概率组合数学

七月算法 邹博

2015年11月8日

花开堪折直须折

- 婚介所登记了N位男孩和N位女孩，每个男孩都对N个女孩的喜欢程度做了排序，每个女孩都对N个男孩的喜欢程度做了排序。你作为月老，能否给出稳定的牵手方案？
  - 分析：每个男孩都对N个女孩做了排序，N个男孩的钟情度形成N×N的矩阵A；同时，N个女孩的钟情度形成N×N的矩阵B；
  - 如果男孩i和女孩a牵手，但男孩i对女孩b更喜欢，而女孩b的男友j拼不过男孩i，则没有力量阻碍男孩i和女孩b的私奔，这即是不稳定的。

题目解析

- 采取贪心的思路：
- 所有男孩同时对各自最喜欢的女孩表白：
  - 若女孩a只有1个男孩i表白，接受男孩i；
  - 若女孩a有多位男孩表白，选择她最喜欢的男孩x
  - 若女孩a没有男孩表白，静静等待下一轮。
- 第k轮，所有单身男孩对各自第k喜欢的女孩表白：
  - 无论该女孩是否有男友；
  - 若女孩a收到了男孩i的表白：
    - 若女孩a没有男友，则接受男孩i；
    - 若相对于现任男友j，女孩a更喜欢男孩i，则接受男孩i；

正确性分析：匹配性和稳定性

- 在某个时刻，任何一个女孩要么已经有了男友，要么还会有男孩对其表白(她此刻没有男友，说明还有单身男孩，所以，该女孩不用着急，只需等待)，她总可以在表白的男孩中选择一个自己最喜欢的，因此，任何一个女孩必然都能找到男友。
  - 因为“N-N”的比例关系，这也导致了每个男孩都能有女友。
- 不稳定情况：如果男孩i和女孩a牵手，但男孩i对女孩b更喜欢，而女孩b的男友j拼不过男孩i，则男孩i和女孩b的私奔。
  - 1、男孩i和女孩a牵手
  - 2、男孩i对女孩b更喜欢
  - 3、相对于男友j，女孩b的更喜欢男孩i
  - 打破上述三个前提中的一个：1和2说明男孩i曾经向女孩b表白却遭拒，这说明相对于男孩i，女孩b的更喜欢现任男友j。

Code

```
int _tmain(int argc, _TCHAR* argv[])
{
    int man[N][N] = {
        {2,3,1,0},
        {2,1,3,0},
        {0,2,3,1},
        {1,3,2,0},
    };  //男孩喜欢的女孩列表
    int woman[N][N] = {
        {0,3,2,1},
        {0,1,2,3},
        {0,2,3,1},
        {1,0,3,2},
    };  //女孩喜欢的男孩列表
    int match[N];   //男孩的女友
    GaleShapley(man, woman, match);
    Print(match, N);
    Validate(man, woman, match);
    return 0;
}
```

```
const int N = 4;
void GaleShapley(const int (&man)[N][N], const int (&woman)[N][N], int (&match)[N])
{
    int wm[N][N];       //wm[i][j]：女孩i对男孩j的排名
    int choose[N];      //choose[i]：女孩i当前的男朋友
    int manIndex[N];    //manIndex[i]：男孩i被多少个女孩拒绝过
    int i,j;
    int w,m;
    for(i = 0; i < N; i++)
    {
        match[i] = -1;      //所有男孩都初始化为光棍
        choose[i] = -1;     //所有女孩都初始化为光棍
        manIndex[i] = 0;    //为0，则意味着从最喜欢的女孩开始选
        for(j = 0; j < N; j++)
            wm[i][woman[i][j]] = j; //对男孩woman[i][j]的排名是第j位
    }

    bool bSingle = false;   //是否所有男孩都有了女友
    while(!bSingle)         //每个男孩当前轮选女友
    {
        bSingle = true;     //尚未发现单身男孩
        for(i = 0; i < N; i++)  //每个男孩i选择尚未被拒绝的最喜欢的女孩
        {
            if(match[i] != -1)  //男孩i已经有女友
                continue;
            bSingle = false;
            j = manIndex[i]++;
            w = man[i][j];      //男孩i第j喜欢的女孩w
            m = choose[w];      //女孩w当前的男友m
            if((m == -1) || (wm[w][i] < wm[w][m]))  //女孩w更喜欢男孩i
            {
                match[i] = w;
                choose[w] = i;
                if(m != -1)
                    match[m] = -1;  //女孩w抛弃前男友m
            }
        }
    }
}
```

改进相互表白的策略？

- 首轮a：所有男孩同时对各自最喜欢的女孩表白：
  - 若女孩a只有1个男孩i表白，接受男孩i；
  - 若女孩a有多位男孩表白，选择她最喜欢的男孩x
  - 若女孩a没有男孩表白，静静等待下一轮。
- 首轮b：所有女孩同时对各自最喜欢的男孩表白：
  - 策略同a
- 第k轮a：所有单身男孩对各自第k喜欢的女孩表白：
  - 无论该女孩是否有男友；
  - 若女孩a收到了男孩i的表白：
    - 若女孩a没有男友，则接受男孩i；
    - 若相对于现任男友j，女孩a更喜欢男孩i，则接受男孩i；
- 第k轮b：所有单身女孩对各自第k喜欢的男孩表白：
  - 策略同a

一个反例

- 男女孩各3位：
- 每个男孩的钟情度矩阵为：

	0	1	2
0	0	1	1
1	1	0	2
2	1	2	0

- 每个女孩的钟情度矩阵为：

	0	1	2
0	1	2	0
1	2	1	0
2	1	2	0

- 一个可行解为：2、0、1

算法的思考和总结

- 本算法能够解决这种情况吗?
  - 如果男孩i和女孩j牵手，但男孩i对女孩a更喜欢，而女孩a的男友b却更喜欢女孩j。
- 这即著名的对稳定婚姻策略的Gale-Shapley算法，由于女孩选择对其表白的男孩的最优者，因此也称为延迟认可算法。
  - 可用于学生志愿填报(员工选择职位、毕业生选择单位)等“二部”问题。
  - 如果没有“有向边”，则该算法不适用。

求1的个数

- 给定一个32位无符号整数N，求整数N的二进制数中1的个数。
  - 显然：可以通过不断的将整数N右移，判断当前数字的最低位是否为1，直到整数N为0为止。
    - 平均情况下，大约需要16次移位和16次加法。
  - 有其他更精巧的办法吗？

两种常规Code

- 思路1：
  - 每次右移一位
  - 奇数则累加1
- 思路2：
  - 每次最低位清0
  - 只需要n&=(n-1)即可

```
int OneNumber(int n)
{
    int c = 0;
    while(n != 0)
    {
        c += (n&1); //奇数则累加1
        n >>= 1;
    }
    return c;
}

int OneNumber2(int n)
{
    int c = 0;
    while(n != 0)
    {
        n &= (n-1); //最低为1的位清0
        c++;
    }
    return c;
}
```

