Problem 1. Consider a random network with, say, 7 nodes. What is the probability that every node has exactly 3 neighbors?

Solution.

We could calculate probability in frame of next formula: $P(A) = \frac{n}{N}$, where n — number of elementary events, which are the components of A event and N — number of all possible elemeentary events.

In our case A event is — network with 7 nodes, each of which has exactly exactly 3 neighbors. Analyzing A event we can conclude, that this event is equal to construction a 3-regular graph on 7 vertices.

But we know, that the sum of the degrees of all vertices is equal to twice the number of vertices. Hence, there is no 3-regular graph on 7 vertices because its degree sum would be $7 \cdot 3 = 21$, which is not even.

Answere: Probability is equal to zero. $P(A) = 0$.

Problem 2. What can you say about the sign of the expression:

$$f(u) = \Big(2u+1\Big)ln\Big(1+\frac{1}{u}\Big) - 2 \quad u > 0.$$

Solution.

We are going to solve this problem graphically. First step - asymptote calculation. As far as we know there are three types of asymptotes(horizontal, vertical and oblique).

Vertical asymptote is obviously $u = 0$, because denominator of $\frac{1}{u}$ could not be equal to zero, this lead to asymptote equation listed above.

Horizontal aymptote calculation:

$$\lim_{u\to\infty}\Big(2u+1\Big)ln\Big(1+\frac{1}{u}\Big) = \lim_{u\to\infty}\Big(2uln\Big(1+\frac{1}{u}\Big)+ln\Big(1+\frac{1}{u}\Big)\Big) = 2\lim_{u\to\infty}\Big(uln\Big(1+\frac{1}{u}\Big)\Big)+$$

$$+\lim_{u\to\infty}\Big(ln\Big(1+\frac{1}{u}\Big)\Big) = 2\lim_{u\to\infty}\Big(uln\Big(1+\frac{1}{u}\Big)\Big)+0 = 2ln\Big(\lim_{u\to\infty}\Big(u\Big(1+\frac{1}{u}\Big)\Big) = [u>0] =$$

$$2ln\Big(\lim_{u\to\infty}\Big(1+\frac{1}{u}\Big)^u\Big) = [\lim_{u\to\infty}\Big(1+\frac{1}{u}\Big)^u = e] = 2ln(e) = 2.$$

Using calculations above, we have come to the next conclusion: $\lim_{u\to\infty} f(u) = 2 - 2 = 0$. So, horizontal aymptote equation is $y = 0$

Now we are going to find oblique aymptote. Equation of a line could be written as follows : $y = au + b$. First of all, we find value of a coeffisient:

$$a = \lim_{u\to\infty}\left(\frac{(2u+1)ln\big(1+\frac{1}{u}\big)}{u}\right) = \lim_{u\to\infty}\Big(ln(1+\frac{1}{u}\Big) + \lim_{u\to\infty}\Big(\frac{1}{u}ln\Big(1+\frac{1}{u}\Big)\Big) =$$

$$= 0 + \lim_{u\to\infty}\Big(\frac{1}{u}ln\Big(1+\frac{1}{u}\Big)\Big) = \lim_{u\to\infty}\Big(\frac{1}{u}\Big) * \lim_{u\to\infty}\Big(ln\Big(1+\frac{1}{u}\Big) = 0.$$

Next, find value of b coefficient:

$$b = \lim_{u\to\infty}\Big(f(u) - au\Big) = b = \lim_{u\to\infty}\Big(f(u) - 0\Big) = \lim_{u\to\infty}\Big(f(u)\Big) = 0.$$

As a result, **oblique asymptote** is equal to horizontal asymptote: $y = 0$. Now, to obtain sign of the expression we need to calculate value of $f(u)$ function in u-point such as $u > 0$.

