به نام خدا

تمرین سری دوم
درس مقدمه ای بیوانفورماتیک
دکتر علی شریفی زارچی

فرزان رحمانی
۴۰۳۲۱۰۷۲۵

سوال اول

الف) ژنوم را به‌عنوان کتابخانه‌ای عظیم تصور کنید که در آن قسمت‌های خاص کتاب چندین بار در جلدهای مختلف تکرار می‌شوند. توالی‌بندی بخش‌های تکراری ژنوم مانند تلاش برای تعیین مکان و زمینه(context) دقیق آن قسمت‌های تکراری با اطلاعات محدود است.

چالش ها در توالی بخش های تکراری:

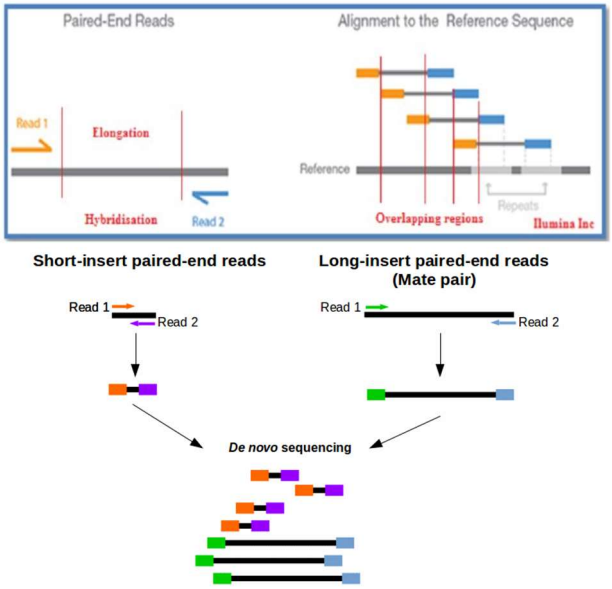
- ابهام در Read Alignment: توالی های تکراری، مانند تکرارهای پشت سر هم یا transposons، می توانند در چندین مکان در ژنوم تقریباً یکسان باشند. خوانش های کوتاه ایجاد شده توسط فناوری های توالی یابی به طور منحصر به فرد در یک منطقه خاص دشوار است که منجر به ابهام در Alignment می شود.
- محدودیت‌های طول Read ها: بسیاری از فناوری‌های توالی خوانی reads های کوتاهی ایجاد می‌کنند (مثلاً ۱۵۰ base pairs در توالی Illumina)، که اغلب کوتاه‌تر از مناطق تکراری هستند. این کار مانع می‌شود که خواندن‌ها کل بخش تکراری را در بر بگیرند و حل توالی کامل آنها را دشوار می‌کند.
- Coverage Bias: مناطق تکراری، مستعد بایاس در توالی هستند، با برخی از تکرارها بیش از حد یا کمتر ارائه شده که منجر به عدم دقت در coverage و assembly می شود.
- تغییرات ساختاری: تکرارها ممکن است در تعداد کپی یا ساختار بین نمونه ای individual متفاوت باشد و بازسازی ترتیب صحیح آنها را دشوارتر می کند.
- انتشار خطا: خطاها در فرآیند توالی یابی می توانند در مناطق تکراری تقویت شوند. یک خطای واحد در خواندن می‌تواند منجر به ناهماهنگی و assembly نادرست توالی تکراری شود.
- پیچیدگی توالی‌های تکراری: نواحی تکراری می‌توانند بسیار پیچیده باشند و از انواع مختلفی از تکرارها مانند تکرارهای پشت سر هم(tandem repeats)، تکرارهای پراکنده(interspersed repeats) و satellite DNA تشکیل شده‌اند. این پیچیدگی طراحی استراتژی های توالی کارآمد و pipeline های تجزیه و تحلیل را دشوار می کند.

راه های مقابله با این چالش ها:

- توالی خوانی طولانی(Long-Read Sequencing): پلتفرم‌هایی مانند PacBio و Oxford Nanopore خواندن طولانی‌تر (تا ده‌ها kilobases) تولید می‌کنند که می‌تواند مناطق تکراری را در بر بگیرد و مونتاژ را آسان‌تر کند.
- Paired-End and Mate-Pair Sequencing: با ایجاد خواندن از هر دو انتهای یک قطعه DNA و حفظ اطلاعات در مورد طول تقریبی قطعه اصلی، محققان می توانند اطلاعات زمینه ای اضافی را ارائه دهند که به رفع ابهامات تکراری منطقه کمک می کند.
- رویکردهای توالی ترکیبی(Hybrid Sequencing Approaches): ترکیب داده های خوانده شده طولانی و خواندن کوتاه می تواند از دقت بالای خواندن کوتاه و طول خواندن طولانی برای بهبود assembly و رفع تکرارها استفاده کند.
- الگوریتم‌های مونتاژ بهبود یافته(Improved Assembly Algorithms): ابزارهای بیوانفورماتیک پیشرفته مانند اسمبلرهای de novo (مانند SPAdes، Canu) و روش‌های مبتنی بر گراف (مانند نمودارهای اسمبلی) برای مدیریت تکرارها با استفاده از همپوشانی خواندن طراحی شده‌اند.

- نگاشت نوری و تکنیک‌های مقیاس کروموزوم(Optical Mapping and Chromosome-Scale Techniques): تکنیک‌هایی مانند نگاشت نوری Bionano Genomics و توالی‌یابی Hi-C اطلاعات ساختاری بیشتری را برای حل مناطق تکراری فراهم می‌کنند.
- توالی هدفمند(Targeted Sequencing): تمرکز بر مناطق تکراری خاص مورد علاقه می تواند عمق و دقت توالی را در این مناطق افزایش دهد.

ب) توالی یابی با Paired-end شامل توالی یابی هر دو انتهای یک قطعه DNA است که دو read را ارائه می دهد که با فاصله مشخصی از هم جدا می شوند. (در زیر دو شکل آمده اند که شهود بهتری ایجاد میکند.)



این استراتژی به ویژه برای بهبود دقت توالی یابی در بخش های تکراری موثر است:

- تثبیت Reads در مناطق منحصر به فرد(Anchoring Reads in Unique Regions): یک read از یک جفت ممکن است به طور منحصر به فرد به یک منطقه غیر تکراری مجاور تکرار نگاشت شود، در حالی که read دیگر در محدوده تکرار قرار می گیرد. اندازه قطعه شناخته شده به استنباط قرارگیری صحیح بخش تکراری کمک می کند.
- Bridging Repeats: Paired-end reads می توانند مناطق تکراری را در بر گیرند و به طور موثر توالی های منحصر به فرد را در هر طرف به هم پیوند دهند. این پل زدن مونتاژ را با ایجاد تداوم در تکرارها بهبود می بخشد.
- رفع ابهام: محدودیت فاصله بین paired reads به رفع ابهام‌های نادرست بالقوه کمک می‌کند و اطمینان حاصل می‌کند که تکرارها در بافت ژنومی صحیح خود قرار می‌گیرند.
- تصحیح خطا: داده های جفت شده پوشش اضافی(redundant coverage) را فراهم می کند، خطاها را کاهش می دهد و دقت call های پایه را در مناطق تکراری بهبود می بخشد.

مثال عملی: در یک ژنوم با تکرارهای پشت سر هم، paired-end reads ممکن است به اسمبل کننده ها اجازه دهد تا تعداد تکرار را با شمارش تعداد قطعاتی که توالی تکراری را پل می کنند، تعیین کنند، نه اینکه صرفاً بر خواندن های کوتاه در ناحیه تکرار شونده تکیه کنند.
یک منطقه repetitive در کروموزوم را در نظر بگیرید که در آن یک توالی ۲۰۰ جفت باز چندین بار ظاهر می شود. یک read از این منطقه می تواند با چندین مکان مطابقت داشته باشد. با این حال، اگر بدانیم که second read از همان قطعه تقریباً ۳۰۰ base pairs فاصله دارد و در یک جهت خاص است، می توانید عدم قطعیت mapping را به طور چشمگیری کاهش دهیم.
آن را مانند حل یک پازل تصور کنید که در آن نه تنها تکه های جداگانه داریم، بلکه موقعیت نسبی آنها را نیز می دانیم. اطلاعات paired-end به عنوان "اتصال" اضافی عمل می کند.

