

Sekvenciranje

- Današnja tehnologija ne omogućuje precizno „čitanje” cijelog genoma odjednom
- Veličina genoma:
 - Virus ~ 50.000 baza
 - Bakterija ~ 5.000.000 baza
 - Kvasac ~ 10.000.000 baza
 - Ptice ~ 1.000.000.000 baza
 - Čovjek ~ 3.000.000.000 baza
 - Jedan kromosom ~ 50.000.000 – 250.000.000 baza
 - Neke biljke čak preko 100.000.000.000 baza

Sekvenciranje

- Današnja tehnologija ne omogućuje precizno „čitanje” cijelog genoma odjednom
- Genom se umnaža (50x), nasumično razbija na manje fragmente koje se može „pročitati” – *shotgun sequencing*
- Kako provjeriti je li rezultat dobar?
- Prve metode sekvenciranja: označavanje pojedinih dijelova kromosoma, sekvenciranje i sastavljanje.
- Prvi sastavljeni ljudski genom koštao je oko 3 milijarde \$

Sekvenciranje

- Današnja tehnologija ne omogućuje precizno „čitanje” cijelog genoma odjednom
- Genom se umnaža (50x), nasumično razbija na manje fragmente koje se može „pročitati” – *shotgun sequencing*
- Kako provjeriti je li rezultat dobar?
- Čarobna granica: genoma za \$1000
- “Everybody talks about the \$1,000 genome, but they don’t talk about the \$2,000 mapping problem behind the \$1,000 genome” Peter Tonellato, University of Wisconsin

Sekvenciranje - tehnologije

- Prva generacija
 - Sangerovo sekvenciranje
 - Očitavanja srednje duljine, precizno
 - Jako sporo
- Druga generacija (NGS – *next generation sequencing*)
 - Illumina, Roche, Ion Torrent ...
 - Očitavanja male duljine, tipično 100-200 baza, precizno (~ 1-3%)
 - Brzo i jeftino
- Treća generacija
 - PacBio, ONT – Oxford Nanopore Technologies
 - Očitavanja velike duljine (deseci tisuća baza) s velikom greškom (~ 5-30%)
 - Brzo i jeftino?

Primjena bioinformatike

- Precizna medicina – lijekovi bolje djeluju na pacijentima koji imaju točno određenu mutaciju na nekom genu (ili genima)
 - Mutacija: zamjena, umetanje, brisanje
- Usporedba s referentnim genomom
 - deepVariant (Google)
- Sastavljanje genoma i čitanje pojedinog gena
 - Iz podataka dobivenih sekvenciranjem
 - Preklapanja između očitavanja

Primjena bioinformatike

- Sastavljanje genoma i čitanje pojedinog gena
 - Iz podataka dobivenih sekvenciranjem
 - Preklapanja između očitavanja
- Uspoređivanje nizova
 - Dinamičko programiranje
 - Indeksiranje manjih podnizova (*kmer*)
 - Kombinacija (*seed & extend*)