For this purpose we choose point $u = \frac{1}{2}$. Calculation of value of f function in this point are listed below: $f(\frac{1}{2}) = 2ln(3) - 2 = 2(ln(3) - 1)$. The only remain thing — is to obtain the sign of $ln(3) - 1$ expression.

$$ln(3) \quad v \quad 1$$

$$3 \quad v \quad e$$

$$3 \quad > \quad e$$

So, the sign of the expression $f(\frac{1}{2}) > 0$. This means, that all graph of $f(u)$ lies in the first quarter for $u > 0$, and the sign of the $f(u)$ expression is $+$.

Answere: sign of the $f(u)$ expression is positive for $u > 0$.

Problem 3. In some city a car starts moving from the central square to some direction. The driver applies a funny algorithm turning right at every intersection. More precisely, the car turns to the nearest right if many available. Assume that all roads are two-ways (in both directions)and the number of roads at any intersection is greater than 2 and can be more than 4. Besides that, there can be multilevel roads so that you cannot assume that roads over the same x;y intersect. Can you guarantee that the driver will return back to the central square in every city like this?

Solution.

The city can be represented as a graph whose nodes are the points of intersections of roads, and the edges - leaving from this intersection roads. This edges lead to next intersection points, which are new nodes of the graph, and so on.

Moreower this graph has exactly even number of neigbours, because each road in intersection point lead to two different next intersections. And the main graph property - it is connected graph, so we can get from any node of the graph to any other node.

This observations lead us to conclusion that this graph is euler graph, in which we could construct euler cicle (driver could visits every edge exactly once and returning to the start point), which means that the driver will return back to the central square.

Answere: Yes, the driver will return back to the central square.

Problem 4. Please indicate the issues you see from the code above. Fix those issues and rewrite the class definition

Solution:

Below you can find rewriting of class *Complex*. For readability I used *re* and *im* fieds name instead of *real* and *imaginary*.

```
#include<iostream>
#include<fstream>

class Complex
{
public:
  Complex();
  Complex(const int& real);
  Complex(const int & real, const int & imaginary);
  ~Complex() {}

  Complex operator+(const Complex & other) const
  {
    return  Complex(re_ + other.re_, im_ + other.im_);
  }

  Complex & operator++()
  {
    ++re_;
    return *this;
  }

  Complex operator++(int)
  {
    Complex result(*this);
    ++(*this);

    return *this;
  }

  friend std::ostream & operator<<(std::ostream & os, const Complex & obj);

private:
  double re_;
  double im_;
};


Complex::Complex() : re_(0), im_(0) { }

Complex::Complex(const int & real) :
        re_(real), im_(0) { }

Complex::Complex(const int & real, const int & imaginary):
        re_(real), im_(imaginary) { }

std::ostream & operator<<(std::ostream & os, const Complex & obj)
{
  os << obj.re_ << " + i * " << obj.im_;
  return os;
}
```

Problem 5. Checking whether a tree is balanced. You need to implement a function checking whether the given tree is balanced or not.Please provide a generation of a tree at random and a test function that generates trees and then checks whether those trees are balanced with different answers happening in the test.

Solution:

To solve the problem I used an algorithm, description of which is listed below:

Main idea, which is taken into account is: if tree is empty - it is balanced, otherwise it will be balanced if three next conditions are performed for all nodes in this tree:

1. Left subtree is balanced.
2. Right subtree is balanced.
3. The height difference between left and right subtrees is not bigger than 1.

So, I developed a function which checks the performance of listed above three conditions for the current node and recursively calls itself to check the performance of this conditions for left and right sons of this node. Recursive calls are held until the current node is not a leaf of the tree. For deeply understanding of algorithm implementations, let's consider the pseudo code is listed below:

```
bool IsBalanced(NodePtr curNode, int* ptrToHeight)
{
  if (curNode is Leaf)
  *ptrToHeight = 0;
  return true;

  // Recursive calls
  IsBalanced(curNode->left, &ptrLeftHeight);
  IsBalanced(curNode->right, &ptrRightHeight);

  update *ptrToHeight;

  if (abs(leftHeight - rightHeight) > 1)
    return false;
  else if (leftIsBalansed && rightIsBalanced)
    return true;
}
```

Calling function *IsBalanced*(), first of all we check current node is a leaf, and if it is true set height value to be equal to zero. Otherwize we make two recursive calls to check left and right sons. Then we calculate height of the current node, based on the heights of left and right sons, obtained on the basis of recursive calls. At last, with already calculated height of left and right subtrees and information about their balance state we obtain a balance state for current node using three ideas listed above.

