Qishi Quiz 1 (Group D)

* Math

Question 1. What is the probability that three points on a circle will be on a semi-circle?

Solution:

Let us place each point A, B, and C on the successively on the circle. For the first placement, A can be anywhere. After A has been placed, there are two semicircles, divided by the line going through A and the center of the circle. The placement of B and C can be on both sides of the circle with equal probability 1/2, and the probability for B and C on the same semicircle is therefore 1/4.
Since we can place either A or B or C at the very first, the probability of these three points on the circle is therefore 3/4.

Question 2. An ant walks randomly on the edges of a cube. It starts from a vertex, and each step it has equal probability to choose one of the three edges and walk to the other vertex of this edge.

What is the expectation of the number of steps for the ant to walk from one vertex to the opposite vertex?

Solution:

Let f(n) denotes the expected number of steps needed. Here n is the number of steps the starting position is away from the destination. Therefore, once we place the ant on a vertex, this vertex is vertex 3 and the destination will be 0. Using the conditional expectation, we have

$$f(3) = 1 + f(2)$$

$$f(2) = \frac{2}{3}f(1) + \frac{1}{3}f(3)$$

$$f(1) = 1 + \frac{1}{3}f(0) + \frac{2}{3}f(2)$$

$f(0) = 0$, solving this set of equations, yielding

$$f(3) = 10$$

Question 3. From a deck of 52 cards, you can pick one card each time without replacement. If the card color is black, you win 1$. If the card color is red, you lose 1$. You can stop the game whenever you want. Questions: Will you play the game? If you want, how much would you pay to play this game?

Solution:

This question resembles the American option pricing. Denote (b, r) denotes the number of blue and red cards left in the deck, respectively. At each (b, r), we face the decision whether to sop or keep playing. If we stop, the payoff is r - b. If we keep going, there is b/(b+r) probability that the next card will be black, and we switch to the state (b - 1; r), and r/(b+r) probability that the next card will be black, and we switch to the state (b, r - 1). We will stop if the expected payoff of drawing more cards is less than r - b. This gives us the system equation

$$\mathbb{E}[f(b,r] = max\left\{r-b, \frac{b}{b+r}\mathbb{E}[f(b-1,r)] + \frac{r}{b+r}\mathbb{E}[f(b,r-1)]\right\} \qquad (*)$$

Obviously, the boundary conditions are

$$f(0,r) = r \quad \forall r = 0,1,\dots,26$$
$$f(b,0) = 0 \quad \forall b = 0,1,\dots,26$$

Running the system equation (*) with the boundary conditions, we get the expected payoff at the beginning of the game is

$$\mathbb{E}[f(26,26] = \$2.62$$

i.e., we would pay 2.62 dollar for this game.

Question 4. Given a coin with probability p of landing on heads after a flip, what is the probability that the number of heads will ever equal the number of tails assuming an in finite number of flips?

Solution:

Question 5. You have ten light bulbs. Five have an average life of 100 hours, and the other five have an average life of 200 hours. These light bulbs have a memoryless property in that their current age (measured in how long they have already been on) has no bearing on their future life expectancy. Assuming they are all already on what is the expected number of hours before the first one burns out?

Solution:

Since the life of the light bulbs has memoryless property, we assume it follows the exponential distribution, and denote X_i, i = 1,2,…,5 be i.i.d. exponential distribution with $\lambda = \frac{1}{100}$ and Y_i, i = 1,2,,…,5 be i.i.d. exponential distribution with $\lambda = \frac{1}{200}$. We want to compute

$$\mathbb{E}[\min(X_1, \dots, X_5, Y_1, .. Y_5)]$$

Denote $Z = \min(X_1, \dots, X_5, Y_1, .. Y_5)$, its pmf can be computed as

$$\mathbb{P}(Z \le z) = 1 - \mathbb{P}(Z > z) = 1 - \prod_{i=1}^{5} \mathbb{P}(X_i > z)\mathbb{P}(Y_i > z)$$

$$= 1 - \left(\int_z^\infty \frac{1}{100} e^{-\frac{1}{100}t} dt\right)\left(\int_z^\infty \frac{1}{200} e^{-\frac{1}{200}t} dt\right) = 1 - e^{-\frac{3}{40}t}$$

