ШИНЖЛЭХ УХААН ТЕХНОЛОГИЙН ИХ СУРГУУЛЬ
Мэдээлэл, Холбооны Технологийн Сургууль

F.CS213 Биоалгоритм

Finding Patterns in Sequences

Дараалал дээрх паттерн илрүүлэх

Лекц 4

Багш ***Ганбаатарын ГАНБАТ.*** *Өрөө* ***#304,*** *ganbatg@must.edu.mn,* ***99999467***

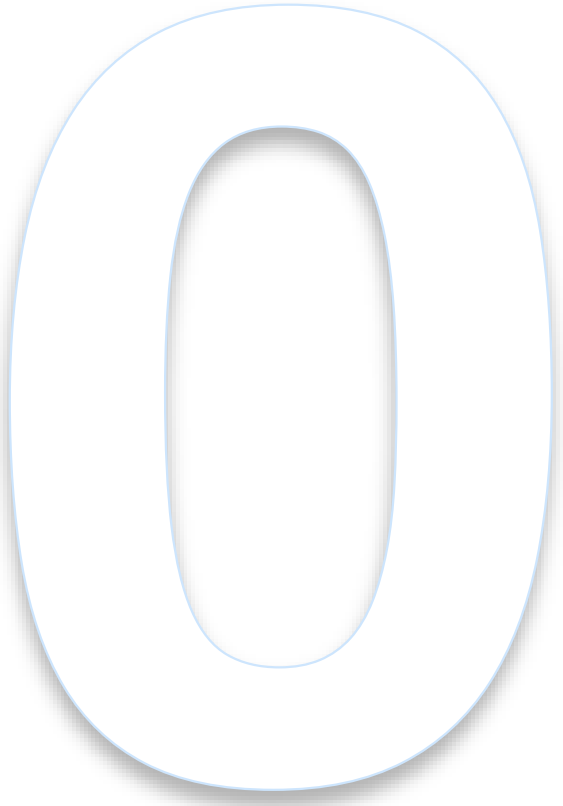
Лекцийн агуулга

- Паттерн илрүүлэх
- Тогтмол паттерн илрүүлэх

  гэнэн(naive) алгоритм
- Хьюристик (Heuristic) алгоритм:

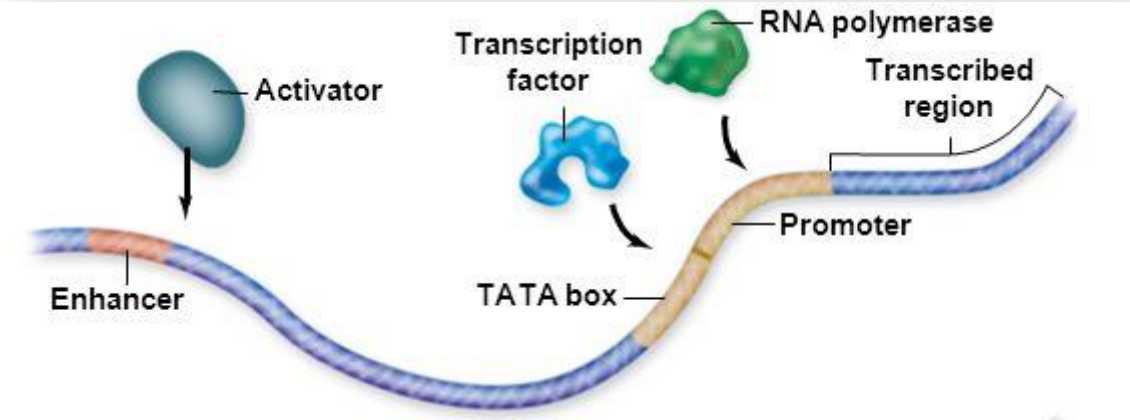
  Бойер-Мур
- Төгсгөлөг төлөвт детерминистик

