概率算法作业

Problem 1

P20. EX

若将$y \leftarrow uniform(0,1)$ 改为$y \leftarrow x$，则算法估计的值是什么？

```
Darts (n) {
    k ← 0;
    for i ← 1 to n do {
        x ← uniform(0, 1);
        y ← uniform(0, 1); // 随机产生点(x,y)
        if (x2 + y2 ≤ 1) then k++; //圆内
    }
    return 4k/n;
}
```

答：

若改为$y \leftarrow x$，则相当于在线段$y = x$（其中，$x, y \in (0,1)$）上取点如果上面的点到原点的长度小于等于1，$k++$。这也表明最终计算的是从原点开始长度为1的线段，然而所取范围的线段总长度为$\sqrt{2}$。故对应计算的值为$\frac{4\times 1}{\sqrt{2}} = 2\sqrt{2}$。

Problem 2

P23. EX2

在机器上用$4\int_0^1 \sqrt{1-x^2}dx$估计π值，给出不同的n值及精度。

答：

```
// P23 EX2
#include<bits/stdc++.h>
using namespace std;
double f(double x) {
    return sqrt(1 - x * x); //需要积分的函数定义
}
double calPi(long long n) {
    double k = 0;
    for (int i = 0; i < n; ++i) {
```

```
        double x = (double)rand() / RAND_MAX; //求得x = uniform(0, 1)
        double y = (double)rand() / RAND_MAX; //求得y = uniform(0, 1)
        if(y <= f(x)) k += 1;
    }
    return 4 * k / n; //求得的面积是Pi的四分之一，因此需要乘以4
}
int main () {
    long long n; //指定算法运行次数
    srand(time(0));
    cin >> n;
    cout << calPi(n) << endl;
    return 0;
}
```

运行结果：（$\pi = 3.141592654$）

n	1×10^5	1×10^7	1×10^9
数值	3.13964	3.14167	3.14153
精度	2	4	5

实验分析：

很容易发现n值越大，给出的π的精度越高。

Problem 3

P23. EX3

设a, b, c和d是实数，且$a \le b, \quad c \le d, \quad f : [a, b] \to [c, d]$是一个连续函数，写一概率算法计算积分：

$$\int_a^b f(x)dx \tag{1}$$

注意，函数的参数是a, b, c, d, n和f，其中f用函数指针实现，请选一连续函数做实验，并给出实验结果。

答：

```
#include <bits/stdc++.h>
using namespace std;
typedef double (*fp)(double x);
double poly1(double x) { // 定义二次函数
    return x * x;
}
double poly2(double x) { // 定义三次函数
    return x * x * x;
}
double poly3(double x) { // 定义四次函数
```

```
    return x * x * x * x;
}
double calInt(double a, double b, double c, double d, long long n, fp f) {//计算积分
    double k = 0;
    double S =(b - a) * d;
    for (int i = 0; i < n; ++i) {
        double x = (double)rand() / RAND_MAX;
        double y = (double)rand() / RAND_MAX;
        x = a + (b - a) * x;
        y = y * b;
        if(y <= f(x)) k += 1;
    }
    return (k / n) * S;
}
map<string, fp> mp = {
    {"poly1", poly1},
    {"poly2", poly2},
    {"poly3", poly3}
};
int main () {
    srand((unsigned long)time(0));
    double a, b, c, d, n;
    cout << "Please Input a, b, c, d, n: " << endl;
    cin >> a >> b >> c >> d >> n;
    string func;
    cout << "Please Input the Function:" << endl;
    cin >> func;
    fp myfunc = mp[func];
    cout << calInt(a, b, c, d, n, myfunc) << endl;
    return 0;
}
```

实验中，对于二次函数、三次函数和四次函数分别进行实验，得到下面的实验结果。（区间均取为$[0,1]$）

函数$y=x^2$，积分值：0. 33333

n	1×10^5	1×10^7	1×10^9
数值	0.33207	0.333169	0.333334
精度	3	4	6

函数$y=x^3$，积分值：0. 25

n	1×10^5	1×10^7	1×10^9
数值	0.24719	0.250007	0.250025
精度	2	6	5

函数$y=x^4$，积分值：0. 2

n	1×10^5	1×10^7	1×10^9
数值	0.19912	0.200017	0.200007
精度	2	5	6

对于该问题，可以看出随着n增大，计算的精度在逐渐增加。对于函数$y = x^3$而言，它的积分值为0.25。最后计算出的结果$n = 1 \times 10^9$的精度仅为5。这是由于计算机内部的计算精度造成的。

