**算法分析与设计第一次作业**

**姓名：赵翔宇 学号：SA14011047**

**第一章 概率算法**

**EX1：**

**问题：**

**若将y ← uniform(0, 1) 改为 y ← x，则上述的算法估计的值是什么？**

**答：**

算法估计的值是。

将y ← uniform(0, 1) 改为 y ← x，原算法中将改

if () then k++; 即当时，k++。

则有, 则算法返回的表示。

**EX2：**

**问题：**

**在机器上用估计值，给出不同的n值及精度。**

**答：**

=3.141592654。

n=1000万，估计值为3.1419408，3位精确

n=1亿，估计值为3.1415852，4位精确

n=10亿，估计值为3.1415063，4位精确。

**算法代码为：**

**import** java.util.Random;

**public** **class** Ex2 {

**public** **double** HitorMiss(**long** n){

**int** index=1;

**int** k=0;

**double** x=0;

**double** y=0;

**while**(index<=n){

x=Math.*random*();

y=Math.*random*();

**if**((y-Math.*cbrt*(1-x\*x))<=0){

k++;

}

index++;

}

**return** ((4\*(**double**)k)/n);

}

**public** **static** **void** main(String[] args){

Ex2 ex=**new** Ex2();

System.*out*.println(ex.HitorMiss(10000000));

System.*out*.println(ex.HitorMiss(100000000));

System.*out*.println(ex.HitorMiss(1000000000));

}

}

**EX3：**

**问题：**

**设a, b, c和d是实数，且a ≤ b, c ≤ d, f:[a,b] → [c, d]是一个连续函数，写一概率算法计算积分：。**

**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：**

**概率算法表示为：**

Ex3(f,n,a,b,c,d){

k ← 0;

for i ← 1 to n do {

x ← uniform(a,b);

y ← uniform(c,d);

if y <= f(x) then k++;

}

return (b-a)\*c + k\*(b-a)\*(d-c)/n;

}

**运行结果如下所示：**

n=1000万次，估计值为：0.4999917

n=1亿次，估计值为：0.5000984

n=10亿次，估计值为：0.500015138

**算法代码为：**

public class Ex3 {

// f(x)=x

public double f(double x) {

return x;

}

public double ex3(long n, double a, double b, double c, double d) {

long k = 0;

for (long i = 0; i < n; i++) {

double x = Math.*random*();

double y = Math.*random*();

if (y <= this.f(x))

k++;

}

return (b - a) \* c + 1.0 \* k / n \* (b - a) \* (d - c);

}

public static void main(String args[]){

Ex3 ex=new Ex3();

System.*out*.println(ex.ex3(10000000,0,1,0,1));

System.*out*.println(ex.ex3(100000000,0,1,0,1));

System.*out*.println(ex.ex3(1000000000,0,1,0,1));

}

}

**EX4：**

**问题：**

**设ε,δ是(0,1)之间的常数，证明：**

**若I是 的正确值，h是由HitorMiss算法返回的值，则当n ≥ I(1-I)/ε2δ时有：**

**Prob[|h-I| < ε] ≥ 1 – δ**

**答：**

证明：k为n次中点落在积分阴影中的次数，故，其中为每次落入阴影的概率，由等概率假设知p=I，即有，故欲证，即。即，即,由切比雪夫不等式，有

同时由二项分布性质:

故有

所以需：

即：

**EX5：**

**问题：**

**用上述算法，估计整数子集1~n的大小，并分析n对估计值的影响。**

**答：**

**结果如下：**

N=10万， 估计值为：93897.93225723281

N=100万， 估计值为：1117717.143802979

N=1000万，估计值为：10589957.10681534

N=1亿， 估计值为： 103126298.7541977

N=10亿， 估计值为： 954863704.3805844

**算法代码如下：**

**import** java.util.HashSet;

**import** java.util.Random;

**public** **class** Ex5 {

**public** **double** setCount(**int** n){

Random r = **new** Random();

HashSet<Integer> S=**new** HashSet<Integer>();

