

# 法律声明

□ 本课件包括演示文稿、示例、代码、题库、视频和声音等内容，小象学院和主讲老师拥有完全知识产权的权利；只限于善意学习者在本课程使用，不得在课程范围外向任何第三方散播。任何其他人或机构不得盗版、复制、仿造其中的创意及内容，我们保留一切通过法律手段追究违反者的权利。

□ 课程详情请咨询

■ 微信公众号：小象

■ 新浪微博：ChinaHadoop



# 算法总结

---



小象学院  
ChinaHadoop.cn

邹博

# 主要内容

---

- ☐ Callatz猜想问题
- ☐ 海明距离
- ☐ Eratosthenes筛法求素数
- ☐ 循环染色方案
- ☐ 马踏棋盘
- ☐ 蚁群算法
- ☐ 三字母字符串组合
- ☐ 热点话题讨论: AI/ML/DL

# Callatz猜想问题

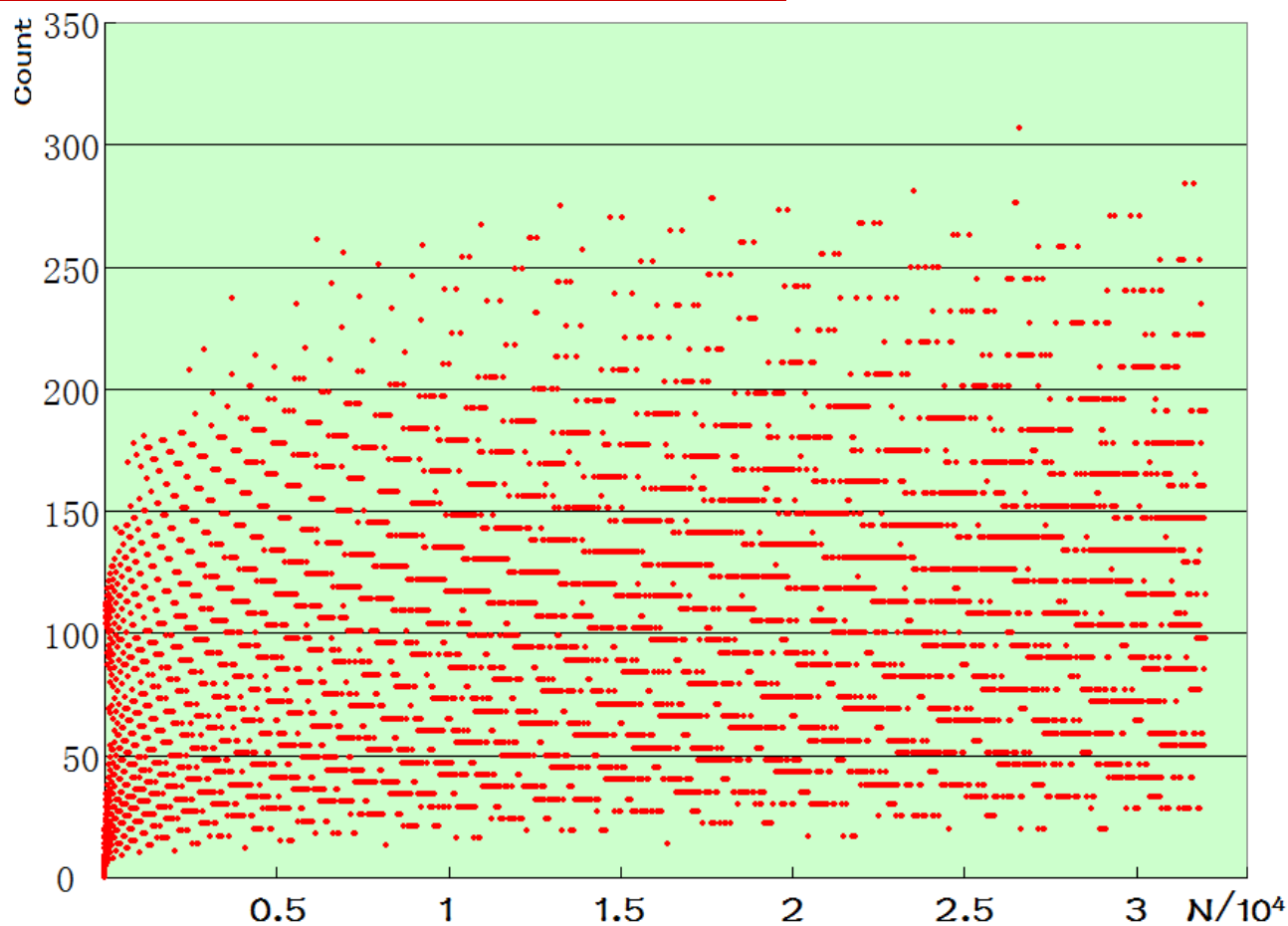
- 该问题又称 $3n+1$ 猜想、角谷猜想、哈塞猜想、乌拉姆猜想、叙拉古猜想等。
- 过程非常简单：给定某正整数 $N$ ，若为偶数，则 $N$ 被更新为 $N/2$ ；否则， $N$ 被更新为 $3*N+1$ ；问：(1)经过多少步 $N$ 变成1？(2)是否存在某整数 $X$ 无法变成1？
- 思考：
  - 如果已经计算得到 $1\sim N-1$ 的变换次数，如何计算 $N$ 的变换次数？

