Math

1. The probability can be written as
2. We have four different types of vertex:

Tier zero: the initial one;

Tier one: the ones that are one step away;

Tier two: two steps away and

Tier three: the final one.

The expected steps from tier zero to the tier three should be one step more than the c expected steps from the tier one to tier three:

The expected steps from tier one to tier three should be

And the expected steps from tier two to tier three should be

Solve the set of equations above we can get

1. I will play this game since I cannot loss. It is always possible to play all the cards and get zero profit.

The expected profit from first draw is

The expected profit from second draw is

The expected profits from every draw are all zero so the safest strategy will be pay nothing to play this game.

1. Consider this problem as a random walk in 1D. Let’s start from zero, move one step right if we got a head and move left if a tail. The probability of getting head is p.

If we got a head from the first flip, then the probability of getting back by one step from position 1 is P(1), it should satisfy the following equation:

Where is the probability of getting back to zero from position 2. Getting back from position 2 to zero requires two steps, first we need to move back to position 1 then from 1 to zero. The probability of first step is essentially equal to and the probability of second step is obviously also . Therefore:

Solve for P(1) we can get:

If we got a tail from first flip, then following similar argument, the probability of getting back from position -1 is

Hence the probability of getting back to zero from zero is

1. The probability of a light bulb dies in specific hour is a geometric distribution. Namely

And this distribution has the expected value of x equal to

Therefore there are five light bulbs have , and five have .

Hence the total probability that at least one of light bulb dies in some specific hour given all of them didn’t die in previous hour is

And the expected hour for all the light bulbs survive is

1. Consider the problem this way. If we toss the coin once per second for 100 years the total number of tossing is N=86400\*100, the total number of outcomes is . If we want at least 100 consecutive heads, it can only happen at N-100 places, namely if we get the first head at N-99 place then there is no way we can get 100 consecutive heads after that. For each consecutive 100 heads, the other tosses can have outcomes.

Therefore the probability of getting 100 consecutive heads for N tosses is

hence , therefore .

1. This is basically asking what is the probability of all breaking points fall into the same half of the stick. For N pieces we need N-1 breaking point. The position of first point does not matter. So the probability of all the other N-2 points fall into same half of the stick is . Hence the probability of break a stick into N pieces and form a N sided polygon is
2. I’d use moving average because from my experience the moving median usually has larger volatility than moving average in short time interval.
3. The expectation of number of local maxima can be calculated as the sum of probability of having local maximum at position k over k=1,…,n. The probability of having a local maximum at the end points is ½. The probabilities of having local maxima at points between the end points are 1/3.

Therefore the average number of local maxima over all permutation is

1. return !(a&(a-1))&&(a!=0);
2. 抄来的



1. **struct** LListNode\* reversemn(**struct** LListNode \* head, **int** m, **int** n) {
2. **int** icurrent=1;
3. **struct** LListNode \* curNode=head,\*firstNode,\*nextNode, \*preNode;
4. **if**(m==n)
5. {
6. **return** head;
7. }
8. **else**
9. {
10. firstNode=head;  //s
11. **while**(icurrent<m)
12. {
13. firstNode=curNode;
14. curNode=curNode->next;
15. icurrent++;
16. }
17. preNode=firstNode;
18. **while**(icurrent<n)
19. {
20. nextNode=curNode->next;
21. curNode->next=preNode;
22. preNode=curNode;
23. curNode=nextNode;
24. icurrent++;
25. }
26. nextNode=curNode->next;
27. firstNode->next->next=nextNode;
28. curNode->next=preNode;
29. **if**(m>1)
30. {
31. firstNode->next=curNode;
32. }
33. **else**
34. {
35. head=curNode;
36. }
37. **return** head;
39. }
40. }
41. Calculate the moving average in N time complexity;

|  |
| --- |
| 1. **bool** MA(**float**\* data, **int** M, **int** N, **float** P) 2. { 3. **int** i, j; 4. **float** average = 0; 5. **for** (i = 0; i<M; ++i) 6. { 7. average += data[i]; 8. } 9. average = average / M; 10. **for** (j = M; j<N; ++j) 11. { 12. **if** (average <= P) **return** **true**; 13. average = average\*M; 14. average -= data[j - M]; 15. average += data[j]; 16. average = average / M; 17. } 18. **return** **false**; 19. } |