**long** k=0;

**int** a=r.nextInt(n);

**do**{

k++;

S.add(a);

a=r.nextInt(n);

}**while**(!S.contains(a));

**return** 2\*k\*k/Math.*PI*;

}

**public** **static** **void** main(String args[]){

Ex5 ex=**new** Ex5();

System.*out*.println(ex.setCount(100000));

System.*out*.println(ex.setCount(1000000));

System.*out*.println(ex.setCount(10000000));

System.*out*.println(ex.setCount(100000000));

System.*out*.println(ex.setCount(1000000000));

}

}

**EX6：**

**问题：**

**分析dlogRH的工作原理，指出该算法相应的u和v**

**答：**

dlogRH算法就是Sherwood算法的一个实例，通过随机预处理，将输入实例p随机变换为c，一定概率上可以降低计算dlog的时间复杂度，这里采用的函数u就是ModuleExponent和模乘法将p转换为c， 然后利用确定性算法dlog算法计算c的离散对数y，最后根据数论知识，通过变换(y-r)mod(p-1)也就是函数v将y还原出原问题的解x。

**EX7：**

**问题：**

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

**答：**

算法 比较次数

A 18

B 2

C 3

D 6

**算法代码如下：**

**import** java.util.Random;

**public** **class** Ex7 {

**int** n=20;

**int** head=3;

**int** val[]={2,17,11,1,7,15,20,3,19,14,8,5,18,10,13,6,4,12,16,9};

**int** ptr[]={7,12,17,0,10,18,100,16,6,5,19,15,8,2,9,4,11,14,1,13};

**public** **int** search(**int** x,**int** i){

**int** k=0;

**while**(x>val[i]){

i=ptr[i];

k++;

}

System.*out*.print(",共比较了"+k+"次,");

**return** i;

}

**public** **int** A(**int** x){// 时间为O(n)的确定性算法

**return** search(x,head);

}

**public** **int** B(**int** x){//时间为O(√n)的确定性算法

**int** j ,y, i=head;

**int** max=val[i];

**for**(j=0;j<Math.*sqrt*(1.0\*n);j++){

y=val[j];

**if**(max<y&&y<=x){

i=j;

max=y;

}

}

**return** search(x,i);

}

**public** **int** C(**int** x){//在算法那B上改进的sherwood算法

Random r=**new** Random();

**int** j,k,i=head;

**int** max=val[i];

**for**(j=0;j<Math.*sqrt*(1.0\*n);j++){

k=r.nextInt(n);

**if**(max<val[k]&&val[k]<=x){

i=k;

max=val[k];

}

}

**return** search(x,i);

}

**public** **int** D(**int** x){ //时间为O(n)的概率算法

Random r = **new** Random();

**int** i=r.nextInt(n);

**int** y=val[i];

**if**(x<y)

**return** search(x,head);

**else** **if**(x>y)

**return** search(x,ptr[i]);

**else**

**return** i;

}

**public** **static** **void** main(String args[]){

Ex7 ex=**new** Ex7();

**int** pos=0;

**int** target=19;

System.*out*.println("有序静态链表:");

System.*out*.println("val[]={2,17,11,1,7,15,20,3,19,14,8,5,18,10,13,6,4,12,16,9}");

System.*out*.println("ptr[]={7,12,17,0,10,18,100,16,6,5,19,15,8,2,9,4,11,14,1,13}");

System.*out*.print("算法A查找"+target);

pos=ex.A(target);

System.*out*.println(",位置是:"+pos);

System.*out*.print("算法B查找"+target);

pos=ex.B(target);

System.*out*.println(",位置是:"+pos);

System.*out*.print("算法C查找"+target);

pos=ex.C(target);

System.*out*.println(",位置是:"+pos);

System.*out*.print("算法D查找"+target);

pos=ex.D(target);

System.*out*.println(",位置是:"+pos);