分治

- 假定能够求出N的高16位中1的个数a和低16位中1的个数b，则a+b即为所求。
- 为了节省空间，用一个32位整数保存a和b：
  - 高16位记录a，低16位记录b，
  - (0xFFFF0000&N)筛选得到a；
  - (0x0000FFFF&N)筛选得到b；
  - (0xFFFF0000&N) + (0x0000FFFF&N)>>16
- 如何得到高16位中1的个数a呢？
  - 如何得到低16位中1的个数b呢？
  - 递归

递归过程

- 如果二进制数N是16位，则统计N的高8位和低8位各自1的数目a和b，而a、b用某一个16位数X存储，则使用0xFF00、0x00FF分别于X做与操作，筛选出a和b；
  - 原问题中的数据是32位，因此需要2个0xFF00/0x00FF，即0xFF00FF00/0x00FF00FF
- 如果二进制数是8位，则统计高4位和低4位各自1的数目，使用0xF0/0x0F分别做与操作，筛选出高4位和低4位；
  - 需要4个0xF0/0x0F，即0xF0F0F0F0/0x0F0F0F0F
- 如果是4位则统计高2位和低2位各自1的数目，用0xC/0x3筛选；
  - 需要8个0xC/0x3，即0xCCCCCCCC/0x33333333
- 如果是2位则统计高1位和低1位各自1的数目，用0x2/0x1筛选；
  - 需要16个0x2/0x1，即0xAAAAAAAA/0x55555555

Code

```
int HammingWeight(unsigned int n)
{
    n = (n & 0x55555555) + ((n & 0xaaaaaaaa)>>1);
    n = (n & 0x33333333) + ((n & 0xcccccccc)>>2);
    n = (n & 0x0f0f0f0f) + ((n & 0xf0f0f0f0)>>4);
    n = (n & 0x00ff00ff) + ((n & 0xff00ff00)>>8);
    n = (n & 0x0000ffff) + ((n & 0xffff0000)>>16);
    return n;
}
```

总结与应用

- HammingWeight使用了分治/递归的思想，将问题巧妙解决，降低了运算次数。
  - 还可以使用其他分组方案，如3位一组等。
- 如果定义两个长度相等的0/1串中对应位不相同的个数为海明距离(即码距)，则某0/1串和全0串的海明距离即为这个0/1串中1的个数。
- 两个0/1串的海明距离，即两个串异或值的1的数目，因此，该问题在信息编码等诸多领域有广泛应用。

猜数字游戏

- 两个聪明人A和B玩猜数字的游戏。他们在脑门上各贴一个正整数数字，两个数字只相差1，A和B只能看到对方的数组而看不到自己的。
- 以下是两人的对话：
  - A：我不知道
  - B：我也不知道
  - A：我知道了
  - B：我也知道了
- 上述4句对话结束后，聪明的你帮助A、B推算下，他们的数字各是多少呢？

引理

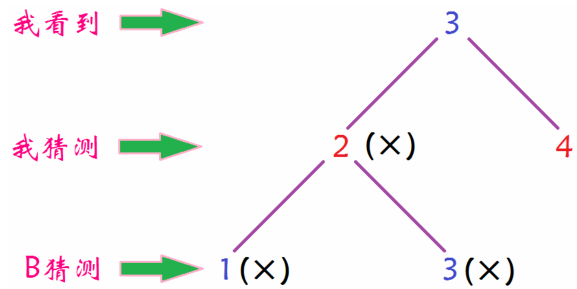
- 引理1：如果A看到B是1，则A马上可以断定：自己是2。
  - 因为条件给定了数字都是正整数。
- 引理2：如果A看到B是2，并且B说“我不知道”，则A马上可以断定：自己是3。
  - 因为A根据B是2可以推断自己是1或者3；
  - 如果自己是1，根据引理1，B会说“我知道”。

考察“A是3，B是2”这种情况

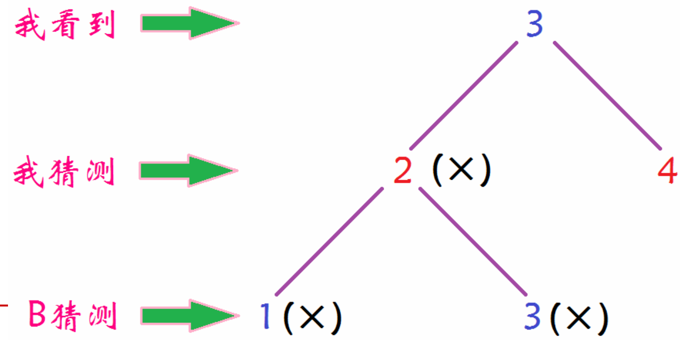
- 第一次，A看到2，无法判断自己是1或3，只好说不知道；
- 第二次，B看到3，无法判断自己是2或4，只好说不知道；
- 第三次，A得知了“B不知道”，因此，B看到的一定不是1(根据引理1)，所以，A断定了自己是3；
- 第四次，B得知了“A知道”，因此，A如果看到4是无法断定自己是3还是5，因此，A一定是看到了自己是2。

考察“A是4，B是3”这种情况

- 前两次，A看到3，B看到4，无法判断自己是几，都说不知道；
- 第三次，A的心理活动：
  - 可以断定我是4而不可能是2。理由如下：
  - 整理当前信息：假定我是2，且我说“我不知道”。根据引理2，B一定会断定自己是3。与当前B说“我不知道”矛盾。
- 第四次，B的心理活动：
  - 可以断定自己是3而不可能是5。理由如下：
  - 整理当前信息：假定A看到我是5，A会猜测他自己是4或者6
    - 如果A自己是4或者6，A和我都会顺次说“我不知道”；
    - 因此，A无法得出自己是5的结论。
  - 我不是5，那么我只能是3。



