
Perrinkimo ir godieji algoritmai bioinformatikoje

Bendroji perrinkimo technika (kombinatorinių algoritmų elementai)

Algoritmų konstravimo technika

Pilnas perrinkimas:

- Kaip perrinkti visus aibės poaibius?
- Taikymo pavyzdys:
  - Objektų poaibis, atitinkantis „gerus“ požymius;

Algoritmų konstravimo technika

Pilnas *poaibių* perrinkimas. Algoritmo elementai:

- –Dvejetainis skaitliukas
  - *a[i]* – dvejetainiai skaitmenys, realizuojame *Kitas(int[] a)*;
- –Dvejetainė sistema
  - *for(int i = 0; i < (1<<N); i++)*
    - *DvejetainisKodas(a,i)*.....

Algoritmų konstravimo technika

Pilnas Dekarto sandaugos perrinkimas:

- Kaip perrinkti visus aibės AxB elementus? (A ir B yra realizuoti, pavyzdžiui, kaip masyvai)
- ... aibės AxBxC?
- ... aibės A1xA2x...xAN?
- Taikymo pavyzdys: „receptas“, t.y., objektas iš A1xA2x...xAN, kurį įmanoma atrasti tik perrenkant visus įmanomus variantus

Algoritmų konstravimo technika

Pilnas Dekarto sandaugos perrinkimas:

- Pavyzdys
  - A1={1,2,3,4}
  - A2={1,2,3}
  - A3={1,2}
  - A4={1,2,3}
  - A5={1,2,3,4,5}

Klausimas: kiek variantų turi A1xA2x...xA5?

Pilnas Dekarto sandaugos perrinkimas:

- **Klausimas: kaip apibendrinti poaibių generavimo algoritmą?**
- **Pavyzdžiui, turime variantą iš sandaugos (1,1,1,1,1), koks bus kitas? (1,1,1,1,2)**
- **O jeigu variantas yra (1,1,1,3,5), tai koks kitas?**

Pratimas

- **Tarkime, kad mes sunumeravome pagal minėtą tvarką A1x...xA5 elementus nuo 1 iki 360.**
  - Nurodykit elementus, kurie yra 10, 100, 200, 300
  - Suformuluokite algoritmo idėją

Keitiniai...

- **Yra daug galimybių juos generuoti:**
  - pasinaudoti „cikliniais“ postūmiais;
  - pasinaudoti leksikografine tvarka;
  - pasinaudoti faktorialinę skaičiavimo sistema

Cikliniai postūmiai

- 1 2 3
- 1 3 2
- 2 3 1
- 2 1 3
- 3 1 2
- 3 2 1

Leksikografinė tvarka (idėja)

- **a b c**
- **a c b**
- **b a c**
- **b c a**
- **c a b**
- **c b a**

Keitiniai (faktorialinė skaičiavimo sistema)...

Mūsų tradicinė skaičiavimo sistema gali būti pavadinta „geometrine“.

Bet galima konstruoti ir faktorialinę sistemą

$$\sum_{i=0}^{n} i \cdot i! = (n+1)! - 1.$$

18_{10}	$3_3 0_2 0_1 0_0$
19_{10}	$3_3 0_2 1_1 0_0$
20_{10}	$3_3 1_2 0_1 0_0$
21_{10}	$3_3 1_2 1_1 0_0$
22_{10}	$3_3 2_2 0_1 0_0$
23_{10}	$3_3 2_2 1_1 0_0$

Keitiniai (faktorialinė skaičiavimo sistema)...

- **Pratimas. Užrašykite faktorialinėje skaičiavimo sistemoje skaičius:**
  - 1
  - 8
  - 64
  - 100

Keitiniai (faktorialinė skaičiavimo sistema)...

- **Pratimas. Užrašykite faktorialinėje skaičiavimo sistemoje skaičius:**
  - 1
  - 8
  - 64
  - 100

Jeigu naudojame STL iš C++

```
//  Arba tiesiog pasinaudoti STL :)
#include <algorithm>
#include <iterator>
#include <vector>
#include <iostream>
using namespace std;
int main(void) {
        vector<int> v;
        v.push_back(1); v.push_back(2); v.push_back(3);
        while (next_permutation(v.begin(), v.end() ) ) {
                // pagal visus ketinius
            copy(v.begin(), v.end(), ostream_iterator<int>(cout, ""));
            cout << endl;
        }
        return 0;
}
```

Java kodas

http://code.google.com/p/algorithms-java/source/browse/trunk/src/main/java/com/google/code/Permutations.java?r=3

```java
package com.google.code;

/**
 * Works with <a href='http://en.wikipedia.org/wiki/Permutation'>permutations</a>
 */
public class Permutations {
    /**
     * Accepts an array of <b>ints</b> and reorders it's elements
     * to recieve lexicographically next permutation
     *
     * @param p permutation
     * @return false, if given array is lexicographically last permutation, true otherwise
     */
    public static boolean nextPermutation(int[] p) {
        int a = p.length - 2;
        while (a >= 0 && p[a] >= p[a + 1]) {
            a--;
        }
        if (a == -1) {
            return false;
        }
        int b = p.length - 1;
        while (p[b] <= p[a]) {
            b--;
        }
        int t = p[a];
        p[a] = p[b];
        p[b] = t;
        for (int i = a + 1, j = p.length - 1; i < j; i++, j--) {
            t = p[i];
            p[i] = p[j];
            p[j] = t;
        }
        return true;
    }
}
```

O kaip su deriniais?

- **Pavyzdys. Reikia išrinkti iš aibės skaičių, reprezentuojamos masyvo A[0..99], tris tokius, kad f(A[i1],A[i2],A[i3]) tenkina tam tikrą nurodytą sąlygą,**

  indeksai i1,i2,i3 tarpusavyje skirtingi.
- **Kaip spręsti? (Mokyklinis uždavinys :))**

Bendras atvejis

for i_1:= 0 to N do

  for i_2 := i_1+1 to N do

    ....

      for i_k := i_{k-1}+1 to N do

        If <sąlyga> ...