Bioinformatika

2020/2021

Dinamičko programiranje

Fibonacci

$$F_0 = 0,$$

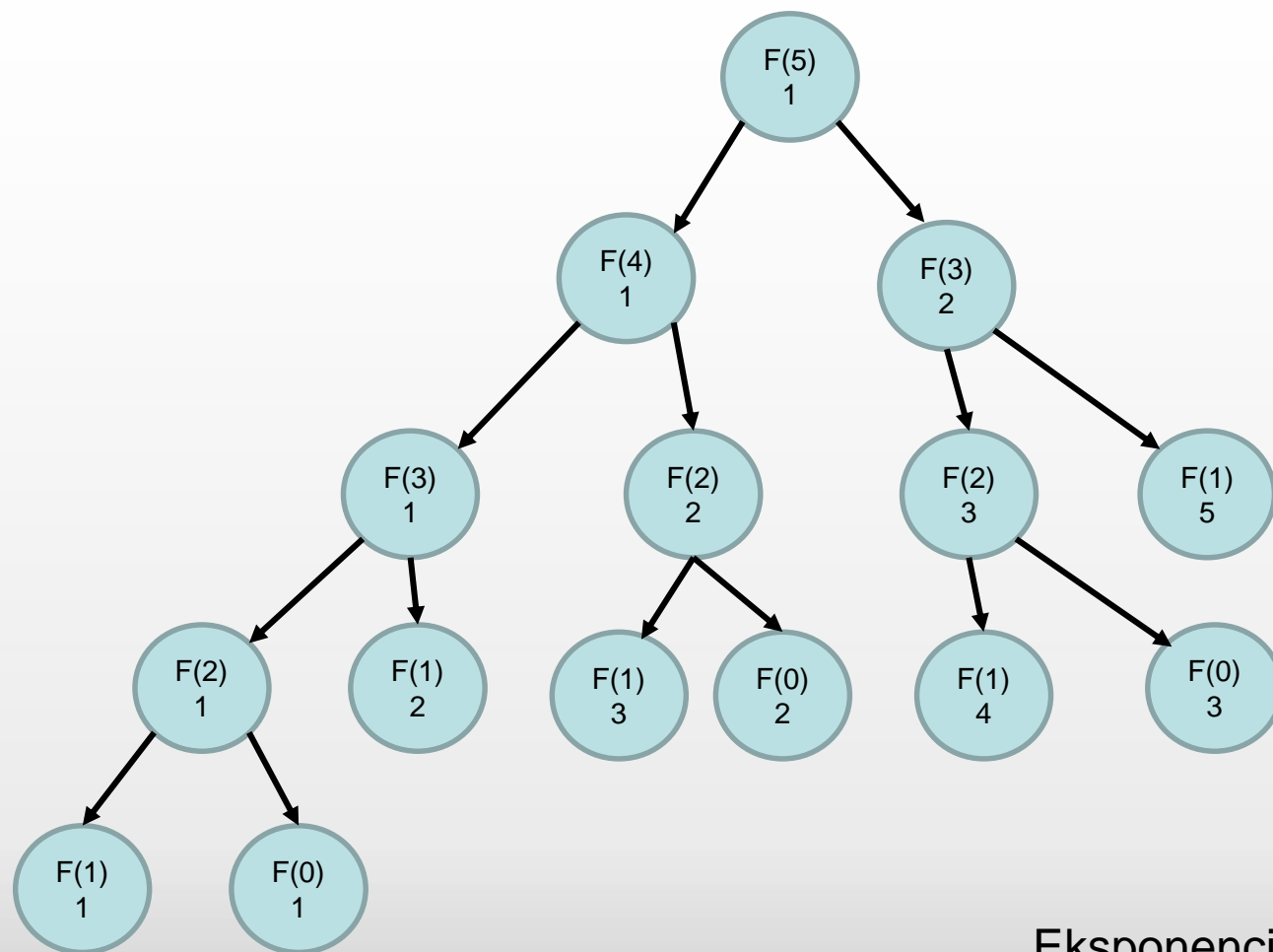
$$F_1 = 1,$$

$$F_n = F_{n-1} + F_{n-2}, \forall n > 1$$

Naivni algoritam

```
Fibonnaci (n) :  
    ako je n = 0:  
        vrati 0  
    inače ako je n = 1:  
        vrati 1  
    inače:  
        vrati Fibonnaci (n-1) + Fibonnaci (n-2)
```

Fibonacci – stablo aktiviranja



Eksponencijalna vremenska
složenost

Memoizirani naivni algoritam

Fib je niz $n+1$ cijelih brojeva

Inicijaliziraj $\text{Fib}[i] = -1$ za sve $i \leq n$

$\text{Fib}[0] = 0$

$\text{Fib}[1] = 1$

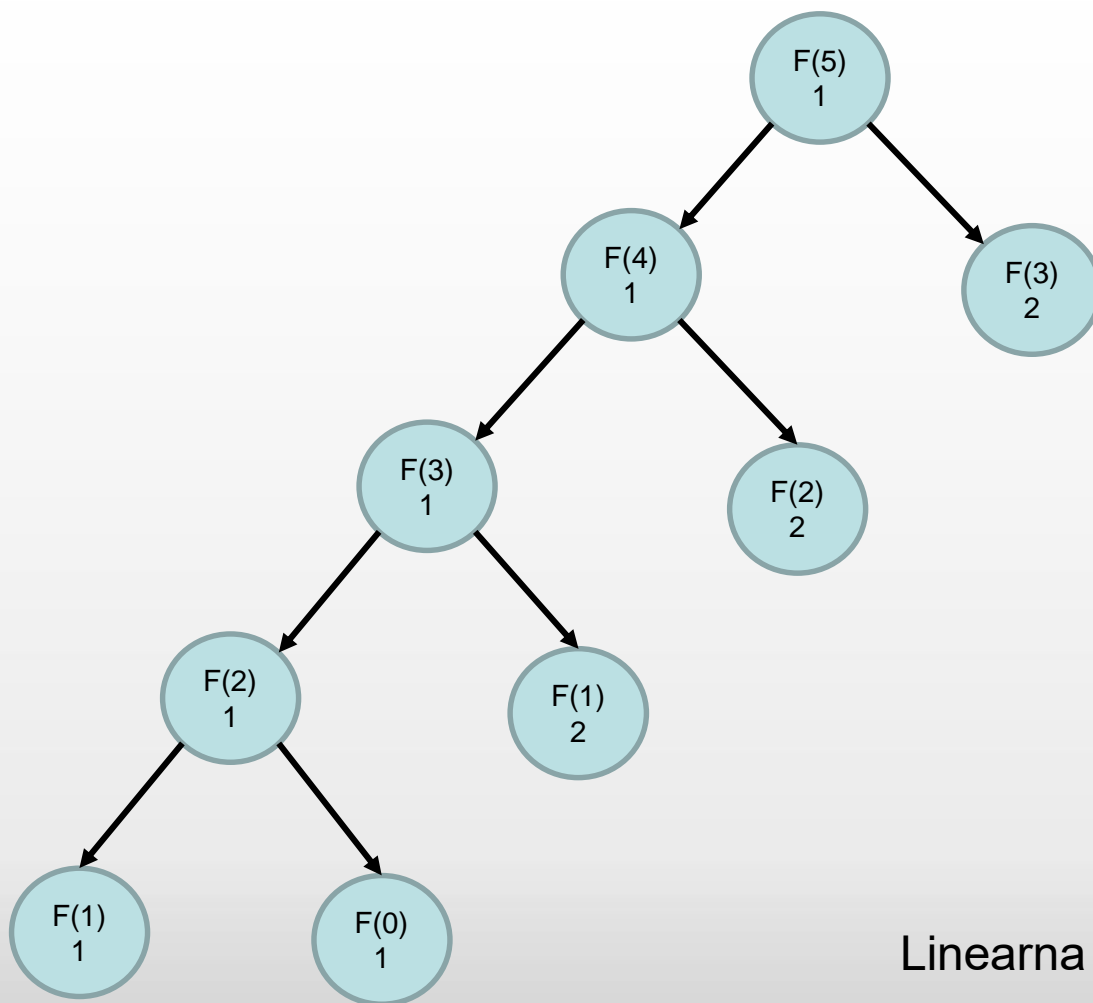
Fibonacci(n) :

ako je $\text{Fib}[n] = -1$:

$\text{Fib}[n] = \text{Fibonacci}(n-1) + \text{Fibonacci}(n-2)$:

vрати $\text{Fib}[n]$

Fibonacci – stablo aktiviranja



Linearna vremenska složenost

Memoizirani naivni algoritam

Fin je niz $n+1$ cijelih brojeva

Inicijaliziraj $\text{Fib}[i] = -1$ za sve $i \leq n$

$\text{Fib}[0] = 0$

$\text{Fib}[1] = 1$

za i od 2 do n :

$\text{Fib}[i] = \text{Fib}[i-1] + \text{Fib}[i-2]$

Rješavanje tablice po redu
Linearna složenost

Dinamičko programiranje

- Klasa algoritama koja rješava probleme rješavanjem njihovih manjih dijelova, spremanjem rješenja i nakon toga kombiniranjem istih za rješenje većega problema.
- Rješenja su obično trivijalna i rezultati se spremaju u jednostavne strukture podataka (varijable, polja i tablice). Ti rezultati se onda koriste za rješavanje nešto većih problema, koji spremaju i koriste za rješavanje još većih problema.

