华东师范大学计算机科学技术系上机实践报告

课程名称：人工智能	**年级**：2016 级	**上机实践成绩**：
指导教师：周爱民	**姓名**：汪春雨	**创新实践成绩**：
上机实践名称：N皇后问题	**学号**：10152150127	**上机实践日期**：2018/3/27
上机实践编号：No.1	**组号**：	**上机实践时间**：

一、 问题介绍

1、问题描述

n皇后问题是一个以国际象棋为背景的问题：如何能够在 n×n 的国际象棋棋盘上放置n个皇后，使得任何一个皇后都无法直接吃掉其他的皇后？为了达到此目的，任两个皇后都不能处于同 一条横行、纵行或斜线上。

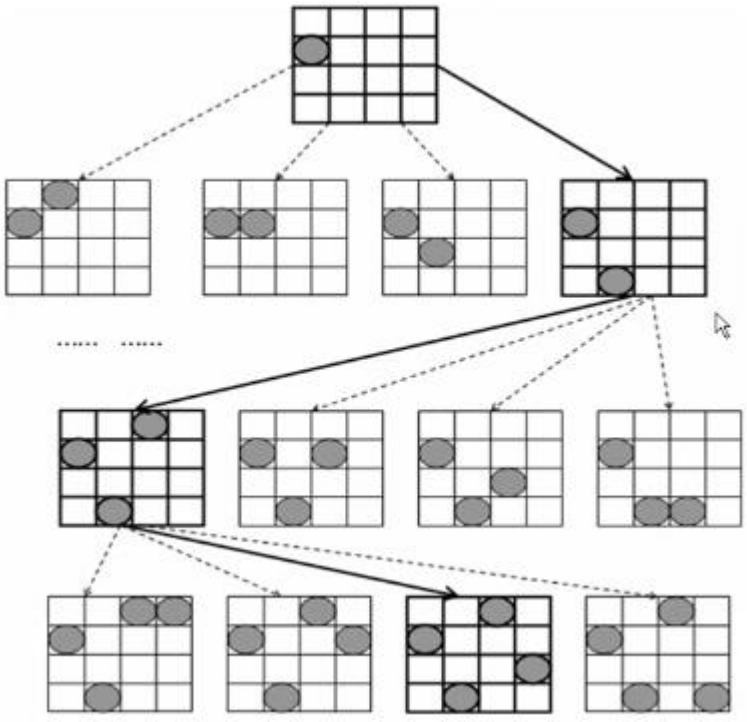
输入：皇后的数目，例 4

输出：第 1 行至第 N 行皇后对应的列号，例(1, 3, 0, 4)

2、求解方法

（1）回溯算法

先从首位开始检查，如果不能放置，接着检查该行第二个位置，依次检查下去，直到在该行找到一个可以放置一个皇后的地方，然后保存当前状态，转到下一行重复上述方法的检索。
如果检查了该行所有的位置均不能放置一个皇后，说明上一行皇后放置的位置无法让所有的皇后找到自己合适的位置，因此就要回溯到上一行，重新检查该皇后位置后面的位置。



（2）宽度优先搜索（bfs）

根据一般图搜索框架，对open表按深度递增排序

（3）深度优先搜索（dfs）

根据一般图搜索框架，对open表按深度递减排序

二、 程序设计与算法分析

（一）回溯算法

1、算法描述

1）满足目标（皇后个数是N），返回Ture，打印结果

2）根据下棋规则把一个新皇后放到下一行第一个位置

3）如果皇后安全，转 1）

4）回退到当前皇后，把当前皇后放到当前行的右边位置上，转 1）

（二）宽度优先搜索（bfs）

数据结构：

搜索节点 ：

```
class Node(object):
    def __init__(self,parent,state,deepth):
        self.parent=parent #标记父亲节点指针
        self.state=state#当前节点状态
        self.deepth=deepth#当前节点深度
```

有节点构成的图：

```
class Graph(object):
    def __init__(self):
        self.node=[]

    def add_node(self,node):
        self.node.append(node)
```

算法描述

1、构建图 G,并将初始节点加入到图 G 和 open 表中
2、Closed 表初始化，置空
3、如果 open 表为空，返回 false
4、取出 open 表第一个节点 n，并将 n 移除 open 表，加入 closed 表
5、如果 n 是目标节点，返回 True，打印结果
6、根据规则扩展 n 节点，把被扩展到的节点加入图 G 中
7、把以上被扩展的节点加入 open 表，标记指向父节点的指针
8、将 open 表中的节点**按照深度升序排列**
9、转 3

（三）深度优先搜索（dfs）

数据结构与2 相同

算法描述

1、构建图 G,并将初始节点加入到图 G 和 open 表中
2、Closed 表初始化，置空
3、如果 open 表为空，返回 false
4、取出 open 表第一个节点 n，并将 n 移除 open 表，加入 closed 表
5、如果 n 是目标节点，返回 True，打印结果
6、根据规则扩展 n 节点，把被扩展到的节点加入图 G 中
7、把以上被扩展的节点加入 open 表，标记指向父节点的指针
8、将 open 表中的节点按照**深度降序排列**
9、转 3