Problem 4

P36. EX

用上述算法，估计整数子集$1 \sim n$的大小，并分析n对估计值的影响。

```
SetCount (X) {
    k ← 0; S ← Φ;
    a ← uniform(X);
    do {
        k++;
        S ← S∪{a}; a ← uniform(X);
    } while (a ∉ S)
    return 2 * k * k / π
}
```

答:

对于该问题$c++$由于大数存储受选限制，因此补充$python$版本代码。

```
//c++版本
#include<bits/stdc++.h>
using namespace std;
double pi = acos(-1); //pi的近似值
long long SetCount(long long n) {
    long long k = 0;
    unordered_map<long long, long long> mp;
    long long a = rand() % n + 1;
    while(mp.find(a) == mp.end()) {
        k += 1;
        mp[a] = 1;
        a = rand() % n + 1;
    }
    return 2 * k * k / pi;
}
int main () {
    long long n, m;
    cin >> n;    //估算值n
    cin >> m;    //运行次数m
    double ans = 0;
    srand((int)(time(0)));
```

```
    for(int i = 0; i < m; ++i) ans += SetCount(n);
    cout << ans / m << endl;
    return 0;
}
```

```python
#python版本
import math
import random
import time
def SetCount(n):
    k = 0
    dict ={}
    a = random.randint(1,n)
    while(a not in dict):
        k += 1
        dict.update({a: 1})
        a = random.randint(1,n)
    return 2 * k * k / math.pi;
if __name__=="__main__":
    n = int(input()) #估算值n
    m = int(input()) #运行次数
    ans = 0.0
    for i in range(m):
        ans += SetCount(n)
    ans /= m
    print(ans)
    print(abs(ans - n) / n)
```

n的变化对效果的影响（设定运行次数为10000次取平均）

n	1.00×10^2	1.00×10^3	1.00×10^4	1.00×10^5	1.00×10^6
数值	1.17×10^2	1.25×10^3	1.28×10^4	1.28×10^5	1.26×10^6
相对误差	16.90%	24.68%	27.74%	28.42%	26.32%

从分析可以看出，随着n增大，计算的准确率会先上升后下降，并没有明显的规律。从理论上来讲k的期望值是$\sqrt{n\pi/2}$。因此，n应该越大越好，且得多次运行取平均值。

Problem 5

P54. EX

分析$dlogRH$的工作原理，指出该算法相应的u和v。

```
dlogRH(g, a, p) { // 求logg,pa, a = gx mod p, 求x
    // Sherwood算法
    r ← uniform(0..p-2);
    b ← ModularExponent(g, r, p); //求幂模b=gr mod p
    c ← ba mod p; //((gr modp)(gxmodp))modp=gr+xmodp=c
    y ← log(g,p)c; // 使用确定性算法求logp,g c, y=r+x
    return (y-r) mod (p-1); // 求x
}
```

答：

通过分析，很容易知道$dlogRH$的工作原理是基$Sherwood$算法的思想。该算法思想分为三步：1. 将原实例x转化为随机实例c（基于u）。2. 使用确定性的算法求c的解y。3. 将解y转化为x的解（基于v）。很容易，发现该算法对应的u为
$c = u(r, x) = (g^r \ mod \ p)a = (g^r \ mod \ p)(g^x \ mod \ p) \ mod \ p = g^{r+x} \ mod \ p$。然后使用确定性算法：$y = f(c) = log_{g,p}c$。最后该算法对应的$v$为
$x = v(r, y) = (y - r) \ mod \ (p - 1)$。上面的式子之所以成立基于下面两个公式。

$$log_{g,p}(st \ mod \ p) = (log_{g,p}s + log_{g,p}t) \ mod \ (p-1) \tag{2}$$

$$log_{g,p}(g^r \ mod \ p) = r(0 \le r \le p-2) \tag{3}$$

由公式(2)(3)可以知道：

$$\begin{aligned} x &= log_{g,p}a = log_{g,p}(g^x \ mod \ p) = log_{g,p}(g^{x+r} \ mod \quad p - g^r \ mod \ p) \\ &= (log_{g,p}(g^{x+r}) - log_{g,p}g^r) \ mod \ (p-1) = (y - r) \quad mod \ (p-1) \end{aligned} \tag{4}$$

很容易发现y是可以计算得到的。由下面的公式。

$$\begin{aligned} y &= log_{g,p}(g^{x+r}) \ mod \ (p-1) = (log_{g,p}g^x)(log_{g,p}g^r) \ mod \ (p-1) \\ &= log_{g,p}(g^{x+r} \ mod \ p) = log_{g,p}(g^x \ mod \ p \times g^r \ mod \ p) = log_{g,p}(a \times b \ mod \ p) \end{aligned} \tag{5}$$

