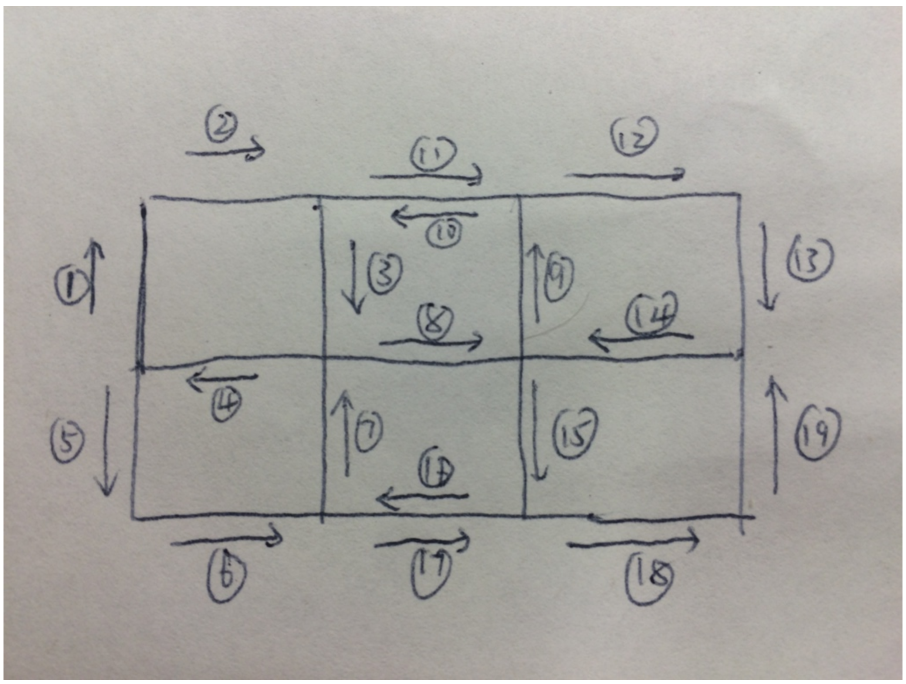
1 Math/Stat.

Q1.

Strictly speaking,χ^2 test is any statistical hypothesis test in which the sampling distribution of the test statistic is a chi-square distribution when the null hypothesis is true. In most cases, test statistic arises from an assumption of independent normally distributed data, and a chi-squared test can then be used to reject the hypothesis that the data are independent.

Q2.

The answer is 19. We'll first show that the shortest path cannot be either 17 or 18. If it is 17 then every edge is covered exactly once. In this situation, every vertex must have even degree except for the starting and end point. However, there are 6 vertices in the graph with odd degree – a conflict. If the shortest path is 18, then there is exactly one edge that is repeated and only repeated once. By replicating one edge in the graph, there will still be at least 4 vertices in the graph with odd degree. And by replicating, the 18 edges are covered exactly once. But this is impossible based on the same reasoning. Now we list one path with shortest path 19.



Q3.

X, Y are iid N(0,1), then for any t in R,

P(X<=t|X+Y>0)= P(X<=t, X+Y>0) /P(X+Y>0)

For the numerator,

$$
\begin{aligned}
P(x \le t, X+Y>0) = P(x \le t, Y > -X) &= \int_{-\infty}^{t} \left[\phi(t) \int_{-x}^{\infty} \phi(y)dy \right] dx \\
&= \int_{-\infty}^{t} \left[\phi(t) \int_{-\infty}^{x} \phi(y)dy \right] dx \\
&= \int_{-\infty}^{t} \left[\phi(t)\Phi(x) \right] dx,\ where\ \phi = [\Phi]'. \\
&= 1/2\Phi^2(x)|_{-\infty}^{t} = 1/2\Phi^2(t)
\end{aligned}
$$

For the denominator, since X+Y are N(0,2), P(X+Y>0)=1/2
Then $P(X \le t | X+Y>0 = \Phi^2(t)$ where $\Phi(t)$ is the CDF of standard normal distribution

Q4.
(1) Whether to keep rolling or stop depends on whether the expected gain of keep rolling is larger than zero. If next is 1, the gain is -35, otherwise we could safely reach 43 or above, corresponding to a gain of 8 or larger. Hence the expected gain of keep rolling is larger than -35/6 + 8 * 5/ 6 >0. Therefore, it's better to keep rolling.