三、 实验结果

1、 回溯算法

N=4: (1,3,0,2)
N=5: (0,2,4,1,3)
N=6: (1,3,5,0,2,4)
N=7: (0,2,4,6,1,3,5)
N=8: (0,4,7,5,2,6,1,3)
N=9: (0,2,5,7,1,3,8,6,4)
N=10: (0,2,5,7,9,4,8,1,3,6)
N=11: (0,2,4,6,8,10,1,3,5,7,9)
N=12: (0,2,4,7,9,11,5,10,1,6,8,3)
N=13: (0,2,4,1,8,11,9,12,3,5,7,10,6)
N=14: (0,2,4,6,11,9,12,3,13,8,1,5,7,10)
N=15: (0,2,4,1,9,11,13,3,12,8,5,14,6,10,7)
N=16: (0,2,4,1,12,8,13,11,14,5,15,6,3,10,7,9)
N=17: (0,2,4,1,7,10,14,6,15,13,16,3,5,8,11,9,12)
N=18: (0,2,4,1,7,14,11,15,12,16,5,17,6,3,10,8,13,9)
N=19: (0,2,4,1,3,8,12,14,16,18,6,15,17,10,5,7,9,11,13)
N=20: (0,2,4,1,3,12,14,11,17,19,16,8,15,18,7,9,6,13,5,10)
N=21: (0,2,4,1,3,8,10,14,20,17,19,16,18,6,11,9,7,5,13,15,12)
N=22: (0,2,4,1,3,9,13,16,19,12,18,21,17,7,20,11,8,5,15,6,10,14)
N=23: (0,2,4,1,3,8,10,12,17,19,21,18,20,9,7,5,22,6,15,11,14,16,13)
N=24: (0,2,4,1,3,8,10,13,17,21,18,22,19,23,9,20,5,7,11,15,12,6,16,14)
N=25: (0,2,4,1,3,8,10,12,14,18,20,23,19,24,22,5,7,9,6,13,15,17,11,16,21)
N=26: (0,2,4,1,3,8,10,12,14,20,22,24,19,21,23,25,9,6,15,11,7,5,17,13,18,16)

N=27: (0, 2, 4, 1, 3, 8, 10, 12, 14, 16, 18, 22, 24, 26, 23, 25, 5, 9, 6, 15, 7, 11, 13, 20, 17, 19, 21)

N=28: (0, 2, 4, 1, 3, 8, 10, 12, 14, 16, 22, 24, 21, 27, 25, 23, 26, 6, 11, 15, 17, 7, 9, 13, 19, 5, 20, 18)

N=29: (0, 2, 4, 1, 3, 8, 10, 12, 14, 5, 19, 23, 25, 20, 28, 26, 24, 27, 7, 11, 6, 15, 9, 16, 21, 13, 17, 22, 18)

N=30:

(0, 2, 4, 1, 3, 8, 10, 12, 14, 6, 22, 25, 27, 24, 21, 23, 29, 26, 28, 15, 11, 9, 7, 5, 17, 19, 16, 13, 20, 18)

N=31:

(0, 2, 4, 1, 3, 8, 10, 12, 14, 5, 17, 22, 25, 27, 30, 24, 26, 29, 6, 16, 28, 13, 9, 7, 19, 11, 15, 18, 21, 23, 20)

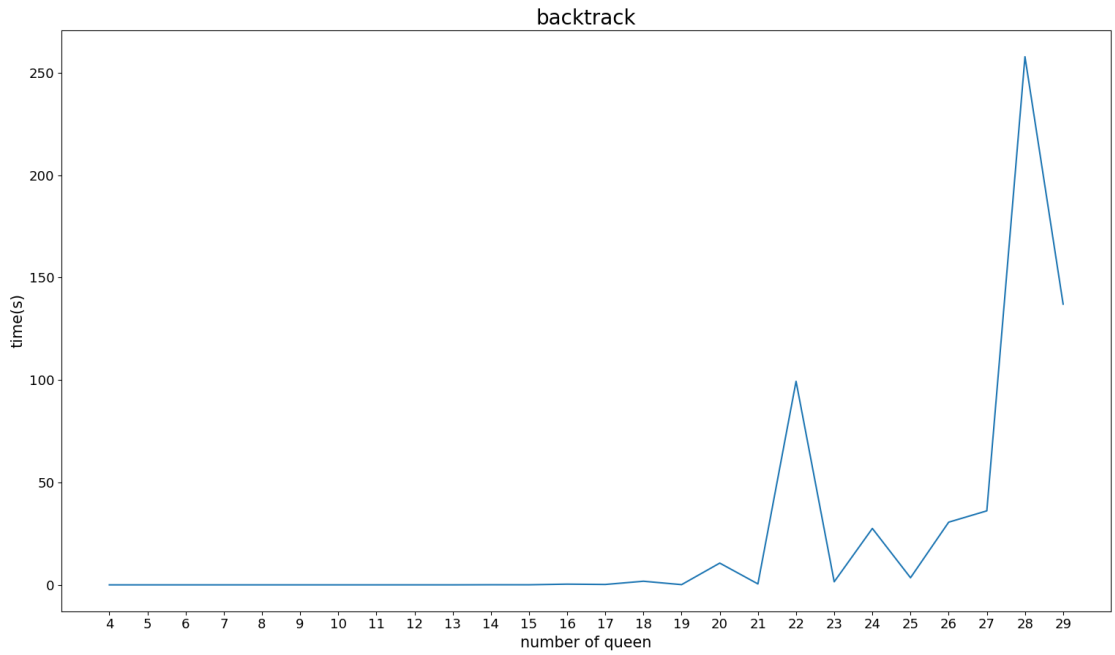
N=32:

(0, 2, 4, 1, 3, 8, 10, 12, 14, 5, 17, 23, 25, 29, 24, 30, 27, 31, 26, 28, 15, 18, 9, 7, 16, 11, 20, 6, 13, 22, 19, 21)

N=33:

(0, 2, 4, 1, 3, 8, 10, 12, 14, 5, 7, 24, 26, 32, 30, 22, 27, 25, 28, 31, 29, 15, 17, 11, 9, 16, 6, 13, 20, 18, 23, 21, 19)

时间评估：



2、 宽度优先搜索（一般图搜索方法）

格式： 皇后个数

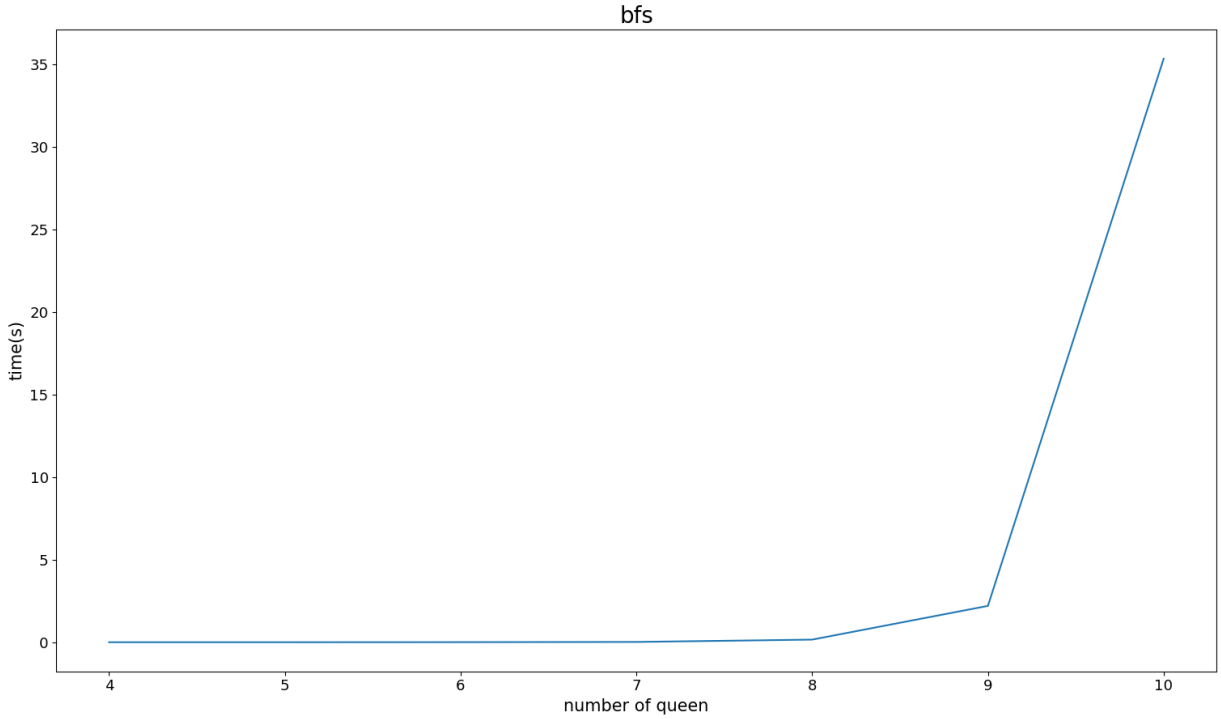
第 1 行至第 N 行皇后对应的列号，例[1, 3, 0, 2]

所有时间（s）

4
[1, 3, 0, 2]
0.0006041294514034234
5
[0, 2, 4, 1, 3]
0.0005569318379912147
6

[1, 3, 5, 0, 2, 4]
0.005119619519291518
7
[0, 2, 4, 6, 1, 3, 5]
0.017610373506727228
8
[0, 4, 7, 5, 2, 6, 1, 3]
0.15857793968439182
9
[0, 2, 5, 7, 1, 3, 8, 6, 4]
2.200871909830653
10
[0, 2, 5, 7, 9, 4, 8, 1, 3, 6]
35.33786996793208