Bet tiesiogiai kalbos sintaksė taip neleidžia parašyti

Sprendimas

1) Rekursija
2) Iteracija

   Stekas: jis parodys atitinkamo indekso reikšmes.

   [i1] [i2] [i3] ... [ik]

Einamasis indeksas asocijuojamas su SP(steko rodykle). Kai tam tikras indeksas jau „prabėgo“ visas reikšmes, tiesiog darome pop operaciją, po to – push su pradine „vidinio“ indekso reikšme, t.y., vienetu didesne, negu esantis indeksas

Dar vienas sprendimas

Jis remiasi **leksikografine** derinių tvarka. Pabandykit suformuluoti šią idėją

Tarkime abėcėlė yra {a,b,c,d,e}, o domina 3 raidžių žodžiai, kur raidės „didėja“ (nes jie ir bus atitinkamų *derinių atstovai*)

a b c

a b d

a b e

a c d

a c e

a d e

b c d

b c e

b d e

c d e

Restriktazės padengimo uždavinys

Restrikcijos fermentai

- **Tam tikrų fermentų pagalba galima sukarpyti DNR tam tikrose vietose;**
- **Pavyzdys: HindIII fermentas turi kirpimo schemą:**

```
5'-A |A G C T T-3'
3'-T T C G A| A-5'
```

Problemos formulavimas

- **Tarkime, kad turime L ilgio DNR seką, kuri yra kerpama mums nežinomuose vietose tam tikro restrikcijos fermento pagalba;**
- **Elektroforezės pagalba mes galime sužinoti sukarpytų dalių ilgius: jų aibę žymėsime ΔX;**
- **Uždavinys: pagal ΔX reikia rasti bent vieną galima X variantą.**

Kur sunkumai?

Pratimai

- **Tarkime, kad X yra {1, 5, 7, 10}**
  - Raskite ΔX;
- **Tarkime, kad X yra sudaryta iš N skaičių. Kiek elementų gali turėti ΔX?**
- **Įrodykite teiginį: jeigu X yra {t1,t2,..., tN}, o Y yra {t1+a,t2+a. ...., tN+a}, tai ΔX = ΔY**
- **Sukurkite dvi vienodo dydžio nesutampančias aibes X ir Y, kurios prasideda nuo 0 taip, kad būtų ΔX = ΔY**

Pastaba

- **Toliau visur tarsime, kad X visada prasideda nuo 0**

Kaip ieškoti X pagal ΔX?

- **Pirma idėja:**
  - Kadangi X prasideda nuo 0, tai galime rasti didžiausią skaičių iš ΔX, pažymėkime jį L ir taip pat aibės X elementų skaičių pagal ΔX elementų skaičių N (**kaip?**);
  - Tada imame visas įmanomus N-2 ilgio vektorius V(v1,v2,...vN-2) tokius, kad:
    - 0<v1<v2<..<v{N-2}<L
    - ΔV = ΔX - {0,L};
  - Bet kuris iš minėtų vektorių ir bus uždavinio sprendinys

Aptarimas

- **Pagrindinis techninis klausimas: kaip ieškoti minimus vektorius?**
  - Atsakymas: rekursija;
- **Kada sustoti?**
  - Kai radome bent vieną tinkamą vektorių arba „baigėsi“ variantai;

Pratimas

- **Tarkime DX={2,3,4,5,7,9}. Reikia rasti X pagal pateiktą algoritmą.**

Patobulinimas

- **Galime labai paprastai patobulinti paiešką:**
  - Reikia ieškoti tokių vektorių V, kurių komponentės priklauso ΔX **(kodėl?)**
- Kodėl toks būdas tobulesnis?
  - Tiesiog elementų skaičius pačiame DX ir visos sekos ilgis L gali labai skirtis: dažniausiai L būna labai didelis palyginus su #ΔX.

Kaip dar galima patobulinti?

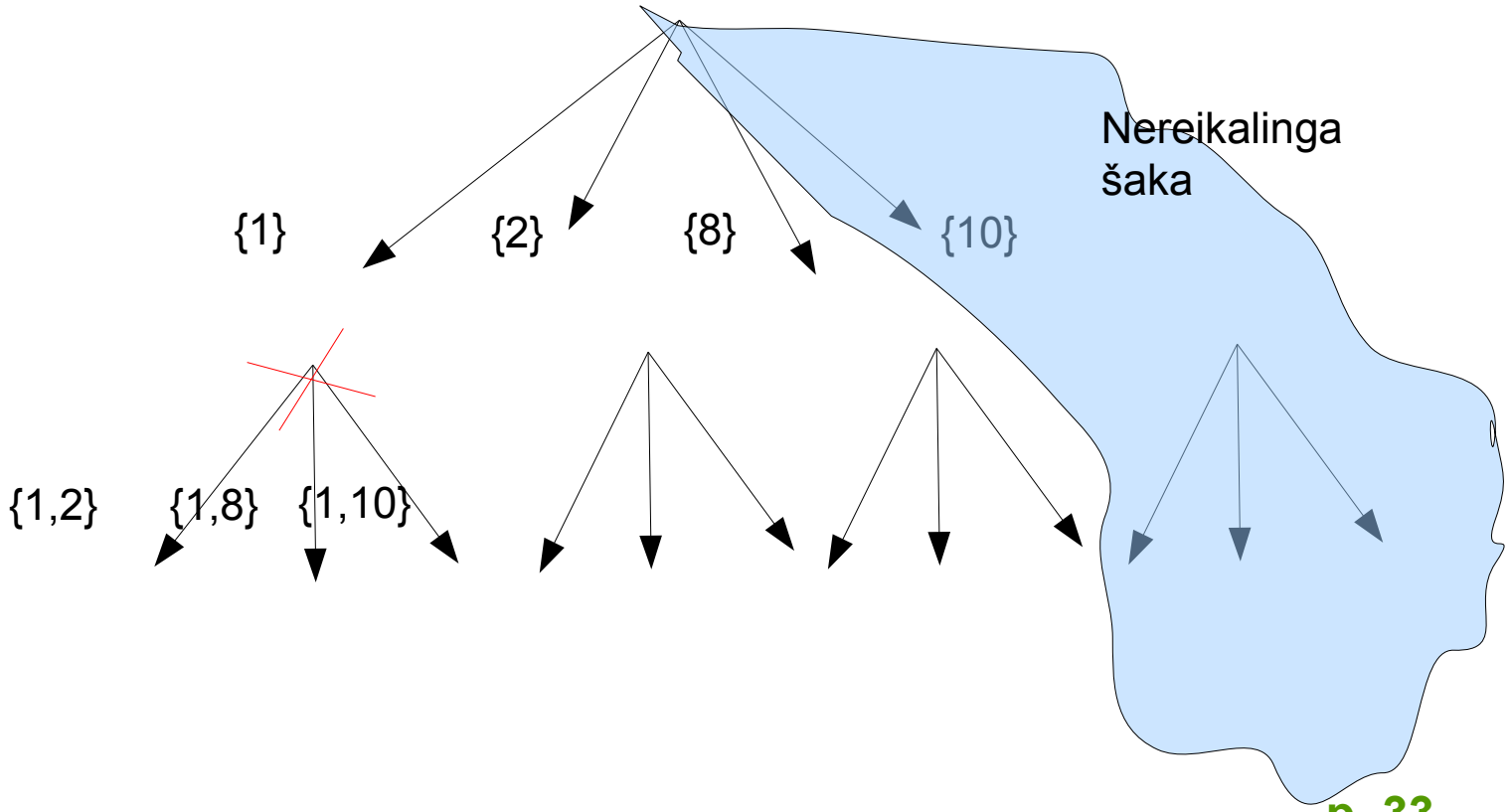
- **Panaudosime šakų ir rėžių metodą**
  - Jo pagrindinė idėja bendru atveju: jeigu skaičiavimo medžio mazge matome, kad sprendinio „jau nesigaus“, tai nėra prasmės leistis žemyn – reikia grįžti prie prieš tai buvusio mazgo