(2) The 1st rolling could be 2, 3, 4, 5, 6. Then we consider the most likely value when we stop after the first rolling.
If the 1st rolling is 2, we continue with at least another two rollings, and the most probable outcome of two rollings' sum is 7 (= 1+6 = 2+5 = 3+4 = 4+3 = 5+2 = 6+1, 6 cases reach 7), giving a total stop value at 35+2+7 = 44.
if the 1st rolling is 3, we would stop if 2nd rolling is 6, with total value reaching 44. Otherwise, we continue with at least another two rolling, with the most probable outcome at 7 or 6. Hence, the most likely ending value is still 44.
if the 1st rolling is 4, we would stop if 2nd rolling is 5 or 6, with total value reaching 44 or 45. Otherwise, we continue with another two rollings, which gives the most probable value at 7 or 6 or 5. Hence the most likely ending value is 44 or 45. By the same token, any other case all have the most probable ending value at 44. Therefore the 44 is the most probable outcome.

Q5. The solution is the same as it is in problem 43 The Broken Bar in the book Fifty Challenging Problems in Probability and Solutions.

Q6.

There are N! possible permutation of the N persons 1,2,…,N with equal probability, hence each permutation has probability 1/N!. Denote X_i as the hat that the i^{th} person gets.

$P_N (X_1=1, X_2=2, …, X_{N-2}=N-2, X_{N-1}=N-1, X_N=N) = 1/N!$

$P_N (X_1=1, X_2=2, …, X_{N-2}=N-2 ,X_{N-1}=N-1) =1/N!$

$P_N (X_1=1, X_2=2, …, X_{N-2}=N-2) = 2!/N!$

……

$P_N (X_1=1) = (N-1)!/N!$

(1) $E(Y) = E (sum(i=1 \text{ to } N) 1_{\{Xi=i\}}) = N*P(X_i=i) =N* (N-1)!/N!=1$

(2) $E(Y^2)= E(sum(i=1 \text{ to } N) 1_{\{Xi=i\}})^2$

$= sum(i=1 \text{ to } N) 1_{\{Xi=i\}} + 2*sum(1<=i<j<=N) 1_{\{Xi=i, Xj=j\}}$

$= E(Y) +2 * C(N,2) * P(X_i=i, X_j=j) =1+N(N-1) * (N-2)!/N!=1+1=2$

Then Var(Y) =E(Y^2) –[E(Y)]^2 =1.

(3)

Since $R_N = Y(N) + R_{N-Y(N)}$

$E(R_N) = E(Y(N)) + E(R_{N-Y(N)})$.

$E(R_N) = 1+ P(Y(N)=0) * E(R_N) + P(Y(N)=1) * E(R_{N-1}) +…… +P(Y(N)=N-1) * E(R_1)$

We want to get P(Y(N)=k) ==denoted as== $P_N(Y=k)$.

$P_N (Y=0) = 1 - P_N (Y>=1)$.

Since $P_N (Y>=1) = P_N (\{X_1=1\} U \{X_2=2\} U \{X_3=3\} U…U \{X_{N-1}=N-1\}… U \{X_N=N\})$

$= Sum(i=1 \text{ to } N) P_N (\{X_i=si\}) –Sum(1<=i<j<=n) P_N (\{X_i=i\}*\{X_j=j\}) +Sum(1<=i<j<k<=n)P_N (\{X_i=i\}*\{X_j=j\}*\{X_k=k\})-… +(-1)^N Sum(1<=i_1<i_2<…<i_{N-1}<=N) P_N (\{X_{i1}=i_1\}*\{X_{i2}=i_2\}*….*\{X_{iN-1}=i_{N-1}\}) +(-1)^(N+1) Sum(1<=i_1<i_2<…<i_N<=N) P_N (\{X_{i1}=i_1\}*\{X_{i2}=i_2\}*….*\{X_{iN-1}=i_{N-1}\}*\{X_{iN}=i_N\})$

$=C(N,1)*P_N (X_1=1)-C(N,2)*P_N (X_1=1,X_2=2)+C(N,3)*P_N (X_1=1,X_2=2,X_3=3)-… +(-1)^N*C(N,N-1)*P_N (X_1=1, X_2=2, …, X_{N-2}=N-2 ,X_{N-1}=N-1) +(-1)^(N+1)* C(N,N)*P_N (X_1=1, X_2=2, …, X_{N-2}=N-2, X_{N-1}=N-1, X_N=N)$

=C(N,1)* (N-1)!/N! –C(N,2)* (N-2)!/N!+C(N,3)* (N-3)!/N! -…+(-1)^N*C(N,N-1)*1!/N! +(-1)^(N+1)* C(N,N)*0!/N!