时间评估：



3 深度优先搜索（一般图搜索方法）

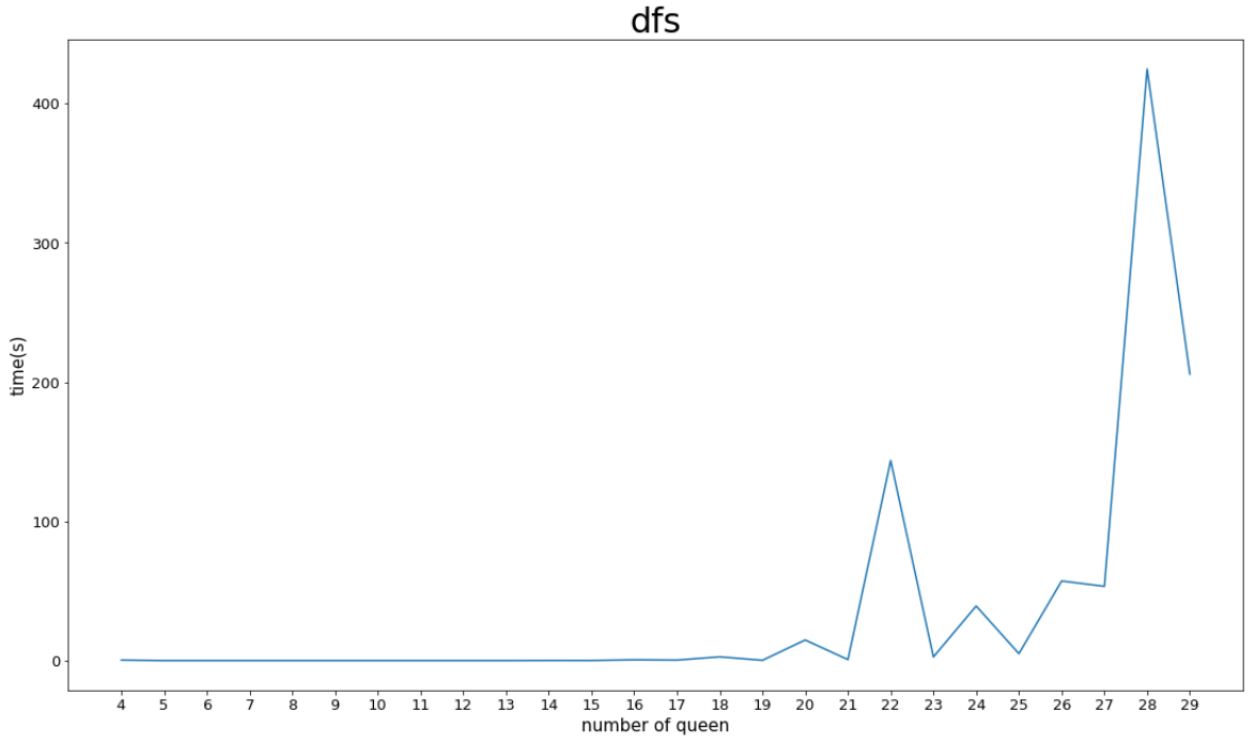
格式： 皇后个数
第 1 行至第 N 行皇后对应的列号，例[1, 3, 0, 2]
所有时间（s）

4
[1, 3, 0, 2]
0.14872760898106208
5
[0, 2, 4, 1, 3]
0.00017217689310200512
6

[1, 3, 5, 0, 2, 4]
0.0009205422520608408
7
[0, 2, 4, 6, 1, 3, 5]
0.00030130956474749837
8
[0, 4, 7, 5, 2, 6, 1, 3]
0.0036274197736929636
9
[0, 2, 5, 7, 1, 3, 8, 6, 4]
0.0016828781026561046
10
[0, 2, 5, 7, 9, 4, 8, 1, 3, 6]
0.005451513135994901
11
[0, 2, 4, 6, 8, 10, 1, 3, 5, 7, 9]
0.002696305256904452
12
[0, 2, 4, 7, 9, 11, 5, 10, 1, 6, 8, 3]
0.009596596335541108
13
[0, 2, 4, 1, 8, 11, 9, 12, 3, 5, 7, 10, 6]
0.0041084578497248
14
[0, 2, 4, 6, 11, 9, 12, 3, 13, 8, 1, 5, 7, 10]
0.08999150065392314
15
[0, 2, 4, 1, 9, 11, 13, 3, 12, 8, 5, 14, 6, 10, 7]
0.06510703097228543
16
[0, 2, 4, 1, 12, 8, 13, 11, 14, 5, 15, 6, 3, 10, 7, 9]
0.5215876219899656
17
[0, 2, 4, 1, 7, 10, 14, 6, 15, 13, 16, 3, 5, 8, 11, 9, 12]
0.3166185817372025
18
[0, 2, 4, 1, 7, 14, 11, 15, 12, 16, 5, 17, 6, 3, 10, 8, 13, 9]
2.690006830438506
19
[0, 2, 4, 1, 3, 8, 12, 14, 16, 18, 6, 15, 17, 10, 5, 7, 9, 11, 13]
0.178468524289201
20
[0, 2, 4, 1, 3, 12, 14, 11, 17, 19, 16, 8, 15, 18, 7, 9, 6, 13, 5, 10]
15.258219653162087
21
[0, 2, 4, 1, 3, 8, 10, 14, 20, 17, 19, 16, 18, 6, 11, 9, 7, 5, 13, 15, 12]
0.8020932330327923
22
[0, 2, 4, 1, 3, 9, 13, 16, 19, 12, 18, 21, 17, 7, 20, 11, 8, 5, 15, 6, 10, 14]
149.09703829312275
23

```
[0, 2, 4, 1, 3, 8, 10, 12, 17, 19, 21, 18, 20, 9, 7, 5, 22, 6, 15, 11, 14, 16, 13]
2.8850217052386142
24
[0, 2, 4, 1, 3, 8, 10, 13, 17, 21, 18, 22, 19, 23, 9, 20, 5, 7, 11, 15, 12, 6, 16, 14]
42.008153482107446
25
[0, 2, 4, 1, 3, 8, 10, 12, 14, 18, 20, 23, 19, 24, 22, 5, 7, 9, 6, 13, 15, 17, 11, 16, 21]
5.198346648725419
26
[0, 2, 4, 1, 3, 8, 10, 12, 14, 20, 22, 24, 19, 21, 23, 25, 9, 6, 15, 11, 7, 5, 17, 13, 18, 1
6]
46.105718500595685
27
[0, 2, 4, 1, 3, 8, 10, 12, 14, 16, 18, 22, 24, 26, 23, 25, 5, 9, 6, 15, 7, 11, 13, 20, 17, 1
9, 21]
54.64349867978672
28
[0, 2, 4, 1, 3, 8, 10, 12, 14, 16, 22, 24, 21, 27, 25, 23, 26, 6, 11, 15, 17, 7, 9, 13, 19,
5, 20, 18]
543.359155713988
29
[0, 2, 4, 1, 3, 8, 10, 12, 14, 5, 19, 23, 25, 20, 28, 26, 24, 27, 7, 11, 6, 15, 9, 16, 21, 1
3, 17, 22, 18]
211.4924825529306
```

