Curs 4.

Tehnici algoritmice utilizate în bioinformatică
Problema maparii fragmentelor

Biblio:

cap. 4 din “An introduction to Bioinformatics algorithms”, N.Jones, P. Pevzner

cap 2 din Computational Molecular Biology. An Algorithmic Approach, P. Pevzner

Tipuri de probleme intalnite in bioinformatica

- Reconstituirea secvențelor (ADN) pornind de la fragmente (restriction mapping)
- Identificarea șabloanelor (motif finding) în secvențe ADN/aminoacizi
- Alinierea secvențelor ADN / aminoacizi
  - Locală
  - Globală
  - Multiplă
- Construirea arborilor filogenetici
- Predicția structurii proteinelor

Tehnici algoritmice utilizate in bioinformatica

- Tehnica căutarii exhaustive (metoda forței brute)
  - Sistematizarea căutarii: backtracking
  - Limitarea căutarii: branch and bound

- Tehnica căutarii local optimale (greedy)

- Programare dinamică

Obs. Tehnicile de mai sus se încadrează în două categorii:

- Exacte (căutare exhaustivă, programare dinamică) – costisitoare
- Aproximative (greedy) - eficiente

Tehnica cautarii exhaustive

- Specific:
  - Se explorează întreg spațiul soluțiilor
  - Conduce la algoritmi simpli însă de cele mai multe ori ineficienți (complexitate exponențială)

- Aplicații în bioinformatică:
  - Maparea fragmentelor (“restriction mapping”)
  - Identificarea motivelor (șabloanelor) în secvențe ADN (“motif finding”)

Maparea fragmentelor (restriction mapping)

Contextul problemei:

- In 1970 s-a descoperit (studiindu-se bacteria *Haemophilus influenzae)* că o anumită enzimă (HindII) "taie" molecula ADN în fiecare poziție unde apare o secvență de forma GTGCAC sau GTTAAC;
- o alta enzimă (EcoRI) secționează ADN în pozițiile în care apare GAATTC
- In felul acesta o secvența ADN lungă este "secționată" în fragmente mai scurte ale căror lungimi pot fi determinate (prin electroforeză în gel)

- Harta fragmentelor ("restriction map") arată pozițiile în secvența ADN completă unde s-a realizat secționarea
- Dacă secvența ADN inițială ar fi cunoscută atunci construirea hărții fragmentelor ar fi o simplă problemă de căutare de subșiruri într-un șir
- Secvențierea ADN-ului s-a realizat însă abia spre sfârșitul anilor 1990 motiv pentru care la momentul la care a fost enunțată problema, harta fragmentelor trebuia realizată fără a se cunoaște secvența ADN

Maparea fragmentelor (restriction mapping)

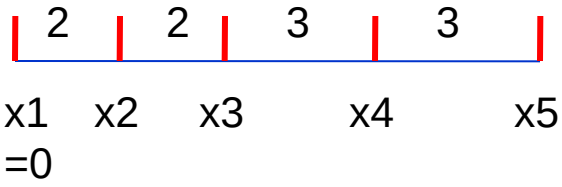
Formulare generală a problemei

- Se consideră o secvență (sau mai multe copii ale aceleiași secvențe) secționată într-un anumit număr de fragmente.

- Se cunosc distanțe dintre punctele de secționare
- Se caută pozițiile de secționare

In funcție de setul de distanțe cunoscut există mai multe variante:

- Complete Digest Problem
- Double Digest Problem
- Partial Digest Problem



Complete Digest Problem:

- Se cunosc doar distanțele dintre punctele consecutive de secționare
- D(X)= {2,2,3,3}

Maparea fragmentelor (restriction mapping)

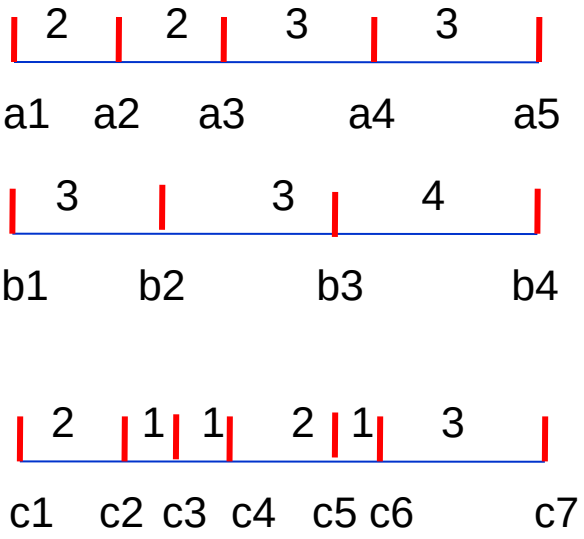
Formulare generală a problemei

- Se consideră o secvență (sau mai multe copii ale aceleiași secvențe) secționată într-un anumit număr de fragmente.

- Se cunosc distanțe dintre punctele de secționare
- Se caută pozițiile de secționare

Double Digest Problem:

- Se utilizeaza 3 secvente si două enzime de secționare A si B
- Se cunosc 3 seturi de distante:
  - $D_A(X)= \{2,2,3,3\}$
  - $D_B(X)=\{3,3,4\}$
  - $D_{AB}(X)=\{1,1,1,2,2,3\}$



Maparea fragmentelor (restriction mapping)

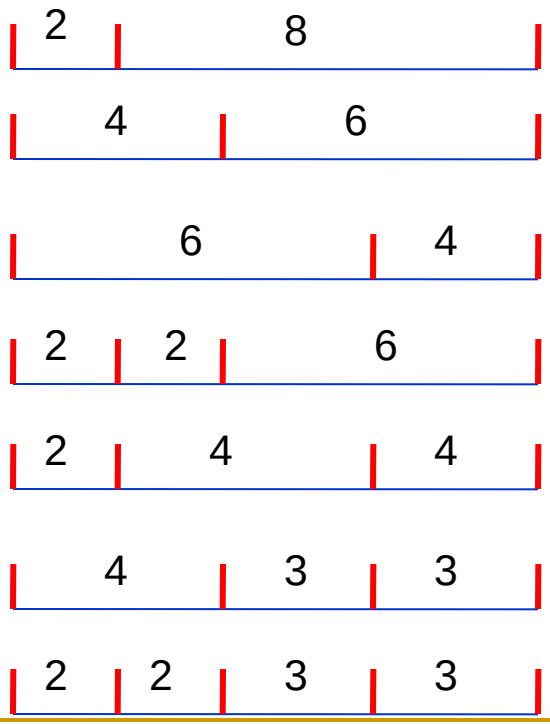
Formulare generală a problemei

- Se consideră o secvență (sau mai multe copii ale aceleiași secvențe) secționată într-un anumit număr de fragmente.
- Se cunosc distanțe dintre punctele de secționare
- Se caută pozițiile de secționare