=1/1!-1/2!+1/3!-…. +(-1)^N*1/(N-1)! +(-1)^(N+1)*1/N!

We have $P_N (Y=0) = 1 - P_N (Y>=1) =$ 1/2!-1/3!+…. +(-1)^(N-1)*1/(N-1)! +(-1)^N*1/N!

Consider Y=k (1<=k<=N-2), which could be thought of as the first k persons out of N can take their own hats, while for the other N-k, none can get his own hat.

$P_N (Y=k) = 1/k! * P_{N-k} (Y=0) =1/k! *[1/2!-1/3!+…. +(-1)^(N-k)*1/(N-k)!]$

For k>=N-1, we can easily get $P_N (Y=N-1)=0$, P(Y=N)=1/N!

So $E(R_N) = 1+ (1/2!-1/3!+.... +(-1)^{(N-1)}*1/(N-1)! +(-1)^N*1/N!) * E(R_N)$

$+1/1!*(1/2!-1/3!+.... +(-1)^{(N-1)}*1/(N-1)!) * E(R_{N-1})$

$+1/2!*(1/2!-1/3!+.... +(-1)^{(N-2)}*1/(N-2)!) * E(R_{N-2})$

+......

$+1/k!*(1/2!-1/3!+.... +(-1)^{(N-k)}*1/(N-k)!) * E(R_{N-k})$

+......

$+1/(N-3)!*(1/2!-1/3!) * E(R_3)$

$+1/(N-2)!*(1/2!) * E(R_2)$

Since $E(R_1)=1$,

$E(R_2)=1+1/2!*E(R_2) \Rightarrow E(R_2)=2$.

$E(R_3)=1+(1/2!-1/3!)*E(R_3)+1/2!*E(R_2) \Rightarrow E(R_3)=3$.

We try mathematical induction method. Assume $E(R_k)=k$ for all $k<=m$, then

$E(R_{m+1})= 1+(1/2!-1/3!+.... +(-1)^m*1/m! +(-1)^{(m+1)}*1/(m+1)!) * E(R_{m+1})$

$+1/1!*(1/2!-1/3!+.... +(-1)^m*1/m!) * m$

$+1/2!*(1/2!-1/3!+.... +(-1)^{(m-1)}*1/(m-1)!) * (m-1)$

+......

$+1/k!*(1/2!-1/3!+.... +(-1)^{(m+1-k)}*1/(m+1-k)!) * (m+1-k)$

+......

$+1/(m+1-3)!*(1/2!-1/3!) * 3$

$+1/(m+1-2)!*(1/2!) * 2$

Reorganize the equation, and by calculation, we get $E(R_{m+1}) =m+1$.

So $E(R_k)=k$ for all $k<=m+1$.

Then by mathematical induction method, we get $E(R_N)=N$ for all $N>=1$.

(4) Since $S_N = N + S_{N-Y(N)}$

$E(S_N) = N+ E(S_{N-Y(N)})$.

$E(S_N) = N+ P(Y(N)=0) * E(S_N) + P(Y(N)=1) * E(S_{N-1}) +...... +P(Y(N)=N-1) * E(S_1)$

Note that $E[S_N-N(N-1)/2] – E[S_{N-Y(N)}- (N-Y(N)) (N-Y(N)-1)/2]$

$=E[S_{N-} S_{N-Y(N)}] –N(N-1)/2 + N(N-1)/2 + ½ * E[Y^2(N) -2N*Y(N) +Y(N)]$

$=N+1/2* [1-2N+1]=1$

So $E[S_N-N(N-1)/2] = 1+ E[S_{N-Y(N)}- (N-Y(N)) (N-Y(N)-1)/2]$.

So $E[S_N-N(N-1)/2]$ has the same property as that of $E[R_N]$.

Then $E[S_N-N(N-1)/2] = E[R_N]=N$.

So $E[S_N]= N(N-1)/2+N = N(N+1)/2$.

(5) The expected number of false selections for each of the N persons

$1/N* E(S_N)-1 = 1/N* N(N+1)/2 – 1 = (N-1)/2$.