时间评估：



4.分析

回溯法和深度优先搜索本质上没有太大区别，搜索较快，并且随着皇后个数的增加，时间成本增长不如宽度明显，而且特定的皇后个数下，耗时明显。

宽度优先搜索算法，耗时随皇后个数呈指数上升趋势，比较明显。

宽度优先总是可以找到最终解，但是深度优先不一定。

四、 附件

附件一：backtrack

附件二：bfs

附件三：dfs

```
#附件一：backtrack
```

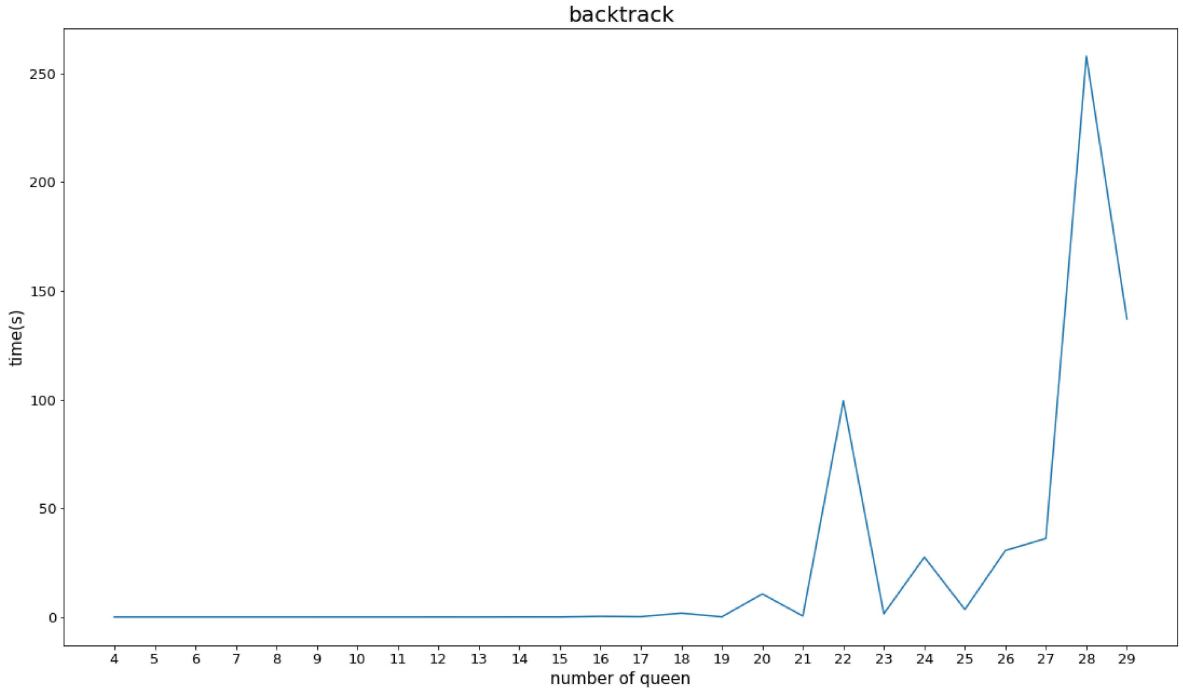
In [1]:

```
import time
%matplotlib inline
import matplotlib.pyplot as plt
global N  # 皇后个数


def print_solution(x):
    print(x)
    #fo = open("backtrack.txt", "a")
    #fo.write("N="+str(N)+":\t(")
    #for i in range(len(x) - 1):
        #fo.write(str(x[i]) + ",")
    #fo.write(str(x[len(x) - 1]) + ")" + "\n")
    #fo.close()


def is_safe(k):
    for i in range(k):
        if (x[i] == x[k]) or (abs(x[i] - x[k]) == (k - i)):
            return False
    return True


def backtrack(t):
    if t >= N:
        print_solution(x)
        return True
    else:
        for i in range(N):
            x[t] = i
            if is_safe(t):
                 if backtrack(t + 1):
                    return True
        return False


if __name__ == "__main__":
    res=[]
    numofqueen=range(4, 30)
    for k in numofqueen:
        global N
        N = k
        print(k)
        start=time.clock()
        x = [0 for i in range(N)]
        backtrack(0)
        end=time.clock()
        res.append(end-start)
        print(end-start)

    #统计绘图
    plt.plot(numofqueen,res)
    plt.title("backtrack",fontsize=20)
    plt.xlabel("number of queen",fontsize=15)
    plt.ylabel("time(s)",fontsize=15)
    plt.xticks(numofqueen,fontsize=13)
    plt.yticks(fontsize=13)
    fig = plt.gcf()
    fig.set_size_inches(18.5, 10.5)
```

```
    fig.savefig('bscktrack.png', dpi=100)
    plt.show
```

```
4
[1, 3, 0, 2]
0.0001431031637881786
5
[0, 2, 4, 1, 3]
3.360470073126095e-05
6
[1, 3, 5, 0, 2, 4]
0.00019558690987408052
7
[0, 2, 4, 6, 1, 3, 5]
6.305601148450082e-05
8
[0, 4, 7, 5, 2, 6, 1, 3]
0.0013736393400036778
9
[0, 2, 5, 7, 1, 3, 8, 6, 4]