Pavyzdys („iš kitos operos“).

- **Tarkime, kad turime rasti skaičių aibės {n1,n2,...,nk} poaibį, kurio elementų suma yra duotas skaičius S**
  - Jeigu kuriame nors paieškos medžio (kita skaidrė) mazge suma jau viršijo S, tai nėra prasmės ieėkoti toliau;

Paieškos medis

Tarkime kad S = { 1,2,8,10 }.Reikia rasti poaibį, kurio elementų suma yra 9:



Praktikoje *tinkamas* restrikcijos padengimo algoritmas

„Tinkamas“ algoritmas (S. Skiena) Pavyzdys

$$\Delta X = \{2,2,3,3,4,5,6,7,8\}$$

„Tinkamas“ algoritmas (S. Skiena) Pavyzdys

$X=\{0,10\}$, $\Delta X = \{2,2,3,3,4,5,6,7,8\}$

Ir lieka:

arba x4 = 8 arba x2 = 2. kadangi abu atvejai simetriniai – imame x2 = 2;

Šaliname iš ΔX visus atstumus tarp x2 ir X narių.

„Tinkamas“ algoritmas (S. Skiena) Pavyzdys

X={0,2,10}, ΔX = {2,3,3,4,5,6,7}

Ir lieka:

arba x4 = 7 arba x3 = 3.

Kadangi x3-x2=1, o 1 nėra aibėje Δx, tai x4=7.

„Tinkamas“ algoritmas (S. Skiena) Pavyzdys

$X=\{0,2,7,10\}$, $\Delta X = \{2,3,4,6\}$

Ir lieka:

arba x3 = 4, arba x3 = 6.

Jeigu x3=6,tai x4-x3 = 1 turėtų priklausyti Δx. Lieka x3=4

Rezultatas

$$X = \{0, 2, 4, 7, 10\}$$

Formalus algoritmas

PADENGIMASSUGRIZIMAIS(ΔX)

1. plotis ← Max(ΔX)
2. TRINK(plotis, ΔX)
3. X ← (0, plotis)
4. BANDYKPADENGTI(ΔX, X)

Rekursinis, žr. kitą skaidrę

Formalus algoritmas

Žymėjimas: $\Delta(y,X)$ – visi atstumai tarp y ir visų X elementų:

$$\Delta(2,\{1,3,4,5\}) = \{1,1,2,3\}$$

Formalus algoritmas

BANDYKPADENGTI(ΔX, X)

1. Jeigu ΔX yra tuščia
   Rašyk X
   Baigti
2. y ← Max(ΔX)
3. Jeigu Δ(y,X) yra poaibis iš ΔX
   Pridėk y prie X ir pašalink visus ilgius Δ(y,X) iš ΔX
   BANDYKPADENGTI(ΔX, X)
   Išmesk y iš X ir pridėk atgal visus ilgius Δ(y,X) iš ΔX
4. y1 ← plotis - y
5. Jeigu Δ(y,X) yra poaibis iš ΔX
   Pridėk y1 prie X ir pašalink visus ilgius Δ(y1,X) iš ΔX
   BANDYKPADENGTI(ΔX, X)
   Išmesk y1 iš X ir pridėk atgal visus ilgius Δ(y1,X) iš ΔX
6. **Baigti**

Pratimas

Tegul ΔX yra {2,3,3,3,5,6,6,8,9,11}. Raskit X pagal Skienos algoritmą

Greitis

Jeigu turėtume TIK vieną altternatyvą, tai

$$T(n)=T(n-1)+ O(n),$$

O jeigu kiekvieną kartą – dvi alternatyvos, tai

$$T(n)=2*T(n-1)+O(n)$$

Klausimas

Kokią išvadą apie galima iš to padaryti?

