**概率算法**

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

解：题中要求将y不用随机获得，而是令y=x,x通过随机函数uniform获得,这就代表y=x 这条直线在圆内的长度和在正方形内的长度的比例，即2 。

程序运行的数据如下：当n=100000时，算出的值为2.828513；当n=1000000时，算出的值为2.828467；当n=100000时，算出的值为2.828312

**Ex2.在机器上用估计值，给出不同的值及精度**

关键代码：

srand((unsigned)time(NULL));

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

{

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

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

if(y<=sqrt(1-x\*x))

{

k++;

}

}

程序运行的数据如下：当n=100000时，算出的值为3.141756；当n=1000000时，算出的值为3.141533；当n=100000时，算出的值为3.141542

**Ex3.设和是实数，且，，是一个连续函数，写一概率算法计算积分：**

关键代码：

void HitorMiss(double n,double a,double b,double c,double d,double (\*fa)(double))

{

double k=0;

srand((unsigned)time(NULL));

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

{

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

double y=(double)rand()/(double)RAND\_MAX\*d;

if(y<=(\*fa)(x))

{

k++;

}

}

cout<<"n的值："<<n<<"\t估计值："<<k/n<<endl;

}

void main()

{

HitorMiss(100,1,2,1,2,f);

HitorMiss(1000,1,2,1,2,f);

HitorMiss(10000,1,2,1,2,f);

HitorMiss(100000,1,2,1,2,f);

HitorMiss(1000000,1,2,1,2,f);

HitorMiss(10000000,1,2,1,2,f);

HitorMiss(100000000,1,2,1,2,f);

HitorMiss(1000000000,1,2,1,2,f);

HitorMiss(10000000000,1,2,1,2,f);

HitorMiss(100000000000,1,2,1,2,f);

}

其中，函数f(x)=x;a-1,b=2,c=1,d=2

**Ex4设ε,δ是(0,1)之间的常数，证明：若I是∫𝑓(𝑥)𝑑𝑥1 0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估计整数子集1~n的大小，并分析n对估计值的影响。**

关键代码：

int uniform(set<int> X)

{

int m = rand() % X.size();

set<int>::iterator iter = X.begin();

for (int i = 0; i < m; iter++, i++);

return \*iter;

}

int SetCount(set<int> X)

{

set<int> S;

int a;

while (S.find(a = uniform(X))==S.end())

{

S.insert(a);

}

return (int)(2 \* S.size()\*S.size() / PI);

}

int main()

{

set<int> X;

srand((unsigned)time(NULL));

for (int i = 0; i < SizeX; i++){

X.insert(i);

}

int N;

for (N = 10; N <= 1000; N \*= 10){

int Sum = 0;

for (int i = 0; i < N; i++){

Sum += SetCount(X);

}

printf("N=%d Xsize = %d\n",N,(Sum/N));

}

return 0;

}

算法运行结果：

程序运行的数据如下：当n=10时，算出的值为1483；当n=100时，算出的值为1205；当n=1000时，算出的值为1136

**Ex6 分析dlogRH 的工作原理,指出该算法相应的u 和v**

解： Sherwood 算法的一般过程：

1). 将被解的实例变换到一个随机实例。 //预处理函数u

2). 用确定算法解此随机实例，得到一个解。

3). 将此解变换为对原实例的解。 //后处理函数v

dlogRH 是Sherwood 算法的一个具体应用，dlogRH 为了消除输入实例中a 的取

值对执行时间的影响对其中的a=gx mod p 作随机预处理，得到与其对应的随机

实例c=u(x, r)，并且对c 使用确定性算法得到y ，最后再把随机实例的结果y 变

换为输入实例a 的解x=v(y, r)。其中

u: u(x, r) = logg,p c = (r+x) mod (p-1)

v: v(y, r) = (y-r) mod (p-1)

**Ex7 写一Sherwood 算法C，与算法A, B, D 比较，给出实验结果。**

关键代码：

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;

