

算法设计与分析

2018年3月20日

姓名：姜贵平
学号：SA17011142

1 Probabilistic Algorithm

1.1 若将 $y \leftarrow \text{uniform}(0, 1)$ 改为 $y \leftarrow x$, 则上述的算法估计的值是什么?

因为 $\frac{k}{n} = \frac{1}{\sqrt{2}}$, 所以结果应该趋于 $2\sqrt{2}$

1.2 在在机器上用 $4 \int_0^1$, 给出不同的n 值其精度

表 1: Source 1

<pre>#include<stdio.h> #include<stdlib.h> #include<time.h> int main() { float x,y; int s[10]={1e4,1e5,1e6,1e7,1e8,1e9}; for(int j=0;j<6;j++) { int count=0; srand(((unsigned int)time(NULL)));</pre>	<pre>for(int i=0;i<s[j];i++) { x=(float)rand()/RAND_MAX; y=(float)rand()/RAND_MAX; if(x*x<1-y*y) count++; } printf("N=%d result is%f",s[j],4.0*count/s[j]); return 0; }</pre>
---	---

```
C:\Users\Administrator\Desktop\未命名1.exe
N=10000
result is 3.142800
N=100000
result is 3.146720
N=1000000
result is 3.144160
N=10000000
result is 3.142431
N=100000000
result is 3.141475
N=1000000000
result is 3.141522

-----
Process exited after 25.82 seconds with return value 0
请按任意键继续. . .
```

图 1: result1

由图可知，不同的n 对应的精确值为：

n	1e4	1e5	1e6	1e7	1e8	1e9
精度	0.01	0.01	0.01	0.01	0.001	0.0001

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

表 2: Source 2

<pre>#include<stdio.h> #include<stdlib.h> #include<time.h> #define N 1e9 float function(float x) {return x*x-1;} float call(float(*f)(float),float a,float b,float c,float d) { int count=0; for(int i=0;i<N;i++) { float x=(float)rand()/RAND_MAX*(b-a)+a; float y=(float)rand()/RAND_MAX*(d-c)+c;</pre>	<pre>float y0=f(x); if(y > 0&& y < y0) count++; if(y < 0&& y0 < y) count--; } return count/N*(d-c)*(b-a); int main() { srand((unsigned int)time(NULL)); float a=-2,b=2,c=-1,d=3; printf("result is %f",call(function,a,b,c,d)); return 0; }</pre>
--	--

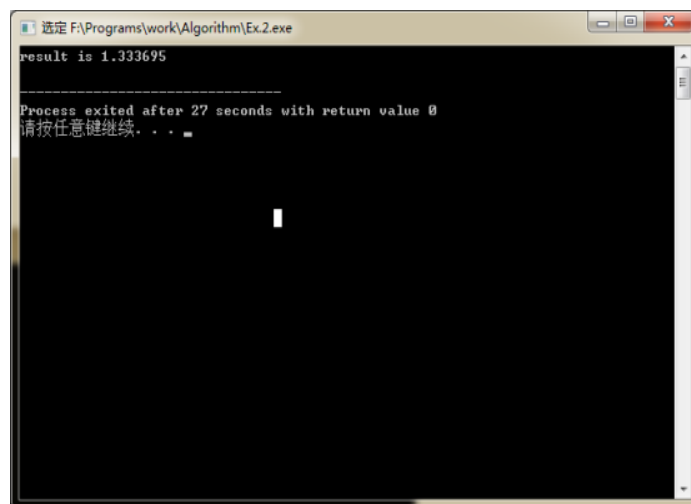


图 2: result2

由图可知, 实际积分等于 $4/3$ 。在 $N=1e9$, 实验精确度为 0.001 。

1.4 设 ε, σ 是(0,1) 之间的常数, 证明

若 I 是 $\int_0^1 f(x)$ 的正确值, h 是由HitorMiss 算法返回的值, 则当 $n \geq I(1-I)/\varepsilon 2\sigma$ 时有:

$$Prob[|h - I| < \varepsilon] \leq 1 - \sigma$$

Proof: 设 X 是落入 $1/4$ 圆的概率, 显然 $X \sim (n, I)$, 所以有:

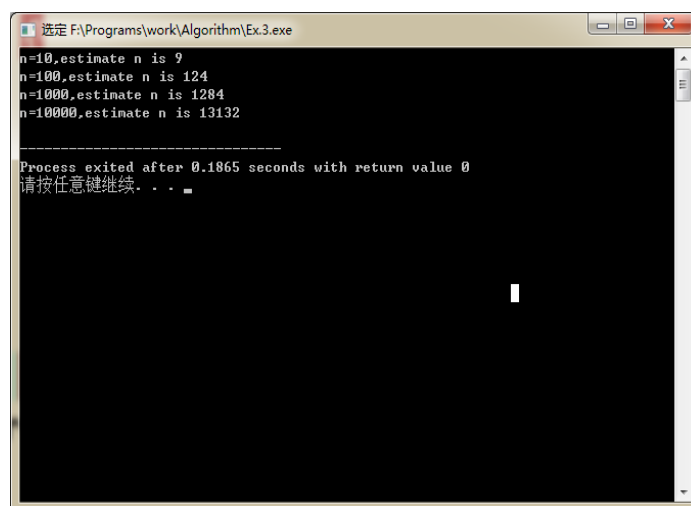
$$E(x) = nI \quad D(x) = nI(1-I)$$

由切比雪夫不等式可知 $P(|nh - nI| < \delta) \geq 1 - \frac{D(x)}{\delta^2}$, 可得 $P(|h - I| < \frac{\delta}{n}) \geq 1 - \frac{D(x)}{\delta^2}$, 那么令 $\frac{\delta}{n} = \varepsilon$, $D(x) = nI(1-I)$ 以及带入可得 $Prob[|h - I| < \varepsilon] \leq 1 - \frac{nI(1-I)}{n^2 * \varepsilon^2}$, 将 $n = \frac{I(1-I)}{\varepsilon^2 \sigma}$ 带入, 可得 $Prob[|h - I| < \varepsilon] \leq 1 - \sigma$

1.5 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

表 3: Source 3

<pre>#include< stdio.h > #include< stdlib.h > #include< time.h > #include< string.h > #define pi 3.1415926 #define N 10000 int Set[N]={0},visited[N]={0}; int setcount(int X_size) { for(int temp=0;temp < N;temp++) Set[temp]=0,visited[temp]=0; int count=0,rand_num=rand()%X_size; do{ count++; visited[rand_num]=1; rand_num = rand()%X_size; }while(visited[rand_num]==0); return int((float)(2*count*count)/pi);</pre>	<pre>} int main() { srand((unsigned int)time(NULL)); int n; for(n=10;n<N;n*=10) { int count=0; for(int j=0;j < 200;j++) { count+=setcount(n); } printf("n=%d,estimate n is %d",n,count/200); } return 0;</pre>
---	---



```
选定 F:\Programs\work\Algorithm\Ex.3.exe
n=10, estimate n is 9
n=100, estimate n is 124
n=1000, estimate n is 1284
n=10000, estimate n is 13132

Process exited after 0.1865 seconds with return value 0
请按任意键继续. . .
```

图 3: result3

实验采用取200次的平均结果，由结果可知，随着实验的n增大，每次估计都比实际的n 要大，且随着n 增大，误差也在增大，不过效果还可以。

1.6 分析dlogRH的工作原理，指出该算法相应的u和v

dlogRH将原来单独求 $\log_{g,p} a$ 转化为 $\log_{g,p} c$ 其中 $c = g^{r+x} \bmod p$ 相当于做了一个加法变换，其中u变换为 $g^{r+x} \bmod p$, v变化为 $(y - r) \bmod (p - 1)$ 。

