule 1 lähimast tipust. □

Nüüd on lahendus sisuliselt käes. Sorteerime graafi tipud kauguse järgi (s.t. eeldame, et tipp 2 on tipule 1 lähim tipp ja tipp N tipust 1 kaugeim). Hoiame jooksvalt meeles arvu

$$\frac{1}{i} \left(KN + \operatorname{dist}(1) + \operatorname{dist}(2) + \cdots + \operatorname{dist}(i) \right).$$

Leiame selle miinimumi D ning positsiooni k, kus miinimum saavutati. Kui tipp esineb sorteeritud järjekorras pärast k, siis väljastame D, vastasel juhul tipu kauguse tipust 1. Lahendus võtab $\mathcal{O}(N \log N + M)$ aega: logaritm tuleb sorteerimisest.

7. Ruudustiku värvimine (`ruudustik`) 100 punkti

Idee ja teostus: Oliver Nisumaa, visualiseerimisrakendus ja lahenduse selgitus: Tähvend Uustalu

Antud on arvud N, P ja Q. Värvi $N \times N$ ruudustiku ruudud võimalikult väikese värvide arvuga nii, et kui kahe punkti kaugus on täpselt $\frac{P}{Q}$, siis nad ei ole sama värvi. Värvitakse ainult ruudustiku punktide sisemused.

$N = 50$, $1 \leqslant P, Q \leqslant 50$.

Testide kokkuvõte

Testi number	P/Q	Parim teadaolev värvide arv
1	$3/2 = 1{,}5$	9
2	$11/5 = 2{,}2$	12
3	$5/2 = 2{,}5$	10
4	$17/6 \approx 2{,}83$	8
5	$22/7 \approx 3{,}14$	8
6	$7/2 = 3{,}5$	9
7	$4/1 = 4$	7
8	$9/2 = 4{,}5$	7
9	$5/1 = 5$	7
10	$11/2 = 5{,}5$	8
11	$6/1 = 6$	7
12	$13/2 = 6{,}5$	7
13	$34/5 = 6{,}8$	7
14	$15/2 = 7{,}5$	7
15	$8/1 = 8$	7
16	$9/1 = 9$	7
17	$10/1 = 10$	7
18	$20/1 = 20$	7
19	$25/1 = 25$	6
20	$31/1 = 31$	5

Tüüpilisemad lahendused

Lahendused võib mõtteliselt jagada kaheks. Ühel pool on konstruktsioonid, mis põhinevad süstemaatilistel mustritel; teisel pool on "läbipaistmatu" optimeerimisalgoritmi genereeritud ruudustikud. Väärib märkimist, et valdavas enamuses testides langeb parim esitatud lahendus esimesse klassi.

Toore jõuga

Kõige ilmsem lähenemine on: proovida läbi kõik kombinatsioonid ja kontrollida, millised nendest vastavad ülesande tingimusele. Loomulikult on selline lähenemine liiga aeglane: 50×50 ruudustikul on 10^{2500} erinevat võimalust valida igale ruudule üks värv kümnest erinevast. Võib tekkida "varakult peatumise" mõte: proovime rekursiivselt ruute värvida ja pöördume tagasi kohe, kui tekib vastuolu ülesande tingimusega. Aga ka see ei päästa. Kombinatsioone on liiga palju ja toore jõuga kaugele ei jõua.

Täiesti mõttetu toore jõud siiski ei ole. Kui P/Q on suhteliselt väike, võib proovida järgmist strateegiat: eeldada, et vastuseks on mingi väikeste mõõtmetega korduv muster ja proovime kõik sellised läbi. Kui 50×50 ruudustiku jaoks on liiga palju variante, siis 5×5 ruudustikuga võib toores jõud varakult peatumist kasutades toimida küll.

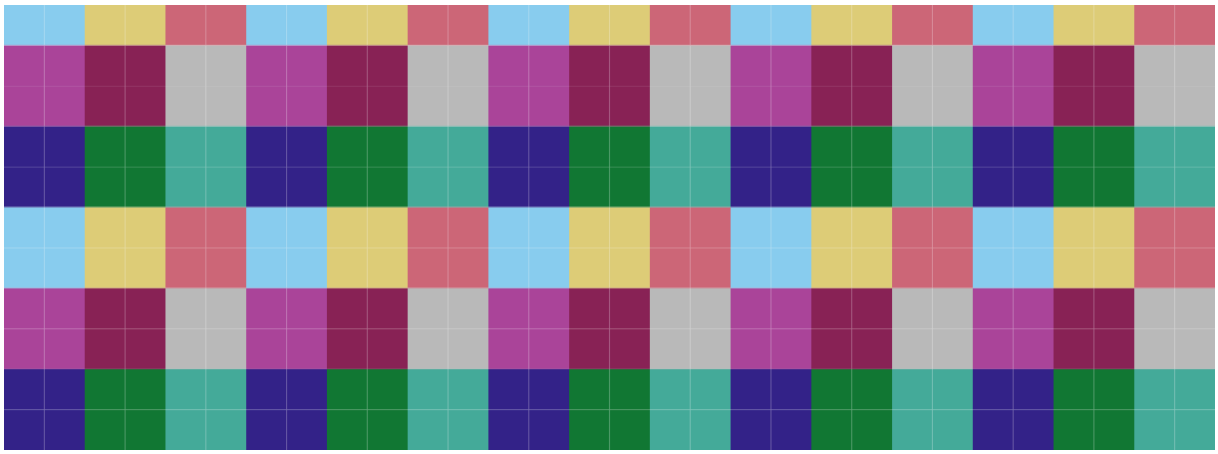
Ahned algoritmid

Teine klassikaline lähenemine on värvida ruudustik ahnelt. Käime kõik ruudustiku ruudud (mingis järjekorras) läbi. Iga ruudu S korral vaatame läbi kõik ruudud, mis on juba värvitud ja sisaldavad ruudust S kaugusel olevat punkti. Seejärel paneme ruudule S mingi värvi, mida tema naabrite seas ei leidunud.

