Qishi quiz 2

Group 2

1. Problem 1

χ^2 test is any statistical hypothesis test in which the sampling distribution of the test statistic is a χ^2 distribution when the null hypothesis is true. It can be used to test the residuals distribution of model fitting.

The χ^2 test static is :

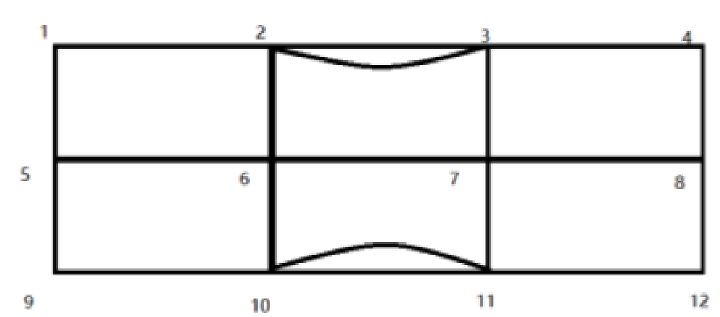
$$\chi^2 = \sum_{i=1}^{n} \frac{(O_i - E_i)^2}{E_i}$$

Where n is the number of observation, O_i is the observed frequency of outcome i. E_i is the expected frequency of outcome i. Once we get χ^2 we can calculate the corresponding p-value of χ^2 distribution with dof $k = n - q - 1$.

If p-value is smaller than certain conventional criteria say 0.05, we can reject the null hypothesis(normal distribution).

2. Problem 2

This is a Chinese postman problem. For a 2 by 3 grid we have 6 vertices with odd order which are 2,3,5,8,10,11. We can make it a semi Eulerian graph by connecting 2-3 and 10-11 directly. Now we can start from 5 and end at 8 to traverse all edges of the new graph. The length of the path will be 17+2=19.



3. Problem 3

X and Y are iid N(0,1).

$$\begin{aligned} P(X|X+Y>0) &= P(X \cap X+Y>0)/P(X+Y>0) \\ &= P(X)P(Y>-X) \\ &= \frac{\frac{1}{\sqrt{2\pi}}e^{-\frac{x^2}{2}} \int_{-x}^{\infty} \frac{1}{\sqrt{2\pi}}e^{-\frac{y^2}{2}}dy}{0.5} \\ &= \sqrt{\frac{2}{\pi}}e^{-\frac{x^2}{2}}(1-\Phi(-x)) \\ &= \sqrt{\frac{2}{\pi}}e^{-\frac{x^2}{2}}\Phi(x) \end{aligned}$$

4. Problem 4

(1) If keep rolling for once, the payoff is 0 with probability 1/6 and is $35+i$ for $i = 2, ...6$ with probability 1/6 respectively. In this case the expected payoff is $\frac{1}{6} \cdot \sum_{i=2}^{6}(35+i) = 32.5$. Since by rolling once, the maxmimum sum we can get is $35 + 6 = 41$ and $41 + 6 = 47 < 7^2 = 49$, we could still keep rolling without worrying about to get a square. After rolling twice the payoff become $32.5 + 3.5 = 36$ which is geater than 35. Hence we should keep rolling if the sum is 35.

(2) If we get 2, then we could continue rolling at least twice and in this case the most possible number to get is 7+2=9. Fo the other cases $3, 4, 5, 6$ we get the same results. Since the probability to get $2, 3, 4, 5, 6$ are the same, the most possible number is $35 + 9 = 44$.

(3) Suppose we arrive at $n^2 - 6 \leq k \leq n^2$, then if we stop we at least get $n^2 - 6$. If we continue we could nearly get $(n + 1)^2 - 1$ with probability at most 5/6. So we stop when we have $n^2 - 6 > 5/6(n^2 - 2n)$ which implies $n \geq 13$. So we stop between 163 and 169.