}

实验结果：

A算法需要18次，B算法需要2次；C算法需要3次；D算法需要6次。

**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： 解：当放置第（k+1）th 皇后时，如果有n 个位置开放，依次记为{S1,S2,…,Sn}。

下面计算选择位置Si 的概率Pi ：Si 被选中，则uniform(1,…,i)=1,且对于所有j>i

有uniform(1,…,j)=0。显然uniform(1,…,i)=1 的概率为1/i, uniform(1,…,j)=0 的概

率为(j-1) / j。所以



故对于皇后，若有个开放位置，则每个位置被选中的概率都相等

**Ex9. (4.1 8 后问题)写一算法, 求时最优的StepVegas值**

关键代码：

bool notbelong(int\*a,int l,int x)

{

for(int i=0;i<l;i++)

{

if(a[i]==x)

return false;

}

return true;

}

bool backtrace(int k,int co1,int d451,int d1351)

{

if(k==n+1)

return true;

for(int j=1;j<=n;j++)

{

if(notbelong(col,co1,j)&&notbelong(diag45,d451,j-k)&&notbelong(diag135,d1351,j+k))

{

tryy[k]=j;

col[co]=j;

diag45[d45]=j-k;

diag135[d135]=j+k;

if(backtrace(k+1,co1+1,d451+1,d1351+1))

{

return true;

}

}

}

return false;

}

bool queenslv(int stepvegas)

{

Sleep(1000);

co=0;

d45=0;

d135=0;

int k=0;

int nb;

int j;

srand((unsigned)time(NULL));

do

{

nb=0;

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

{

if(notbelong(col,co,i)&&notbelong(diag45,d45,i-k-1)&&notbelong(diag135,d135,i+k+1))

{

nb+=1;

if((rand()%nb+1)==1)

j=i;

}

}

if(nb>0)

{

k+=1;

tryy[k]=j;

col[co]=j;

co++;

diag45[d45]=j-k;

d45++;

diag135[d135]=j+k;

d135++;

}

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

if(nb>0)

{

return backtrace(k+1,co,d45,d135);

}

return false;

}

根据程序对12~20 个皇后在不同stepVegas 取值情况下的运行结果：

棋盘大小 stepVegas

12 3

13 4

14 5

15 6

16 7

17 7

18 8

19 9

20 9

**Ex10 素数测定**

PrintPrimes{ //打印1万以内的素数

print 2，3；

n ←5；

repeat

if RepeatMillRab(n,⌊lgn⌋ ) 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。

关键代码：

int modular\_exponent(int a, int b, int n) {

int ret = 1;

for( ;b; b>>=1,a=(int) ((i64)a)\*a%n )

if(b&1)

ret = (int)((i64)ret)\*a%n;

return ret;

}

int log(int n, int a)

{

int count = 0;

while( (n>>=1) >= 1) count ++ ;

return count;

}

/\* returns true means n is a prime or a strong

psudo prime. note: n is odd, and a is a number

between 2 to n - 2 \*/

bool Btest(int a, int n)

{

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

do{

s++; t >>= 1;

}while(t&1 != 1);

int x = modular\_exponent(a, t, n);

if(x == 1 || x == n - 1) return true;

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

{

x = modular\_exponent(x, 2, n);

if(x = n - 1) return true;

}

return false;

}

bool MillRab(int n)

{

int a = rand()%(n-3) + 2;

return Btest(a, n);

}

bool RepeatMillRab(int n, int k)

{

while(k--)

if(!MillRab(n)) return false;

return true;

}

int PrintPrimes()

{

int count = 2;

for(int n = 5; n < 10000; n += 2)

if(RepeatMillRab(n, log(n,2))){

count++;

}

return count;

}

/\* search prime numbers using deterministic

algorithm \*/

int plist[1300], pcount = 0;

int prime(int n)

{

int i;

if((n!=2&&!(n%2))||(n!=3&&!(n%3))||(n!=5&&!