Ahned algoritmid kipuvad joonistama ebamäärase kujuga "mulle", nt. nagu joonistel 19 ja 20 (nende korral ei ole küll žüriile teada, kas kasutati ahnet või mingit muud lähenemist). Sellised lahendused olid sagedasemad võistluse alguses, kuid lõpupoole hakkas laekuma rohkem mustripõhiseid lahendusi.

9 värviga ruutude muster

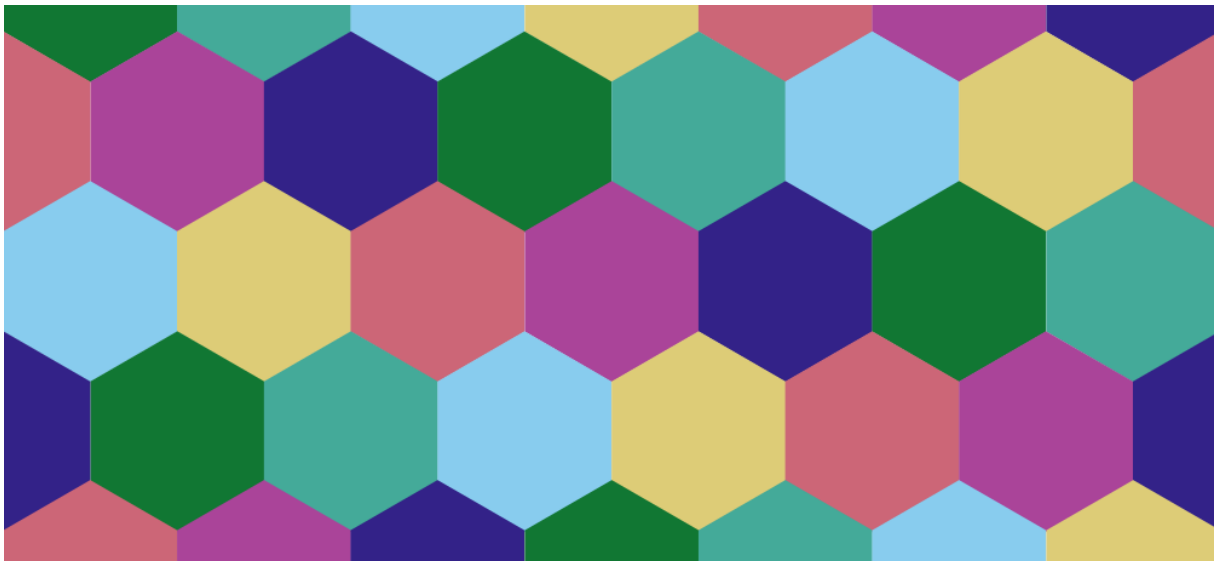
Ehitame lahenduse ruudukujulistest plokkidest. Ploki suuruse valime nii, et ploki diagonaal oleks ülimalt $\frac{P}{Q}$, aga siiski võimalikult suur. Värvime plokid üheksa värviga nii:



Kui ploki diagonaal on arvule $\frac{P}{Q}$ piisavalt lähedal, siis on kaks erinevat sama värvi plokki üksteisest kaugemal, kui $\frac{P}{Q}$. Mõnes testis see aga ei toimi: näiteks testides 2 ja 3 ei ole sobivat ploki suurust võimalik valida.

7 värviga kuusnurkade muster

Vaatleme meekärjekujulist kuusnurkset ruudustikku. Värvime selle "ruudud" seitsme värviga nii:



Täpsemalt: igas reas on kõik värvid esindatud ja alati samas järjekorras. Iga rida on eelmisest 2,5 sammu võrra nihkes. Paneme tähele, et sellise värvimise korral on ühelt kuusnurgalt teisele samasugusele kuusnurgale liikumiseks vaja teha vähemalt 2 sammu.

Niisuguse mustriga tõestatakse, et (klassikalise) Hadwiger-Nelsoni küsimuse vastus on ülimalt 7. Valime mustrile sellise skaala, et sama kuusnurga punktide omavaheline kaugus on alati väiksem kui 1, aga erinevate sama värvi kuusnurkade punktide omavaheline kaugus on alati suurem kui 1. Näeme, et tasandit saab värvida 7 värviga nii, et kaugusel 1 punktid on alati erinevat värvi.

Sama strateegiat võib proovida kasutada ka selles ülesandes. Selleks tuleb kuusnurk asendada mingi kuusnurgale võimalikult lähedase kujundiga nii, et oleks võimalik korrata sama mustrit. Nagu ka eelmise idee puhul ei pruugi see iga $\frac{P}{Q}$ väärtuse korral toimida.