Thanks to example 1.3(a) and 1.5(e) in the book “Stochastic Processes” by S.M. Ross.

Qishi Quiz 2

I. Question 7

Consider linear regression of Y on feature X_1, X_2: Model1-(Y, X_2), $R^2 = 0.1$; Model2-(Y, X_2), $R^2 = 0.2$; Model3-(Y, X_1, X_2), calculate the range of R^2 of Model3.

A. Solution

In a univariate regression:

$$R^2 = \rho^2. \tag{1}$$

e.g., in a simple linear regression with one feature: $y = a + bx$, $R^2 = \rho^2_{x,y}$.
For Model1-(Y, X_1), we have

$$R^2 = 0.1 = \rho^2_{y,x_1} \tag{2}$$

For Model2-(Y, X_2), we have

$$R^2 = 0.2 = \rho^2_{y,x_2} \tag{3}$$

The correlation matrix of (Y, X_1, X_2) is positive semi-definite, where

$$Corr(Y, X_1, X_2) = \begin{pmatrix} 1 & \sqrt{0.1} & \sqrt{0.2} \\ \sqrt{0.1} & 1 & \rho_{1,2} \\ \sqrt{0.2} & \rho_{1,2} & 1 \end{pmatrix}. \tag{4}$$

Hence,

$$\rho_{1,2} - 0.2\sqrt{2}\rho_{1,2} - 0.7 \leq 0. \tag{5}$$

Thus, we have

$$-0.5\sqrt{2} \leq \rho_{1,2} \leq 0.7\sqrt{2}. \tag{6}$$

Since Y and X_1 are positively correlated and Y and X_2 are positively correlated, we have X_1 and X_2 are positively correlated, thus,

$$0 \leq \rho_{1,2} \leq 0.7\sqrt{2} \tag{7}$$

In regression $y = a + bx_1 + cx_2$, we have

$$R^2 = \frac{\rho^2_{y,1} + \rho^2_{y,2} - 2\rho_{y,1}\rho_{y,2}\rho_{1,2}}{1 - \rho^2_{1,2}}. \tag{8}$$

where $\rho_{y,1}$ is the correlation of y and x_1, $\rho_{y,2}$ is the correlation of y and x_2, and $\rho_{1,2}$ is the correlation of x_1 and x_2. Since we have $\rho^2_{y,1} = 0.1$ and $\rho^2_{y,2} = 0.2$, we have,

$$R^2 = \frac{0.3 - 0.2\sqrt{2}\rho_{1,2}}{1 - \rho^2_{1,2}}. \tag{9}$$

Let $R^2 = x$, we have

$$\rho_{1,2} = \frac{0.1\sqrt{2} \pm \sqrt{0.02 + x^2 - 0.3x}}{x}. \tag{10}$$

Since $x = R^2 \geq 0.2$, as $R_{y,x_2} = 0.2$, we have,

$$x^2 - 0.3x + 0.02 \geq 0. \tag{11}$$

From (6) and (10), we have

$$0 \leq \frac{0.1\sqrt{2} \pm \sqrt{0.02 + x^2 - 0.3x}}{x} \leq 0.7\sqrt{2}. \tag{12}$$

First, let us take a look at

$$0 \leq \frac{0.1\sqrt{2} \pm \sqrt{0.02 + x^2 - 0.3x}}{x}. \tag{13}$$

Since $x \geq 0$, we have for sure that

$$\frac{0.1\sqrt{2} + \sqrt{0.02 + x^2 - 0.3x}}{x} \geq 0. \tag{14}$$

From

$$0.1\sqrt{2} - \sqrt{0.02 + x^2 - 0.3x} \geq 0, \tag{15}$$

we can get

$$0 \leq x \leq 0.3 \tag{16}$$

Now let us take a look at

$$\frac{0.1\sqrt{2} \pm \sqrt{0.02 + x^2 - 0.3x}}{x} \leq 0.7\sqrt{2} \tag{17}$$

From

$$\frac{0.1\sqrt{2} + \sqrt{0.02 + x^2 - 0.3x}}{x} \leq 0.7\sqrt{2}, \tag{18}$$

we have

$$\sqrt{0.02 + x^2 - 0.3x} \leq 0.7\sqrt{2} - 0.1\sqrt{2}, \tag{19}$$

where the right hand side of (19) ≥ 0 since $x \geq 0.2$. Thus,

$$0 \leq x \leq 1. \tag{20}$$

From

$$\frac{0.1\sqrt{2} - \sqrt{0.02 + x^2 - 0.3x}}{x} \leq 0.7\sqrt{2}, \tag{21}$$

we have

$$-\sqrt{0.02 + x^2 - 0.3x} \leq 0.7\sqrt{2}x - 0.1\sqrt{2}, \tag{22}$$

where (22) is always true since $x \geq 0.2$ and thus $0.7\sqrt{2}x - 0.1\sqrt{2} \geq 0$. Combining (16) and (20), we have

$$0 \leq x \leq 0.3. \tag{23}$$

Q8.