0.00044592305127661965
10
[0, 2, 5, 7, 9, 4, 8, 1, 3, 6]
0.001544683490916725
11
[0, 2, 4, 6, 8, 10, 1, 3, 5, 7, 9]
0.0008295452528829248
12
[0, 2, 4, 7, 9, 11, 5, 10, 1, 6, 8, 3]
0.00517323600807872
13
[0, 2, 4, 1, 8, 11, 9, 12, 3, 5, 7, 10, 6]
0.0025671725873240812
14
[0, 2, 4, 6, 11, 9, 12, 3, 13, 8, 1, 5, 7, 10]
0.04858673354379693
15
[0, 2, 4, 1, 9, 11, 13, 3, 12, 8, 5, 14, 6, 10, 7]
0.03860689258842661
16
[0, 2, 4, 1, 12, 8, 13, 11, 14, 5, 15, 6, 3, 10, 7, 9]
0.3475734196634319
17
[0, 2, 4, 1, 7, 10, 14, 6, 15, 13, 16, 3, 5, 8, 11, 9, 12]
0.19394367776641253
18
[0, 2, 4, 1, 7, 14, 11, 15, 12, 16, 5, 17, 6, 3, 10, 8, 13, 9]
1.767282538884226
19
[0, 2, 4, 1, 3, 8, 12, 14, 16, 18, 6, 15, 17, 10, 5, 7, 9, 11, 13]
0.10812973226870648
20
[0, 2, 4, 1, 3, 12, 14, 11, 17, 19, 16, 8, 15, 18, 7, 9, 6, 13, 5, 10]
10.623559764827508
21
[0, 2, 4, 1, 3, 8, 10, 14, 20, 17, 19, 16, 18, 6, 11, 9, 7, 5, 13, 15, 12]
0.46282168477356045
22
```

```
[0, 2, 4, 1, 3, 9, 13, 16, 19, 12, 18, 21, 17, 7, 20, 11, 8, 5, 15, 6, 10, 14]
99.35312197109317
23
[0, 2, 4, 1, 3, 8, 10, 12, 17, 19, 21, 18, 20, 9, 7, 5, 22, 6, 15, 11, 14, 16, 13]
1.511164878632286
24
[0, 2, 4, 1, 3, 8, 10, 13, 17, 21, 18, 22, 19, 23, 9, 20, 5, 7, 11, 15, 12, 6, 16, 1
4]
27.540678112654277
25
[0, 2, 4, 1, 3, 8, 10, 12, 14, 18, 20, 23, 19, 24, 22, 5, 7, 9, 6, 13, 15, 17, 11, 1
6, 21]
3.467947723168237
26
[0, 2, 4, 1, 3, 8, 10, 12, 14, 20, 22, 24, 19, 21, 23, 25, 9, 6, 15, 11, 7, 5, 17, 1
3, 18, 16]
30.59283562883647
27
[0, 2, 4, 1, 3, 8, 10, 12, 14, 16, 18, 22, 24, 26, 23, 25, 5, 9, 6, 15, 7, 11, 13, 2
0, 17, 19, 21]
36.125261333185335
28
[0, 2, 4, 1, 3, 8, 10, 12, 14, 16, 22, 24, 21, 27, 25, 23, 26, 6, 11, 15, 17, 7, 9,
13, 19, 5, 20, 18]
257.76426151404655
29
[0, 2, 4, 1, 3, 8, 10, 12, 14, 5, 19, 23, 25, 20, 28, 26, 24, 27, 7, 11, 6, 15, 9, 1
6, 21, 13, 17, 22, 18]
137.03394074773854
```



#附件二：bfs

In [5]:

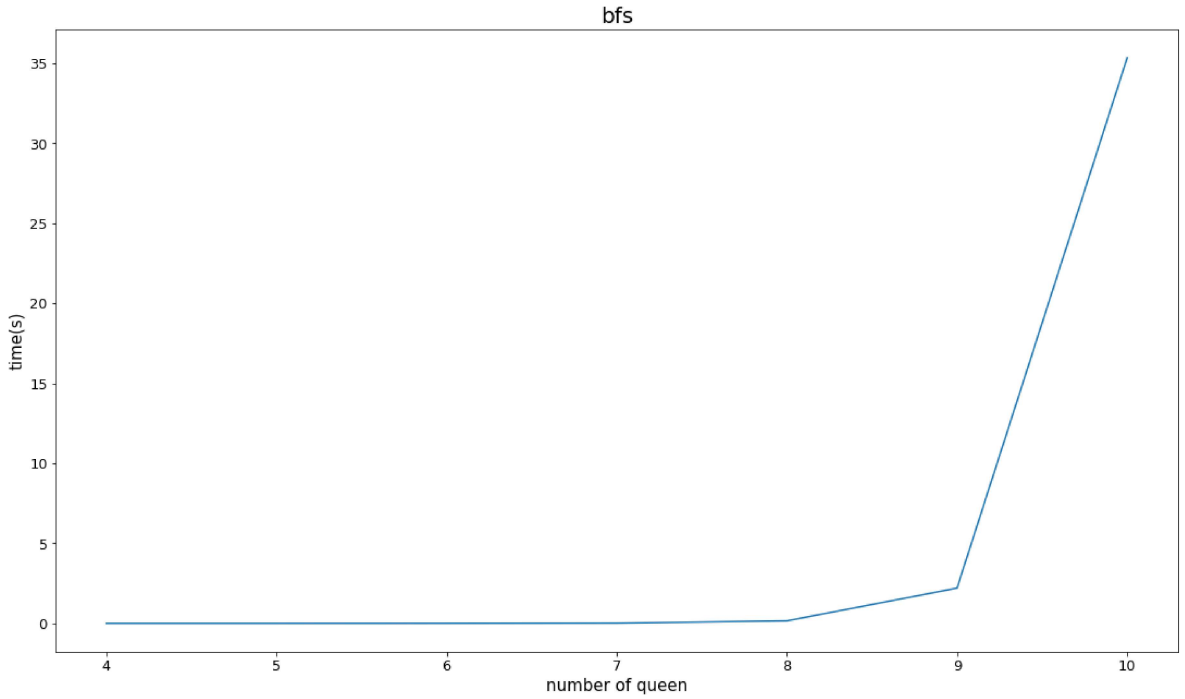
```
import time
%matplotlib inline
import matplotlib.pyplot as plt
global N

class Node(object):
    def __init__(self,parent,state,deepth):
        self.parent=parent #标记父亲节点指针
        self.state=state#当前节点状态
        self.deepth=deepth#当前节点深度


class Graph(object):
    def __init__(self):
        self.node=[]

    def add_node(self,node):
        self.node.append(node)


def goal(node):
    global N
    state = node.state
    #print(state)
    if(len(node.state)==N):
        print(N)
        print(node.state)
        return True
    else:
        return False


def issafe(state):
    k=len(state)-1
    for i in range(k):
        if (state[i] == state[k]) or (abs(state[i] - state[k]) == (k - i)):
            return False
    return True


def open_sort(open):
    open.sort(key=lambda k: k.deepth)

def expand(n,G,open):
    global N
    if(len(n.state)==N):
        return
    for i in range(N):
        deepth=n.deepth+1
        istate=list(n.state)
        istate.append(i)
        if(issafe(istate)==False):
            continue
        inode=Node(n,istate,deepth)
        G.add_node(inode)
        open.append(inode)
```

```
res=[]
numofqueen=range(4, 11)
for k in numofqueen:
    global N
    N = k

    start=time.clock()
    open=[]
    closed=[]
    G=Graph()
    s=Node(None,[ ],0)
    open.append(s)
    G.add_node(s)

    while(len(open)):
        n=open[0]
        open=open[1:]
        closed.append(n)
        if(goal(n)):
            break
        expand(n,G,open)
        open_sort(open)

    end=time.clock()
    res.append(end-start)
    print(end-start)

#统计绘图
plt.plot(numofqueen,res)
plt.title("bfs",fontsize=20)
plt.xlabel("number of queen",fontsize=15)
plt.ylabel("time(s)",fontsize=15)
plt.xticks(numofqueen,fontsize=13)
plt.yticks(fontsize=13)
fig = plt.gcf()
fig.set_size_inches(18.5, 10.5)
fig.savefig('bfs.png', dpi=100)
plt.show
```

```
4
[1, 3, 0, 2]
0.0006041294514034234
5
[0, 2, 4, 1, 3]
0.0005569318379912147
6
[1, 3, 5, 0, 2, 4]
0.005119619519291518
7
[0, 2, 4, 6, 1, 3, 5]
0.017610373506727228
8
[0, 4, 7, 5, 2, 6, 1, 3]
0.15857793968439182
9
[0, 2, 5, 7, 1, 3, 8, 6, 4]
2.200871909830653
10
[0, 2, 5, 7, 9, 4, 8, 1, 3, 6]
35.33786996793208
```

Out[5]:

<function matplotlib.pyplot.show>



In [ ]:

附件三：dfs

In [9]:

```
import time
%matplotlib inline
import matplotlib.pyplot as plt
global N

class Node(object):
    def __init__(self,parent,state,deepth):
        self.parent=parent
        self.state=state
        self.deepth=deepth


class Graph(object):
    def __init__(self):
        self.node=[]

    def add_node(self,node):
        self.node.append(node)


def goal(node):
    global N
    state = node.state
    #print(state)
    if(len(node.state)==N):
        print(N)
        print(node.state)
        return True
    else:
        return False


def issafe(state):
    k=len(state)-1
    for i in range(k):
        if (state[i] == state[k]) or (abs(state[i] - state[k]) == (k - i)):
            return False
    return True


def open_sort(open):
    open.sort(key=lambda k: -k.deepth)

def expand(n,G,open):
    global N
    if(len(n.state)==N):
        return
    for i in range(N):
        deepth=n.deepth+1
        istate=list(n.state)
        istate.append(i)
        if(issafe(istate)==False):
            continue
        inode=Node(n,istate,deepth)
```

```
        G.add_node(inode)
        open.append(inode)



res=[]
numofqueen=range(4, 30)
for k in numofqueen:
    global N
    N = k

    start=time.clock()
    open=[]
    closed=[]
    G=Graph()
    s=Node(None,[ ],0)
    open.append(s)
    G.add_node(s)

    while(len(open)):
        n=open[0]
        open=open[1:]
        closed.append(n)
        if(goal(n)):
            break
        expand(n,G,open)
        open_sort(open)

    end=time.clock()
    res.append(end-start)
    print(end-start)

#统计绘图
plt.plot(numofqueen,res)
plt.title("dfs",fontsize=30)
plt.xlabel("number of queen",fontsize=15)
plt.ylabel("time(s)",fontsize=15)
plt.xticks(numofqueen,fontsize=13)
plt.yticks(fontsize=13)
fig = plt.gcf()
fig.set_size_inches(18.5, 10.5)
fig.savefig('dfs.png', dpi=100)
plt.show
```

```
4
[1, 3, 0, 2]
0.3554320110815752
5
[0, 2, 4, 1, 3]
0.0004311973971198313
6
[1, 3, 5, 0, 2, 4]
0.0004474333763937466
7
[0, 2, 4, 6, 1, 3, 5]
0.0003186782851116732
8
[0, 4, 7, 5, 2, 6, 1, 3]
0.0023719632590655237
9
[0, 2, 5, 7, 1, 3, 8, 6, 4]
0.001050430080795195
10
[0, 2, 5, 7, 9, 4, 8, 1, 3, 6]
0.0024580517056165263
11
[0, 2, 4, 6, 8, 10, 1, 3, 5, 7, 9]
0.0014434918084589299
12
[0, 2, 4, 7, 9, 11, 5, 10, 1, 6, 8, 3]
0.008191617776901694
13
[0, 2, 4, 1, 8, 11, 9, 12, 3, 5, 7, 10, 6]
0.003867183648253558
14
[0, 2, 4, 6, 11, 9, 12, 3, 13, 8, 1, 5, 7, 10]
0.07866709408699535
15
[0, 2, 4, 1, 9, 11, 13, 3, 12, 8, 5, 14, 6, 10, 7]
0.06299597612087382
16
[0, 2, 4, 1, 12, 8, 13, 11, 14, 5, 15, 6, 3, 10, 7, 9]
0.5347610422613798
17
[0, 2, 4, 1, 7, 10, 14, 6, 15, 13, 16, 3, 5, 8, 11, 9, 12]
0.3305664204453933
18
[0, 2, 4, 1, 7, 14, 11, 15, 12, 16, 5, 17, 6, 3, 10, 8, 13, 9]
2.6906872312320047
19
[0, 2, 4, 1, 3, 8, 12, 14, 16, 18, 6, 15, 17, 10, 5, 7, 9, 11, 13]
0.1818969589257904
20
[0, 2, 4, 1, 3, 12, 14, 11, 17, 19, 16, 8, 15, 18, 7, 9, 6, 13, 5, 10]
14.774434676426608
21
[0, 2, 4, 1, 3, 8, 10, 14, 20, 17, 19, 16, 18, 6, 11, 9, 7, 5, 13, 15, 12]
0.699818270310061
22
[0, 2, 4, 1, 3, 9, 13, 16, 19, 12, 18, 21, 17, 7, 20, 11, 8, 5, 15, 6, 10, 14]
143.67276988444792
23
[0, 2, 4, 1, 3, 8, 10, 12, 17, 19, 21, 18, 20, 9, 7, 5, 22, 6, 15, 11, 14, 16, 13]
2.615855604002718
24
```

```
[0, 2, 4, 1, 3, 8, 10, 13, 17, 21, 18, 22, 19, 23, 9, 20, 5, 7, 11, 15, 12, 6, 16,
14]
39.18463366533979
25
[0, 2, 4, 1, 3, 8, 10, 12, 14, 18, 20, 23, 19, 24, 22, 5, 7, 9, 6, 13, 15, 17, 11,
16, 21]
4.970211131916585
26
[0, 2, 4, 1, 3, 8, 10, 12, 14, 20, 22, 24, 19, 21, 23, 25, 9, 6, 15, 11, 7, 5, 17,
13, 18, 16]
57.14820881281412
27
[0, 2, 4, 1, 3, 8, 10, 12, 14, 16, 18, 22, 24, 26, 23, 25, 5, 9, 6, 15, 7, 11, 13,
20, 17, 19, 21]
53.30370455955563
28
[0, 2, 4, 1, 3, 8, 10, 12, 14, 16, 22, 24, 21, 27, 25, 23, 26, 6, 11, 15, 17, 7,
9, 13, 19, 5, 20, 18]
424.7855563220437
29
[0, 2, 4, 1, 3, 8, 10, 12, 14, 5, 19, 23, 25, 20, 28, 26, 24, 27, 7, 11, 6, 15, 9,
16, 21, 13, 17, 22, 18]
205.65194063371018
```

Out[9]:

<function matplotlib.pyplot.show>