Pastaba: 2002 metais atrastas polinominis algoritmas šiam uždaviniui (Maurice Nivat)

A. Duarat, Y. Gerard, and M. Nivat. The chords problem. *Theoretical Computer Science*, 282:319–336, 2002.


010
011
000
ATG
CTT
011
...
GTT
GTC
GTA
GTG
...
TAA
....
000
GTA
GTG
...
TAA
....
GTA
GTG
...
TAA
....

Motyvo paieškos problema

Įvadas

- Reguliatoriniai motyvai DNR sekose.
- Kaip patikrinti imuninės sistemos atsaką.
- Transkripcijos faktoriai ir reguliatoriniai motyvai.
- Motyvai tekstuose (bendrai)

Pvz. (nebiologinis). Kas čia parašyta?

HSN OBTGBZ PUBY J KXJLW DTMBD GOTMNU TG JKBAH PTPHNNZ HTYNG YBUN OBHNZH HSJZ JZ NRAJX JYBAZH PUBY J UJHHXNGZJWN.

Atsakymas

The poison from a black widow spider is about fifteen times more potent than an equal amount from a rattlesnake.

Motyvacija.

Koks ilgiausias bendras fragmentas gali būti rastas visose eilutėse?

CGGGGCTGGGTCGTCACATTCCCCTTTCGATA

TTTGAGGGTGCCCAATAACCAAAGCGGACAAA

GGGATGCCGTTTGACGACCTAAATCAACGGCC

AAGGCCAGGAGCGCCTTTGCTGGTTCTACCTG

AATTTTCTAAAAAGATTATAATGTCGGTCCTC

CTGCTGTACAACTGAGATCATGCTGCTTCAAC

TACATGATCTTTTGTGGATGAGGGAATGATGC

Motyvacija.

Koks ilgiausias fragmentas gali būti rastas visose eilutėse, jeigu galimos mutacijos?

CGGGGCTGGGTCGTCACATTCCCCTTTCGATA

TTTGAGGGTGCCCAATAACCAAAGCGGACAAA

GGGATGCCGTTTGACGACCTAAATCAACGGCC

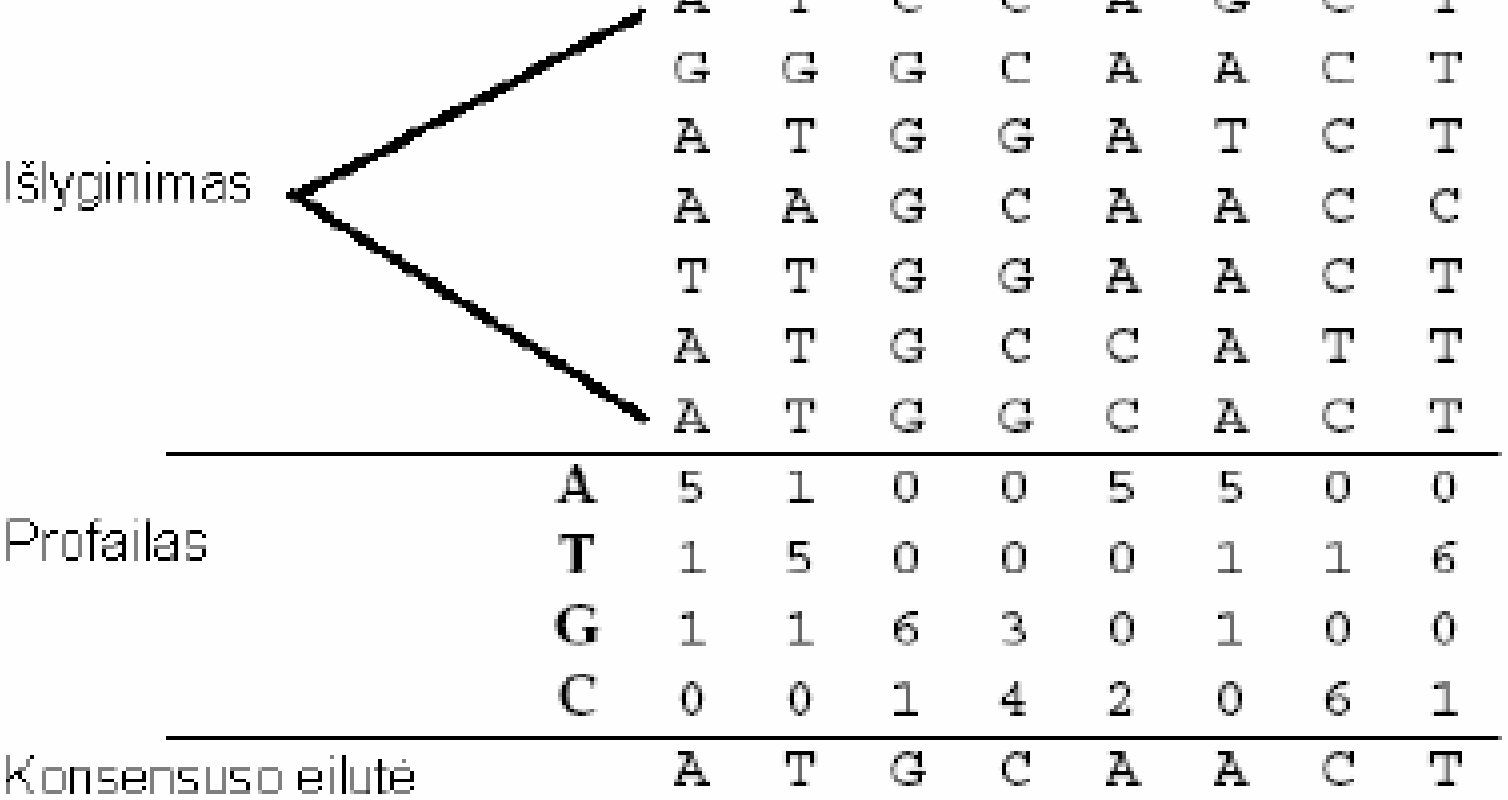
AAGGCCAGGAGCGCCTTTGCTGGTTCTACCTG

AATTTTCTAAAAAGATTATAATGTCGGTCCTC

CTGCTGTACAACTGAGATCATGCTGCTTCAAC

TACATGATCTTTTGTGGATGAGGGAATGATGC

Kaip elgtis su mutacijomis? Konsensuso eilutė



Superpozicijos ir profiliai

Kiekvienai superpozicijai skaičiuojame konsensuso eilutės svorį: mūsų atveju tai bus 5+5+6+4+5+5+6+6 = 42