We only have to generate numbers 0-(n-1) with equal probability. Let 2^k <= n-1 < 2^{k+1}, we toss the coin k times and use the result to make a k bit binary number (head as 1, tail as 0). If the number is >= n, discard this number and generate again until we have some number <n. This random number generator can produce numbers from 0 to n-1 with equal probability 1/n.

Q9.

Denote E_i as the expected number of acrossed bridges starting from island i.

Easy to obtain from island 1, we have:

$$E_1 = 1/2(1+E_2) + 1/4(2+E_2) + 1/8(3+E_2) + \ldots = 2 + E_2$$

On the other hand, we have:

$$E_2 = \frac{1}{2}E_1 + \frac{1}{2}(E_3+1) = \frac{1}{2}E_1 + \frac{1}{2}(\frac{1}{2}E_1 + \frac{1}{2}(E_4+1)) + \frac{1}{2} = (\frac{1}{2}+\frac{1}{4})E_1 + \frac{1}{4}E_4 + (\frac{1}{2}+\frac{1}{4})$$
$$= \ldots = 1 - \frac{1}{2^8} + (1 - \frac{1}{2^8})E_1$$

By solving above two equations, we have $E_1 = 3\times 2^8 - 1 = 767$

2 Programming.

Q10.const function

Input: reference to (a const pointer to a const integer) it is a reference to pointer

Output: const pointer to a const integer

Q11.

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

struct Node{
    double value;
    Node* next;
};  // struct is a class, and all of its variables are public


Node* deleteNode(Node* Node_i, Node* head);
void printList(Node* head);

int main()
{
    Node * node1 = new Node;
    Node * node2 = new Node;
    Node * node3 = new Node;

    node1->value = 1;
    node1->next = node2;
    node2->value = 2;
    node2->next = node3;
    node3->value = 3;
    node3->next = NULL;

    printList(node1);

    deleteNode(node2, node1);

    printList(node1);

    Node* new_head = deleteNode(node1, node1);
    printList(new_head);

    Node* new_head2 = deleteNode(node3, new_head);
    printList(new_head2);

    system("pause");
}

void printList(Node* head)
{
    if (head == NULL)
    {
      std::cout << "There is no Node left" << std::endl;
      return;    // the function ends here.
    }

    for (Node* tmp = head; tmp != NULL; tmp = tmp->next)
    {
```

```
      std::cout << tmp->value << ",";
    }

    std::cout << std::endl;
}

Node* deleteNode(Node* Node_i, Node* head)
{
    Node* tmp = head;
    Node* previous = NULL;
    // if Node_i is the header
    if (Node_i == head)
    {
      tmp = Node_i->next;
      delete Node_i;
      return tmp;
    }
    else
    {
        // if Node_i is not the header
      for (; tmp != NULL; tmp = tmp->next)
      {
            if (tmp->next == Node_i)
            {
                  previous = tmp;
                  tmp = tmp->next;
                  break;
            }
      }
      previous->next = tmp->next;
      delete tmp;  // delete Node_i
      return head;
    }

}
```

Q12.

```cpp
// Main.cpp  file
#include "matrix.h"

using namespace NMethod;
int main(int argc, char* argv[])
{
    Matrix a(2,2);
    ValueType data[] = {1,2,3,4};
    Matrix b(data,2,2);
    std::cout << "create matrix a with element zero " <<
std::endl;
    a.print();
    std::cout << "create matrix b with element 1 2 3 4 " <<
std::endl;
    b.print();
    ValueType data2[] = {1,1,1,1};
    Matrix c = Matrix::diag(data2,
sizeof(data2)/sizeof(data2[0]));
    std::cout << "create diagonal matrix c " << std::endl;
    c.print();
    Matrix d = Matrix::ones(3,1);
    std::cout << "create a 3 x 1 matrix with all element 1 " <<
std::endl;
    d.print();
    Matrix e = Matrix::indentity(3);
    std::cout << "create identity matrix e " << std::endl;
    e.print();
    ValueType data3[] = {2,-1,-2,-4, 6, 3,-4,-2,8};
    Matrix f(data3, 3, 3);
    std::cout << "matrix f = " << std::endl;
    f.print();

