

习题课

·AIPKU·

人工智能引论实践课 计算机视觉小班

主讲人：胡越予

作业：

1. 对下述断言，请判断真假并给出解释或支持实例
 - a. 一个Agent只能感知状态的部分信息，那么它不可能是完美理性的.
 - b. Agent程序与Agent函数的输入是相同的.
 - c. 在不可观测的环境中，每个Agent都是理性的.
 - d. 一个完美理性的玩扑克Agent是不可能输的.
2. 请写出基于目标的Agent的伪代码Agent程序

- a. 一个Agent只能感知状态的部分信息，那么它不可能是完美理性的。
- False. Perfect rationality refers to the ability to make good decisions given any sensory information received.

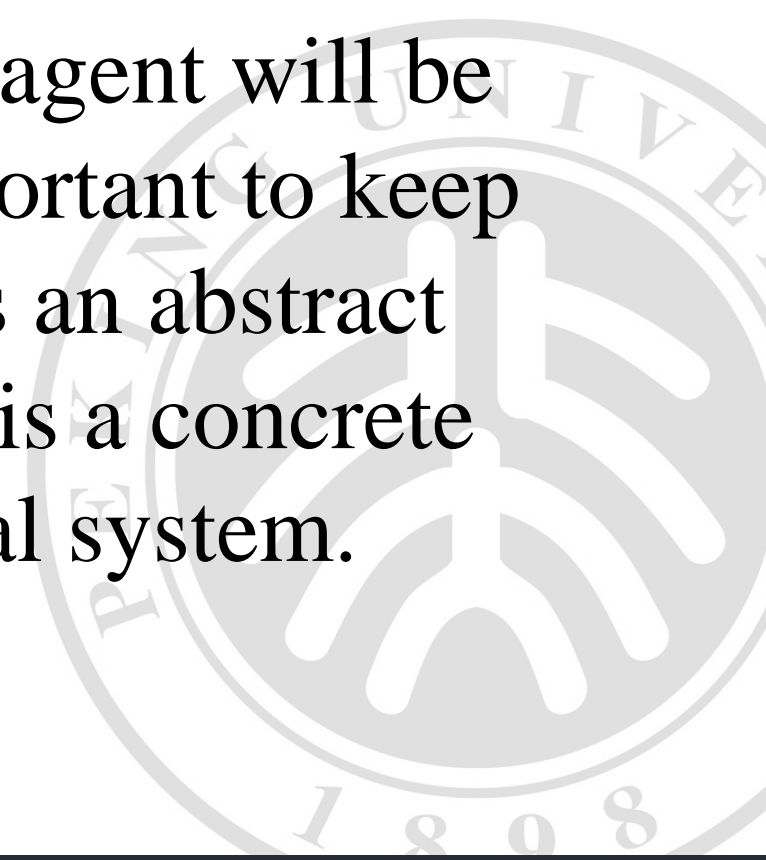
Perfect rationality. A perfectly rational agent PERFECT acts at every instant in such a way as to RATIONALITY maximize its expected utility, given the information it has acquired from the environment. We have seen that the calculations necessary to achieve perfect rationality in most environments are too time consuming, so perfect rationality is not a realistic goal. p. 1049

- b. Agent**程序与Agent函数的输入是相同的**.
- False. The agent function, notionally speaking, takes as input the entire percept sequence up to that point, whereas the agent program usually takes the current percept only.

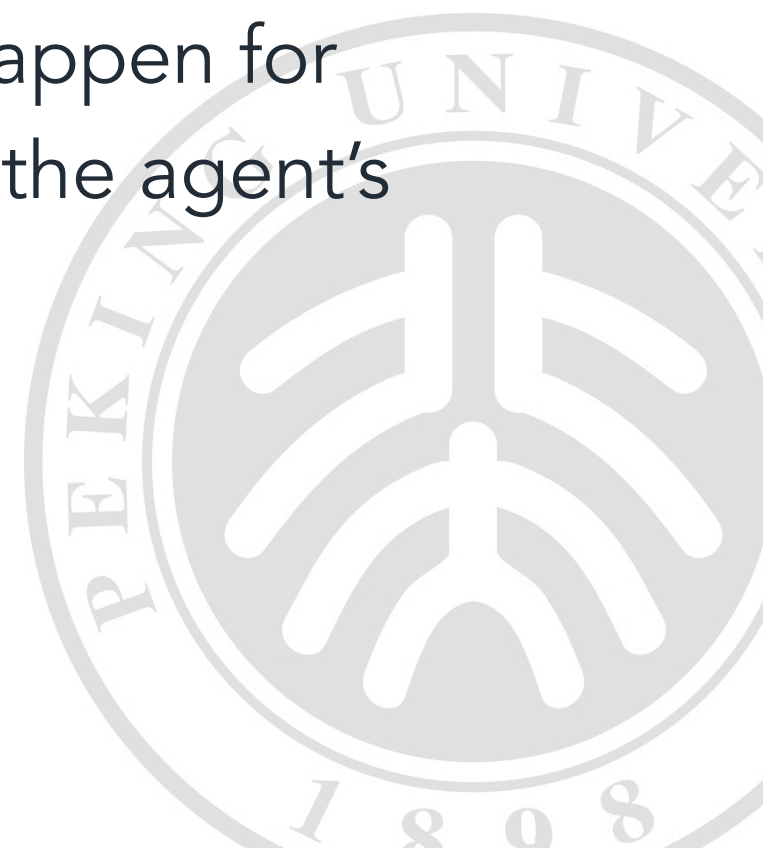


Agent Function. Mathematically speaking, we say that an agent's behavior is described by the agent function **AGENT FUNCTION** that maps any given percept sequence to an action.