Partial Digest Problem:

- Se cunosc distanțele dintre oricare două puncte de secționare
- D(X)= {2,2,3,3,4,5,6,7,8,10}

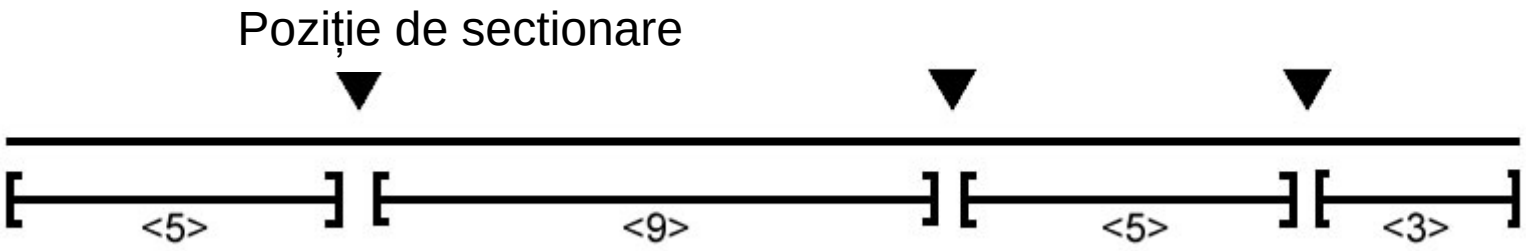
- Dpdv experimental necesită secționarea mai multor copii ale secvenței folosind timpi diferiți (cel mai scurt timp permite ca enzima să realizeze o singură secționare, cel mai lung timp permite secționarea în toate pozițiile posibile) – dificil de realizat



Maparea fragmentelor (restriction mapping)

- Varianta 1: “complete digest problem”

  - Secvența este secționată în fiecare poziție unde poate acționa enzima; fragmentele vor corespunde unor locații de secționare consecutive
  - In construirea hărții fragmentelor se pornește de la setul de lungimi ale fragmentelor și se dorește identificarea pozițiilor de secționare



Problema: pornind de la lungimile tuturor fragmentelor: {3,5,5,9} să se determine pozițiile de secționare

Maparea fragmentelor (restriction mapping)

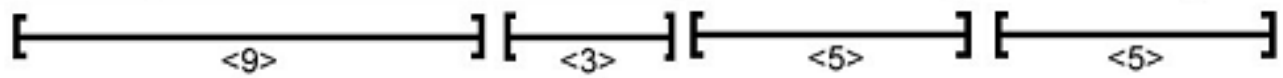
- Formularea problemei (“complete digest problem”):

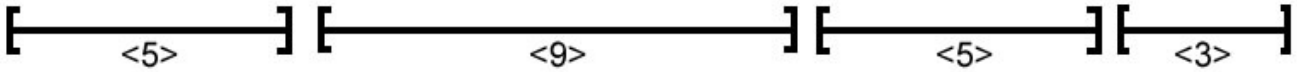
Fie $X=\{x_1=0,x_2,\ldots,x_n\}$ pozițiile necunoscute ale punctelor de secționare și

$D(X) = \{x_{i+1}-x_i,\ 1<=i<n\}$ setul cunoscut al distanțelor dintre oricare două perechi de poziții consecutive.

Să se determine X pornind de la D(X).

- Observație:
  - Problema nu are soluție unică (sunt necesare informații suplimentare pentru a identifica pozițiile de secționare)





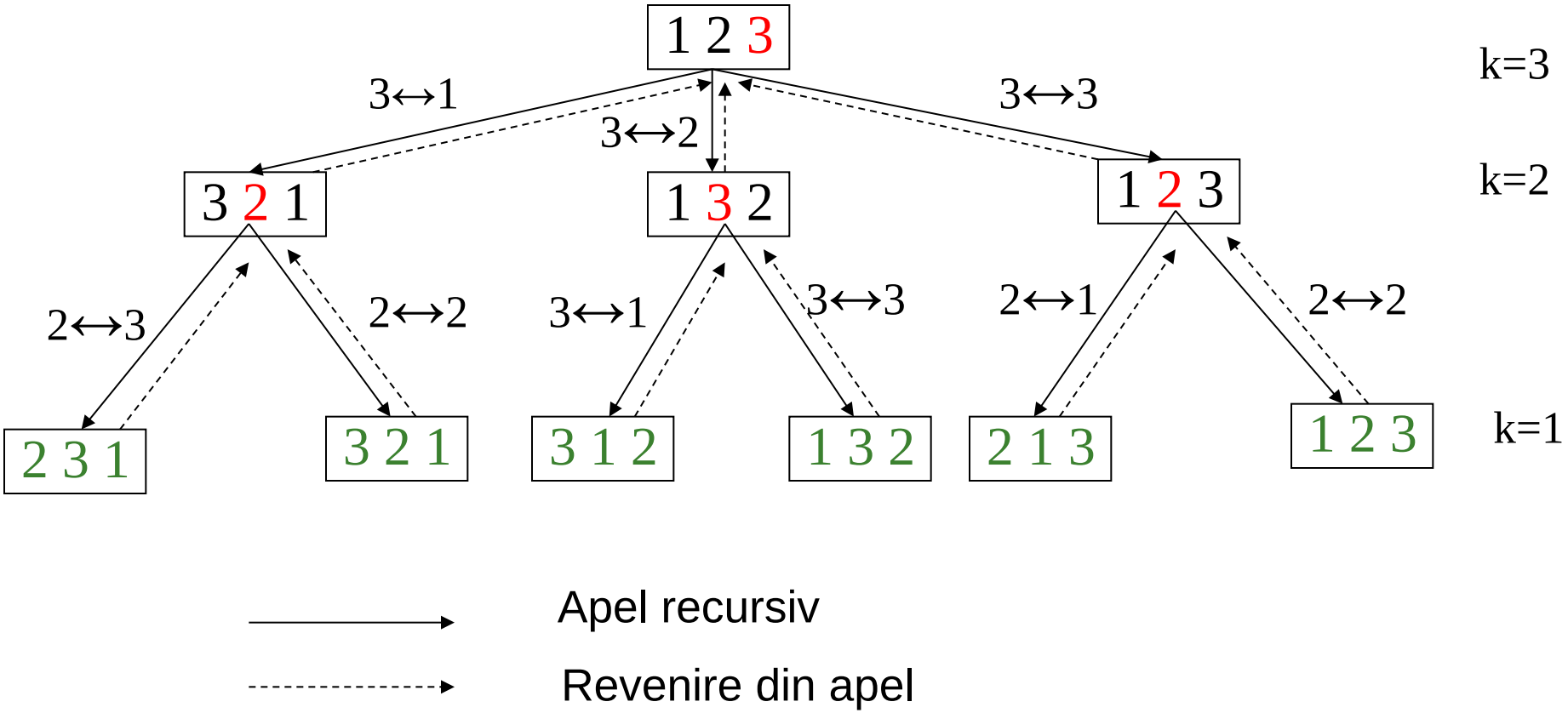
Maparea fragmentelor (restriction mapping)