逻辑总结

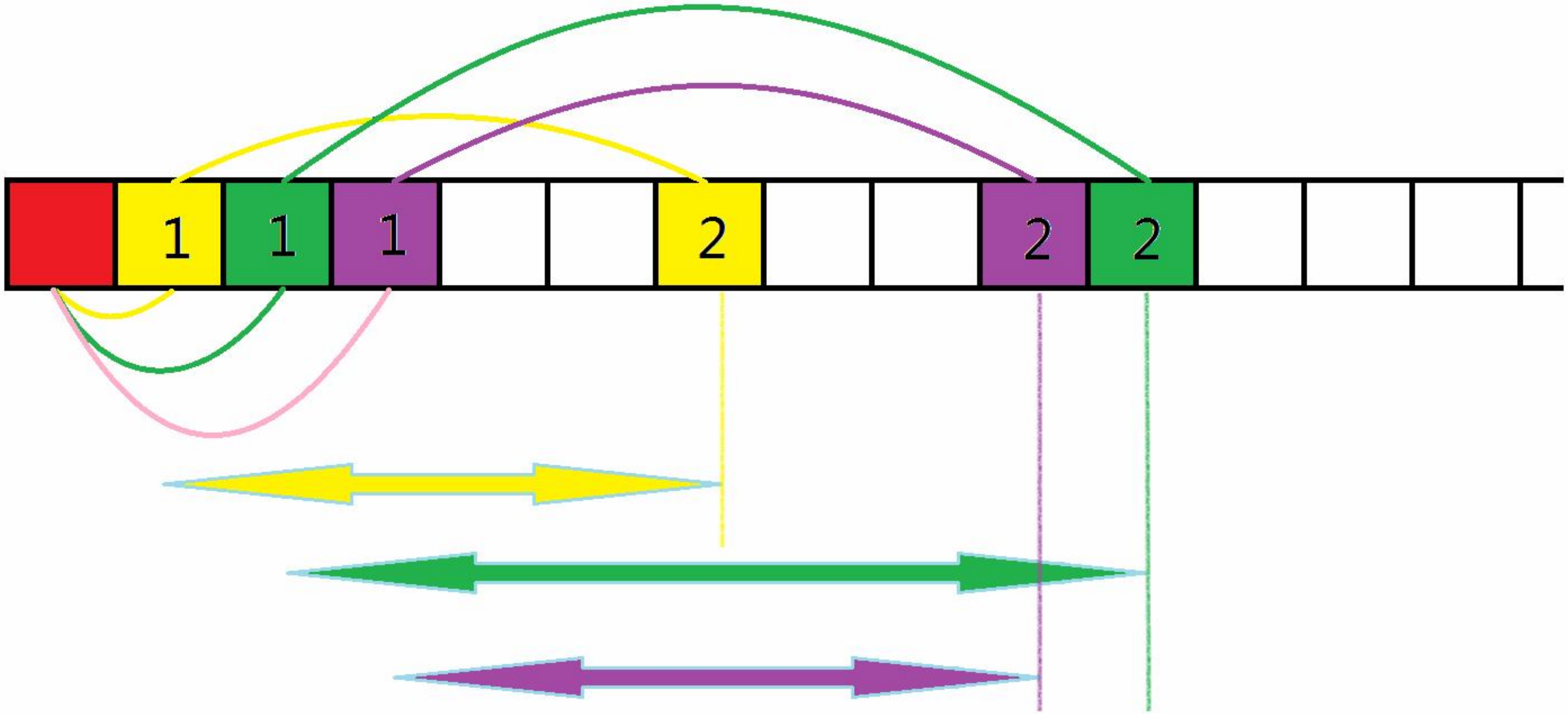


- 复杂的逻辑题可以用二叉树(N叉树)做辅助推理，原理是：只要某个结点的两个孩子(所有孩子)都不可能，则这个结点不可能。
  - 如A4B3情况下，A的推理树如右上图所示。
- 注意：两人的说话顺序是有决定作用的，是不对称消息：
  - A说话，则A是在看到B的内容后做判断，B可根据A的内容在自己的推理树上做剪枝。

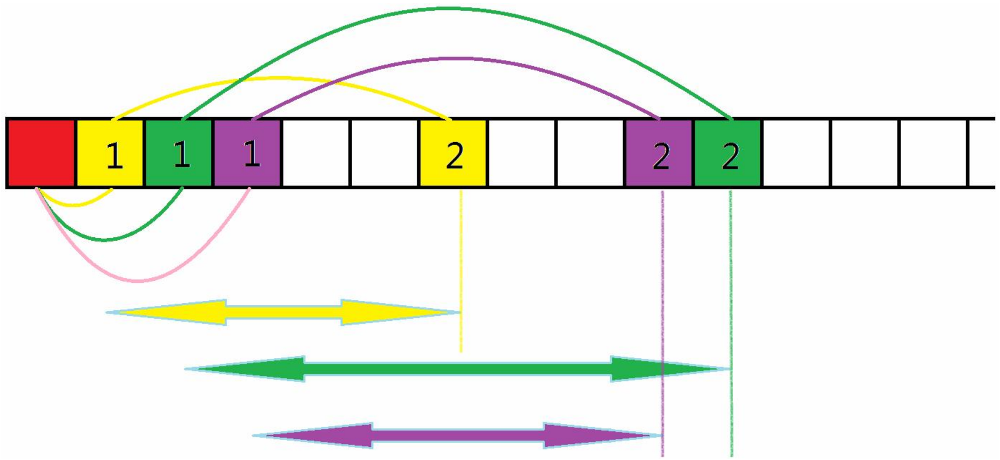
Jump

- 跳跃问题
- 给定非负整数数组，初始时在数组起始位置放置一机器人，数组的每个元素表示在当前位置机器人最大能够跳跃的数目。它的目的是用最少的步数到达数组末端。例如：给定数组A=[2,3,1,1,2]，最少跳步数目是2，对应的跳法是：2→3→2。
- 如：2,3,1,1,2,4,1,1,6,1,7，最少需要几步？

跳跃问题分析



跳跃问题算法步骤



- 初始步数step赋值为0；
- 记当前步的控制范围是[i,j]，则用k遍历i到j
  - 计算A[k]+k的最大值，记做j2；
- step++；继续遍历[j+1,j2]；

Code

```
int Jump(int A[], int n)
{
    if (n == 1)
        return 0;

    int step = 0; //最小步数
    int i = 0;
    int j = 0;  //[i,j]是当前能覆盖的区间
    int k,j2;
    while(j < n)  //覆盖区间尚未包含最后元素
    {
        step++;
        j2 = j;
        for(k = i; k <= j; k++)
        {
            j2 = max(j2, k + A[k]);
            if(j2 >= n-1)   //已经跳跃到最后一步
                return step;
        }
        i = j+1;
        j = j2;
        if(j < i)   //覆盖区间为负，说明无法跳到末尾
            return -1;
    }
    return step;
}

int _tmain(int argc, _TCHAR* argv[])
{
    int A[] = {2,3,1,1,2,4,1,1,6,1,7};
    Jump(A, sizeof(A) / sizeof(int));
    return 0;
}
```

Jump问题“知识挖掘”

- 上述代码的时间复杂度是多少？
  - O(N) or O(N^2)
- 该算法能够天然处理无法跳跃到末尾的情况。
  - 若无法跳到莫问，则返回-1
- 该算法在每次跳跃中，都是尽量跳的更远，并记录j2——属于贪心法；也可以认为是从区间[i,j](若干结点)扩展下一层区间[j+1,j2] (若干子结点)——属于广度优先搜索。
  - 可见，贪心法是需要详细分析才能放心使用。
  - 回忆图论中的概要说明：
    - 广度优先搜索往往和“最少”、“最短”相关联。
- 思考：是否可以使用动态规划解决？
  - 记dp[i]为：到达A[i]时，还剩余多少步没有用。
  - 则：dp[i+1]=max(dp[i],A[i])-1

