# QiShi Quiz 1

Xiangyu Zhang

May 2019

## 1 Math

### Problem 1

According to wiki, a chi-squared test, also written as $\chi^2$ test, is any statistical hypothesis test where the sampling distribution of the test statistic is a chi-squared distribution when the null hypothesis is true.

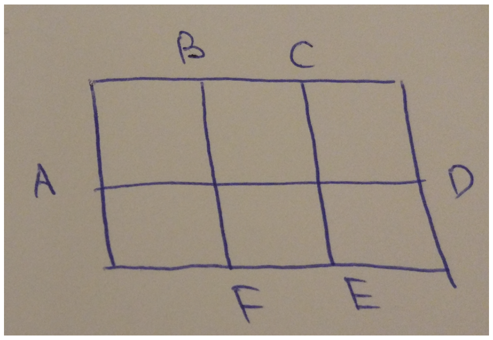
There are many $\chi^2$ test. A famous one is Pearson Pearson's chi-squared test. Assume there are $k$ categories in total, and the probability item sampled from category $i$ is $p_i$. We observe $n$ sampled item, and there are $x_i$ in category $i$.

$$PearsonTest = \sum_{i=1}^{k} \frac{(x_i - np_i)^2}{np_i} \to \chi^2 \tag{1}$$

with freedom $k-1$.

### Problem 2

By Euler Theorem, there exists a loop which goes through all the edges in a graphic is equivalent to there exists at most 2 vertex with odd degree.



So by vertex A, B, C, D, E and F having odd degree, we know the shortest routine going through all edges is at least $17 + 2 = 19$. To achieve this lower bound, we can collect B and C, E and F by another two edges.

## Problem 3

$$\begin{aligned} P(X = x | X + Y > 0) &= \frac{P(X = x, X + Y > 0)}{P(X + Y > 0)} \\ &= \frac{P(X = x, Y > -x)}{\frac{1}{2}} \\ &= 2P(X = x)P(Y > -x) \\ &= \frac{1}{\pi} exp(-\frac{x^2}{2}) \int_{-x}^{+\infty} exp(-\frac{y^2}{2}) dy \end{aligned}$$

## Problem 4

For subproblem (1), (2) and (3)

(1) We should keep rolling the dice. To prove that, we use the strategy describing in (2) has higher expected reward than direct stopping.

Expected reward by direct stopping = 35.

Expected reward by strategy in (2) $\geq \frac{5}{6} * 43 > 35$.

(2) The sum value before exceeding 43 could be 37, 38, 39, 40, 41 and 42. We use $p_{37}, p_{38}, p_{39}, p_{40}, p_{41}$ and $p_{42}$ to denote the density for each sum value. The stop value could be 43, 44, 45, 46, 47 and 48, and we use $p_{43}, p_{44}, p_{45}, p_{46}, p_{47}$ and $p_{48}$ to denote the corresponding density. It is easy to see

$$\begin{aligned} p_{43} &= \frac{1}{6}(p_{37} + p_{38} + p_{39} + p_{40} + p_{41} + p_{42}) \\ p_{44} &= \frac{1}{6}(p_{38} + p_{39} + p_{40} + p_{41} + p_{42}) \\ p_{45} &= \frac{1}{6}(p_{39} + p_{40} + p_{41} + p_{42}) \\ p_{46} &= \frac{1}{6}(p_{40} + p_{41} + p_{42}) \\ p_{47} &= \frac{1}{6}(p_{41} + p_{42}) \\ p_{48} &= \frac{1}{6}(p_{42}) \end{aligned}$$

So we can conclude that 43 is most probable amount to stop.

(3) There exits at least one state we need to stop. If not, we will never stop rolling the dice. In this case, the expected reward is 0 because with probability 1, the sum will hit some square value, which implies non optimality.

## Problem 5

Given a stick, and randomly cut into $n$ pieces, then average length of $k^{th}$ longest segment $S_{(k)}$ is

$$\mathbb{E}[S_{(k)}] = \frac{1}{n} \sum_{j=1}^{k} \frac{1}{n-j+1}$$

The proof is given in [2].

## Problem 6

We give calculation of Problem 6 below.