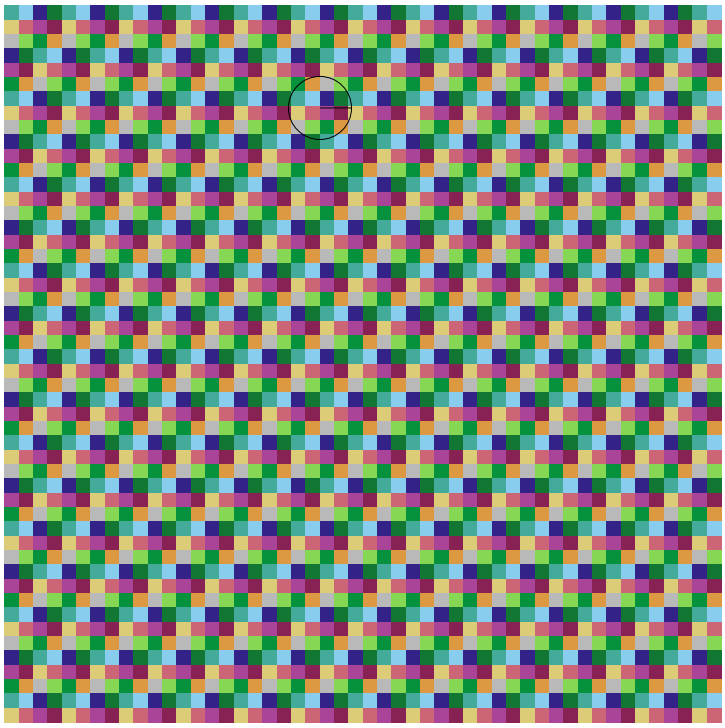
Enamus testide parimad lahendused teevad midagi väga sarnast. Samas ei piisa ülesandes 100 punkti saamiseks pelgalt selle mustri teadmisest. Osa ülesande lahendamisest on välja mõtlemine, millega kuusnurki asendada ja kuidas neid täpselt paigutada. Näiteks ei õnnestunud ühelgi võistlejal konstrueerida sellist lahendust $\frac{P}{Q} = 5{,}5$ jaoks, kuigi teiste lähedaste $\frac{P}{Q}$ väärtuste jaoks on lahendus olemas.

Üks konkreetsem võimalus selle meetodiga ülesannet lahendada on eeldada, et kuusnurkade asemel on ristkülikukujulised "tellised" (vt. nt. jooniseid 7, 9, 14). Proovime toore jõuga läbi kõik telliste mõõtmete ja ridade nihete kombinatsioonid. Iga kombinatsiooni korral kontrollime, kas too annab ülesande tingimusele vastava lahenduse.

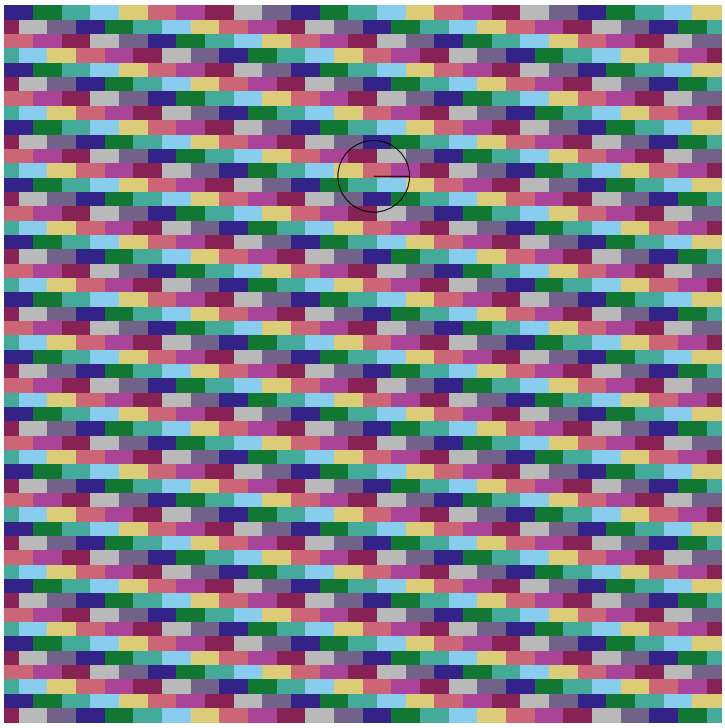
Parimad esitused

Ringid on joonistele joonistatud informatiivselt, et anda aimu arvu $\frac{P}{Q}$ suurusest.

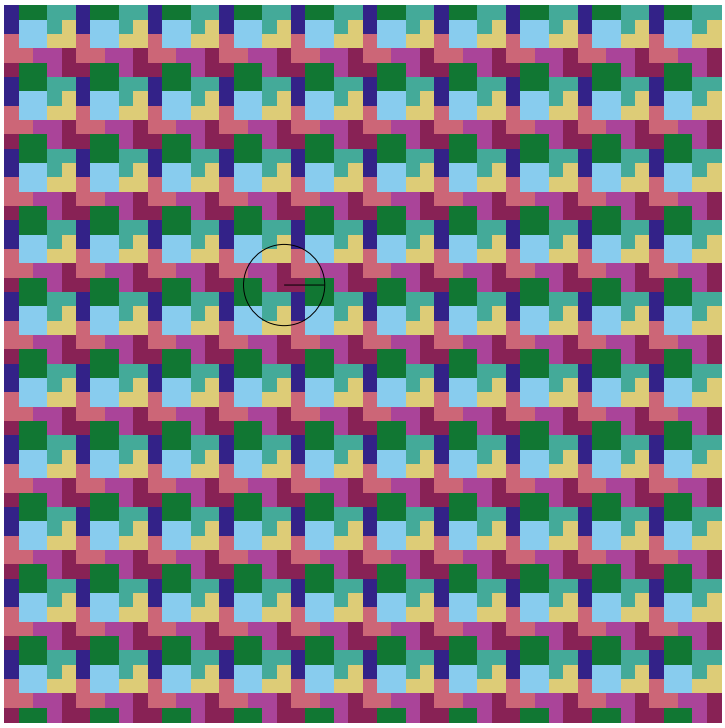
Joonis 1: Test 1, 9 värvi, autor: paljud sõltumatud võistlejad