计算概率

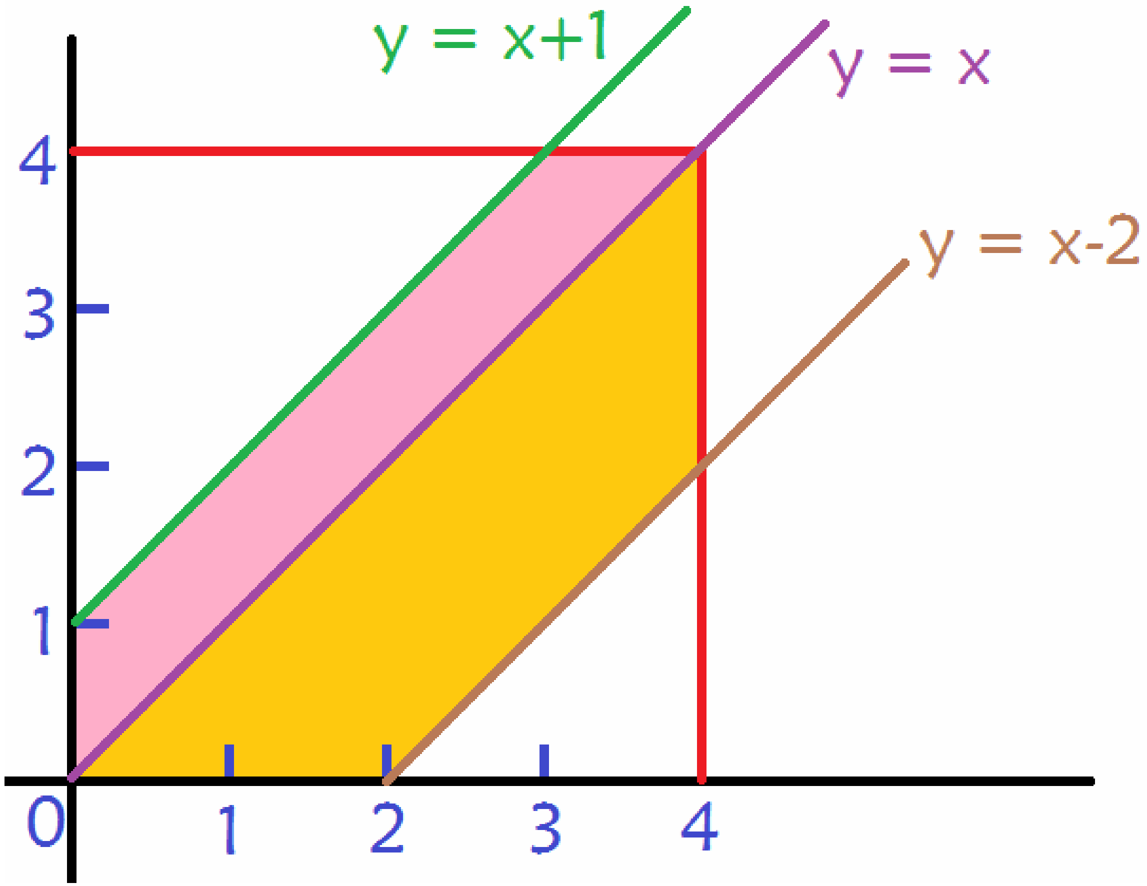
- A、B两国元首相约在首都机场晚20点至24点交换一份重要文件。如果A国的飞机先到，A会等待1个小时；如果B国的飞机先到了，B会等待2个小时。假设两架飞机在20点至24点降落机场的概率是均匀分布，试计算能够在20点至24点完成交换的概率。
  - 假设交换文件本身不需要时间。

事件的形式化表达

- 假定A到达的时刻为x，B达到的时刻为y，则完成交接需满足$0<y-x<1$或者$0<x-y<2$；
- 同时要求$24<x<20$，$24<y<20$；
- 由于x，y系数都为1，为作图方便，可以将$24<x<20$，$24<y<20$平移成$4<x<0$，$4<y<0$。

计算面积

- 三角形面积
  - 9/2、2
- 矩形面积
  - 16
- 概率
  - 19/32

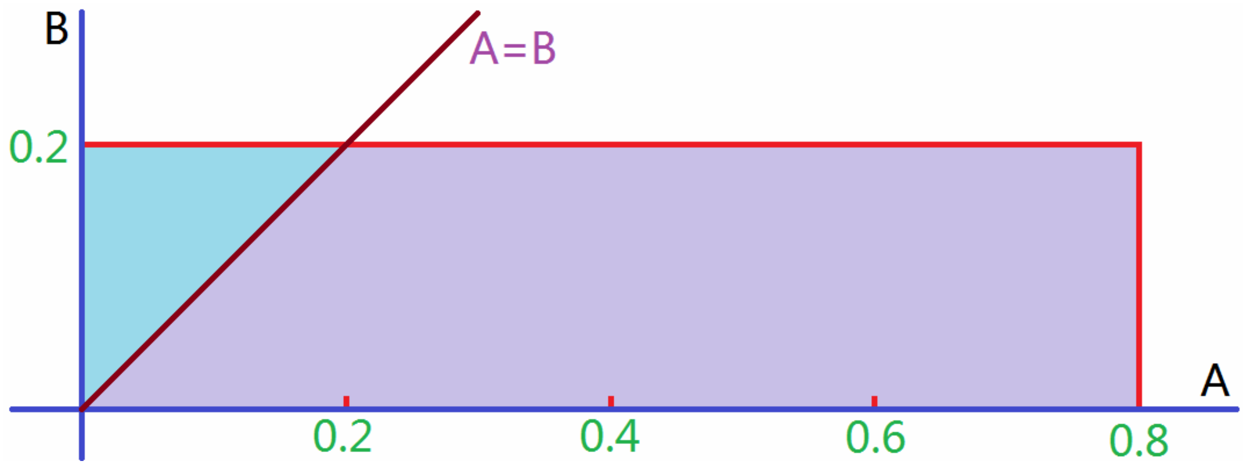


商品推荐

- 商品推荐场景中过于聚焦的商品推荐往往会损害用户的购物体验，在有些场景中，系统会通过一定程度的随机性给用户带来发现的惊喜感。
- 假设在某推荐场景中，经计算A和B两个商品与当前访问用户的匹配度分别为0.8分和0.2分，系统将随机为A生成一个均匀分布于0到0.8的最终得分，为B生成一个均匀分布于0到0.2的最终得分，试计算最终B的分数大于A的分数的概率。

