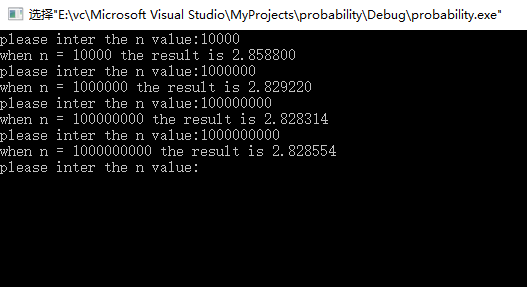
**概率算法**

Ex1.若将y<-uniform(0, 1)改为y<-x, 则上述的算法估计的值是什么？

首先因为(x,y)满足x\*x+y\*y<=1,所以(x,y)还是单位圆的点，其次因为y = x,所以点分布在y=x这条直线上，综上所述现在算法所求的值变为了求y=x在单位圆内的长度与y=x在单位正方形内的长度的比值的四倍，即。

以下是执行结果：



以下是具体的代码:

Main.c

#include<stdio.h>

#include<stdlib.h>

#include<time.h>

double darts(long n){

long k=0;

long i;

double x,y;

srand(time(0));

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

x = (double)(rand())/(RAND\_MAX+1.0);

y = x;

if(x\*x+y\*y <= 1){

k++;

}

}

return (double)(4.0\*k)/n;

}

int main(){

long n;

double result;

while(true){

printf("please inter the n value:");

scanf("%ld", &n);

result = darts(n);

printf("when n = %d the result is %f\n", n, result);

}

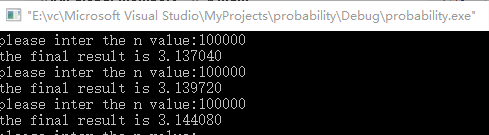
return 0;

}

Ex2.在机器上用估计π值，给出不同的n值及精度

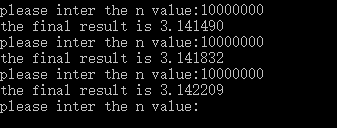
由π=3.141592....进行精度估算

当n= 100000时，执行三次结果为：



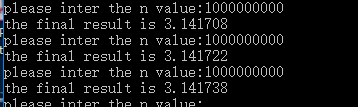
可以看出此时的精度为1位小数

当n= 10000000时，执行三次结果为：



可以看出此时的精度为2位小数

当n= 1000000000时，执行三次结果为：



可以看出此时的精度为3位小数

以下是代码：

Main.c

#include <stdlib.h>

#include <stdio.h>

#include "function.h"

#include <time.h>

int main(){

//define the function point

double(\*f)(double x);

long k;

long i;

long n;

double x, y, result, finalresult;

//this is the random seed

srand(time(0));

//initial the function point

f = circle;

while(true){

printf("please inter the n value:");

scanf("%ld", &n);

k = 0;

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

x = (double)rand()/(RAND\_MAX + 1.0);

y = (double)rand()/(RAND\_MAX + 1.0);

result = (\*f)(x);

if(y <= result){

k++;

}

}

finalresult = (double)(4.0 \* k)/n;

printf("the final result is %f\n", finalresult);

}

return 0;

}

function.h

#ifndef \_\_FUNCTION\_H\_

#define \_\_FUNCTION\_H\_

double circle(double);

#endif

function.c

#include<math.h>

#include "function.h"

double circle(double x){

double y;

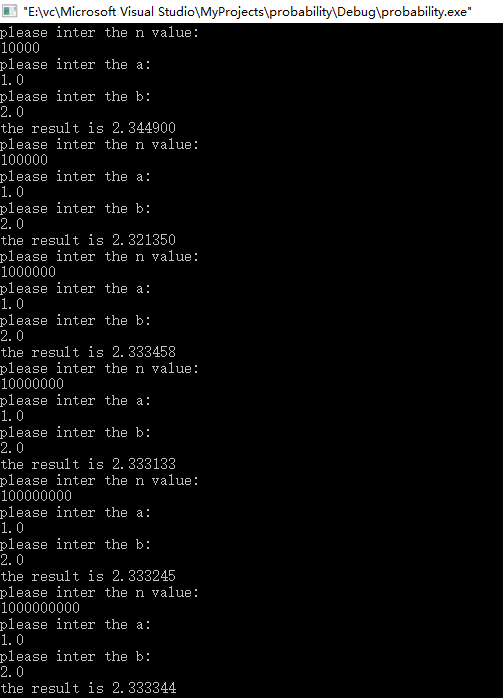
y = sqrt(1-x\*x);

return y;