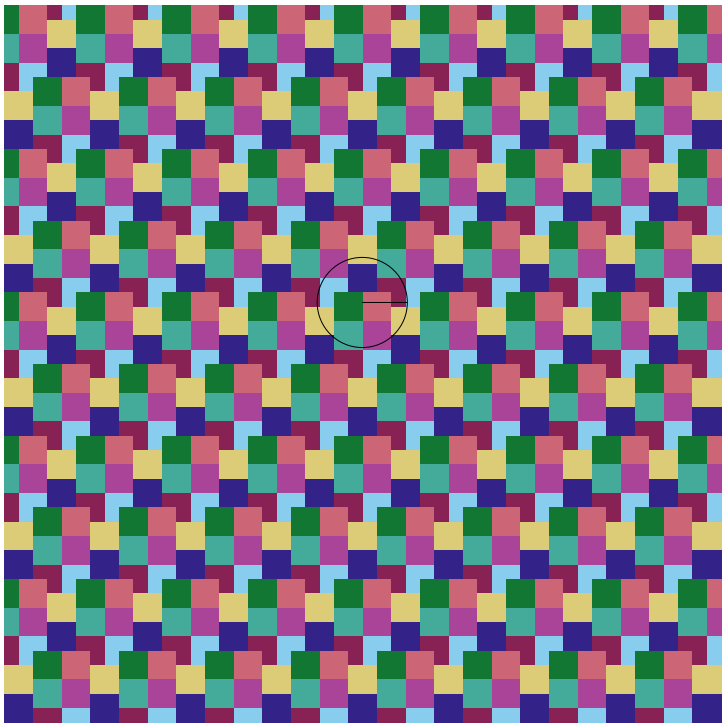
Joonis 2: Test 2, 12 värvi, autor: paljud sõltumatud võistlejad



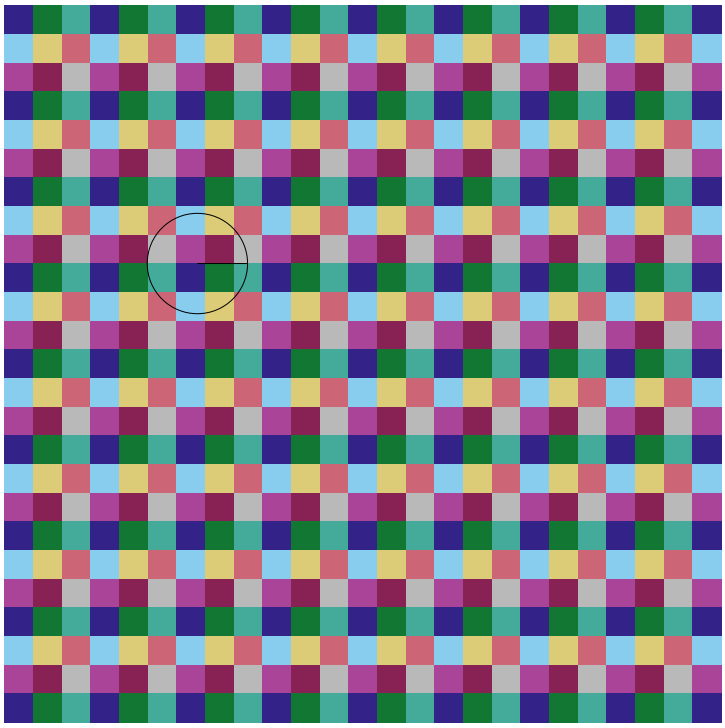
Joonis 3: Test 3, 10 värvi, autor: Benjamin Kleyn



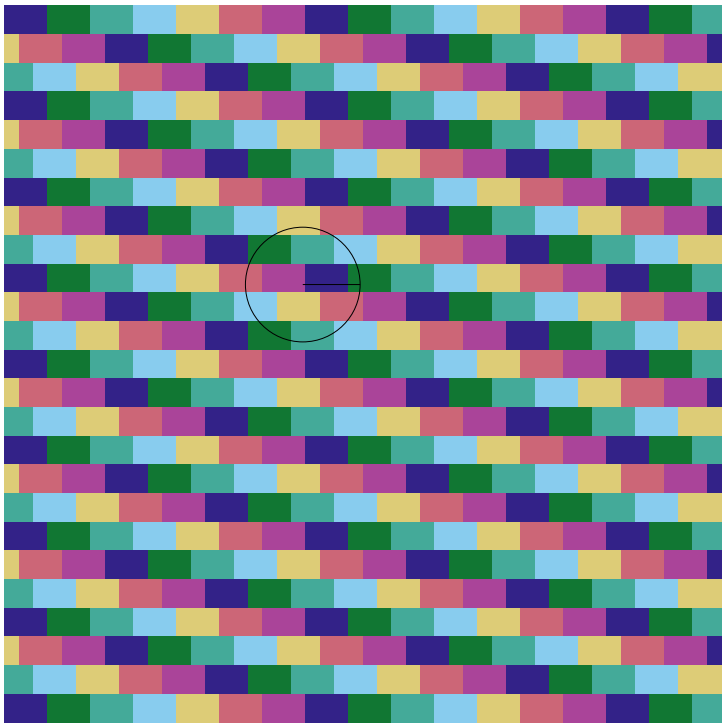
Joonis 4: Test 4, 8 värvi, autor: Birgit Veldi