5. Problem 5

The two breaking points x and y are uniformlly distributed on $[0, 1]$. Let c_1, c_2, c_3 be the length of each piece. Then $c_1+c_2+c_2 < 1$ and the area of volumn of it is $1/3! = 1/6$. So the joint distribution of them is 1/6. Let $A = min(X, Y), B = max(X, Y)$ and $C = min(A, 1-B, B-A)$. Let $F_C(a)$ be the cdf of C. Then $1-F_C(a) = Prob(C \geq a) = P(a \leq X \leq 1, a \leq Y \leq 1-a, |X-Y| \geq a) = 1-(1-3a)^2$. So the pdf of C is $f_C(a) = F'_C(a) = 6(1-3a)$. So the expected length of the smallest piece is $\int_0^{1/3} 6a(1-3a)da = 1/9$.

Similarly we can compute the expected length of the longest piece. Let $D = max(A, 1-B, B-a)$. Then the cdf for D is $F_D(a) = Prob(D \leq a) = Prob(A \leq a, 1-B \leq a, B-A \leq a)$. So $F_D(a) = (3a-1)^2$ if $1/3 \leq a \leq 1/2$ and $F_D(a) = 1-3(1-a)^2$ if $1/2 \leq a \leq 1$. So the pdf of D is $f_D(a) = 6(3a-1)$ if $1/3 \leq a \leq 1/2$ and $f_D(a) = 6(1-a)$ if $1/2 \leq a \leq 1$. So the expected length of the longest piece is $\int_{1/3}^{1/2} 6a(3a-1)da + \int_{1/2}^{1} 6a(1-a)da = 11/18$.

Hence, the expected length of the middle-sized piece is $1-1/9-11/18 = 5/18$.

6. Problem 6

(1) For $i = 1, ..., N$, define random variable $I_i = 1$ if person i choose his/her own hat and $I_i = 0$ otherwise. Then $E(I_i) = Prob(I_i = 1) = 1/N$. Then $Y = \sum_i I_i$. So $E(Y) = E(\sum_i I_i) = \sum_i E(I_i) = N \cdot 1/N = 1$.

(2) We have $E(Y^2) = E((\sum_i I_i)\cdot(\sum_j I_j) = E(\sum_{i,j=1} I_i I_j) = sum_{i,j=1} E(I_i I_j) = \sum_{i,j=1,i\neq j} Prob(I_i = 1, I_j = 1) + \sum_i Prob(I_i = 1) = \sum_{i,j=1,i\neq j} \frac{1}{N(N-1)} + \sum_i Prob(I_i = 1)\frac{1}{N} = 1 + 1 = 2$. So the variance of Y is $var(Y) = E(Y^2) - (E(Y))^2 = 2 - 1^2 = 1$.

(3) Let X_N be the random variable counting the nuber of people picking their own hats in the first round. Since from (1) we know that on average there are one person who picks his/her own hat, we might guess that be the number of rounds that are run are $R(N) = N$. In fact, it is easy to see that it is true for $N = 1$. For general N, we could prove it by induction. In fact, suppose that it holds up to $N - 1$. Then

$$\begin{aligned}
E(R(N)) &= \sum_{i=0}^{N} E(R(N)|X_N = i)Prob(X_N = i) \\
&= \sum_{i=0}^{N} (1 + E(R(N-i))Prob(X_N = i) \\
&= \sum_{i=0}^{N} Prob(X_N = i) + E(R(N))Prob(X_N = 0) + \sum_{i=1}^{N} (N - i)Prob(X_N = i) \\
&= 1 + E(R(N))Prob(X_N = 0) + N(1 - Prob(X_N = 0)) - E(X_N) \\
&= E(R(N)))Prob(X_N = 0) + N(1 - Prob(X_N = 0)),
\end{aligned}$$

where the third equality follows from induction hypothesis and the last equality follows from (1) that $E(X_N) = 1$. So $E(R(N))(1 - Prob(X_N = 0)) = N(1 - Prob(X_N = 0))$ which implies $E(R(N)) = N$ since $P(X_N = 0) \neq 1$. This finishes the induction process.