Metrike sličnosti

- Hammingova udaljenost

- Računanje broja zamjena za transformaciju jednoga niza u drugi

GATTACA

|||X|||

GATCACA

1

ATTACCC

XX|XX|X

GATTACA

5

- Udaljenost uređivanja (Levenshtein)

- Minimalan broj zamjena, umetanja i brisanja za transformaciju jednoga niza u drugi

GATTACA

|||X|||

GATCACA

1

-ATTACCC

X|||||XX

GATTAC-A

3

Svojstva metrika

Svojstva metrika/mjerenja udaljenosti između dvije sekvence

1. $d(x,y) \geq 0$ za sve x,y .
2. $d(x,y) = 0$, ako i samo ako $x = y$
3. $d(x,y) = d(y,x)$ (simetrija)
4. $d(x,y) \leq d(x,z) + d(z,y)$ za sve x,y i z
(nejednakost trokuta)

Udaljenost uređivanje i mjera sličnosti

- Biolozi koriste mjeru sličnosti koja raste što su sličnosti veće.
- Peter H. Sellers (1974)
 - $\text{Min}(\text{udaljenost uređivanja}) \sim \text{max}(\text{sličnosti})$
- Slični algoritmi se mogu koristiti za optimiziranje obje mjere

Problem udaljenosti uređivanja

- Ulaz: dvije sekvence, npr.
t: ATCTGAT
s: TGCATAT
- Izlaz: minimalni težinski broj operacija uređivanje za transformaciju s u t.

Operacije uređivanja

- Postoje tri tipa promjena:
 - Zamjena – promjena jednoga znaka iz niza s u drugačiji znak u nizu t, npr. “most” u “mast”
 - Umetanje – umetanje jednoga znaka u niz s u cilju da odgovara nizu t, npr. “kos” u “kosa”
 - Brisanje – brisanje jednoga znaka iz niza s u cilju da odgovara nizu t, npr. “brod” u “rod”

Udaljenost uređivanja (Primjer)

TGCATAT → ATCCGAT u 5 koraka

TGCATAT	→ (obriši zadnji T)
TGCAT	→ (obriši zadnji A)
TGCAT	→ (umetni A na početak)
ATGCAT	→ (zamijeni G sa C)
ATCCAT	→ (ubaci G prije zadnjeg A)
ATCCGAT	(kraj)

Udaljenost uređivanja (Primjer)

Poravnanje (TGCATAT u ATCCGAT)

ATCCGAT--

-TGC-ATAT

IMMIMMDD

Udaljenost uređivanja (Primjer)

TGCATAT → ATCCGAT u 4 koraka

TGCATAT → (obriši A)

ATGCTAT → (ubaci A)

ATGCTAT → (zamijeni G sa C)

ATCCTAT → (zamijeni T s G)

ATCCGAT (kraj)

Udaljenost uređivanja (Primjer)

Poravnanje (TG CATAT u ATCCGAT)

ATCC-GAT

-TGCATAT

IMMMDMMM

Poravnanje dva niza

- Elegantan algoritam za traženja minimalnog troška transformacije niza S u T je temeljen na zapažanju da ispravna akcija na najdesnijim znakovima od S i T može biti izračunata znajući cijenu poravnanja različitih prefiksa!

Udaljenost uređivanja – obrnuti inženjering

$$D(\text{TGCATAT}, \text{ATCCGAT}) = ?$$

Zamislamo da već imamo optimalno poravnanje nizova. U tom slučaju zadnji stupac može biti samo jedna od tri opcije:

...Z	...U	...B
...T	...-	...T
...T	...T	...-

Optimalno poravnanje zadnja dva stupca je jedna od devet mogućnosti:

...ZZ	...UZ	...BZ	...ZU	...UU	...BU	...ZB	...UB	...BB
...AT	...-T	...AT	...T-	...--	...T-	...AT	...-T	...AT
...AT	...AT	...-T	...AT	...AT	...-T	...T-	...T-	...--

Optimalno poravnanje zadnja tri stupca je jedna od 27 mogućnosti:

...Z...	...U...	...B...
...X...	...-...	...X...
...Y...	...Y...	...-...

Na kraju potrošimo sve moguće nizove od {U, Z, B}

Poravnanje dva niza

```
#define MATCH          0          /* symbol for match */
#define INSERT        1          /* symbol for insert */
#define DELETE        2          /* symbol for delete */

int string_compare(char *s, char *t, int i, int j) {
    int k;                /* counter */
    int opt[3];           /* cost of the three options */
    int lowest_cost;      /* lowest cost */

    if (i == 0) return(j * indel(' ')); /* boundary conditions */
    if (j == 0) return(i * indel(' ')); /* boundary conditions */

    opt[MATCH]  = string_compare(s,t,i-1,j-1) + match(s[i],t[j]);
    opt[INSERT] = string_compare(s,t,i,j-1) + indel(t[j]);
    opt[DELETE] = string_compare(s,t,i-1,j) + indel(s[i]);

    lowest_cost = opt[MATCH];
    for (k=INSERT; k<=DELETE; k++)
        if (opt[k] < lowest_cost) lowest_cost = opt[k];
    return( lowest_cost );
} /* Steven Skiena, http://www.algorithm.cs.sunysb.edu/computationalbiology/ */
```

Složenost!