}

}

**EX8：**

**问题：**

**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。**

**答：**

证明： 对于任意 满足，第m个位置被选中的概率等于

故对于(k+1)th皇后，若有个开放位置，则每个位置被选中的概率都是

**EX9：**

**问题：**

**写一算法，求n=12~20时最优的StepVegas值。**

**答：**

**部分程序运行结果：**

棋盘大小 StopStep 成功率% 平均成功运行时间(ms)

12 1 100 27

12 2 100 3

12 3 98 0

12 4 70 0

12 5 46 0

12 6 20 0

12 7 2 0

由此看来，一半略小的随机放置效果比较好，同理可以得到12~20皇后的所有结果。

**最优的StopStep如下所示：**

棋盘大小 Best StopStep

12 3

13 4

14 5

15 6

16 7

17 7

18 8

19 9

20 9

**算法的代码如下：**

**import** java.util.Random;

**public** **class** Ex9 {

**public** **static** **final** **int** *MAX\_N* = 20;// 棋盘最大尺寸

**public** **static** **final** **int** *REPEAT* = 50;// 程序重复次数

Random r = **new** Random();

**int** x[] = **new** **int**[*MAX\_N* + 1];

**int** y[] = **new** **int**[*MAX\_N* + 1];

**public** **boolean** isValid(**int** k) {// 判断第k个皇后是否合法

**for** (**int** j = 1; j < k; j++) {

**if** ((Math.*abs*(k - j) == Math.*abs*(x[j] - x[k])) || (x[j] == x[k]))

**return** **false**;

}

**return** **true**;

}

**public** **void** backtrack(**int** t, **int** n) {// 回溯判断

**if** (t > n) {

**for** (**int** i = 1; i <= n; i++)

y[i] = x[i];

**return**;

} **else** {

**for** (**int** i = 1; i <= n; i++) {

x[t] = i;

**if** (**this**.isValid(t)) {

**this**.backtrack(t + 1, n);

}

}

}

}

**public** **boolean** QueensLV(**int** stopLV, **int** n) {

**int** j, count = 1, k = 1;

**while** ((k <= stopLV) && (count > 0)) {

count = 0;

j = 0;

**for** (**int** i = 0; i <= n; i++) {

x[k] = i;

**if** (**this**.isValid(k)) {

count++;

**if** (r.nextInt(count) == 1)

j = i;

}

}

**if** (count > 0) {

x[k++] = j;

}

}

**if** (count > 0)

**return** **true**;

**else**

**return** **false**;

}

**public** **static** **void** main(String args[]) {

Ex9 ex = **new** Ex9();

**long** time\_now, time\_end;

**for** (**int** n = 12; n <= *MAX\_N*; n++) {

**for** (**int** stop = 1; stop <= n - n / 4 - 1; stop++) {

**int** j = 0, success = 0;

time\_now = System.*currentTimeMillis*();

**do** {

**if** (stop < n)

**while** (!ex.QueensLV(stop, n))

;

**else**

ex.QueensLV(stop, n);

ex.backtrack(stop + 1, n);

**if** (ex.y[n] != 0) {

success++;

**for** (**int** i = 0; i <= n; i++)

ex.x[i] = ex.y[i] = 0;

}

j++;

} **while** (j < *REPEAT*);

System.*out*.println("Size:" + n + " StopStep" + stop + ": Run "

+ *REPEAT* + " time,success " + success

+ " times!, success rate is " + 100.0 \* success

/ *REPEAT* + "%");

time\_end = System.*currentTimeMillis*();

System.*out*.println("it takes " + (time\_end - time\_now)

/ success + "ms to solve this problem");

