# 16 MOKSLEIVIŲ INFORMATIKOS OLIMPIADA

3 ETAPO 2 DALIS

2005 m. balandžio 6–9 d.

*Ukmergė*

Sprendimai

## 1. Labdara.

*Pirmasis sprendimo būdas.*

Patogiausia visus duomenis surašyti į masyvą ir nuosekliai masyvo elementų reikšmes poromis perskaičiuoti pagal uždavinio sąlygą.

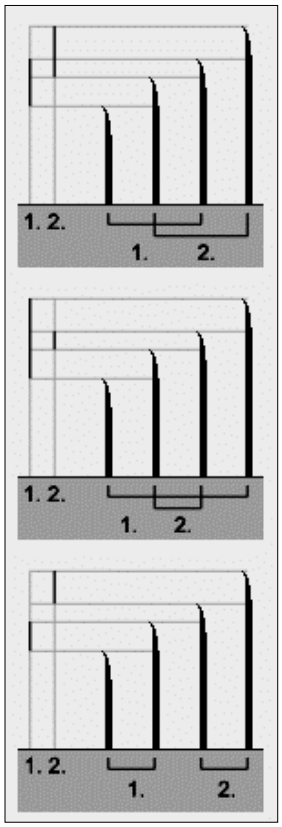
*Antrasis sprendimo būdas.*
Rezultatų formatas leidžia iš bylos nuosekliai skaityti skaičius, jų nesurašant į masyvą. Iš karto pirmoji pora vaikų pasidalija pinigus ir atspausdinami pirmajam poros vaikui likę pinigai. Antrasis poros vaikas tampa naujos poros pirmuoju vaiku.

```
// Vartojamas masyvas
program Labdara1;
const Duomenys = 'LABDARA.DAT';
      Rezultatas = 'LABDARA.REZ';
type Mas = array[1..100] of integer;
var A : Mas; n : integer;   // Duomenų masyvas
    L : integer;            // Labdara
//-----------------------------------
procedure Skaityti;
var F : text;
    i : integer;
begin
    Assign(F, Duomenys); Reset(F);
    Read(F, n);
    for i := 1 to n do
         Read(F, A[i]);
    Close(F);
end;
//-----------------------------------
procedure Skaiciuoti;
var F : text;
    i : integer;
    s : integer;
begin
    L := 0;
    Assign(F, Rezultatas); Rewrite(F);
    for i := 1 to n - 1 do
      begin
        s := A[i] + A[i + 1];    // Poros pradinė pinigų turima suma
        if s mod 2 = 1           // Jeigu nelyginis skaičius,
          then L := L + 1;       //   tai skiriamas labdarai vienas litas
        A[i] := s div 2;         // Pinigų dalyba: pirmasis iš poros
        A[i + 1] := s div 2;     // Pinigų dalyba: antrasis iš poros
        Write(F, A[i]:4);           // Pirmojo iš poros pinigai nekis. Jie
spausdinami
        if i mod 5 = 0 then WriteLn(F);// Eilutėje bus spausdinami penki
skaičiai
      end;
    WriteLn(F, A[n]:4);          // Paskutinio vaiko pinigai
    if L > 0
      then Write(F, 'Labdara = ', L:4)  // Labdarai yra pinigų
      else Write(F, 'Labdarai neliko'); // Labdarai neliko pinigų
    Close(F);
end;
//-----------------------------------
begin
  Skaityti;
  Skaiciuoti;
end.
```

**// Sprendimas be masyvo**

```
program Labdara2;
const Duomenys = 'LABDARA.DAT';
      Rezultatas = 'LABDARA.REZ';
var n : integer;            // Vaikų skaičius
    L : integer;            // Labdara
    F, R : text;
    i : integer;
    s1, s2, s : integer;
begin
    Assign(F, Duomenys); Reset(F);
    Read(F, n);             // Skaitomas vaikų skaičius
    Read(F, s2);            // Skaitomi pirmojo vaiko turimi pinigai
    L := 0;
    Assign(R, Rezultatas); Rewrite(R);
    for i := 1 to n - 1 do
      begin
        s1 := s2;           // Antras poroje tampa pirmuoju naujai porai
        Read(F, s2);        // Skaitomi antrojo iš poros pinigai
        s := s1 + s2;       // Poros turimi pinigai
        if s mod 2 = 1      // Jeigu nelyginis skaičius, tai
          then L := L + 1;  //   labdarai skiriamas vienas litas
        s1 := s div 2;      // Pirmojo iš poros pinigai po dalybos
        s2 := s div 2;      // Antrojo iš poros pinigai po dalybos
        Write(R, s1:4);     // Spausdinami pirmojo iš poros gauti pinigai
        if i mod 5 = 0 then WriteLn(R); // Spausdinama eilutėje po penkis
skaičius
      end;
    WriteLn(R, s2:4);       // Paskutiniojo vaiko gauti pinigai
    if L > 0
      then Write(R, 'Labdara = ', L:4)  // Labdarai yra pinigų
      else Write(R, 'Labdarai neliko'); // Labdarai neliko pinigų
    Close(R);
    Close(F);
end.
```

## 2. Slidės. *(VIII–IX klasės).*




Išrikiuokime slides pagal jų ilgius nemažėjimo tvarka. Kiekvienai slidei į porą vertės imti tik jai gretimą šiame sąraše slidę.