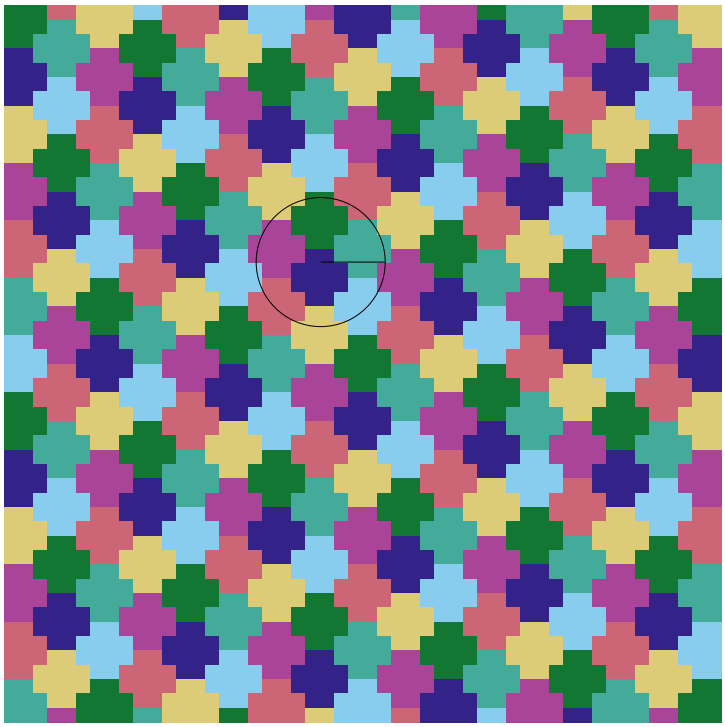
Joonis 5: Test 5, 8 värvi, autor: Oskar Märtin



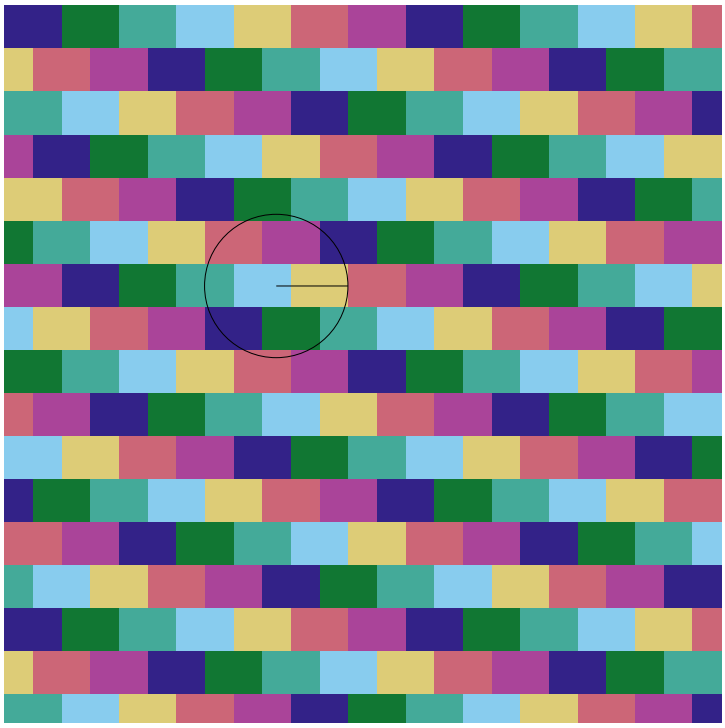
Joonis 6: Test 6, 9 värvi, autor: paljud sõltumatud võistlejad



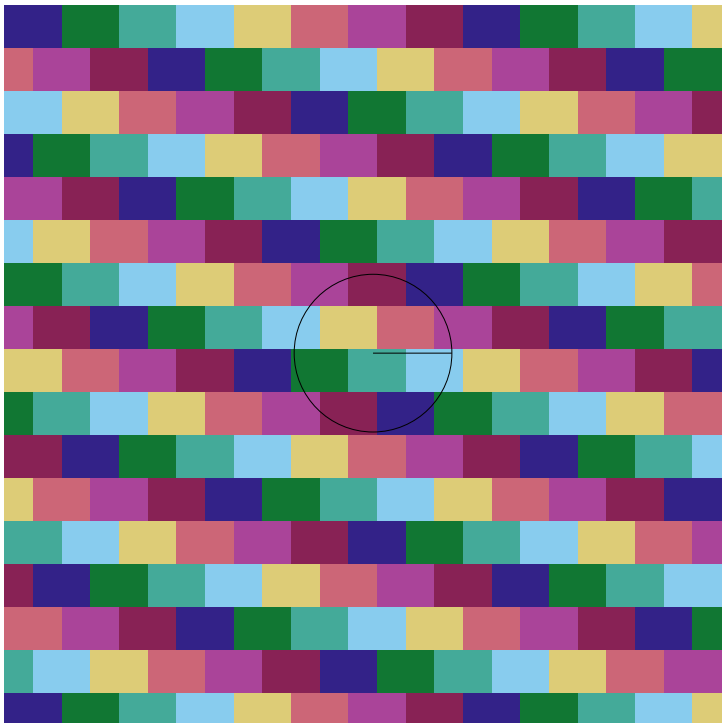
Joonis 7: Test 7, 7 värvi, autor: Benjamin Kleyn



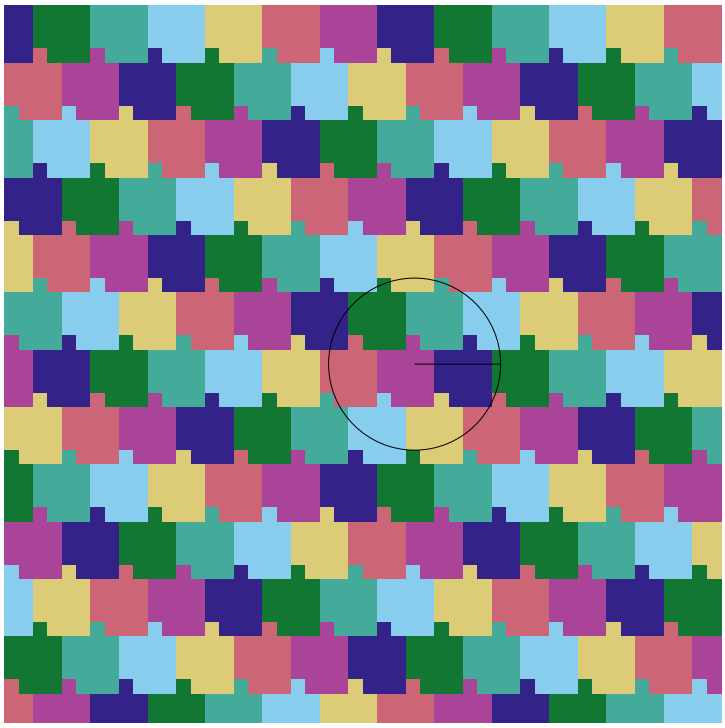
Joonis 8: Test 8, 7 värvi, autor: Minkyum Kim