- Rezolvare:
  - Este suficient să se genereze permutările setului de distanțe $\{d_1,d_2, \ldots,d_n\}$
  - Pozițiile de secționare (inclusiv extremitățile) vor fi: $\{0,d_1,d_1+d_2, \ldots., d_1+d_2+\ldots+d_n\}$

- Generarea permutărilor:
  - Metoda reducerii (decrease and conquer): generarea permutărilor de ordin n se reduce la generarea permutărilor de ordin (n-1). Se pornește de la permutarea identică.
  - Metoda căutarii cu revenire (backtracking): componentele permutării se completează succesiv începând de la prima; după ce s-au încercat toate valorile posibile pentru o componentă se revine la componenta anterioară.

Maparea fragmentelor (restriction mapping)

- Ilustrare pentru n=3 (metoda reducerii)



Maparea fragmentelor (restriction mapping)

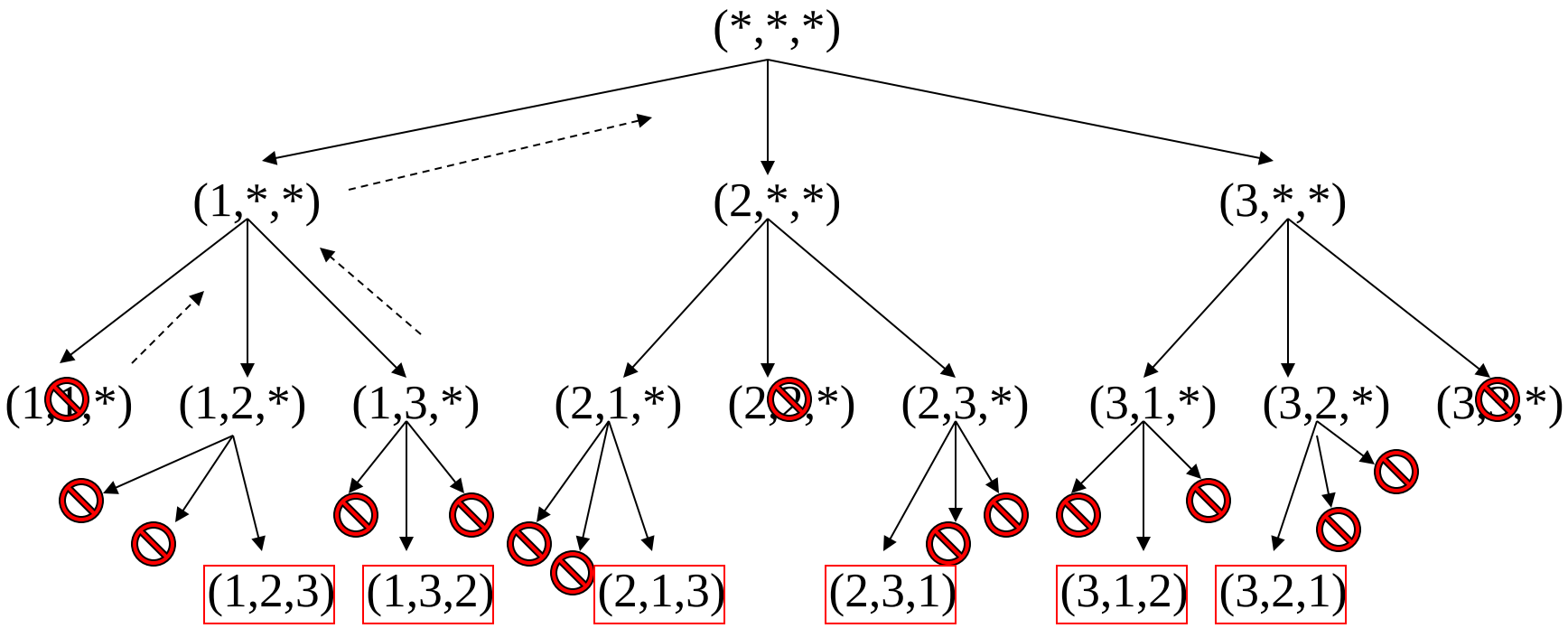
- Algoritm de generare a permutărilor (metoda reducerii – “decrease and conquer”):

```
perm(k)
  IF k==1 THEN WRITE s[1..n]
  ELSE
       FOR i=1,k DO
            s[i] ↔s[k]          // interschimbare
            perm(k-1)
            s[i] ↔s[k]
       ENDFOR
  ENDIF
```

- Se pornește de la permutarea identică
- s[1..n] e variabilă globală
- Apel: perm(n)

Maparea fragmentelor (restriction mapping)

- Ilustrare pt. n=3 (metoda backtracking):



Maparea fragmentelor (restriction mapping)

- Algoritm de generare a permutărilor (metoda backtracking):

```
perm(k)
IF k==n+1 THEN WRITE s[1..n]
ELSE
   FOR i=1,n DO
      s[k]=i
      IF valid(s[1..k])
          THEN perm(k+1)
      ENDIF
   ENDFOR
ENDIF
```

Obs: s[1..n] e variabilă globală

Apel: perm(1)

