Solution to Quiz 2

Group A

October 12, 2015

1. Generally speaking, a chi-squared test, also referred to as test (or chi-square 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.

   There are different applications of χ^2 test. The most commonly used χ^2 test is the Pearson's χ^2 test. This is one of goodness-of-fit tests mainly to handle the discrete distribution. Pearson's χ^2 test is used to test whether a given set of samples follow a specific discrete distribution P. If the number of samples is N, the test statistic is

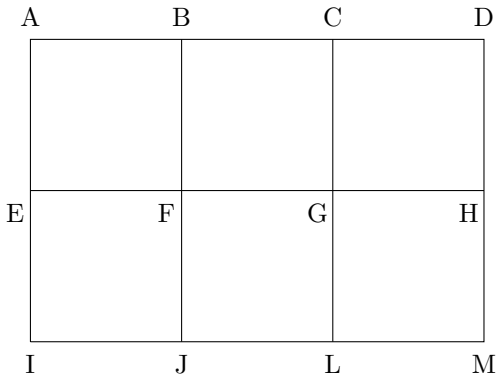
   $$\sum_{x \in \mathcal{X}} \frac{(N\hat{p}_x - Np_x)^2}{Np_x}$$

   where $\hat{p}_x$ is the empirical distribution. The sum above asmptotically approaches χ^2 distribution with degree $|\mathcal{X}| - p$ as N goes to infinity, where $p = s+1$, s is the number of parameters used to characterize the distribution P. Assume P is discrete uniform distribution on $1 \sim k$. Assume N is large, denote N_1 as the number of 1s ($N\hat{p}_1$ is a realization of N_1) which has binomial distribution $B(N, 1/k) \sim Poisson(N/k) \sim N(N/k, N/k)$. Then the above sum approximates to sum of square of independent Gaussian, which is the χ^2 distribution.

   Other chisquare tests include

   - CMH test: for stratified 2 by 2 tables to test whether rows and cols are independent
   - McNemar test for paired 2 by 2 table to test marginal homogeneity
   - Likelihood Ratio Test: test whether two nested models are equally well-fitted for the data
   - LjungBox test: in time series analysis, test whether any group of autocorrelation among residuals are different from 0.

2. Given a 2×3 grid with 6 blocks and 17 edges



Assuming edge length is 1, we want to find the shortest route to visit all edges. This is a variant of the famous *Seven Bridges of Königsberg Problem*. In the original paper of Euler, he proved that

Theorem (Euler)**.** *(a) A finite graph G contains an Euler circuit if and only if G is connected and contains no vertices of odd degree.*

(b) A finite graph G contains an Euler path if and only if G is connected and contains at most two vertices of odd degree.

In this problem, we can see that nodes B, C, E, H, J, L are vertices of odd degrees. So we need to connect nodes B, C and nodes J, L to guarentee the existence of an Euler path. Once we have these done, we know the shortest route to visit all edges should be with length $17 + 2 = 19$. A possible shortest route would be

$$E - A - B - C - D - H - G - C - B - F - E - I - J - L - G - F - J - L - M - H$$

3. By Bayes's rule, we have $f(X = x|X + Y > 0) = \frac{f(X=x, Y>-x)}{P(Y>-X)} = \frac{f(X=x)P(Y>-x)}{0.5} = 2f(X = x)\Phi(x)$. We obtain $P(Y + X > 0) = 0.5$ by symmetry.

4. (1). It depends on the individual risk attitude. If not stop, we could keep rolling until the sum exceed 42. For this strategy, the possible amount we receive could be 0 or any number between 43 and 48. This strategy has non-zero variance (risk), its expectation would be greater than $\frac{1}{6} * 0 + \frac{5}{6} * 43 > 35$, where 35 is the amount we receive riskless if stop. Thus if not stop, there exists a strategy with higher return and higher risk. It really depends on individual risk attituede. The relationship is like stock and riskless bond.

