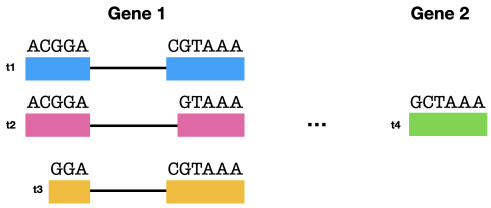
Pseudoalignment

Consider the following transcriptome:

```
>t1_g1
ACGGACGTAAA
>t2_g1
ACGGAGTAA
>t3_g1
GGACGTAAA
>t4_g2
GCTAAA
```

Here is a visual representation of the transcriptome:



(a) Draw the colored de Bruijn graph that one would use for pseudoalignment with $k = 3$. In this particular case, the nodes have 3-mers, not the edges (this is the setup we used in class). Refer to the slides on construction.

(b) Pseudoalign $GGACGT$ and show the relevant steps you might take (including skips). Refer to the slides for the pseudoalignment.

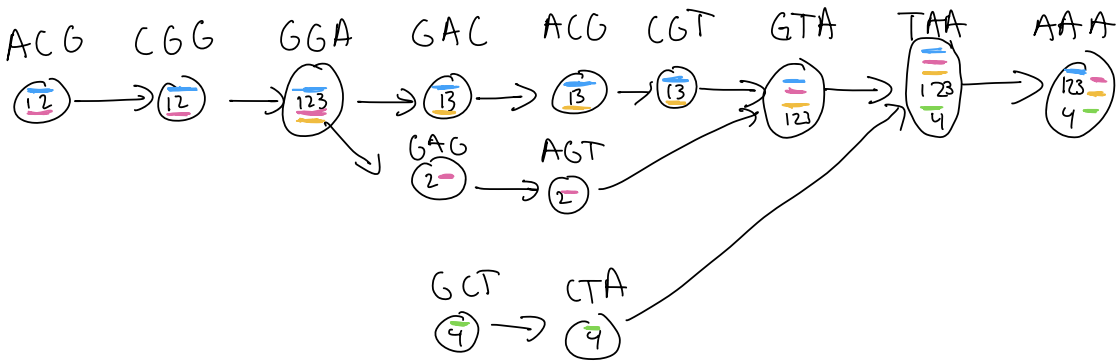
(c) Pseudoalign $GGATGT$ and show the relevant steps you might take (including skips). Remember, every k-mer has an equivalence class which is a set. If a k-mer is in your data but not in your transcriptome, its equivalence class is the null set.

(d) The previous read might arise if there is an error at position 4 (using 1-based indexing). Describe an algorithm to deal with the error. Describe the benefits and drawbacks of your algorithm. These properties might come in speed, loss of data, or false alignments (or other things). Note: there isn't one correct answer.

(e) The following sequence is a valid RNA-seq read $TTTACG$. Clearly it won't give you a non-empty equivalence class as-is. How did this data arise and how might you pseudoalign it? Hint: think about how RNA-seq is generated. That is, the orientation of the reads matters and can generate some annoying things we have to keep track of.

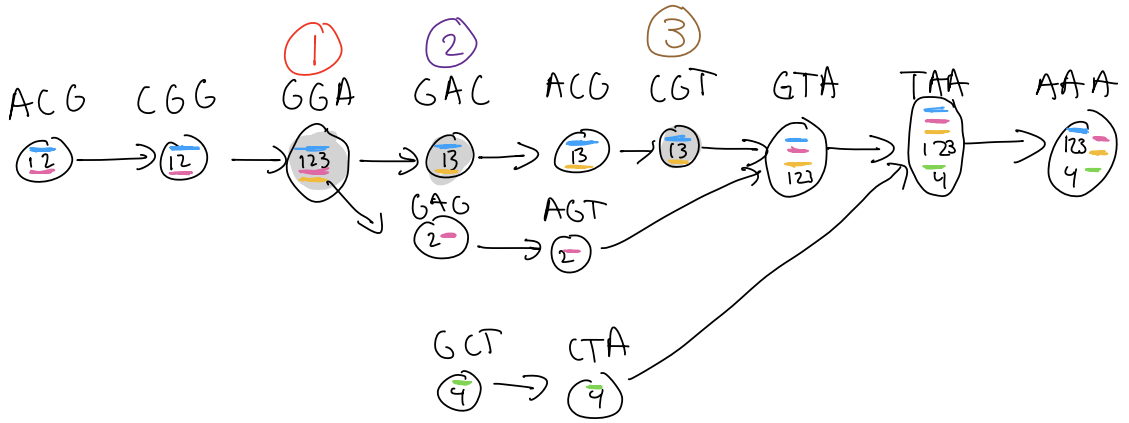
a)

```
>t1_g1
ACGGACGTAAA
>t2_g1
ACGGAGTAAA
>t3_g1
GGACGTAAA
>t4_g2
GCTAAA
```



b) We will color the nodes gray & # them as we use them.

