Задачи:

1. Имплементирайте ориентиран и неориентиран граф API.
   Нека съдържат методите:
   - toString()

   и предефиниран оператор за изход
   - operator<<(...)

   Нека ориентираният граф да има следните методи:
   - indegree(int v)
   - outdegree(int v)
   - reverse()

   Бонус: Направете абстрактен клас граф, който е наследен от ориентирания и неориентирания граф.
   Навсякъде да се използват ваши лични имплементации на структурите stack, queue, list, etc.

Неориентирани графи

2. Имплементирайте простичък graph-processing API за неориентиран граф със следните методи:
   - degree(Graph G, inv v) -> степента на върха v
   - maxDegree(Graph G)
   - avgDegree(Graph G)
   - numberOfSelfLoops(Graph G)

   Note.
   Design pattern. Decouple graph data type from graph processing.
   - Create a Graph object.
   - Pass the Graph to a graph-processing routine.
   - Query the graph-processing routine for information.

Каква е **сложността** на всеки един от алгоритмите по **време** и **памет**? **Това условие се отнася за всяка една задача**.

3. Имплементирайте Graph client, който използва DFS и BFS за неориентиран граф, за да провери дали има път от даден връх до всички други. Имплементирайте DFS рекурсивно и нерекурсивно.
4. Имплементирайте Graph client за неориентиран граф, който намира броят на свързаните компоненти в граф, както и отговаря на въпроса дали 2 върха са свързани.
5. Имплементирайте Graph client, който проверява дали неориентиран граф е:
   - цикличен
   - двуделен
6. **Symbol graph**. Често срещано приложение е обработването на графи, дефинирани чрез файлове, стрингове и/или индекси, които не са цели числа. Трябва да се справим с входни данни със следните свойства:
   - Имената на върховете са стрингове.
   - Специално обозначен разделител дели имената на върховете.
   - Всеки ред съдържа два върха, разделени с оказания разделител.
   - Броят върхове и ребра не са експлицитно посочени.

Бонус, ако го имплементирате, така че да чете подаденият вход само веднъж.
Обяснение: интуитивната идея е да се прочете входът веднъж, за да се преброят върховете, и втори път, за да се добавят ребрата.

Примерен тестов клиент:

```
String filename = args[0];
String delim = args[1];
SymbolGraph sg = new SymbolGraph(filename, delim);
Graph G = sg.G();
while (std::cin)
{
        String source;
        std::cin.getline(source);
        for (int w : G.adj(sg.index(source)))
                std::cout << " " << sg.name(w));
        std::cout << std::endl;
}
```

Задача за вкъщи:
The eccentricity of a vertex v is the the length of the shortest path from that vertex to the furthest vertex from v. The diameter of a graph is the maximum eccentricity of any vertex. The radius of a graph is the smallest eccentricity of any vertex. A center is a vertex whose eccentricity is the radius.The girth of a graph is the length of its shortest cycle. If a graph is acyclic, then its girth is infinite.
Имплементирайте API, който по даден граф намира:

- eccentricity
- diameter
- radius
- center
- girth

// Hint : Run BFS from each vertex. The shortest cycle containing s is a shortest path from s to some vertex v, plus the edge from v back to s.

Ориентирани графи

7. Имплементирайте Graph client, който сортира топологично DAG.
8. Имплементирайте Graph client, който проверява дали ориентиран граф:
   - е цикличен
   - имплементира алгоритъмът на Kosaraju за силно свързани компоненти.
9. ** Имплементирайте Graph client, който проверява дали ориентиран граф има Ойлеров:
   - път
   - цикъл
10. **2-satisfiability**. Given boolean formula in conjunctive normal form with M clauses and N literals such that each clause has exactly two literals, find a satisfying assignment(if one exists). **Hint** : Form the implication digraph with 2N vertices (one per literal and its negation). For each clause x + y, include edges from y' to x and from x' to y. To satisfy the clause x + y, (i) if y is false, then x is true and (ii) if x is false, then y is true.

Claim: The formula is satisfiable if and only if no variable x is in the same strong component as its negation x'. Moreover, a topological sort of the kernel DAG (contract each strong component to a single vertex) yields a satisfying assignment.

11. **Boggle**.
    Given a dictionary, a method to do lookup in dictionary and a M x N board where every cell has one character. Find all possible words that can be formed by a sequence of adjacent characters. Note that we can move to any of 8 adjacent characters, but a word should not have multiple instances of same cell.

    Example:

    Input: dictionary[] = {"VANKATA", "JORETO", "KUR", "MERRY", "CHRISTMAS"};
    boggle[][] = {{'U', 'V', 'J', 'S', 'T'},
    {'Y', 'O', 'A', 'N', 'K'},
    {'R', 'R', 'E', 'M', 'A'},
    {'V', 'O', 'T', 'A', 'T'}};
    isWord(str): returns true if str is present in dictionary
    else false.

    Output: Following words of dictionary are present
    VANKATA
    JORETO
    MERRY
    // **Hint**: we may not use a graph object, rather imply the idea of graph traversal in a matrix