- Koliko je složenost izvršenja ovoga programa? Eksponencijalna s duljinom nizova!
- Primijetimo da stalno ponavljamo isto računanje na svakom paru prefiksa

Poravnanje dva niza

- Stanje rekurzivnih poziva je vođeno pozicijom indeksa u nizovima. Stoga postoji samo $|S| \times |T|$ različitih poziva!
- Spremanjem odgovara u tablicu i njihovim pozivanjem umjesto ponovnoga računanja, algoritam postiže kvadratnu složenost

Ideja temeljena na dinamičkom programiranju

- Neka je n duljina niza x
- Neka je m duljina niza y
- Konstruirati maticu F dimenzija $(n+1) \times (m+1)$
- $F(j, i) =$ rezultat najboljega poravnanja $x_1 \dots x_i$ s $y_1 \dots y_j$

		x			
		A	G	C	
y		0	d	2d	3d
	A	d			
	A	2d			
	A	3d			
	C	4d			

Rezultat najboljega poravnanja
AAA do AG

DP algoritam

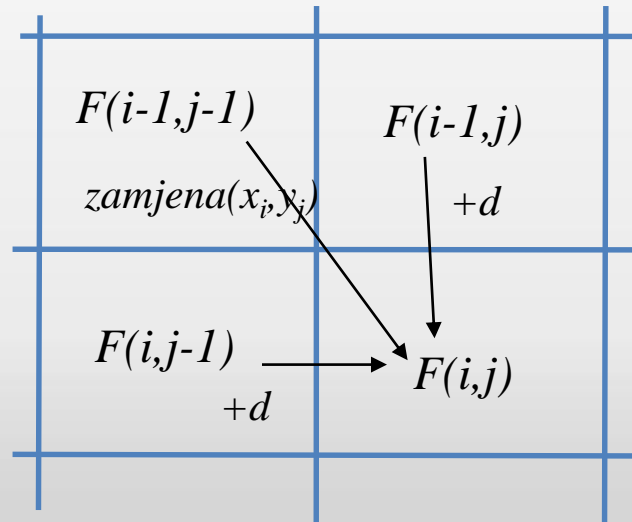
- Inicijalizirati prvi red i prvi stupac matrice
- Popuniti ostatak matrice od vrha do dna, s lijeva na desno
- Za svaki $F(i, j)$, spremiti od kuda se došlo
- $F(m, n)$ sadrži optimalno poravnanje
- Vratiti se natrag od $F(m, n)$ do $F(0, 0)$ za određivanje poravnanja
- **Wagner-Fisher** algoritam za određivanje udaljenosti uređivanja

Princip poravnanja

$$F(i, j) = \min \begin{cases} F(i-1, j-1) + \text{zamjena}(x_i, y_j) \\ F(i-1, j) + d \\ F(i, j-1) + d \end{cases}$$

Rezultat najboljeg djelomičnoga poravnanja između $x_1..x_i$ i $y_1..y_j$

Praznina (engl. indel) – umetanje ili brisanje



Tablica dinamičkog programiranja

- Tablica čuva informaciju o cijeni dostizanja pojedine pozicije plus zadnji potez koji je vodio do te pozicije.

```
typedef struct {  
    int cost; /* cost of reaching this cell */  
    int parent; /* parent cell */  
} cell;  
  
cell m[MAXLEN][MAXLEN]; /* dynamic programming table */
```


Opći algoritam računanja udaljenosti uređivanja

```
int string_compare(char *s, char *t)
{
    int i,j,k; /* counters */
    int gi, gj; /* goal cell coordinates */
    int opt[3]; /* cost of the three options */
    /* initialize the boundary conditions */
    for (i=0; i<MAXLEN; i++) {
        row_init(i);
        column_init(i);
    }
```

Opći algoritam računanja udaljenosti uređivanja

```
for (i=1; i<=strlen(s); i++)
    for (j=1; j<=strlen(t); j++) {
        opt[MATCH] = m[i-1][j-1].cost + match(s[i],t[j]);
        opt[INSERT] = m[i][j-1].cost + indel(t[j]);
        opt[DELETE] = m[i-1][j].cost + indel(s[i]);

        m[i][j].cost = opt[1];
        m[i][j].parent = 1;

        for (k=2; k<=3; k++) {
            if (opt[k] < m[i][j].cost) {
                m[i][j].cost = opt[k];
                m[i][j].parent = k;
            }
        }
    }
goal_cell(s,t,&gi,&gj); /* returns the desired final cell */
return( m[gi][gj].cost );
} /* Steven Skiena,
http://www.algorithm.cs.sunysb.edu/computationalbiology/ */
```

Standardno određivanje udaljenosti uređivanja

- Funkcija `string_compare` je opća i mora biti prilagođena primjeni.
- Koristi funkcije `match` i `indel` za vraćanje cijene tranzicije dva znaka
- `row_init` i `column_init` postavljaju rubne uvjete
 - $F(i,0) = i * d$
 - $F(0,j) = j * d$
- Cijena:
 - nema zamjene: 0
 - zamjena: 1
 - praznina: 1

Standardno određivanje udaljenosti uređivanja

- Funkcija `goal_cell` vraća željenu konačnu ćeliju u matrici
 - Određivanje udaljenosti - zadnji stupac, zadnji redak
 - Promjena te funkcije dopušta nam preklapanje podnizova, traženje najduljeg zajedničkog podniza i maksimalnog monotonog podniza kao specijalnih slučajeva