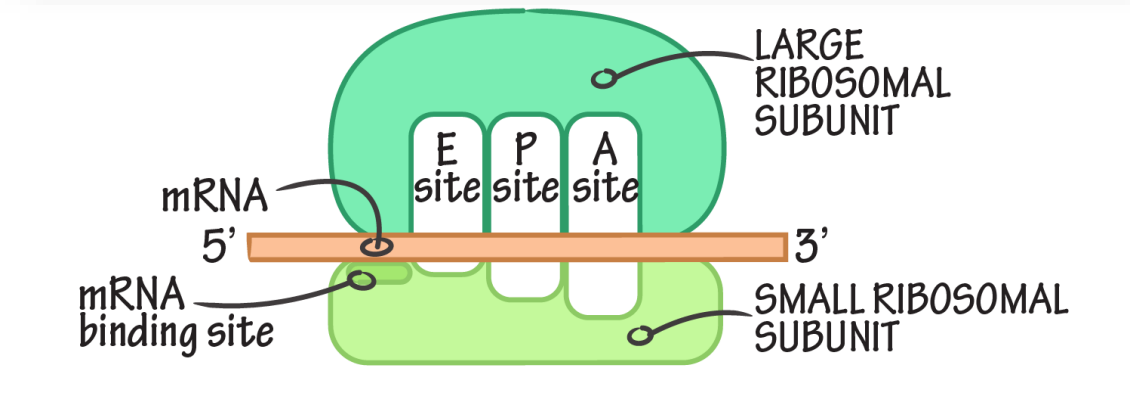
  автомат (Deterministic Finite Automata)
- Regular Expressions



Паттерн илрүүлэх

Амин бие, ялангуяа комплекс организм (хүн)–ын геномын хувьд паттерны урт, давтагдах тоо зэрэг нь паттерн илрүүлэх ач холбогдлыг нэмэгдүүлдэг.