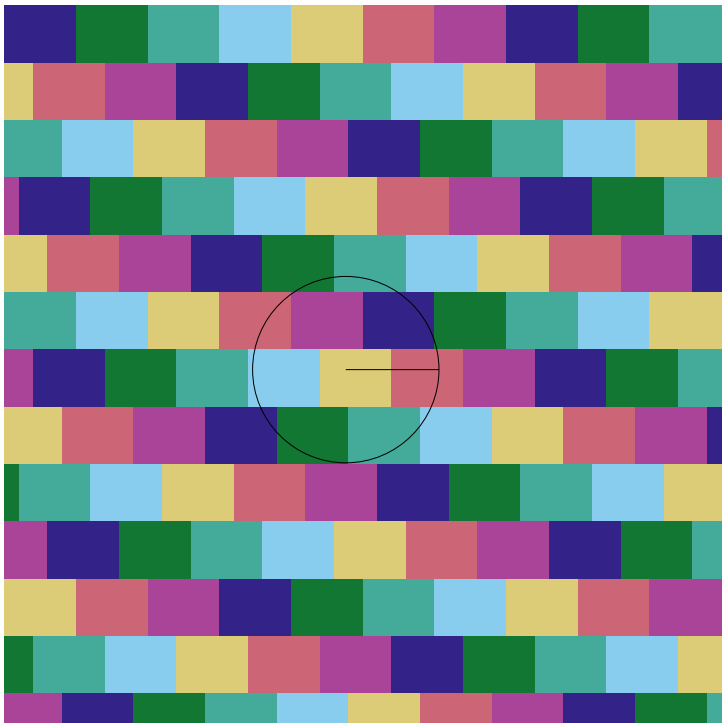
Joonis 9: Test 9, 7 värvi, autor: Benjamin Kleyn



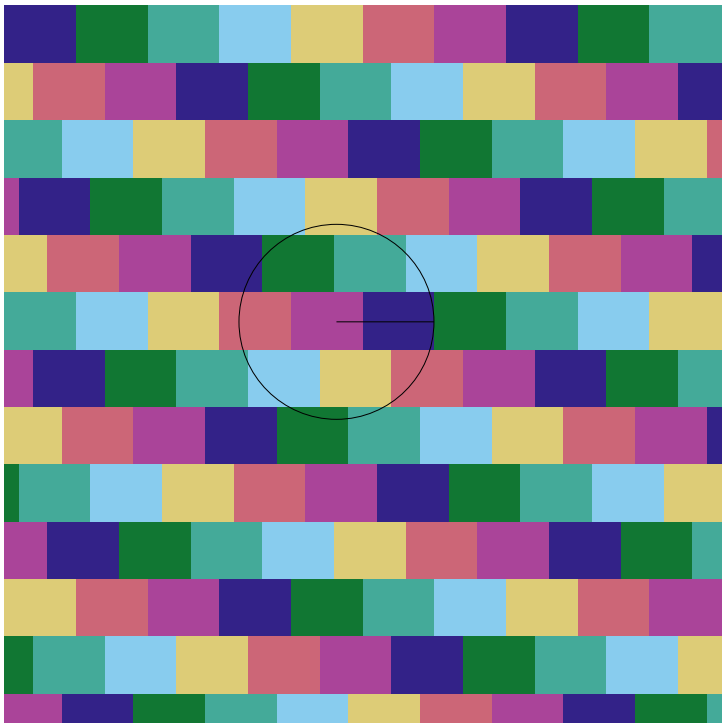
Joonis 10: Test 10, 8 värvi, autor: Benjamin Kleyn



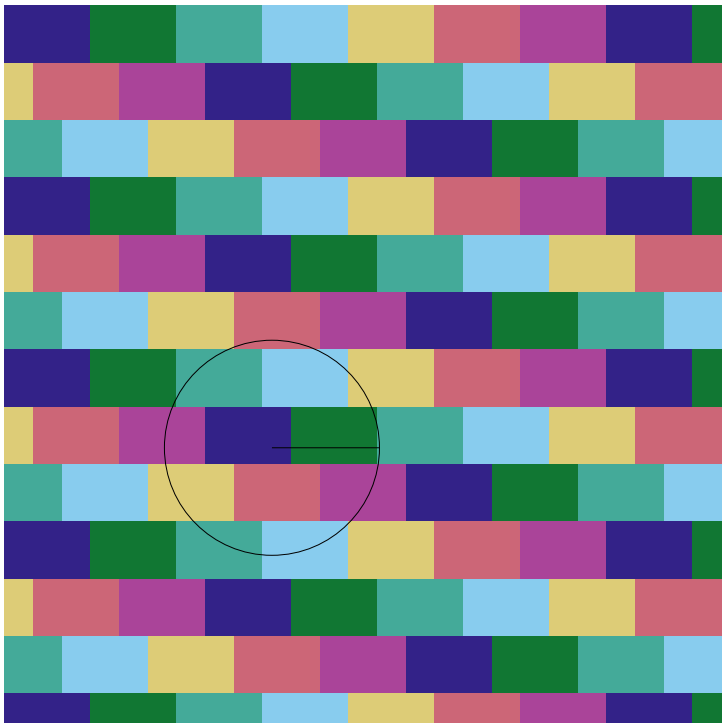
Joonis 11: Test 11, 7 värvi, autor: Ralf Robert Paabo