Primjer poravnanja

Inicijalizacija matrice

		A	G	C	
		0	d	2d	3d
A		d			
A		2d			
A		3d			
C		4d			

Primjer poravnanja

		A	G	C
		0 ← 1 ← 2 ← 3		
A		1	0 ← 1 ← 2	
A		2	1	1 ← 2
A		3	2	2
C		4	3	3

Određivanje poravnanja

		A	G	C
A	0	1	2	3
A	1	0	1	2
A	2	1	1	2
A	3	2	2	2
C	4	3	3	2

x: A A A C

y: - A G C

Jedno optimalno poravnanje

Biološka usporedba nizova

- Konstruiranje smislenoga poravnanja dvaju nizova zahtjeva korištenje odgovarajuće funkcije za mjerenje zamjene između svakoga mogućeg para simbola.
- Za biološke sekvence za kažnjavanje zamjene nukleotida koriste se PAM matrice (engl. *point accepted mutation*)
- Kod proteina su popularne BLOSUM matrice, posebice BLOSUM62

DNA matrice

- DNA PAM matrice: NCBI BLAST sličnost i tranzicija/transverzija

	A	T	C	G
A	5	-4	-4	-4
T	-4	5	-4	-4
C	-4	-4	5	-4
G	-4	-4	-4	5

	A	T	C	G
A	0	5	5	1
T	5	0	1	5
C	5	1	0	5
G	1	5	5	0

- Nukleotide klasificiramo kao purine (adenin i gvanin) ili pirimidine (citozin i timin)
- Tranzicija purin -> purin ili pirimidin -> pirimidin
- Transverzija purin -> pirimidin ili pirimidin -> purin
- Tranzicije češće nego transverzije
- **Za maksimiziranje sličnosti koristimo Needleman-Wunsch algoritam:**
 - Globalno poravnanje (od kraja do kraja niza)
 - Početni uvjeti su $F(i,0) = -i*d$ i $F(0,j) = -j*d$ (d je kazna za umetanje ili brisanje; pozitivan broj)

Preklapanje

- Preklapanje dvije sekvence je poravnavanje u kome su praznine na početku i kraju nizova zanemarene
- Dvije sekvence
 - $x = \text{CAGCACTTGGATTCTCGG}$
 - $y = \text{CAGCGTGG}$
 - Kako izgleda njihovo globalno poravnanje ?

CAGCACTTGGATTCTCGG

CAGC-----G--T-----GG

Preklapanje

- Ono što želimo postići je:

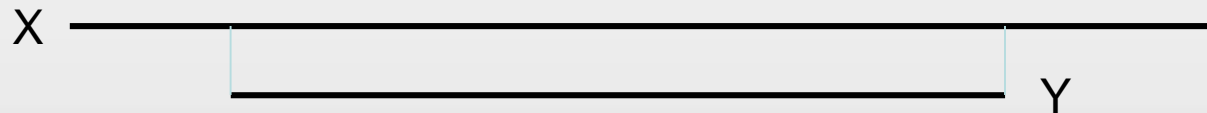
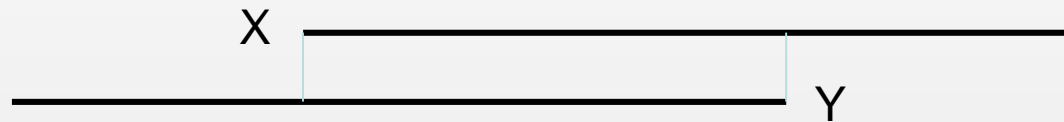
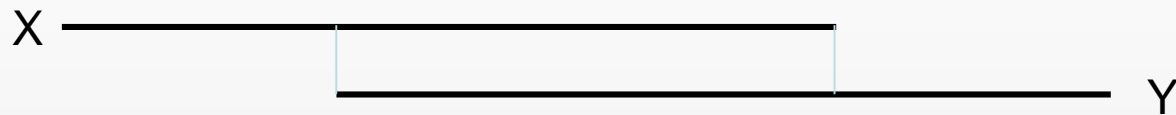
CAGCA-CTTGGATTCTCGG

---CAGCGTGG-----

- Moguća preklapanja:
 - Sufiks od x se poravnava s prefiksom od y
 - Sufiks od y se poravnava s prefiksom od x
 - y se poravnava s podnizom od x
 - x se poravnava s podnizom od y

Preklapanje

- Primjeri preklapanja



Preklapanje

- Algoritam polu-globalnog poravnavanja
 - Rubni uvjeti 0 – poravnanje može početi s prazninama
 - `goal_cell` – traženje maksimuma (Needleman-Wunsch) u zadnjem retku i stupcu (završetak s prazninama)

Lokalno poravnanje

- Kritični problem od biološkoga interesa je usporedba dva dugačka niza i traženje lokalnih područja sličnosti
- Primjene:
 - Sačuvanost regija između vrsta
 - Prepoznavanje kodirajućih regija
- Korištenje sličnosti ima više smisla nego udaljenost uređivanja
- Želimo da $d(i,j)$ bude najveće lokalno poravnanje koje završava u $S[i]$ i $T[j]$
- CG**ATG**TC
AA**ATG**GA
- *Smith-Waterman* algoritam.

Lokalno poravnanje

- To je u stvari isti algoritam kao udaljenost uređivanja osim:
 - Maksimiziramo umjesto da minimiziramo.
 - Imamo opciju da krenemo s lokalnim poravnanjem u svakoj ćeliji matrice (0 je uvijek dostupna vrijednost).
 - Maksimum može biti u bilo kojoj ćeliji ne samo u zadnjem retku ili stupcu.