Motyvo paieškos problema

- **Duota *t* DNR sekų,kurių visų ilgis yra *n*. Reikia rasti *l*-merų aibę – po vieną iš kiekvienos sekos, tokią, kad ji maksimizuoja konsensuso eilutės svorį;**
- **Algoritmo argumentai:**
  - *t x n matrica, l – ieškomo fragmento (motyvo) ilgis*
- ***Algoritmo rezultatai:***
  - *Pradinių pozicijų s1,s2,..., st masyvas.*

Klausimas

- **Jeigu norėtume spręsti „brute-force“ metodu: perrinkti visus variantus, tai kiek jų yra?**

– Ats.: $(n - l + 1)^t$

$$(1,1,1,\ldots,1,1) \rightarrow (1,1,1,\ldots,1,2) \rightarrow \ldots \rightarrow (1,1,1,\ldots,1,n-l+1)$$

$$\rightarrow (1,1,1,\ldots,2,1) \rightarrow (1,1,1,\ldots,2,2) \rightarrow \ldots\ldots\ldots \rightarrow (n-l+1,n-l+1,n-l+1)$$

Variantų perrinkimas

```
MotyvoPaieška_1(DNR, t, n, l)
 maxSvoris ← 0
 for (s1, . . . , st) from (1,...,1) to (n-l+1,...,n-l+1)
    if Svoris(s,DNR) > maxSvoris
        maxSvoris ← Svoris(s,DNR)
        geriausiasMotyvas ← (s1,s2,...,st)
 return geriausiasMotyvas
```

Klausimas

Kaip perrinkinėti, t.y., kaip parašyti minėtą ciklą for?

```
VisiLapai (s,t,n,l)
    s ← (1, . . . , 1)
    while forever
        rašyk s
        s ← KitasLapas(s,t,n, k)
        if s = (1, 1, . . . , 1)
            return
```

... kitas lapas...

```
KitasLapas(s,t,n,l)
    k ← n − l + 1
    for i ← t to 1
        if s_i < k
            s_i ← s_i + 1
            return s
        s_i ← 1
return s
```

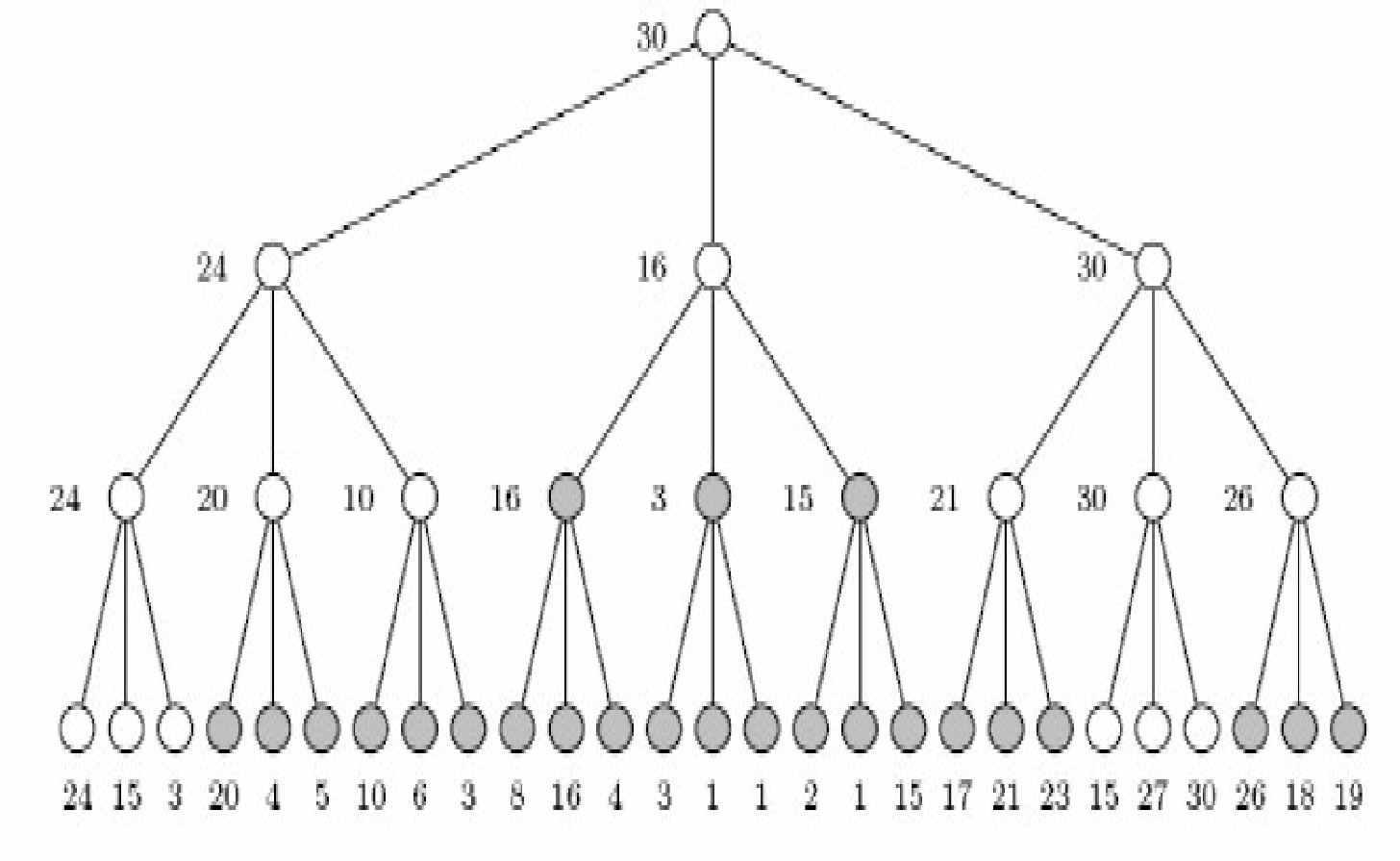
Pratimas

- **Paaiškinkite kaip veikia paskutiniųjų dviejų skaidrių algoritmas**
- **Kaip jį galima būtų perrašyti panaudojant mod operaciją?**

Klausimas?