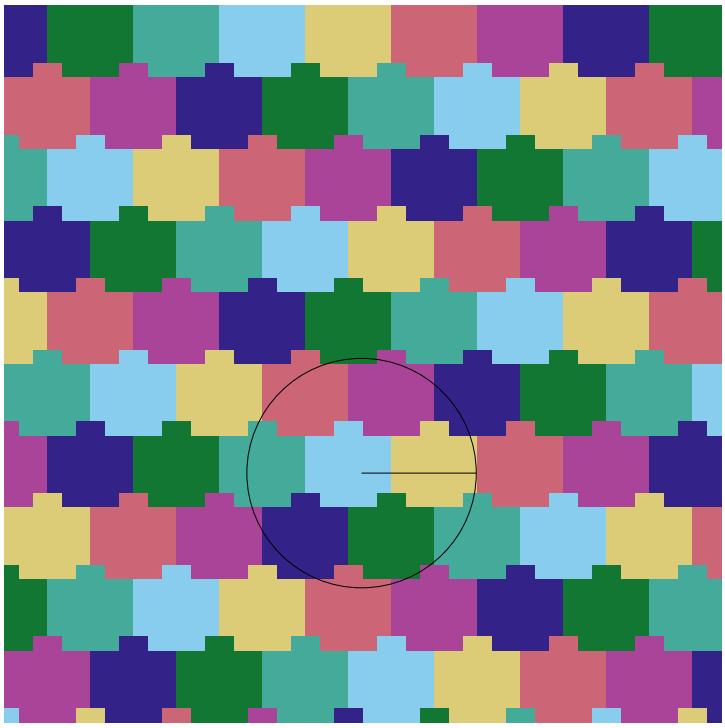
Joonis 12: Test 12, 7 värvi, autor: Benjamin Kleyn



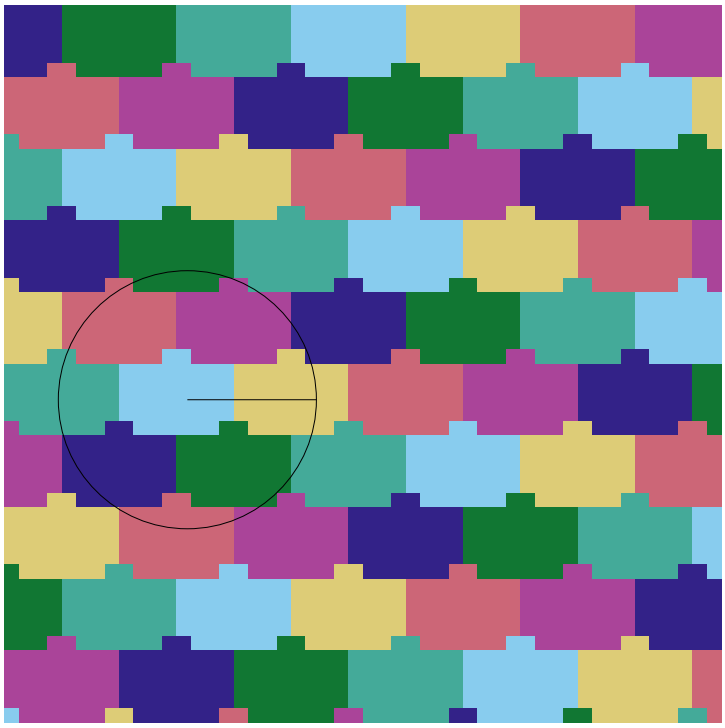
Joonis 13: Test 13, 7 värvi, autor: Benjamin Kleyn



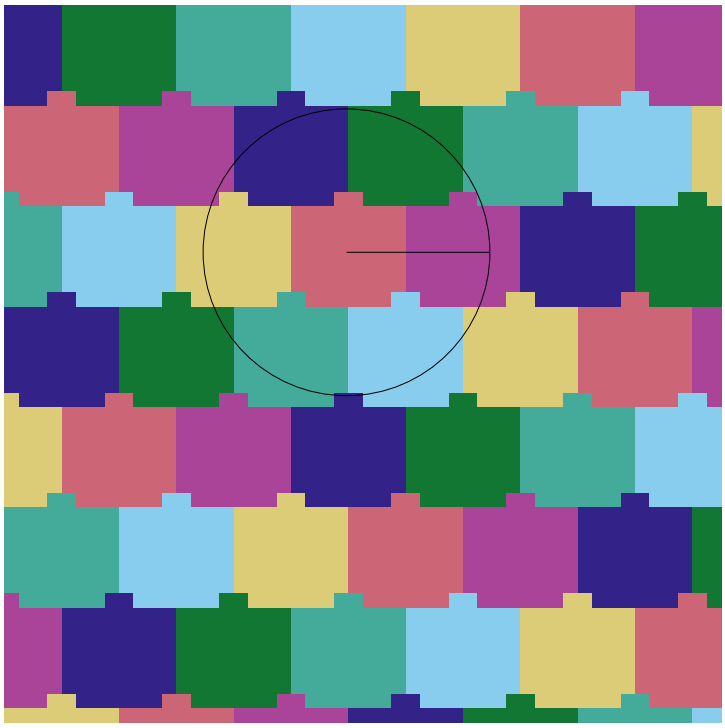
Joonis 14: Test 14, 7 värvi, autor: Benjamin Kleyn