ملاحظات بیشتر:

- اثربخشی استراتژی‌های paired-end به اندازه fragment ، طول read و ویژگی‌های منطقه تکراری خاص بستگی دارد.
- برخی از عناصر تکراری، مانند تکرارهای بسیار طولانی یا sequences بسیار مشابه، حتی با رویکردهای paired-end چالش برانگیز باقی می مانند.

با ادغام paired-end reads با نرم‌افزار assembly پیچیده، محققان می‌توانند ساختار و توالی نواحی ژنومی تکراری را که گامی مهم در بازسازی دقیق ژنوم است، بهتر حل کنند.

همچنین نحوه کارکرد آن به طور خلاصه در ادامه آمده است:

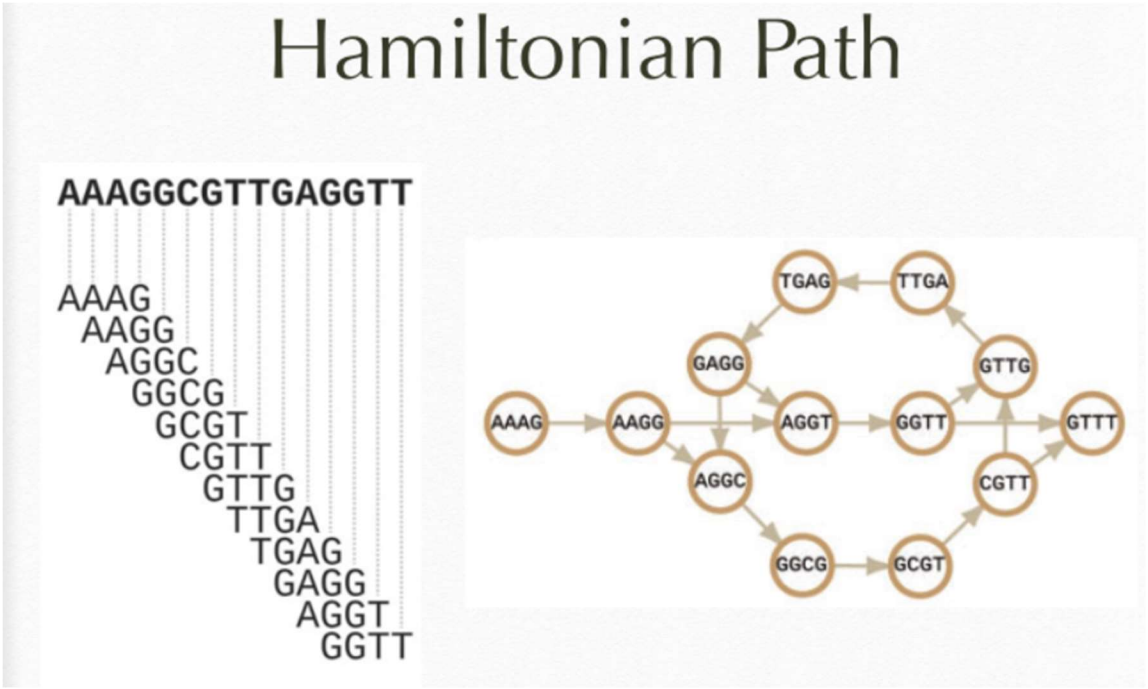
۱. Fragment Preparation: یک نمونه DNA به قطعات، معمولا ۲۰۰-۵۰۰ جفت باز تقسیم می شود. بر خلاف توالی یابی تک انتهایی که فقط یک انتهای قطعه را توالی می کند، توالی یابی paired-end از هر دو انتهای هر قطعه خوانده می شود.
۲. Contextual Mapping:
- دو read از هر قطعه به طور جداگانه توالی می شوند اما دانش در مورد مجاورت و جهت اصلی خود را حفظ می کنند.
- هنگامی که این read ها به ژنوم مرجع نگاشت می شوند، فاصله شناخته شده بین آنها اطلاعات زمینه ای مهمی را ارائه می دهد.
- در مناطق تکراری، این زمینه اضافی به رفع ابهام مکان های mapping بالقوه کمک می کند.

سوال دوم

این مسئله در مورد بازسازی دو توالی اولیه (ژن) از raed های توالی مخلوط شده است. الگوریتمی که برای حل این مسئله استفاده میکنیم، شامل یافتن همپوشانی(overlap) بین دنباله ها و استفاده از آنها برای بازسازی دنباله های اصلی است. در ادامه توضیح گام به گام آمده است:

الگوریتم

۱. تجزیه ورودی:
   - قطعات دنباله را بخوانیم و parse کنیم.
۲. تشخیص همپوشانی:
   - حداکثر همپوشانی بین پسوند یک قطعه(read) و پیشوند قطعه دیگر را پیدا کنیم. در این مسئله همان طور که میبینیم حداکثر طول هم پوشانی بین read ها برابر با عدد ۳ است.
۳. ساختن گراف:
   - از همپوشانی ها برای ایجاد یک گراف جهت دار استفاده میکنیم که در آن گره ها قطعات(reads) را نشان می دهند و یال ها نشان دهنده همپوشانی ها هستند. (شکل زیر که در اسلاید های درس موجود است به خوبی نشان دهنده این مرحله است.)

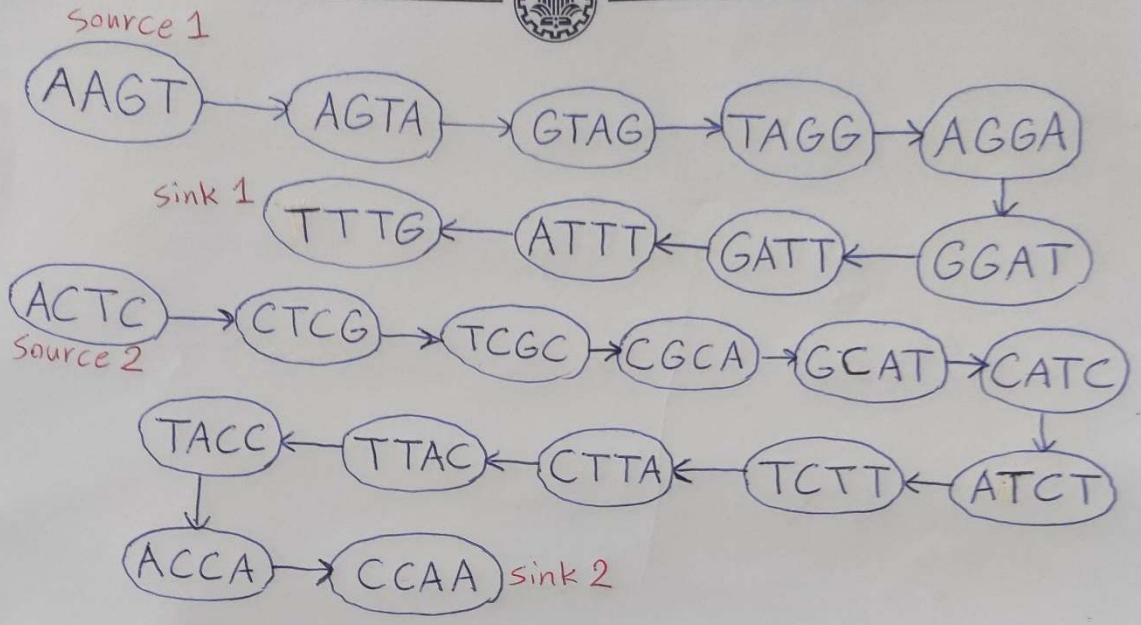


۴. مونتاژ توالی(**Sequence Assembly**):
   - از همپوشانی ها برای ادغام مکرر(iterative) قطعات در دنباله های بزرگتر استفاده میکنیم.
   - با اطمینان از عدم همپوشانی بین دو گروه، دو دنباله اصلی را شناسایی و جدا میکنیم.
۵. خروجی:
   - دو sequence بازسازی شده را بر می گردانیم.

در نهایت پس از اجرای این الگوریتم ژن های اولیه بازسازی می شوند که به صورت زیر هستند:

$$PrimarySequence_1 = gene_1 = AAGTAGGATTTG$$
$$PrimarySequence_2 = gene_2 = ACTCGCATCTTACCAA$$

راه حل اجرا شده در ادامه آمده است:



همان طور که می بینیم دوگراف جهت دار همبند تشکیل شد که هرکدام مربوط به یکی از ژن ها است. با شروع از source های دوگراف و تشکیل Hamilton Path و پیمایش تا sink ها دو ژن اولیه بازسازی می شوند.