(4) Conditioning on X_N gives

$$\begin{aligned}
E(S(N)) &= \sum_{i=0}^{N} E(S(N)|X_N = i)Prob(X_N = i) \\
&= \sum_{i=0}^{N} (N + E(S(N-i))Prob(X_N = i) \\
&= N + \sum_{i=0}^{N} E(S(N-i)Prob(X_N = i).
\end{aligned}$$

This implies

$$E(S(N)) = N + E(S(N - X_N)).$$

Because of this relation, it is natural to assume that $E(S(N))$ is a polynomial function of N. But if there is exactly only one match in each round, we would have $\sum_{i=1}^{N} i = N(N+1)/2$ selections. So it would be natural to assume $E(S(N))$ is a quadratic polynomial function of N. So suppose that $E(S(N)) = aN^2 + bN$ and plug this into the above equation. We get $aN^2 + bN = N + E(a(N - X_N)^2 + b(N - X_N)]$. Since $E(X_N) = var(X_N) = 1$, we have $aN^2 + bN = aN^2 + (b - 2a + 1)N + 2a - b$. So $a = 1/2$ and $b = 1$. This implies

$E(S(N)) = N + N^2/2$. To prove it rigoriously, we could use induction. In the case of $N = 2$, it is true because the number of rounds is geometric distribution with parameter $1/2$. Suppose that it is true up to $N-1$. Then

$$\begin{aligned}
E(S(N)) &= N + E(S(N))Prob(X_N = 0) + \sum_{i=1}^{N} E(S(N-i)Prob(X_N = i) \\
&= N + E(S(N))Prob(X_N = 0) + \sum_{i=1}^{N}((N-i)^2/2 + (N-i))Prob(X_N = i) \\
&= N + E(S(N))Prob(X_N = 0) + (N^2/2 + N)(1 - Prob(X_N = 0)) - (N+1)E(X_N) \\
&\quad + E(X_N^2)/2
\end{aligned}$$

where the second equality follows from induction hypoethesis. Plugging $E(X_N) = 1, E(X_N^2) = 2$ into the above equation gives $E(S(N)) = N + N^2/2$, as desired.

(5) Let H_i be the randon variable counting the number of hats chosen by person i. Then $\sum_i H_i = S(N)$. Hence $E(\sum_i H_i) = E(S(N))$. By symmetry, H_i has the same mean, so $E(H_i) = S(N)/N = 1 + n/2$. So the expected number of false selections made by one of the N people is $E(H_i - 1) = n/2$.

7. Problem 7

The upper bound is 1 and the lower bound is 0.02. Let ι be a column vector every element of which is 1. Let x be a $n \times k$ matrix and let $S(x)$ be the subspace generated by the column vectors of x. Then $P_x = x(x^T x)^{-1} x^T$ is the projection matrix projecting onto $S(x)$ and $M_x = I - P_x = I - x(x^T x)^{-1} x^T$ is the annihilator space projecting onto the orthogonal complement subspace $S^{\perp}(X)$. Then the (centered) R^2 is

$$R^2 = \frac{\|P_x M_\iota y\|^2}{\|M_\iota y\|^2},$$

where $M_\iota y = (I - P_\iota)y = y - \iota(\iota^T\iota)^{-1}\iota^T y = y - \bar{y}\iota$ is the centered version of y and $\bar{y} = \frac{1}{n}\sum_{t=1}^{n} y_t$ is the mean of y. Here we assume that ι is contained in the span $S(x)$ of the regressors. So in this problem $S(x)$ is $S(x_1, \iota)$ or $S(x_2, \iota)$ or $S(x_1, x_2, \iota)$.

Upper bound: If $M_\iota y \in S(x_1, x_2)$, the vector space generated by the column vectors of x_1 and x_2, then $\|P_{(x_1,x_2)} M_\iota y\|^2 = \|M_\iota y\|^2 \Rightarrow R^2 = 1$. Since $R^2 \leq 1$, the upper bound of R^2 is 1. Geometrically, this means that $M_\iota y$ belongs to the subspace span by (the column vectors of) x_1 and x_2, so the model is perfectly fit and $R^2 = 1$.