    ValueType data4[] = {1,2,3,-2,5,6,2,3,5};
    Matrix g(data4, 3, 3);
    Matrix q(3,3);
    std::cout << "matrix g = " << std::endl;
    g.print();
    q = f + g;
    std::cout << "matrix f + g =  " << std::endl;
    q.print();
    q = f * g;
    std::cout << "matrix f*g = " << std::endl;
    q.print();
    system ("pause");
}
```

```
// Matrix.cpp contains matrix operator definition
#include "matrix.h"
#include <iostream>
#include <exception>
#include <iomanip>

namespace NMethod
{
    Matrix::Matrix(){}

    Matrix::Matrix(int row, int column)
      :d_rows(row),d_cols(column)
    {
      if(row < 0 || column < 0)
           throw std::invalid_argument("Matrix rows and columns
should be positive");
      d_data.resize(row);
         for(int i = 0; i < row; ++i)
           d_data[i].resize(column, 0);
    }
    Matrix::Matrix(ValueType * data, int row, int column)
      :d_rows(row),d_cols(column)
    {
      if(row < 0 || column < 0)
           throw std::invalid_argument("Matrix rows and columns
should be positive");
      if(data == NULL)
           throw std::invalid_argument("data is NULL");
      d_data.resize(row);
      for(int i = 0; i < row; ++i)
           for(int j = 0; j < column; ++j)
                d_data[i].push_back(data[i * column + j]);
    }

    Matrix::Matrix(const Matrix& rhs)
    {
      d_rows = rhs.rows();
      d_cols = rhs.cols();
      d_data.resize(rhs.rows());
      for(int i = 0; i < rhs.rows(); ++i)
           for(int j = 0; j < rhs.cols(); ++j)
                d_data[i].push_back(rhs[i][j]);
    }

    Matrix& Matrix::operator=(const Matrix& rhs)
    {
      if(this == &rhs)
           return *this;
      d_data.clear();
      d_rows = rhs.rows();
```

```
      d_cols = rhs.cols();
      d_data.resize(rhs.rows());
      for(int i = 0; i < rhs.rows(); ++i)
           for(int j = 0; j < rhs.cols(); ++j)
                d_data[i].push_back(rhs[i][j]);
      return *this;
    }

    std::vector<ValueType>& Matrix::operator[](int row)
    {
      if(row < 0 || row > d_rows)
           throw std::range_error("index out of bound");
      else
           return d_data[row];
    }

    //std::vector<ValueType> Matrix::operator[](int row) const
    /**
    {
      std::vector<ValueType> b;
      if(row < 0 || row > d_rows)
           throw std::invalid_argument("index out of bound");
      else
      {
           b = d_data[row];
           return b;
      }
    }
    **/

    const std::vector<ValueType>& Matrix::operator[](int row)
const
    {
      if(row < 0 || row > d_rows)
           throw std::invalid_argument("index out of bound");
      else
           return d_data[row];
    }

    /**
    std::vector<ValueType> Matrix::operator[](int row) const
    {
      if (row < 0 || row > d_rows)
           throw std::invalid_argument("index out of bound");
      else
           return d_data[row];
    }
    **/

    Matrix Matrix::operator+(const Matrix& b) const
```

```
{
     if(d_rows != b.rows() && d_cols != b.cols())
          throw std::invalid_argument("Matrix a+b: Matrix b
should have the same size as Matrix a");
     Matrix c(d_rows, d_cols);
     for(int i = 0; i < d_rows; ++i)b[i][j] = 4;
          for(int j = 0; j < d_cols; ++j)
               c[i][j] = d_data[i][j] + b[i][j];
//std::vector<ValueType> vbi = b[i]; vbi[j] = 4;
     return c;
    }