}

Ex3.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：

这里我采用的函数是y = x\*x这个连续函数。计算当n取不同的值时它在x为1到2之间的积分。以下是结果：



以下是我的代码

Main.c

#include<stdio.h>

#include<stdlib.h>

#include "function.h"

#include<time.h>

//define the function point type

typedef double(\*funtype)(double);

double intergrations(double a, double b, long n, funtype ft){

long i;

long k = 0;

double result;

//the random seed

srand(time(0));

double c,d;

c = ft(a);

d = ft(b);

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

double x = (double)(rand()/(RAND\_MAX+1.0))\*(b-a)+a;

double y = (double)(rand()/(RAND\_MAX+1.0))\*(d-c)+c;

if(y<=ft(x)){

k++;

}

}

result = (double) (k\*1.0/n)\*(b-a)\*(d-c)+(b-a)\*c;

return result;

}

int main(){

double a;

double b;

long n;

double result;

while(true){

printf("please inter the n value:\n");

scanf("%ld", &n);

printf("please inter the a:\n");

scanf("%lf", &a);

printf("please inter the b:\n");

scanf("%lf", &b);

result = intergrations(a, b, n, continuty);

printf("the result is %f\n", result);

}

return 0;

}

Function.h

#ifndef \_\_FUNCTION\_H\_

#define \_\_FUNCTION\_H\_

double circle(double);

double continuty(double);

#endif

Function.c

#include<math.h>

#include "function.h"

double circle(double x){

double y;

y = sqrt(1-x\*x);

return y;

}

double continuty(double x){

double y;

y = x\*x;

return y;

}

Ex4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.

证明：为点落在圆内的点数量，则，所以有：

，

根据切比雪夫不等式：



那么

令，则

那么又因为



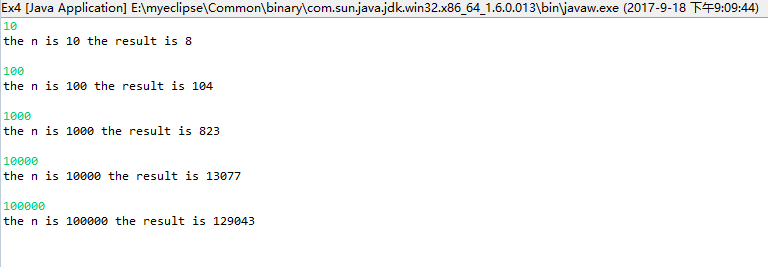
所以：

原命题得证

Ex5.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。

一般来说当n越大的时候，估计值是越准确的。并且我们应该取运行多次后的平均值作为n.

以下是我的结果：



以下是我的代码：

**package** probability;

**import** java.io.IOException;

**import** java.util.HashSet;

**import** java.util.Iterator;

**import** java.util.Random;

**import** java.util.Scanner;

**import** java.util.Set;

**public** **class** Ex4 {

**private** **int** uniform(Set<Integer> x){

Random rand = **new** Random();

**int** position = rand.nextInt(x.size());

**int** result = 0;

Iterator it = x.iterator();

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

result = (Integer) it.next();

}

**return** result;

}

**private** **int** setCount(Set<Integer> x){

**int** k =0;

Set<Integer> s = **new** HashSet<Integer>();

**int** a = uniform(x);

**do**{

k++;

s.add(a);

a = uniform(x);

}**while**(s.contains(a)==**false**);

**return** (**int**) ((**int**)2\*k\*k/Math.*PI*);

}

**private** Set<Integer> initialSet(Set<Integer> x, **int** n){

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

x.add(i);

}

**return** x;

}

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

Ex4 ex = **new** Ex4();

Set<Integer> x = **new** HashSet<Integer>();

**int** n = 0;

**while**(**true**){

Scanner s = **new** Scanner(System.*in*);

n = s.nextInt();

ex.initialSet(x, n);

**int** sum = 0;

//set the average value as result

**for**(**int** i = 0; i<100;i++){

sum += ex.setCount(x);

}

**int** result = sum/100;

System.*out*.println("the n is "+n+" the result is "+result+"\n");

}

}

}

EX6.分析dlogRH的工作原理，指出该算法相应的u和v。

它的工作原理是：利用sherwood算法，首先进行预处理将求解x的问题转化为求解的问题；然后利用确定算法求解c，得到c问题的解是y = r+x；最后我们利用变换x= y-r，即可得到问题的解，因为我们设置的x的值是0<=x<=p-1，所以需要将最后的结果mod(p-1)，也就是说最后的解为x = (y-r)mod(p-1)。