(2). 44. If start from 35 and keep rolling until sum exceed 43, possible amount received, which is denoted as X, could be 0, or anynumber from 44 to 48. For $44 \le k \le 48$,

$$\begin{aligned} P(X = k) &= \sum_{i=1}^{6} P(\text{final sum minus last time roll is } k-i)\frac{1}{6} I_{\{k-i\le 43 \text{ and k-i is not square number}\}} \quad (1)\\ &= \sum_{i=1}^{6} P(\text{start from 35 and keep rolling, sum reaches } k-i)\frac{1}{6} I_{\{k-i\le 43 \text{ and k-i is not square number}\}} \quad (2)\end{aligned}$$

For $44 \le k \le 48$, $P(X = k)$ are listed below,

$$0.236, 0.203, 0.165, 0.121, 0.070,$$

$P(X = 0) = 1 - \sum_{i\neq 0} P(X = i) = 0.204$. Thus 44 is the most probable number. The program is made executable online: `http://cpp.sh/64bp`.

(3). Keep rolling if the sum doesn't exceed 18. Let A_n denote the amount received if we can keep rolling until the sum exceed n.

$$EA_n = \sum_{i=1}^{6}(n+i)P(\text{final sum is n+i} \mid \text{stop once sum is greater than n})I_{\{\text{n+i is not squre number}\}},$$

and we have,

$$\begin{aligned} & P(\text{final sum is k}| \text{ stop once sum is greater than n}) \quad (3)\\ = & \sum_{i=1}^{6} P(\text{start from 0 and keep rolling, sum is k-i})\frac{1}{6} I_{\{0\le k-i\le n \text{ and k-i is not square number}\}}. \quad (4)\end{aligned}$$

Similarly, we can calculate $Var(A_n)$.

The following program print EA_n and $Var(A_n)$ for $1 \le n \le 50$;
P(n) calculate all P(start from 0 and keep rolling, sum is i) for $1 \le i \le n$;
PG(m,n,vec) returns P(final sum is m| stop once sum is greater than n).

```
// Example program
#include <iostream>
#include <string>
#include <vector>
#include <cmath>

using std::vector;

vector<double> P(int n){
    vector<double> p(n);
    p[0]=1;
    for(int j=1; j<n; ++j){
        if(int(sqrt(j))*int(sqrt(j))==j)
            p[j]=0;
        else{
            for(int i=1; j-i>=0 && (i<=6); ++i){
                p[j]+=p[j-i]*1.0/6;
                }
            }
        }
    return p;
}

double PG(int m, int n, vector<double> &vec){
    double temp=0;
    if(int(sqrt(m))*int(sqrt(m))==m)
        return 0;
    for(int i=1; i<=6; ++i){
        if(m-i<=n && (m-i>=0))
            temp+=vec[m-i]*1.0/6;
    }
    return temp;
}

int main()
{
  vector<double> vec=P(50);
  for(int i=1; i<vec.size(); ++i){
      double e=0, v=0;
      for(int j=1; j<=6; ++j){
        e+=(i+j)*PG(i+j,i,vec);
      }
      for(int j=1; j<=6; ++j){
        v+=((i+j)-e)*((i+j)-e)*PG(i+j,i,vec);
      }
      std::cout <<"E is "<< e << ", Var is "<<v<< " when n is " << i<<std::endl ;
      }
}
/*Result
E is 2.66667, Var is 2.85185 when n is 1
E is 3.13889, Var is 3.33948 when n is 2
E is 3.39815, Var is 4.17314 when n is 3
E is 3.39815, Var is 4.17314 when n is 4
E is 3.85185, Var is 7.23971 when n is 5
E is 4.38117, Var is 11.571 when n is 6
E is 4.66538, Var is 14.3787 when n is 7
E is 4.99696, Var is 18.3424 when n is 8
E is 4.99696, Var is 18.3424 when n is 9
E is 5.10798, Var is 21.6762 when n is 10
E is 5.23752, Var is 26.5407 when n is 11
E is 5.35713, Var is 31.9271 when n is 12
E is 5.45992, Var is 37.3333 when n is 13
E is 5.56011, Var is 43.366 when n is 14
E is 5.65397, Var is 49.7352 when n is 15
E is 5.65397, Var is 49.7352 when n is 16
```

```
E is 6.03616, Var is 56.8327 when n is 17
E is 6.39138, Var is 63.6968 when n is 18
E is 6.32839, Var is 68.5778 when n is 19
E is 6.26861, Var is 73.8487 when n is 20
E is 6.21222, Var is 79.4216 when n is 21
E is 6.15895, Var is 85.2538 when n is 22
...
*/
```

It is clear that 18 is the first local maximum for EA_n, while the $Var(A_n)$ is monotone increasing with n, just as expected. Thus in such a game, we should keep rolling if the sum doesn't exceed 18. The program and result is made executable online: `http://cpp.sh/263z`.