Smith-Waterman primjer

- Pronaći najbolje lokalno poravnanje sekvenci "ACCTAAGG" i "GGCTCAATCA" koristeći +2 za podudaranje, -1 za zamjenu i -2 za prazninu. Popuniti 0-ti stupac i 0-ti redak s nulama

		G	G	C	T	C	A	A	T	C	A
		0	0	0	0	0	0	0	0	0	0
A		0									
C		0									
C		0									
T		0									
A		0									
A		0									
G		0									
G		0									

Smith-Waterman primjer

Prvo izračunamo $F(1,1)$ koristeći rekurziju:

$$F(i, j) = \max \begin{cases} F(i-1, j-1) + \text{zamjena}(x_i, y_j) = 0 - 1 = -1 \\ F(i-1, j) + d = 0 - 2 = -2 \\ F(i, j-1) + d = 0 - 2 = -2 \\ 0 \end{cases}$$

Maksimalna vrijednost je 0, stoga $F(1,1) = 0$

		G	G	C	T	C	A	A	T	C	A
		0	0	0	0	0	0	0	0	0	0
A		0	0	?							
C		0									
C		0									
T		0									
A		0									
A		0									
G		0									
G		0									

Nakon toga računamo $F(1,2)$

Smith-Waterman primjer

Popuniti cijelu tablicu, pamteći koja ćelija je prethodila svakoj ćeliji.
Pronaći ćeliju s maksimalnim rezultatom

		G	G	C	T	C	A	A	T	C	A
		0	0	0	0	0	0	0	0	0	0
A		0	0	0	0	0	2	2	0	0	2
C		0	0	0	2	0	2	0	1	1	2
C		0	0	0	2	1	2	1	0	0	3
T		0	0	0	0	4	2	1	0	2	1
A		0	0	0	0	2	3	4	3	1	3
A		0	0	0	0	0	1	5	6	4	2
G		0	2	2	0	0	0	3	4	5	3
G		0	2	4	2	0	0	1	2	3	4

Smith-Waterman primjer

Pretragom unatrag doći do poravnanja.

	G	G	C	T	C	A	A	T	C	A
A	0	0	0	0	0	0	0	0	0	0
C	0	0	0	2	0	2	0	1	1	2
T	0	0	0	0	4	2	1	0	2	1
A	0	0	0	0	2	3	4	3	1	3
G	0	2	2	0	0	0	3	4	5	3
G	0	2	4	2	0	0	1	2	3	4

CTCAA

|| ||

CT-AA

Ušteda u prostoru

- Primijetiti da možemo izbaci prethodne redove u matrici kako ju punimo

Ovaj redak je temeljen samo na ovom prethodnom

		a	c	t	c	g
	0	-1	-2	-3	-4	-5
a	-1	1	0	-1	-2	-3
c	-2	0	2	1	0	-1
a	-3	-1	1	2	1	0
g	-4	-2	0	1	2	2
t	-5	-3	-1	1	1	2
a	-6	-4	-2	0	1	1
g	-7	-5	-3	-1	0	2

Ušteda u prostoru

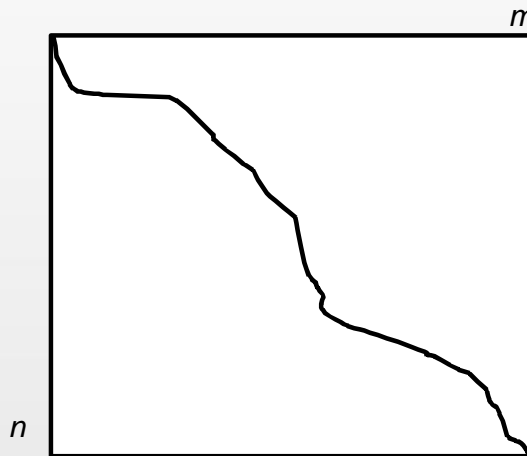
- ***Svaki redak*** tablice sadrži rezultate poravnanja ***prefiksa*** lijeve sekvence ***sa svim prefiksima*** sekvence na vrhu:

Rezultati
poravna-
nja **aca**
sa svim
prefiksi-
ma od
actcg

		a	c	t	c	g
	0	-1	-2	-3	-4	-5
a	-1	1	0	-1	-2	-3
c	-2	0	2	1	0	-1
a	-3	-1	1	2	1	0
g	-4	-2	0	1	2	2
t	-5	-3	-1	1	1	2
a	-6	-4	-2	0	1	1
g	-7	-5	-3	-1	0	2

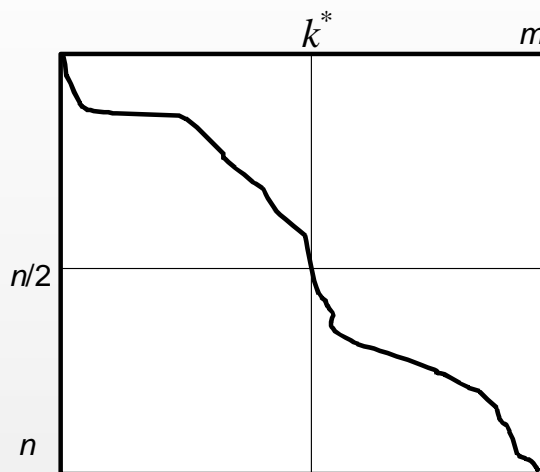
Poravnanje u linearnom prostoru

- Hirschberg (1975)
- Razmotrimo put poravnanja u matrici



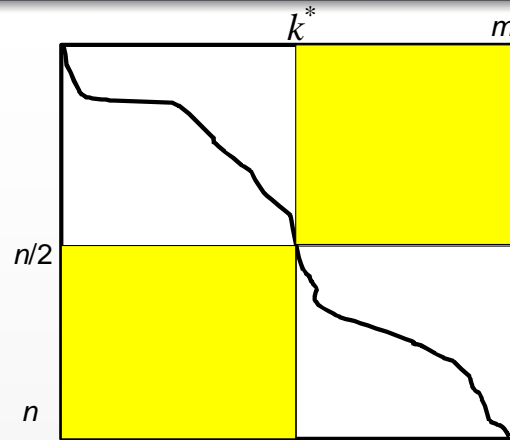
Poravnanje u linearnom prostoru

- Pretpostavimo da znamo da put optimalnoga poravnanja prolazi kroz ćeliju $(n/2, k^*)$