1.7 写一Sherwood算法C, 与算法A, B, D比较, 给出实验结果

表 4: Source 4

<pre> int Search(int x,int i) { count =1; while($x > val[i]$) { i=ptr[i]; count++; return i;} int A(int x) { return Search(x,head);} int B(int x) { int y,i=head; int max=val[i]; for(int j=1;$j * j \leq N$;j++) { countB++; y=val[j]; if($max < y \&\& y \leq x$){i=j;max=y;} } return Search(x,i); } int C(int x) { int y,i=head; int max=val[i]; for(int j=1;$j < N/8$;j++) { y=val[j]; countC++; if($max < y \&\& y < x$){i=j;max=y;} } return Search(x,i); } int D(int x) { int i=rand()%N+1;y=val[i]; countD++; if($x < y$) {return Search(x,head);} if($x > y$){return Search(x,ptr[i]);} if($y == x$)return i; } </pre>	<pre> void Gen_Data() { int index, pre; head = (rand()%N+1); val[pre=head] = 1; for(int i = 2;i= N;) {index = rand()%N+1 ; if(0==val[index]) { val[index] = i; ptr[pre] = index; pre = index; i++; } } } int main() { int find_num; for(int i = 1; $i \leq REAPT_TIMES$; i++) { find_num=rand()%N+1; A(find_num); countA+ = count; B(find_num); countB+ = count; C(find_num); countC+ = count; D(find_num); countD+ = count; } printf("countA = %lld \n",countA/REAPT_TIMES); printf("countB = %lld \n",countB/REAPT_TIMES); printf("countC = %lld \n",countC/REAPT_TIMES); printf("countD = %lld",countD/REAPT_TIMES); return 0; } </pre>
---	--

```

C:\Users\Administrator\CLionProjects\source\cmake-build-debug\source.exe
countA = 15569
countB = 362
countC = 4104
countD = 10514

Process finished with exit code 0

```

图 4: result4

实验中同样采用的多次计算取平均值的方法(200),来减少偶然误差,从实验结果来看,使用概率算法查找次数少于其他算法。D算法优于A算法,本算法采用1到 $RAND_MAX(32767)$ 之间随机查找,算法收敛于一半, D收敛于1/3, 而算法B 最优, 收敛于 $2\sqrt{n}$,而我随机设置大小为 $n/8$,明显低于B算法的 \sqrt{n} ,但也优先于其他算法,理论与实验基本符合。

1.8 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

证明: 满足条件第i个位置别选中的概率为 $\frac{1}{i}$, 若 $j = i$,则意味着在i+1到nb, 没有被选中, 设选中位置为i的概率为 $p(i)$ 。

$$p(i) = \frac{1}{i} * (1 - \frac{1}{i+1}) * \dots * (1 - \frac{1}{nb}) = \frac{1}{nb}$$

1.9 写一算法, 求 $n = 12$ 20时最优的StepVegas值

表 5: Source 5

```
bool is_legal(int row, int col)
{
    if(row ≥ 2)
    for(int m = 1; m < row; m++)
    if((col + row) == (chess[m] + m) || (col - row) == (chess[m] - m) || (col == chess[m]))
    return false;
    return true;
}

bool backtrace(int k)
{
    int i = k + 1; int j = 1;
    while( i <= CHESS_SIZE && i >= k + 1 )
    { for(; j = CHESS_SIZE; j++)
    if(is_legal(i, j))
    {chess[i] = j; st.push(j); i = i + 1; j = 1; break;}
    if( j == CHESS_SIZE + 1 )
    { i = i - 1; if(i <= k) return false; j = st.top() + 1;
    st.pop(); }
    }
    if( i <= k) return false;
    return true;
}

bool QueenLV()
{
    int i, j, nb, k = 0;
    if(stepVegas == k) return backtrace(k);
    while(true)
    { nb = 0; /* number of open positions for the (k+1)th queen
    for(i = 1; i <= CHESS_SIZE; i++)
    if(is_legal(k+1, i)){nb += 1; if( (rand())%nb + 1 == 1 ) j = i;}
    if(nb > 0){k = k + 1; chess[k] = j;}
    if(nb == 0 || k == stepVegas) break;
    }
    if(nb > 0) return backtrace(k);
    return false;
}
```

