GraphSpace Preliminaries

The `HW6_utils.py` code will post graphs to the **Bio131 Spring 2017 HW6** on GraphSpace.

1. Go to GraphSpace: `www.reed.edu/biology/ritz/graphspace`
2. Log in with username `compbio@reed.edu` and password `compbio`.
3. Click on **Groups** in the top banner, then **Bio131 Spring 2017 HW6**.
4. Alternatively, you can click on **Graphs** in the top banner. The most recently uploaded graphs will appear at the top of the list.

Other notes to keep in mind:

- Remember to add your name to all your graphs posted (e.g., `'Anna-Ritz-overlap-graph'`).
- If you get HTML-related errors, there may be an issue with GraphSpace (rather than a bug in your code). Email Anna the text of the error, and she will help you debug.
- You can work on the graphs in either order - the DeBruijn graph is more straightforward to code up. There are additional exercises in the Overlap Graph.
- I encourage you to customize your graph with node styles - refer to Lab7 to see how to build a dictionary of node attributes. The graph will automatically contain **directed** edges.

Handin: Submit your code to Moodle with the string `'TAATGCCATGGGATGTT'` and $k = 3$. I will also look for your graphs on GraphSpace. I will evaluate both correctness *and* organization/commenting of your code.

Inputs

Begin with `HW6.py`. The input to both graph-building functions is a list of k-mers. First, write a function that returns all k-mers from a string:

`generate_kmers()`: *Generate all k-mers from a string.*
Inputs: A string `genome` and an integer `k`
Returns: A list of k-mers, including duplicates.

If you store the k-mer list as a variable (let's call it `kmerlist`), you can test that you can post a graph by running

```
post_test_graph(kmerlist,'Your-Name')
```

Where `'Your-Name'` is your name.

The rest of these instructions provide the solutions for the string `'abcdbcde'` and $k = 3$.

Build a De Bruijn Graph

Given a list of k-mers, a De Bruijn graph is a graph where the **edges** are k-mers that connect **nodes** representing the **prefix** of the k-mer and the **suffix** of the k-mer. The prefix is the first $k-1$ letters of the k-mer and the suffix is the last $k-1$ letters of the k-mer. For example, the edge `'abc'` will connect nodes `'ab'` to `'bc'`. Be sure that your code works with different values of k.

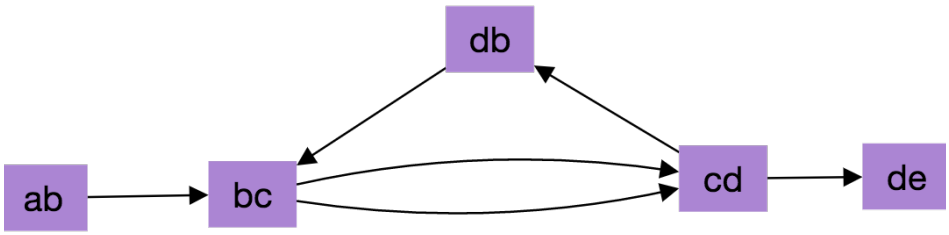
`de_bruijn()`: *Build a De Bruijn graph from a list of k-mers.*
Inputs: A list of k-mers.
Returns: A list of *edges*, where each edge is a list of two strings.

GraphSpace interprets and posts a list of lists. For example, an edge is `['ab','bc']` and a list of edges starts with `[['ab','bc'],['bc','cd'],...]`.

Once you have your edge list (in a variable called `edge_list`, for example), you can post this graph by writing:

```
node_attrs = basic_node_attributes(edge_list) # some node attributes
HW6_utils.postBio131Graph(edge_list,node_attrs,'Your-Name-deBruijn')
```

Remember: In a De Bruijn graph, we wish to find a path that contains every **edge** exactly once. GraphSpace will automatically collapse nodes and it will *not* collapse edges. This is the correct thing to do. The De Bruijn graph for `'abcdbcde'` and $k = 3$ is the following:



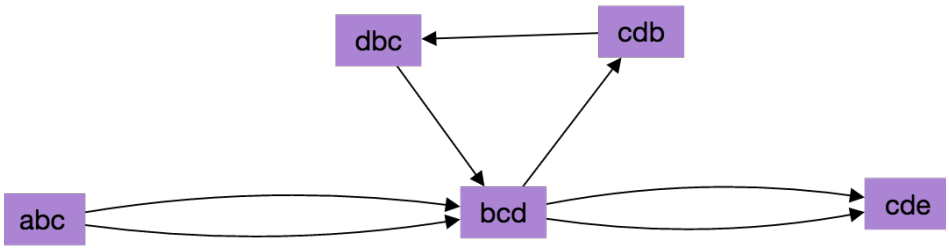
Build an Overlap Graph

Given a list of k-mers, an overlap graph is a graph where the **nodes** are k-mers, and **edges** connect k-mers if the suffix (last $k-1$ letters) of one k-mer matches the prefix (first $k-1$ letters) of another k-mer. For example, the node `'abc'` will be connected to the node `'bcd'`. Be sure that your code works for different values of k.