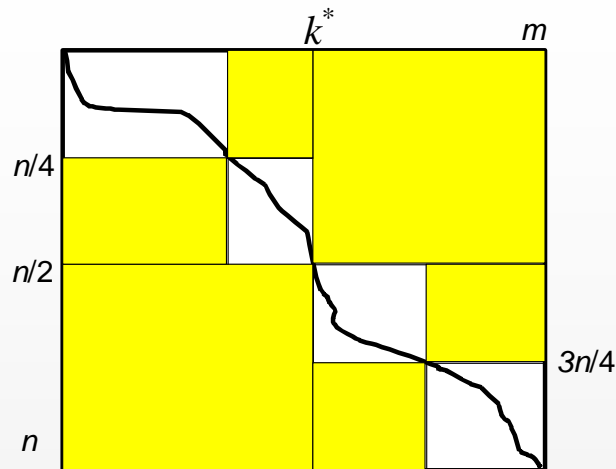
- Znajući to možemo riješiti problem spajajući putove za dijagonalne kvadrante.

Poravnanje u linearnom prostoru



- Važno je primijetiti da možemo ignorirati kvadrante na pomoćnoj dijagonali
- Bez obzira na to gdje se nalazi k^* površina žutih kvadrata je točno polovica ukupne
- Možemo ponavljati ovaj proces reducirajući količinu prostora potrebnog za pronalazak optimalnoga poravnanja.

Poravnanje u linearnom prostoru



- Ponavljajući ovaj proces dok ne dođemo do jedne ćelije možemo izračunati cjelokupni put poravnanja.
- Vremenska složenost
 $O((1+1/2+1/4+\dots) \times (n \times m)) = O(2 \times n \times m)$

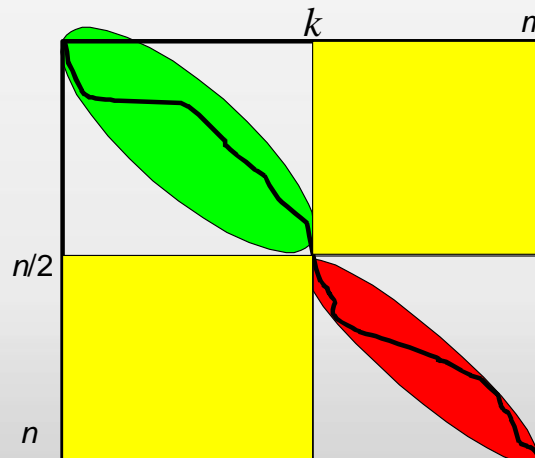
Poravnanje u linearnom prostoru

- Kako pronaći ćeliju $(n/2, k^*)$?
- **Definicija.** Ako s α^r označimo reverzni niz α .
- **Definicija.** $F^r(i, j)$ je sličnost prvih i znakova od S_1^r s prvim j znakovima od S_2^r .

$$\begin{array}{l} S_1^r \text{ --- } i \text{ --- } n \\ S_2^r \text{ --- } j \text{ --- } m \end{array}$$

Poravnanje u linearnom prostoru

- **Lema:** $F(n, m) = \max_{0 \leq k \leq m} [F(n/2, k) + F(n/2, m-k)]$
- Rješenje poravnanja $F(n, m)$ je suma kraćih poravnanja $F(n/2, k)$ & $F(n/2, m-k)$ gdje je k odabran tako da daje najveću sumu.
- U praksi uzmemo dva retka u sredini i nađemo maksimalno poravnanje.



Hirschberg primjer – korak 1

	-	A	G	C	A	T	G	C	-
-	0	-1	-2	-3	-4	-5	-6	-7	
A	-1	2	1	0	-1	-2	-3	-4	
C	-2	1	1	3	2	1	0	-1	
A	-3	0	0	2	5	4	3	2	
A	-4	-1	-1	1	4	4	3	2	
T									
C									
C									
-									

Hirschberg primjer – korak 2

	-	A	G	C	A	T	G	C	-
-									
A									
C									
A									
A	-4	-1	-1	1	4	4	3	2	
T		-1	0	1	2	3	0	0	-3
C		-2	-1	1	-1	0	1	1	-2
C		-4	-3	-2	-1	0	1	2	-1
-		-7	-6	-5	-4	-3	-2	-1	0

Hirschberg primjer – korak 3

	-	A	G	C	A	T	G	C	-
-									
A									
C									
A									
A	-4	-1	-1	1	4	4	3	2	
T		-1	0	1	2	3	0	0	-3
C									
C									
-									

Ograničeno dinamičko programiranje koristeći udaljenost uređivanja

Pretpostavimo da znamo da su x i y vrlo slični

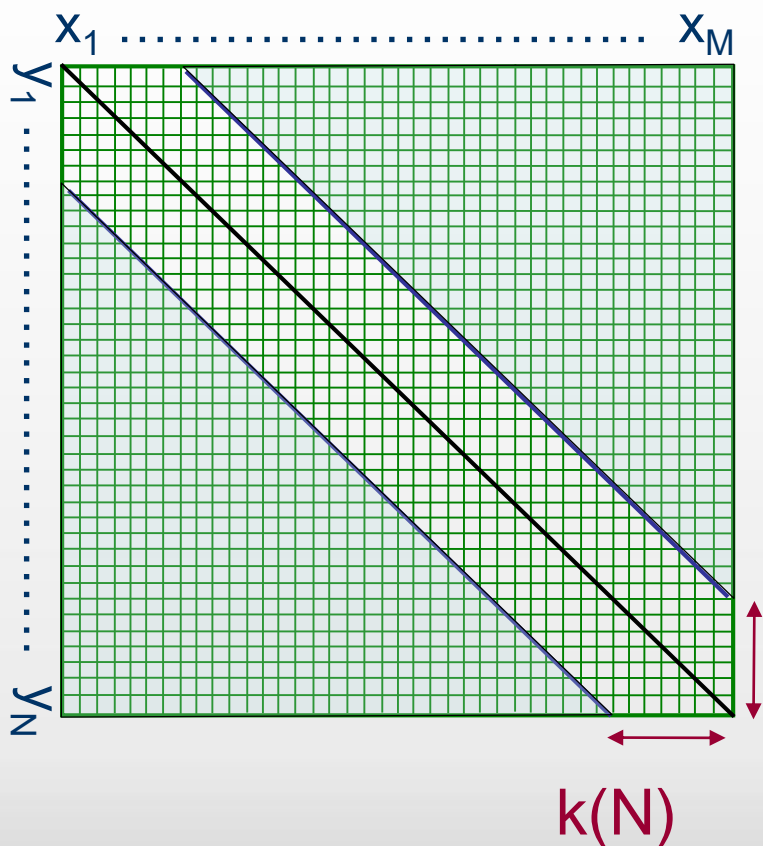
Pretpostavka: broj praznina(x, y) $< k(N)$ (uz $N > M$)