Therefore the pdf of Z is

$$f(z) = \frac{d}{dz}\mathbb{P}(Z \le z) = \frac{3}{40} e^{-\frac{3}{40}z}$$

The expectation can be computed as

$$\mathbb{E}[\min(X_1, \dots, X_5, Y_1, .. Y_5)] = \int_0^\infty z f(z) dz = \int_0^\infty z \frac{3}{40} e^{-\frac{3}{40}z} dz = \frac{40}{3}$$

Question 6. If a person tosses a coin once per second and he tosses 100 years, ask whether the following statement is correct or not: the probability of tossing 100 consecutive heads is less than 0.01%.

Solution:

Here, we can consider it as a combination problem. Suppose we want to calculate the probability of having exactly 100 consecutive heads. Let n be the total tossing number. The number of situations we have exactly 100 consecutive heads is

$$M = 2 \times 2^{n-100-1} + C_{n-100-1}^{1} \times 2^{n-100-1-2}$$

The first part are the situations that the 100 consecutive heads are at the head and end of the total tossing. The second part is the rest case.

The total number of possibility for the tossing is

$$N = 2^n$$

The total tossing number is $n = 60 \times 60 \times 24 \times 365 \times 100$

So the probability of having 100 consecutive heads is

$$\frac{M}{N} \approx \frac{10^9}{2^{103}} < \frac{1}{10000}$$

The statement is correct.

Question 7. Given a stick, if randomly cut into N pieces, what's the probability that the N pieces can form an N sided polygon?

Solution:

If we randomly cut a stick into N pieces and denote the length of each piece as X_i, i = 1,…,N. Without loss of generality, we assume the length of the stick to be 1, therefore

$$\sum_{i=1}^{N} X_i = 1$$

In order for this N pieces to form a polygon, we must have

$$X_i < \frac{1}{2} \quad \forall i = 1,2, \dots, N$$

This reminds us of Question 1. The problem is then equivalent to that not all N points are on the same semicircle of a circle with the circumference of 1. Following the same convention as of question, we calculate the probability that all N points are on the same semicircle. Given a cut point i, we define an event Ei as the next N - 1) point are in the clockwise semicircle and $P(E_i) = \frac{1}{2^{N-1}}$. Thus the probability of all N points are on the same clockwise semicircle is

$$\mathbb{P}\left(\bigcup_{i=1}^{N} E_i\right) = \bigcup_{i=1}^{N} \mathbb{P}(E_i) = \frac{N}{2^{N-1}}$$

Since E_i is mutually exclusive because there must be a point when we start from j that has a distance to the point starting from i greater than a clockwise semicircle when i ≠ j. Thus the probability to be calculated is

$$\mathbb{P} = 1 - \frac{N}{2^{N-1}}$$

Question 8. Suppose in a trading environment, to describe 20 mins prices movement, should we choose moving median or moving average? Why?

Solution:

Question 9. What is the average number of local maxima of a permutation of 1,…, n, over all permutations? Maxima at ends also count. (Putnam problem.)

Solution:

Let S be the number of local maxima of a permutation of 1,…,n, then we have

$$S = \mathbb{I}_1 + \cdots + \mathbb{I}_n,$$

where $\mathbb{I}_i$ is the indicator variable that represents whether the *i*th number in a permutation is a local maxima or not ($\mathbb{I}i = 1$ if it is a local maxima and $\mathbb{I}i = 0$ if it is not).