# Code

```
void Calc(long long i, int* p, int N, int timeStart)
{
    int cur = (int) i;
    int t = 0;
    while(true)
    {
        if(i % 2 == 0)
        {
            i /= 2;
            t++;
        }
        else
        {
            i = i * 3 + 1;
            t++;
        }
        if((i < N) && (p[(int) i] != -1))    //已经有部分值
        {
            p[cur] = p[(int) i] + t;
            break;
        }
    }

    if(cur % 10000 == 0)    //顺便记录时间
    {
        tt.push_back(GetTickCount() - timeStart);
    }
}
```

# N-count图像



# Code运行时间



# 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。

# Eratosthenes筛法求素数

---

- 给定正整数 $N$ ，求小于等于 $N$ 的全部素数。
- Eratosthenes筛法
  - 将2到 $N$ 写成一排；
  - 记排头元素为 $x$ ，则 $x$ 是素数；除 $x$ 以外，将 $x$ 的倍数全部划去；
  - 重复以上操作，直到没有元素被划去，则剩余的即小于等于 $N$ 的全部素数。
  - 为表述方便，将排头元素称为“筛数”。

# Eratosthenes筛计算100以内的素数

---

□ 2 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31  
33 35 37 39 41 43 45 47 49 51 53 55 57 59 61  
63 65 67 69 71 73 75 77 79 81 83 85 87 89 91  
93 95 97 99

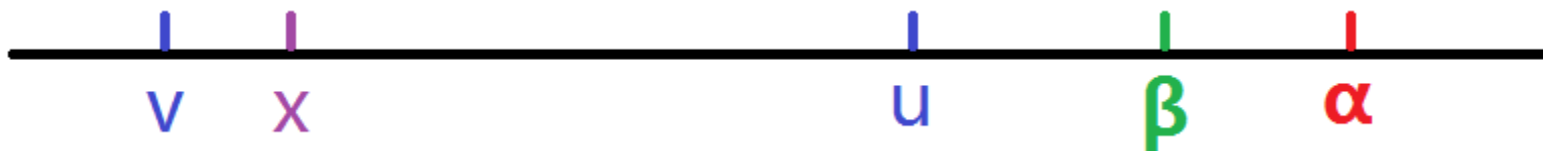
□ 2 3 5 7 11 13 17 19 23 25 29 31 35 37 41 43  
47 49 53 55 59 61 65 67 71 73 77 79 83 85 89  
91 95 97

□ 2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 49  
53 59 61 67 71 73 77 79 83 89 91 97

□ 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

# 算法改进：筛选 $\alpha$ 以内的素数

- $\alpha$ 以内素数的最大筛数为  $\sqrt{\alpha}$ ，记  $x = \sqrt{\alpha}$
- 对于  $\beta < \alpha$
- 若 $\beta$ 为合数，即： $\beta = v \cdot u$
- 显然， $u$ 、 $v$ 不能同时大于 $x$ ，不妨 $v < u$ ，将它们记录在数轴上：



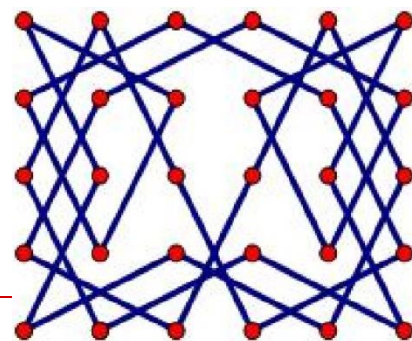
- 在使用 $x$ 作为筛数之前， $\beta$ 已经被 $v$ 筛掉。



# Code

```
void Eratosthenes(bool* a, int n)
{
    a[1] = false; //a[0]不用
    int i;
    for (i = 2; i <= n; i++) //筛法，默认是素数
        a[i] = true;
    int p = 2; //第一个筛孔
    int j = p*p;
    int c = 0;
    while (j <= n)
    {
        while (j <= n)
        {
            a[j] = false;
            j += p;
        }
        p++;
        while (!a[p]) //查找下一个素数
            p++;
        j = p*p;
    }
}
```

# 马踏棋盘




□ 给定 $m \times n$ 的棋盘，将棋子“马”放在任意位置上，按照走棋规则将“马”移动，要求每个方格只能进入一次，最终使得“马”走遍棋盘的所有位置。

■ 如给定 $8 \times 8$ 的国际象棋棋盘(右)