(n%5))||(n!=7&&!(n%7)))

return 0;

for(i = 0; plist[i]\*plist[i] <= n; i++)

if(!(n%plist[i]))

return 0;

return n > 1;

}

void initprime()

{

int i;

for(plist[pcount++]=2,i=3;i<10000;i++)

if(prime(i))

plist[pcount++]=i;

}

为尽量减少偶然误差，程序中设置调用PrintPrime为4096次，最后计算时取这4096次的平均值，这样多次调用结果的误差控制在了0.000068%左右。

**近似算法作业**

**EX1 G中最大团的size为α当且仅当Gm里最大团的size是mα。**

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，。

**EX2 完善多机调度的LPT算法性能（近似）比：的证明。**

证明：课件中有一部分证明。

当m > 1,假设定理不成立,即有

对于违反该定理具有最少作业数的实例I.

有

这与假设矛盾。则说明在违反该定理的实例中存在一个作业数最少的实例满足要求。由于违反定理实例集合的作业的数量集合为全序。若不存在最小值，则说明集合为空。即不存在违反该定理的实例。所以对于m > 1时，定理依然成立。

**分布式算法作业**

**2.1 分析在同步和异步模型下，汇集算法的复杂性**

①在同步模型中，在汇集算法的每个容许执行里，树中每个高为t子树的根结点在第 t 轮里收到所有孩子的消息msg；使用归纳法证明如下：

归纳基础：t=1，每个叶子节点pi发送消息至其双亲，即每个叶子节点的双亲均受到来自市叶子节点的孩子发来的信息；

归纳假设：假设树中每个高为 t-1>=1子树的根结点在第 t -1轮里收到所有孩子的消息msg

归纳步骤：设pj到叶子节点的距离为t, pj是pm 的双亲，因pm到pi的距离为t-1，由归纳假设，在第t-1轮pm收到消息。由算法描述知，在第t轮里pj收到来自于pi的消息

②在异步模型中，在汇集算法的每个容许执行里，树中每个高为t子树的根结点在第 t 轮里收到所有孩子的消息msg；使用归纳法证明如下：

归纳基础：对t=1，初始时，消息处在从pi到所有距离叶子节点为1的处理器的传输之中，由异步模型的时间复杂性定义知，pi的双亲至多在时刻1收到消息。

归纳假设：假设树中每个高为 t-1>=1子树的根结点在至多在时刻 t -1收到所有孩子的消息

归纳步骤：pj∈ {距pi为t的处理器}，设pj是pm的双亲，则pm与叶子节点pi的距离为t-1，由归纳假设知，pm至多在时刻t-1收到M，由算法描述知，pi发送给pj的消息至多在t时刻到达。

**2.2 证明在引理2.6中，一个处理器在图G中是从Pr可达的，当且仅当它的parent变量曾被赋过值**

①首先证明一个处理器在图G中是从Pr可达的，可以退出它的parent变量曾被赋过值：因为从pr可达，（因为图G是由parent与children确定的静止图）收到m才会加入图中，所以可达结点收到过m，执行了alg2.2第5行，即upon receiving M from neighbor pj。由于是容许执行，第7行，即parent：=j也会执行。也就是被赋值。

②再来证明一个处理器的parent变量曾被赋过值，能推出它在图G中是从Pr可达的：因为的parent变量曾被赋过值，所以alg2.2第7行执行过，由于是容许执行，第5行也执行过，即收到过m，而m是由pr发出的，所以它在图G中是从Pr可达的。

**2.3 证明Alg2.3构造一颗以Pr为根的DFS树**

①证明连通性：

反证：假设某结点在G中从pr不可达，因网络是连通的，若存在两个相邻的结点pi和pj使得pj在G中是从pr可达的(以下简称pj可达)，但pi不可达。因为G里一结点从pr可达当且仅当它曾设置过自己的parent变量(Ex 证明)，所以pi的parent变量在整个执行中仍为nil，而pj在某点上已设置过自己的parent变量，于是pj发送M到pi(line9)，因该执行是容许的，此msg必定最终被pi接收，使pi将自己的parent变量设置为j。矛盾！

②证明无环：

假设有一环，pi1,…pikpi1，若pi是pj的孩子，则pi在pj第1次收到M之后第1次收到M。因每个处理器在该环上是下一处理器的双亲，这就意味着pi1在pi1第1次接收M之前第1次接收M。矛盾！