ژن 1 = AAGTAGGATTTG

ژن 2 = ACTCGCATCTTACCAA

به طور مثال برای ژن 1 overlap ها به شکل زیر می شود:

```
AAGT
 AGTA
  GTAG
   TAGG
    AGGA
     GGAT
      GATT
       ATTT
        TTTG
```

reads for gene 1

ژن 1 = AAGTAGGATTTG

همچنین میتوانیم برای حل این سوال از کد زیر هم استفاده کنیم:

```python
from itertools import permutations
def find_overlap(seq1, seq2):
    """Finds the maximum overlap between suffix of seq1 and prefix of seq2."""
    max_overlap = 0
    overlap_seq = ""
    for i in range(1, min(len(seq1), len(seq2)) + 1):
        if seq1[-i:] == seq2[:i]:
            max_overlap = i
            overlap_seq = seq1 + seq2[i:]
    return max_overlap, overlap_seq
def assemble_sequences(reads):
    """Reconstruct two primary sequences from mixed reads."""
    sequences = list(reads)
    while len(sequences) > 2:
        max_len = -1
        max_seq = None
        seq_to_merge = None
        # Find the pair of sequences with the maximum overlap
        for seq1, seq2 in permutations(sequences, 2):
            overlap_len, merged_seq = find_overlap(seq1, seq2)
            if overlap_len > max_len:
                max_len = overlap_len
                max_seq = merged_seq
                seq_to_merge = (seq1, seq2)
        # Merge the sequences with the maximum overlap
        if seq_to_merge:
            sequences.remove(seq_to_merge[0])
            sequences.remove(seq_to_merge[1])
            sequences.append(max_seq)
    return sequences
# Input mixed reads
reads = [
    "AAGT", "TCTT", "AGTA", "CCAA", "GTAG", "ATCT", "TAGG",
    "ACCA", "AGGA", "GGAT", "GATT", "ATTT", "ACTC", "CTCG",
    "TCGC", "TTTG", "GCAT", "CATC", "CTTA", "CGCA", "TTAC", "TACC"
]
# Reconstruct the sequences
primary_sequences = assemble_sequences(reads)
# Output results
for i, seq in enumerate(primary_sequences):
    print(f"Primary Sequence {i+1}: {seq}")
```

سوال سوم

برای حل این مسئله تراز چندگانه (MSA) با استفاده از روش ارائه شده، باید مراحل الگوریتم را به دقت دنبال کنیم. در ادامه یک تفکیک دقیق از هر مرحله، آمده است.

مرحله ۱: امتیازات(score) تراز دو به دو را محاسبه کنیم.

ما امتیازهای هم ترازی زوجی D(S_i, S_j) را برای همه دنباله ها با استفاده از جدول امتیاز داده شده محاسبه می کنیم. امتیاز هم ترازی با تکرار کاراکترهای دو دنباله و جمع کردن مقادیر مربوط به جدول امتیاز، با معرفی شکاف ها در صورت لزوم محاسبه می شود.

جدول برنامه نویسی پویا

جدول تراز ردیف به ردیف ساخته می شود و کاراکترهای دنباله ها را با هم مقایسه می کند. فرض کنید dp[i][j] نشان دهنده امتیاز تراز بهینه برای اولین کاراکترهای i از S_1 و اولین کاراکترهای j از S_2 باشد. روابط پر کردن جدول عبارتند از:

1. $dp[i][0] = i \times \text{gap\_penalty}$
2. $dp[0][j] = j \times \text{gap\_penalty}$
3. For $i, j > 0$:

$$dp[i][j] = \max \begin{cases} dp[i-1][j-1] + \text{scoreboard}(S_1[i], S_2[j]) & \text{match/mismatch} \\ dp[i-1][j] + \text{gap\_penalty} & \text{insert gap in } S_2 \\ dp[i][j-1] + \text{gap\_penalty} & \text{insert gap in } S_1 \end{cases}$$

مرحله ۲: دنباله مرکزی(Center Sequence) را پیدا کنیم.

هنگامی که امتیازهای زوجی محاسبه شد، دنباله مرکزی S_c با به حداقل رساندن مجموع فواصل زوجی تعیین می شود:

$$\text{Center Sequence} = S_c = \arg\min_{S_i} \sum_j D(S_i, S_j)$$

مرحله ۳: همه دنباله ها را به دنباله مرکزی تراز کنیم.

برای هر دنباله S_i (به استثنای S_c)، با استفاده از تراز بهینه، آن را با S_c تراز کنیم.

مرحله ۴: تراز چندگانه نهایی را بسازیم.

همه ترازها را با شکاف های درج شده برای هماهنگی بین دنباله ها ترکیب کنیم. نتیجه مجموعه ای از توالی های هم تراز شده خواهد بود.

حال این الگوریتم را مر حله به مرحله اجرا میکنیم.

ابتدا هر چهار دنباله دو به دو به هم تراز میکنیم و جدول dp و تراز نهایی آنها را نشان می دهیم.