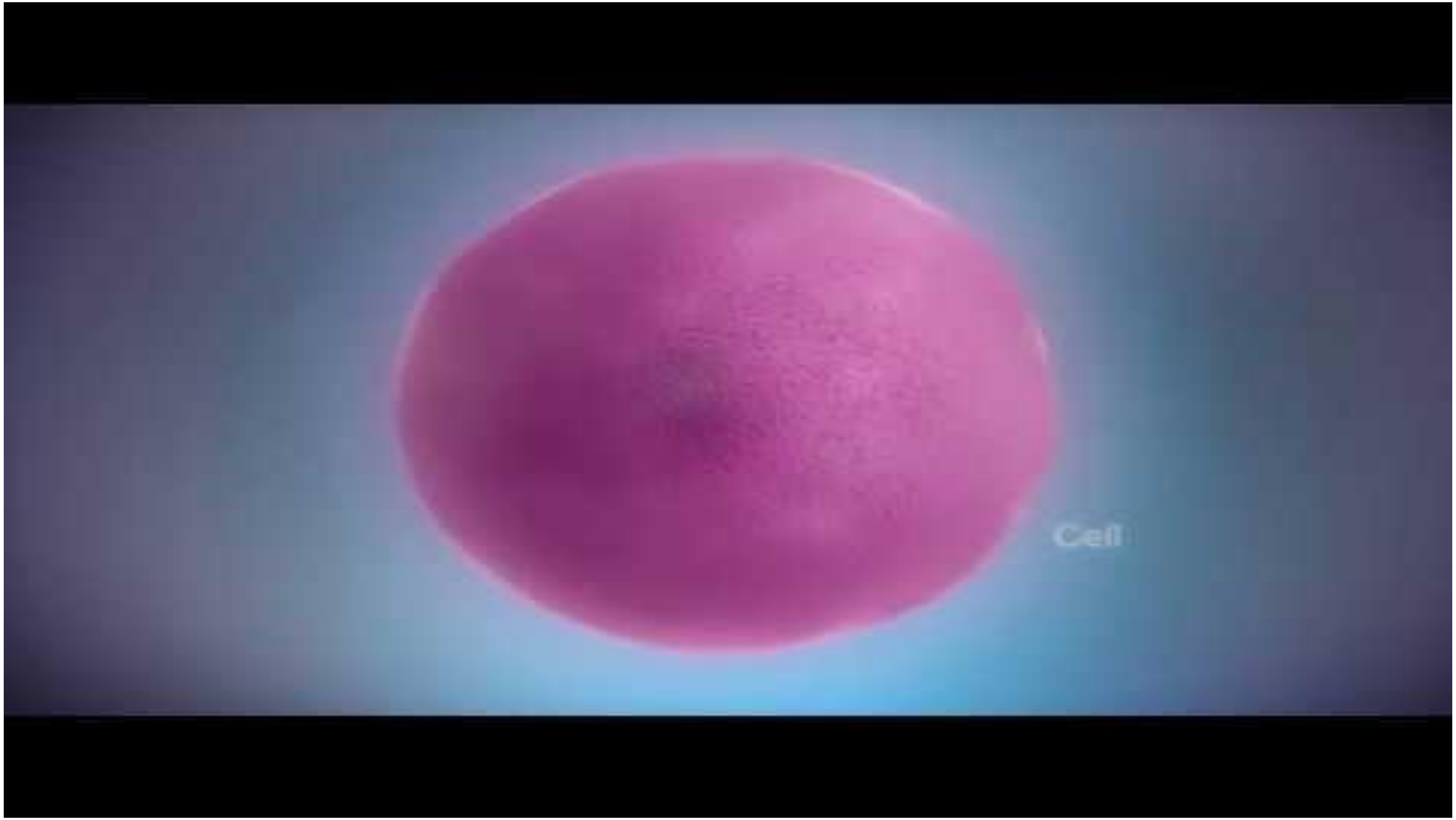


Паттернууд нь тухайн молекулын тусгай үүргүүдээс нягт хамааралтай байдаг. Жнь :

- *Онцлог паттернууд*: тусгай үүрэг бүхий *уургийн домайн (protein domains)*
  - Бэхлэгдэх хэсгүүд (Binding sites)
    - Энзим дээрх Лиганд холбогдох зориулалттай дэд хэсэг
    - Зохицуулалт (regulatory)-ын уураг дээрх DNA молекул холбогдох зориулалттай дэд хэсэг
- *Нуклойтид паттернууд*: DNA дэд хэсгүүд
  - Ген экспрешны зохицуулалтанд үед бэхлэгдэх хэсэгтэй холбогддог. (promoters, enhancers, transcription factors ...)

ПАТТЕРН ИЛРҮҮЛЭХ

Уургийн синтезийн анимэшн



- Эсийн бүтэц, функц, -Зургууд, -Электрон микроскоф

Тогтмол паттерн илрүүлэх гэнэн(naive) алгоритм

Даалгавар: s (N>k байх N урттай) дараалалд p (k урттай) паттернийг хайх

- k урттай байх s-ийн боломжит бүх дэд дарааллыг авч үзнэ
- Дэд дарааллуудын тэмдэгт бүрээр нь p-тэй харьцуулна.
  - Хэрэв дэд дараалал нь бүх тэмдэгт паттернтэй тохирч байвал p-ийн тохиолдлыг бүртгэнэ.
  - Үгүй бол дараагийн дэд дарааллыг шалгана.

```
seqDNA = "ATAGAATAGATAATAGTC"
print( search_first_occ(seqDNA, "GAAT") )
print( search_first_occ(seqDNA, "TATA") )
print( search_all_occurrences(seqDNA, "AAT") )
```

