Quiz 3

Group 2

1. Problem 1

Define indicator random variables $I_i, i = 1, ...16$ by $I_i = 1$ if a boy sits at seats i and his neighbor in the counterclockwise is a girl or a girl sits at seats i and her neighbor in the counterclockwise is a boy. Otherwise $I_i = 0$. Then expected boy-girl neighbors are $E(X) = E(\sum_{i=1}^{16} I_i) = \sum_{i=1}^{16} E(I_i) = \sum_{i=1}^{16} (\frac{9}{16} * \frac{7}{15} + \frac{7}{16} * \frac{9}{15}) = 16 * \frac{126}{240} = 8.4$.

2. Problem 2

Independence implies uncorrelatedness. But the other way around is not necessarily true. For example, if X is standard normal, then X and X^2 are uncorreated, but not independent. $cov(X, X^2) = EXEX^2 - EX^3 = 0$, so they are uncorrelated. But $P(0 < X < 1, X^{>}2) = 0 \neq P(0 < X < 1)P(X^2 > 1)$, so they are not independent.

Reference: `https://en.wikipedia.org/wiki/Normally_distributed_and_uncorrelated_does_not_imply_independent`

3. Problem 3

Solve this problem backward. If one tosesed the 3rd time, his expectation is $(1+2+3+4+5+6)/6 = 3.5$. That means in the second toss if he gets 4,5,6 he should stop at 2nd toss. If he gets 1,2,3 he should use the 3rd chance. So in the 2nd toss he has 1/6 chance to get 4,5,6 separately and 1/2 chance to get 3.5. The expectation is 4.25. That means if in the 1st round he gets 5 or 6 he should stop otherwise continue. In the 1st round he has 1/6 chance to get 5 or 6 seperately and 2/3 chance to get 4.25. The expectation is $(5+6)/6 + 4.25 * 2/3 = 14/3 = 4.67$.

The value of the game is 4.67.

4. Problem 4

It is $\frac{1}{n}\sum_{i=1}^{n} \frac{1}{i}$. See

$http://math.stackexchange.com/questions/14190/average-length-of-the-longest-segment$

If V_n denotes the largest piece: then $P(V_n > x) = P(V_1 > x \text{ or } V_2 > x...)$. By inclusion and exclusion principal:

$$P(V_n > x) = n(1-x)^{n-1} - \binom{n}{2}(1-2x)^{n-1} + \cdots.$$

So $E(V_n) = \int_0^\infty P(V_n > x)dx = \sum_{k=1}^{n} \binom{n}{k}(-1)^{k-1} \int_0^{1/k} (1-kx)^{n-1}dx = \sum_{k=1}^{n} \binom{n}{k}(-1)^{k-1}\frac{1}{nk} = \frac{1}{n}\sum_{k=1}^{n}\frac{1}{k}$.

5. Problem 5

The determinat of the covariance matrix for X, Y, Z is $det = 1-r(r-0)+r(0-r) = 1-2r^2$. Since $det \geq 0$, we get $0 \leq 1 - 2r^2$. So $-\frac{\sqrt{2}}{2} \leq r \leq \frac{\sqrt{2}}{2}$.

6. Problem 6

(a) Define stopping time by $\min\{n : S_n = -1 \text{ or } 99\}$. Then since S_N is a martingle, $E(S_N) = p_{-1} * (-1) + (1 - p_{-1}) * 99 = 0$, which implies $p_{-1} = 99/100$. SInce $S_N^2 - N$ is a margintale, $E(N) = E(S_N^2) = \frac{99}{100} * (-1)^2 + \frac{1}{100} * (99)^2 = 99$.

(b) If the drunk man hits the left locked door, he will go back, so it amount to the same problem as the (a) except here we need to replace -1 by 101. So the expected number of steps he takes to go home is $E(N) = E(S_N^2) = \frac{99}{200} * 101^2 + \frac{101}{200} * 99^2 = 101 * 99 = 9999$.

7. Problem 7

Ridge regression penalizes the size of the regression coefficients using L^2 norm. Specifically, the ridge regression estimate $\hat{\beta}$ is defined as the value of β that minimizes

$$\sum_i (y_i - x_i^T \beta)^2 + \lambda \sum_i \beta_i^2.$$

The solution of the above ridge problem is $\hat{\beta} = (x^T x + \lambda I)^{-1} x^T y$. Reference: `https://en.wikipedia.org/wiki/Least_squares` Reference: `https://en.wikipedia.org/wiki/Tikhonov_regularization`

8. Problem 8

Lasso regression penalizes the size of the regression coefficients using L^1 norm. Specifically, the lasso regression estimate $\hat{\beta}$ is defined as the value of β that minimizes

$$\sum_i (y_i - x_i^T \beta)^2 + \lambda \sum_i |\beta_i|.$$

Reference: `https://en.wikipedia.org/wiki/Least_squares`

9. Problem 9

Suppose that all the money in the world is less than 2^{46} US dollars. So after playing this game for 46 times, the money you could possibly get is always 2^{46} US dollars. Hence the expected payoff of this game is $46 \times 2^{46} * \sum_{n=47}^{\infty} \frac{1}{2^n} = 46 + 1 = 47$.