**Įrodymas.** Panagrinėkime, kaip optimalu paskirstyti keturias slides dviems vaikams (žr. pav.). Akivaizdu, jog labiausiai apsimoka vienam vaikui skirti dvi mažesniąsias, o kitam – dvi didesniąsias slides (trečias atvejis paveiksle).

Jeigu skirstant slides vaikams į porą bus paimamos dvi negretimos išrikiuotame sąraše slidės, visuomet atsiras toks ketvertas slidžių, kurias tarpusavy galima paskirstyti „pigiau“ (kaip parodyta paveiksle).

Vadinasi, optimaliame sprendinyje neatsiras poros slidžių, kurios išrikiuotame sąraše nestovėtų greta[1]. ■

Kadangi slidžių yra lygiai 2*N*, vienintelis būdas parinkti *N* porų taip, kad kiekvienos poros slidės išrikiuotame sąraše stovėtų greta, yra poruoti slides iš eilės. T.y. išrikiavę slides nemažėjimo tvarka, pirmam vaikui skiriame pirmą ir antrą slides iš šio sąrašo, antram – trečią ir ketvirtą ir t.t. Gautasis paskirstymas bus optimalus.

Pavyzdžiui, 4 vaikams paskirstykime 8 slides, kurių ilgiai: 1541, 1967, 1834, 1500, 1669, 1724, 1978, 1858 (tai pavyzdinis testas iš sąlygos).

Išrikiavę slides pagal jų ilgius gauname seką:

| | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** |
|---|---|---|---|---|---|---|---|---|
| Ilgis | 1500 | 1541 | 1669 | 1724 | 1834 | 1858 | 1967 | 1978 |
| Numeris | 4 | 1 | 5 | 6 | 3 | 8 | 2 | 7 |

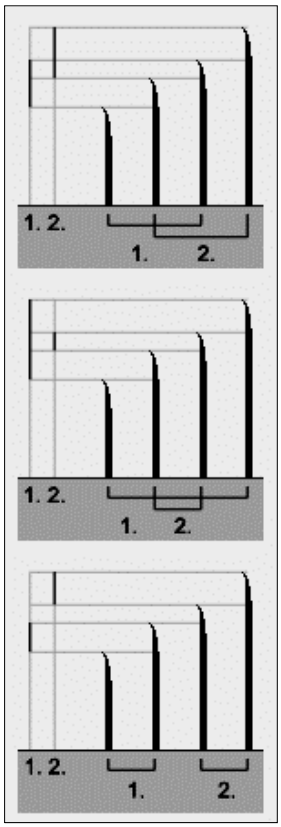
Skirstydami vaikams slides iš eilės, pirmam vaikui skirsime 4-ą ir 1-ą, antram – 5-ą ir 6-ą, trečiam – 3-ą ir 8-ą, o ketvirtam – 2-ą ir 7-ą slides. Šio paskirstymo kaina yra:

$$S = (1541-1500) + (1724-1669) + (1858-1834) + (1978-1967) = 131.$$

[1] Žinoma, su išimtimi – jei dviejų ar daugiau slidžių ilgiai vienodi, tarpusavy jas galima paskirstyti bet kaip.

### 3. Slidės. *(X–XII klasės).*

Išrikiuokime slides pagal jų ilgius nemažėjimo tvarka. Kiekvienai slidei į porą vertės imti tik jai gretimą šiame sąraše slidę.




**Įrodymas.** Panagrinėkime, kaip optimalu paskirstyti keturias slides dviems vaikams (žr. pav.). Akivaizdu, jog labiausiai apsimoka vienam vaikui skirti dvi mažesniąsias, o kitam – dvi didesniąsias slides (trečias atvejis paveiksle).

Tarkime, skirstant slides vaikams, į porą paimamos dvi negretimos išrikiuotame sąraše slidės *a* ir *b*. Jei bent viena iš sąraše tarp jų stovinčių slidžių (tokių nemažiau negu viena) liks nepanaudota, sprendinys akivaizdžiai neoptimalus, nes šią laisvą slidę galėjome imti į porą su *a* arba *b*. Kita vertus, jei bent viena iš tarp jų stovinčių slidžių bus panaudota, atsiras ketvertas slidžių, kurias galima tarpusavy paskirstyti „pigiau“ (kaip parodyta paveiksle).

Vadinasi, optimaliame sprendinyje neatsiras poros slidžių, kurios išrikiuotame sąraše nestovėtų greta[2]. ■

Naudodamiesi šia savybe, uždavinį, kaip *m* slidžių optimaliai paskirstyti *n* vaikų, galime suskaidyti į mažesnius uždavinius. (Čia *i*-tąja slide vadinsime slidę, kuri stovi *i*-tojoje pozicijoje išrikiuotame sąraše).