该算法中, , 

EX7.写一sherwood算法C，与算法A,B,D比较，给出实验结果

以下是实验结果（它展示的是不同程序查询不同值时，search循环执行的次数）：

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 程序名 | 2 | 3 | 13 | 1 | 5 | 21 | 8 | 平均次数 |
| A | 1 | 2 | 5 | 0 | 3 | 6 | 4 | 3 |
| B | 0 | 0 | 0 | 0 | 1 | 1 | 2 | 4/7 |
| C | 1 | 1 | 2 | 0 | 0 | 0 | 1 | 5/7 |
| D | 1 | 1 | 4 | 0 | 3 | 5 | 0 | 2 |

以下是我的代码：  
**package** probability;

**import** java.util.Random;

**import** java.util.Scanner;

**class** linknode{

**public** **int** val;

**public** **int** ptr;

**public** **int** rank;

linknode(**int** val, **int** ptr, **int** rank){

**this**.val = val;

**this**.ptr = ptr;

**this**.rank = rank;

}

}

**public** **class** Search {

**public** **static** linknode[] *list* = **new** linknode[7];

**public** **static** **int** *head* = 3;

**public** **static** **int** *n* = 7;

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

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

i = *list*[i].ptr;

}

**return** i;

}

**public** **static** **long** A(**int** x){

**long** time = System.*currentTimeMillis*();

*search*(x, *head*);

**long** time2 = System.*currentTimeMillis*();

**return** time2-time;

}

**public** **static** **long** D(**int** x){

**long** time = System.*currentTimeMillis*();

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

**int** i = r.nextInt(7);

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

**if**(x<y){

*search*(x, *head*);

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

*search*(x, *list*[i].ptr);

}

**long** time2 = System.*currentTimeMillis*();

**return** time2-time;

}

**public** **static** **long** B(**int** x){

**long** time = System.*currentTimeMillis*();

**int** i = *head*;

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

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

**int** y = *list*[j].val;

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

i = j;

max = y;

}

}

*search*(x, i);

**long** time2 = System.*currentTimeMillis*();

**return** time2-time;

}

**public** **static** **long** C(**int** x){

**long** time = System.*currentTimeMillis*();

**int** i = *head*;

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

Random rand = **new** Random();

**int** number = (**int**)Math.*sqrt*(1.0\**n*);

**int** number2 = *n* -number;

**int** r = rand.nextInt(number2);

**for**(**int** j = r;j<=r+number;j++){

**int** y = *list*[j].val;

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

i = j;

max = y;

}

}

*search*(x, i);

**long** time2 = System.*currentTimeMillis*();

**return** time2-time;

}

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

linknode node1 = **new** linknode(2, 1, 2);

linknode node2 = **new** linknode(3, 4, 3);

linknode node3 = **new** linknode(13, 5, 6);

linknode node4 = **new** linknode(1, 0, 1);

linknode node5 = **new** linknode(5, 6, 4);

linknode node6 = **new** linknode(21, -1, 7);

linknode node7 = **new** linknode(8, 2, 5);

*list*[0] = node1;

*list*[1] = node2;

*list*[2] = node3;

*list*[3] = node4;

*list*[4] = node5;

*list*[5] = node6;

*list*[6] = node7;

Scanner ins = **new** Scanner(System.*in*);

**int** x = ins.nextInt();

**long** l1 = *A*(x);

**long** l2 = *B*(x);

**long** l3 = *C*(x);

**long** l4 = *D*(x);

System.*out*.println(l1);

System.*out*.println(l2);

System.*out*.println(l3);

System.*out*.println(l4);

}

}

EX8.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。

证明如下：

假设总共有n个位置是开放的，选中第k位置的概率是p(k)，当前的循环数是nb:

我们想要证明算法QueensLV选中其中任一位置的概率相等只要证明当nb=n时，p(k)=1/n

当nb=k时，才有可能选中第k个位置，此时选中此位置的概率是1/k;当nb=nb+1时，继续选中k位置的概率是p(k)=1/k\*(k/(k+1))=1/(k+1),之所以将原来的值乘以k/(k+1),是表示随机算法不选择1的概率，因为只有选择1时才会改变选中的位置，以此类推，想要最终选择的位置是k,也就是当nb=n时，选择的位置是k,此时的p(k)=1/k\*(k/(k+1))\*....((n-2)/(n-1))\*((n-1)/n)=1/n。所以，原命题得证。以下是简单举例：  
 当k=1时：

P(k=1) = 1\*(1/2)\*(2/3)\*......\*((n-1)/n)=1/n

当k=2时：  
 p(k=2)=(1/2)\*(2/3)\*......\*((n-1)/n)=1/n

当k=3时：

P(k=3)=(1/3)\*(3/4)\*.....\*((n-1)/n)=1/n

.....