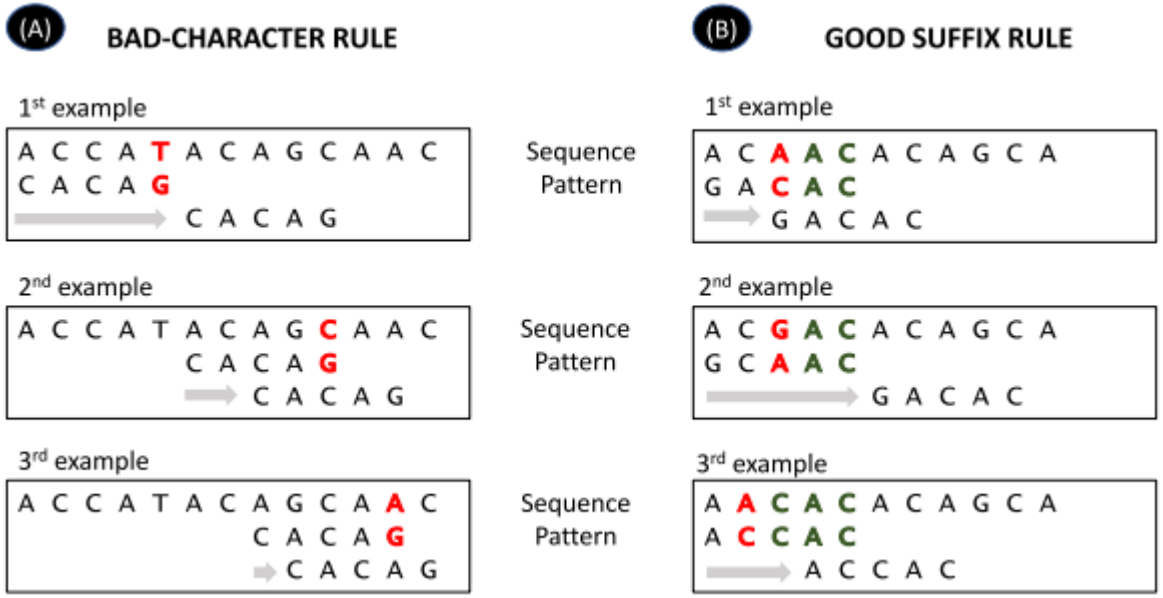
Хялбар функцүүд : s.find(p), s.count(p), in

```
def search_all_occurrences(seq, pattern):
    res = []
    for i in range(len(seq)-len(pattern)+1):
        j = 0
        while j < len(pattern) and pattern[j]==seq[i+j]:
            j = j + 1
        if j == len(pattern):
            res.append(i)
    return res
```

```
def search_first_occ(seq, pattern):
    found = False
    i = 0
    while i <= len(seq)-len(pattern) and not found:
        j = 0
        while j < len(pattern) and pattern[j]==seq[i+j]:
            j = j + 1
        if j== len(pattern): found = True
        else: i += 1
    if found: return i
    else: return -1
```

Хьюристик (Heuristic) алгоритм: Бойер-Мур

Дараалал дээр нэгээс олон тэмдэгтээр шилжихэд чиглэсэн зарчим дээр суурилдаг.



- *Тохиромжгүй тэмдэгтийн дүрэм (Bad-character rule):* Ялгаатай тэмдэгт олдсон үед дарааллын тэмдэгт паттерн дээр олдох байрлал хүртлэх уртаар шилжүүлнэ.
- *Тохиромжтой залгаврын дүрэм (Good suffix rule):* Ялгаатай тэмдэгт хүртлэх хэсэг паттерн дээр олдох хүртлэх уртаар шилжинэ

- Гэнэн алгоритмын тооцооллын хүндрэл нь тооцооллын үр ашгийг үгүй хийдэг.
- Паттерн хайлтын тооцооллын үр ашгийг дээшлүүлэх, хайлтыг хурдасгахын тулд паттерн бүтцийг ашигладаг өөр алгоритмууд байдаг
- Тэмдэгтүүдийн харьцуулалтын тоог багасгадаг.
- Хамгийн муу тохиолдолд тооцооллын хүндрэл нь гэнэн алгоритмтай ижил байх ч ихэнх тохиолдолд гүйцэтгэлд мэдэгдэхүйц ашигтай.
- Гэнэн алгоритмын нэгэн адил дарааллыг зүүнээс баруун тийш унших боловч паттерны харьцуулалтыг баруунаас зүүн тийш хийдэг.