- **Kodėl šis algoritmas lėtas?**
- **Kaip galima būtų jį pagreitinti?**
  - Pagrindinė idėja perrinkimo uždavinių pagreitinimo:
    - Pabandyti išsivaizduoti visus variantus kaip tam tikrą kryptinį medį, kuriame paieška prasideda nuo šaknies, o variantai – lapai. Tada jeigu yra sugalvota tam tikra šakų atmetimo strategija, tai galima žymiai pagreitinti perrinkimą

Variantų medis



Kaip gaunamas medis?

- **Tarkime, kad visi variantai yra tiesiog aibių A1xA2x ...An Dekarto sandauga. Tada galima vaizduoti medį, iš kurio šaknies išeina |A1| šakų, iš jų - |A2| šakų ir t.t.**
- **Pratimas. Sukonstruokit pilnojo perrinkimo medį uždaviniui: rasti visus triženklius skaičius, kurių skaitmenų sandauga yra 15. (A1={1..9},A2=A3={0..9}).**
  - Pažymėkite sukonstruotame medyje „beviltiškas“ šakas.

Kaip atlikti paieška medyje?

```
MotyvoPaieška_2(DNR, t, n, l)
 s ← (1, . . . , 1)
 maxSvoris ← 0
 i ← 1
 while i > 0
    if i < t
       (s, i) ← KitaViršūnė(s,t,n,l,i)
    else
       if Svoris(s,DNR) > maxSvoris
          maxSvoris ← Svoris(s,DNR)
          geriausiasMotyvas ← (s1,s2,...,st)
          (s, i) ← KitaViršūnė(s,t,n,l,i)
 return geriausiasMotyvas
```

... kita viršūnė

```
KitaViršūnė(s, t, n, l, i)
    k ← n − l + 1
    if i < t
        s_{i+1} ← 1
        return (s, i + 1)
    else
        for j ← t to 1
        if s_j < k
            s_j ← s_j + 1
            return (s, j)
    return (s, 0)
```

Kaip elgtis mūsų atveju?

```
MotyvoPaieška_3(DNR, t, n, l)
 s ← (1, . . . , 1)
 maxSvoris ← 0
 i ← 1
 while i > 0
    if i < t
       maxGeriausiasSvoris ← Svoris(s, i,DNR) + (t − i) · l
       if maxGeriausiasSvoris < maxSvoris
          (s, i) ← Atmesk(s,t,n,l,i)
       else
          (s, i) ← KitaViršūnė(s,t,n,l,i)
    else
       if Svoris(s,DNR) > maxSvoris
          maxSvoris ← Svoris(s)
          geriausiasMotyvas ← (s1,s2,...,st)
       (s, i) ← KitaViršūnė(s,t,n,l,i)
 return geriausiasMotyvas
```

... atmesk

```
Atmesk(s,t,n,l,i)
    k ← n - l + 1
    for j ←  i to 1
        if s_j < k
            s_j ← s_j + 1
            return (s, j)
    return (s, 0)
```

Kaip elgtis mūsų atveju?

```
MotyvoPaieška_3(DNR, t, n, l)
 s ← (1, . . . , 1)
 maxSvoris ← 0
 i ← 1
 while i > 0
    if i < t
       maxGeriausiasSvoris ← Svoris(s, i,DNR) + (t − i) · l
       if maxGeriausiasSvoris < maxSvoris
          (s, i) ← Atmesk(s,t,n,l,i)
       else
          (s, i) ← KitaViršūnė(s,t,n,l,i)
    else
       if Svoris(s,DNR) > maxSvoris
          maxSvoris ← Svoris(s)
          geriausiasMotyvas ← (s1,s2,...,st)
       (s, i) ← KitaViršūnė(s,t,n,l,i)
 return geriausiasMotyvas
```

... atmesk

```
Atmesk(s,t,n,l,i)
    k ← n − l + 1
    for j ← i to 1
        if s_j < k
            s_j ← s_j + 1
            return (s, j)
    return (s, 0)
```

Godieji algoritmai

Kas yra godieji algoritmai?