算法主要由三部分组成，判断位置是否合法，回退n步，以及QueenLV算法。下面看实验结果。

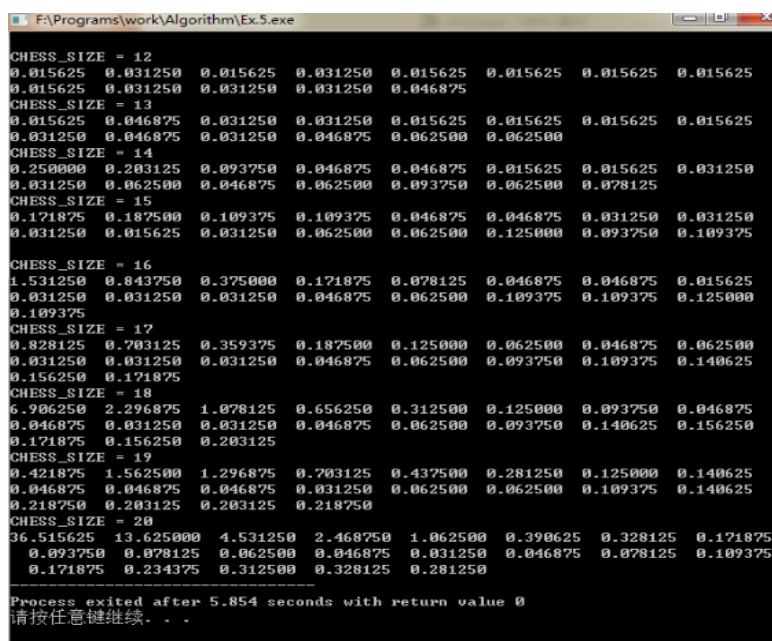


图 5: result5

上图是1到20皇后采用不同的stepvagas(1到 *CHESE_SIZE*) 所用时间的平均毫秒数（重复200次），有图可得到下面的结果。

表 6: 最优的stepVagas对应的步数为

CHESE.SIZE	12	13	14	15	16	17	18	19	20
stepvagas	6	6	7	7	8	10	10	11	12

由实验结果可知，大概在一半的时候，速度最快。特别当CHESE.SIZE变大时，这个趋势会变大。

1.10 与确定性算法相比较，并给出100~10000以内错误的比例

python 代码6:

```
import random
import numpy as np
```

```

import math
def PrintPrimes():
    list=[2,3]
    for n in range(5,10000,2):
        if RepeatMillRab(n,(int)(math.log(n,2))):
            list.append(n)
    return list
def RepeatMillRab(n,k):
    for i in range(1,k+1):
        if MillRob(n)==False:
            return False
    return True
def MillRob(n):
    a=random.randint(2,n-2)
    return Btest(a,n)
def Btest(a,n):
    s=0
    t=n-1
    while(t%2==1):
        s=s+1
        t=t/2
    x=a**t%n
    if (x==1)|(x==n-1):
        return True
    for i in range(1,s):
        x=x*x%n
        if x==n-1:
            return True
    return False
def prime():
    list=[2,3]
    for i in range(5,10000,2):
        flag=0
        for j in range(2,(int)(np.sqrt(i))+1):
            if i%j==0:
                flag=1
        if flag==0:
            list.append(i)
    return list

```

```

if __name__ == "__main__":
    list_certer=prime()
    count=0;
    for i in range(100):
        list_uncerter = PrintPrimes()
        for j in list_uncerter:
            if j not in list_certer:
                count+=1
    print(count)

```

运行了100次，但是只有37个错误，可见错误率非常的低，大约在 $37/10/10000 \times 100\% = 0.0037\%$ ，算法的准确非常的高。