ХЬЮРИСТИК АЛГОРИТМ

Урьдчилсан боловсруулалт (Pre-processing)

- Алгоритмыг үр ашигтай болгох буюу тухайн тохиолдол бүрт аль дүрмийг хэрэглэж болохыг хурдан шалгахын тулд хайлт эхлүүлэхээс өмнө шаардлагатай мэдээллийг ашигтай байж болох өгөгдлийн бүтцэд хадгална.
- Шилжих урт нь зөвхөн паттернаас хамаардаг тул ямар дараалал байхаас үл хамааран шаардлагатай мэдээллийг бэлэн байлгахын тулд энэхүү урьдчилсан боловсруулалтыг зөвхөн паттерны хувьд авч үзнэ.
- bad-character rule:
  - Dictionary төрөл ашиглана: *Түлхүүр*: Боломжит бүх тэмдэгт, *Утга*: тухайн тэмдэгтийн паттер дах хамгийн баруун талын байрлал. Байхгүй бол -1
  - Шилжих уртыг хурдан тооцоолох: *ялгаатай тэмдэгтийн паттен дахь байрлал – dictionary дахь утга*.
  - Шилжих утга сөрөг байж болно, энэ тохиолдолд тухайн давталт дотор алгасна гэсэн үг.
- good suffix rule ?
  - Паттерн дээрх ялгаатай тэмдэгтийн байрлалаас хамааран шилжих уртыг хадгалах жагсаалт байгуулна.

- Энэ алгоритмыг хэрэгжүүлэх **Python** классыг сурах бичиг дээрээс үзнэ үү. 5-р бүлэг, 112-р хуудас, **class BoyerMoore**
  - bad-character rule - **process_bcr**
  - good suffix rule - **process_gsr**

Төгсгөлөг төлөвт детерминистик автомат (Deterministic Finite Automata)

DFA формал тодорхойлолт $M = (Q, A, q_0, \delta, F)$	• Q нь төлвийн олонлог, • A нь тэмдэгтийн цагаан толгой,	• $q_0 \in Q$ нь эхлэх төлөв, • $\delta: Q, A \rightarrow Q$ нь шилжилтийн функц, • F нь зогсох төлвийн олонлог

- Тэмдэгтүүдийг дарааллыг баруун талаас нь эхлэн уншдаг, дотоод төлвөө өөрчлөх замаар тэмдэгтүүд дээр боловсруулалт хийн ажилладаг Төгсгөлөг төлөвт автоматыг тодорхойлж болно.
  - Шинэ төлөв нь өмнөх төлөв болон уншиж байгаа тэмдэгтээс хамаардаг.
  - Өгөгдсөн паттерны хувьд боломжит цагаан толгой, төлвүүд болон шилжилтийн функцийг тодорхойлсноор DFA нь паттерн хайлтыг гүйцэтгэхэд хэрэглэж болно.
- $Q = \{0, 1, \dots, m\}$ ба энд m нь паттерний урт. DFA нь k гэсэн төлөвт байна гэдэг нь өмнөх байрлал дээр дараалал паттерны эхний k тэмдэгттэй тохирсон гэсэн үг.
- $q_0 = 0$ ба $F = \{m\}$. Зогсох төлөв нь паттерн илэрсэн байх нэг л тохиолдол байна.
- *Шилжилтийн функц (transition function)*-ийг байгуулах нь бас нэг чухал алхам.