Funcție pentru a verifica dacă o soluție parțială este validă sau nu

```
valid(s[1..k])
FOR i=1,k-1 DO
   IF s[k]==s[i]
      THEN RETURN FALSE
   ENDIF
ENDFOR
RETURN TRUE
```

Maparea fragmentelor (restriction mapping)

Observație:

- Pentru a reduce numărul de variante posibile se pot folosi informații suplimentare cum ar fi:
  - Distantele dintre punctele de sectionare obtinute aplicând două enzime atât separat cât și simultan (3 seturi de distanțe) - Double Digest Problem
  - Distanțele dintre toate perechile de puncte de secționare nu doar între cele consecutive (in cazul a n puncte vor fi C(n,2) distanțe) -Partial Digest Problem)

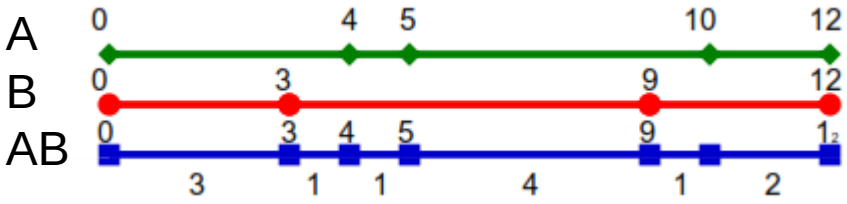
Maparea fragmentelor (restriction mapping)

Varianta 2: „double digest problem”

- Intrare: 3 seturi de distante; Iesire: pozitiile de sectionare
- Abordare simplă dar ineficientă: tehnica forței brute

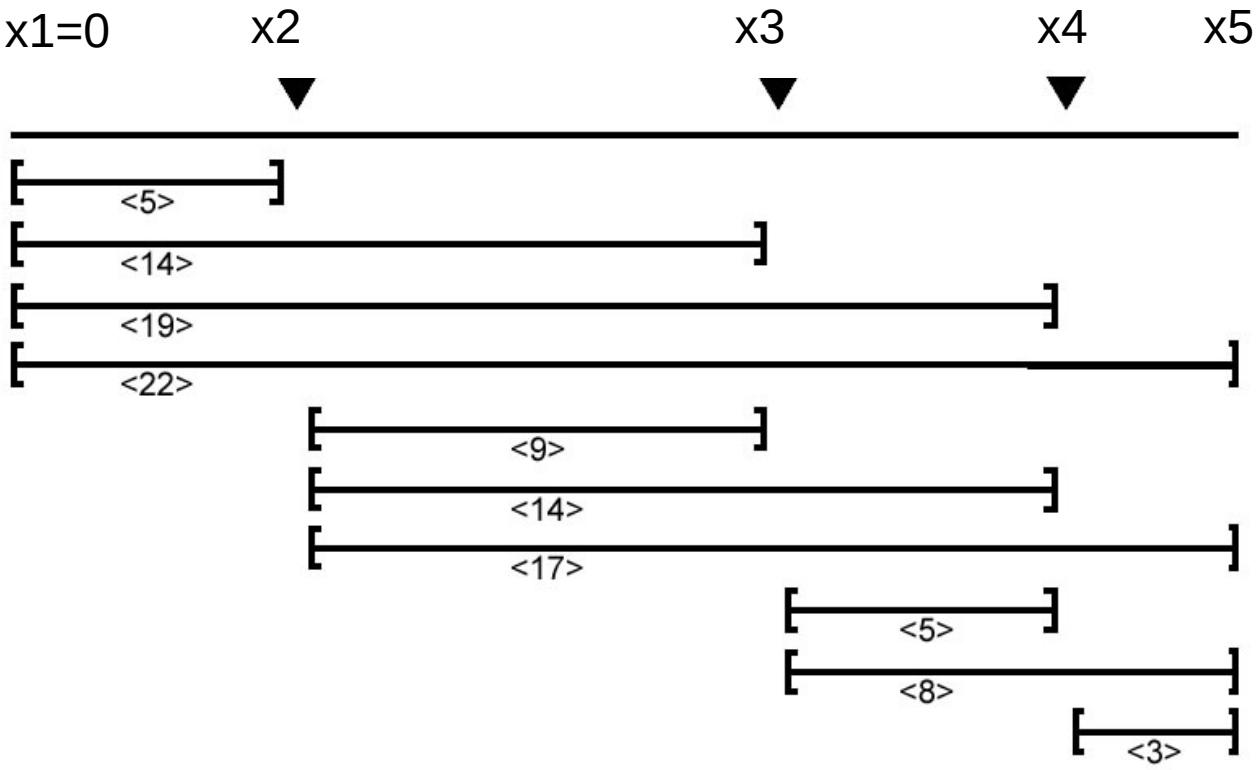
FOR „fiecare permutare p_A a setului de distante d_A” DO
  FOR „fiecare permutare p_{AB} a setului de distante d_B” DO
    IF p_A este compatibila cu p_{AB} THEN
      FOR „fiecare permutare p_B a setului de distante d_B” DO
        IF p_B este compatibila cu p_{AB} THEN return (p_A , p_B , p_{AB})

Obs: alte variante de rezolvare se bazează pe corespondența dintre o mapare și un ciclu eulerian în graful asociat problemei



Maparea fragmentelor (restriction mapping)

- Varianta 3: “partial digest problem”
  - Fiind date cele C(n,2) distanțe dintre oricare două poziții de secționare să se determine pozițiile de secționare



n=5

C(5,2)=10

D(x)={3,5,5,8,9, 14,14,17,19,22}

multiset

Maparea fragmentelor (restriction mapping)

- Formularea problemei ("partial digest problem")
  - Fiind date cele C(n,2) distanțe dintre oricare două poziții de secționare să se determine pozițiile de sectionare

- Observație
  - Nici această variantă a problemei nu are soluție unică întrucât pot exista mulțimi diferite A și B pentru care multiseturile corespunzătoare distanțelor între perechi, D(A) și D(B), sunt egale (astfel de mulțimi sunt numite mulțimi homometrice)

  - Exemplu:

$$A = U + V = \{u + v, u \in U, v \in V\}$$

$$B = U - V = \{u - v, u \in U, v \in V\}$$

$$D(A) = D(B)$$

Maparea fragmentelor (restriction mapping)

