Solutions to Qishi Quiz One

Si Chen, Yupeng Li, Jie Wang, Xian Xu and Yuanda Xu

September 29, 2015

Math Question

1)

Consider the general case. We have totally N points, pick up one and put it at 12:00. Then the circle is separated into two parts. The probability that the rest $N-1$ points are in the clockwise semicircle starting at the pick point is $1/2^{N-1}$. Any of these N points could be the one put at 12:00. So the totally probability is $N/2^{N-1}$.

2)

Let 0 denote the starting vertex and 3 denote the opposite vertex. There are two types of intermediary vertices: vertex type 1 which is the neighbor of 0 and vertex type 2 which is the neighbor of vertex 3. Let $f(n)$ denote the additional steps needed to reach vertex 3 if we are at vertex type n. We then have the following equations:

$$\begin{aligned} f(3) &= 0, \\ f(2) &= 1 + \frac{1}{3}f(3) + \frac{2}{3}f(1), \\ f(1) &= 1 + \frac{1}{3}f(0) + \frac{2}{3}f(2), \\ f(0) &= 1 + f(1). \end{aligned}$$

Solve the above equations, we obtain

$$f(0) = 10,$$

i.e. the expectation of the number of steps for the ant to walk from one vertex to the opposite vertex is 10.

3)

Suppose we choose to play the game. Let (b, r) denote the number of black and red cards left in the deck. At each (b, r), we need to decide whether to continue the game or stop:

- If we choose to stop, then our final payoff would be $r - b$.
- If we choose to continue, then the state would change to $(b-1, r)$ with probability $\frac{b}{b+r}$, and change to $(b, r-1)$ with probability $\frac{r}{b+r}$.

Therefore, if we let $f(b, r)$ denote the expected payoff at state (b, r), then it follows from the above reasoning that

$$f(b, r) = \max\left\{r - b, \frac{b}{b+r}f(b-1, r) + \frac{r}{b+r}f(b, r-1)\right\}.$$

Given the boundary conditions that $f(b, 0) = 0$ and $f(0, r) = r$, we can recursively calculate $f(b, r)$ for all possible states (b, r). After some calculation we obtain the expected payoff before playing the game, i.e. $f(26, 26) =$ \$2.62.

Given the above analysis, we should play the game at a cost of no higher than \$2.62.

4)

This problem is a biased random walk problem. We use $f(n)$ to indicates the probability of starting from state n and return to 0. $f(1) = min((1-p)/p, 1)$ and $f(-1) = min(p/(1-p), 1)$. Therefore $f(0) = p*f(1)+(1-p)*f(-1) = 2*min(p, 1-p)$.

5)

Recall that a random variable τ has exponential distribution if its density function has the form.

$$f(t) = \begin{cases} \lambda e^{-\lambda t} & t \geq 0, \\ 0 & t < 0. \end{cases}$$

Exponential random variables have memoryless property. A simple computation shows

$$\mathbb{E}\tau = \int_0^\infty \lambda t e^{-\lambda t} = \frac{1}{\tau},$$

and

$$\mathbb{P}(\tau > y) = \int_y^\infty \lambda e^{-\lambda t} = e^{-\lambda y}.$$

Thus we can assume the light bulbs with average life of 100 hours corresponds to exponential variable with $\lambda = 0.01$ and the ones with 200 hours of average life corresponds to $\lambda = 0.005$. Now suppose we have exponential variabes τ_1,...,τ_5 with $\lambda = 0.01$ and τ_6,...,τ_{10} with $\lambda = 0.005$. They are independent. The time for the first light bulb to burn out is the random variable

$$\xi = \min\{\tau_i \mid i = 1, ..., 10\}.$$

We compute the distribution of ξ directly, using independency,

$$\mathbb{P}(\xi > y) = \mathbb{P}(\tau_i > y, i = 1, ..., 10.) = \mathbb{P}(\tau_1 > y) \cdot ... \cdot \mathbb{P}(\tau_{10} > y) = (e^{-0.01y})^5 (e^{-0.005y})^5 = e^{-0.075y}.$$

Therefore ξ is an exponential variable with $\lambda = 0.075$, and

$$\mathbb{E}\xi = \frac{1}{0.075} = 13.33.$$

Thus it takes an average of about 13.33 hours for the first light bulb to burn out.