商品推荐

- A=B的直线上方区域，即为B>A的情况。
- $S_{蓝色}=0.02$　　　$S_{矩形}=0.16$
- $p=0.02/0.16=0.125$

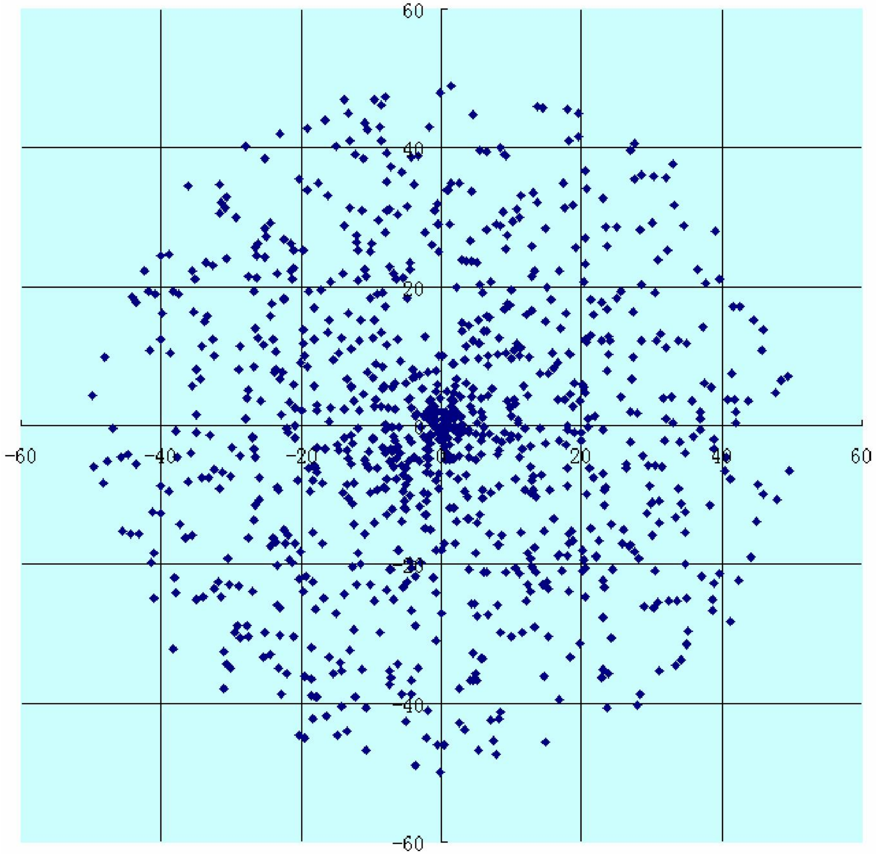


圆内均匀取点

- 给定定点$O(x_0,y_0)$和半径r，使得二维随机点(x,y)等概率落在圆内。
- 分析
  - 因为均匀分布的数据是具有平移不变性，生成半径为r，定点为圆心的随机数(x_1,y_1)，然后平移得到(x_1+x_0,y_1+y_0)即可。
  - 直接使用$x=r*\cos\theta$，$y=r*\sin\theta$是否可以呢？
    - 具体试验一下。

圆内均匀取点代码与效果

```
int rand50()
{
    return rand() % 100 - 50;
}

int _tmain(int argc, _TCHAR* argv[])
{
    ofstream oFile;
    oFile.open(_T("D:\\rand.txt"));
    double r,theta;
    double x,y;
    for(int i = 0; i < 1000; i++)
    {
        r = rand50();
        theta = rand();
        x = r*cos(theta);
        y = r*sin(theta);
        oFile << x << '\t' << y << '\n';
    }
    oFile.close();
    return 0;
}
```

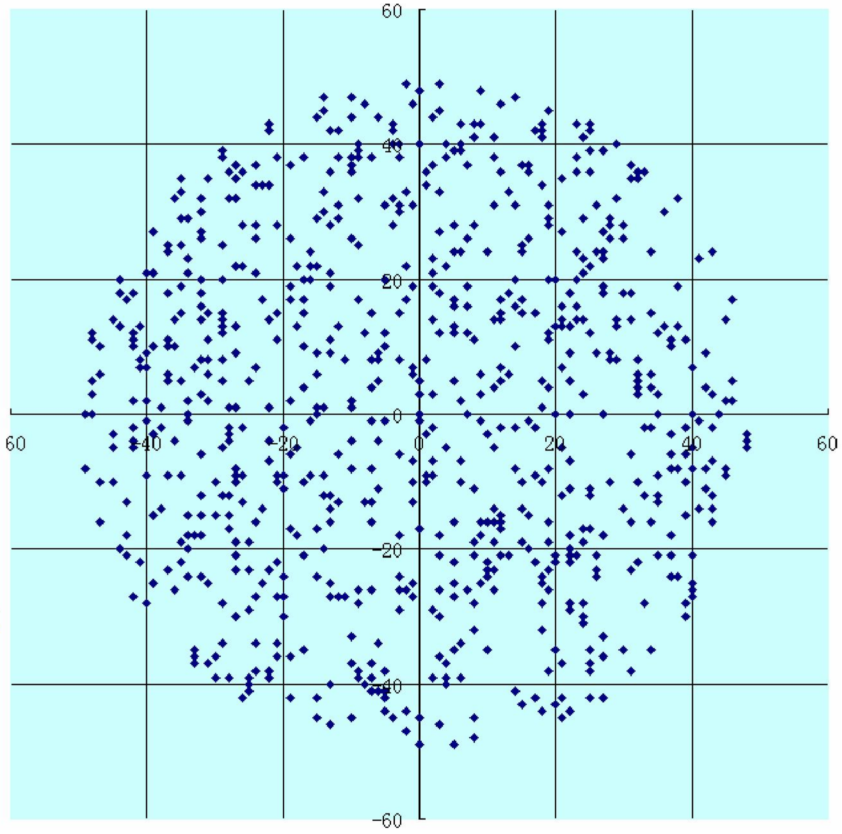


代码与效果

- 显然上述做法是不对的。但可以使用二维随机点的做法，若落在圆外，则重新生成点。结果如下。

代码与效果

```
int rand50()
{
    return rand() % 100 - 50;
}

int _tmain(int argc, _TCHAR* argv[])
{
    ofstream oFile;
    oFile.open(_T("D:\\rand.txt"));
    int x,y;
    for(int i = 0; i < 1000; i++)
    {
        x = rand50();
        y = rand50();
        if(x*x + y*y < 2500)
            oFile << x << '\t' << y << '\n';
    }
    oFile.close();
    return 0;
}
```



思想分析

- 不是每次生成随机数都能退出该算法
  - 有一定的接受率。
  - 思考：
    - 以多大的概率1次退出：接受率是多少？
    - 得到随机数的需要的平均次数(期望)是多少？
    - 利用该方法计算圆周率？
- 这个做法简洁、有效，值得推荐；
  - 许多相关问题，往往可以如此解决。

一定接受率下的采样

- 已知有个rand7()的函数，返回1到7随机自然数，让利用这个rand7()构造rand10() 随机1~10。
- 解：因为rand7仅能返回1~7的数字，少于rand10的数目。因此，多调用一次，从而得到49种组合。超过10的整数倍部分，直接丢弃。