(1) We use $I_k$ to denote the $k^{th}$ person takes his own hat. Then we have

$$Y = \sum_{k=0}^{N} I_k \tag{2}$$

and we have

$$\mathbb{E}[Y] = \mathbb{E} \sum_{k=0}^{N} I_k = \sum_{k=0}^{N} \frac{1}{N} = 1 \tag{3}$$

(2) According to notation in (1), we have

$$Y^2 = (\sum_{k=0}^{N} I_k)^2 = \sum_{k=0}^{N} I_k + 2 \sum_{i<j} I_i I_j \tag{4}$$

By $\mathbb{E}[I_k] = \frac{1}{N}$, and $\mathbb{E}[I_i I_j] = \frac{1}{N(N-1)}$, we know

$$\mathbb{E}[Y^2] = 2 \tag{5}$$

which implies

$$var(Y) = 1 \tag{6}$$

(3) We denote the probability that $k$ people take the right hat, and $N-k$ people take the wrong hat as $p_k$.

According to (1) and (2), we know

$$\sum_{k=0}^{N} p_k = 1, \sum_{k=0}^{N} k p_k = 1, \sum_{k=0}^{N} k^2 p_k = 2 \tag{7}$$

And if we denote the $R(N)$ as number of round, $S(N)$ as number of selection and $F(N)$ as number of false selection, we can the following dynamic equation

$$\begin{aligned} R(N) &= 1 + \sum_{k=0}^{N} p_k R(N-k), R(0) = 0, R(1) = 1 \\ S(N) &= N + \sum_{k=0}^{N} p_k S(N-k), S(0) = 0, S(1) = 1 \\ F(N) &= N - 1 + \sum_{k=0} p_k F(N-k), F(0) = 0, F(1) = 1 \end{aligned} \tag{8}$$

According to 7 and induction, we could conclude

$$\begin{aligned}
&R(0) = 0, R(1) = 1, R(N) = N \text{ for } N \geq 2 \\
&S(0) = 0, S(1) = 1, S(N) = \frac{N(N+2)}{2} \text{ for } N \geq 2 \\
&F(0) = 0, F(1) = 1, F(N) = \frac{N^2}{2} \text{ for } N \geq 2
\end{aligned} \tag{9}$$

## Problem 7

Without loss of generality, we assume $Y^TY = 1, X_1^TX_1 = 1$ and $X_2^TX_2 = 1$. Also we denote $\rho_1 = cov(X_1, Y), \rho_2 = cov(X_2, Y)$ and $\rho = cov(X_1, X_2)$. The $R^2$ of Model 3 is

$$R^2 = \frac{\rho_1^2 + \rho_2^2 - 2\rho_1\rho_2\rho}{1 - \rho^2}$$

subject to constraint $\rho^2 - 2\rho_1\rho_2\rho + \rho_1^2 + \rho_2^2 \leq 1$. By $R^2$ of Model 1 and Model 2, we know $\rho_1 = \sqrt{0.1}, \rho_2 = \sqrt{0.2}$. So by calculating the derivative, we know the maximum of $R^2$ of Model 3 is achieved at

$$\rho^*_{max} = \rho_1\rho_2 - \sqrt{(1 - \rho_1^2)(1 - \rho_2^2)}$$

And the minimum if achieved at

$$\rho^*_{min} = \frac{\rho_1}{\rho_2}$$

We conclude $0.2 \leq R^2 \leq 1$.

The lower bound is intuitive, for why is the upper bound 1?

For Model 1 $R^2 = 1 - cov(X_1, Y)^2$, and for Model 2 $R^2 = 1 - cov(X_2, Y)^2$. So if we treat $X_1$, $X_2$ and $Y$ as vectors, the information we know is the angle between $X_1$ and $Y$, the angle between $X_2$ and $Y$. And by $R^2 = 1 - \frac{(Y - \hat{Y})^2}{Y^2}$, we want to minimize $(Y - \hat{Y})^2$ to maximize $R^2$. If $Y$ lies on the two dimensional linear space generated by $X_1$ and $X_2$, then $(Y - \hat{Y})^2 = 0$, which implies $R^2 = 1$.

## Problem 8

We can write the sampling algorithm as below.

1. Find the smallest $k$, s.t. $2^k > n$.

2. Toss the coin for $k$ times, and get a sequence of 0 and 1 as $a_1$, $a_2$, ..., $a_k$. We could form a binary number $v = a_1a_2...a_k$ s.t. $0 \leq v \leq 2^k - 1$.