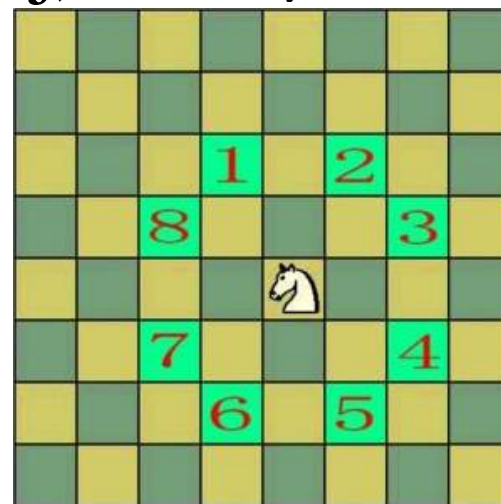
■ 如给定 $8 \times 9$ 的中国象棋棋盘(左)

01	70	17	36	03	52	07	38	05
18	35	02	71	60	37	04	45	08
69	16	67	58	51	56	53	06	39
34	19	72	61	54	59	46	09	44
15	68	33	66	57	50	55	40	29
20	65	22	49	62	47	28	43	10
23	14	63	32	25	12	41	30	27
64	21	24	13	48	31	26	11	42

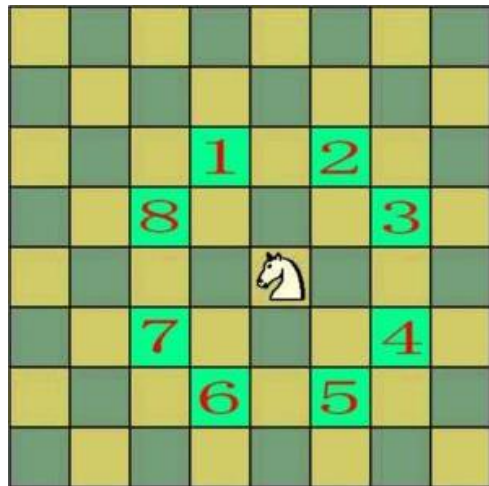
	23	48	35	10	21	56	33
49	36	11	22	55	34	9	20
12	1	24	47	40	57	32	63
25	50	37	54	43	46	19	8
2	13	44	41	58	39	62	31
51	26	53	38	45	42	7	18
14	3	28	59	16	5	30	61
27	52	15	4	29	60	17	6

# 问题分析

- 显然，如果从A点能够跳到B点，则从B点也能够跳到A点。所以，马的起始位置可以从任意一点开始，不妨从左上角开始。
- 若当前位置为 $(i,j)$ ，则遍历 $(i,j)$ 的八邻域，如果邻域尚未经过，则跳转。
  - 深度优先搜索



# Code



```
int iHorse[] = {-2, -2, -1, +1, +2, +2, +1, -1};
int jHorse[] = {-1, +1, +2, +2, +1, -1, -2, -2};
int m = 8;
int n = 9;
```

```
bool CanJump(const vector<vector<int> >& chess, int i, int j)
{
    if((i < 0) || (i >= m) || (j < 0) || (j >= n))
        return false;
    return (chess[i][j] == 0);
}

bool Jump(vector<vector<int> >& chess, int i, int j, int step)
{
    if(step == m*n) //遍历结束
        return true;
    int iCur, jCur;
    for(int k = 0; k < 8; k++)
    {
        iCur = i + iHorse[k];
        jCur = j + jHorse[k];
        if(CanJump(chess, iCur, jCur))
        {
            chess[iCur][jCur] = step+1;
            if(Jump(chess, iCur, jCur, step+1))
                return true;
            chess[iCur][jCur] = 0;
        }
    }
    return false;
}
```

```
int _tmain(int argc, _TCHAR* argv[])
{
    vector<vector<int> > chess(m, vector<int>(n));
    chess[0][0] = 1;
    Jump(chess, 0, 0, 1);
    Print(chess);
    return 0;
}
```

# 启发式搜索

- 若棋盘规模较大，则在较深的棋位才能发现“无路可走”而不得不回溯。
- 贪心的启发式策略：
  - 最多情况下，每个棋位有8个后继。由于棋盘边界和已经遍历的原因，往往是少于8个的。
- 当前棋位可以跳转的后继棋位记为X个，这X个棋位的后继棋位数记做 $h_1h_2...h_x$ ，优先选择最小的 $h_i$ 。
  - 策略：优先选择孙结点数目最少的那个子结点
  - 原因：孙结点最少的子结点，如果当前不跳转则最容易在后期无法跳转。