Code

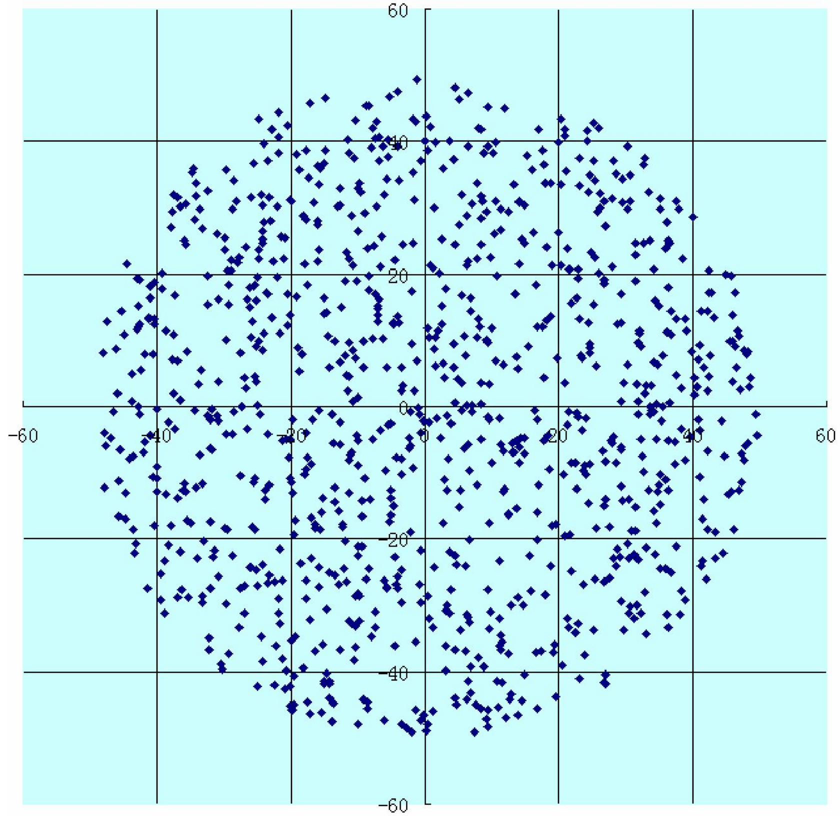
```
int rand10()
{
    int a1, a2, r;
    do
    {
        a1 = rand7() - 1;
        a2 = rand7() - 1;
        r = a1 * 7 + a2;
    } while (r >= 40);
    return r / 4 + 1;
}
```

圆内均匀取点的1次成功算法(朴素)

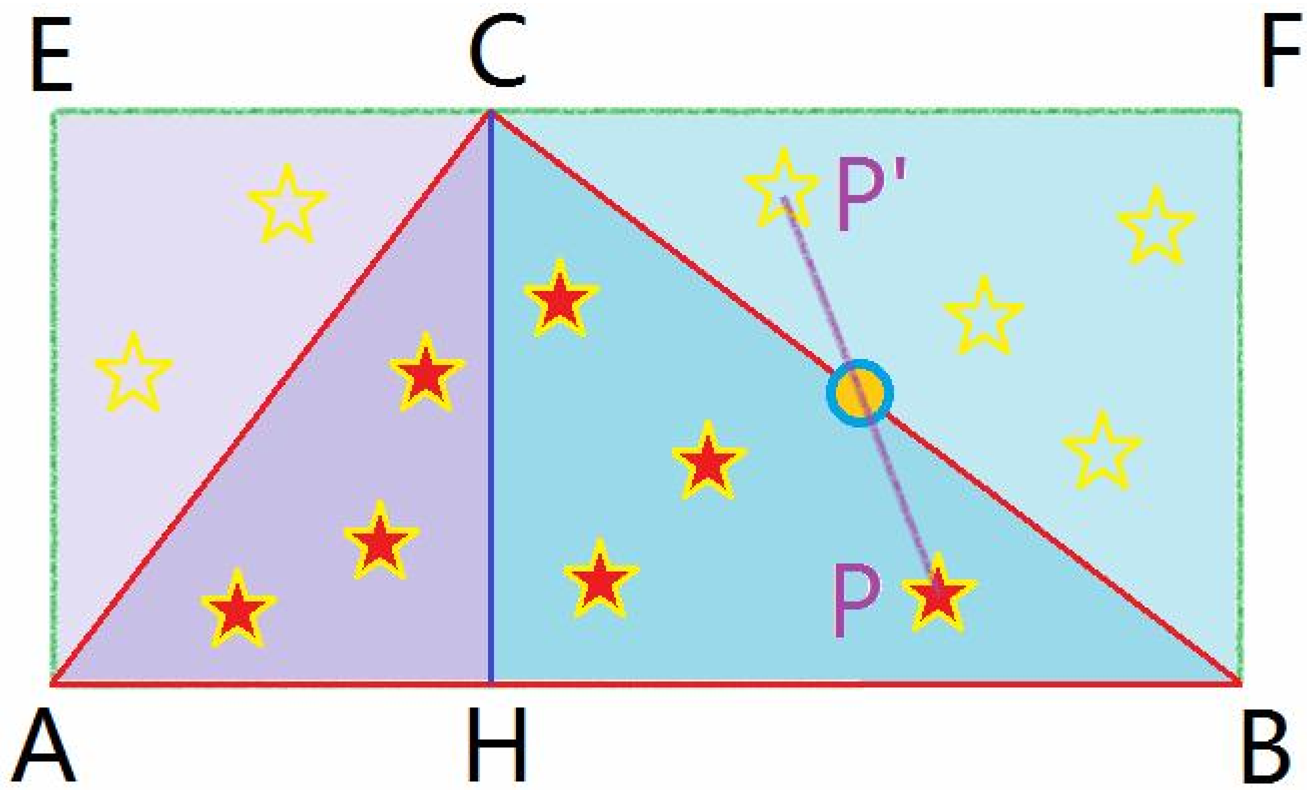
- 问题分析：把随机点看做面积很小的区域，圆内均匀取点意味着随机点P的面积与圆的面积成正比。
- $S_p=kS$
- $S=\pi r^2$，与半径的平方成正比
- 从而，$S_p(r)=k\pi r^2$
- 将均匀生成的随机数x取平方根赋值给r；则$S_p(r)$即为均匀分布。
- 同时，是与角度θ无关的，即：取均匀分布的随机数θ作为旋转角即可。

代码与效果

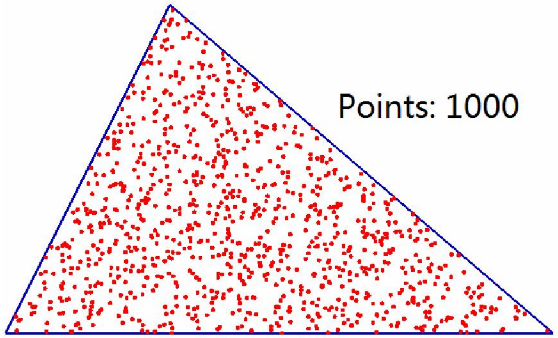
```
double rand2500()
{
    return rand() % 2500;
}

int _tmain(int argc, _TCHAR* argv[])
{
    ofstream oFile;
    oFile.open(_T("D:\\rand.txt"));
    double r,theta;
    double x,y;
    for(int i = 0; i < 1000; i++)
    {
        r = sqrt(rand2500());
        theta = rand();
        x = r*cos(theta);
        y = r*sin(theta);
        oFile << x << '\t' << y << '\n';
    }
    oFile.close();
    return 0;
}
```