By linearity of expectation, we have

$$E(S) = E(\mathbb{I}_1) + \cdots + E(\mathbb{I}_n) = p(\mathbb{I}_1 = 1) + \cdots + p(\mathbb{I}_n = 1),$$

where $p(\mathbb{I}_i = 1)$ is the probability that the *i*th number in a permutation is a local maxima. When i = 1 or n, $p(\mathbb{I}_i = 1) = ½$, because for the pair of numbers $\{\pi(1), \pi(2)\}$ randomly picked from 1,…,n, it is equally likely that $\pi(1)$ or $\pi(2)$ is bigger (any pair can be switched to achieve the other outcom). When $1 < i < n$, $p(\mathbb{I}_1 = 1) = 1/3$, because for the permutation $\{\pi(1), \pi(2), \pi(3)\}$, it is equally likely that any of the three is the largest. Hence, the average number of local maxima is

$$E(S) = 2 \times \frac{1}{2} + (n-2) \times \frac{1}{3} = \frac{n+1}{3}.$$

* Programming

Question 10. Give a one-line C expression to test whether a number is a power of 2.

Solution:

// This line will return a bool (c++) value

return x != 0 && x != INT_MIN && (x & (x - 1)) == 0;

Question 11. Implement a smart pointer in C++.

Solution:

A smart pointer is an object that stores a pointer to a heap allocated object. If you use a smart pointer correctly, you no longer have to remember when to delete the new created memory.

Reference: “Cracking the Coding Interview”

```
template <class X>
  class Smart_pointer
  {
        public:
          // constructor
           Smart_pointer(X* ptr)
           {
                ref = ptr;
                ref_count = (unsigned*)malloc(sizeof(unsigned));
                *ref_count = 1;
           }
          //copy constructor
           Smart_pointer(Smart_pointer<X> & sptr)
           {
                ref = sptr.ref;
                ref_count = sptr.ref_count;
                ++*ref_count ;
           }
           //destructor
           ~Smart_pointer()
          {
```

```
                --*ref_count;
                if(*ref_count == 0)
                {
                    delete ref;
                    free(ref_count);
                    ref = NULL;
                    ref_count = NULL;
                }
            }
        protected:
            T* ref;
            unsigned* ref_count;
    };
```

Question 12. Reverse a linked list from position m to n. Do it in-place and in one-pass.

Solution:

```
class ListNode {
  public:
    int value;
    ListNode* next;
   ListNode(int v) : value(v), next(NULL) {}
};
class Solution {    // Assuming the position is counted from 1
    public:
        ListNode *reverseBetween(ListNode *head, int m, int n) {
```

```
        ListNode* newHead = new ListNode(-1);
        newHead->next = head;
        ListNode* prev = newHead;
        for(auto i = 0 ; i < m  - 1; i++){
            prev = prev->next;
        }
        ListNode* const reversedPrev = prev;
        //position m
        prev = prev->next;
        ListNode* cur = prev->next;
        for(auto i = m ; i < n ; i++){
            prev->next = cur->next;
            cur->next = reversedPrev->next;
            reversedPrev->next = cur;
            cur = prev->next;
        }
        return newHead->next;
    }
};
```

Question 13. Implement a program to find out whether there exist M days within the last N(N >= M) trading days that the average closing price of these M days is at most P. Assume we have collected the history of the closing prices of the last N trading days for a stock. Requirements:

Inputs are positive integer M and N, M <= N; An array of N float elements containing the closing prices of the last N trading days; And a float P. Please design and implement the program in C, C++, Java or Python to produce the answer in most time/space efficient way.

Solution:

Assuming M elements are continuous. Time complexity is O(N). Space complexity is O(1).

```
public class Solution {    // Moving average problem
        // return the starting index if this M days exsit, otherwise return -1
        public static int checkaverage(int[] prices, int N, int M, double P)
        {
                double target_sum = P * M;
                int sum = 0;
                boolean foundit = false;
                int i =0;
                for(; i < M; i++)
                {
                        sum += prices[i];
                }
                while(i < N)
                {
                        if(sum <= target_sum)
                        {
                                foundit = true;
                                break;
                        }
                        sum = sum + prices[i] - prices[i - M];
                        i++;
                }
```

```
                return foundit? i - M:-1;
        }
}
```

Question 14. Implement a string indexOf method that returns index of matching string.