جدول و دنباله های تراز شده در صفحات بعد آمده است:

```
dp table of 'ACCCTGAACC' and 'ACTCGGAGC' :
[[  0.  -1.  -2.  -3.  -4.  -5.  -6.  -7.  -8.  -9.]
 [ -1.   2.   1.   0.  -1.  -2.  -3.  -4.  -5.  -6.]
 [ -2.   1.   5.   4.   3.   2.   1.   0.  -1.  -2.]
 [ -3.   0.   4.   4.   7.   6.   5.   4.   3.   2.]
 [ -4.  -1.   3.   3.   7.   9.   8.   7.   6.   6.]
 [ -5.  -2.   2.   5.   6.   8.  10.   9.   8.   7.]
 [ -6.  -3.   1.   4.   7.   9.  11.  10.  12.  11.]
 [ -7.  -4.   0.   3.   6.   8.  10.  13.  12.  12.]
 [ -8.  -5.  -1.   2.   5.   7.   9.  12.  13.  12.]
 [ -9.  -6.  -2.   1.   5.   7.   9.  11.  14.  16.]
 [-10.  -7.  -3.   0.   4.   7.   9.  10.  13.  17.]]

Aligned Sequences:
ACCCTGAACC
ACTCGG-AGC
----------------------------------------
```

```
dp table of 'ACCCTGAACC' and 'CTGGAATCT' :
[[  0.  -1.  -2.  -3.  -4.  -5.  -6.  -7.  -8.  -9.]
 [ -1.   0.   0.  -1.  -2.  -2.  -3.  -4.  -5.  -6.]
 [ -2.   2.   1.   2.   1.   0.  -1.  -2.  -1.  -2.]
 [ -3.   1.   1.   3.   4.   3.   2.   1.   1.   0.]
 [ -4.   0.   0.   3.   5.   4.   3.   2.   4.   3.]
 [ -5.  -1.   2.   2.   4.   6.   5.   5.   4.   6.]
 [ -6.  -2.   1.   5.   5.   5.   6.   6.   7.   6.]
 [ -7.  -3.   0.   4.   5.   7.   7.   7.   6.   8.]
 [ -8.  -4.  -1.   3.   4.   7.   9.   8.   7.   7.]
 [ -9.  -5.  -2.   2.   5.   6.   8.   8.  11.  10.]
 [-10.  -6.  -3.   1.   4.   5.   7.   7.  11.  10.]]

Aligned Sequences:
AC-CCTGAACC
-CTGGA-ATCT
----------------------------------------
```

```
dp table of 'ACCCTGAACC' and 'GCTAGGACC' :
[[  0.  -1.  -2.  -3.  -4.  -5.  -6.  -7.  -8.  -9.]
 [ -1.   0.  -1.  -1.  -1.  -2.  -3.  -4.  -5.  -6.]
 [ -2.   1.   3.   2.   1.   1.   0.  -1.  -1.  -2.]
 [ -3.   0.   4.   3.   2.   3.   3.   2.   2.   2.]
 [ -4.  -1.   3.   3.   3.   4.   5.   4.   5.   5.]
 [ -5.  -2.   2.   5.   4.   4.   5.   6.   5.   4.]
 [ -6.  -2.   1.   4.   5.   7.   7.   6.   8.   7.]
 [ -7.  -3.   0.   3.   6.   6.   7.   9.   8.   8.]
 [ -8.  -4.  -1.   2.   5.   6.   6.   9.   9.   8.]
 [ -9.  -5.  -1.   1.   4.   7.   8.   8.  12.  12.]
 [-10.  -6.  -2.   0.   3.   6.   9.   8.  11.  15.]]

Aligned Sequences:
ACCCTGAACC
-GCTAGGACC
----------------------------------------
```

```
----------------------------------------
dp table of 'ACTCGGAGC' and 'CTGGAATCT' :
[[ 0. -1. -2. -3. -4. -5. -6. -7. -8. -9.]
 [-1.  0.  0. -1. -2. -2. -3. -4. -5. -6.]
 [-2.  2.  1.  2.  1.  0. -1. -2. -1. -2.]
 [-3.  1.  4.  3.  3.  2.  1.  1.  0.  1.]
 [-4.  0.  3.  6.  5.  4.  3.  2.  4.  3.]
 [-5. -1.  2.  6.  9.  8.  7.  6.  5.  5.]
 [-6. -2.  1.  5.  9.  9.  8.  8.  8.  7.]
 [-7. -3.  0.  4.  8. 11. 11. 10.  9.  9.]
 [-8. -4. -1.  3.  7. 10. 11. 12. 12. 11.]
 [-9. -5. -2.  2.  6.  9. 10. 11. 15. 14.]]

Aligned Sequences:
ACTCGGAGC-
-CTGGAATCT
----------------------------------------
```

```
dp table of 'ACTCGGAGC' and 'GCTAGGACC' :
[[ 0. -1. -2. -3. -4. -5. -6. -7. -8. -9.]
 [-1.  0. -1. -1. -1. -2. -3. -4. -5. -6.]
 [-2.  1.  3.  2.  1.  1.  0. -1. -1. -2.]
 [-3.  0.  2.  5.  4.  3.  2.  1.  0. -1.]
 [-4. -1.  3.  4.  5.  6.  5.  4.  4.  3.]
 [-5. -1.  2.  4.  4.  8.  9.  8.  7.  6.]
 [-6. -2.  1.  3.  4.  7. 11. 10. 10.  9.]
 [-7. -3.  0.  2.  5.  6. 10. 13. 12. 11.]
 [-8. -4. -1.  1.  4.  8.  9. 12. 15. 14.]
 [-9. -5. -1.  0.  3.  7. 10. 11. 15. 18.]]

Aligned Sequences:
ACTCGGAGC
GCTAGGACC
----------------------------------------
```

```
dp table of 'CTGGAATCT' and 'GCTAGGACC' :
[[ 0. -1. -2. -3. -4. -5. -6. -7. -8. -9.]
 [-1.  2.  2.  1.  0. -1. -2. -3. -4. -5.]
 [-2.  1.  1.  4.  3.  2.  1.  0. -1. -2.]
 [-3.  1.  3.  3.  4.  6.  5.  4.  3.  2.]
 [-4.  0.  3.  4.  3.  7.  9.  8.  7.  6.]
 [-5. -1.  2.  4.  6.  6.  8. 11. 10.  9.]
 [-6. -2.  1.  3.  6.  6.  7. 10. 11. 10.]
 [-7. -3.  0.  3.  5.  7.  7.  9. 10. 10.]
 [-8. -4.  0.  2.  4.  7.  9.  8. 12. 13.]
 [-9. -5. -1.  2.  3.  6.  8. 10. 11. 12.]]

Aligned Sequences:
-CT-GGAATCT
GCTAGGAC-C-
----------------------------------------
```

حال امتیاز های هم ترازی دو به دو هر دو دنباله(pairwise scores) از مجموعه دنباله ها داریم که به شکل زیر است (توجه کنید چون در الگوریتم نیاز به شباهت هر دنباله به خودش نداریم قطر اصلی ماتریس صفر است.):

```
sequence 1 = 'ACCCTGAACC'
sequence 2 = 'ACTCGGAGC'
sequence 3 = 'CTGGAATCT'
sequence 4 = 'GCTAGGACC'
pairwise_scores of sequences from each other:
[[ 0. 17. 10. 15.]
 [17.  0. 14. 18.]
 [10. 14.  0. 12.]
 [15. 18. 12.  0.]]
----------------------------------------
```

حال برای پیدا کردن دنباله مرکزی (Center Sequence) با استفاده از فرمول
Center Sequence $= S_c = \arg\min_{S_i} \sum_j D(S_i, S_j)$ ، ابتدا دنباله ها را جمع میکنیم و سپس ماکزیمم میگیریم چرا که در اینجا
در ماتریس امتیازات similarity ها را داریم. (maximize similarity = minimize distance)

```
----------------------------------------
sum of scores:
[42. 49. 36. 45.]
center_index = 1, center_sequence = ACTCGGAGC
----------------------------------------
```

همان طور که می بینیم دنباله ACTCGGAGC به عنوان Center Sequence یا همان S_c در نظر گرفته می شود.
حال دنباله های دیگر (به غیر از Center Sequence) را یکی یکی با ACTCGGAGC تراز میکنیم و با اضافه کردن spaces(gaps)
خروجی نهایی یا همان M multiple alignment را میسازیم.

```
step 1:
dp table of 'ACTCGGAGC' and 'ACCCTGAACC' :
[[  0.  -1.  -2.  -3.  -4.  -5.  -6.  -7.  -8.  -9. -10.]
 [ -1.   2.   1.   0.  -1.  -2.  -3.  -4.  -5.  -6.  -7.]
 [ -2.   1.   5.   4.   3.   2.   1.   0.  -1.  -2.  -3.]
 [ -3.   0.   4.   4.   3.   5.   4.   3.   2.   1.   0.]
 [ -4.  -1.   3.   7.   7.   6.   7.   6.   5.   5.   4.]
 [ -5.  -2.   2.   6.   9.   8.   9.   8.   7.   7.   7.]
 [ -6.  -3.   1.   5.   8.  10.  11.  10.   9.   9.   9.]
 [ -7.  -4.   0.   4.   7.   9.  10.  13.  12.  11.  10.]
 [ -8.  -5.  -1.   3.   6.   8.  12.  12.  13.  14.  13.]
 [ -9.  -6.  -2.   2.   6.   7.  11.  12.  12.  16.  17.]]
aligned_center = ACTCGG-AGC, aligned_seq = ACCCTGAACC
----------------------------------------
```

```
step 2:
dp table of 'ACTCGG-AGC' and 'CTGGAATCT' :
[[ 0. -1. -2. -3. -4. -5. -6. -7. -8. -9.]
 [-1.  0.  0. -1. -2. -2. -3. -4. -5. -6.]
 [-2.  2.  1.  2.  1.  0. -1. -2. -1. -2.]
 [-3.  1.  4.  3.  3.  2.  1.  1.  0.  1.]
 [-4.  0.  3.  6.  5.  4.  3.  2.  4.  3.]
 [-5. -1.  2.  6.  9.  8.  7.  6.  5.  5.]
 [-6. -2.  1.  5.  9.  9.  8.  8.  8.  7.]
 [-6. -2.  1.  5.  9.  9.  8.  8.  8.  7.]
 [-7. -3.  0.  4.  8. 11. 11. 10.  9.  9.]
 [-8. -4. -1.  3.  7. 10. 11. 12. 12. 11.]
 [-9. -5. -2.  2.  6.  9. 10. 11. 15. 14.]]
aligned_center = ACTCGG-AGC-, aligned_seq = -CTGGA-ATCT
----------------------------------------
```

```
step 3:
dp table of 'ACTCGG-AGC-' and 'GCTAGGACC' :
[[ 0. -1. -2. -3. -4. -5. -6. -7. -8. -9.]
 [-1.  0. -1. -1. -1. -2. -3. -4. -5. -6.]
 [-2.  1.  3.  2.  1.  1.  0. -1. -1. -2.]
 [-3.  0.  2.  5.  4.  3.  2.  1.  0. -1.]
 [-4. -1.  3.  4.  5.  6.  5.  4.  4.  3.]
 [-5. -1.  2.  4.  4.  8.  9.  8.  7.  6.]
 [-6. -2.  1.  3.  4.  7. 11. 10. 10.  9.]
 [-6. -2.  1.  3.  4.  7. 11. 10. 10.  9.]
 [-7. -3.  0.  2.  5.  6. 10. 13. 12. 11.]
 [-8. -4. -1.  1.  4.  8.  9. 12. 15. 14.]
 [-9. -5. -1.  0.  3.  7. 10. 11. 15. 18.]
 [-9. -5. -1.  0.  3.  7. 10. 11. 15. 18.]]
aligned_center = ACTCGG-AGC-, aligned_seq = GCTAGG-ACC-
----------------------------------------
----------------------------------------
Multiple Alignment = ACTCGG-AGC-
----------------------------------------
```