- Exemplu: A={5,6,7,8,10,12}, B={1,2,3,4,6,8}

A=U+V, U={4,5,9}, V={1,3}

B=U-V

	5	6	7	8	10	12
5		1	2	3	5	7
6			1	2	4	6
7				1	3	5
8					2	4
10						2
12						

	1	2	3	4	6	8
1		1	2	3	5	7
2			1	2	4	6
3				1	3	5
4					2	4
6						2
8						

Maparea fragmentelor (restriction mapping)

- Algoritm simplu bazat pe metoda forței brute:

Intrare: L = multiset cu cele C(n,2) lungimi

Ieșire: $X=\{0,x_2, \ldots, x_n\}$

1: M= max L
2: FOR "fiecare submulțime $\{x_2,\ldots,x_{n-1}\}$ de (n-2) numere naturale cuprinse între 1 și M-1 și care conține elemente din L" DO
3: $X= \{0,x_2,\ldots,x_{n-1},M\}$
4: calcul D(X)
5: IF D(X)==L THEN RETURN X ENDIF
6: ENDFOR
7: RETURN "nu există soluție"

Maparea fragmentelor (restriction mapping)

Generarea tuturor submulțimilor unei mulțimi finite:

Intrare: m (în cazul problemei anterioare m=M-1)

Ieșire: lista submulțimilor lui {1,2,…,m}

Reprezentarea unei submulțimi:

vector caracteristic: $a=(a_1,\ldots,a_m)$ unde $a_i=1$ dacă valoarea i aparține submulțimii și 0 in caz contrar

Exemplu: m=3

Vectorii caracteristici corespunzători tuturor submulțimilor lui {1,2,3} sunt: (0,0,0), (0,0,1), (0,1,0), (0,1,1), (1,0,0),(1,0,1),(1,1,0),(1,1,1)

Maparea fragmentelor (restriction mapping)

Generarea tuturor vectorilor caracteristici:

- Numărare în baza 2
- Tehnica reducerii (Decrease and conquer)
- Tehnica căutarii cu revenire (Backtracking)

Selectarea submulțimilor care satisfac restricțiile (n-2 elemente din L):

- Numărul de elemente egale cu 1 în vectorul caracteristic trebuie să fie n-2
- Pozițiile pe care este 1 în vectorul caracteristic indică elementele ce aparțin lui L

Numarare in baza 2

- Se pornește de la vectorul caracteristic cu toate componentele egale cu 0
- La fiecare etapă se generează următorul vector prin "incrementare" cu 1:

```
Next(b[1..m])
   S=b[m]+1; b[m]=S MOD 2; r=S DIV 2
   i=m-1
  WHILE r>0 and i>=1 DO
     S=b[i]+r; b[i]=S MOD 2; r=S DIV 2
      i=i-1
  ENDWHILE
```

- Vectorul generat se acceptă doar dacă satisface restricțiile

Tehnica reducerii (decrease and conquer)

Idee: problema curentă se reduce la o problemă de dimensiune mai mică

```
submultimi(k)
  IF k=1 THEN
     b[1]:=0;  IF validare(b[1..m]) THEN “output b[1..m]” ENDIF
     b[1]:=1;  IF validare(b[1..m]) THEN “output b[1..m]” ENDIF
  ELSE
     b[k]:=0; submultimi(k-1);
     b[k]:=1; submultimi(k-1);
ENDIF
```

Apel: submultimi(m)

Funcția validare verifică faptul că pozițiile egale cu 1 din b[1..m] corespund unor elemente din L iar numărul lor este n-2

Tehnica cautarii cu revenire (backtracking)

Idee: elementele lui b se completează succesiv cu toate valorile posibile; dupa ce s-au încercat toate valorile posibile pentru o componentă se revine la componenta anterioară și se încearcă următoarea valoare corespunzătoare acesteia

```
Submultimi(k)
  IF solutie(b[1..k-1]) THEN "output b[1..k-1]"
  ELSE
     b[k]:=0; submultimi(k+1);
     b[k]:=1; IF validare(b[1..k]) THEN submultimi(k+1) ENDIF
  ENDIF
```

Apel: submultimi(1)

Functiile de validare verifică:

- pozițiile egale cu 1 din b[1..k-1] corespund unor elemente din L (pt funcțiile solutie și validare)
- numărul pozițiilor egale cu 1 este n-2 (doar pentru funcția soluție)

Maparea fragmentelor (restriction mapping)

Este un astfel de algoritm eficient ?

- Răspunsul la o astfel de întrebare necesită stabilirea ordinului de complexitate al algoritmului

Cum măsuram complexitatea unui algoritm ?

- estimând dependența dintre volumul de resurse (de exemplu numărul de execuții ale unor operații) și dimensiunea problemei (numărul de fragmente, lungimea totală a secvenței)

Algoritmi eficienți (aplicabili pentru probleme reale):

- Dependență logaritmică sau polinomială de ordin mic (ex: liniară)

Algoritmi ineficienți (inaplicabili pentru probleme de dimensiuni mari):

- Dependență exponențială sau polinomială de grad mare

Analiza complexitatii

Scop: stabilirea dependenței dintre numărul de operații efectuate și dimensiunea problemei

Notații: f(n) = funcția care descrie dependența timpului de execuție de dimensiunea problemei (n)

g(n) = expresia care definește ordinul de complexitate

Clase de complexitate (notații asimptotice)

$f(n) \in \Theta(g(n))$ d.n.d. există $c_1, c_2 \in R_+$ si $n_0 \in N$ astfel încât

$c_1 g(n) \leq f(n) \leq c_2 g(n)$ pentru orice $n \geq n_0$

(f(n) crește la fel de repede ca și g(n))

$f(n) \in O(g(n))$ d.n.d. există $c \in R_+$ si $n_0 \in N$ astfel încât

$f(n) <= c\, g(n)$ pentru orice $n >= n_0$

(f(n) crește cel mult la fel de repede ca g(n))