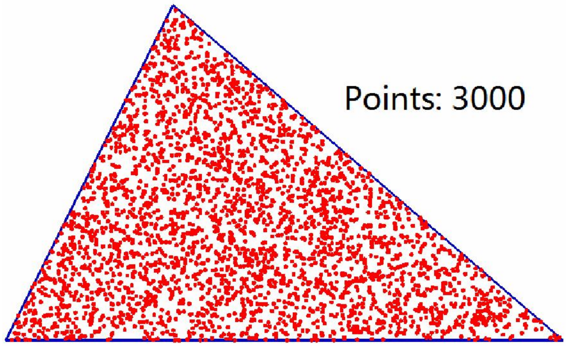


思考：将圆域换成三角形呢？

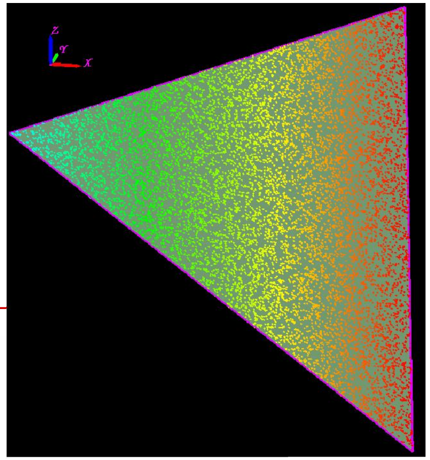


代码与效果






```
void CRandomTriangle::Random2(int nSize)
{
    CalcRotate();
    m_nSize = nSize;
    if(m_pRandomPoint)
        delete[] m_pRandomPoint;
    m_pRandomPoint = new CDelPoint[nSize];
    CDelPoint pt;
    for(int i = 0; i < nSize; i++)
    {
        pt.RandomInRectangle(m_ptExtend, m_ptHeight);
        if(m_tsBig.IsIn(pt))
        {
            pt += m_ptBase;
            m_pRandomPoint[i] = pt;
        }
        else if(m_tsLeft.IsIn(pt))
        {
            CDelPoint::MirrorPoint(pt, m_ptLeft0);
            pt += m_ptBase;
            m_pRandomPoint[i] = pt;
        }
        else if(m_tsRight.IsIn(pt))
        {
            CDelPoint::MirrorPoint(pt, m_ptRight0);
            pt += m_ptBase;
            m_pRandomPoint[i] = pt;
        }
    }
    CDelPoint::Save(m_pRandomPoint, m_nSize, _T("D:\\random.pt"), 0);
}
```



“概率章节”的进一步思考

- 由于三点共面，所以三角形内的所有点必然在某平面上，因此，上述算法能够方便的推广到三维空间。
- 问题：请设计多边形内随机取点算法。
  - 圆内取点的思想：计算多边形的外包围矩形盒，生成外包围盒内的二维点，若点在多边形内，则退出；否则，继续探测。
  - 将多边形剖分成三角形集合，调用三角形内均匀取点算法。
- 算法2思路：
  - 按照面积为权重，选择某个三角形；
  - 生成该三角形内的随机点。
- 拓展
  - 每首歌有不同的分值，设计算法，根据分值随机推荐歌曲。
  - 如何将多边形快速剖分成三角形？注：Delaunay三角剖分

分值推荐

- 假定歌曲库中N首歌，每首歌给定一个整数分数。现在要求从N首歌中随机选择若干首推荐给用户，要求推荐的这些歌是和其分数作为正比的。
- 给定整数数组A[0…N-1]，按照A[i]的值作为权值随机取数。

分析

- 根据权值A[0…N-1]计算累积权值B[0…N-1]
  - $B_0=A_0$
  - $B_i=B_{i-1}+A_i$
- 区间$[B_{i-1}, B_i)$对应元素A_i

Code

```
int RandSong(const int* song, int size)
{
    int i;
    int* pCumulate = new int[size];//i的范围：[i-1,i)
    pCumulate[0] = song[0];
    for(i = 1; i < size; i++)
        pCumulate[i] = pCumulate[i-1] + song[i];

    int nRec = rand() % pCumulate[size-1];
    int low = 0;
    int high = size-1;
    int mid;
    int nSong = -1;
    while(low < high)
    {
        mid = (low + high) / 2;
        if(nRec < pCumulate[mid])
            high = mid;
        else if(nRec > pCumulate[mid])
            low = mid + 1;
        else
        {
            nSong = mid+1;
            break;
        }
    }
    if(nSong == -1)
        nSong = low;
    delete[] pCumulate;
    return nSong;
}
```

另外的思路

- 计算所有歌的权值和sum，每首歌的权值除以sum，认为是各自的概率P[0…N-1]；
- 等概率选择nSong∈[0,N-1)
- 生成p∈[0,1]，若p＜P[nSong]，则选择nSong，否则，计算随机生成新的nSong，继续探测。

Code

```
int RandSong2(const int* song, int size)
{
    int nSum = song[0];
    for(int i = 1; i < size; i++)
        nSum += song[i];

    bool bFind = false;
    int nCandidate = 0;
    while(!bFind)
    {
        nCandidate = rand() % size; //候选
        if(rand() % nSum < song[nCandidate])
        {
            bFind = true;
            break;
        }
    }
    return nCandidate;
}
```

试验结果

- 初始值： song = {43,63,43,89,322,2,5,32}
- 真实概率：
  - 0.0718 0.105 0.0718 0.149 0.538 0.00334 0.00835 0.0534
- 算法1
  - 10^3： 0.073 0.096 0.077 0.152 0.536 0.004 0.013 0.049
  - 10^4： 0.0758 0.107 0.0723 0.143 0.54 0.0028 0.0087 0.0505
- 算法2
  - 10^3： 0.076 0.095 0.084 0.137 0.539 0.003 0.008 0.058
  - 10^4： 0.0719 0.101 0.0726 0.144 0.542 0.0025 0.0093 0.0567

思考

- 给定N个数，设计算法，输出随机排列的一个结果。
  - 要求：输出任何一个排列的概率是相同的。
  - STL std::random_shuffle

Code

```
int Random(int N)    //[0,N]
{
    return rand() % (N+1);
}

void RandomShuffle(int* a, int size)
{
    int j;
    for(int i = 1; i < size; i++)    //待生成倒数第i个数
    {
        j = Random(size-i); //[0,size-i]
        swap(a[j],a[size-i]);
    }
}

int _tmain(int argc, _TCHAR* argv[])
{
    const int N = 10;
    int a[N];
    for(int i = 0; i < N; i++)
        a[i] = i+1;
    RandomShuffle(a, N);
    Print(a, N);
    return 0;
}
```