`overlap()`: *Build an overlap graph from a list of k-mers.*
Inputs: A list of k-mers.
Returns: A list of *edges*, where each edge is a list of two strings.

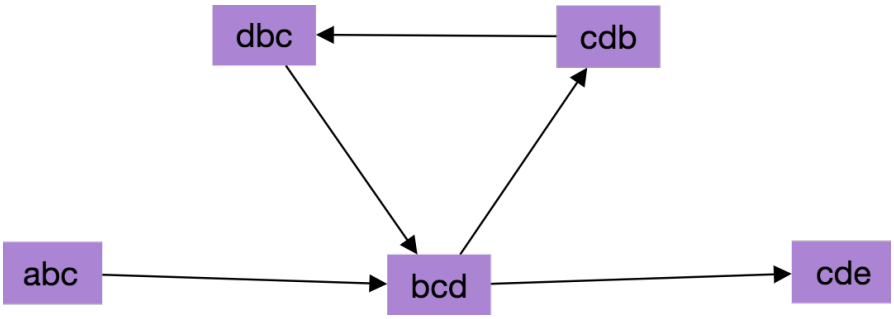
Once you have your edge list, you can post this graph by writing similar code to above (with a different graph name, e.g. `'Your-Name-Overlap'`).

Remember: In an overlap graph, we wish to find a path that contains every **node** exactly once. GraphSpace will automatically collapse nodes and it will *not* collapse edges. This is the wrong thing to do. Your first attempt at the overlap graph for `'abcdbcde'` and $k = 3$ may look like the following:

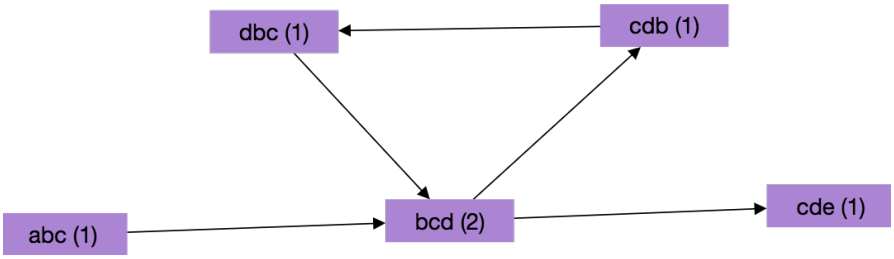


Complete at least one of the following improvements. The remaining improvements are optional additional exercises.

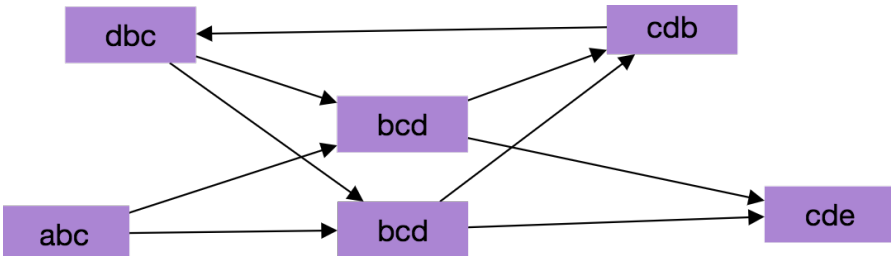
1. **Remove duplicate edges**. Unlike the De Bruijn graph, multiple edges don't tell us anything. Modify your edge list to only include **unique** edges. This is similar to removing duplicate k-mers in frequent words. Your graph may now look like this:



2. **Include the number of nodes that *should* be present.** In an overlap graph, nodes should not be collapsed. GraphSpace collapses them, though! Add a `'content'` node attribute that displays the text of the node (which may be different than the node name). Add the number of times each k-mer occurs as the node label. This is similar to counting the number of times each word appears in the text (dictionaries!). Your graph may now look like this:



3. **Finally, duplicate the nodes.** This is tricky - you need to make every k-mer unique (e.g., you will have `'bcd1'` and `'bcd2'`), retain the underlying edges. That is, every edge that was connected to `'bcd'` is now connected to *both* `'bcd1'` and `'bcd2'`. The `'content'` attribute will contain the original k-mer.



This is the **real** overlap graph.

Handin: Submit your code to Moodle with the string `'TAATGCCATGGGATGTT'` and $k = 3$. I will also look for your graphs on GraphSpace. I will evaluate both correctness *and* organization/commenting of your code.