6)

Consider the mathematical fact that the expected number of coins in order to get 100 consective heads is 2^101 while on the other hand, the problem only has 2^30 throws. Therefore, the probability is less than 0.01%.

We can also think of this problem as a test of random sequence of Heads and Tails. The expected consecutive heads or tail is 30 and the variance is less than 2. Therefore, the probability is less than 0.01%

7)

This solution is motivated by the "N points on a circle" problem from Xinfeng Zhou's book, which happens to be a generalization of problem 1 in this quiz. Consider a circle with the same circumfence as the length of the stick. Cutting the stick into N random pieces is equivalent to cutting the circle into N random arcs with N random points. The necessary and sufficient condition for the N random arcs to form an N sided polygon is that none of the arcs is more than half the circumfence of the circle. If such a condition is violated, then it must be the case that all the N random points are within a semicircle. According to the conclusion from problem 1, we know that the probability that all the N random points are within a semicircle is

$$\frac{N}{2^{N-1}}.$$

Therefore the probability that N random pieces can form an N sided polygon is

$$1 - \frac{N}{2^{N-1}}.$$

8)

The answer to this question depends on what we want. Distributions of daily and monthly stock returns are only approximately symmetric around their means, but the tails are fatter (i.e., there are more extreme values) than what would be expected from normal distributions, so the price moving is not so normally distributed. So, moving median would be a better choice if we want to remove the influence of extreme values. On the other hand, if we are really want to capture the trend of the price, then moving average would be a better choice, since it can describe the one-sided (either keep increasing or keep decreasing) trend of the price better than moving median could.

9)

This exact solution to this problem is $\frac{n+1}{3}$. We can treat the position (rather than the number) as the key and we take expecation of the sum of each position to be local max. For the two end positions, the probability of local maxima is 1/2. For all the intermediary positions, the probability of local maxima is 1/3. Therefore the average number of local maxima is $\frac{1}{2} * 2 + \frac{1}{3} * (n-2) = \frac{n+1}{3}$.

Programming Question

10)

```
bool isPowerOfTwo(int n) {
    // Power of two
    return n > 0 && (n&(n-1)) == 0;
}
```

11)

```
class SmartPtr {
    int *ptr; // Actual pointer
public:
    // Constructor: refer http://www.geeksforgeeks.org/g-fact-93/
    // for use of explicit keyword
    explicit SmartPtr(int *p = NULL) { ptr = p; }

    // Destructor
    ~SmartPtr() { delete(ptr); }

    // Overloading dereferencing operator
    int &operator *() { return *ptr; }
};
```