# Code

```
int _tmain(int argc, _TCHAR* argv[])
{
    vector<vector<int>> chess(m, vector<int>(n));
    chess[0][0] = 1;
    Jump2(chess, 0, 0, 1);
    Print(chess);
    return 0;
}
```

```
typedef struct tagSHorse
{
    int nDirect;    //跳转方向
    int nValidStep; //有效步数

    bool operator < (const tagSHorse& horse)
    {
        return nValidStep < horse.nValidStep;
    }
} SHorse;
```

```
int GatherHorseDirect(SHorse* pHorse, int i, int j, const vector<vector<int>>& chess, bool bLast)
{
    int nHorse = 0;
    int iCur, jCur;
    for(int k = 0; k < 8; k++)
    {
        iCur = i + iHorse[k];
        jCur = j + jHorse[k];
        if(bLast) //最后一步
        {
            if(CanJump(chess, iCur, jCur))
            {
                pHorse[nHorse].nValidStep = 1;
                pHorse[nHorse].nDirect = k;
                nHorse++;
                break;
            }
        }
        else //正常情况
        {
            pHorse[nHorse].nValidStep = GetNextStep(chess, iCur, jCur);
            if(pHorse[nHorse].nValidStep != 0)
            {
                pHorse[nHorse].nDirect = k;
                nHorse++;
            }
        }
    }
    if(nHorse == 0)
        return 0;
    sort(pHorse, pHorse+nHorse);
    return nHorse;
}

bool Jump2(vector<vector<int>>& chess, int i, int j, int step)
{
    if(step == m*n) //遍历结束
        return true;
    SHorse pHorse[8];
    int nHorse = GatherHorseDirect(pHorse, i, j, chess, step == m*n-1);
    int iCur, jCur;
    int nDirect;
    step++;
    for(int k = 0; k < nHorse; k++)
    {
        nDirect = pHorse[k].nDirect;
        iCur = i + iHorse[nDirect];
        jCur = j + jHorse[nDirect];
        chess[iCur][jCur] = step;
        if(Jump2(chess, iCur, jCur, step))
            return true;
        chess[iCur][jCur] = 0;
    }
    return false;
}
```

# Code

```
bool Jump2(vector<vector<int> >& chess, int i, int j, int step)
{
    if(step == m*n) //遍历结束
    {
        AddSolution(chess);
        return true;
    }
    SHorse pHorse[8];
    int nHorse = GatherHorseDirect(pHorse, i, j, chess, step == m*n-1);
    int iCur, jCur;
    int nDirect;
    step++;
    for(int k = 0; k < nHorse; k++)
    {
        nDirect = pHorse[k].nDirect;
        iCur = i + iHorse[nDirect];
        jCur = j + jHorse[nDirect];
        chess[iCur][jCur] = step;
        if(Jump2(chess, iCur, jCur, step)) //找到一个解
        {
            //return true; //删去本行, 则算法计算所有解
        }
        chess[iCur][jCur] = 0;
    }
    return false;
}
```

# Code

```
int GatherHorseDirect(SHorse* pHorse, int i, int j, const vector<vector<int>>& chess, bool bLast)
{
    int nHorse = 0;
    int iCur, jCur;
    for(int k = 0; k < 8; k++)
    {
        iCur = i + iHorse[k];
        jCur = j + jHorse[k];
        if(bLast)    //最后一步
        {
            if(CanJump(chess, iCur, jCur))
            {
                pHorse[nHorse].nValidStep = 1;
                pHorse[nHorse].nDirect = k;
                nHorse++;
                break;
            }
        }
        else    //正常情况
        {
            pHorse[nHorse].nValidStep = GetNextStep(chess, iCur, jCur);
            if(pHorse[nHorse].nValidStep != 0)
            {
                pHorse[nHorse].nDirect = k;
                nHorse++;
            }
        }
    }
    if(nHorse == 0)
        return 0;
    sort(pHorse, pHorse+nHorse);
    return nHorse;
}
```

32				
=====				
1	20	5	12	9
6	13	10	17	4
19	2	15	8	11
14	7	18	3	16
=====				
1	18	5	12	9
6	13	10	17	4
19	2	15	8	11
14	7	20	3	16
=====				
1	16	5	12	9
6	13	10	19	4
17	2	15	8	11
14	7	18	3	20
=====				
1	16	5	12	9
6	13	10	17	4
15	2	19	8	11
20	7	14	3	18
=====				
1	16	5	12	9
6	19	10	15	4
17	2	13	8	11
20	7	18	3	14
=====				



# 蚁群算法

---

- 蚁群优化算法1991年由Dorigo提出并应用于TSP，已经发展了20多年，具有鲁棒性强、全局搜索、并行分布式计算、易与其他问题结合等优点。
  - 使用传统算法难以求解或无法求解的问题，可以尝试蚁群算法及其改进版本。
- 基本思想：蚂蚁在爬行中会在路径中释放外激素，也能感知路径中已有的外激素：蚂蚁倾向于朝外激素强度高的方向移动。
  - 信息正反馈现象：某路径经过的蚂蚁越多，则后来者选择该路径的概率就越大。

# 蚁群算法的步骤

---

- 核心：路径中的信息素以一定的比例挥发减少，而某蚂蚁经过的路径，信息素以一定的比例释放增加。
- 算法过程： $m$ 只蚂蚁，最大迭代次数为 $K$ 
  - 信息素的初始化
  - 路径构建
  - 信息素更新
  - 2、3步迭代 $K$ 次或最短路径不再变化