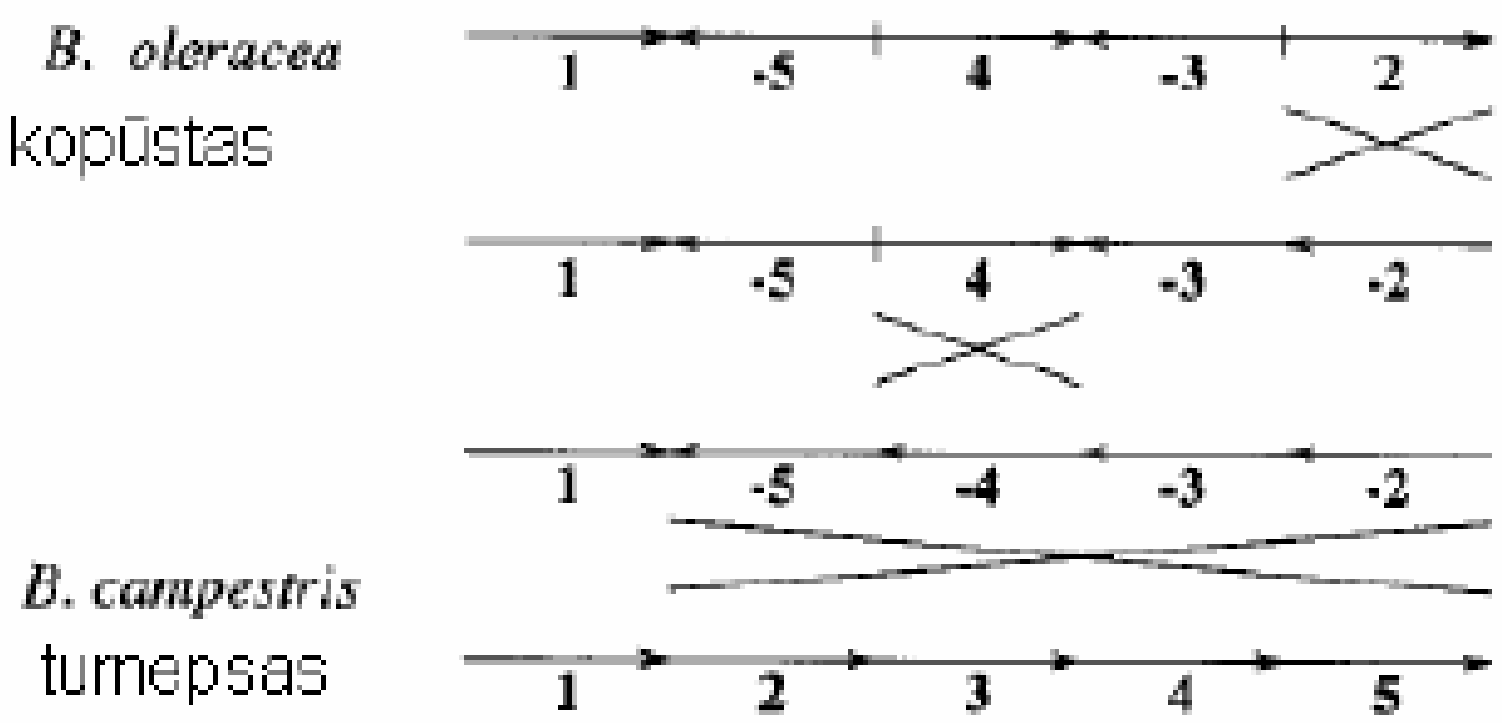
- **Tai algoritmai, kurie remiasi kokią paprastą (lokalią) optimizavimo einamosios būsenos idėją**
- **Pavyzdys: „burbulo“ rūšiavimo algoritmas;**
- **Labiausiai užimtam(-i) žmogui/auditorijai tvarkaraštį reikia sukurti pačioje pradžioje**
- **etc.**

Genomo perstatos

Motyvacija 1.

Waardenburgo sindromas, genomo perstatos, inversijos (arba kas bendra tarp žmonių ir pelių).

Turnepsas ir kopūstas



Genomo perstatos: motyvacija

Evoliucijos metu genų nuskaitymo tvarka gali pasikeisti. Mūsų domins kol kas tik *inversijos*. Uždavinys – gauti *trumpiausią* inversijų seką, kuri perveda viena genomo elementą prie kito. Tame reikalingas tam tikras formalizavimas.

Keitinio π inversija $\rho(i,j)$ vadinsime naują keitinį, kurio elementai nuo *i* iki *j* eina priešinga kryptimi:

$\pi_1, \pi_2, \ldots \pi_i \ldots \pi_j \ldots \pi_n * \rho(i,j) \rightarrow$

$\pi_1, \pi_2, \ldots \pi_j \ldots \pi_i \ldots \pi_n$

Pavyzdys: $(1\ 2\ 3\ 5\ 4) * \rho(2,4) = (1\ 5\ 3\ 2\ 4)$

Inversijų atstumo problema: Duotiems dviems keitiniams π ir σ reikia rasti trumpiausią inversijų seką $\rho_1, \rho_2, ..., \rho_t$ po kurios nuo vieno keitinio pereinama prie kito. *Tos sekos narių skaičius vadinamas atstumu tarp* π ir σ *ir žymimas* $d(\pi, \sigma)$. Paprastai patogu spręsti šią problemą šiek tiek ją performulavus: vietoje σ galima imti *(1 2 3 ... n)* keitinį ir tada problema tampa *rikiavimu inversijų pagalba.*

„Prastas“ algoritmas

```
PrastasInversijuRikiavimas(p)
1 for  i ← 1 to n - 1
2      j ← elemento i pozicija keitinyje p  (t.y., p_j = i)
3      if  j ≠ i
4         p ← p * r(i, j)
5         rašyk p
6      if p yra identiškas keitinys
7        return
```

Pavyzdys

Pavyzdys: turime 6 1 2 3 4 5 .

Algoritmas:

1. 1 6 2 3 4 5
2. 1 2 6 3 4 5
3. 1 2 3 6 4 5
4. 1 2 3 4 6 5
5. 1 2 3 4 5 6

Bet:

1. 5 4 3 2 1 6

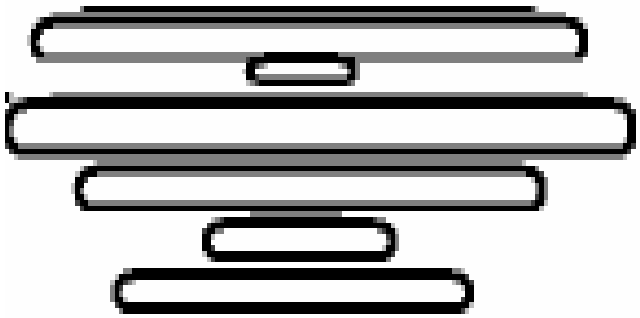
1 2 3 4 5 6

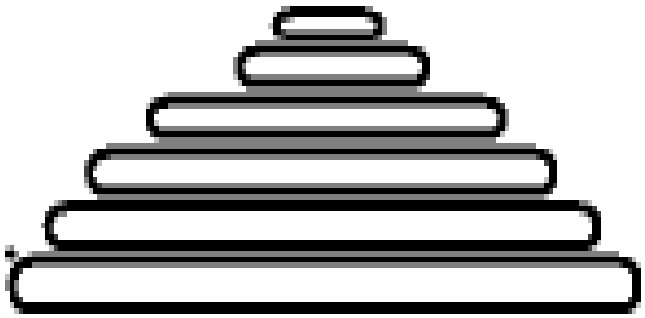
Pratimas

Pratimas. Kaip veiktų prastas inversijų rikiavimas tokiam atvejui: 3 5 7 1 2 4 6

`PrastasInversijuRikiavimas` yra tikrai prastas, nes jam reikia $n - 1$ inversijos keitiniui (n 1 2 3 ... $n - 1$), o reliai šiam keitiniui reikia 2 inversijų.