当k=n时：

P(k=n)=1/n

EX9.写一算法，求n=12~20时最优的StepVegas值

我求出了对应的n，当StepVegas取不同值时，程序执行10次的平均需要放置皇后的次数。

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| step | n | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
| 1 | | 777 | 718 | 2029 | 2096 | 8752 | 7296 | 25947 | 8517 | 31324 |
| 2 | | 509 | 675 | 934 | 1409 | 5341 | 7615 | 18488 | 9787 | 38564 |
| 3 | | 299 | 313 | 862 | 1058 | 1619 | 2695 | 7506 | 8229 | 54494 |
| 4 | | 113 | 162 | 432 | 742 | 712 | 1320 | 7188 | 4661 | 6081 |
| 5 | | 79 | 98 | 169 | 377 | 453 | 600 | 2191 | 4640 | 3547 |
| 6 | | 117 | 248 | 277 | 292 | 256 | 185 | 1397 | 843 | 2578 |
| 7 | | 99 | 124 | 90 | 124 | 212 | 462 | 1066 | 969 | 1074 |
| 8 | | 131 | 174 | 210 | 80 | 397 | 188 | 300 | 720 | 998 |
| 9 | | 173 | 113 | 187 | 279 | 252 | 141 | 340 | 619 | 1024 |
| 10 | | 132 | 264 | 126 | 186 | 135 | 133 | 307 | 310 | 513 |
| 11 | | 161 | 129 | 339 | 302 | 213 | 261 | 211 | 253 | 357 |
| 12 | | 153 | 236 | 251 | 404 | 144 | 348 | 221 | 359 | 314 |
| 13 | |  | 171 | 245 | 501 | 332 | 640 | 312 | 271 | 197 |
| 14 | |  |  | 446 | 266 | 554 | 278 | 357 | 406 | 242 |
| 15 | |  |  |  | 272 | 218 | 424 | 396 | 413 | 330 |
| 16 | |  |  |  |  | 220 | 412 | 640 | 808 | 534 |
| 17 | |  |  |  |  |  | 621 | 573 | 206 | 705 |
| 18 | |  |  |  |  |  |  | 845 | 1092 | 601 |
| 19 | |  |  |  |  |  |  |  | 613 | 768 |
| 20 | |  |  |  |  |  |  |  |  | 769 |

从表格中我们可以看到以下的结果

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| n | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
| best | 5 | 5 | 7 | 8 | 10 | 10 | 11 | 11 | 13 |

以下是我的代码：  
**package** probability;

**import** java.util.HashSet;

**import** java.util.Random;

**import** java.util.Scanner;

**import** java.util.Stack;

**public** **class** QueenLv {

**long** cal = 0;

**boolean** record = **false**;

**public** **int**[] obstinate(**int** n , **int** stepvegas){

**int**[] trysed = **new** **int**[n+1];

**boolean** success = **false**;

**do**{

trysed = queenLv(n, stepvegas, success);

}**while**(record==**false**);

**return** trysed;

}

**public** **boolean** backtrace(**int** k, HashSet col, HashSet diag45, HashSet diag135, **boolean** success, **int** n, **int**[] trys){

**boolean** flag = **true**;

Stack<Integer> stack = **new** Stack<Integer>();

**int** i = k+1;

**int** j = 1;

**int** m = 0;

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

**for**(m = j;m<=n;m++){

**if**(col.contains(m)==**false** && diag45.contains(m-i)==**false** && diag135.contains(m+i)==**false**){

**break**;

}

}

**if**(m<=n){

col.add(m);

diag45.add(m-i);

diag135.add(m+i);

stack.push(m);

trys[i]=m;

cal++;

i++;

j = 1;

}**else**{

**if**(stack.isEmpty() == **false**){

**int** value = stack.pop();

i--;

trys[i]=0;

cal++;

col.remove(value);

diag45.remove(value-i);

diag135.remove(value+i);

j = value+1;

}**else**{

flag = **false**;

**break**;

}

}

}

**return** flag;

}

**public** **int**[] queenLv(**int** n, **int** stepvegas, **boolean** success){

HashSet col = **new** HashSet();

HashSet diag45 = **new** HashSet();

HashSet diag135 = **new** HashSet();

Random rand = **new** Random();

**boolean** flag = **true**;

**int**[] trys = **new** **int**[n+1];

**int** k = 0;

**int** j = 0;

**int** nb = 0;

**do**{

nb = 0;

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

**if**(col.contains(i)==**false** && diag45.contains(i-k-1)==**false** && diag135.contains(i+k+1)==**false**){

nb++;

**if**((rand.nextInt(nb)+1)==1){

j = i;