# 信息素的初始化

□ 如果初值太小，算法容易早熟，蚂蚁会很快全部集中到一条局部最优路径中。反之，如果初值太大，信息素对搜索方向的指导作用太低，影响算法性能。

□ 一般可以如下初始化：

$$\tau_{ij} = \frac{m}{dist}, \forall i, j$$

■ 其中，m是蚂蚁的数目，dist是路径的估计值

# 路径构建

□ 每只蚂蚁随机选择一城市作为出发点，维护该蚂蚁经过城市的列表(即路径)。在构建路径的每一步中，依概率选择下一个城市。

□ 从第*i*个城市到第*j*个城市的转移概率：

$$P_{ij} \propto \frac{\tau_{ij}^{\alpha}}{d_{ij}^{\beta}}, j \in \{neighbors\ of\ i\}$$

□ 其中， $d_{ij}$  为城市*i*和城市*j*之间的距离，

□  $\alpha, \beta$  为权值调节因子。

# 信息素更新

- 为了模拟蚂蚁在较短的路径留下较多的信息素，当所有蚂蚁到达终点时，更新各路径的信息素浓度。
- 更新公式：
$$\tau_{ij} = (1 - \rho)\tau_{ij} + \sum_{k=1}^m v_{ij}^{(k)}$$
- 其中， $\rho \in (0, 1]$  为信息素的挥发率。
- $v_{ij}^{(k)}$  为第k只蚂蚁在城市i到城市j的边释放的信息素，该值往往取该蚂蚁经过的整个路径的长度倒数。

# Code

```
double AS(const vector<CCity>& pCity, const vector<vector<double>>& ppDistance, vector<int>& bestRoute)
{
    int nCity = (int)pCity.size();
    int m = nCity/2;    //蚂蚁数目
    double t = m/(nCity*GetAvgDistance(ppDistance));    //假定C是某一条路径长度，则t初始化为m/C
    int times = 50;    //进行50轮迭代
    double p = 0.5;    //挥发率
    int i, j;

    //初始化任意两点间的信息素
    vector<vector<double>> pheromone(nCity, vector<double>(nCity));
    for(i = 0; i < nCity; i++)
    {
        for(j = 0; j < nCity; j++)
            pheromone[i][j] = t;
        pheromone[i][i] = 0;
    }

    //AS
    int k;
    vector<vector<int>> r(m, vector<int>(nCity+1));    //蚂蚁按照信息素随机得到的一条路径
    vector<double> rLen(m);    //rLen[i]: r[i]的长度
    double bestRouteLen = -1;    //最优路径的长度
    int best;    //最优路径是哪只蚂蚁得到的
    for(t = 0; t < times; t++)    //迭代若干次
    {
        best = -1;
        for(k = 0; k < m; k++)    //每一只蚂蚁
        {
            RandomRoute(ppDistance, pheromone, r[k]);    //计算第k只蚂蚁的路径
            rLen[k] = CalcLength(r[k], ppDistance);
            if((bestRouteLen < 0) || (bestRouteLen > rLen[k]))
            {
                bestRouteLen = rLen[k];
                best = k;
            }
        }
        if(best != -1)
            bestRoute = r[best];    //当前的最好路径

        //挥发
        Volatilize(pheromone, p);

        //遗留
        for(k = 0; k < m; k++)    //每一只蚂蚁
            AddPheromone(pheromone, r[k], 1/rLen[k]);
    }
    return bestRouteLen;
}
```