5. Choose two random variables X and Y that uniformly lie in [0,1]. Let $A = \min(A, B)$ and $B = \max(A, B)$. Without loss of generality, we divide the stick into three pieces and let the length of the first piece be A, the second piece has length B-A and the third piece has length $1-B$. We first calculate the expected length for the smallest one. Suppose $S = \min(A, B-A, 1-B)$, then the cdf of S is the following:

$$\begin{aligned} F(a) &= 1 - P(A \geq a, B-A \geq a, 1-B \geq a) \\ &= 1 - P(x \geq a, y \geq x, y-x \geq a, 1-y \geq a) - P(y \geq a, x \geq y, x-y \geq a, 1-x \geq a) \\ &= 1 - 2P(x \geq a, y \geq x, y-x \geq a, 1-y \geq a) \\ &= 1 - 2 * P(x \geq a, y \geq x+a, y \leq 1-a) \\ &= 1 - (1-3a)^2, \end{aligned}$$

for $a \leq \frac{1}{3}$. Thus we know the pdf is 6(1-3a),

$$E[S] = \int_0^{\frac{1}{3}} 6a(1-3a)da = \frac{1}{9}.$$

Similarly, if we denote the length of the largest one as $L = \max(A, B-A, 1-B)$, then the cdf of L is the following:

$$\begin{aligned} F(a) &= P(A \leq a, B-A \leq a, 1-B \leq a) \\ &= P(x \leq a, y \geq x, y-x \leq a, 1-y \leq a) - P(y \leq a, x \geq y, x-y \leq a, 1-x \leq a) \\ &= 2P(x \leq a, y \geq x, y-x \leq a, 1-y \leq a) \\ &= 2 * P(x \leq a, 1-a \leq y \leq a+x, y \geq x) \\ &= \begin{cases} -3a^2+6a-2 & a \geq \frac{1}{2} \\ (3a-1)^2 & \frac{1}{3} \leq a \leq \frac{1}{2} \end{cases}. \end{aligned}$$

Thus, we know the pdf is

$$f(a) = \begin{cases} 6-6a & a \geq \frac{1}{2} \\ 6(3a-1) & \frac{1}{3} \leq a \leq \frac{1}{2} \end{cases}.$$

$$E[L] = \int_{\frac{1}{3}}^{\frac{1}{2}} 6a(3a-1)da + \int_{\frac{1}{2}}^{1} 6a - 6a^2 da = \frac{1}{9} + \frac{1}{2} = \frac{11}{18}.$$

Since the all three pieces sum up to 1, we know the expected length of the middle size one is $1 - \frac{11}{18} - \frac{1}{9} = \frac{1}{3}$.

If we divide the stick into n segments then the k_{th} largest segment will have expected length $\frac{1}{n}(\frac{1}{n} + \cdots + \frac{1}{k})$[**?**][**?**]. ref: [1] H.A. David and H.N.Nagaraja LATEX: Order Statistics, Addison Wesley, Massachusetts, 3rd edition, 2003. [2] http://math.stackexchange.com/questions/13959/if-a-1-meter-rope-is-cut-at-two-uniformly-randomly-chosen-points-what-is-the-av

6. (a) Let Y be the number of people who select their own hats. To compute $\mathbb{E}[Y]$, we introduce i.i.d. random variables

$$Y_i = \begin{cases} 1 & \text{the i-th person select his own hat} \\ 0 & \text{otherwise} \end{cases}$$

So, we can compute

$$\mathbb{E}[Y] = \sum_{i=1}^{N} \mathbb{E}[Y_i] = N \cdot \frac{1}{N} = 1.$$

(b) To compute $var(Y)$, again we have

$$\begin{aligned} var(Y) &= var\left(\sum_{i=1}^{N} Y_i\right) = \sum_{i=1}^{N} var(Y_i) + \sum_{i \neq j} cov(Y_i, Y_j) \\ &= N \cdot \frac{N-1}{N^2} + N(N-1) \cdot \frac{1}{N^2(N-1)} = 1. \end{aligned}$$

(c) Let $R(N)$ be the number of rounds that are run.
Let $S(N)$ be the total number of selections made by these N individuals,
Let $F(N)$ be the number of false selections made by these N individuals.

