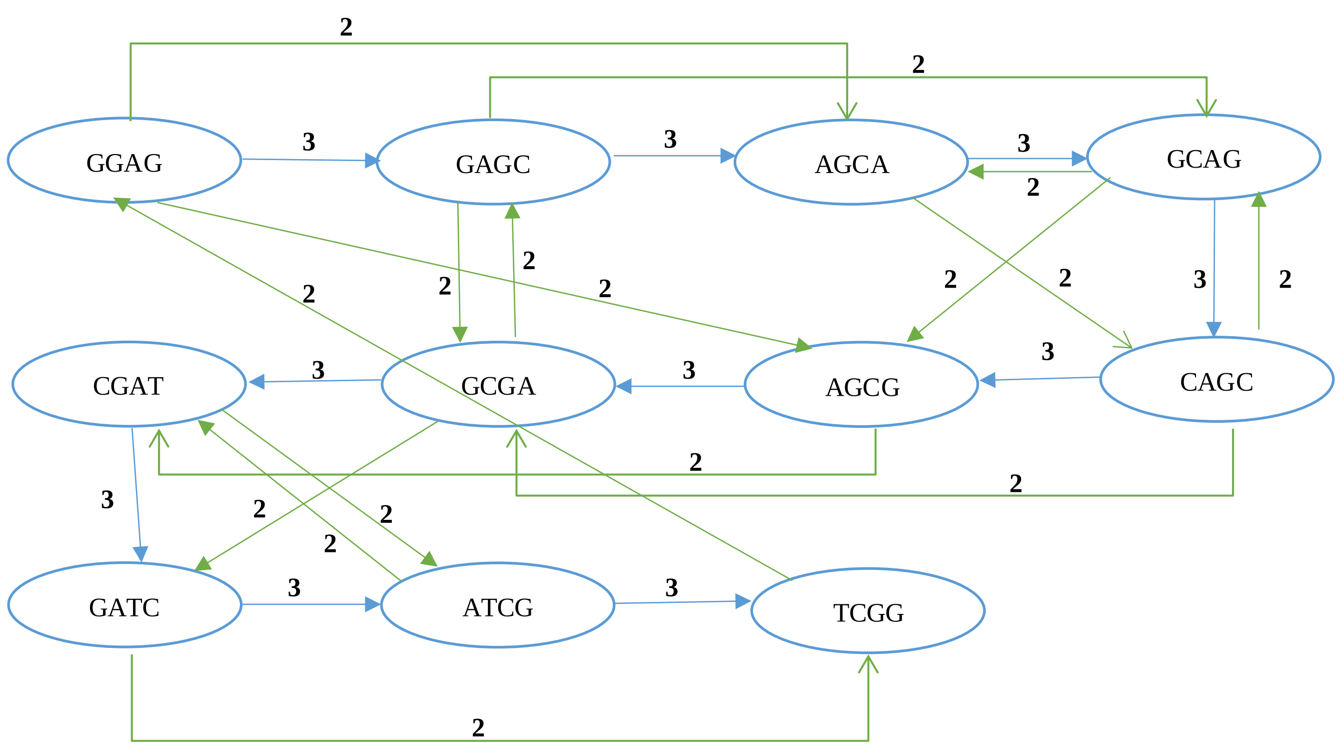
Problem 1 (60pts).

Suppose you are given the following 4-mers or reads of an unknown string.

CGAT
ATCG
GCAG
AGCG
GAGC
GATC
CAGC
GGAG
AGCA
GCGA
TCGG

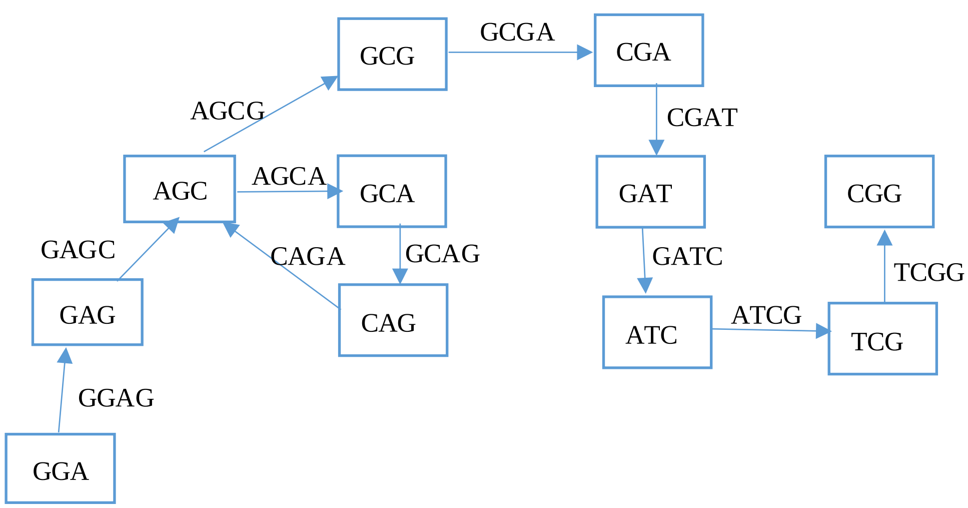
(a) Create an overlap graph of these reads that may be used to reconstruct the original string. (15 points)



(b) Reconstruct a string that covers the above 4-mers using the overlap graph by constructing the longest path that visits all nodes. (15 points)

String: GGAGCAGCGATCGG

(c) Create a De Bruijn graph of these reads that may be used to reconstruct the original string. (15 points)



(d) Reconstruct a string that covers the above 4-mers using the De Bruijn graph by constructing the Eulerian path of this graph. (15 points)

String: GGAGCAGCGATCGG

Problem 2 (Extra credits : 20 points)

We consider a greedy algorithm for the shortest superstring problem (See lecture slides): merge a pair of strings with maximum overlap and repeat until only one string left.

Provide an efficient implementation in pseudo-code, runtime analysis, and an example to show that this greedy algorithm cannot find the optimal solution.

We are given S, a collection S of strings, and an overlap graph OG(S) = (V, E) is a weighted directed graph. And in OG(S), we are looking for paths that use every vertex exactly once so that we will going to use Hamiltonian paths.

The greedy algorithm always tries to take heaviest edge which not selected so far, and we avoid cycles.

We set inputs as weighted directed graph OG(S) with n vertices.
Output as Hamiltonian path in OG(S) – shortest common superstring

Pseudo-code:

```
For i = 1 to n:
        inputs[i] = 0;
        outputs[i] = 0;
        connected[i] = {i};
Sort edges by using weight from heaviest to lightest
For each edge (i, j) in sorted edge
        if inputs[j] == 0 and outputs[i] == 0 and connected[i] != connected[j]
                then select (i, j); inputs[j] = 1; outputs[i] = 1; merge(connected[i], connected[j]);
        if only one string
                then break
return selected edges
```

For initializing, we will have O(n) time, sorting will have O(n^2logn), for processing edge, we will have O(n^2), finally we will return, so it going to be O(n) again.

Totally, we will have O(n)+ O(n^2logn) + O(n^2) + O(n) = **O(n^2logn)**

Example:
We will have 5 vertices:
AAA, AAB, ABB, BBA, BBB

AAAB, ABB, BBA, BBB

AAAB, ABBA, BBB

AAABBA, BBB

AAABBABBB → the results we get from greedy algorithm, length = 9

AAABBBA → The optimal shortest common superstring, length = 7