Lower bound: First we show that $R^2_{M_3} \geq R^2_{M_1}, R^2_{M_2}$, where we use subscripts to differentiate R^2 from different models. Without loss of generality, we just need to prove the case for M_2. In fact, since both $P_{x_1,x_2,\iota}$ and $P_{x_2,\iota}$ are symmetric, so is $P_{x_1,x_2,\iota} - P_{x_2,\iota}$. Moreover, since $S(x_2, \iota) \subset S(x_1, x_2, \iota)$, we have $P_{x_1,x_2,\iota} P_{x_2,\iota} = P_{x_2,\iota} P_{x_1,x_2,\iota} = P_{x_2,\iota}$. So $(P_{x_1,x_2,\iota} - P_{x_2,\iota})^2 = P^2_{x_1,\iota,x_2,\iota} - P_{x_1,\iota,x_2,\iota} P_{x_2,\iota} - P_{x_2,\iota} P_{x_1,x_2,\iota} + P^2_{x_2,\iota} = P_{x_1,x_2,\iota} - P_{x_2,\iota}$ as both of $P_{x_1,x_2,\iota}$ and $P_{x_2,\iota}$ are idempotents. So $P_{x_1,x_2,\iota} - P_{x_2,\iota}$ is an orthogonal projection matrix. Hence $\|P_{x_1,x_2,\iota} M_\iota y\|^2 - \|P_{x_2,\iota} M_\iota y\|^2 = (M_\iota y)^T (P_{x_1,x_2,\iota} - P_{x_2,\iota}) M_\iota y = \|(P_{x_1,x_2,\iota} - P_{x_2,\iota}) M_\iota y\|^2 \geq 0 \Rightarrow \frac{\|P_{x_1,x_2,\iota} M_\iota y\|^2}{\|M_\iota y\|^2} \geq \frac{\|P_{x_2,\iota} M_\iota y\|^2}{\|M_\iota y\|^2}$. So $R^2_{M_3} \geq R^2_{M_2} = 0.02$. But this equality can be attained. In fact, if $\|P_{x_1,x_2,\iota} M_\iota y\|^2 = \|P_{x_2,\iota} M_\iota y\|^2$, then $R^2_{M_3} = 0.02$. Geometrically, to do the OLS for model M_3, it amounts to find a vector v inside $S(x_1, x_2, \iota)$ such that $\|M_\iota y - v\|^2$ is minimized. In our case that $R^2_{M_3} = 0.02$ this minimization is exactly achieved by $P_{x_2,\iota} M_\iota y$.

So the range of R^2 of Model 3 is $[0.02, 1]$.

8. Problem 8

$1/n$ is a rational number so it can be written as binary number with finite sum: $1/n = \sum_{i=1}^{m} p_i 2^{-i}$ for $p_i = 0, 1$. Then the function biasedCoin() can be implement (by using the function fairCoin()) in the following ways:

```
int baisedCoin(int[] p, int m) {
    for (int i = 0; i < m; i++) {
        int s = fairCoin();
        if (s < p[i]) {
            return 1;
        }
        else if (s > p[i]) {
            return 0;
        }
     }
     return 0;
}
```

The algorithm is explained as follows: when we coss a fair coin, the heads count as 1 and the tails count as 0. Let $s_i \in \{0, 1\}$ be the result of i-th toss starting with $i = 1$. After each toss, we compare s_i with p_i. If $s_i < p_i$ (resp. $s_i > p_i$), the baisedCoin() return a head (resp. tail) and the tossing stops. If $s_i = p_i$, we continue tossing the fair coin. If we end up with $s_n = p_n$, baisedCoin() return a tail. This will give a function which return heads with probabilty $1/n$.

9. Problem 9

There's a confusion that if the man gets on a weak bridge and fall, will it count as one cross? We won' count that in in our solution.

There are different understanding of this problem. If "miraculously fixed" means all bridges are fixed strong, then we have Xiang Ni's answer below:

Let A_i be the event that the i-th bridge is weak and the previus $i-1$ bridges are all strong (i.e i is the first weak bridge), then A_i are disjoint and $P(\cup_{i=1}^{10} A_i) = 1$, where A_{10} means that all the bridges are strong. Moreover, we have $Prob(A_i) = \frac{1}{2^i}$ for $i \leq 9$ and $Prob(A_{10}) = \frac{1}{2^9}$. Define random variables I_i as $I_i = 1$ on A_i and $I_i = 0$, otherwise. Let X be the random variable counting how many brdges the man has to cross. Then $X|_{A_i} = i - 1 + 9 = i + 8$ for $i \leq 9$ and $X|_{A_{10}} = 9$. So $E(X) = \sum_{i=1}^{9}(i + 8) \cdot \frac{1}{2^i} + 9 \cdot \frac{1}{2^9} = 10 - \frac{10}{2^9}$. So to arrive the 10th island, on average the man has to cross $10 - \frac{10}{2^9}$ bridges.