    Matrix Matrix::operator-(const Matrix& b) const
    {
     if(d_rows != b.rows() && d_cols != b.cols())
          throw std::invalid_argument("Matrix a-b: Matrix b
should have the same size as Matrix a");
     Matrix c(d_rows, d_cols);
     for(int i = 0; i < d_rows; ++i)
          for(int j = 0; j < d_cols; ++j)
               c[i][j] = d_data[i][j] - b[i][j];
     return c;
    }
    Matrix Matrix::operator*(const Matrix& b) const
    {
     if(d_cols != b.rows())
          throw std::invalid_argument("Matrix a*b: cols of
Matrix a should equal to rows of Matrix b");
     Matrix c(d_rows, b.cols());
     for(int i = 0; i < d_rows; ++i)
     {
          for(int j = 0; j < b.cols(); ++j)
          {
               c[i][j] = 0;
               for(int k = 0; k < b.rows(); ++k)
               {
                    c[i][j] += d_data[i][k] * b[k][j];
               }
          }
     }
     return c;
    }

    Matrix Matrix::diag(ValueType * data, int length)
    {
     //all zero matrix
     Matrix A(length, length);
       for(int i = 0; i < length; ++i)
          A[i][i] = data[i];
     return A;
```

```
    }

    Matrix Matrix::ones(int row, int column)
    {
     Matrix A(row, column);
     for(int i = 0; i < row; ++i)
          for(int j = 0; j < column; ++j)
               A[i][j] = 1;
     return A;
    }
    Matrix Matrix::indentity(int length)
    {
     if(length < 0)
          throw std::invalid_argument("Matrix rows and columns
should be positive");
     Matrix A(length, length);
        for(int i = 0; i < length; ++i)
          for(int j = 0; j < length; ++j)
               A[i][i] = 1.0;
     return A;
    }

    void Matrix::print() const
    {
     std::cout << std::endl;
     for(int i = 0; i < d_rows; ++i)
     {
          for(int j = 0; j < d_cols; ++j)
          {
               std::cout << std::setw (8) << d_data[i][j] <<
",";
          }
          std::cout << std::endl;
     }
    }
}//end namespace NMethod
```

```
// Matrix.h defines header file

#ifndef INCLUDED_MATRIX_H
#define INCLUDED_MATRIX_H
#include<iostream>
#include<vector>

namespace NMethod
{
    typedef double ValueType;
    class Matrix
    {
    private:
      std::vector<std::vector<ValueType> > d_data;
      int d_rows;
      int d_cols;
    public:
      //constructor
      Matrix();
      Matrix(int row, int column);
      Matrix(ValueType * data, int row, int column);

      //return a diagonal matrix with diagonal value data
      static Matrix diag(ValueType * data, int length);
      static Matrix ones(int row, int column);
      static Matrix indentity(int length);
      Matrix(const Matrix& rhs);
      Matrix& operator=(const Matrix& rhs);

      //operator overloading
      std::vector<ValueType>& operator[](int row);
      const std::vector<ValueType>& operator[](int row) const;
      //std::vector<ValueType> operator[](int row) const;
      Matrix operator+(const Matrix& b) const;
      Matrix operator-(const Matrix& b) const;
      Matrix operator*(const Matrix& b) const;

      //getters
      int rows()const{return d_rows;}
      int cols()const{return d_cols;}

      //utils
      void print() const;
    };
}
#endif
```

Q13.
The answers are found in the following two FAQs at https://isocpp.org/wiki/faq/virtual-functions

What is a "virtual constructor"?

An idiom that allows you to do something that C++ doesn't directly support. You can get the effect of a `virtual` constructor by a `virtual clone()` member function (for copy constructing), or a `virtual create()` member function (for the default constructor).

```
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();      }
```

In the `clone()` member function, the `new Circle(*this)` code calls `Circle`'s copy constructor to copy the state of `this` into the newly created `Circle` object. (Note: unless `Circle` is known to be final (AKA a leaf), you can reduce the chance of slicing by making its copy constructor `protected`.) In the `create()` member function, the `new Circle()` code calls `Circle`'s default constructor.

Users use these as if they were "`virtual` constructors":