③证明是DFS树：

只需证明在有子结点与兄弟结点未访问时，子结点总是先加入树中。

设有节点P1，P2和P3。P2和P3是P1的直接相邻节点。P1在第12~14行中先选择向P2发送M，则P1当且仅当P2向其返回一个<parent>（第17行，第22行）时才有可能向P3发送M。而P2仅在其向所有的相邻节点发送过M后才会向P1返回<parent>。所以P2的子节点是永远先于P3加入树中的，即G是DFS树。

**2.4 证明Alg2.3的时间复杂性为O(m)**

①同步模型：每一轮中，根据算法，有且只有一个消息在传输，从算法的第6 、14、16、20、25行发送消息的语句中可以发现：消息只发往一个处理器结点，除根结点外，所有的处理器都是收到消息后才被激活，所以，不存在多个处理器在同一轮发送消息的情况，所以时间复杂度与消息复杂度一致。

②异步模型：在一个时刻内至多有一个消息在传输，因此，时间复杂度也与消息复杂度一致。

③消息复杂度：对任一边，可能传输的消息最多有4个，即2个msg ，2个回应消息（parent or reject），所以消息复杂度为O(m)

综上，该算法的时间复杂度为O(m)。

**2.5 修改Alg2.3获得一新算法，使构造DFS树的时间复杂性为O(n)**

解：当一个节点认定了它的双亲时，它向其他邻居发一个广播消息，让其他邻居将这个节点从未访问过的节点集合中删去，这样便不会再向它发消息，每个节点只用三个单位的时间，总时间变为3n;规范化语言为：在每个处理器中维护一个本地变量，同时添加一个消息类型，在处理器Pi转发M时，发送消息N通知其余的未访问过的邻居，这样其邻居在转发M时便不会向Pi转发；或在消息M和<parent>中维护一个发送数组，记录已经转发过M的处理器名称。

两种方式都是避免向已转发过M的处理器发送消息M，这样DFS树外的边不再耗时，时间复杂度也降为O(n)。

**3.1证明同步环上不存在匿名的、一致性的Leader选举算法**

假设A是同步环上的一个匿名算法，每个处理器在系统中具有相同的状态机，如果它选中某处理器为leader，因为环是同步的且只有一种初始配置，故在R上A只有唯一的合法执行。Lemma3.1： 在环R上算法A的容许执行里，对于每轮k，所有处理器的状态在第k轮结束时是相同的。即在每轮里，各处理器均发出同样的message，所以在各轮里各个处理器接收到相同的message，则状态改变也相同。导致每个处理器同时宣布自己是Leader。所以同步环系统中匿名的、一致性的领导者选举算法的算法是不存在的。

**3.2 证明异步环系统中不存在匿名的Leader选举算法**

每个处理器的初始状态相同，状态机相同，接收的消息序列也相同（只有接收消息的时间可能不同），故最终处理器的状态一致。由于处理一条消息的至多需要1时间单位，若某时刻某个处理器宣布自己是Leader （接收到m条消息），则在有限时间内（m时间单位）其他处理器也会宣布自己是Leader。所以异步环系统中不存在匿名的Leader选举算法不存在。

3.9**若将环Rrevn划分为长度为j(j是2的方幂)的连续片段，则所有这些片段是次序等价的**

证明：对一个整数P(0≤P≤n−1),可以表示为：

其中m=lg n,则有rev(P)=。

设P、Q在同一个片段上，P1、Q1在同一片段上，且设这两个片段时相邻的，由模运算的加法可得：P1=P+ l；Q1=Q + l。l表示片段的长度，l=2k。

又因为：

且P、Q在同一个片段上，有

|P-Q|<l=2k

所以存在r(0≤r≤k),满足 ar ≠ br。否则，|P−Q|≥l。这与P、Q在同一个片段上矛盾。

设s=min⁡{r},则根据rev(P),rev(Q)的表示方法可得：

sign(rev(P)-rev(Q))=sign (as-bs)

而

显然，P与P1的前k位相同，Q与Q1的前k位相同。由0≤s≤k得

sign(rev(P1)-rev(Q1))=sign (as-bs)

这两个相邻片段是序等价的，根据等价的传递关系，可得所有的片段都是次序等价