If "miraculously fixed" means all bridges are "shuffled", including the broken bridge can now be strong or weak, we have Tianmu Xin's answer:

Use E_i to denote the expected number of bridges one need to cross from ith island to 10th island. Then

$$E_i = \frac{1}{2}(1 + E_{i+1}) + \frac{1}{2}E_1$$

The first term means there is 1/2 probability that one will make a cross from ith island to island i+1 by crossing one bridge and 1/2 probability of starting over from 1st island. And we know that $E_10 = 0$. We can use E_10 to express E_9 and use E_9 to express E_8, so on so forth we can eventually express E_1 with E_10 as

$$E_1 = \sum_{i=1}^{8} \frac{1}{2^i} + \frac{1}{2^9}(1 + E_{10}) + \sum_{i=1}^{9} \frac{1}{2^i}E_1$$

Solve this we can get

$$E_1 = 2^9 - 1$$

If "miraculously fixed" means only the broken bridge is fixed and is strong now, here is Cong Chen's answer: The equation we have should be:

$$E_i = \frac{1}{2}(1 + E_{i+1}) + \frac{1}{2}(i + E_{i+1})$$

And this gives us $E_1 = 27$.

10. Problem 10

const before * means the value the pointer points to is constant.

const after * means the address the pointer points to cannot be changed.

const at the end of class method declaration means this method cannot change any class members.

Reference: `http://duramecho.com/ComputerInformation/WhyHowCppConst.html`

11. Problem 11

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode(int x) : val(x), next(NULL) {}
 * };
 */
class Solution {
public:
    void deleteNode(ListNode* node) {
        node->val = node->next->val;
        node->next = node->next->next;
    }
```

```
};
```

12. Problem 12

The .hh file for the matrix class is as follows:

```
class Matrix {

private:
    int numRows;
    int numCols;
    // matriarray to store the data in Matrix
    int *matrixarray;

public:
    // Constructors
    Matrix();   // default constructor
    Matrix(int sizeRow, int sizeCol);
    Matrix(Matrix &a); // copy constructor

    // Destructor
    ~Matrix();

    // Mutator methods
    void setelem(int row, int col, int elem);
    void add (Matrix &a);
    void subtract (Matrix &a);
    bool equals (Matrix &a);

    // Accessor methods
    int getrows();
    int getcols();
    int getelem(int row, int col);
};
```

The .cpp file for the matrix class is as follows:

```
#include "Matrix.hh"
#include <cassert>
using namespace std;
```

```
// The default constructor creates a 0 by 0 matrix.
Matrix::Matrix() {
    numRows = 0;
    numCols = 0;
    matrixarray = new int[0];
}


// Copy constructor
Matrix::Matrix(Matrix &a) {
    numRows = a.getrows();
    numCols = a.getcols();
    matrixarray = new int[numRows * numCols];
    for (int i = 0; i < numRows; i++) {
        for (int j = 0; j < numCols; j++) {
            matrixarray[i * numCols + j] =
            a.matrixarray[i * numCols + j];
        }
    }
}


// Initialize matrix of size sizeRow by sizeCol
Matrix::Matrix (int sizeRow, int sizeCol) {
    matrixarray = new int[sizeRow * sizeCol];
    numRows = sizeRow;
    numCols = sizeCol;
    for (int i = 0; i < sizeRow; i++) {
        for (int j = 0; j < sizeCol; j++) {
            matrixarray[i * sizeCol + j] = 0;
        }
    }
}


// Get the rows tall of the matrix
int Matrix::getrows () {
    return numRows;
}
```

```
// Get the columns wide of the matrix
int Matrix::getcols () {
    return numCols;
}


// Set the element at row-th row, col-th column as elem
void Matrix::setelem (int row, int col, int elem) {
    assert ((row >= 0) && (row < numRows));
    assert ((col >= 0) && (col < numCols));
    matrixarray[row * numCols + col] = elem;
}