如果已知y和生成的r，很容易就可以求得最终的x。

Problem 6

P67. EX

写一$Sherwood$算法C，与算法A, B, D比较，给出实验结果。

答：

```
#include<bits/stdc++.h>
#define MAXN 0x3f3f3f3f //用于生成随机array
using namespace std;
typedef pair<int, int> pii;
const int NN = 10000; //随机生成array大小
const int SN = 100; //用于测试的下标数目
```

```cpp
//生成数组产生val和ptr
vector<int> getVal() {
    vector<int> val(NN);
    for (int i = 0; i < val.size(); ++i) {
        val[i] = i;
    }
  //利用时间实现随机交换
    uniform_int_distribution<unsigned> newr(0, NN - 1);
    default_random_engine usetime(time(0));
    for (int i = 0; i < val.size(); ++i)
    {
        int j = newr(usetime);
        int k = newr(usetime);
        swap(val[i], val[k]);
    }
    return val;
}
vector<int> getPtr(vector<int> val) {
    vector<int> ptr(val.size(), -1);
    auto min_iteration = min_element(val.begin(), val.end());
    int pos = min_iteration - val.begin(); //获得最小元素的位置
    int head = pos;
    while (*min_iteration != MAXN) {
        *min_iteration = MAXN;
        pos = min_iteration - val.begin();
        min_iteration = min_element(val.begin(), val.end());
        ptr[pos] = min_iteration - val.begin();
    }
    ptr[pos] = head;
    return ptr;
}
pii Search(int x, int i, vector<int> val, vector<int> ptr) {
    int count = 1;//统计比较次数
    while (x > val[i]) {
        i = ptr[i];
        ++count;
    }
    return {i, count}; //返回值为查找到的下表和比较的次数
}
//确定性算法A(x), 复杂度O(n)
pii A(int x, vector<int> val, vector<int> ptr) {
    int head = min_element(val.begin(), val.end()) - val.begin();
    return Search(x, head, val, ptr);
}
//确定性算法B(x), 复杂度O(n^{1/2})
pii B(int x, vector<int> val, vector<int> ptr) {
    int L = sqrt(val.size());
    int head = min_element(val.begin(), val.end()) - val.begin();
    int max = val[head];
    int i = head;
    int y = -1;
    for (int j = 0; j < L; ++j) {
        y = val[j];
        if (max < y && y <= x) {
            i = j;
```

```cpp
            max = y;
        }
    }
    return Search(x, i, val, ptr);
}
//概率算法C(x), 复杂度O(n^{1/2}), Sherwood算法
pii C(int x, vector<int> val, vector<int> ptr) {
    int L = sqrt(val.size());
    uniform_int_distribution<int> newr(0, val.size());
    default_random_engine usetime(time(0));
    vector<int> randv;
    for (auto i = 0; i < L; ++i) //随机产生一组下标
        randv.push_back(newr(usetime));
    int i = randv[0];
    int y = -1;
    int max = i;
    for (int j = 0; j < L; ++j) {
        y = val[randv[j]];
        if (max < y && y < x) {
            i = randv[j];
            max = y;
        }
    }
    return Search(x, i, val, ptr);
}
//概率算法D(x),  复杂度O(n)
pii D(int x, vector<int> val, vector<int> ptr) {
    uniform_int_distribution<int> newr(0, val.size());
    default_random_engine usetime(time(0));
    int head = min_element(val.begin(), val.end()) - val.begin();
    int i = newr(usetime); //随机生成一个下标
    int y = val[i]; //进行比较
    if (x < y)
        return Search(x, head, val, ptr);
    else if (x > y)
        return Search(x, ptr[i], val, ptr);
    else
        return {i ,0};
}
vector<int> randVal() {
    uniform_int_distribution<int> newr(0, NN - 1);
    default_random_engine usetime(time(0));
    vector<int> r;
    for (auto i = 0; i < SN; ++i)
        r.push_back(newr(usetime));
    return r;
}
pair<double, double> searchTime(pii(*f)(int , vector<int> , vector<int> ), vector<int> val,vector<int> ptr,vector<int> V) {
    int sumCount = 0;
    double sumTime = 0.0;
    clock_t start, end;
    for (auto i : V) {
        start = clock();
        pair<int, int> p = f(i, val, ptr);
```

```
        end = clock();
        double totaltime=(double)(end - start)/CLOCKS_PER_SEC;
            sumCount += p.second;
        sumTime += totaltime * 1000;
    }
    return {1.0 * sumCount / v.size(), sumTime / v.size()};
}
int main() {
    vector<int> val = getVal();
    vector<int> ptr = getPtr(val);
    vector<int> v = randVal();
    pair<double, double> countNum;

    cout <<"Algo A:" << endl;
    countNum = searchTime(A, val,ptr,v);
    cout << "Average Search Num.:" << countNum.first << endl;
    cout << "Average Search Time.:" << countNum.second << endl;

    cout <<"Algo B:" << endl;
    countNum = searchTime(B, val,ptr,v);
    cout << "Average Search Num.:" << countNum.first << endl;
    cout << "Average Search Time.:" << countNum.second << endl;

    cout <<"Algo C:" << endl;
    countNum = searchTime(C, val,ptr,v);
    cout << "Average Search Num.:" << countNum.first << endl;
    cout << "Average Search Time.:" << countNum.second << endl;

    cout <<"Algo D:" << endl;
    countNum = searchTime(D, val, ptr, v);
    cout << "Average Search Num.:" << countNum.first << endl;
    cout << "Average Search Time.:" << countNum.second << endl;
    return 0;
}
```