Jei $m=2n$, turėsime panaudoti visas slides. *m*-toji slidė turi tik vieną „kaimynę“, todėl *m*-tąją slidę imame į porą su (*m*-1)-aja, o likusias *m*-2 slidžių padalinsime likusiems *n*-1 vaikų.

Jei $m>2n$, naudosime ne visas slides. Galime rinktis „pigesnį“ iš šių dviejų variantų:

- skirti *m*-tąją ir *(m-1)*-ąją slides *n*-tajam vaikui, o likusias *m-2* slides paskirstyti likusiems *n-1* vaikų.
- neimti *m*-tosios slidės, ir paskirstyti likusias *m-1* slides *n* vaikų.

Be to, aišku, jog paskirstyti *m* slidžių 0 vaikų „nekainuoja“ nieko.

Taigi sąlygoje apibrėžtą minimalią paskirstymo „kainą“ *S* galime apibrėžti kaip funkciją tokiu būdu:

$$S(n,m)=\begin{cases}0,\ jei\ n=0\\ S(n-1,m-2)+\left|l_m-l_{m-1}\right|,\ jei\ m=2n\neq 0\\ \min(S(n-1,m-2)+\left|l_m-l_{m-1}\right|,S(n,m-1)),\ jei\ m>2n\neq 0\end{cases}$$

Čia $l_i$ žymimas *i*-tosios slidės išrikiuotame sąraše ilgis.

Taigi reikia išrikiuoti slides pagal jų ilgius nemažėjimo tvarka ir suskaičiuoti *S(N, M)*. Tačiau skaičiuoti šią funkciją rekursyviai, kai *N* ir *M* dideli, o $M > N$, užtruktų ilgai, nes tos pačios jos reikšmės būtų skaičiuojamos labai daug kartų. Kita vertus, mums gali tekt daugiausiai skaičiuoti $N_{max}\cdot M_{max} = 1000\cdot 2000 = 2000000$ skirtingų funkcijos reikšmių, o tokį kiekį galime saugoti atmintyje. Todėl galime panaudoti dinaminio programavimo techniką.

[2] Žinoma, su išimtimi – jei kelių slidžių ilgiai vienodi, tarpusavy jas galima paskirstyti bet kaip.

Vienas iš būdų yra suskaičiavus *S(i, j)* reikšmę, įsiminti ją *N*×*M* dydžio reikšmių lentelėje (masyve), ir kitą kartą prireikus šios reikšmės neskaičiuoti, o tiesiog grąžinti ją iš lentelės.

Kitas būdas yra pildyti funkcijos reikšmių lentelę nuo „apačios“ iki „viršaus“. Šį būdą pailiustruosime sąlygoje pateiktu pavyzdžiu: trims vaikams reikia paskirstyti aštuonias slides, kurių ilgiai: 1731, 1572, 2041, 1561, 1682, 1572, 1609, 1731.

Išrikiavę slides pagal jų ilgius, gauname seką:

| | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** |
|---|---|---|---|---|---|---|---|---|
| Ilgis | 1561 | 1572 | 1572 | 1609 | 1682 | 1731 | 1731 | 2041 |
| Numeris | 4 | 2 | 6 | 7 | 5 | 1 | 8 | 3 |

Vienam vaikui prireiks mažiausiai dviejų slidžių. Skirti jam pirmąsias dvi slides iš išrikiuoto sąrašo „kainuos“ 1572-1561 = 11. Vienam vaikui paskirstyti pirmas tris slides galime dviem būdais: likti prie ankstesniojo paskirstymo arba skirti antrą ir trečią slides. Renkamės antrąjį variantą, nes 1572-1572 = 0 < 11. Vienam vaikui paskirstyti pirmas keturias slides vėlgi galime dviem būdais: taip, kaip paskirstėme pirmas tris slides, arba skirti trečią ir ketvirtą slides.

Nebereikia skaičiuoti, kaip vienam vaikui paskirstyti pirmas penkias slides, nes likusiems dviems vaikams prireiks bent keturių slidžių. Tačiau jei ir suskaičiuotume, sprendimas nuo to nenukentėtų.

Mokėdami optimaliai paskirstyti pirmąsias *j* slidžių vienam vaikui, galime skaičiuoti, kaip tai padaryti dviems vaikams. Dviems vaikams prireiks bent keturių slidžių. Antram vaikui skirsime trečią ir ketvirtą slides, o pirmam vaikui paskirstyti dvi slides jau mokame. Iš viso gausime 1609-1572 + 11 = 48. Dviems vaikams paskirstyti penkias slides galime dviem būdais: taip, kaip paskirstėme pirmas keturias slides, arba, antram vaikui skirti ketvirtą ir penktą slides, o pirmąjam paskirstyti pirmas tris (jau mokame). Tai kainuotų 1682-1609 + 0 = 73 > 48, todėl liekame prie ankstesniojo paskirstymo.