Intuitively, we can answer $\mathbb{E}[R(N)]$, $\mathbb{E}[S(N)]$, and $\mathbb{E}[F(N)]$ pretty easily. Since by (i), on average we know each round there should be 1 person selects his/her hat. Hence, the game should last for N rounds, i.e. $\mathbb{E}[R(N)] = N$.
Since on average, every round there should be 1 person quits the game, the total number of selections made should be $N + (N-1) + ... + 1 = (N+1)N/2$, i.e. $\mathbb{E}[S(N)] = (N+1)N/2$.
Similarly, the number of false selections made should be $(N-1) + (N-2) + ... + 1 = N(N-1)/2$, i.e. $\mathbb{E}[F(N)] = N(N-1)/2$. In terms of the expected number of false selections made by 1 person, we have by symmetry the answer equals $(N-1)/2$.

However, to make the calculation rigorously, we need the following facts

$$\begin{aligned} \sum_{n=0}^{N} \mathbb{P}(Y = n) &= 1, \\ \mathbb{E}[Y] = \sum_{n=0}^{N} n\mathbb{P}(Y = n) &= 1, \\ \mathbb{E}[Y^2] = \sum_{n=0}^{N} n^2\mathbb{P}(Y = n) &= 1. \end{aligned}$$

Let us do the mathematically rigorous calculation.

(d) Prove by induction that $\mathbb{E}[R(N)] = N$. Trivially, $\mathbb{E}[R(0)] = 0$. Assume that $\mathbb{E}[R(n)] = n$ for all

$0 \leq n < N$, we have

$$\begin{aligned}
\mathbb{E}[R(N)] &= \sum_{n=0}^{N} \mathbb{E}[R(N)|Y=n]\mathbb{P}(Y=n) \\
&= \sum_{n=0}^{N} \left(\mathbb{E}[R(N-n)]+1\right)\mathbb{P}(Y=n) \\
&= 1 + \mathbb{E}[R(N)]\mathbb{P}(Y=0) + \sum_{n=1}^{N} (N-n)\,\mathbb{P}(Y=n) \\
&= 1 + \mathbb{E}[R(N)]\mathbb{P}(Y=0) + N\sum_{n=0}^{N} \mathbb{P}(Y=n) - N\mathbb{P}(Y=0) - \sum_{n=0}^{N} n\mathbb{P}(Y=n) \\
&= \mathbb{E}[R(N)]\mathbb{P}(Y=0) + N(1-\mathbb{P}(Y=0))
\end{aligned}$$

Hence, using the fact that $\mathbb{P}(Y=0) > 0$, we can solve exactly $\mathbb{E}[R(N)] = N$.

(e) Prove by induction that $\mathbb{E}[S(N)] = (N+1)N/2$. Trivially, $\mathbb{E}[S(0)] = 0$. Assume that $\mathbb{E}[S(n)] = (n+1)n/2$ for all $0 \leq n < N$, we have

$$\begin{aligned}
\mathbb{E}[S(N)] &= \sum_{n=0}^{N} \mathbb{E}[S(N)|Y=n]\mathbb{P}(Y=n) = \sum_{n=0}^{N} \left(\mathbb{E}[S(N-n)]+N\right)\mathbb{P}(Y=n) \\
&= N + \mathbb{E}[S(N)]\mathbb{P}(Y=0) + \sum_{n=0}^{N} \frac{(N-n+1)(N-n)}{2}\mathbb{P}(Y=n) - \frac{(N+1)N}{2}\mathbb{P}(Y=0) \\
&= N + \mathbb{E}[S(N)]\mathbb{P}(Y=0) + \frac{(N+1)N}{2}\sum_{n=0}^{N} \mathbb{P}(Y=n) - N\sum_{n=0}^{N} n\mathbb{P}(Y=n) \\
&+ \frac{1}{2}\sum_{n=0}^{N} n^2\mathbb{P}(Y=n) - \frac{1}{2}\sum_{n=0}^{N} n\mathbb{P}(Y=n) \\
&= \mathbb{E}[S(N)]\mathbb{P}(Y=0) + \frac{(N+1)N}{2}(1-\mathbb{P}(Y=0))
\end{aligned}$$

Hence, using the fact that $\mathbb{P}(Y=0) > 0$, we can solve exactly $\mathbb{E}[R(N)] = (N+1)N/2$.