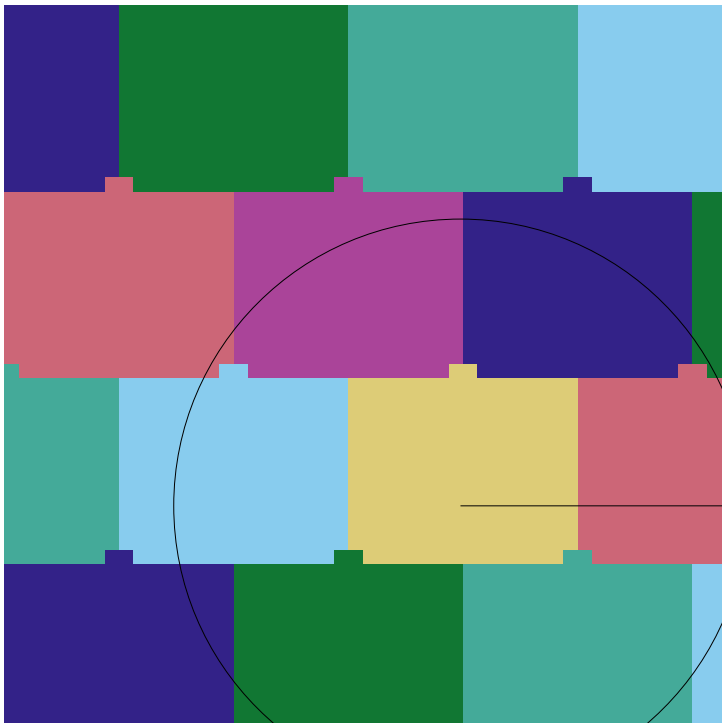
Joonis 15: Test 15, 7 värvi, autor: Ralf Robert Paabo



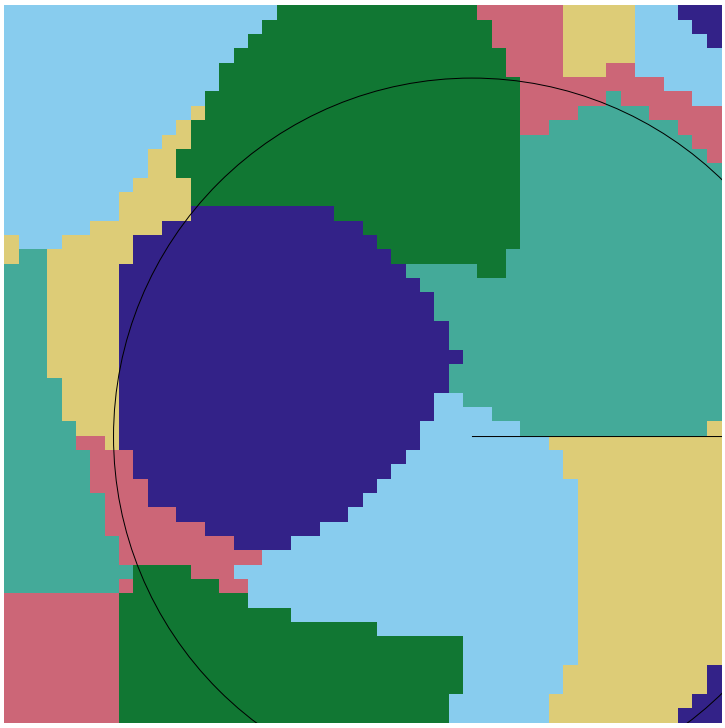
Joonis 16: Test 16, 7 värvi, autor: Ralf Robert Paabo



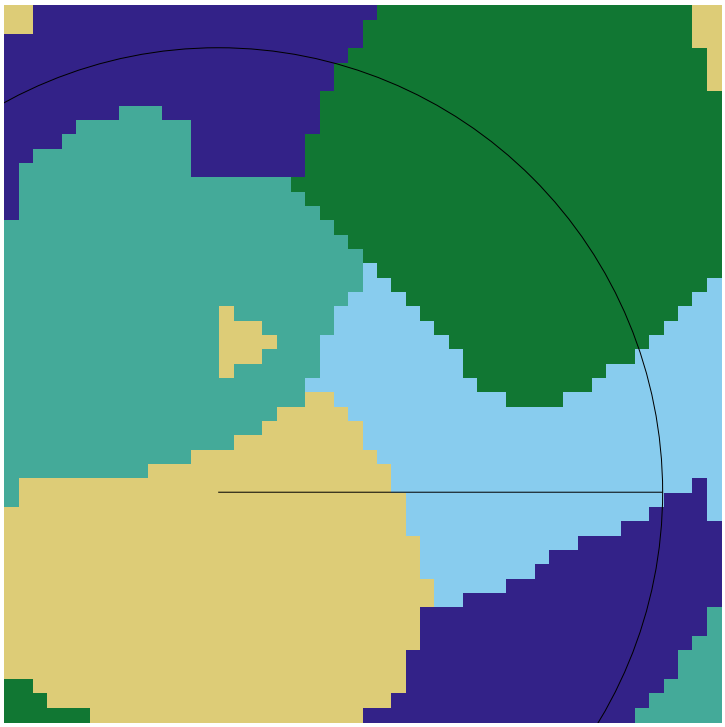
Joonis 17: Test 17, 7 värvi, autor: Ralf Robert Paabo



Joonis 18: Test 18, 7 värvi, autor: Ralf Robert Paabo



Joonis 19: Test 19, 6 värvi, autor: Oskar Märtin



Joonis 20: Test 20, 5 värvi, autor: Hans Gustav Kõljalg