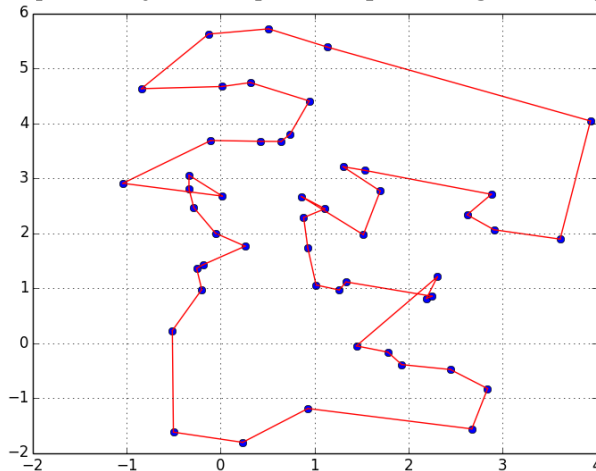
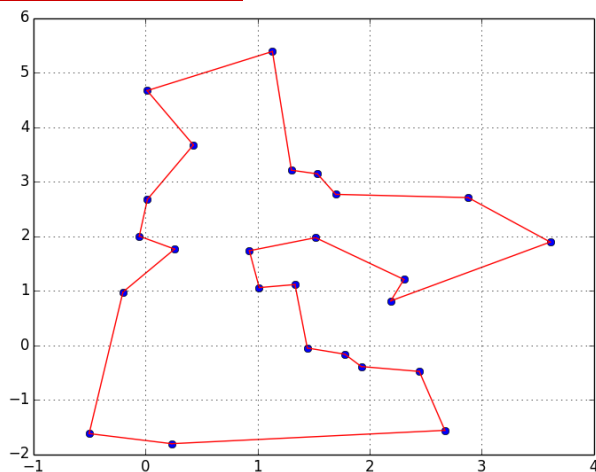
# Code

```
void RandomRoute(const vector<vector<double> >& ppDistance, const vector<vector<double> >& pheromone, vector<int>& r)
{
    int nCity = (int)ppDistance.size(); //城市数目
    r[0] = Rand(nCity); //随机挑选初始城市, [0,nCity)
    for(int i = 1; i < nCity; i++)
    {
        r[i] = Select(r, i, ppDistance[r[i-1]], pheromone[r[i-1]]);
    }
    r[nCity] = r[0];
}
```

```
double CalcLength(const vector<int>& r, const vector<vector<double> >& ppDistance)
{
    double s = 0;
    for(int k = 1; k < (int)r.size(); k++)
        s += ppDistance[r[k-1]][r[k]];
    return s;
}
```

```
void Volatilize(vector<vector<double> >& pheromone, double p)
{
    p = 1-p; //输入的p为挥发因子, 所以, 1-p即剩余因子
    int size = (int)pheromone.size();
    int i, j;
    for(i = 0; i < size; i++)
        for(j = 0; j < size; j++)
            pheromone[i][j] *= p;
}
```

```
void AddPheromone(vector<vector<double> >& pheromone, vector<int>& r, double a)
{
    for(int k = 1; k < (int)r.size(); k++)
        pheromone[r[k-1]][r[k]] += a;
}
```

[illegible]



# 三字母字符串组合

- 仅由三个字符A、B、C构成字符串，且字符串任意三个相邻元素不能完全相同。如“ACCCAB”不合法，“ABBCBCA”合法。求满足条件的长度为n的字符串个数。
- 假定不考虑整数溢出
- 要求时间和空间复杂度不高于 $O(N)$ 。

# 问题分析

---

- 若当前已经有了所有长度为 $n-1$ 的合法字符串，如何**在末端增加一个字符**，形成长度为 $n$ 的字符串呢？
- 将长度为 $n-1$ 字符串分成“末尾两个字符**不相等**”和“末尾两个字符**相等**”**两种情况**，各自数目记做 $dp[n-1][0]$ ,  $dp[n-1][1]$ ：

# dp[n][0]结尾不相等 / dp[n][1]结尾相等

□ ♂ ♀ ◎

□ ××.....× ♂ ♀ / ××.....× ♂ ♂

□  $dp[n][0] = 2 * dp[n-1][0] + 2 * dp[n-1][1]$

■ ××.....× ♂ ♀ ♂ , ××.....× ♂ ♀ ◎

■ ××.....× ♂ ♂ ♀ , ××.....× ♂ ♂ ◎

□  $dp[n][1] = dp[n-1][0]$

■ ××.....× ♂ ♀ ♀

□ 初始条件

■  $dp[1][0] = 3$

■  $dp[1][1] = 0$

# 状态转移方程总结与改进

## □ 状态转移方程：

$$\begin{cases} dp[n][0] = 2 * dp[n-1][0] + 2 * dp[n-1][1] \\ dp[n][1] = dp[n-1][0] \end{cases}$$

## □ 滚动数组：

$$\begin{cases} dp[0] = 2 * dp[0] + 2 * dp[1] \\ dp[1] = dp[0] \end{cases}$$

■ 使用滚动数组，将空间复杂度由O(N)降到O(1)

# Code

$$\begin{cases} dp[0] = 2 * dp[0] + 2 * dp[1] \\ dp[1] = dp[0] \end{cases}$$

```
int CalcCount(int n)
{
    int nNonRepeat = 3;
    int nRepeat = 0;
    int t;
    for(int i = 2; i <= n; i++)
    {
        t = nNonRepeat;
        nNonRepeat = 2*(nNonRepeat + nRepeat);
        nRepeat = t;
    }
    return nRepeat + nNonRepeat;
}
```

# 矩阵表示与O(logN)时间复杂度

□ 由状态转移方程：
$$\begin{cases} dp[0] = 2 * dp[0] + 2 * dp[1] \\ dp[1] = dp[0] \end{cases}$$

□ 得矩阵形式：
$$(dp[0] \quad dp[1])_{new} = (dp[0] \quad dp[1])_{old} \cdot \begin{bmatrix} 2 & 1 \\ 2 & 0 \end{bmatrix}$$