}

}

}

**if**(nb > 0){

k++;

trys[k] = j;

cal++;

col.add(j);

diag45.add(j-k);

diag135.add(j+k);

}

}**while**(nb!=0&&k!=stepvegas);

**if**(nb>0){

flag = backtrace(k, col, diag45, diag135, success, n, trys);

success = flag;

record = flag;

**return** trys;

}**else**{

success = **false**;

record = **false**;

**return** **new** **int**[n];

}

}

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

**int** n;

Scanner scan = **new** Scanner(System.*in*);

n = scan.nextInt();

**int** stepvegas;

**long**[][] recordarray = **new** **long**[10][n];

**for**(**int** j = 0; j<10;j++){

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

stepvegas = i;

**int**[] trys = **new** **int**[n+1];

QueenLv q = **new** QueenLv();

trys = q.obstinate(n, stepvegas);

recordarray[j][i-1]=q.cal;

}

}

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

**long** temp = 0;

**for**(**int** j = 0;j<10;j++){

temp += recordarray[j][i];

}

temp/=10;

System.*out*.println(temp);

}

}

}

EX10.

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;

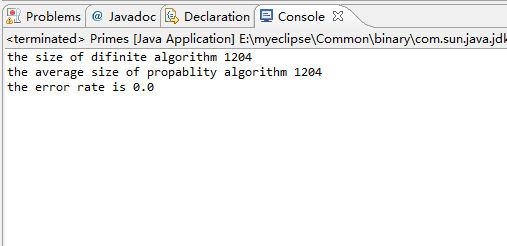
until n=10000;

}

与确定性算法相比较，并给出100~10000以内错误的比例。

我将printPrimes执行了1000，取它的平均值作为结果，从下面的结果图可以看出它的错误率为0.0%，同时确定算法得到的100-10000之间的素数有1204，用概率算法得到的100-1000之间的素数平均个数也是1204。

结果图：



以下我的java代码：

**package** probability;

**import** java.util.ArrayList;

**import** java.util.Random;

**public** **class** Primes {

Random rand = **new** Random();

**public** ArrayList printPrimesSimple(){

ArrayList lists = **new** ArrayList();

**for**(**int** j = 101; j<10000;j++){

**int** flag = 1;

**for**(**int** i=2; i<=(**int**)Math.*sqrt*(j);i++){

**if**(j%i==0){

flag = 0;

**break**;

}

}

**if**(flag==1){

lists.add(j);

}

}

**return** lists;

}

**public** ArrayList printPrimes(){

**int** n = 101;

ArrayList lists = **new** ArrayList();

**do**{

**if**(repeatMillRab(n, (**int**)(Math.*log*(n)/Math.*log*(2)))==**true**){

lists.add(n);

}

n = n+2;

}**while**(n<10000);

**return** lists;

}

**public** **boolean** repeatMillRab(**int** n, **int** b){

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

**if**(millRob(n)==**false**){

**return** **false**;

}

}

**return** **true**;

}

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

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

a = 2 + (a+1)\*(n-2-2)/n;

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

}

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

**int** s = 0;

**int** t = n-1;

**do**{

s++;

t /= 2;

}**while**(t%2!=1);

**int** x = modularExponent(a, t, n);

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

**return** **true**;

}

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

x = (x\*x)%n;

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

**return** **true**;

}

}

**return** **false**;

}

**public** **int** modularExponent(**int** a, **int** t, **int** n){

**int** s = 1;

**while**(t>0){

**if**(odd(t)){

s = (s\*a)%n;

}

a = (a\*a)%n;

t = t/2;

}

**return** s;

}

**public** **boolean** odd(**int** t){

**if**(t%2==0){

**return** **false**;

}**else**{

**return** **true**;

}

}

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

Primes p = **new** Primes();

ArrayList temp1 = **new** ArrayList();

ArrayList temp2 = **new** ArrayList();

temp2 = p.printPrimesSimple();

**long** allcorrect = 0;

**long** averagenum = 0;

**int** k = 1000;

**while**(k>0){

temp1.clear();

temp1 = p.printPrimes();

**int** correct = 0;

**for**(**int** i = 0;i<temp1.size();i++){

**if**(temp2.contains(temp1.get(i))){

correct++;

}

}

allcorrect += correct;

averagenum += temp1.size();

k--;

}

System.*out*.println("the size of difinite algorithm "+temp2.size());

System.*out*.println("the average size of propablity algorithm "+(averagenum/1000));

**double** correctRate = (**double**)allcorrect/(temp2.size()\*1000);

System.*out*.println("the error rate is "+(1.0 - correctRate));

}

}