12)

```
#include <iostream>

using namespace std;

class ListNode {
public:
    int Data;
    ListNode* NodePtr;
};
```

```
ListNode* ReverseList(ListNode* head, int m, int n) {
    // We assume m<n. If the list has less than n nodes, we throw an exception.
    if (m < 0 || n < 0 || head == NULL) throw "Position not available.";

    ListNode* temp; //Pointer to current node.
    ListNode* temp1; //Pointer to prevoius node.
    ListNode* temp2; //Pointer to next node.

    if (m == 0) {
        temp  = head;
        temp1 = NULL;
        temp2 = head;
    }
    else {
        temp=head;
        for (int i=0; i<m-1; i++) {
            if (temp->NodePtr == NULL) throw "Position not available.";
            else temp = temp->NodePtr;
        }
        temp1 = temp;
        temp  = temp->NodePtr;
        temp2 = temp;
    }

    ListNode* FrontMark=temp1; // Mark the (m-1)-st node, if m==0, FrontMark=NULL.
    ListNode* Markm=temp;   // Mark the m-th node.

    for (int i=0; i<=n-m;i++) {
        if (temp == NULL) throw "Position not available.";
        else {
            temp2 = temp->NodePtr;
            temp->NodePtr = temp1;
            temp1 = temp;
            temp  = temp2;
        }
    }

    Markm->NodePtr = temp;

    if (FrontMark == NULL) return temp1;
    else {
        FrontMark->NodePtr = temp1;
        return head;
    }
}


int main(int argc, const char * argv[]) {
    ListNode* head;
    ListNode* temp;

    head = new ListNode;
    temp = head;

    for (int i=0; i<100; i++) {
        //set up a test liked list from node 0 to 100.
        temp->NodePtr = new ListNode;
        temp->Data = i;
        temp = temp->NodePtr;
    }
    temp->Data = 100;
    temp->NodePtr = NULL;
    temp = head;

    temp = ReverseList(head, 5,18); //test the reverse function

    while (temp != NULL) {
        cout << temp->Data << '\t';
        temp = temp->NodePtr;
    }

    return 0;
}
```

13)

```
#include <iostream>
#include <queue>
#include <vector>

using namespace std;

bool isAvgLow(vector<double>& prices, int M, double P) {
    // Construct a heap so that we can access min in O(1)
    priority_queue<double,vector<double>,greater<double>> aHeap;
    for (int i = 0; i < prices.size(); i++) aHeap.push(prices[i]);

    // Calculate the average of M smallest closing prices
    double target = 0.0;
    for (int i = 1; i <= M; i++) {
        double price = aHeap.top();
        aHeap.pop();
        target = target * (1-1.0/i) + price/i;
    }

    return target <= P;
}
```

14)

```
def IndexOf(strings):
    strind,res = dict(), dict()
    for i in range(len(strings)):
        st = strings[i]
        if st not in strind:
            strind[st] = str(i)
        else:
            strind[st] = strind[st]+' '+str(i)

    for st, ind in strind.iteritems():
        if len(ind.split()) > 1:
            res[st] = ind
    return res

strings = ['a','a','a','b','b','c','c','d','e']
index  = IndexOf(strings)
```

```
print index
```

15)

```cpp
#include <iostream>
#include <stdio.h>

using namespace std;

float exponential(int n, float x);

int main() {
    int n;
    float x;

    printf("Set number of terms n and power x \n");
    cin >> n >> x;
    cout << "n=" << n << endl;
    cout << "x=" << x << endl;

    printf("Approximate value of e^x is: %f",exponential(n,x));
    return 0;
}

float exponential(int n, float x) {
    int i;
    float sum = 1.0f;
    for(i=n; i>=1; i--) {
        sum = 1+sum*x/i;
    }
    return sum;
}
```

16)

```python
def medianII(self, nums):
    import heapq
    if not nums or len(nums) == 0: return None
    maxheap, minheap = [], []
    median = nums[0];
    res = [median]
    for i in range(1, len(nums)):
        if nums[i] < median:
            heapq.heappush(maxheap, -nums[i])
        else:
            heapq.heappush(minheap, nums[i])
        if len(maxheap) > len(minheap):
            heapq.heappush(minheap, median)
            median =- heapq.heappop(maxheap)
        elif len(maxheap)+1 < len(minheap):
            heapq.heappush(maxheap, -median)
            median=heapq.heappop(minheap)

        res.append(median)
    return res
```

17)

```cpp
int maxProfit(vector<int>& prices) {
    int len = prices.size();
    if (len <= 1) return 0;

    int max_so_far = 0;
    for (int i = 1; i < len; i++) {
        max_so_far += max(prices[i]-prices[i-1],0); // Greedy
    }

    return max_so_far;
}
```