1. Brutal force string search.

|  |
| --- |
| 1. **int** ioSubstr(**char**\* haystack, **char**\* needle) { 2. **int** i, j; 3. **if** (strlen(haystack) < strlen(needle)) **return** -1; 4. **for** (i = 0; i < strlen(haystack) - strlen(needle)+1; i++) 5. { 6. **for** (j = 0; j < strlen(needle); j++) 7. { 8. **if** (haystack[i + j] != needle[j]) 9. { 10. **break**; 11. } 12. } 13. **if** (j == strlen(needle)) 14. **return** i; 15. } 16. **return** -1; 17. } |

1. Using Taylor expansion: . Stop when difference between n+1 and n equal to the limit of double data type.

|  |
| --- |
| 1. **double** expx(**double** x) 2. { 3. **int** i=1; 4. **int** power=1; 5. **while** (x > 1||x<-1) 6. { 7. x = x / 10;  //reduce the x to the range of [-1,1] 8. Power\*=10; 9. } 10. **double** result = 0; 11. **double** nplus1 = 1; 13. **while** (((nplus1 > numeric\_limits<**double**>::min()) && (nplus1<numeric\_limits<**double**>::max())) || 14. ((nplus1 < -numeric\_limits<**double**>::min()) && (nplus1>-numeric\_limits<**double**>::max()))) 15. { 16. nplus1 = nplus1\*x / i; 17. result += nplus1; 18. i++; 19. } 20. **return** pow(result + 1, power)); 21. } |

1. The idea is to bin the data and only record histogram. The number of bin can be chosen so that it guarantees to cover the whole range of data and the step of the bin can be chosen so that it meets the requirement on resolution.

|  |
| --- |
| 1. **void** updateBin(vector<**int**>\* bins, **float** data, **float**\* endoBin,**float** step) 2. { 3. **int** i=0;//aditional bins needed to be added. 4. **if** (data > \*endoBin) 5. { 6. i = (data - \*endoBin) / step; 7. \*endoBin = i>0?data:\*endoBin; 8. **while** (i > 0) 9. { 10. (\*bins).push\_back(0); 11. i--; 12. } 13. (\*bins).back()++; 14. } 15. **else** 16. { 17. **int** mid, upper, lower; 18. lower = 0; 19. upper = (\*bins).size(); 20. mid = (upper + lower) / 2; 22. **while** (upper - lower > 1) 23. { 24. lower = data >= mid\*step ? mid : lower; 25. upper = data > mid\*step ? upper : mid; 26. mid = (upper + lower) / 2; 27. } 28. (\*bins)[lower]++; 29. } 30. } 31. **float** binMedian(**float** data, vector<**int**>\* bins, **float** step,**int** n) 32. { 33. **int** i=0; 34. **int** count = 0; 35. **while** (count < n / 2) 36. { 37. count += (\*bins)[i]; 38. i++; 39. } 40. cout << "The approximate median of first "<<n<<" item is "<<(i-1)\*step<<endl; 41. **return** 0; 42. } |

1. Since we know the “future” in this problem. We can always avoid the loss. Namely we can follow the profiting trend as long as possible then close the position. If we are allowed to short then the max profit will be the sum of absolute values of all differences between two consecutive prices. If we are not allowed to short then the max profit will be the sum of all positive differences.

|  |
| --- |
| 1. **int** maxProfit(**int** prices[], **int** n){ 2. **int** profit = 0; 3. **for**(**int** i=0;i<n-1;i++){ 4. profit += prices[i+1]-prices[i]>0? prices[i+1]-prices[i]:0; 5. } 6. **return** profit; 7. } |