Solution 2

Pengfei Yang

1) There are two types of χ^2 test. First one is the Pearson's χ^2 test. This is one of goodness-of-fit tests mainly to handle the discrete distribution. Pearson's χ^2 test is used to test whether a given set of samples follow a specific discrete distribution P. If the number of samples is N, the test statistic is

$$\sum_{x\in\mathcal{X}} \frac{(N\hat{p}_x - Np_x)^2}{Np_x}$$

where $\hat{p}_x$ is the empirical distribution. The sum above asmptotically approaches χ^2 distribution with degree $|\mathcal{X}| - p$ as N goes to infinity, where $p = s + 1$, s is the number of parameters used to characterize the distribution P.

Assume P is discrete uniform distribution on $1 \sim k$. Assume N is large, denote N_1 as the number of 1s ($N\hat{p}_1$ is a realization of N_1) which has binomial distribution $B(N, 1/k) \sim Poisson(N/k) \sim N(N/k, N/k)$. Then the above sum approximates to sum of square of independent Gaussian, which is the χ^2 distribution.

2) 19.

3) $f(X = x|X + Y > 0) = \frac{f(X=x, Y>-x)}{P(Y>-X)} = \frac{f(X=x)P(Y>-x)}{0.5} = 2f(X = x)\Phi(x)$

4) (1). It depends on the individual's risk attitude. If not stop, we could keep rolling until the sum exceed 42. For this strategy, the possible amount we receive could be 0 or any number between 43 and 48. This strategy has non-zero variance (risk), its expectation would be greater than $\frac{1}{6} * 0 + \frac{5}{6} * 43 > 35$, where 35 is the amount we receive riskless if stop. Thus if not stop, there exists a strategy with higher return and higher risk. It really depends on individual's risk attituede. The relationship is like stock and riskless bond.

(2). 44. If start from 35 and keep rolling until sum exceed 43, possible amount received, which is denoted as X, could be 0, or anynumber from 44 to 48. For $44 \le k \le 48$,

$$P(X = k) = \sum_{i=1}^{6} P(\text{final sum minus last time roll is } k-i)\frac{1}{6} I_{\{k-i\le 43 \text{ and k-i is not square number}\}} \quad (1)$$

$$= \sum_{i=1}^{6} P(\text{start from 35 and keep rolling, sum reaches } k-i)\frac{1}{6} I_{\{k-i\le 43 \text{ and k-i is not square number}\}} \quad (2)$$

For $44 \le k \le 48$, $P(X = k)$ are listed below,

$$0.236, 0.203, 0.165, 0.121, 0.070,$$

$P(X = 0) = 1 - \sum_{i\neq 0} P(X = i) = 0.204$. Thus 44 is the most probable number. The program is made executable online: http://cpp.sh/64bp.

(3). Keep rolling if the sum doesn't exceed 18. Let A_n denote the amount received if we can keep rolling until the sum exceed n.

$$EA_n = \sum_{i=1}^{6}(n+i)P(\text{final sum is n+i} \mid \text{stop once sum is greater than n})I_{\{\text{n+i is not squre number}\}},$$

and we have,

$$P(\text{final sum is k}| \text{ stop once sum is greater than n}) \tag{3}$$

$$= \sum_{i=1}^{6} P(\text{start from 0 and keep rolling, sum reaches k-i})\frac{1}{6}I_{\{0\le k-i\le n \text{ and k-i is not square number}\}}. \tag{4}$$

Similarly, we can calculate $Var(A_n)$.

The following program print EA_n and $Var(A_n)$ for $1 \le n \le 50$;

P(n) calculate all P(start from 0 and keep rolling, sum reaches i) for $1 \le i \le n$;

PG(m,n,vec) returns P(final sum is m| stop once sum is greater than n).

```
// Example program
#include <iostream>
#include <string>
#include <vector>
#include <cmath>

using std::vector;

vector<double> P(int n){
    vector<double> p(n);
    p[0]=1;
    for(int j=1; j<n; ++j){
        if(int(sqrt(j))*int(sqrt(j))==j)
            p[j]=0;
        else{
            for(int i=1; j-i>=0 && (i<=6); ++i){
                p[j]+=p[j-i]*1.0/6;
                }
            }
        }
    return p;
}

double PG(int m, int n, vector<double> &vec){
    double temp=0;
    if(int(sqrt(m))*int(sqrt(m))==m)
        return 0;
    for(int i=1; i<=6; ++i){
        if(m-i<=n && (m-i>=0))
            temp+=vec[m-i]*1.0/6;
    }
    return temp;
}
```

```
int main()
{
  vector<double> vec=P(50);
  for(int i=1; i<vec.size(); ++i){
      double e=0, v=0;
      for(int j=1; j<=6; ++j){
        e+=(i+j)*PG(i+j,i,vec);
      }
      for(int j=1; j<=6; ++j){
        v+=((i+j)-e)*((i+j)-e)*PG(i+j,i,vec);
      }
      std::cout <<"E is "<< e << ", Var is "<<v<< " when n is " << i<<std::endl ;
      }
}
/*Result
E is 2.66667, Var is 2.85185 when n is 1
E is 3.13889, Var is 3.33948 when n is 2
E is 3.39815, Var is 4.17314 when n is 3
E is 3.39815, Var is 4.17314 when n is 4
E is 3.85185, Var is 7.23971 when n is 5
E is 4.38117, Var is 11.571 when n is 6
E is 4.66538, Var is 14.3787 when n is 7
E is 4.99696, Var is 18.3424 when n is 8
E is 4.99696, Var is 18.3424 when n is 9
E is 5.10798, Var is 21.6762 when n is 10
E is 5.23752, Var is 26.5407 when n is 11
E is 5.35713, Var is 31.9271 when n is 12
E is 5.45992, Var is 37.3333 when n is 13
E is 5.56011, Var is 43.366 when n is 14
E is 5.65397, Var is 49.7352 when n is 15
E is 5.65397, Var is 49.7352 when n is 16
E is 6.03616, Var is 56.8327 when n is 17
E is 6.39138, Var is 63.6968 when n is 18
E is 6.32839, Var is 68.5778 when n is 19
E is 6.26861, Var is 73.8487 when n is 20
E is 6.21222, Var is 79.4216 when n is 21
E is 6.15895, Var is 85.2538 when n is 22
...
*/
```