DFA Шилжилтийн функц

$$\delta(k, a) = max\_overlap(p_0 \dots p_{k-1}\ a, p)$$

- p нь паттерн,
- p_i нь паттерний i дэх тэмдэгт,

- Хэрэв $k-1$ төлөвт байх үед дараалал дээрх дараагийн тэмдэгт нь паттерний k дахь тэмдэгттэй ижил бол k төлөвт шилжинэ.
- Үгүй бол ялгаатай тэмдэгт олдож, автоматаар $q_0 = 0$ төлөв рүү буцах ёстой.
  - Гэхдээ энэ нөхцлөөс өмнөх тэмдэгтүүд нь өөр нэг тохиолдолд паттернтай давхцсан байж болно.
  - Тиймээс дараалал болон паттерний эхний $k-1$ тэмдэгт давхцаж байгаа эсэхийг шалгах хэрэгтэй.
- Эдгээр дарааллын хамгийн их давхцлын урт тухайн тэмдэгтийн хувьд дараагийн төлөв болно.

```
def overlap(s1, s2):
    maxov = min(len(s1), len(s2))
    for i in range(maxov,0,-1):
        if s1[-i:] == s2[:i]: return i
    return 0
```

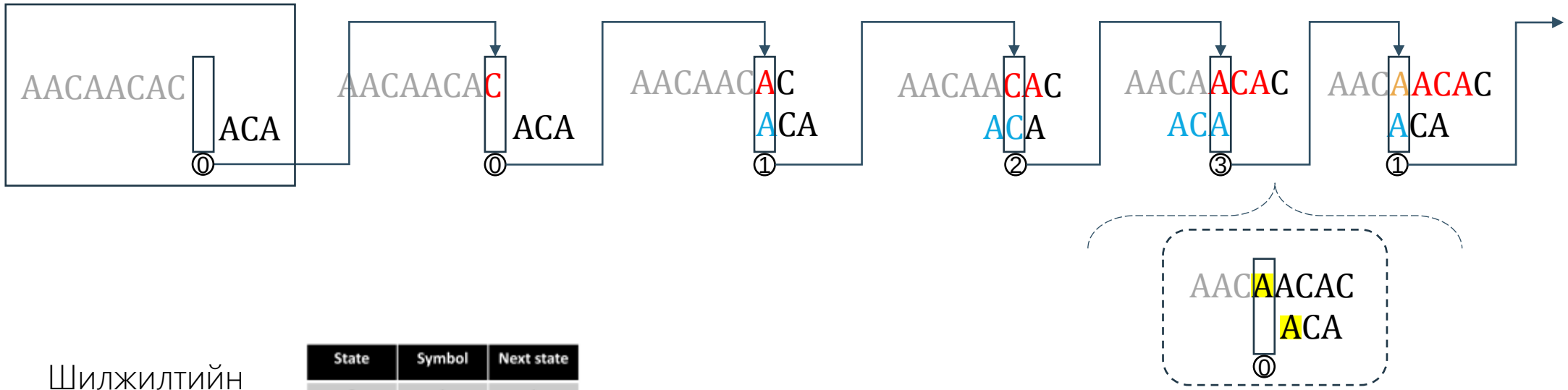
s ба t хоёр дарааллын хамгийн урт давхцал:

- x-ийн хамгийн их утга
- s-ийн сүүлийн x тэмдэгтүүд t-ийн эхний x тэмдэгтүүдтэй ижил байх.

DFA Жишээ

Sequence: $AACAACAC$

Alphabet: A, C
Pattern: **A C A**



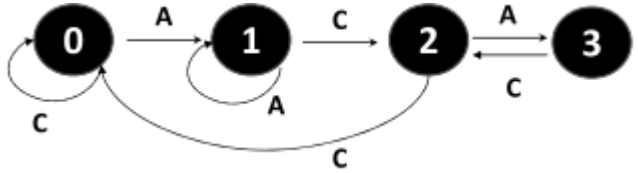
Шилжилтийн функц

↓

Шилжилтийн хүснэгт

State	Symbol	Next state
0	A	1
0	C	0
1	A	1
1	C	2
2	A	3
2	C	0
3	A	1
3	C	2

Sequence		C	A	C	A	A	C	A	A
State	0	0	1	2	**3**	1	2	3	1
Occurrence					1			3	



Цагаан толгойн тэмдэгт A ба C бөгөөд паттерн нь "ACA" байх автомат.