Mokėdami optimaliai paskirstyti pirmąsias *j* slidžių dviems vaikams, galėsime skaičiuoti, kaip tai padaryti trims vaikams ir taip tęsdami skaičiavimus rasime, kaip optimaliai *M* slidžių paskirstyti *N* vaikų. Visa reikšmių lentelė atrodys taip:

| **Vaikų skaičius (n)** \ **Slidžių skaičius (m)** | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** |
|---|---|---|---|---|---|---|---|---|---|
| **0** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **1** | | | 11 | 0 | 0 | 0 | 0 | 0 | 0 |
| **2** | | | | | 48 | 48 | 48 | 11 | 11 |
| **3** | | | | | | | 97 | 48 | 48 |

Taigi minimali paskirstymo kaina yra 48. Kiekviename žingsnyje taip pat turime pasižymėti (pavyzdžiui, atskirame *N*×*M* dydžio masyve), koks pasirinkimas buvo atliktas (ar *j*-oji ir (*j*-1)-oji slidės buvo skirtos *i*-tajam vaikui). Lentelėje šiuos pasirinkimus žymi rodyklės. Pradėdami nuo langelio [*N*, *M*] ir „sekdami rodyklėmis“ iki [0, 0], galėsime surašyti, kurias slides iš išrikiuoto sąrašo kiekvienam vaikui skyrėme.

Aprašytojo algoritmo sudėtingumas yra $O(N \cdot M)$, t.y. algoritmo atliekamų veiksmų skaičius auga taip pat kaip sandauga $N \cdot M$.

## 5. Kortelės.

Tarkim, kad kortelės jau kaip nors sudėliotos. Natūralu, kad statydami kortelę prieš pliusą, atversime jos mažesnįjį skaičių, o prieš minusą - atvirkščiai. Panagrinėkime, kada apsimoka sukeisti dvi korteles? (Be abejo, sukeisti gali būti verta tik tada, kai kortelės stovi prieš skirtingus ženklus.)

Sakykime, priešais pliusą stovi kortelė su skaičiais $a_1$ ir $a_2$ ($a_1 \geq a_2$, todėl atversta $a_2$); O priešais minusą stovi kortelė su skaičiais $b_1$ ir $b_2$ ($b_1 \geq b_2$, todėl atversta $b_1$). Jas verta sukeisti, jei $a_2-b_1>b_2-a_1$ (t.y. sukeitę gausime mažesnį skaičių), taigi jei $a_1+a_2>b_1+b_2$.

*Pavyzdžiui*, $a_1 = 10$, $a_2 = 5$; $b_1 = 6$, $b_2 = 3$; Kadangi 10+5>6+3, korteles sukeisti verta. Prieš sukeitimą turime 5-6 = -1, po sukeitimo: 3-10 = -7;

Vadinasi, optimalu prie pliuso ženklų statyti korteles su mažiausiomis skaičių sumomis (kitaip atsirastų pora kortelių, kurias sukeitus vietomis reiškinio reikšmė sumažėtų). Taigi, galime suskaičiuoti ant kiekvienos kortelės užrašytų skaičių sumas ir korteles išrikiuoti kuria nors (sumų didėjimo ar mažėjimo) tvarka. Tuomet prieš pliusus dėti korteles, kurių abiejų skaičių suma kuo mažesnė, o prie minusų – likusias korteles su didesnėmis sumomis.

Kortelių gali būti iki 100000, o tokį kiekį išrikiuoti per leistiną laiką gali tik efektyvus[3] algoritmas. Kita vertus, mums tereikia žinoti, kurioje išrikiuotos sekos pusėje (tarp mažesniųjų ar tarp didesniųjų) kiekviena kortelė atsidurtų. Užtektų sužinoti, kokia kortelė (tiksliau, kokia suma) stovi išrikiuotos sekos viduryje, o visas kitas galėtume palyginti su ja.

Pastebėkime, jog visos sumos priklauso gana nedideliam intervalui [-4000, 4000]. Skaitydami duomenis, galime masyve *mas[x]* pasižymėti, kiek yra kortelių, kurių skaičių suma lygi *x*. Tada iš eilės skaičiuoti, kiek yra kortelių, kurių sumos mažesnės arba lygios -4000, -3999, -3998 ir t.t. ir sustoti, kai viršysime pusę kortelių skaičiaus. Suma, ties kuria sustosime, bus ant pirmos „didesniosios“ kortelės užrašytų skaičių suma.

Korteles, kurių skaičių sumos mažesnės už lyginamosios, „sudedame“ prie pliuso ženklų. Jei padėjome mažiau negu pusę kortelių, tiek, kiek trūksta, pridedame tokių, kurių skaičių sumos sutampa su lyginamosios. Visas likusias „dedame“ prie minuso ženklų.