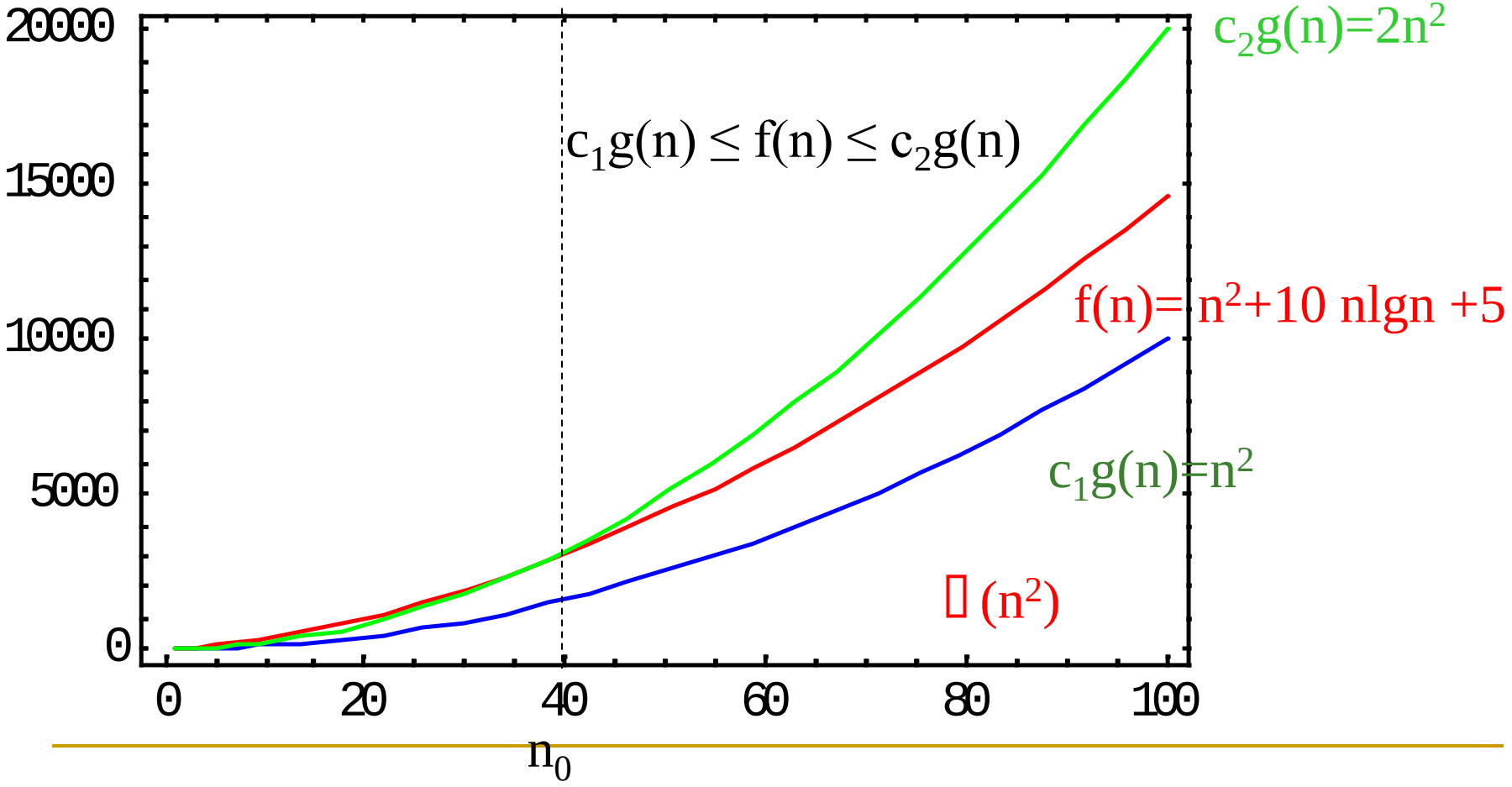
$f(n) \in \Omega(g(n))$ d.n.d. există $c \in R_+$ si $n_0 \in N$ astfel încât

$cg(n) <= f(n)$ pentru orice $n >= n_0$

(f(n) crește cel puțin la fel de repede ca g(n))

Analiza complexitatii

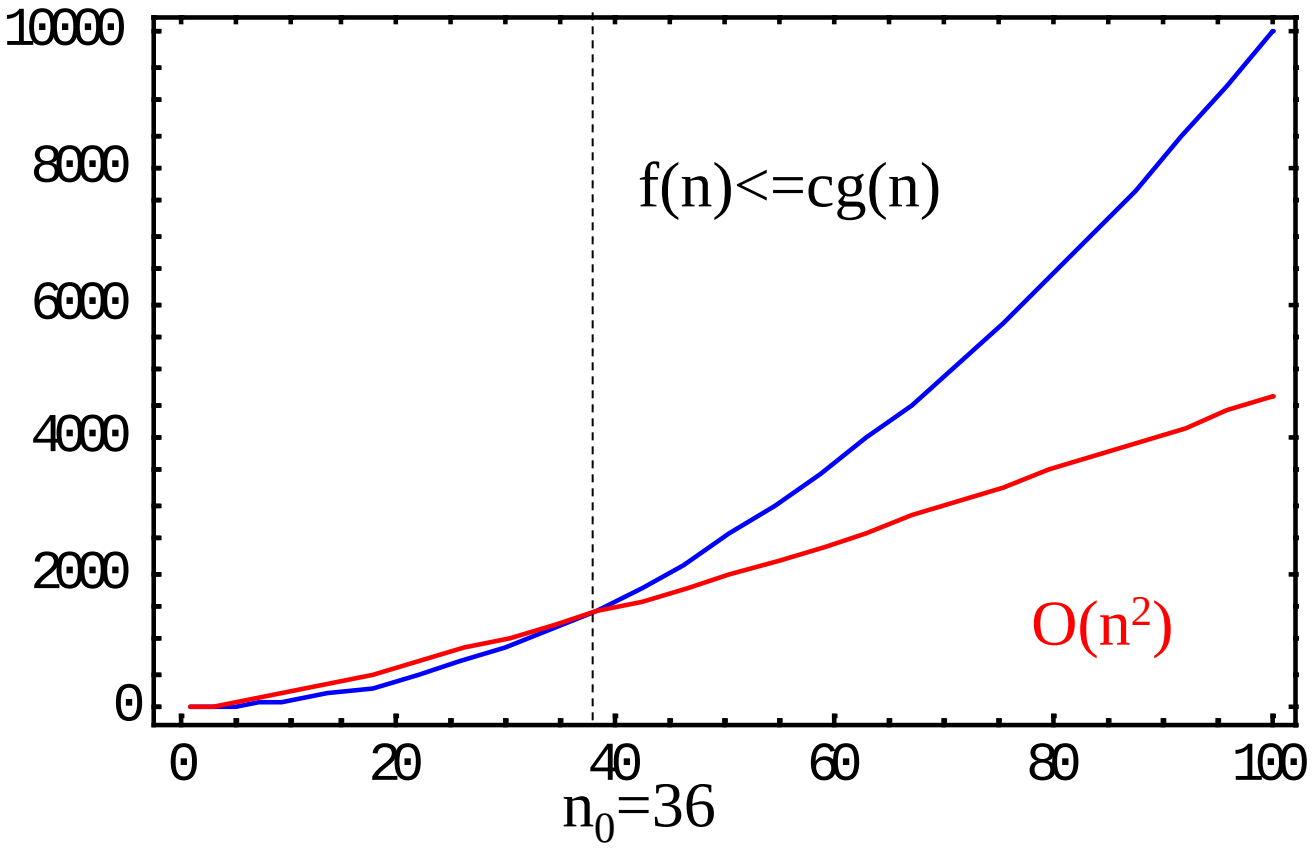
f(n) $\in \Theta(g(n))$ d.n.d. există $c_1, c_2 \in R_+$ și $n_0 \in N$ astfel încât $c_1 g(n) \leq f(n) \leq c_2 g(n)$ pentru orice $n \geq n_0$