10. Problem 10

default means we want to use compiler-generated version of function, so dont need to specify a body. Here are cases we need default constructor: When we want to force compiler to generate constructor. When all parameters have default values;

11. Problem 11

```
class safeSingleton {
        static std::shared_ptr< safeSingleton > instance_;
        static std::once_flag    only_one;

        safeSingleton(int id) {
        std::cout << "safeSingleton::Singleton()" << id << std::endl;
```

```
        }
        safeSingleton(const safeSingleton& rs) {
        instance_ = rs.instance_;
        }
        safeSingleton& operator = (const safeSingleton& rs) {
        if (this != &rs) {
                instance_ = rs.instance_;
        }
        return *this;
        }

public:
        ~safeSingleton() {
        std::cout << "Singleton::~Singleton" << std::endl;
        }

        static safeSingleton & getInstance( int id ) {
        std::call_once( safeSingleton::only_one, [] (int idx) {
        safeSingleton::instance_.reset( new safeSingleton(idx) );
        std::cout << "safeSingleton::create_singleton_() |
            thread id " + idx << std::endl;
        }, id );
        return *safeSingleton::instance_;
        }
        void demo(int id) {
     std::cout << "demo stuff from thread id " << id << std::endl;
        }
};

std::once_flag safeSingleton::only_one;
std::shared_ptr< safeSingleton > safeSingleton::instance_ = nullptr;
```

Reference: `http://silviuardelean.ro/2012/06/05/few-singleton-approaches`

12. Problem 12

```
int maxProfitII(int[] prices) {
        int profit = 0;
        for (int i = 1; i < strlen(prices); i++) {
            if (prices[i] > prices[i-1]) {
                profit += (prices[i]-prices[i-1]);
```

```
            }
        }
    }

int maxProfit(int k, int[] prices) {
        int len = strlen(prices);
        if (len < 2) return 0;
        if (k >= len) {return maxProfitII(prices);}


        int[] local = new int[k + 1];
        int[] global = new int[k + 1];

        for (int i = 1; i < strlrn(prices) ; i++) {
            int diff = prices[i] - prices[i - 1];
            for (int j = k; j > 0; j--) {
                local[j] = Math.max(global[j - 1] +
                        Math.max(diff, 0), local[j] + diff);
                global[j] = Math.max(global[j], local[j]);
            }
        }

        return global[k];
    }


    public int maxProfit2(int[] prices) {
        int maxProfit = 0;

        for (int i = 1; i < strlen(prices); i++) {
            if (prices[i] > prices[i - 1]) {
                maxProfit += prices[i] - prices[i - 1];
            }
        }

        return maxProfit;


    }
}
```

Reference: https://leetcode.com/discuss/15153/a-clean-dp-solution-which-generalizes-to-

13. Problem 13

```
int FindUniq(int vector<int> &vec) {
    res = vec[0];
    for (int i = 1; i < vec.size(); i++) {
        res ^= vec[i];
    }
    return res;
}
```

14. Problem 14

Yes. Its sometimes (not always!) a great idea. For example, suppose all Shape objects have a common algorithm for printing, but this algorithm depends on their area and they all have a potentially different way to compute their area. In this case Shapes area() method would necessarily have to be virtual (probably pure virtual) but Shape::print() could, if we were guaranteed no derived class wanted a different algorithm for printing, be a non-virtual defined in the base class Shape.

```
#include "Shape.h"
void Shape::print() const
{
        float a = this->area();  // area() is pure virtual
        // ...
}
```

Reference: `https://isocpp.org/wiki/faq/strange-inheritance#calling-virtuals-from-base`

15. Problem 15

We can generate random numbers recurrencely use

$$X_{n+1} = (aX_n + b) \mod m$$

where a, b and m are large integers, and $X_{n+1}$is the next in X as a series of pseudo-random numbers. The maximum number of numbers the formula can produce is the modulus, m.

Reference: `https://en.wikipedia.org/wiki/Random_number_generation`

16. Problem 16

We can flip fair coin n times to get a number below 2^n. Then interpret this as a batch of m die flips, where m is the largest number so that $6^m < 2^n$ (as already said, equality never holds here). If we get a number greater or equal to 6^m we must reject the value and repeat all n flips.

Reference: `http://cs.stackexchange.com/questions/29204/how-to-simulate-a-die-given-a-fa`

17. Problem 17

```
void roboMeet() {
            bool reachedZero = false;
            while( !meet() ) {
                  if(reachedZero) {
                        moveLeft();
                        moveLeft();
                  } else {
                        moveLeft();
                        moveLeft();
                        moveRight();
                  }
                  if( at-zero() ) {
                        reachedZero = true;
                  }
            }
      }
```

Reference: https://en.wikibooks.org/wiki/Puzzles/Logic_puzzles/Parachuted_Robots