### *Testai*

| *Nr* | *Kortelių skaičius* | *Minimali reiškinio reikšmė* | *Paaiškinimai* |
|---|---|---|---|
| 1 | 8 | -25 | Nedideli testai. |
| 2 | 10 | -40 | |
| 3 | 24 | -400 | |
| 4 | 100 | -72000 | Vidutiniai testai. |
| 5 | 500 | -332000 | |
| 6 | 2500 | -1682000 | |
| 7 | 5000 | -3313000 | |
| 8 | 10000 | -13456000 | |
| 9 | 50000 | -100000000 | Ant pusės kort. užrašyta 2000 ir 0, ant likusių -2000 ir 0. |
| 10 | 100000 | -134760000 | Didelis testas. Abiejų paskutinių testų neįveikia paprasti rikiavimo algoritmai. |

[3] Įprasti rikiavimo algoritmai (pvz. *Burbuliuko*) yra $O(N^2)$ eilės – šis žymėjimas reiškia algoritmo augimo greitį (šiuo atveju dvigubam kiekiui duomenų išrikiuoti reikėtų keturis kartus daugiau laiko). Efektyviu čia vadinamas $O(N \log N)$ eilės rikiavimo algoritmas, pavyzdžiui *Quicksort*, *Heapsort*.

## 6. Nykštukai.

***Sprendimas***

Neefektyvu imti kiekvieną nykštuką ir tikrinti, ar jis spės išeiti iš namo.

Todėl darysime taip: įsivaizduokime, kad lauke yra vienas nykštukas. Jei rasime visus nykštukus, pas kuriuos šis nykštukas gali ateiti per $t$ ar mažiau laiko vienetų, tai turėsime uždavinio sprendimą.

Jei nykštukas iš lauko gali pasiekti nykštuką kambaryje per $x$ laiko vienetų, tai ir nykštukas iš to kambario per $x$ laiko vienetų gali išeiti į lauką.

Kuriuos kambarius galima pasiekti per ***t*** laiko vienetų iš lauko galima nustatyti naudojant ***paieškos į plotį algoritmą***, kuris pradedamas iš lauko, ir kuris nutraukiamas pasiekus laiko ribą.

Paprastesniu atveju (kai namas yra vienaaukštis) galima uždavinį spręsti bangos metodu.

## 7. Krokodilai.

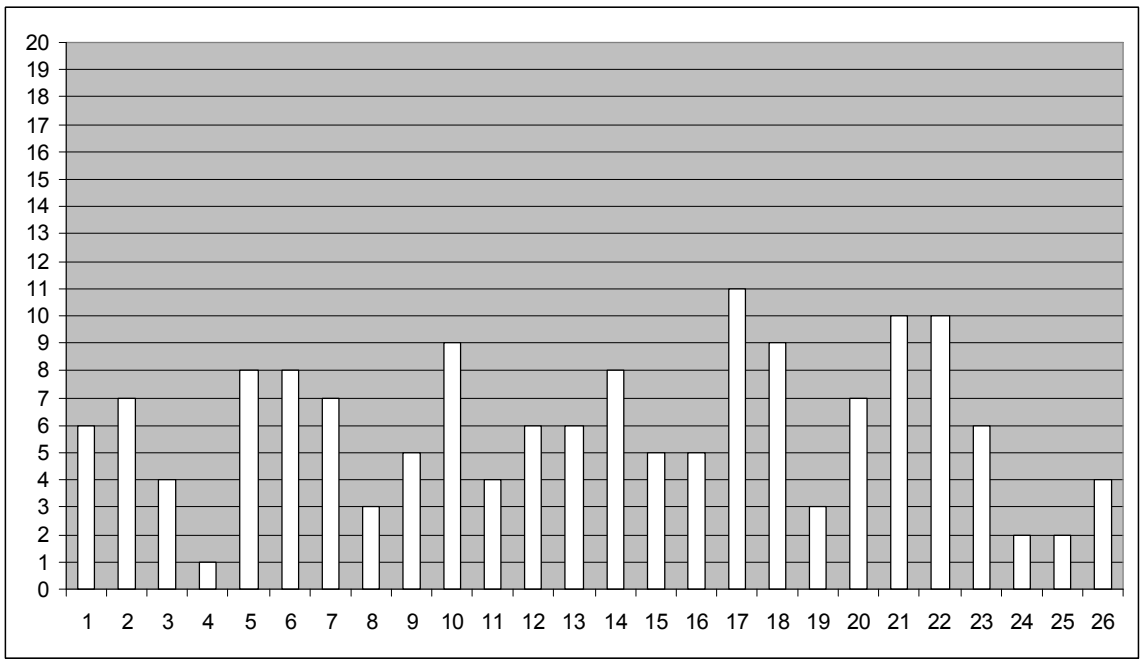
Šiame aprašyme nagrinėjamas tik sudėtingesnis sąlygos variantas. Kai duotos dvi grupės krokodilų – agresyvių ir meilių. Be to, žinomas kiekvieno iš jų odos, padengtos žalia spalva, plotas procentais (keturių skaičių po kablelio tikslumu). Tačiau nežinoma kurie – daugiau žalios ar daugiau baltos spalvos turintys krokodilai yra agresyvesni.