Panaši problema: „Blynų dėstymas“





Šiuo atveju leidžiama daryti *tik* priešdėlių inversijas, t.y., apvertinėti tik viršutinę dalį boštelio.

Apie blynų rikiavimą

Analogas viršutinio algoritmo atlieka tai blogiausiu atveju per $2(n-1)$ inversiją (pvz., 1 2 3 6 4 5 → 6 3 2 1 4 5 → 5 4 1 2 3 6 → 3 2 1 4 5 6 → 1 2 3 4 5 6).

Pratimas. Suformuluokit šį algoritmą.

W.Gates ir Ch.Paradimitriuou : blynus galima surikiuoti per daugiausiai (5/3)(n+1) inversiją

Sąvokos

Aproksimacijos algoritmai. <...>
Aproksimacijos koeficientai. <...>

Minimizacijos problema	Maksimizacijos problema
$\max_{\lvert\pi\rvert=n} \frac{A(\pi)}{OPT(\pi)}$	$\min_{\lvert\pi\rvert=n} \frac{A(\pi)}{OPT(\pi)}$

„Prasto“ inversijų rikiavimo algoritmo aproksimacijos koeficientas

`PrastasInversijuRikiavimas` turi aproksimacijos koeficientą, kuris yra bent $(n-1)/2$ (bet gal ir didesnis?).

„Protingesnis godumas“

Sąvokos: Išplėstinis keitinys. Kaimyniniai elementai. Lūžio taškai. Kaimynų atkarpa.

Pavyzdys.

0 2 1 3 5 6 7 4 8

...tęsinys (0 2 1 3 5 6 7 4 8)

Kaimyniniai elementai : 2 1, 5 6, 6 7

Lūžio taškai: 0 2, 1 3, 3 5, 7 4, 4 8.

Kaimynų atkarpa (atkarpa tarp gretimų lūžio taškų): 0, 2 1, 3, 5 6 7, 4, 8

Naujo algoritmo idėja: požingsniui mažinti lūžio taškų skaičių.

Inversija gali pašalinti daugiausiai du lūžio taškus. Todėl

$d(\pi) \geq b(\pi)/2$, kur $b(\pi)$ – lūžio taškų skaičius

Idėja: reikia remtis lūžio taškais

```
LūžiųTaškųInversijųRikiavimas(π)
1 while b(π ) > 0
2     rask inversiją ρ, kuri sumažina lūžio taškų skaičių b(π • ρ)
3     π ← π • ρ (i, j)
4     rašyk π
5 return
```

Kas yra blogai šioje idėjoje?

...tiesiog reikia idėją pagrįsti

Teorema 1. Jeigu keitinys π turi bent vieną mažėjančią kaimynų atkarpą, tai egzistuoja ρ, kuris sumažina lūžio taškų skaičių (t.y. $b(\pi \bullet \rho) < b(\rho)$)

$(0\ 8\ 2\ 7\ 6\ 5\ 1\ 4\ 3\ 9)$ $\quad b(\pi) = 6$

$(0\ 2\ 8\ 7\ 6\ 5\ 1\ 4\ 3\ 9)$ $\quad b(\pi) = 5$

$(0\ 2\ 3\ 4\ 1\ 5\ 6\ 7\ 8\ 9)$ $\quad b(\pi) = 3$

$(0\ 4\ 3\ 2\ 1\ 5\ 6\ 7\ 8\ 9)$ $\quad b(\pi) = 2$

$(0\ 1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9)$ $\quad b(\pi) = 0$

Patobulintas algoritmas

```
LūžiųTaškųInversijuRikiavimasX(π)
1 while b(π) > 0
2     if π mažėjančią kaimynų atkarpą
3         rask inversiją ρ, kuri      mažina b(π • ρ)
4     else
5         rask inversiją ρ, kuri "padaro" bent vieną mažėjančią  kaimynų atkarpą
6         π ← π • ρ
7     rašyk π
8 return
```

Apie patobulinto algoritmo aproksimacijos koeficientą

Teorema 2. `LūžiųTaškųInversijųRikiavimasX`(p) yra korektiškas aproksimacijos algoritmas, kurio aproksimacijos koeficientas yra 4

Kadangi $d(\pi) \geq b(\pi)/2$, o blogiausiu atveju algoritmas turi daryti po du žingsnius (sudaryti mažėjančią atkarpą ir mažinti lūžio taškų skaičių), tai aproksimacijos koeficientas neviršija $2b(\pi)/d(\pi) \leq 2b(\pi)/(b(\pi)/2) = 4$.

Nes kiekviena inversija
gali panaikinti max du lūžio taškus

Dar kartą apie motyvų paiešką

```
GreedyMotifSearch(DNA, t, n, l)
 1  bestMotif ← (1, 1, ..., 1)
 2  s ← (1, 1, ..., 1)
 3  for s_1 ← 1 to n − l + 1
 4      for s_2 ← 1 to n − l + 1
 5          if Score(s, 2, DNA) > Score(bestMotif, 2, DNA)
 6              BestMotif_1 ← s_1
 7              BestMotif_2 ← s_2
 8  s_1 ← BestMotif_1
 9  s_2 ← BestMotif_2
10  for i ← 3 to t
11      for s_i ← 1 to n − l + 1
12          if Score(s, i, DNA) > Score(bestMotif, i, DNA)
13              bestMotif_i ← s_i
14      s_i ← bestMotif_i
15  return bestMotif
```


Pabaiga