Analiza complexitatii

$f(n) \in O(g(n))$ d.n.d. există $c \in R_+$ si $n_0 \in N$ astfel încât

$f(n) <= c\, g(n)$ pentru orice $n >= n_0$



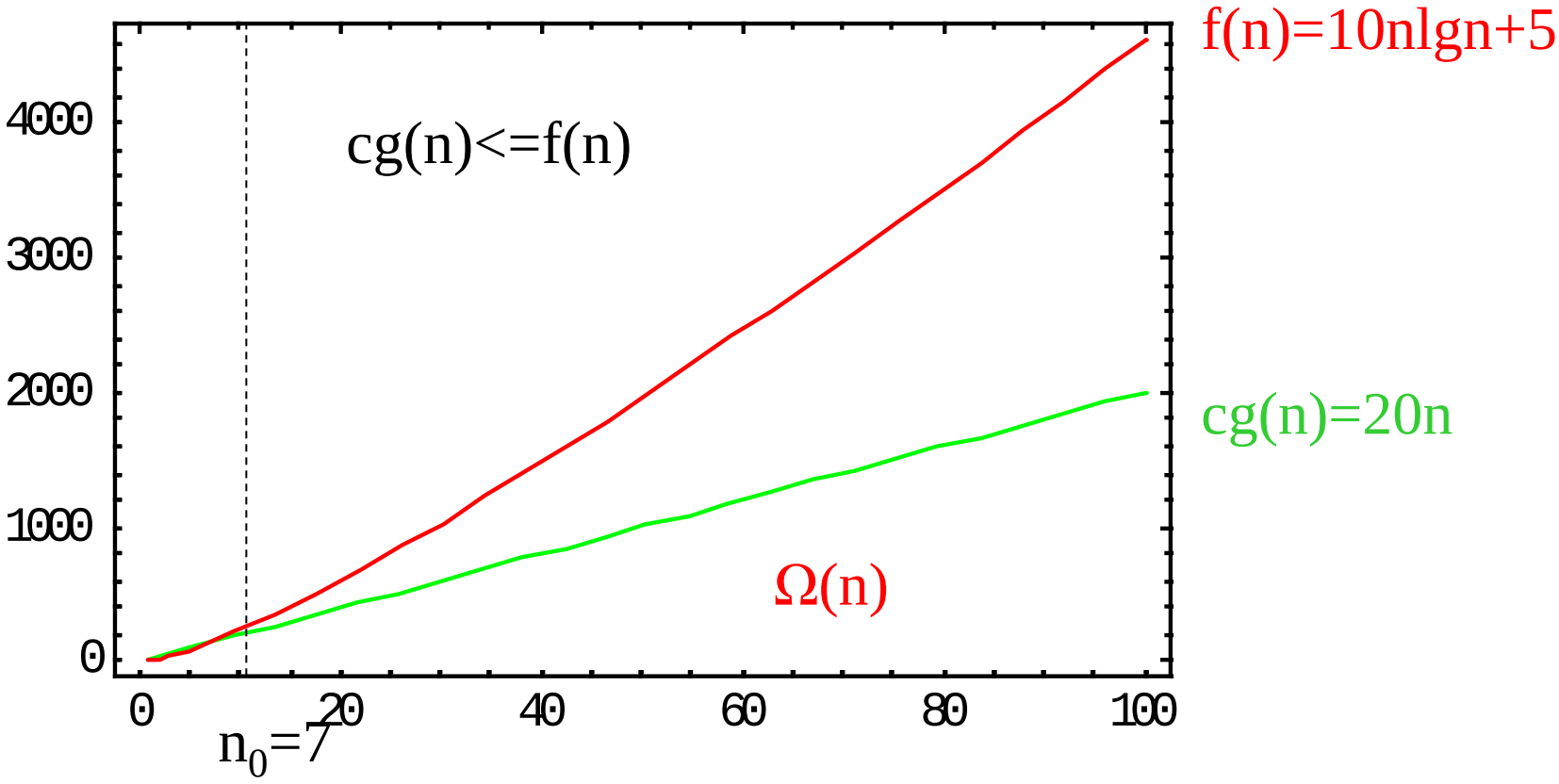
$cg(n)=n^2$

$f(n)= 10n\lg n +5$

Analiza complexitatii

$f(n) \in \Omega(g(n))$ d.n.d. există $c \in R_+$ si $n_0 \in N$ astfel încât

$cg(n) <= f(n)$ pentru orice $n>=n_0$



Analiza complexitatii

Analiza algoritmului bazat pe metoda forței brute:

Dimensiunea problemei: (m, n) (m=M-1=max L-1, n=nr fragmente)

Generarea submulțimilor: $O(m2^m)$

Analiza fiecărei submulțimi (comparare D(L) cu X): $O(n^2)$

Total: $O(2^m (m+n^2))$

Se obține astfel un ordin exponențial de complexitate: inaplicabilă pentru valori mari ale lui m și/sau n

Observație:

O ușoară îmbunătățire se obține dacă în loc să se genereze toate submulțimile din {1,…,m} se generează direct submulțimile cu n-2 elemente din L. In acest caz se obține $O(n^2 C(m,n-2))$ ceea ce poate fi semnificativ mai mic decât $O(2^m (m+n^2))$ dacă n este semnificativ mai mic decât m (nr mic de fragmente într-o secvență de lungime mare)

Varianta mai eficienta a algoritmului (Skiena, 1990)

L = { 2, 2, 3, 3, 4, 5, 6, 7, 8, 10 }

X = { 0 }

Exemplu din Ch04_DNA_mapping.pdf (http://www.bioalgorithms.info)

Varianta mai eficienta a algoritmului (Skiena, 1990)

L = { 2, 2, 3, 3, 4, 5, 6, 7, 8, 10 }

X = { 0 }

Se elimină 10 din L și se inserează în X. Trebuie să fie extremitatea din dreapta a secvenței.