(f) Again prove by induction that $\mathbb{E}[F(N)] = N(N-1)/2$. Trivially, $\mathbb{E}[F(0)] = 0$. Assume that $\mathbb{E}[F(n)] = n(n-1)/2$ for all $0 \leq n < N$, we have

$$\begin{aligned}
\mathbb{E}[F(N)] &= \sum_{n=0}^{N} \mathbb{E}[F(N)|Y=n]\mathbb{P}(Y=n) = \sum_{n=0}^{N} \left(\mathbb{E}[F(N-n)]+N-n\right)\mathbb{P}(Y=n) \\
&= \mathbb{E}(F(N))\mathbb{P}(Y=0) + \sum_{n=0}^{N} \frac{(N-n+1)(N-n)}{2}\mathbb{P}(Y=n) - \frac{N(N-1)}{2}\mathbb{P}(Y=0) \\
&= \mathbb{E}(F(N))\mathbb{P}(Y=0) + \frac{N(N-1)}{2}(1-\mathbb{P}(Y=0))
\end{aligned}$$

Hence, using the fact that $\mathbb{P}(Y=0) > 0$, we can solve exactly $\mathbb{E}[F(N)] = N(N-1)/2$. In terms of the expected number of false selections made by 1 person, we have by symmetry the answer equals $(N-1)/2$.

7. Based on Theorem 3.7 in [1], we know when a new variable is added to the original regression, the change in R^2 followed the following formula:

$$R^2_{1,2} = R^2_1 + (1-R^2_1)r^2, \tag{5}$$

where the explicit formula for r^2 can be found on page 44 in [**?**]. Since$0 \leq r^2 \leq 1$, thus we know

$$0.2 \leq R_{1,2} \leq 1. \tag{6}$$

[1] H. G. William, LATEX: Econometric Analysis, 7th Edition, 2010

8. For given n, make 1/n into binary representation, define accordingly(fair coin head 1, tail 0): for ith flip of the fair coin, if large than br[i], define as tail(for biased coin); else continue. ref: `http://stackoverflow.com/questions/2040814/given-a-function-for-a-fair-coin-write-a-function-for-a-biased`

9. Denote E_i as the expectation accrossed bridges starting from island i, $E_{10} = 0$. Use conditional expectation we can get

$$E_k = 1 + 0.5E_1 + 0.5E_{k+1}.$$

Then we can get

$$E_1 = 2 + E_2, E_2 = \frac{2^8 - 1}{2^7} + \frac{2^8 - 1}{2^8} E_1.$$

Thus we have $E_1 = 1022$.

10. const int* const means the variable is a const pointer to const int, the const at the end means fun is a constant member function which cannot modify member variables. So the code means fun takes the p which is a reference to a constant pointer to a constant integer, fun returns a constant pointer to a constant integer. (`http://stackoverflow.com/questions/1143262/what-is-the-difference-between-const-int-co` `http://stackoverflow.com/questions/10716769/c-difference-between-const-positioning`, `http://stackoverflow.com/questions/30080720/meaning-of-const-int-const-funconst-int-const-p-const`)

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) {
        while (node->next->next != NULL) {
            node->val = node->next->val;
            node = node->next;
        }
        node->val = node->next->val;
        node->next = NULL;
    }
};
```

12.
```
#ifndef _MATRIX_H
#define _MATRIX_H
#include <vector>
template <typename T> class Matrix {
private:
 std::vector<std::vector<T> > mat;
 unsigned rows;
 unsigned cols;
public:
 Matrix(unsigned _rows, unsigned _cols, const T& _initial);
 Matrix(const Matrix<T>& rhs);
 virtual ~Matrix();
 // Operator overloading, for "standard" mathematical matrix operations
 Matrix<T>& operator=(const Matrix<T>& rhs);
```