}

}

}

}

**EX10：**

**问题：**

**PrintPrimes{ //打印1万以内的素数**

**print 2，3；**

**n ←5；**

**repeat**

**if RepeatMillRab(n, ) then print n;**

**n ←n+2;**

**until n=10000;**

**}**

**与确定性算法相比较，并给出100~10000以内错误的比例。**

**答：**

**得到的部分结果如下所示：**

2,3,5,7,11,13,17,19,23,29

31,37,41,43,47,53,59,61,67,71

73,79,83,89,97,101,103,107,109,113

127,131,137,139,149,151,157,163,167,173

179,181,191,193,197,199,211,223,227,229

233,239,241,251,257,263,269,271,277,281

283,293,307,311,313,317,331,337,347,349

353,359,367,373,379,383,389,397,401,409

419,421,431,433,439,443,449,457,461,463

467,479,487,491,499,503,509,521,523,541

547,557,563,569,571,577,587,593,599,601

607,613,617,619,631,641,643,647,653,659

661,673,677,683,691,701,709,719,727,733

739,743,751,757,761,769,773,787,797,809

811,821,823,827,829,839,853,857,859,863

877,881,883,887,907,911,919,929,937,941

947,953,967,971,977,983,991,997,1009,1013

1019,1021,1031,1033,1039,1049,1051,1061,1063,1069

1087,1091,1093,1097,1103,1109,1117,1123,1129,1151

......

9643,9649,9661,9677,9679,9689,9697,9719,9721,9733

9739,9743,9749,9767,9769,9781,9787,9791,9803,9811

9817,9829,9833,9839,9851,9857,9859,9871,9883,9887

9901,9907,9923,9929,9931,9941,9949,9967,9973,

error rate between 100 and 10000 is 0

**算法代码如下：**

**import** java.util.Random;

**public** **class** Ex10 {

**public** **static** **final** **int** *N*=10000;

Random r=**new** Random();

**int** flag[]=**new** **int**[*N*];

**public** **void** getPrime(){

**for**(**int** i=2;i<*N*/2;i++){

**if**(flag[i]!=0){

**for**(**int** j=2\*i;j<*N*;j+=i){

flag[j]=i;

}

}

}

}

**public** **boolean** Btest(**int** a,**int** n){

**int** s=0,t=n-1;

**do**{

s++;

t>>=1;

}**while**(t%2==0);

**int** x=1;

**for**(**int** i=0;i<t;i++){

x=x\*a%n;

**if**(x==n-1)

**return** **true**;

}

**return** **false**;

}

**public** **boolean** MillRab(**int** n){

**int** i=r.nextInt(n-3)+1;

**return** Btest(i,n);

}

**public** **boolean** RepeatMillRab(**int** n,**int** rep){

**for**(**int** i=0;i<rep;i++){

**if**(!**this**.MillRab(n))

**return** **false**;

}

**return** **true**;

}

**public** **double** PrintPrimes(){

**int** err=0,count=2;

System.*out*.println("2,3");

**for**(**int** n=5;n<*N*;n+=2){

**if**(**this**.RepeatMillRab(n, (**int**)Math.*floor*(Math.*log*(n\*1.0)/Math.*log*(2.0)))){

System.*out*.print(n);

count++;

**if**(count%10==0){

System.*out*.println();

}**else**{

System.*out*.println(",");

}

**if**(n>100)

**if**(flag[n]==1)

err++;

}

}

System.*out*.println("error rate between 100 and 10000 is"+1.0\*err/(count-25));

**return** 1.0\*err/(count-25);

}

**public** **static** **void** main(String args[]){

Ex10 ex=**new** Ex10();

ex.getPrime();

ex.PrintPrimes();

}

}

**第二章 近似算法**

**EX：**

**问题：**

**G中最大团的size为α当且仅当Gm里最大团的size是mα。**

**答：**

**证明：**

记G中最大团为G’，G的边集合为E，则的size为，根据最大团的定义，对于，总有使得,故G的m次拷贝的最大团G’的m次拷贝，即里最大团的size是m。

根据的定义，任何一个点与其他拷贝中的所有点都是相连的，可以知道对于，若最大团的size为m，则对应点集合可以写成

，其中刚好对应于G中的点vi，即的最大团刚好对应于G的最大团，其点集合为，假设没有这种对应关系，则必然存在一点u属于G的最大团，然而并不属于最大团，可以发现u与的最大团中所有点都相连，则可以把u加入的最大团，故这种对应关系存在，。