Qishi Quiz 1

Tentative Solutions for 1, 2, 11, 14

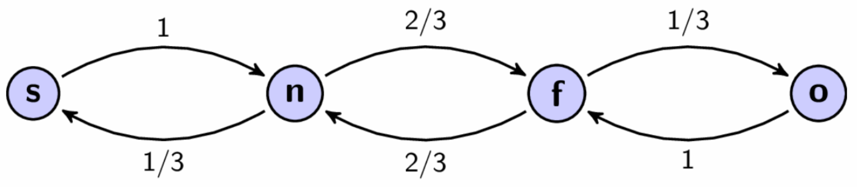
1 Problem 1

Suppose that point i has angle 0, which means that point i is the "first" or "leading" point in the semicircle. Then we want the event that all of the points are in the same semicircle – i.e., that the remaining points end up all in the upper half-plane.

That's a coin-flip for each remaining point, so you end up with $1/2^{n-1}$. There's n points, and the event that any point i is the "leading" point is disjoint from the event that any other point j is, so the final probability is $n/2^{n1}$.

2 Problem 2

A random walk on the cube has eight states, but by symmetry we can reduce this to four states: start, nearest neighbors, further neighbors, and opposite. The "boundary" is the single state o. Define h as the expected time to hit the boundary starting at x. Then we have the


$$\begin{tikzcd} s \arrow[r, bend left, "1"] & n \arrow[l, bend left, "1/3"] \arrow[r, bend left, "2/3"] & f \arrow[l, bend left, "2/3"] \arrow[r, bend left, "1/3"] & o \arrow[l, bend left, "1"] \end{tikzcd}$$

relationship

$$\begin{aligned} h(s) &= 1 + h(n) \\ h(n) &= 1 + \frac{1}{3}h(s) + \frac{2}{3}h(f) \\ h(f) &= 1 + \frac{1}{3}0 + \frac{2}{3}h(n) \end{aligned}$$

which gives us $h(o) = 10$.

3 Problem 11

```
template<typename T>
class smart_pointer{
    T* pointer;
    std::size_t *refs;

    void clear(){
        if (!--*refs){
            delete pointer;
            delete refs;
        }
    }
```

```
public:
    smart_pointer(T* p = NULL)
        : pointer(p), refs(new std::size_t(1))
    {}
    smart_pointer(const smart_pointer<T>& other)
        : pointer(other.pointer), refs(other.refs)
    {
        ++*refs;
    }
    ~smart_pointer(){
        clear();
    }

    smart_pointer<T>& operator=(const smart_pointer<T>& other){
        if (this != &other){
            clear();

            pointer = other.pointer;
            refs    = other.refs;
            ++*refs;
        }
        return *this;
    }

    smart_pointer<T>& operator=(T* p){
        if (pointer != p){
            pointer = p;
            *refs = 1;
        }
        return *this;
    }

    T& operator*(){
        return *pointer;
    }

    const T& operator*() const{
        return *pointer;
    }

    T* operator->(){
        return pointer;
    }

    const T* operator->() const{
        return pointer;
    }

    std::size_t getCounts(){
        return *refs;
    }
};
```

4 Problem 14

```
void preKMP(string pattern, int f[])
{
    int m = pattern.length(), k;
    f[0] = -1;
    for (int i = 1; i < m; i++)
    {
        k = f[i - 1];
        while (k >= 0)
        {
            if (pattern[k] == pattern[i - 1])
                break;
            else
                k = f[k];
        }
        f[i] = k + 1;
    }
}

//check whether target string contains pattern
bool KMP(string pattern, string target)
{
    int m = pattern.length();
    int n = target.length();
    int f[m];
    preKMP(pattern, f);
    int i = 0;
    int k = 0;
    while (i < n)
    {
        if (k == -1)
        {
            i++;
            k = 0;
        }
        else if (target[i] == pattern[k])
        {
            i++;
```

```
                k++;
                if (k == m)
                    return i-m;
            }
            else
                k = f[k];
        }
        return -1;
    }
```