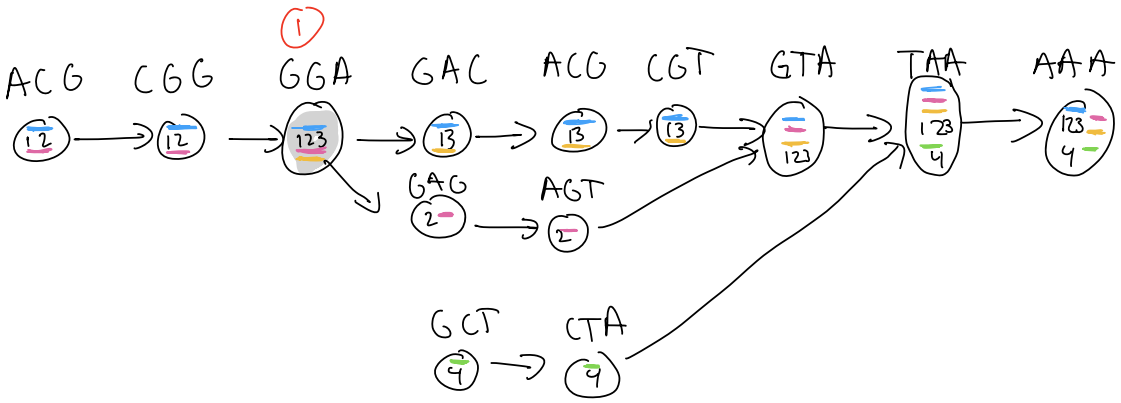
Query : GGACGT



The first k-mer is GGA. This corresponds to (1). Next, there is an immediate branchpoint. At this branchpoint, there are two possible outcomes: GAC, GAG. The read contains GAC, so we visit (2). The next branchpoint is 3 nodes away, which is beyond our read length, so we simply check the final k-mer, which is CGT. We intersect everything & get:

$$123 \cap 13 \cap 13 = 13$$ ✓

c) Query: GGATGT



Like in (b), we start with GGA. Next, the graph has GAC & GAG, however, our next k-mer in the data is GAT. This is a clear mismatch. In fact, the remaining kmers are not in our graph, so:

null set

$123 \cap \{\} \cap \{\} \cap \{\} = \{\}$. ✓

This is the most straightforward solution & is acceptable

An alternative solution would be to backtrack when you see a null equivalence class. In this case, your solution would be (123).

Finally, if the read were longer, you could skip over the error. (See discussion 6).

d) In (c) I described two possibilities.

If I "backtrack", then one drawback is I can potentially have a longer, less informative equivalence class. Assuming the error rate is low, this is likely preferable to simply discarding the read by calling it null. This approach is likely relatively fast because you short circuit the remainder of the comparisons.

If I "skip" over the potential error, I could actually recover a smaller equivalence class than in the backtracking case. However, this assumes I see only one error. We know from real life that errors are correlated, and in fact this might help sometimes but often won't make a considerable difference in real data. This approach is obviously slower, & has potential to get messy if there are several errors.

(e) The following sequence is a valid RNA-seq read *TTTACG*. Clearly it won't give you a non-empty equivalence class as-is. How did this data arise and how might you pseudoalign it? Hint: think about how RNA-seq is generated. That is, the orientation of the reads matters and can generate some annoying things we have to keep track of.

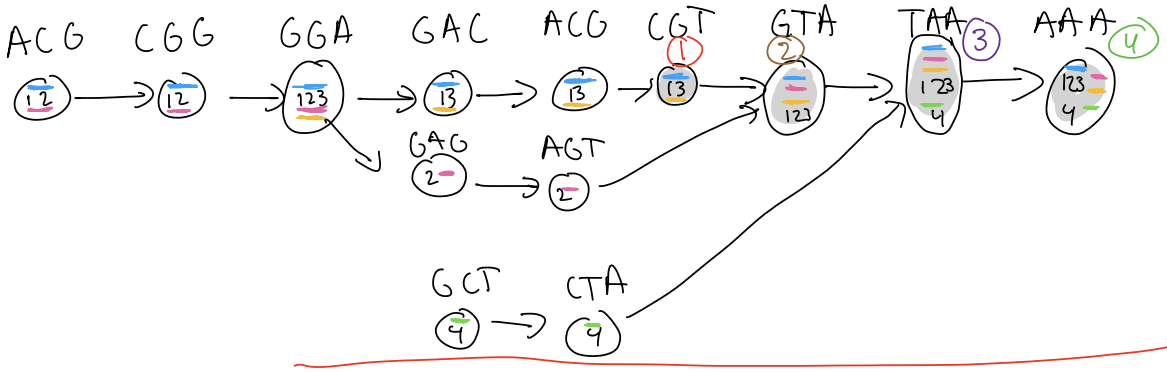
If you take the reverse complement you get a familiar looking sequence:

original: TTTACG

reverse comp: CGTAAA

Upon rigorous eyeballing, you will notice that the rev. comp. string comes from $\{t_1, t_3\}$. One approach, described in Discussion 6 is to pseudoalign the reverse complement:

CGTAAA



Pseudoalignment: $\{1,3\} \cap \{1,2,3\} \cap \{1,2,3,4\} \cap \{1,2,3,4\} = \{1,3\}$

See Disc. 6 for other approaches.