It is clear that 18 is the first local maximum for EA_n, while the $Var(A_n)$ is monotone increasing with n, just as expected. Thus in such a game, we should keep rolling if the sum doesn't exceed 18. The program and result is made executable online: http://cpp.sh/263z.

5) Use order statistics to get the expected length of the smallest and the longest, then use 1 to minus the sum of these two to get the middle one.
6) This can be sovled brutely using dearrangement. ($!n = n! \sum_i (-1)^i / i!$.)

7) When X_1 is noised verstion of X_2, R_3 achieves the minimum at 0.2. When X_1 and X_2 are independent, R_3 can achieve the maximum.
8) http://stackoverflow.com/questions/2040814/given-a-function-for-a-fair-coin-write-a-function-for-a-biased-coin
9) Denote E_i as the expectation accrossed bridges starting from island i, $E_{10} = 0$. Use conditional expectation we can get

$$E_k = 1 + 0.5E_1 + 0.5E_{k+1}.$$

Then we can get

$$E_1 = 2 + E_2, E_2 = \frac{2^8 - 1}{2^7} + \frac{2^8 - 1}{2^8} E_1.$$

Thus we have $E_1 = 1022$.
10) const int* const means the variable is a const pointer to const int, the const at the end means fun is a constant member function which cannot modify member variables. So the code means fun takes the p which is a reference to a constant pointer to a constant integer, fun returns a constant pointer to a constant integer. (http://stackoverflow.com/questions/1143262/what-is-the-difference-between-const-int-const-int-const-and-int-const, http://stackoverflow.com/questions/10716769/c-difference-between-const-positioning, http://stackoverflow.com/que of-const-int-const-funconst-int-const-p-const)
11)
```
class Solution {
public:
    void deleteNode(ListNode* node) {
        auto temp=node->next;
        *node=*node->next;
        if(temp) delete temp;
    }
};
/*
The implicit assignment operator does member-wise assignment of
each data member from the source object. So *node=*next is equavalent to
node->val=next->val;
node->next=next->next;
```
12) ...
13) No, a virtual call is a mechanism to get work done given partial information. In particular, "virtual" allows us to call a function knowing only any interfaces and not the exact type of the object. To create an object you need complete information. In particular, you need to know the exact type of what you want to create. Consequently, a "call to a constructor" cannot be virtual.
14) Yes. https://isocpp.org/wiki/faq/strange-inheritance#calling-virtuals-from-base.
15) DP. Use $f(i, j)$ to denote whether $s[i, j)$ is the palindrome. This might even be reduced to a one demension $f(i)$, so the space complexity is $O(N)$.
16) Use python, first split by whitespace, then reverse the order of all words.
17) Scan the vector, find the maximum of current price minus current minimum.

18) To faciliate the I/O, assume the INPUT is a vector of vector. We can use DFS method. Use s to memorize the visited people, choosing s as a set has two advantages, one is quick search the other is that set is ordered in C++. Once the s contains N unique number and 2 1s at two ends, save s as the result. After DFS, if the size of $result$ is 0 that means no such path exists.

```
void dfs(vector<vector<int>> &vec, set<int> &s, set<int> &result){
    if(*(--s.end())==1){
        if(s.size()==vec.size())
            result=s;  //Once we find the desired path, save it as the result.
        return; //Don't need to go further.
        }
    for(int i=0; i<vec[*(--s.end())].size(); ++i){
        if(s.find(vec[*(--s.end())][i])!=s.end()||vec[*(--s.end())][i]==1){
            s.insert(vec[*(--s.end())][i]);
            dfs(vec, s);
            s.erase((--s.end()));
            }
        }
    }

set<int> path(vector<vector<int>> vec){
    set<int> s, result; //s is a temporary set to record current visited people.
    s.insert(1);
    dfs(vec, s, result);
    return result;
    }
```