□ 从而：

$$\begin{aligned} (dp[0] \quad dp[1])_n &= (dp[0] \quad dp[1])_{n-1} \cdot \begin{bmatrix} 2 & 1 \\ 2 & 0 \end{bmatrix} \\ &= (dp[0] \quad dp[1])_{n-2} \cdot \begin{bmatrix} 2 & 1 \\ 2 & 0 \end{bmatrix} \cdot \begin{bmatrix} 2 & 1 \\ 2 & 0 \end{bmatrix} = (dp[0] \quad dp[1])_{n-3} \cdot \begin{bmatrix} 2 & 1 \\ 2 & 0 \end{bmatrix} \cdot \begin{bmatrix} 2 & 1 \\ 2 & 0 \end{bmatrix} \cdot \begin{bmatrix} 2 & 1 \\ 2 & 0 \end{bmatrix} \\ &= \dots \\ &= (dp[0] \quad dp[1])_1 \cdot \begin{bmatrix} 2 & 1 \\ 2 & 0 \end{bmatrix}^{n-1} = (3 \quad 0) \cdot \begin{bmatrix} 2 & 1 \\ 2 & 0 \end{bmatrix}^{n-1} \end{aligned}$$

# Code2

```
typedef struct tagSMatrix22
```

```
{  
    //第一列
```

```
    int a;
```

```
    int b;
```

```
    //第二列
```

```
    int c;
```

```
    int d;
```

```
    tagSMatrix22(int _a, int _b, int _c, int _d)  
        :a(_a), b(_b), c(_c), d(_d) {}
```

```
    void Set(int _a, int _b, int _c, int _d)  
    {
```

```
        a = _a;
```

```
        b = _b;
```

```
        c = _c;
```

```
        d = _d;
```

```
    }
```

```
} SMatrix22;
```

```
void MatrixMulti(SMatrix22& m, SMatrix22& n)//m *= n  
{  
    int a = m.a * n.a + m.c * n.b;  
    int b = m.b * n.a + m.d * n.b;  
    int c = m.a * n.c + m.c * n.d;  
    int d = m.b * n.c + m.d * n.d;  
    m.Set(a, b, c, d);  
}
```

```
void MatrixN(SMatrix22& m, int n) //矩阵的n次方
```

```
{  
    if(n == 0)  
    {  
        m.Set(1, 0, 0, 1); //单位阵  
        return;  
    }
```

```
    if(n == 1)  
        return;
```

```
    if(n % 2 == 0) //偶数
```

```
{  
        MatrixN(m, n/2);  
        MatrixMulti(m, m);  
    }
```

```
    else //奇数
```

```
{  
        SMatrix22 x = m;  
        MatrixN(m, n/2);  
        MatrixMulti(m, m);  
        MatrixMulti(m, x);  
    }
```

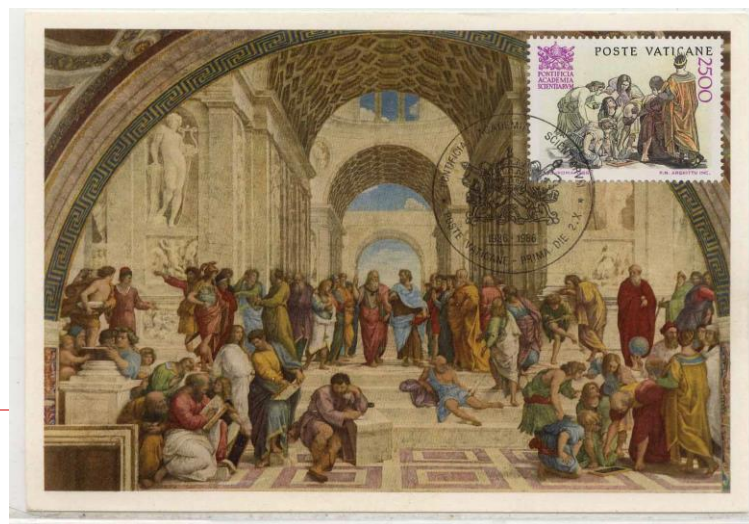
```
int CalcCount2(int n)
```

```
{  
    int nNonRepeat = 3;  
    int nRepeat = 0;  
    SMatrix22 m(2, 2, 1, 0);  
    MatrixN(m, n-1);  
    return 3*(m.a + m.c); // (3 0) * m  
}
```

# 算法



- 所学并非无用，而是知识体系尚未达到能够用它的程度。
- 思想恒久远，算法永流传
  - 据传，三国时期诸葛亮实施空城计。
  - 傅作义偷袭西柏坡，毛主席重演空城计。
- 三千年前欧几里德发明辗转相除法，迄今仍被广大程序员所使用。





# 结束语

□ 知识的掌握是1,2,4,8,16.....的速度;

■ 每天递增0.01:  $1.01^{365} = ?$

■ 设置适合自己的“学习率”。

□ 掌握算法的根本途径是多练习代码。

■ 书读百遍，其义自见。

□ 算法远远没有到此为止.....

■ 下列属于算法范畴吗?