Reikia rasti slenkstį – odos, padengtos tam tikra spalva X, plotą procentais – pagal kurį suskirsčius visus ištirtuosius krokodilus būtų padaryta mažiausiai klaidų. Jeigu yra keletas slenksčių, kurie vienodai gerai (daro lygiai tiek pat klaidų) atskiria agresyvius krokodilus nuo meilių pagal jų odos spalvą, reikia pateikti mažiausią iš jų.

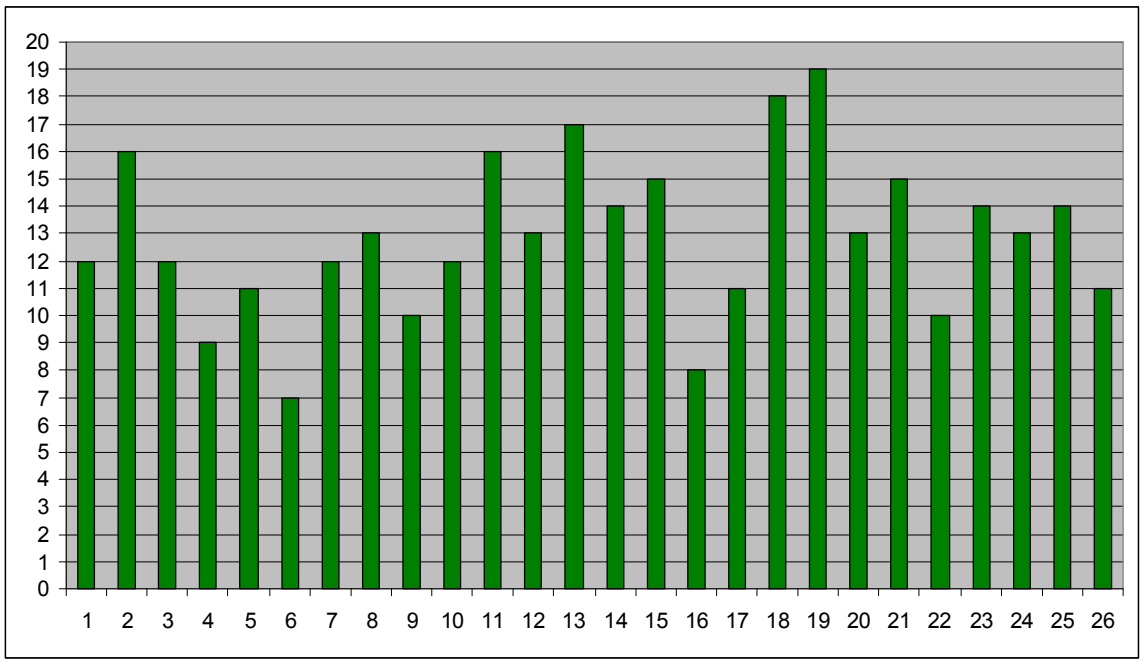
Ieškant slenksčio nebūtina žinoti, kuris kiekvieno krokodilo odos procentas padengta tam tikra spalva, kadangi apribotas tikslumas, kuriuo pateikti duomenys. Krokodilus, kurių odos plotas vienodai padengtas tam tikra spalva, galime sugrupuoti ir saugoti tiktai jų skaičių.

Pavyzdžiui, turime 26 agresyvius ir 26 neagresyvius krokodilus.

Žemiau pateiktoje diagramoje pavaizduoti visi agresyvūs krokodilai, stulpelio aukštis žymi krokodilo odos plotą padengtą žalia spalva:

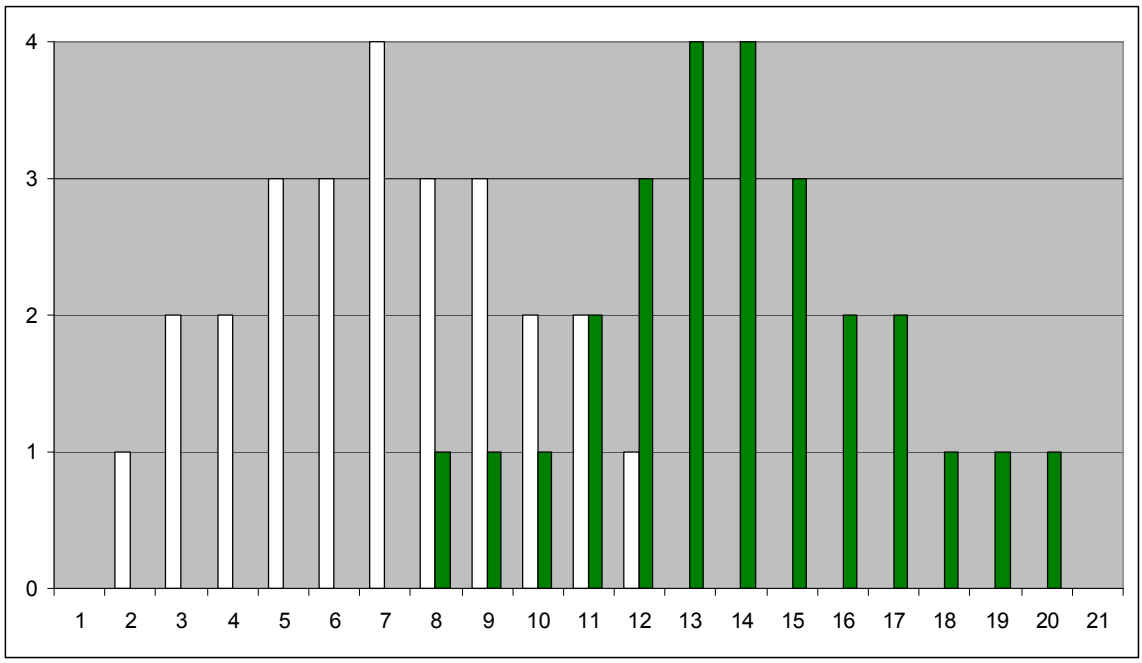



Kitoje diagramoje pavaizduoti visi neagresyvūs krokodilai. Stulpelio aukštis irgi žymi krokodilo odos plotą padengtą žalia spalva:




Pirmiausia atkreipkime dėmesį, kad abiem atvejais dominuoja balta spalva. Agresyvių krokodilai daugiausia 11% odos yra žalios spalvos, o neagresyvių – tik 19%. Tačiau akivaizdu, kad daugiau žalios spalvos turintys krokodilai yra neagresyvūs.

Kaip jau buvo minėta aukščiau, galima krokodilus sugrupuoti pagal tai, kiek jų odos ploto padengta tam tikra spalva. Meilieji vis dar žymimi žalia (tamsia) spalva, o agresyvieji - balta:

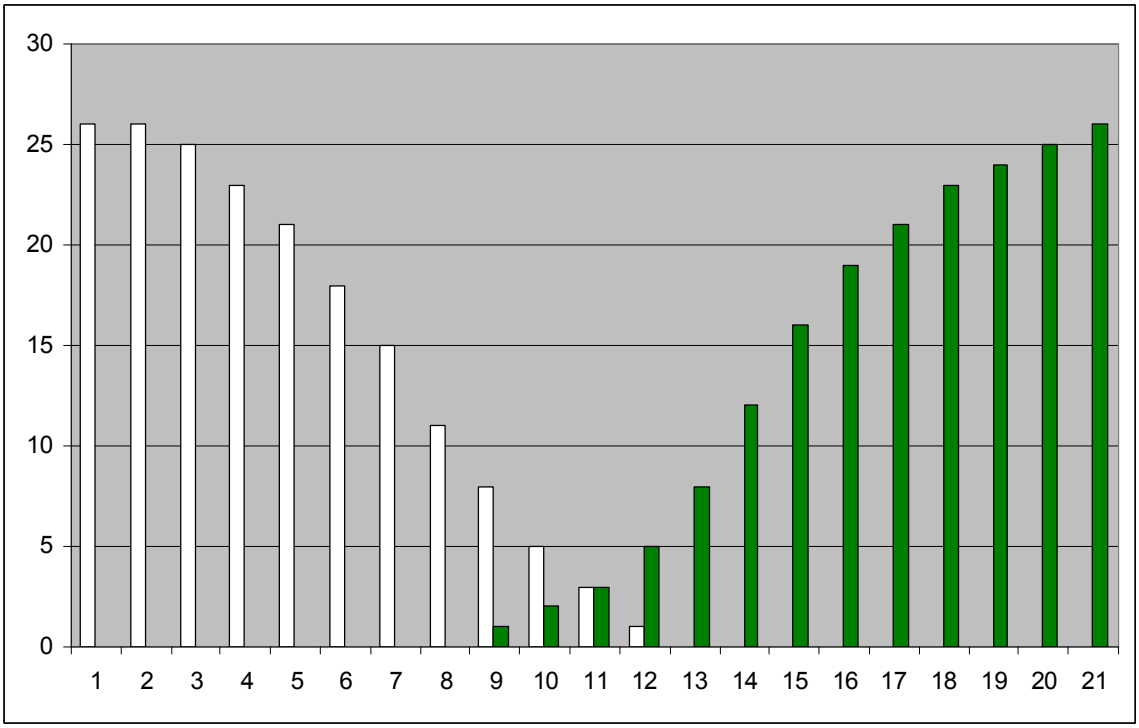



Čia X ašyje atidėta odos plotas procentais, o Y ašyje – krokodilų skaičius. Tokiu būdu, jei krokodilų, kuriuos bandome atskirti, kiekis yra labai didelis, ne tik patogiai sutvarkomi duomenys, bet ir taupoma atmintis.

Toliau, ieškant slenksčio, kuriuo būtų galima atskirti agresyviuosius krokodilus nuo likusių, reikia suskaičiuoti, kiek klaidų būtų daroma rūšiuojant krokodilus pagal odos spalvą kiekvienai galimai slenksčio pozicijai (pavyzdyje slenkstis gali būti bet kurioje pozicijoje nuo 1 iki 21), ir išrinkti

poziciją, kurioje daromas klaidų kiekis būtų mažiausias.