Code2

```
int Random2(int a, int b)
{
    return rand() % (b-a+1) + a;
}

void RandomShuffle2(int* a, int size)
{
    int j;
    for(int i = 0; i < size-1; i++) //i为待生成的第几个数
    {
        j = Random2(i, size-1); //[i,size-1]
        swap(a[i],a[j]);
    }
}
```

金钗赠诗问题

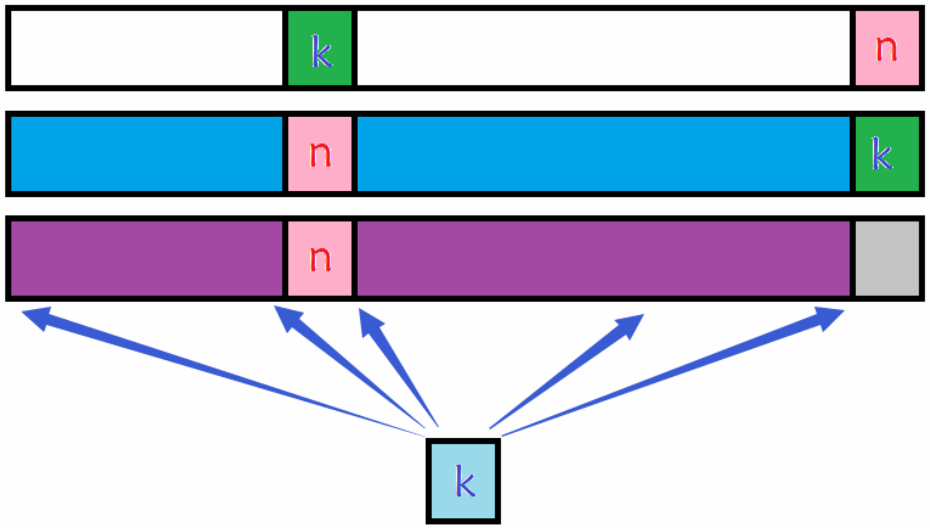
- 赛诗会后，十二金钗待奔前程。分别宴上，12人各写一首诗放入宝囊。大家任取，若取到自己的诗，则再取一首并放回自己的诗。12人都拿到别人的诗作算一种分配。问：共有多少种不同的分配？

问题的由来

- 本质即给定n个人写n首诗，要求赠给其他人，共有多少种分配方法。
- 一般提法：1到n的全排列中，第i个数不是i的排列共有多少种？
  - 最早是由丹尼尔·伯努利(Danid Bernoulli)提出的“错位排列”问题。

问题分析

- 假定n个数的错位排列数目为dp[n]
- 先考察数字n的放置方法：显然，n可以放在从1到n-1的某个位置，共n-1种方法；假定放在了第k位。
- 对于数字k：
  - 要么放置在第n位，
  - 要么不放置在第n位。
  - 下面分别讨论

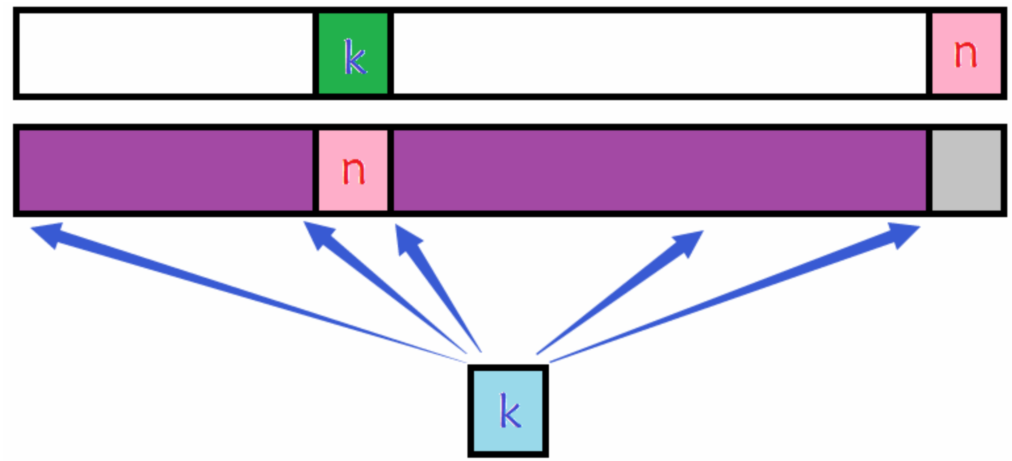


数字k放置在第n位

- 相当于数字k和数字n交互位置后，其他n-2个数字做错位排列，因此有dp[n-2]种方法。

数字k不放置在第n位

- 将数字k暂时更名为n(这是可以做到的：因为真正的n已经放在第k位上，真正的n不再考虑之列)，现在需要将1到k-1以及k+1到n这n-1个数放置在相应位置上，要求数字和位置不相同！显然是n-1个数的错位排列，有dp[n-1]种方法。



错位排列递推公式

- 综上，dp[n]=(n-1)*(dp[n-1]+dp[n-2])
- 初值：
  - 只有1个数字，错位排列不存在，dp[1]=0；
  - 只有2个数字，错位排列即交换排列，dp[2]=1；

$$a(n)=\begin{cases}(n-1)\cdot\left[a(n-1)+a(n-2)\right] & n>2\\ 1 & n=2\\ 0 & n=1\end{cases}$$

错位排列的通项公式

- 使用基本的加法和乘法原理，能够得到错位公式的**通项形式**：

$$S(n)=P_n^n-C_n^1P_{n-1}^{n-1}+-C_n^2P_{n-2}^{n-2}+\cdots+(-1)^n-C_n^nP_0^0$$

- 或等价形式：

$$S(n)=n!\left(1-\frac{1}{1!}+\frac{1}{2!}-\frac{1}{3!}+\frac{1}{4!}+\cdots+(-1)^n\frac{1}{n!}\right)$$

思考题

- 毕业典礼后，某宿舍三位同学把自己的毕业帽扔了，随后每个人随机地拾起帽子，试计算三个人中没有人选到自己原来带的帽子的概率。

思考题

- 篮球比赛中，两队比分暂时相等。最后一次各有两次罚球机会中；已知在一次罚球中，甲队罚球人员投中的概率是0.5，乙队罚球人员投中的概率是0.6，试计算甲乙战平的概率。

我们在这里

7 七月算法 http://www.julyedu.com/

- 视频/课程/社区

- 七月题库APP：Android/iOS
  - http://www.julyapp.com/
- 微博
  - @研究者July
  - @七月题库
  - @邹博_机器学习
- 微信公众号
  - julyedu

感谢大家

恳请大家批评指正！