3. If $v > n - 1$, go back to step 2 and sample $v$ again.

4. If $v = 0$, return head; else return tail.

## Problem 9

We use $N_k$ as the expected number of bridges passed when first successfully passing $k^{th}$ bridge. Initially $N_0 = 0$.

To pass the $(k+1)^{th}$ bridge, we may go through all the previous bridges multiple times due the the weakness of $(k+1)^{th}$ one. The avarage times is 1 by property of geometric distribution. So we have dynamic equation

$$N_{k+1} = N_k + k + 1 \tag{10}$$

So we have $N_k = \frac{k(k+1)}{2}$ and specifically, $N_9 = 45$.

## 2 Programming

### Problem 10

The function name is fun, the input is of type const int* cont &, which means the input is a reference to a constant pointer pointing to a constant int. The return is of type const int* const, which means the return is a constant pointer pointing to a constant int. The function is a constant function, which means if this function is a method in a class, then all the class field can not be changed after calling this function.

### Problem 11

Python code is given below.

```
# problem 11
class node(object):
    def __init__(self, val, nextNode):
        self.val = val
        self.next = nextNode

def delete(root, node):
    if root == None:
        return

    if root == node:
        return root.next

    current = root
    while current.next != None and current.next != node:
        current = current.next

    if current.next == None:
        print("No node found")
        return
```

```
current.next = current.next.next
return
```

## Problem 12

The following code gives one naive implementation of abstract Matrix class in C++.

```
#include <iostream>
#include <string>

using namespace std;


class Matrix
{
    private:
    int row; // number of row
    int col; // number of column
    double** m; // store the matrix value

    public:
    Matrix(int i, int j){
        row = i;
        col = j;
        m = new double*[row];
        for (int p = 0; p < row; p++){
            m[p] = new double[col];
        }
    }

    int numberOfRow(){
        return row;
    }

    int numberOfCol(){
        return col;
    }

    double entry(int i, int j){
        return m[i][j];
    }

    void setEntry(int i, int j, double v){
        m[i][j] = v;
    }

    // return the summation with matrix b
    virtual Matrix* add(Matrix* b) = 0;

    // return the multiplication with matrix b, b is on the right
    virtual Matrix* multiply(Matrix* b) = 0;

    // return inverse of matrix
    virtual Matrix* inverse() = 0;

};
```

## Problem 13

No, the constructor could not be virtual. We could write a static function as the simulated virtual constructor. SEE THE CODE BELOW.

```
#include <iostream>
using namespace std;

//// LIBRARY START
class Base
{
public:

  // The "Virtual Constructor"
  static Base *Create(int id);

  Base() { }

  virtual // Ensures to invoke actual object destructor
  ~Base() { }

};

class Derived1 : public Base
{
public:
  Derived1()
  {
    cout << "Derived1 created" << endl;
  }

  ~Derived1()
  {
    cout << "Derived1 destroyed" << endl;
  }

};

class Derived2 : public Base
{
public:
  Derived2()
  {
    cout << "Derived2 created" << endl;
  }

  ~Derived2()
  {
    cout << "Derived2 destroyed" << endl;
  }

};

```

```
class Derived3 : public Base
{
public:
  Derived3()
  {
    cout << "Derived3 created" << endl;
  }

  ~Derived3()
  {
    cout << "Derived3 destroyed" << endl;
  }

};

Base *Base::Create(int id)
{
  if( id == 1 )
  {
    return new Derived1;
  }
  else if( id == 2 )
  {
    return new Derived2;
  }
  else
  {
    return new Derived3;
  }
}
//// LIBRARY END

int main()
{
  Base* p = Base::Create(1);
  delete p;

  Base* q = Base::Create(2);
  delete q;

  Base* r = Base::Create(3);
  delete r;
}
```

The output is

```
Derived1 created
Derived1 destroyed
Derived2 created
Derived2 destroyed
Derived3 created
Derived3 destroyed
```