بنابراین خروجی نهایی به شکل زیر می شود:

$$
\begin{aligned}
Multiple\ Alignment &= A\ C\ T\ C\ G\ G\ -\ A\ G\ C\ - \\
Aligned\ Sequence\ 1 &= A\ C\ C\ C\ T\ G\ A\ A\ C\ C\ - \\
Aligned\ Sequence\ 2 &= A\ C\ T\ C\ G\ G\ -\ A\ G\ C\ - \\
Aligned\ Sequence\ 3 &= -\ C\ T\ G\ G\ A\ -\ A\ T\ C\ T \\
Aligned\ Sequence\ 4 &= G\ C\ T\ A\ G\ G\ -\ A\ C\ C\ -
\end{aligned}
$$

همچنین در ادامه کد اجرای این الگوریتم آمده است:

```python
import numpy as np
scoreboard = {
    "-": {"-": 0, "A": -1, "T": -1, "C": -1, "G": -1},
    "A": {"-": -1, "A": 2, "T": 1, "C": 0, "G": 0},
    "T": {"-": -1, "A": 1, "T": 2, "C": -1, "G": 1},
    "C": {"-": -1, "A": 0, "T": -1, "C": 3, "G": 2},
    "G": {"-": -1, "A": 0, "T": 1, "C": 2, "G": 3},
}
sequences = [
    "ACCCTGAACC",
    "ACTCGGAGC",
    "CTGGAATCT",
    "GCTAGGACC",
]
def compute_pairwise_score(seq1, seq2):
    len1, len2 = len(seq1), len(seq2)
    dp = np.zeros((len1 + 1, len2 + 1))
    backtrack = np.zeros((len1 + 1, len2 + 1), dtype=str)
    for i in range(len1 + 1):
        dp[i][0] = i * -1
        backtrack[i][0] = "U"  # Up (gap in target_seq)
```

```
    for j in range(len2 + 1):
        dp[0][j] = j * -1
        backtrack[0][j] = "L"  # Left (gap in center_seq)
    for i in range(1, len1 + 1):
        for j in range(1, len2 + 1):
            match = dp[i - 1][j - 1] + scoreboard[seq1[i - 1]][seq2[j - 1]]
            delete = dp[i - 1][j] + scoreboard[seq1[i - 1]]["-"]
            insert = dp[i][j - 1] + scoreboard["-"][seq2[j - 1]]
            dp[i][j] = max(match, delete, insert)
            if dp[i][j] == match:
                backtrack[i][j] = "D"  # Diagonal
            elif dp[i][j] == delete:
                backtrack[i][j] = "U"  # Up
            else:
                backtrack[i][j] = "L"  # Left
    aligned_seq1 = []
    aligned_seq2 = []
    i, j = len1, len2
    while i > 0 or j > 0:
        if backtrack[i][j] == "D":
            aligned_seq1.append(seq1[i - 1])
            aligned_seq2.append(seq2[j - 1])
            i -= 1
            j -= 1
        elif backtrack[i][j] == "U":
            aligned_seq1.append(seq1[i - 1])
            aligned_seq2.append("-")
            i -= 1
        elif backtrack[i][j] == "L":
            aligned_seq1.append("-")
            aligned_seq2.append(seq2[j - 1])
            j -= 1
    print("----------------------------------------")
    print(f"dp table of '{seq1}' and '{seq2}' :")
    print(dp)
    print("\nAligned Sequences:")
    print("".join(reversed(aligned_seq1)))
    print("".join(reversed(aligned_seq2)))
    print("----------------------------------------")
    return dp[-1][-1]
def find_center_sequence(sequences):
    n = len(sequences)
    pairwise_scores = np.zeros((n, n))
    # Compute pairwise alignment scores
    for i in range(n):
```

```
        for j in range(i + 1, n):
            pairwise_scores[i][j] = compute_pairwise_score(sequences[i],
sequences[j])
            pairwise_scores[j][i] = pairwise_scores[i][j]
    ####
    print("-----------------------------------------")
    i = 1
    for seq in sequences:
        print(f"sequence {i} = '{seq}'")
        i += 1
    print("pairwise_scores of sequences from each other:")
    print(pairwise_scores)
    print("-----------------------------------------")
    sum_scores = pairwise_scores.sum(axis=1)
    center_index = np.argmax(sum_scores) # similarity -> arg max
    print("-----------------------------------------")
    print("sum of scores:")
    print(sum_scores)
    print(f"center_index = {center_index}, center_sequence =
{sequences[center_index]}")
    print("-----------------------------------------")
    return center_index, pairwise_scores
def align_to_center(center_seq, target_seq):
    len1, len2 = len(center_seq), len(target_seq)
    dp = np.zeros((len1 + 1, len2 + 1))
    backtrack = np.zeros((len1 + 1, len2 + 1), dtype=str)
    for i in range(len1 + 1):
        dp[i][0] = i * -1
        backtrack[i][0] = "U"  # Up (gap in target_seq)
    for j in range(len2 + 1):
        dp[0][j] = j * -1
        backtrack[0][j] = "L"  # Left (gap in center_seq)
    for i in range(1, len1 + 1):
        for j in range(1, len2 + 1):
            match = dp[i - 1][j - 1] + scoreboard[center_seq[i - 1]][target_seq[j
- 1]]
            delete = dp[i - 1][j] + scoreboard[center_seq[i - 1]]["-"]
            insert = dp[i][j - 1] + scoreboard["-"][target_seq[j - 1]]
            dp[i][j] = max(match, delete, insert)
            if dp[i][j] == match:
                backtrack[i][j] = "D"  # Diagonal
            elif dp[i][j] == delete:
                backtrack[i][j] = "U"  # Up
            else:
                backtrack[i][j] = "L"  # Left
```

```
    aligned_center = []
    aligned_target = []
    i, j = len1, len2
    while i > 0 or j > 0:
        if backtrack[i][j] == "D":
            aligned_center.append(center_seq[i - 1])
            aligned_target.append(target_seq[j - 1])
            i -= 1
            j -= 1
        elif backtrack[i][j] == "U":
            aligned_center.append(center_seq[i - 1])
            aligned_target.append("-")
            i -= 1
        elif backtrack[i][j] == "L":
            aligned_center.append("-")
            aligned_target.append(target_seq[j - 1])
            j -= 1
    print(f"dp table of '{center_seq}' and '{target_seq}' :")
    print(dp)
    return "".join(reversed(aligned_center)), "".join(reversed(aligned_target))
def multiple_sequence_alignment(sequences):
    center_index, _ = find_center_sequence(sequences)
    center_seq = sequences[center_index]
    aligned_sequences = [center_seq]
    step = 1
    for i, seq in enumerate(sequences):
        if i != center_index:
            print(f"step {step}:")
            step += 1
            aligned_center, aligned_seq = align_to_center(center_seq, seq)
            aligned_sequences.append(aligned_seq)
            center_seq = aligned_center  # Update center sequence with gaps
            print(f"aligned_center = {aligned_center}, aligned_seq =
{aligned_seq}")
            print("----------------------------------------")
    print("----------------------------------------")
    print(f"Multiple Alignment = {center_seq}")
    print("----------------------------------------")
    return aligned_sequences, center_seq
aligned_sequences, aligned_center_seq = multiple_sequence_alignment(sequences)
print("Aligned Sequences iteratively:")
for seq in aligned_sequences:
    print(seq)
```

سوال چهارم

الف) روش‌های هم ترازی ساختاری بر مقایسه ساختارهای سه بعدی ماکرومولکول‌ها، مانند پروتئین‌ها یا RNA، برای تعیین شباهت‌های آنها تمرکز دارند. در اینجا مراحل کلیدی آمده است:

۱. ورودی و نمایش (Input and Representation):
   - داده‌های ساختاری، معمولاً از روش‌های تجربی مانند کریستالوگرافی اشعه ایکس یا طیف‌سنجی NMR به دست می‌آیند (معمولاً از پایگاه‌های داده مانند بانک داده‌های پروتئین، PDB).
   - ساختارها بر حسب مختصات اتمی یا اتم های backbone نشان داده می شوند (به عنوان مثال، اتم های Cα برای پروتئین ها).
۲. راه اندازی هم ترازی (Alignment Initialization):
   - ترازهای اولیه با استفاده از روش های heuristic، شباهت توالی یا تطبیق ساختار ثانویه ایجاد می شوند. که به عنوان نقطه شروعی برای تطبیق ساختاری عمل می کند.
۳. انطباق (Superimposition):
   - ساختارهای سه‌بعدی با به حداقل رساندن انحراف ریشه میانگین مربع (RMSD: root-mean-square deviation) بین اتم‌های معادل در دو ساختار روی هم قرار می‌گیرند.
   - الگوریتم ها به طور مکرر ترازها را برای دستیابی به همپوشانی ساختاری بهینه تنظیم می کنند.
۴. امتیازدهی (Scoring):

   یک تابع امتیازدهی کیفیت هم ترازی را ارزیابی می کند. معیارهای رایج عبارتند از:
   - RMSD (اندازه گیری انحراف بین اتم ها بر اساس فاصله).
   - تعداد باقی مانده های تراز شده(aligned residues).
   - Secondary structure agreement.
۵. بهینه سازی:
   - Iterative refinement، هم ترازی را با تنظیم تطابق residue، جایگیری شکاف، و جهت گیری برای به حداکثر رساندن امتیاز هم ترازی، بهبود می بخشد.
۶. خروجی و تفسیر(Output and Interpretation):
   - ساختارهای تراز شده در مدل های سه بعدی visualized می شوند.
   - خروجی ها شامل aligned residue pairs، similarity scores و نقوش ساختاری(structural motifs) است.

ب) چند نمونه الگوریتم هم ترازی ساختاری

۱. DALI (Distance Alignment Tool):

از ماتریس های فاصله برای شناسایی شباهت های ساختاری استفاده می کند. فاصله بین باقیمانده(inter-residue distances) که محاسبه شده از اتم های Cα است را بین دو ساختار را مقایسه می کند.

ویژگی های کلیدی:

- بر حفظ ویژگی های ساختاری global و local تمرکز دارد.
- موثر در تشخیص fold های حفاظت شده حتی زمانی که شباهت توالی کم است.

کاربرد: اغلب برای شناسایی روابط تکاملی و طبقه بندی پروتئین ها به خانواده ها استفاده می شود.

۲. TM-align (Template Modeling Alignment):

از یک الگوریتم heuristic برای به حداکثر رساندن امتیاز TM استفاده می کند، معیاری که برای ارزیابی شباهت ساختاری مستقل از اندازه پروتئین طراحی شده است.

ویژگی های کلیدی:

- مقاوم در برابر تغییرات طول پروتئین.
- ترازهایی را تولید می کند که برای کیفیت ساختاری بهینه شده اند و نه صرفاً مطابقت توالی(sequence correspondence).

کاربردها: به طور گسترده برای تشخیص fold و ارزیابی مدل در پیش بینی ساختار پروتئین استفاده می شود.

۳. الگوریتم FATCAT (Flexible Alignment by Chaining Transformed fragments):

توسعه یافته برای رسیدگی به انعطاف ساختاری پروتئین.

ویژگی های کلیدی:

- ترازهای ساختاری جزئی(partial) را امکان پذیر می کند.
- تغییرات ساختاری محلی را کنترل می کند.
- اجازه تبدیل انعطاف پذیر بین ساختارهای مقایسه شده را می دهد.
- یک رویکرد sliding window را برای تطبیق قطعه(fragment matching) معرفی می کند.

ج)

تفاوت بین هم ترازی مبتنی بر توالی و هم تراز ساختاری

Sequence Alignment:

- بر اساس توالی خطی اسیدهای آمینه است.
- به طور مستقیم با individual residues مطابقت را انجام می دهد.
- از ماتریس های امتیازدهی مانند BLOSUM استفاده می کند.
- در درجه اول شباهت های توالی خطی را شناسایی می کند.
- نسبت به روابط ساختاری سه بعدی حساسیت کمتری دارد.
- سریعتر و از نظر محاسباتی شدت کمتری دارد.

Structural Alignment:

- بر arrangement های فضایی سه بعدی تمرکز دارد.
- عناصر ساختاری و روابط فضایی آنها را مطابقت می دهد.
- مختصات اتمی و تبدیلات هندسی را در نظر می گیرد.
- روابط عملکردی و تکاملی فراتر از sequence را ثبت می کند.
- از نظر محاسباتی پیچیده تر است.
- می تواند روابط فضایی سه بعدی را که در داده های دنباله ای قابل مشاهده نیستند، آشکار کند.

همچنین تفاوت بین هم ترازی مبتنی بر توالی و هم تراز ساختاری به طور خلاصه در جدول زیر آمده است. (چون ترجمه اصطلاحات به فارسی درک مطالب را سخت تر میکند جدول را به انگلیسی نگارش میکنیم)

Aspect	**Sequence Alignment**	**Structural Alignment**
Input Data	Linear sequences of nucleotides or amino acids	3D coordinates of macromolecular structures
Basis of Comparison	Sequence similarity	Spatial arrangement of residues
Tools Used	BLAST, Clustal Omega, MUSCLE	DALI, TM-align, PyMOL
Applications	Identifying sequence homologs	Understanding structural fold conservation
Limitations	Fails with low sequence similarity	Limited to cases where structural data exists

د) تکمیل MSA مبتنی بر روش های هم ترازی ساختاری

۱. چگونه روش های تراز ساختاری MSA را تکمیل می کند:
- بینش عملکردی(Functional Insights): ساختارهای سه بعدی حفاظت شده(Conserved 3D structures) اغلب حاکی از عملکردهای مشابه هستند حتی اگر توالی ها به طور قابل توجهی متفاوت باشند. که همچنین به شناسایی مناطق بحرانی عملکردی(functionally critical regions) کمک می کند.
- روابط تکاملی(Evolutionary Relationships): motif های ساختاری(invisible in sequence data) ممکن است در پروتئین های distantly related حفظ شوند و بینش های تکاملی عمیق تری ارائه دهند. که می تواند درک روابط خانوادگی پروتئین را بهبود ببخشد.
- قرار دادن بهتر شکاف(Better Gap Placement): داده های ساختاری به جایگیری منطقی شکاف کمک می کند و یکپارچگی عملکردی و ساختاری را در ترازها حفظ می کند.

۲. چالش ها و محدودیت ها:
- در دسترس بودن داده ها: داده های ساختاری با وضوح بالا تنها برای بخش کوچکی از پروتئین های شناخته شده در دسترس است.
- پیچیدگی محاسباتی(Computational Complexity): هم ترازی ساختاری از نظر محاسباتی گران تر از هم ترازی مبتنی بر توالی است.
- مشکلات یکپارچه سازی(Integration Issues): ترکیب توالی و داده های ساختاری در یک چارچوب واحد می تواند از نظر فنی چالش برانگیز باشد.
- ساختارهای پویا(Dynamic Structures): پروتئین ها انعطاف پذیر هستند. ساختارهای ایستا ممکن است تمام ترکیبات بیولوژیکی مرتبط را نتوانند در بر بگیرند.
- حساسیت هم تراز(Alignment Sensitivity): الگوریتم های مختلف می توانند نتایج alignment متفاوتی را ایجاد کنند. بنابراین هیچ "استاندارد طلایی" universal برای هم ترازی ساختاری وجود ندارد. که نیاز به اعتبارسنجی دقیق و cross-referencing را پر رنگ میکند.

با ادغام تراز ساختاری با traditional sequence-based MSA، محققان می توانند به درک جامع تری از خانواده های پروتئین دست یابند، اما پرداختن به این چالش ها برای اجرای موثر ضروری است.