```
void userCode(Shape& s)
{
  Shape* s2 = s.clone();
  Shape* s3 = s.create();
  // ...
  delete s2;    // You need a virtual destructor here
  delete s3;
}
```

This function will work correctly regardless of whether the `Shape` is a `Circle`, `Square`, or some other kind-of `Shape` that doesn't even exist yet.

Note: The return type of `Circle`'s `clone()` member function is intentionally different from the return type of `Shape`'s `clone()` member function. This is called *Covariant Return Types*, a feature that was not originally part of the language. If your compiler complains at the declaration of `Circle* clone()`

`const` within class `Circle` (e.g., saying "The return type is different" or "The member function's type differs from the base class virtual function by return type alone"), you have an old compiler and you'll have to change the return type to `Shape*`.

Why don't we have `virtual` constructors?

A virtual call is a mechanism to get work done given partial information. In particular, `virtual` allows us to call a function knowing only an 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.

Techniques for using an indirection when you ask to create an object are often referred to as "Virtual constructors". For example, see TC++PL3 15.6.2.

For example, here is a technique for generating an object of an appropriate type using an abstract class:

```
struct F {  // interface to object creation functions
    virtual A* make_an_A() const = 0;
    virtual B* make_a_B() const = 0;
};

void user(const F& fac)
{
    A* p = fac.make_an_A(); // make an A of the appropriate type
    B* q = fac.make_a_B();  // make a B of the appropriate type
    // ...
}

struct FX : F {
    A* make_an_A() const { return new AX(); } // AX is derived from A
    B* make_a_B() const { return new BX();  } // BX is derived from B
};

struct FY : F {
    A* make_an_A() const { return new AY(); } // AY is derived from A
    B* make_a_B() const { return new BY();  } // BY is derived from B
};

int main()
{
    FX x;
    FY y;
    user(x);    // this user makes AXs and BXs
    user(y);    // this user makes AYs and BYs

    user(FX()); // this user makes AXs and BXs
    user(FY()); // this user makes AYs and BYs
    // ...
}
```

This is a variant of what is often called "the factory pattern". The point is that user() is completely isolated from knowledge of classes such as AX and AY.

Q14.

The answer is found as a FAQ here : https://isocpp.org/wiki/faq/strange-inheritance#calling-virtuals-from-base

Q16 How to inverse a string of sentence (without reverse the word) ?

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

using namespace std;

void reverse(char str[], int start, int end);
void reverseWord(char str[]);

int main()
{
    char str[] = "I am good";
    cout << str << endl;
    reverse(str, 0, strlen(str) - 1);
    cout << str << endl;
    reverseWord(str);
    cout << str << endl;

    system("pause");
}

// reverse a passed-in string
void reverse(char str[], int start, int end)
{
    int len = end - start + 1;
    for (int i = 0; i < len/2; ++i)
    {
      char tmp = str[start + i];
      str[start + i] = str[end - i];
      str[end - i] = tmp;
    }
}

// reverse each word in a passed-in string
void reverseWord(char str[])
{
    for (int i = 0, left = 0; i < strlen(str); ++i)
    {
      // if it is not the end of the string
      if (str[i] == ' ')
      {
           reverse(str, left, i - 1);
           left = i + 1;
      }
      // if it is the end of the string
      if (i == strlen(str) - 1)
      {
           reverse(str, left, i);
      }
    }
}
```

Q17

```python
# -*- coding: utf-8 -*-
"""
Created on Tue Oct 13 21:01:05 2015

@author: Bo He
"""

# One transaction: buy once and sell once, and buy take place
before sell.

stock_prices = [50,60,90,30,60,10,40]

def maxProfit(stock_prices):
    if len(stock_prices) < 2: return 0
    max_profit = 0
    mintillnow = stock_prices[0]
    for r in stock_prices:
        max_profit = max(max_profit, r - mintillnow)
        mintillnow = min(mintillnow, r)

    return max_profit

maxProfit(stock_prices)

# Two transactions: buy twice and sell twice, the first sell must
take place before the second buy
class Solution(object):
    def maxProfit(self, prices):
        """
        :type prices: List[int]
        :rtype: int
        """
        import sys
        if len(prices) < 2 : return 0
        profit1 = 0
        profit2 = -sys.maxint-1
        buy1 = -prices[0]
        buy2 = -sys.maxint-1
        for i in range(1, len(prices)):
            if prices[i] < prices[i-1]:
                buy1 = max(buy1, -prices[i])
                buy2 = max(buy2, profit1-prices[i])
            else:
                profit1 = max(profit1, buy1 + prices[i])
                profit2 = max(profit2, buy2 + prices[i])

        return max(profit1, profit2)
```

Q18.
Analysis and code for this problem can be found at http://blog.csdn.net/skyworth0103/article/details/38472639. The problem is about how to construct Hamiltonian cycle given ORE condition (which is a sufficient condition on the existence of Hamiltonian cycle).