// Get the element at row-th row, col-th column
int Matrix::getelem (int row, int col) {
    assert ((row >= 0) && (row < numRows));
    assert ((col >= 0) && (col < numCols));
    return matrixarray[row * numCols + col];
}


// Add two matrices
void Matrix::add (Matrix &a) {
    assert(numRows == a.getrows());
    assert(numCols == a.getcols());
    for (int i = 0; i < numRows; i++) {
        for (int j = 0; j < numCols; j++) {
            matrixarray[i * numCols + j] =
                matrixarray[i * numCols + j]
                + a.matrixarray[i * numCols + j];
        }
    }
}


// Subtract a from another matrix
void Matrix::subtract (Matrix &a) {
    assert(numRows == a.getrows());
    assert(numCols == a.getcols());
    for (int i = 0; i < numRows; i++) {
```

```
        for (int j = 0; j < numCols; j++) {
            matrixarray[i * numCols + j] =
                matrixarray[i * numCols + j]
                 - a.matrixarray[i * numCols + j];
        }
    }
}


// Test for equality
bool Matrix::equals (Matrix &a) {
    if (numRows != a.getrows() || numCols != a.getcols()){
        return false;
    }
    for (int i = 0; i < numRows; i++) {
        for (int j = 0; j < numCols; j++) {
            if ((matrixarray[i * numCols + j])
                 != (a.matrixarray[i * numCols + j])) {
                     return false;
            }
        }
    }
    return true;
}


// Release the memory
Matrix::~Matrix() {
    delete[] matrixarray;
}
```

13. Problem 13

No, constructor of a class CANNOT be virtual. Here is a quote from Bjarne Stroustrup's C++ Style and Technique FAQ: "*virtual call is a mechanism to get work done given partial information. In particular, "virtual" allows us to call a function knowing only any interfaces and not the exact type of the object. To create an object you need complete information. In particular, you need to know the exact type of what you want to create. Consequently, a "call to a constructor" cannot be virtual.*"

When virtual function is called, compiler looks for the correct function in the vtable. Vtable is not constructed yet when constructor is firstly called so constructor cannot be virtual.

```cpp
// The header file which defines base class
// Shape and derived class Circle

class Shape {
public:
  virtual ~Shape() { }              // A virtual destructor
  virtual void draw() = 0;          // A pure virtual function
  virtual void move() = 0;
  // ...
  virtual Shape* clone()  const = 0;// Uses the copy constructor
  virtual Shape* create() const = 0;// Uses the default constructor
};

class Circle : public Shape {
public:
  Circle* clone()  const;// Covariant Return Types; see below
  Circle* create() const;// Covariant Return Types; see below
  // ...
};


Circle* Circle::clone()  const { return new Circle(*this); }
Circle* Circle::create() const { return new Circle();      }
```

Reference of virtual constructor implementation:
http://www.geeksforgeeks.org/advanced-c-virtual-constructor/

14. Problem 14

Yes, but the non-virtual function was not redefined in any derived class.
Reference: https://isocpp.org/wiki/faq/strange-inheritance#calling-virtuals-from-base

15. Problem 15

Any single character is a palindrome. If the character after this character is the same, then they form a two char palindrome. If the characters before and after this palindrome are the same then we got a four char palindrome. We need a table to record whether the substring from i to j is a palindrome. $T[i, i]$ should be all true. Then check the consecutive char and see if T[i,i+1] is true. After that we need to go over every substring $T[i, j]$ and check if it is palindrome and if the chars before and after it are the same, if so we update the $T[i-1, j+1]$. Then we need to expand this to next possible length of the palindrome, Therefore this will have $O(N^2)$ time complexity.

Cpp code provided by Tianmu Xin:

```
#include <string>
#include <cstdlib>
#include <iostream>
#include <vector>
using namespace std;

string longestPalindrome(string s)
    {
        int n = s.length();
        int longestBegin = 0;
        int maxLen = 1;
        vector <vector <bool> > table(n, vector<bool>(n, false));
        for (int i = 0; i < n; i++)
        {
            table[i][i] = true;
         }
        for (int i = 0; i < n-1; i++)
        {
            if (s[i] == s[i+1])
            {
            table[i][i+1] = true;
            longestBegin = i;
            maxLen = 2;
            }
        }
        for (int len = 3; len <= n; len++)
        {
            for (int i = 0; i < n-len+1; i++)
            {
                int j = i+len-1;
                if (s[i] == s[j] && table[i+1][j-1])
                {
                    table[i][j] = true;
                    longestBegin = i;
                    maxLen = len;
                }
            }
        }
        return s.substr(longestBegin, maxLen);
    }
```

Python code provided by Xiang Ni:

```
def expandAroundCenter(str, c1, c2):
    left = c1
    right = c2
    length = len(str)

    while(left >= 0 and right <=length-1 and str[left] == str[right]):
        left-=1
        right+=1

    return str[left+1:right]

def longestPalindrome(str):
    n = len(str)
    if (n==0): return ""

    longest = str[0:1]
    for i in range(n):
        p1 = expandAroundCenter(str,i,i)
        if (len(p1) > len(longest)): longest = p1

        p2 = expandAroundCenter(str,i,i+1)
        if (len(p2) > len(longest)): longest = p2

    return longest
```

16. Problem 16

We can first reverse the sentence and then reverse each words. The time complexity is $O(N^2)$.

```
# wu chi de python code

def revSentence(sentence):
    s = sentence.split()
    s.reverse()
    print "reversed sentence is:"
    for i in range(len(s)):
        print s[i],
```

17. Problem 17

Use dynamic programming. For each i, find the max profits profit1[i] and profits2[i] with one transection from prices[0] to prices[i] and from prices[i+1] to prices[n-1] respectively. Find the maximal profit1[i]+profit2[i] from 0 to n-1. The time complexity is $O(N)$.

```
import numpy as np

def maxProfit(prices):
    prices=np.array(prices)
    length=prices.size
    if length < 2: return 0
    profit1=np.zeros(length)
    profit2=np.zeros(length)

    # find the maximal profits with one transaction
    #  in sub list prices[0],...,price[i]
    minP=prices[0]
    for i in range(1,length):
        minP=min(minP, prices[i])
        profit1[i]=max(profit1[i-1],prices[i]-minP)

    # find the maximal profits with one transaction
    #  in sub list prices[i+1],...,price[n-1]
    maxP=prices[length-1]
    for i in range(length-2,1,-1):
        maxP=max(maxP,prices[i])
        profit2[i]=max(profit2[i+1],maxP-prices[i])

    res=(profit1+profit2).max()

    return res
```

18. Problem 18

Form a graph such that each node represent each person and if two persons are friends, then there is an edge connecting the corresponding two nodes. Then this problem actually asks to find a Hamiltonian Path in this graph, that is a path that visits each vertex exactly once. We could use back-tracking algorithm to find such path. The input is a N by N matrix graph[N][N] which is the adjacency matrix representation of the graph. A value graph[i][j] is 1 if there is a direct edge from i to j (i.e. person i and person j are friends), otherwise graph[i][j] is 0.

```
#function to check if the vertex v can be
# added at index 'pos'
def isSafe(v,graph,path,pos):
 #Check if this vertex is an adjacent vertex of the
 # previously added vertex
    if (graph[path[pos-1]][v] == 0):
        return False

# Check if the vertex has already been included.
    for i in range(pos):
        if (path[i]==v):
            return False

    return True



# A recursive function to solve hamiltonian cycle
#  problem
def hamPath(graph, path, pos):
    if (pos = N):
# if there is an edge from the last included vertex
# to the first node
        if (graph[path[pos-1]][path[0]] == 1):
            return True
        else:

            return False
# Try different vertices as a next candidate in
# Hamiltonian Cycle.
    for v in range(1,V):
        if (isSafe(v,graph,path,pos):
            path[pos] = v
            if (hamPath(graph,path,pos+1) == True):
                return True

            path[pos] = -1

    return False
```

```
#This function solves the Hamiltonian Cycle problem
#  using Backtracking.

def BookProb(graph, N)
    path = N*[-1]

    path[0] = 0
    if (hamPath(graph,path,1) == False):
        print "No solution"
        return False

    for i in range(N):
         print(path[i])
         print(" ")

    return True
```