مثال برای درک عمیق تر:
پروتئین ها را به عنوان پازل های سه بعدی پیچیده تصور کنید. هم ترازی توالی مانند مقایسه قطعات پازل بر اساس شکل لبه آنها است، در حالی که هم ترازی ساختاری مانند درک نحوه قرار گرفتن آن قطعات در فضای سه بعدی با یکدیگر است. برخی از پازل ها

ممکن است لبه های مشابهی داشته باشند اما ساختار کلی کاملاً متفاوتی داشته باشند، دقیقاً به همین دلیل است که هم ترازی ساختاری چنین بینش غنی را ارائه می دهد.

بیوانفورماتیک ساختاری با تکنیک های نوظهور یادگیری ماشینی و هوش مصنوعی که روش‌های تراز پیچیده‌تر و دقیق‌تری را در آینده نوید می‌دهند، به تکامل خود ادامه می‌دهد.

سوال پنجم

الف) الگوریتم های Maximum Likelihood و Bayesian Inference چگونه کار می کنند؟

Maximum Likelihood (ML):

ML درختی را جستجو می کند که احتمال مشاهده داده های داده شده (مثلاً sequences) را تحت یک مدل تکاملی مشخص به حداکثر می رساند.

$$Likelihood = P(\text{Data} \mid \text{Tree}, \text{Model})$$

مراحل کلیدی:

۱. Model Selection: یک مدل تکاملی را تعریف کنیم (به عنوان مثال، Jukes-Cantor، GTR و همچنین substitution rates ، transition probabilities).
۲. Tree Evaluation: درخت های مختلف را تولید کنیم و احتمال داده ها را برای هر توپولوژی درختی ممکن محاسبه کنیم.
۳. Optimization: درختی را که بیشترین امتیاز احتمال را دارد انتخاب کنیم. از بهینه سازی عددی برای به حداکثر رساندن تابع احتمال استفاده میکنیم.
۴. Output: محتمل ترین درخت، همراه با طول شاخه های مرتبط و امتیازات احتمال.

ویژگی ها:

- بهترین درخت را با طول شاخه‌های بهینه‌شده تحت مدل داده شده ارائه می‌کند.
- محاسبات زیادی دارد زیرا بسیاری از توپولوژی های درختی را ارزیابی می کند.

Bayesian Inference (BI):

BI احتمال posterior یک درخت را با توجه به داده ها و Bayes theorem محاسبه می کند و دانش قبلی(prior knowledge) را با احتمال داده ها تحت یک مدل مشخص ترکیب می کند.

$$P(\text{Tree}|\text{Data}) = \frac{P(\text{Data}|\text{Tree}) \cdot P(\text{Tree})}{P(\text{Data})}$$

مراحل کلیدی:

۱. Prior Specification: یک توزیع قبلی برای توپولوژی های درختی، طول شاخه ها و پارامترهای مدل مشخص کنیم.
۲. Model Evaluation: احتمال داده های یک درخت را محاسبه کنیم و آن را با داده های قبلی ترکیب کنیم.
۳. Sampling: از زنجیره مارکوف مونت کارلو (MCMC: Markov Chain Monte Carlo) برای نمونه برداری از درختان از توزیع posterior استفاده کنیم.
۴. احتمالات posterior درختان نمونه برداری شده را خلاصه کنیم.
۵. Output: توزیع درختان، با احتمالات posterior برای هر توپولوژی درخت.

ویژگی ها:

- توزیع درختان احتمالی را به جای یک درخت واحد تولید می کند.
- دانش قبلی و عدم قطعیت را در تجزیه و تحلیل ادغام می کند.

ب) معمولا زمانی از Maximum Likelihood/Bayesian Inference استفاده میکنیم که داده ها و منابع محاسباتی باکیفیت داریم، به خصوص اگر هدف استنتاج مبتنی بر مدل از روابط تکاملی باشد. همچنین زمانی از Neighbor-Joining استفاده میکنیم که نیاز به تقریب های سریع داریم یا زمانی که منابع محاسباتی محدود هستند، اگرچه ممکن است دقت آن با مجموعه داده های پیچیده یا noisy کاهش یابد.

مقایسه Maximum Likelihood و Bayesian Inference با الگوریتم Neighbor-Joining در جدول زیر به طور خلاصه آمده است. (چون ترجمه اصطلاحات به فارسی درک مطالب را سخت تر میکند جدول را به انگلیسی نگارش میکنیم)

جنبه (Aspect)	**Maximum Likelihood (ML)**	**Bayesian Inference (BI)**	**Neighbor-Joining (NJ)**
Type of Method	Model-based	Model-based	Distance-based
Accuracy(دقت)	Generally more accurate for complex data	High accuracy with incorporation of priors	Relatively less accurate, especially for large datasets
Input (ورودی)	Requires sequence data and a model	Requires sequence data and a model	Uses a distance matrix
Output(خروجی)	Single best tree	Distribution of trees	Single tree
Computational Cost	High	Very high	Low
Scalability	Poor for large datasets	Poor for large datasets	Good
Use Case(استفاده)	Best for small-to-moderate datasets	Best for detailed probabilistic studies	Quick approximation with large datasets

زمان استفاده از هر الگوریتم:

- (NJ) Neighbor-joining: چون سریع است، زمانی که منابع محاسباتی محدود هستند یا برای مجموعه داده های بزرگ کاربرد دارد.
- ML: زمانی که دقت مهمتر از سرعت و منابع موجود است.
- BI: برای بینش احتمالی جامع و زمانی که اطلاعات قبلی (prior information) در دسترس است.

ج) مقایسه استنتاج بیزی(BI) و حداکثر کردن احتمال(ML) در جدول زیر به طور خلاصه آمده است. (چون ترجمه اصطلاحات به فارسی درک مطالب را سخت تر میکند جدول را به انگلیسی نگارش میکنیم)

Aspect	Maximum Likelihood (ML)	Bayesian Inference (BI)
Tree Output	Single best tree base on likelihood	Posterior distribution of trees
Incorporation of Priors	Does not use priors	Integrates prior knowledge
Complexity	High	Very high
Uncertainty Representation	Limited	Explicit through posterior probabilities
Computational Time	Faster than BI	Slower due to MCMC sampling
Computational Demand	Lower	Higher
Interpretation	Straightforward	More nuanced with probabilistic distributions
Confidence Assessment	Bootstrapping required	Naturally provides posterior probabilities

برتری:

مزایای ML:

- تفسیر و پیاده سازی ساده تر بدون در نظر گرفتن prior assumptions.
- سریعتر از BI و نیاز به محاسبات کمتر، که آن را برای بسیاری از موارد کاربردی عملی می کند.
- زمانی که هیچ دانش قبلی در دسترس نباشد به خوبی کار می کند.
- برای مجموعه داده های ساده تر یا زمانی که منابع محدود هستند ترجیح داده می شود.

- مزایای BI:

- عدم قطعیت را در نظر می گیرد و تفسیری غنی تر و احتمالی ارائه می دهد.
- دانش قبلی را در بر می گیرد و امکان انعطاف پذیری در مفروضات مدل سازی را فراهم می کند.
- یک توزیع احتمالی را خروجی می دهد و درک دقیقی از عدم قطعیت را امکان پذیر می کند.
- مناسب برای مجموعه داده های پیچیده که برآوردهای اطمینان(confidence estimates) بسیار مهم هستند.

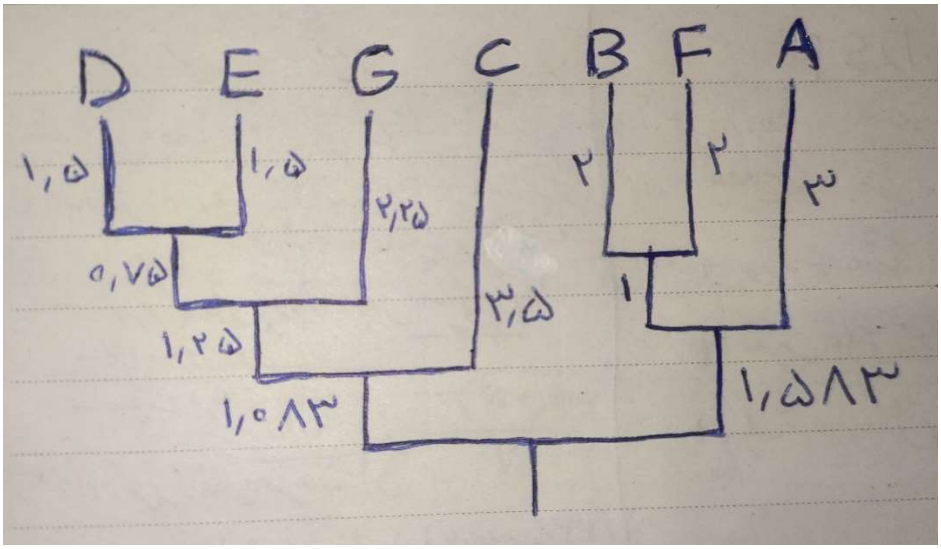
به طور خلاصه، ML برای تجزیه و تحلیل فیلوژنتیکی سریع و دقیق زمانی که اطلاعات قبلی(prior information) در دسترس نیست ترجیح داده می شود، در حالی که BI در موقعیت هایی که بینش های احتمالی یا پیشین ها(priors) مهم هستند برتری دارد.