onda, $\begin{matrix} x_i \\ | \\ y_j \end{matrix}$ implicira $|i - j| < k(N)$

Možemo poravnati x i y efikasnije koristeći:

Vrijeme, Prostor: $O(N \times k(N)) \ll O(N^2)$

Ograničeno dinamičko programiranje



Inicijalizacija:

$F(i,0)$, $F(0,j)$ nedefinirani za $i,j > k$

Iteracija:

for $i = 1 \dots M$

for $j = \max(1, i - k) \dots \min(N, i + k)$

$$F(i, j) = \max \begin{cases} F(i - 1, j - 1) + z(x_i, y_j) \\ F(i, j - 1) - d, \text{ if } j > i - k(N) \\ F(i - 1, j) - d, \text{ if } j < i + k(N) \end{cases}$$

Kraj: isti kao klasični algoritam

Ograničeno - primjer

d=3

A-CAATCC

AGCA-TGC

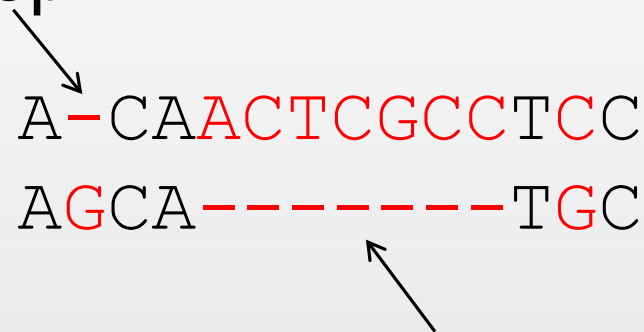
	-	A	G	C	A	T	G	C
-	0	-1	-2	-3				
A	-1	2	1	0	-1			
C	-2	1	1	3	2	1		
A	-3	0	0	2	5	4	3	
A		-1	-1	1	4	4	3	2
T			-2	0	3	6	5	4
C				0	2	5	5	7
C					1	4	4	7

Procijep

- Procijep (*engl. gap*) u poravnanju je maksimalan podniz susjednih praznina u bilo kojem nizu poravnanja

Ovo je procijep

A-CAACTCGCC TCC
AGCA-----TGC



Još jedan procijep

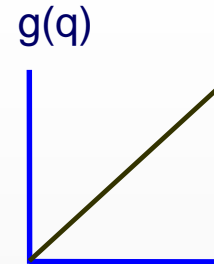
Kažnjavanje procijepa

- Procijepi u poravnanju se mogu modelirati kao uzastopni niz brisanja nukleotida gdje je kazna linearna funkcija duljine
 - To smo do sada pretpostavljali.
 - U mnogim primjenama sama duljina procijepa je relativno nevažna.
- Primjeri pojavljivanja procijepa:
 - Brisanja introna pri formiranju mRNA.
 - Ubacivanja mobilnih elementa u niz DNA kao što su transpozoni.
 - Mutacija može uzrokovati brisanje/umetanje duljeg podniza. Takva vrsta mutacija može biti jednako vjerojatna kao brisanje/umetanje jednog nukleotida.

Preciznije ocjenjivanje procijepa

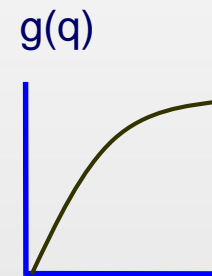
- Jednostavni linearni model:

- duljina procijepa q
- cijena $q \times d$



- Konveksna funkcija kažnjavanja procijepa:

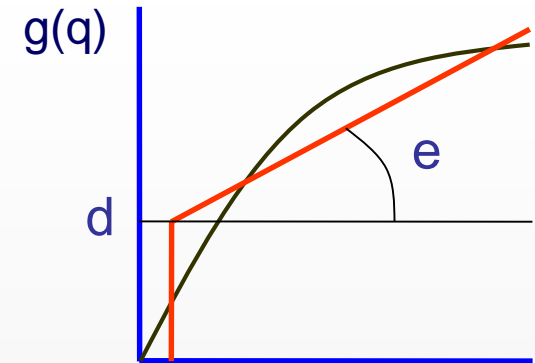
- $g(q)$ takva da za sve q ,
 $g(q + 1) - g(q) \leq g(q) - g(q - 1)$
- Algoritam: $O(N^3)$ vrijeme, $O(N^2)$ prostor



Kompromis: afina procijepi

$$g(q) = -d - (q - 1) \times e$$

| |
procijep procijep
otvaranje proširenje



- Potrebno je na pozicijama x_i i y_i za vrijeme dinamičkoga programiranja zapamtiti jesmo li usred procijepa:
 - Ako jesmo u tom slučaju kazna je samo dodatna praznina izračunata u $(q-1) \times e$ izrazu
 - Ako nismo otvaramo novu prazninu s konstantnom kaznom d