```
// Matrix mathematical operations
Matrix<T> operator+(const Matrix<T>& rhs);
Matrix<T>& operator+=(const Matrix<T>& rhs);
Matrix<T> operator-(const Matrix<T>& rhs);
Matrix<T>& operator-=(const Matrix<T>& rhs);
Matrix<T> operator*(const Matrix<T>& rhs);
Matrix<T>& operator*=(const Matrix<T>& rhs);
Matrix<T> transpose();
// Matrix/scalar operations
Matrix<T> operator+(const T& rhs);
Matrix<T> operator-(const T& rhs);
Matrix<T> operator*(const T& rhs);
Matrix<T> operator/(const T& rhs);
// Matrix/vector operations
std::vector<T> operator*(const std::vector<T>& rhs);
std::vector<T> diag_vec();
// Access the individual elements
T& operator()(const unsigned& row, const unsigned& col);
const T& operator()(const unsigned& row, const unsigned& col) const;
// Access the row and column sizes
unsigned get_rows() const;
unsigned get_cols() const;
};
#endif
```

13. No. A 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. —(Bjarne Stroustup (P424 The C++ Programming Language SE))

    We can get the effect of a "virtual constructor" by a virtual clone() member function, or a virtual create() member function. (reference from `https://isocpp.org/wiki/faq/virtual-functions#virtual-ctors`) Example:

```
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();      }
```

14. Yes, it is okay. Example: void Shape::print() const float a = this-¿area(); // area() is pure virtual // ... `https://isocpp.org/wiki/faq/strange-inheritance#calling-virtuals-from-base`.

15. DP. Use $f(i, j)$ to denote whether $s[i, j)$ is the palindrome. This might even be reduced to a one demension $f(i)$, so the space complexity is $O(N)$. Link: `http://articles.leetcode.com/2011/11/longest-palindromic-substring-part-i.html`

```
string longestPalindromeDP(string s) {
 int n = s.length();
```

```
 int longestBegin = 0;
 int maxLen = 1;
 bool table[1000][1000] = {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);
}
```

16. Hack solution: Use python, first split by whitespace, then reverse the order of all words.

    Serious solution: Two pass algorithm, Reverse the entire string, then reverse the letters of each individual word.

```
void reverse(char *begin, char *end){
  char temp;
  while (begin < end)
  {
    temp = *begin;
    *begin++ = *end;
    *end-- = temp;
  }
}
void reverseWords(char *s){
    char *word_begin = NULL;
    char *temp = s; /* temp is for word boundry */

    while( *temp )
    {
        /*This condition is to make sure that the string start with
          valid character (not space) only*/
        if (( word_begin == NULL ) && (*temp != ' ') )
        {
            word_begin=temp;
        }
        if(word_begin && ((*(temp+1) == ' ') || (*(temp+1) == '\0')))
        {
            reverse(word_begin, temp);
            word_begin = NULL;
        }
        temp++;
    } /* End of while */

    reverse(s, temp-1);
}
```

17.
```
class Solution {
```

```
public:
    int maxProfit(vector<int> &prices) {
        if (prices.size() < 2) {
            return 0;
        }
        vector<int> forward;
        vector<int> backward;
        forward.push_back(0);
        backward.push_back(0);

        int valley = prices[0];
        for (int i = 1; i < prices.size(); i++) {
            forward.push_back(max(forward[forward.size() - 1], prices[i] - valley));
            valley = min(valley, prices[i]);
        }

        int top = prices[prices.size() - 1];
        for (int i = prices.size() - 2; i >= 0; i--) {
            backward.insert(backward.begin(), max(backward[0], top - prices[i]));
            top = max(top, prices[i]);
        }

        int profit = 0;
        for (int i = 0; i < prices.size(); i++) {
            profit = max(profit, forward[i] + backward[i]);
        }
        return profit;
    }
};
```

18. To faciliate the I/O, assume the INPUT is a vector of vector. We can use DFS method. Use s to memorize the visited people, choosing s as a set has two advantages, one is quick search the other is that set is ordered in C++. Once the s contains N unique number and 2 1s at two ends, save s as the result. After DFS, if the size of *result* is 0 that means no such path exists.

```
void dfs(vector<vector<int>> &vec, set<int> &s, set<int> &result){
    if(*(--s.end())==1){
        if(s.size()==vec.size())
            result=s;  //Once we find the desired path, save it as the result.
        return; //Don't need to go further.
        }
    for(int i=0; i<vec[*(--s.end())].size(); ++i){
        if(s.find(vec[*(--s.end())][i])!=s.end()||vec[*(--s.end())][i]==1){
            s.insert(vec[*(--s.end())][i]);
            dfs(vec, s);
            s.erase((--s.end()));
            }
        }
    }

set<int> path(vector<vector<int>> vec){
    set<int> s, result; //s is a temporary set to record current visited people.
    s.insert(1);
    dfs(vec, s, result);
    return result;
    }
```