سوال ششم

الگوریتم خوشه بندی UPGMA مراحل زیر را برای ماتریس فاصله داده شده تولید می کند:

۱. D و E را ادغام میکنیم تا خوشه (D,E) با ارتفاع ۱/۵ تشکیل شود.
۲. B و F را ادغام میکنیم تا خوشه (B,F) با ارتفاع ۲/۰ تشکیل شود.
۳. G را با (D,E) ادغام میکنیم تا خوشه ((G,(D,E)) با ارتفاع ۲/۲۵ تشکیل شود.
۴. A را با (B,F) ادغام میکنیم تا خوشه ((A,(B,F)) با ارتفاع ۳/۰ تشکیل شود.
۵. C را با ((G,(D,E)) ادغام میکنیم تا خوشه ((C,(G,(D,E))) با ارتفاع ۳/۵ تشکیل شود.
۶. ((A,(B,F)) را با ((C,(G,(D,E))) ادغام میکنیم تا خوشه نهایی (((A,(B,F)),(C,(G,(D,E)))) با ارتفاع ۴/۵۸۳ تشکیل شود.

این منجر به ساختار سلسله مراتبی زیر میشود که منعکس کننده روابط تکاملی با توجه به ماتریس فاصله داده شده است.



همچنین در ادامه ماتریس فاصله در هر مرحله، گام به گام آمده است.

```
    A   B   C   D  E   F   G
A   0   5   9   9  8   7  12
B   5   0  10  10  7   4  11
C   9  10   0   8  7  10   6
D   9  10   8   0  3   9   5
E   8   7   7   3  0   6   4
F   7   4  10   9  6   0   9
G  12  11   6   5  4   9   0
---------------------------------
         A     B     C     F     G  (D,E)
A      0.0   5.0   9.0   7.0  12.0   8.5
B      5.0   0.0  10.0   4.0  11.0   8.5
C      9.0  10.0   0.0  10.0   6.0   7.5
F      7.0   4.0  10.0   0.0   9.0   7.5
G     12.0  11.0   6.0   9.0   0.0   4.5
(D,E)  8.5   8.5   7.5   7.5   4.5   0.0
---------------------------------
         A     C     G  (D,E)  (B,F)
A      0.0   9.0  12.0   8.5   6.0
C      9.0   0.0   6.0   7.5  10.0
G     12.0   6.0   0.0   4.5  10.0
(D,E)  8.5   7.5   4.5   0.0   8.0
(B,F)  6.0  10.0  10.0   8.0   0.0
---------------------------------
```

```
                  A     C      (B,F)  (G,(D,E))
A          0.000000   9.0   6.000000   9.666667
C          9.000000   0.0  10.000000   7.000000
(B,F)      6.000000  10.0   0.000000   8.666667
(G,(D,E))  9.666667   7.0   8.666667   0.000000
-----------------------------------
                  C  (G,(D,E))  (A,(B,F))
C          0.000000       7.0   9.666667
(G,(D,E))  7.000000       0.0   9.000000
(A,(B,F))  9.666667       9.0   0.000000
-----------------------------------
              (A,(B,F))  (C,(G,(D,E)))
(A,(B,F))      0.000000       9.166667
(C,(G,(D,E)))  9.166667       0.000000
-----------------------------------
                              ((A,(B,F)),(C,(G,(D,E))))
((A,(B,F)),(C,(G,(D,E))))                         0.0
-----------------------------------
steps:
step 1: ('(D,E)', 1.5)
step 2: ('(B,F)', 2.0)
step 3: ('(G,(D,E))', 2.25)
step 4: ('(A,(B,F))', 3.0)
step 5: ('(C,(G,(D,E)))', 3.5)
step 6: ('((A,(B,F)),(C,(G,(D,E))))', 4.583333333333333)
```

این الگوریتم به طور خلاصه به شیوه زیر کار می کند:

۱. مقداردهی اولیه: هر دنباله به عنوان خوشه خودش شروع می شود.
۲. یافتن حداقل فاصله: جفت خوشه هایی را که کمترین فاصله را در ماتریس دارند شناسایی میشوند.
۳. ادغام خوشه ها: نزدیکترین خوشه ها را در یک خوشه جدید ادغام میکنیم و با استفاده از میانگین موزون حسابی، فاصله بین خوشه جدید و سایر خوشه ها را محاسبه میکنیم.
۴. به روز رسانی ماتریس فاصله: خوشه های ادغام شده را حذف کرده و خوشه جدید را به ماتریس فاصله اضافه می کنیم.
۵. تکرار: تا زمانی که همه خوشه ها در یک درخت ادغام شوند ادامه می دهیم.

همچنین در ادامه کد پیاده سازی شده برای این الگوریتم آمده است:

```python
import numpy as np
import pandas as pd

# Define the distance matrix again
labels = ["A", "B", "C", "D", "E", "F", "G"]
matrix = np.array([
    [0, 5, 9, 9, 8, 7, 12],  # A
    [5, 0, 10, 10, 7, 4, 11], # B
    [9, 10, 0, 8, 7, 10, 6],  # C
    [9, 10, 8, 0, 3, 9, 5],   # D
    [8, 7, 7, 3, 0, 6, 4],    # E
    [7, 4, 10, 9, 6, 0, 9],   # F
```

```
    [12, 11, 6, 5, 4, 9, 0]   # G
])

# Create a DataFrame for easier handling
dist_df = pd.DataFrame(matrix, index=labels, columns=labels)

def upgma_weighted(distance_matrix):
    """
    Perform UPGMA clustering using weighted average.

    Parameters:
    - distance_matrix: DataFrame representing the distance matrix.

    Returns:
    - clustering_steps: List of tuples showing clustering steps.
    """
    clusters = {label: [label] for label in distance_matrix.index}
    cluster_sizes = {label: 1 for label in distance_matrix.index}
    distances = distance_matrix.copy()
    steps = []

    while len(clusters) > 1:
        # Find the closest pair of clusters
        min_dist = np.inf
        closest_pair = None
        for i in distances.index:
            for j in distances.columns:
                if i != j and distances.at[i, j] < min_dist:
                    min_dist = distances.at[i, j]
                    closest_pair = (i, j)

        # Merge the closest clusters
        c1, c2 = closest_pair
        new_cluster = f"({c1},{c2})"
        new_size = cluster_sizes[c1] + cluster_sizes[c2]
        clusters[new_cluster] = clusters.pop(c1) + clusters.pop(c2)
        cluster_sizes[new_cluster] = new_size
        steps.append((new_cluster, min_dist / 2))

        # Update the distance matrix using weighted average
        for cluster in distances.index:
            if cluster not in closest_pair:
                size_c1 = cluster_sizes[c1]
                size_c2 = cluster_sizes[c2]
                weighted_distance = (
```

```
                    size_c1 * distances.at[c1, cluster] +
                    size_c2 * distances.at[c2, cluster]
                ) / new_size
                distances.at[new_cluster, cluster] = weighted_distance
                distances.at[cluster, new_cluster] = weighted_distance

        distances = distances.drop(index=[c1, c2], columns=[c1, c2])
        distances.at[new_cluster, new_cluster] = 0

        print(distances)
        print("-----------------------------------")

    return steps

print(dist_df)
print("-----------------------------------")
# Perform UPGMA with weighted average
upgma_weighted_steps = upgma_weighted(dist_df)
print("steps:")
i = 1
for step in upgma_weighted_steps:
    print(f"step {i}:", step)
    i += 1
```

پایان