Skaičiuoti daromų klaidų skaičių kiekvienoje pozicijoje iš naujo gali būti labai neefektyvu. Todėl duomenis vėl galima transformuoti į patogesnę formą. Kiekvienai krokodilų charakterio grupei, perbėgant sugrupuotus duomenis vieną kartą, galima paskaičiuoti, kiek klaidų daroma pasirinkus slenkstį kiekvienoje pozicijoje tokiu būdu:




Čia X ašyje atidėta odos plotas procentais, o Y ašyje – kiek klaidų būtų daroma skirstant vienos ar kitos charakterio grupės krokodilus. Beliko dar karta perbėgti transformuotus duomenis ir surasti minimalią daromą klaidą (kiek agresyvių ir meilių krokodilų sumoje būtų klaidingai surūšiuota pasirinkus tam tikrą slenkstį).

Pateiktame pavyzdyje,

- jeigu slenkstis būtų ties 10 – klaidingai būtų suskirstyti 8 krokodilai (5 agresyvūs ir 2 meilūs),
- jeigu 11 – 6 krokodilai (3 agresyvūs ir 3 meilūs),
- jeigu 12 – 6 krokodilai (1 agresyvus ir 5 meilūs),
- jeigu 13 – 8 krokodilai (visi 8 meilūs).

Teisingas sprendimas – mažiausiai 11% krokodilo odos ploto turi būti padengta žalia spalva.

Pavyzdyje meilūs krokodilai yra labiau žali negu balti, tuo tarpu jeigu meilūs krokodilai kartais būtų labiau balti negu žali, viską reikėtų daryti lygiai taip pat, tik sukeisti spalvas vietomis. O jeigu iš anksto nebūtų žinoma, kaip susieta spalva su charakteriu, tektų patikrinti abu variantus.

## 8. Po mūšio.

*Atstumu tarp dviejų lentos langelių vadinsime mažiausią raitininko ėjimų skaičių iš vieno langelio į kitą*. Jei raitininkas negali nukeliauti iš vieno langelio į kitą, tai atstumas tarp jų yra neapibrėžtas.

Pažymėkime $A_r(i,j)$ atstumą nuo *r*-ojo raitininko pradinio langelio iki lentos langelio $(i,j)$. Pažymėkime $K(i,j)$ atstumą nuo karaliaus pradinio langelio iki lentos langelio $(i,j)$; jis rodo, per kiek mažiausiai ėjimų karalius gali būti nuneštas į langelį $(i,j)$.

Vykdydami paiešką platyn randame visus $K(i,j)$ bei kiekvienam raitininkui visus $A_r(i,j)$. Tada paeiliui įvertiname kiekvieną lentos langelį: ar gali jame susirinkti visos figūros ir, jei gali, tai per kiek ėjimų. Iš šių įvertinimų išrenkame geriausią.

Langelis $(i,j)$ gali *būti įvertintas taip.* Pirmiausia nustatome, ar langelyje gali įvykti susitikimas:

- Visi raitininkai turi galėti pasiekti langelį $(i,j)$ ($A_r(i,j)$ visiems *r* turi būti apibrėžta)
- Iš karaliaus langelio turi būti galima pasiekti langelį $(i,j)$ ($K(i,j)$ turi būti apibrėžta)
- Bent vienas raitininkas(jei raitininkų yra) turi galėti pasiekti karaliaus langelį, nes karalius turi būti nuneštas į susirinkimo langelį ($A_r(k_i,k_j)$ turi būti apibrėžta bent vienam *r*, čia $(k_i,k_j)$ -pradinis karaliaus langelis). Pastebėsime, kad ši sąlyga galioja ir kai karaliaus nešti nereikia, t.y. kai karaliaus langelis yra $(i,j)$ - tada visi raitininkai turi galėti pasiekti karaliaus langelį (1. sąlyga).

Jei langelyje susitikimas gali įvykti, tai *mažiausią skaičių ėjimų*, reikalingų susirinkti langelyje $(i,j)$ *nustatome tokiu būdu*. Jei *v*-as raitininkas neša karalių į langelį $(i,j)$, tai mažiausias reikalingų ėjimų skaičius bus $(SUM(r=1,R)\ A_r(i,j)) - A_v(i,j) + A_v(k_i,k_j) + K(i,j)$ – *v*-as raitininkas neina tiesiai į susitikimo langelį$(i,j)$, bet eina į karaliaus langelį $(k_i,k_j)$, o iš ten(kartu su karaliumi) – į langelį $(i,j)$.

Apskaičiavę šį dydį visiems raitininkams išrenkame, kuriam raitininkui labiausiai apsimoka nešti karalių į langelį $(i,j)$ ir kiek mažiausiai ėjimų iš viso reikės tam, kad visos figūros susirinktų langelyje $(i,j)$.