Varianta mai eficienta a algoritmului (Skiena, 1990)

L = { 2, 2, 3, 3, 4, 5, 6, 7, 8 }

X = { 0,10 }

Varianta mai eficienta a algoritmului (Skiena, 1990)

L = { 2, 2, 3, 3, 4, 5, 6, 7, 8}

X = { 0,10 }

- Se alege maximul din L (8). Poziția de tăietură (y) poate fi 2 sau 8. Se alege y=2.
- Se elimină 8 și un 2 din L

L = { 2, 3, 3, 4, 5, 6, 7}

X = { 0,2,10 }

Varianta mai eficienta a algoritmului (Skiena, 1990)

L = {2, 3, 3, 4, 5, 6, 7}

X = { 0,2,10 }

- Se determină maximul din L (7). Există două variante posibile: y=3 și y=7. Varianta y=3 nu poate fi aleasă întrucât L nu conține valoarea 1=3-2 (care ar fi distanța dintre punctele 3 și 2).

- Prin urmare se alege y=7. Distanțele de la y la elementele curente din X sunt: {7,5,3}. Se adaugă 7 la X și se elimină 7,5 și un 3 din L.

L = { 2, 3, 4,6}
X = { 0,2,7,10 }

Varianta mai eficienta a algoritmului (Skiena, 1990)

L = {2, 3, 4,6}

X = {0,2,7,10 }

- Se alege maximul din L (6). Pentru y=6, distanțele de la y la elementele curente din X sunt: {6,4,1,4}. Aceasta submulțime nu este inclusă în L. Se ignoră 6 și se alege 4. Pentru y=4 distanțele sunt: {4,2,3,6}. Se adaugă 4 la X și se elimină {4,2,3,6} din L

L = { }

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

- L fiind vidă înseamnă că s-a obținut o soluție. Pentru a determina celelalte soluții se trece înapoi pentru a explora variantele rămase neexplorate (ex: 8 de la prima etapă)

Varianta mai eficienta a algoritmului (Skiena, 1990)

- PartialDigest(*L*):
  m = Max *L*
  DELETE(*m*, *L*)
  X = {0, *m*}
  PLACE(m,*L*, *X*)

Obs.
D(y,X) = mulțimea distanțelor de la y la punctele din mulțimea X
L=lista cu distanțele

```
PLACE(m,L, X)
 IF L este vidă
    output X; return        // s-a obținut o soluție
ENDIF
y = Max L                   // determină cea mai mare distanță
DELETE(y,L)                 // elimină distanța din L
IF mulțimea D(y, X ) este inclusă în L
    Adaugă y la X și elimină lungimile D(y, X) din L
    PLACE(m,L,X )
    Sterge y  din X și adaugă D(y, X) la L
ENDIF
IF  mulțimea D(m-y, X ) este inclusă în L
    Adaugă m-y la X și elimină lungimile  D(m-y, X) din L
    PLACE(m,L,X )
    Elimină m-y din X și adaugă lungimile D(m-y, X ) la L
ENDIF
```

Varianta mai eficienta a algoritmului

Analiza complexității:

Cazul cel mai favorabil:

la fiecare etapă prima ramură analizată este validă

Relația de recurență pt. timpul de execuție

$T(n)=O(1), \quad n=1$

$T(n)=T(n-1)+O(n), \quad n>1$

$T(n) \in O(n^2)$

```
PLACE(m,L, X)
 IF L este vidă
    output X; return        // s-a obținut o soluție
ENDIF
y = Max L                   // determină cea mai mare distanță
DELETE(y,L)                 // elimină distanța din L
IF mulțimea D(y, X ) este inclusă în L
    Adaugă y la X și elimină lungimile D(y, X) din L
    PLACE(m,L,X )
    Sterge y  din X și adaugă D(y, X) la L
ENDIF
IF  mulțimea D(m-y, X ) este inclusă în L
    Adaugă m-y la X și elimină lungimile  D(m-y, X) din L
    PLACE(m,L,X )
    Elimină m-y din X și adaugă lungimile D(m-y, X ) la L
ENDIF
```

Varianta mai eficienta a algoritmului

Analiza complexitatii:

Cazul cel mai nefavorabil: la fiecare etapă se parcurg ambele ramuri

Relația de recurență pt. timpul de executie

$T(n)=O(1), \quad n=1$

$T(n)=2T(n-1)+O(n), \; n>1$

$T(n) \in O(n2^n)$

```
PLACE(m,L, X)
 IF L este vidă
    output X; return        // s-a obținut o soluție
ENDIF
y = Max L                   // determină cea mai mare distanță
DELETE(y,L)                 // elimină distanța din L
IF mulțimea D(y, X ) este inclusă în L
    Adaugă y la X și elimină lungimile D(y, X) din L
    PLACE(m,L,X )
    Elimină y  din X și adaugă D(y, X) la L
ENDIF
IF  mulțimea D(m-y, X ) este inclusă în L
    Adaugă m-y la X și elimină lungimile  D(m-y, X) din L
    PLACE(m,L,X )
    Elimină m-y din X și adaugă lungimile D(m-y, X ) la L
ENDIF
```

Varianta mai eficienta a algoritmului

Algoritm de complexitate (pseudo)polinomiala:

[Daurat, Gerard, Nivat – 2002]

- se bazează pe factorizări de polinoame cu coeficienți intregi (lungimile fragmentelor intervin ca grade ale termenilor polinomului)

Alta varianta ale problemei:

-Simplified Partial Digest: se folosește o enzimă însă doar două intervale de timp (unul scurt care permite o singura secționare/secvența și unul lung care permite toate secționările/secvența)