Algorithm complexity:

So, we have a two recursive calls from a function, each of which is dealing in average with only half of the nodes (left or right half of the tree). Besides we calculate height of the tree after each recursive calls. Height update time cost is about $O(1)$. So, we have come to the next recurrent formula of algorithm complexity:

$$T(n) = 2T\left(\frac{n}{2}\right) + O(1).$$

Using the theorem about recursive algorithm complexity we obtain that the time complexity is equal to $T(n) = O(n^{log_2 2}) = O(n)$.

Space complexity is $O(n)$ because of n stack frames usage (We deal only with several variables created in the body of the function).

Answere: Time and space complexity is about $O(n)$.

Problem 5. Military tanks ride only by asphalt roads on the planet Barsoom. While moving, tanks destroy roadbed under themselves. A very modern tank Armata made a trip from the Capital to Zodanga city. We know that minimal road width on the Barsoom didn't change after this trip. What values can Armata's width take?

Solution.

Main idea, which is taken into account is: a map of the cities on the planet Barsoom could be represented as an oriented graph, each edge of which store roads width as na additional parameter.

But before algorithm discussion, first thing we need to do — we need randomly generate roads between the cities. This tasks solved in *Generator* class.

Depending on the number of input number of cities(n) and number of roads(N) we has two different algorithms of roads generation:

- If $N > n(n-1)$, or input number of roads is bigger them maximum value of roads between n cities. In this case we generate connected graph win random roads width.
- If $N < n(n-1)$, we generate two random numbers in range $[0, n)$(Indexes of start and finish of the road) and a random value of roads width. If this values are already generated we generate a new one untill all number of generated roads is equal to N.

When we have read the conditions from generated file, for task solution we must perform next steps (this is main algorithm in this task):

1. Construct a graph of the map, based on the data reading from file. And in the same time obtain minimum road width.
2. Capital city always has index 0, while Zodanga has index (n-1)(It could be easily changed in the code). So, to perform the task we need to obtain all paths between this two nodes of graph. To solve this task we will use method based on depth first traversal task solution. Pseudo code of such procedure is listed below.
3. When we find such a way, we loop by all roads on this way and find minimum possible Armata's width on this way.
4. If we have several different paths, we need to choose maximum from all possible Armata's widths on this paths. This value is the upped boundary of all possible Armata's width.

```
void FindStep(start, finish, vector isVisited, vector path)
{
  ...
  if (start == finish)
  {
        FindMaxArmatasWidth(path);
  }
  else
  {
        for(all connected to this node edges)
          if(node is not visited && road width > min width)
            FindStep(node start, finish, isVisited, path);
  }
  ...
}
```

Algorithm complexity.

For the algorithm listed above, it is obvious that we can find single path for $O(n + N)$ time. Because we visit n nodes, for at most N times(While we found the path, we will check at most n edges while we passing by N roads). Space cost is $O(n)$ because we need to store vector *isVisited* and *path*.

When we need to obtain several paths we depend on the number of neighbors of the node, let it be p. So, to find all paths, we need np times calls single $FindStep$ function, so the algorithm complexity is $O(np * (n + N))$. In worst case each node has n neighbors, so the complexity in worst case is $O(n^3)$. Space complexity depends on the recursion height in the same time.

Answere: Time dependancy is about $O(np * (n + N))$, where n —number of the cities, N — number of the roads and p — is the average value of neighbors, space complexity is about $O(n)$.