Regular Expressions

Regular Expressions (REs) нь орчин үеийн бүх программчлалын хэлэнд байдаг бөгөөд тэмдэгт мөрөнд хайлт хийх паттернийг уян хатан байдлаар илрүүлэх боломжийг олгодог.

- Тэмдэгтүүдийн заримыг паттерн илэрхийлэх *мета-тэмдэгт (meta-characters)* болгон ашигладаг. "." – нэг тэмдэгт, "^" - үгүйсгэл үүсгэнэ. "|" - логик эсвэл (disjunction), "$" нь төгсгөл, "^" эхлэл.
- Давталтыг зөвшөөрнө
  - * - Паттерн нь 0 ба түүнээс их давтагдана
  - \+ - Паттерн нь 1 ба түүнээс их давтагдана
  - ? - 0 эсвэл 1 давтагдана (паттерн байна, эсвэл байхгүй)
  - $\{n\}$ - Яг n удаа, n нь бүхэл тоо;
  - $\{m,n\}$ - m ба n-ын хооронд давтагдана, m ба n нь бүхэл тоо, $n \geq m$
- Хаалт ашиглан нэг тэмдэгт эсвэл бүлэг тэмдэгтүүдэд хэрэглэж болно.

  [A-Z] - Том үсэг
  [a-z] - Жижиг үсэг
  [A-Za-z] - Дурынүсэг
  [0-9] - Дурын тоо
  [ACTGactg] - DNA нуклойтед тэмдэгт (том, жижиг)
  [ACDEFGHIKLMNPQRSTVWY] Амин хүчил тэмдэгт (Том үсэг)

- \s – includes all white space (spaces, newlines, tabs, etc);
- \S – is the negation of the previous, thus matches with all non-white-space characters;
- \d – matches with digits;
- \D – matches with non-digits.

Function	Description
re.search(*regexp, str*)	checks if *regexp* matches *str*; returns results on the first match
re.match(*regexp, str*)	checks if *regexp* matches *str* in the beginning of the string
re.findall(*regexp, str*)	checks if *regexp* matches vstr; returns results on all matches as a list
re.finditer(*regexp, str*)	same as previous, but returns results as an iterator

RE Био дарааллын шинжилгээнд хэрэглэх нь

Пайтоны **re** сан: RE-тэй ажиллах хэд хэдэн хэрэгслийг багтаасан бөгөөд тэдгээрт тохирох тэмдэгт мөрийг шалгах боложмтой

```
def validate_dna_re (seq):
    from re import search
    if search("[^ACTGactg]", seq) != None:
        return False
    else:
        return True

>>> validate_dna_re("ATAGAGACTATC
True
>>> validate_dna_re("ATAGAGACTAXT
False
```

```
def largest_protein_re (seq_prot):
    import re
    mos = re.finditer("M[^_]*_", seq_prot)
    sizem = 0
    lprot = ""
    for x in mos:
        ini = x.span()[0]
        fin = x.span()[1]
        s = fin - ini + 1
        if s > sizem:
            lprot = x.group()
            sizem = s
    return lprot
```

```
def translate_codon_re (cod):
    import re
    if re.search("GC.", cod): aa = "A"
    elif re.search("TG[TC]", cod): aa = "C"
    elif re.search("TA[AG]|TGA", cod): aa = "_";
    else: aa = ""
    return aa
```

- Таамаг ген илрүүлэх RE: **M[^_]*_**
  - ‘**M**’-ээр эхлэнэ. ‘**_**’-ээр төгсөнө.
  - Дунд хэсэгт ‘**M**’ тохиолдож болно
  - Дунд хэсэгт ‘**_**’ тохиолдохгүй
  - Жнь: ‘**M...M...M..._**’

- "TA[AG]|TGA"
  - TA – дээр A, G аль нэгийг нь нэмнэ
  - эсвэл TGA гурвал

RE Уургийн мотиф (Motifs)-ийг ирлүүлэх.