■ 网站开发/OA工作流

■ 操作系统资源调度/编译原理词法、语法、语义分析/数据库设计/计算机网络协议包解析

■ 机器学习/数据挖掘/计算机视觉

■ .....

```
double Pow(double x, int n)
{
    if(n == 0)
        return 1;
    if(n == 1)
        return x;
    if(n == 2)
        return x*x;
    double p = Pow(x, n/2);
    p *= p;
    return (n % 2 == 0) ? p : p*x;
}

double Power(double x, int n)
{
    if(n < 0)
        return 1/Pow(x, -n);
    return Pow(x, n);
}

int _tmain(int argc, _TCHAR* argv[])
{
    cout << Power(1.01, 365) << endl;
    return 0;
}
```

# 我们在这里

□ <http://wenda.ChinaHadoop.cn>

■ 视频/课程/社区

□ 微博

■ @ChinaHadoop

■ @邹博\_机器学习

□ 微信公众号

■ 小象学院

■ 大数据分析挖掘

互联网新技术在线教育领航者

小象问答 搜索标题、用户 全站内容搜索 提问 首页 动态 发现 话题 通知

全部 招聘求职 机器学习 大数据平台技术 DCon 大数据行业应用 NoSQL数据库 数据科学 江湖救急

发现 最新 推荐 热门 等待回复

graphviz has no attribute 'write' 贡献  
邹博 回复了问题 • 2 人关注 • 1 个回复 • 3 次浏览 • 2017-04-09 15:48

sklearn中如何理解Pipeline机制 贡献  
数据分析与数据挖掘 邹博 回复了问题 • 2 人关注 • 1 个回复 • 28 次浏览 • 2017-04-09 15:39

关于9.Logistic回归的ppt中第9页的对数线性函数 贡献  
机器学习 邹博 回复了问题 • 3 人关注 • 3 个回复 • 39 次浏览 • 2017-04-09 15:35

关于“贝叶斯估计中，最大后验概率估计就是结构化风险最小化的例子：当模型是条件概率分布，损失函数为对数损失函数，模型的复杂度由模型的先验概率表示，结构化风险最小化就等价于最大后验概率估计” 贡献  
机器学习 邹博 回复了问题 • 2 人关注 • 1 个回复 • 26 次浏览 • 2017-04-09 15:27

关于连续值的预测 贡献  
咨询 邹博 回复了问题 • 2 人关注 • 1 个回复 • 31 次浏览 • 2017-04-09 15:24

拉格朗日对偶函数为什么一定是凸函数 贡献  
数据科学 邹博 回复了问题 • 2 人关注 • 2 个回复 • 26 次浏览 • 2017-04-09 15:20

梯度下降公式中的斯堪J 是 贡献  
机器学习 邹博 回复了问题 • 2 人关注 • 1 个回复 • 29 次浏览 • 2017-04-09 15:17

深度学习适合做预测吗？ 贡献  
深度学习 邹博 回复了问题 • 2 人关注 • 1 个回复 • 27 次浏览 • 2017-04-09 15:15

关于6.4PCA\_FeatureSelection.py中plt.legend的参数疑问 贡献  
机器学习 邹博 回复了问题 • 2 人关注 • 1 个回复 • 28 次浏览 • 2017-04-09 15:04

@邹博 有哪些可以下载数据源的网站？ 贡献  
数据分析与数据挖掘 邹博 回复了问题 • 4 人关注 • 1 个回复 • 31 次浏览 • 2017-04-09 14:53

LDA主题模型 贡献  
机器学习 邹博 回复了问题 • 2 人关注 • 1 个回复 • 29 次浏览 • 2017-04-09 14:45

代码10.6bagging\_ridged老师提到了采样率设为0.2能够使峰值部分的数据被体现出来。这是为什么呢？ 贡献  
机器学习 邹博 回复了问题 • 2 人关注 • 1 个回复 • 22 次浏览 • 2017-04-09 14:26

GraphViz's executables not found 贡献  
机器学习 邹博 回复了问题 • 3 人关注 • 2 个回复 • 23 次浏览 • 2017-04-09 13:47

决策树中关于feature\_importances代码的问题 贡献  
机器学习 邹博 回复了问题 • 2 人关注 • 1 个回复 • 6 次浏览 • 2017-04-09 13:11

专题  
招聘求职  
大数据行业应用  
数据科学  
系统与编程  
云计算技术

热门话题 更多 >  
机器学习 907 个问题, 230 人关注  
spark 387 个问题, 172 人关注  
hadoop 1059 个问题, 155 人关注  
python数据分析 171 个问题, 28 人关注  
数据分析与数据挖掘 54 个问题, 111 人关注

热门用户 更多 >  
小心巴 14 个问题, 0 次赞同  
叉叉V 45 个问题, 22 次赞同  
铁甲无声 10 个问题, 0 次赞同  
带刀锦衣卫 13 个问题, 0 次赞同

---

感谢大家！

恳请大家批评指正！