、

实验结果（数组规模大小10000，设定运行次数为100次取平均）

$Algorithm$	A	B	C	D
复杂度	$O(n)$	$O(\sqrt{n})$	$O(\sqrt{n})$	$O(n)$
比较次数	4985.91	84	**60.76**	2525.37
运行时间/ms	0.20473	0.15748	**0.03773**	0.18174

可以看出无论是在比较次数还是在运行时间，$Sherwood$算法C都具有强大的优势。概率算法在实验中表现最好。

Problem 7

P77. EX

证明：当放置$(k+1)th$皇后时，若有多个位置是开放的, 则算法$QueensLV$选中其中任一位置的概率相等。

```
QueensLv (success){ //贪心的LV算法，所有皇后都是随机放置
    //若Success=true，则try[1..8]包含8后问题的一个解。
    col,diag45,diag135←Φ; //列及两对角线集合初值为空
    k ←0; //行号
    repeat //try[1..k]是k-promising，考虑放第k+1个皇后
        nb ←0; //计数器，nb值为（k+1)th皇后的open位置总数
        for i ←1 to 8 do { //i是列号，试探（k+1,i）安全否?
            if (i col) and (i-k-1 diag45) and (i+k+1 diag135) then{
                //列i对（k+1）th皇后可用，但不一定马上将其放在第i列
                nb ←nb+1;
                if uniform(1..nb)=1 then //或许放在第i列
                j ←i; //注意第一次uniform一定返回1，即j一定有值i
            }//endif
        }//endfor，在nb个安全的位置上随机选择1个位置j放置
        if(nb > 0) then{ //nb=0时无安全位置，第k+1个皇后尚未放好
            //在所有nb个安全位置上，(k+1)th皇后选择位置j的概率为1/nb
            k←k+1; //try[1..k+1]是(k+1)-promising
            try[k] ←j; //放置(k+1)th个皇后
            col ←col∪{ j };
            diag45 ←diag45 ∪{ j-k };
            diag135 ←diag135 ∪{ j+k };
        } //endif
    until （nb=0）or（k=8）; //当前皇后找不到合适的位置或try是
    // 8-promising时结束。
    success ← （nb>0）;
}
```

答：

当放置$(k+1)th$皇后时，若有t个位置开放，不妨假设为$W_1, W_2, \ldots, W_n$。算法选择其中任意一个位置W_i的概率$P_i = (\frac{1}{i} \times \frac{i}{i+1} \times \cdots \times \frac{n-1}{n})$。由于前面$i-1$都无法选到$i$因此概率为0，对于$i+1 \sim n$都应该不能选到$i$，否则会被覆盖，因此概率为$\frac{j-1}{j}$（其中，$j > i$）。故，算法$QueensLV$选中其中任一位置的概率相等。