## Problem 14

In C++, it is legal for a non-virtual function to can a virtual function. But this is not recommended. SEE THE CODE BELOW.

```cpp
#include <iostream>
using namespace std;

class Base
{
public:
  virtual void print()
  {
    cout << "Base class print function \n";
  }
  void invoke()
  {
    cout << "Base class invoke function \n";
    this -> print();
  }
};

class Derived: public Base
{
public:
  void print()
  {
    cout << "Derived class print function \n" ;
  }
  void invoke()
  {
    cout << "Derived class invoke function \n";
    this -> print(); // called under non - virtual function
  }
};

int main()
{
  Base *b = new Derived;
  b -> invoke();
  return 0;
}
```

The output should be

```
Base class invoke function
Derived class print function
```

## Problem 15

Python code is given below.

```python
# problem 15
def longest_palindrome_subsequence(s):
     n = len(s)
     palind = [["" for i in range(n)] for j in range(n)]
     for i in range(n):
          palind[i][i] = s[i]

     for length in range(2, n + 1):
        for start in range(0, n - length + 1):
            end = start + length - 1
            if s[end] == s[start]:
                palind[start][end] = s[start] + palind[start + 1][end - 1]
                                     + s[end]
            else:
                a = palind[start + 1][end]
                b = palind[start][end - 1]
                palind[start][end] = a if len(a) > len(b) else b
    return palind[0][n-1]
```

## Problem 16

Python code is given below.

```
# problem 16
def reverse_sentence(sent):
    word_list = sent.split(" ")
    word_list = word_list[::-1]
    return " ".join(word_list)
```

## Problem 17

Python code is given below.

```
# problem 17
def max_subarray(l):
    cur_sum, max_sum = 0, 0
    for v in l:
        cur_sum = max(0, cur_sum + v)
        if cur_sum > max_sum:
            max_sum = cur_sum
    return max_sum

def max_profit(price):
    n = len(price)
    if n <= 1:
        return 0

    diff = [price[i + 1] - price[i] for i in range(0, n - 1)]
    return max_subarray(diff)
```

## Problem 18

Finding a Hamiltonian Cycle in general is NP-hard. But in this seeting, we could find an algorithm achieving $O(N^2)$ time complexity. The key reason is that by

Dirac Thereom [1], there exists at least one Hamiltonian Cycle. Understanding the proof for Dirac Theorem is essential for understanding the algorithm.

We give pseudo-code below.

```
input: graph G = (V, E), we denote the edge between vertex u and v
    as u-v.

// Extend path u_1 - u_2 - u_3 - ... - u_k to a new path v_1 - v_2
    - v_3 - ... - v_n, s.t. all the adjacent vertex of v_1 and v_n
    are in the path.
def path_extension(u_1 - u_2 - ... - u_k):
    path = u_1 - u_2 - ... - u_k
    while there exits u adjacent to u_1 not in the path:
        path = u - path
    while there exits v adjacent to u_k not in the path:
        path = path - v
    return path

// Find the cycle consisting of vertex v_1, v_2, ..., v_n.
// Precondition: all the adjacent vertex of v_1 and v_n are in the
    path.
def path_to_circle(v_1 - v_2 - ... -v_n):
    if v_n - v_1 in E:
        cycle = v_1 v_2 v_3 ... v_n v_1
        return cycle

    find v_i s.t. v_1 - v_{i+1} in E and v_i - v_n in E.
    cycle = v_1 - v_{i+1} - v_{i+2} - ... - v_{n-1} - v_n  - v_i -
    v_{i-1} - ... - v_3 - v_2 - v_1
    return cycle

// Extend cycle to a longer path.
// Precondition: cycle is not Hamiltonian.
def cycle_to_path(v_1 - v_2 - ... - v_n - v_1):
    find v_i s.t. there exits u - v_i in E and u is not in cycle.
    return path = u - v_i - v_{i+1} - ... - v_n - v_1 - ... v_{i-1}

// Find the Hamiltonian cycle in G = (V, E)
def find_hamiltonian_cycle(G = (V, E)):
    random pick path P = u - v in E
    P = path_extension(P)
    while length of P < N:
        C = path_to_cycle(P)
        P = cycle_to_path(C)
        P = path_extension(P)
    return path_to_cycle(P)
```

## 3 Reference

[1] http://faculty.wwu.edu/sarkara/dirac.pdf

[2] https://math.stackexchange.com/questions/13959/if-a-1-meter-rope-is-cut-at-two-uniformly-randomly-chosen-points-what-is-the-av