Уурийн хувьд биологийн функцэд нь чухал үүрэгтэй байдаг хэд хэдэн төрлийн паттернийг ерөнхийдөө *мотиф* гэнэ.

"Zinc finger RING-type signature" (PS00518) motif:

"C-x-H-x-[LIVMFY]-C-x(2)-C-[LIVMYA]"

```
def find_zync_finger(seq):
    from re import search
    regexp = "C.H.[LIVMFY]C.{2}C[LIVMYA]"
    mo = search(regexp, seq)
    if (mo != None):
        return mo.span()[0]
    else:
        return -1

def test():
    seq = "HKMMLASCKHLLCLKCIVKLG"
    print(find_zync_finger(seq))

test()
```

- Ихэвчлэн уургийн домэйнуудтай холбоотой, Тухайн тусгай биологийн функцийг бий болгоход чиглэсэн бодитой гурван хэмжээст тогтоцыг илрүүлэх.
- Prosite ӨС (http://prosite.expasy.org/) нь уургийн мотифуудын янз бүрийн форматтай өгөгдлийг агуулдаг. Амин хүчлийн 20 тэмдэгт, тусгай мета тэмдэгтүүд бүхий дүрслэлийн хэлтэй:
  - амин хүчил бүрийг нэг тэмдэгээр илэрхийлнэ;
  - [ ] доторх амин хүчлийн жагсаалт нь боломжит амин хүчлүүдийн илэрхийлнэ;
  - "x" тэмдэг нь дурын амин хүчлийг илэрхийлнэ;
  - амин хүчлийн дараах хаалт доторх тоо нь тэдгээр амин хүчлүүдийн тохиолдох тоог илэрхийлнэ;
  - хаалтанд таслалаар тусгаарлагдсан хос тоо нь эхний болон хоёр дахь тооны хоорондох тохиолдлын тоон мужийг заана;
  - "-" тэмдгийг байрлалуудыг ялгахад ашигладаг.

RE Хязгаарлалтын (Restriction) энзимын хэрэглээ

Хязгаарлалтын энзим: Тодорхой дэд дараалал (паттерн эсвэл мотиф) агуулсан хэсгүүдээр нь DNA-г *огтолдог (cut)* уураг юм.

EcoRI хязгаарлалтын энзим нь "GAATTC" паттерныг агуулсан DNA дарааллыг "G" ба эхний "A" хоёрын дундуур тасалдаг.

```
def iub_to_RE (iub):
    dic = {"A":"A", "C":"C", "G":"G", "T":"T", "R":"[GA]", "Y":"[CT]"
   , "M":"[AC]", "K":"[GT]", "S":"[GC]", "W": "[AT]", "B":"[CGT]", "
   D":"[AGT]", "H":"[ACT]", "V":"[ACG]", "N":"[ACGT]"}

    site = iub.replace("^","")
    regexp = ""

    for c in site:
        regexp += dic[c]

    return regexp
```

Зарим энзимүүд *зорилтот (target)* бүсэд хувьсах урттай байдаг тул нуклейтидын дарааллал төдийгүй тодорхой бус байдлыг зөвшөөрдөг тэмдэгтүүд агуулсан цагаан толгойтой (IUPAC)

- Хязгаарлалтын хуваарь (дарааллыг таслах байрлалууд) нь молекул биологийн чухал хэрэгсэл.
  - Уг паттерн нь ***биологийн палиндром (biological palindrome)***, урвуу гүйцэлттэй ижил ойлгоно.
  - Хязгаарлалтын энзим нь DNA-ийн мушгианы дарааллыг хоёуланг нь тасалдаг
  - Яг ижил байрлалд огтолдоггүй тул молекул биологийн хувьд хуулалт, дараалалжуулалтыг дэмдэг.
- REBASE нь хязгаарлалтын энзимын ӨС (http://rebase.neb.com/),

АНХААРАЛ ТАВЬСАНД БАЯРЛАЛАА