Problem 8

P83. EX

写一算法，求$n = 12 \sim 20$时最优的$StepVegas$值。

答：

```cpp
#include<bits/stdc++.h>
#define TIMES 100 //对于每个StepVegas的运行次数（取平均作为该StepVegas的节点数）
using namespace std;
bool queens_lv(int n, int StepVegas, vector<int> chose, vector<bool> col,
vector<bool> diag45, vector<bool> diag135, long long& count) {
    //使用算法QueensLv进行查找，查找StepVegas个
    int i, row;
    for (i = 0; i < n; i++)
        col[i] = 0;
    for ( i =0; i < 2 * n; i++)
        diag45[i] = diag135[i] = 0;
    for (row = 0; row < StepVegas; row++) {
        int nb = 0;
        int sl = 0;
        for (i = 0; i < n; i++) {
            if (!col[i] && !diag45[n + i - row] && !diag135[i + row]) {
                ++nb;
                int r = rand() % nb + 1;
                if (r == 1)
                    sl = i;
            }
        }
        if (nb == 0)
            return 0;
        ++count;
        chose[row] = sl;
        col[sl] = 1;
        diag45[n + sl - row] = 1;
        diag135[sl + row] = 1;
    }
    return 1;
}
bool backtrack(int n, int row, vector<int> chose, vector<bool> col,
vector<bool> diag45, vector<bool> diag135, long long& count) {
    //回溯法进行查找
    int i;
    if (row == n)
        return 1;
    for (i=0; i<n; i++) {
        if (!col[i] && !diag45[n + i - row] && !diag135[i + row]) {
            ++count;
            chose[row] = i;
            col[i] = diag45[n + i - row] = diag135[i + row] = 1;
            if (backtrack(n, row + 1, chose, col, diag45, diag135, count))
                return 1;
            col[i] = diag45[n + i - row] = diag135[i + row] = 0;
        }
    }
    return 0;
}
pair<int, double> search_queens(int n, int StepVegas) {
    vector<int> chose(n);
    vector<bool> col(n);
    vector<bool> diag45(2 * n);
    vector<bool> diag135(2 * n);
```

```cpp
    long long count = 0;
    long long total_count = 0;
    long long successcount = 0;
    while (true) {
        if (queens_lv(n, StepVegas, chose, col, diag45, diag135, count)) {
            if (backtrack(n, StepVegas, chose, col, diag45, diag135, count))
                //判断是否成功运行
                successcount += 1;
                total_count += 1;
                return {count, (double)successcount / total_count};
        } else {
            total_count += 1;
        }
    }
}
int main() {
    long long total, min, bestsv;
    for (int n = 12; n <= 20; n++) {
        bestsv = 0;
        for (int StepVegas = 0; StepVegas <= n; StepVegas++) {
            total = 0;
            clock_t start, end;
            start = clock();
            double success = 0.0;
            for (int i=0; i < TIMES; i++) {
                pair<int, double> tmp = search_queens(n, StepVegas);
                total += tmp.first;
                success += tmp.second;
            }
            end = clock();
            if (bestsv == 0 || total <= min) {
                bestsv = StepVegas;
                min = total;
            }
            cout << "StepVegas: " << StepVegas << ", Count: " << total * 1.0 / TIMES << " , Time: " << (double)(end - start) / CLOCKS_PER_SEC / TIMES * 1000 << "ms , Success Rate: " << success / TIMES << endl;

        }
        printf("n=%d bestStepVegas=%d nodes=%f\n", n, bestsv, min * 1.0 / TIMES);
    }
    return 0;
}
```

实验结果（$Count$ or $nodes$：成功时搜索的结点的平均数）

n	$BestStepVegas$	$Count$	$Time\backslash ms$	$SuccessRate$
12	5	12	0.05806	1
13	6	13	0.06283	1
14	8	14	0.0621	1

n	$BestStepVegas$	$Count$	$Time\backslash ms$	$SuccessRate$
15	8	15	0.07327	1
16	9	13	0.07722	1
17	9	17	0.06161	1
18	10	18	0.09084	1
19	11	19	0.09705	1
20	11	20	0.0774	1