Solution:

```
// Sundy algorithm: faster than KMP and BM algorithm. Linear time complexity O(M+N) , in java
  public static  HashMap<Character, Integer> buildindex(String pattern)
 {
        HashMap<Character, Integer> position = new HashMap<Character, Integer>();
        for(int i = pattern.size() – 1; i >=0; i--)
        {
              if (!position.contaisKey(pattern.charAt(i)))
                    position.put(pattern.charAt(i), i);
         }
        return position;
 }


public static int indexOf(String pattern, String text)
{
       HashMap<Character, Integer> position = buildindex(pattern);
        if(pattern == null || text == null) return -1;
        pLen = pattern.size();
        tLen = text.size();
        if(tLen < pLen) return -1;
```

```
        int I = 1;
        int j = 1;
        while(i < pLen && j < tLen)
       {
            if(pattern.charAt(i) == text.charAt(j))
           {
                  i++;
                  j++;
           }
           else
          {
                char check = text.charAt(j + pLen);
                int pos;
                if(position.containsKey(check))
                    pos = position.get(check);
                else
                     pos = -1;
                i = 0;
                j +=  pLen – pos;
          }
        }
        if(i == pLen) return j-i;
        else return -1;
}
```

Question 15. Write a function to calculate exp(x).

Solution:

$$e^x = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots + \frac{x^n}{n!}$$

```
// C++

double exp(double x) { // x is the power exponent, delta is the practical limit for 0

    double term = 1;  // first term

    double sum = 0;  // initial value

    for (int i = 1; sum != sum + term; ++i) {

        sum = sum + term;

        term = term*x/i;

    }

    return exp;

}
```

Question 16. Given streaming data, design an algorithm to get approximate median of all previous data, use constant memory.

Solution:

```
//java

public class Solution {

        public static int getMedian(int new_elem,int initialmedian, PriorityQueue<Integer> left, PriorityQueue<Integer> right)

        {

                if(left.size() == right.size())
```

```
{
        int result;
        if(new_elem < initialmedian)
        {
                left.add(new_elem);
                result = left.peek();
        }
        else
        {
                right.add(new_elem);
                result = right.peek();
        }
        return result;
}
else if(left.size() > right.size())
{
        if(new_elem < initialmedian)
        {
                right.add(left.poll());
                left.add(new_elem);
        }
        else
        {
                right.add(new_elem);
```

```
                }
                return (left.peek() + right.peek()) / 2;
        }
        else
        {
                if(new_elem < initialmedian)
                {
                        left.add(new_elem);
                }
                else
                {
                        left.add(right.poll());
                        right.add(new_elem);
                }
                return (left.peek() + right.peek()) / 2;
        }
}

public static void main(String[] args)
{
          int initialmedian = 0;
              int[] prices = {3,6,7,9,3,2,4,8,9};

             PriorityQueue<Integer> left = new PriorityQueue<Integer>((x, y) -> y - x);
```

```
                PriorityQueue<Integer> right = new PriorityQueue<Integer>();


                for(int i = 0; i < prices.length; i++)
                {
                    initialmedian = getMedian(prices[i], initialmedian, left, right);
                    System.out.println(initialmedian);
                }
        }
}
```

Question 17. Say you have an array for which the i-th element is the price of a given stock on day i.

Design an algorithm to find the maximum profit. You may complete as many transactions as you like (i.e. buy one and sell one share of the stock multiple times). However, you may not engage in multiple transactions at the same time (i.e. you must sell the stock before you buy again).

Solution:

```
// c++
class Solution {
  public:
    int maxProfit(vector<int> &prices) {
        int len = prices.size();
        int profit = 0;
        for(int i = 0; i < len; i++)
        {
            if (prices[i + 1] > prices[i])
```

```
                profit += prices[i+1] – price[i];
        }
        return profit;
    }
};
```