从实验可以看出，$BestStepVegas$最优取值大概是n的一半。

```
#n=12
StepVegas: 0, Count: 261 , Time: 1.12814ms , Success Rate: 1
StepVegas: 1, Count: 54 , Time: 0.2337ms , Success Rate: 1
StepVegas: 2, Count: 36 , Time: 0.21685ms , Success Rate: 1
StepVegas: 3, Count: 23 , Time: 0.13381ms , Success Rate: 1
StepVegas: 4, Count: 21 , Time: 0.1169ms , Success Rate: 1
StepVegas: 5, Count: 12 , Time: 0.05806ms , Success Rate: 1
StepVegas: 6, Count: 12.05 , Time: 0.05337ms , Success Rate: 0.995
StepVegas: 7, Count: 12.12 , Time: 0.04944ms , Success Rate: 0.99
StepVegas: 8, Count: 12.88 , Time: 0.0463ms , Success Rate: 0.941667
StepVegas: 9, Count: 14.05 , Time: 0.04485ms , Success Rate: 0.889167
StepVegas: 10, Count: 18.95 , Time: 0.05036ms , Success Rate: 0.731262
StepVegas: 11, Count: 55.17 , Time: 0.11862ms , Success Rate: 0.340883
StepVegas: 12, Count: 194.06 , Time: 0.38881ms , Success Rate: 0.180727
n=12 bestStepVegas=5 nodes=12.000000

#n=13
StepVegas: 0, Count: 111 , Time: 0.72038ms , Success Rate: 1
StepVegas: 1, Count: 111 , Time: 0.72308ms , Success Rate: 1
StepVegas: 2, Count: 84 , Time: 0.54557ms , Success Rate: 1
StepVegas: 3, Count: 42 , Time: 0.26702ms , Success Rate: 1
StepVegas: 4, Count: 19 , Time: 0.11247ms , Success Rate: 1
StepVegas: 5, Count: 13 , Time: 0.06862ms , Success Rate: 1
StepVegas: 6, Count: 13 , Time: 0.06283ms , Success Rate: 1
StepVegas: 7, Count: 13.06 , Time: 0.05869ms , Success Rate: 0.995
StepVegas: 8, Count: 13.26 , Time: 0.05427ms , Success Rate: 0.98
StepVegas: 9, Count: 14.02 , Time: 0.05055ms , Success Rate: 0.941667
StepVegas: 10, Count: 17.8 , Time: 0.05523ms , Success Rate: 0.779167
StepVegas: 11, Count: 24.8 , Time: 0.06398ms , Success Rate: 0.638917
StepVegas: 12, Count: 62.95 , Time: 0.13992ms , Success Rate: 0.36988
StepVegas: 13, Count: 270.3 , Time: 0.55239ms , Success Rate: 0.117422
n=13 bestStepVegas=6 nodes=13.000000

#n=14
StepVegas: 0, Count: 1899 , Time: 11.592ms , Success Rate: 1
StepVegas: 1, Count: 290 , Time: 1.96263ms , Success Rate: 1
StepVegas: 2, Count: 86 , Time: 0.5802ms , Success Rate: 1
StepVegas: 3, Count: 23 , Time: 0.14902ms , Success Rate: 1
StepVegas: 4, Count: 23 , Time: 0.14414ms , Success Rate: 1
```

```
StepVegas: 5, Count: 23 , Time: 0.13919ms , Success Rate: 1
StepVegas: 6, Count: 14 , Time: 0.07346ms , Success Rate: 1
StepVegas: 7, Count: 14 , Time: 0.06756ms , Success Rate: 1
StepVegas: 8, Count: 14 , Time: 0.0621ms , Success Rate: 1
StepVegas: 9, Count: 14.24 , Time: 0.05807ms , Success Rate: 0.985
StepVegas: 10, Count: 15.88 , Time: 0.05686ms , Success Rate: 0.9075
StepVegas: 11, Count: 20.6 , Time: 0.06244ms , Success Rate: 0.737667
StepVegas: 12, Count: 29.4 , Time: 0.07589ms , Success Rate: 0.644984
StepVegas: 13, Count: 72.35 , Time: 0.16482ms , Success Rate: 0.356946
StepVegas: 14, Count: 338.09 , Time: 0.71292ms , Success Rate: 0.129042
n=14 bestStepVegas=8 nodes=14.000000

#n=15
StepVegas: 0, Count: 1359 , Time: 8.94874ms , Success Rate: 1
StepVegas: 1, Count: 357 , Time: 2.03493ms , Success Rate: 1
StepVegas: 2, Count: 84 , Time: 0.58898ms , Success Rate: 1
StepVegas: 3, Count: 84 , Time: 0.59499ms , Success Rate: 1
StepVegas: 4, Count: 15 , Time: 0.091ms , Success Rate: 1
StepVegas: 5, Count: 15 , Time: 0.08609ms , Success Rate: 1
StepVegas: 6, Count: 15 , Time: 0.08121ms , Success Rate: 1
StepVegas: 7, Count: 15 , Time: 0.07564ms , Success Rate: 1
StepVegas: 8, Count: 15 , Time: 0.07327ms , Success Rate: 1
StepVegas: 9, Count: 15.24 , Time: 0.06899ms , Success Rate: 0.985
StepVegas: 10, Count: 15.71 , Time: 0.06339ms , Success Rate: 0.96
StepVegas: 11, Count: 17.3 , Time: 0.06269ms , Success Rate: 0.893333
StepVegas: 12, Count: 21.71 , Time: 0.06772ms , Success Rate: 0.7785
StepVegas: 13, Count: 34.38 , Time: 0.09074ms , Success Rate: 0.585707
StepVegas: 14, Count: 78.83 , Time: 0.18415ms , Success Rate: 0.336905
StepVegas: 15, Count: 317.85 , Time: 0.69524ms , Success Rate: 0.15628
n=15 bestStepVegas=8 nodes=15.000000

#n=16
StepVegas: 0, Count: 10052 , Time: 72.9982ms , Success Rate: 1
StepVegas: 1, Count: 1378 , Time: 10.1822ms , Success Rate: 1
StepVegas: 2, Count: 39 , Time: 0.28703ms , Success Rate: 1
StepVegas: 3, Count: 39 , Time: 0.27335ms , Success Rate: 1
StepVegas: 4, Count: 39 , Time: 0.27148ms , Success Rate: 1
StepVegas: 5, Count: 16 , Time: 0.09967ms , Success Rate: 1
StepVegas: 6, Count: 16 , Time: 0.09438ms , Success Rate: 1
StepVegas: 7, Count: 16 , Time: 0.08896ms , Success Rate: 1
StepVegas: 8, Count: 16 , Time: 0.08319ms , Success Rate: 1
StepVegas: 9, Count: 16 , Time: 0.07722ms , Success Rate: 1
StepVegas: 10, Count: 16.27 , Time: 0.07283ms , Success Rate: 0.985
StepVegas: 11, Count: 16.76 , Time: 0.06859ms , Success Rate: 0.96
StepVegas: 12, Count: 19.17 , Time: 0.06849ms , Success Rate: 0.856667
StepVegas: 13, Count: 23.81 , Time: 0.07476ms , Success Rate: 0.753167
StepVegas: 14, Count: 39.29 , Time: 0.10293ms , Success Rate: 0.544008
StepVegas: 15, Count: 83.3 , Time: 0.19201ms , Success Rate: 0.354003
StepVegas: 16, Count: 442.72 , Time: 1.01022ms , Success Rate: 0.118406
n=16 bestStepVegas=9 nodes=16.000000

#n=17
StepVegas: 0, Count: 5374 , Time: 40.018ms , Success Rate: 1
StepVegas: 1, Count: 187 , Time: 1.43494ms , Success Rate: 1
StepVegas: 2, Count: 187 , Time: 1.39683ms , Success Rate: 1
```

```
StepVegas: 3, Count: 27 , Time: 0.1924ms , Success Rate: 1
StepVegas: 4, Count: 20 , Time: 0.13525ms , Success Rate: 1
StepVegas: 5, Count: 20 , Time: 0.12272ms , Success Rate: 1
StepVegas: 6, Count: 17 , Time: 0.07356ms , Success Rate: 1
StepVegas: 7, Count: 17 , Time: 0.06947ms , Success Rate: 1
StepVegas: 8, Count: 17 , Time: 0.06561ms , Success Rate: 1
StepVegas: 9, Count: 17 , Time: 0.06161ms , Success Rate: 1
StepVegas: 10, Count: 17.09 , Time: 0.05792ms , Success Rate: 0.995
StepVegas: 11, Count: 17.29 , Time: 0.0541ms , Success Rate: 0.985
StepVegas: 12, Count: 19.32 , Time: 0.05408ms , Success Rate: 0.896667
StepVegas: 13, Count: 20.77 , Time: 0.05271ms , Success Rate: 0.857
StepVegas: 14, Count: 28.24 , Time: 0.06208ms , Success Rate: 0.69654
StepVegas: 15, Count: 51.25 , Time: 0.09729ms , Success Rate: 0.50902
StepVegas: 16, Count: 125.11 , Time: 0.21746ms , Success Rate: 0.212695
StepVegas: 17, Count: 487.31 , Time: 0.94848ms , Success Rate: 0.109195
n=17 bestStepVegas=9 nodes=17.000000

#n=18
StepVegas: 0, Count: 41299 , Time: 313.898ms , Success Rate: 1
StepVegas: 1, Count: 4516 , Time: 33.2639ms , Success Rate: 1
StepVegas: 2, Count: 164 , Time: 1.28759ms , Success Rate: 1
StepVegas: 3, Count: 95 , Time: 0.7423ms , Success Rate: 1
StepVegas: 4, Count: 19 , Time: 0.1329ms , Success Rate: 1
StepVegas: 5, Count: 19 , Time: 0.12809ms , Success Rate: 1
StepVegas: 6, Count: 19 , Time: 0.12212ms , Success Rate: 1
StepVegas: 7, Count: 18 , Time: 0.10791ms , Success Rate: 1
StepVegas: 8, Count: 18 , Time: 0.10363ms , Success Rate: 1
StepVegas: 9, Count: 18 , Time: 0.0972ms , Success Rate: 1
StepVegas: 10, Count: 18 , Time: 0.09084ms , Success Rate: 1
StepVegas: 11, Count: 18.29 , Time: 0.08653ms , Success Rate: 0.985
StepVegas: 12, Count: 18.33 , Time: 0.07971ms , Success Rate: 0.985
StepVegas: 13, Count: 19.81 , Time: 0.07871ms , Success Rate: 0.923333
StepVegas: 14, Count: 23.58 , Time: 0.08143ms , Success Rate: 0.818333
StepVegas: 15, Count: 30.82 , Time: 0.09338ms , Success Rate: 0.685595
StepVegas: 16, Count: 53.68 , Time: 0.1424ms , Success Rate: 0.442309
StepVegas: 17, Count: 102.12 , Time: 0.25102ms , Success Rate: 0.329539
StepVegas: 18, Count: 616.42 , Time: 1.45235ms , Success Rate: 0.0751512
n=18 bestStepVegas=10 nodes=18.000000

#n=19
StepVegas: 0, Count: 2545 , Time: 19.6038ms , Success Rate: 1
StepVegas: 1, Count: 2545 , Time: 19.122ms , Success Rate: 1
StepVegas: 2, Count: 506 , Time: 4.14497ms , Success Rate: 1
StepVegas: 3, Count: 287 , Time: 2.06813ms , Success Rate: 1
StepVegas: 4, Count: 95 , Time: 0.54128ms , Success Rate: 1
StepVegas: 5, Count: 22 , Time: 0.11132ms , Success Rate: 1
StepVegas: 6, Count: 19 , Time: 0.08993ms , Success Rate: 1
StepVegas: 7, Count: 19 , Time: 0.10257ms , Success Rate: 1
StepVegas: 8, Count: 19 , Time: 0.11496ms , Success Rate: 1
StepVegas: 9, Count: 19 , Time: 0.10864ms , Success Rate: 1
StepVegas: 10, Count: 19 , Time: 0.1021ms , Success Rate: 1
StepVegas: 11, Count: 19 , Time: 0.09705ms , Success Rate: 1
StepVegas: 12, Count: 19.32 , Time: 0.09107ms , Success Rate: 0.985
StepVegas: 13, Count: 19.68 , Time: 0.08657ms , Success Rate: 0.973333
StepVegas: 14, Count: 21.61 , Time: 0.08473ms , Success Rate: 0.911667
```

```
StepVegas: 15, Count: 25.8 , Time: 0.09025ms , Success Rate: 0.824262
StepVegas: 16, Count: 34.6 , Time: 0.10601ms , Success Rate: 0.672333
StepVegas: 17, Count: 57.19 , Time: 0.15543ms , Success Rate: 0.497009
StepVegas: 18, Count: 137.22 , Time: 0.34455ms , Success Rate: 0.243384
StepVegas: 19, Count: 647.6 , Time: 1.58246ms , Success Rate: 0.116297
n=19 bestStepVegas=11 nodes=19.000000

#n=20
StepVegas: 0, Count: 199635 , Time: 1552.84ms , Success Rate: 1
StepVegas: 1, Count: 5799 , Time: 45.5952ms , Success Rate: 1
StepVegas: 2, Count: 530 , Time: 3.84027ms , Success Rate: 1
StepVegas: 3, Count: 261 , Time: 2.21019ms , Success Rate: 1
StepVegas: 4, Count: 47 , Time: 0.37265ms , Success Rate: 1
StepVegas: 5, Count: 35 , Time: 0.26731ms , Success Rate: 1
StepVegas: 6, Count: 25 , Time: 0.17775ms , Success Rate: 1
StepVegas: 7, Count: 20 , Time: 0.13053ms , Success Rate: 1
StepVegas: 8, Count: 20 , Time: 0.12484ms , Success Rate: 1
StepVegas: 9, Count: 20 , Time: 0.11157ms , Success Rate: 1
StepVegas: 10, Count: 20 , Time: 0.08164ms , Success Rate: 1
StepVegas: 11, Count: 20 , Time: 0.0774ms , Success Rate: 1
StepVegas: 12, Count: 20.22 , Time: 0.0735ms , Success Rate: 0.99
StepVegas: 13, Count: 20.35 , Time: 0.06932ms , Success Rate: 0.985
StepVegas: 14, Count: 21.62 , Time: 0.06749ms , Success Rate: 0.938333
StepVegas: 15, Count: 23.09 , Time: 0.06587ms , Success Rate: 0.910833
StepVegas: 16, Count: 29.36 , Time: 0.07375ms , Success Rate: 0.771167
StepVegas: 17, Count: 40.4 , Time: 0.0898ms , Success Rate: 0.637563
StepVegas: 18, Count: 60.76 , Time: 0.12244ms , Success Rate: 0.53226
StepVegas: 19, Count: 155.92 , Time: 0.2911ms , Success Rate: 0.298709
StepVegas: 20, Count: 702.93 , Time: 1.59067ms , Success Rate: 0.0958049
n=20 bestStepVegas=11 nodes=20.000000
```