Internally, the agent function for an artificial agent will be implemented by an **agent program**. It is important to keep these two ideas distinct. The agent function is an abstract mathematical description; the agent program is a concrete implementation, running within some physical system.

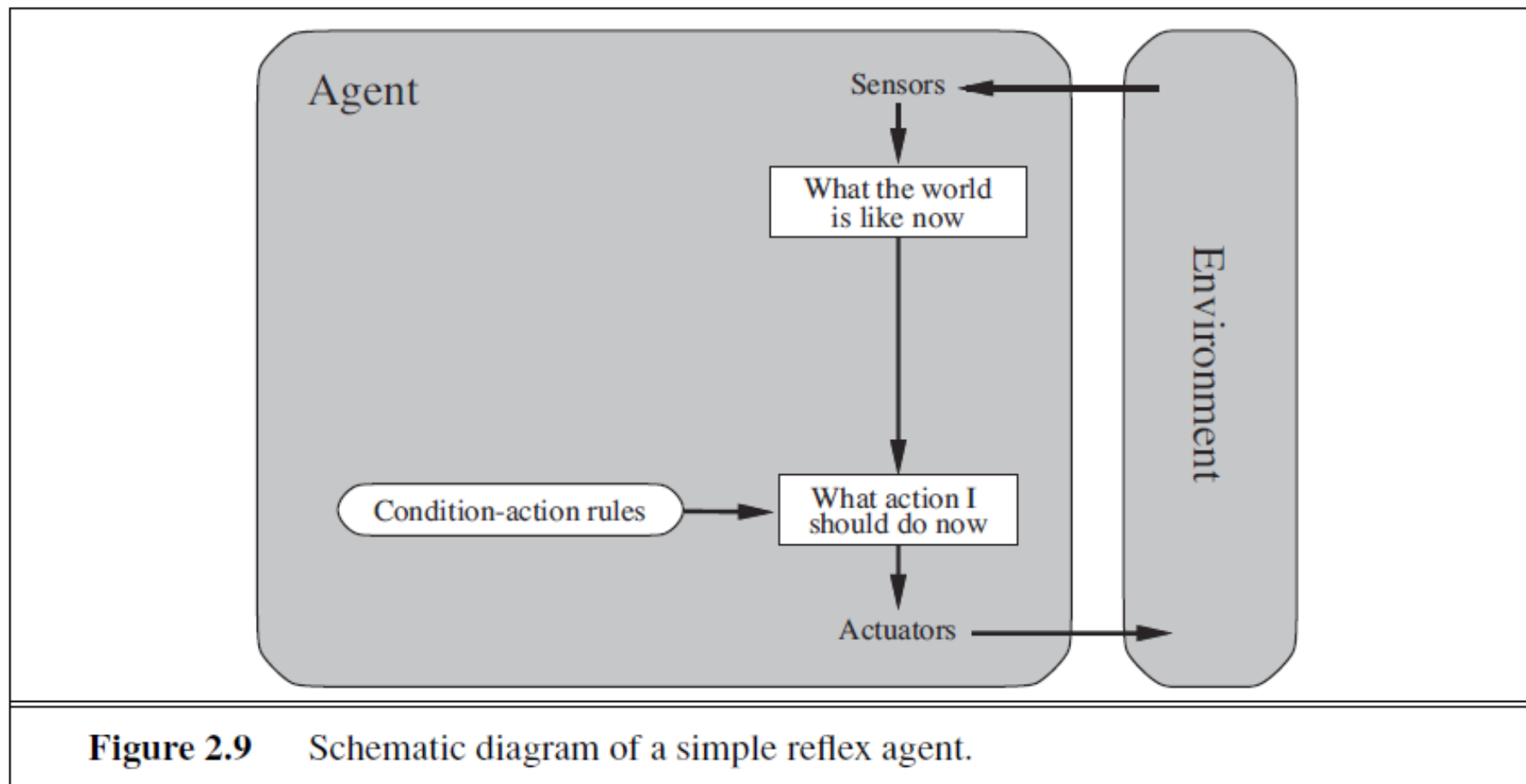


- c. **在不可观测的环境中，每个Agent都是理性的。**
- False. Some actions are stupid—and the agent may know this if it has a model of the environment—even if one cannot perceive the environment state.
- d. **一个完美理性的玩扑克Agent是不可能输的。**
- False. Unless it draws the perfect hand, the agent can always lose if an opponent has better cards. This can happen for game after game. The correct statement is that the agent's expected winnings are nonnegative.



- 2. 请写出基于目标的Agent的伪代码Agent程序





function SIMPLE-REFLEX-AGENT(*percept*) **returns** an action

persistent: *rules*, a set of condition–action rules

state ← INTERPRET-INPUT(*percept*)

rule ← RULE-MATCH(*state*, *rules*)

action ← *rule*.ACTION

return *action*

Figure 2.10 A simple reflex agent. It acts according to a rule whose condition matches the current state, as defined by the percept.

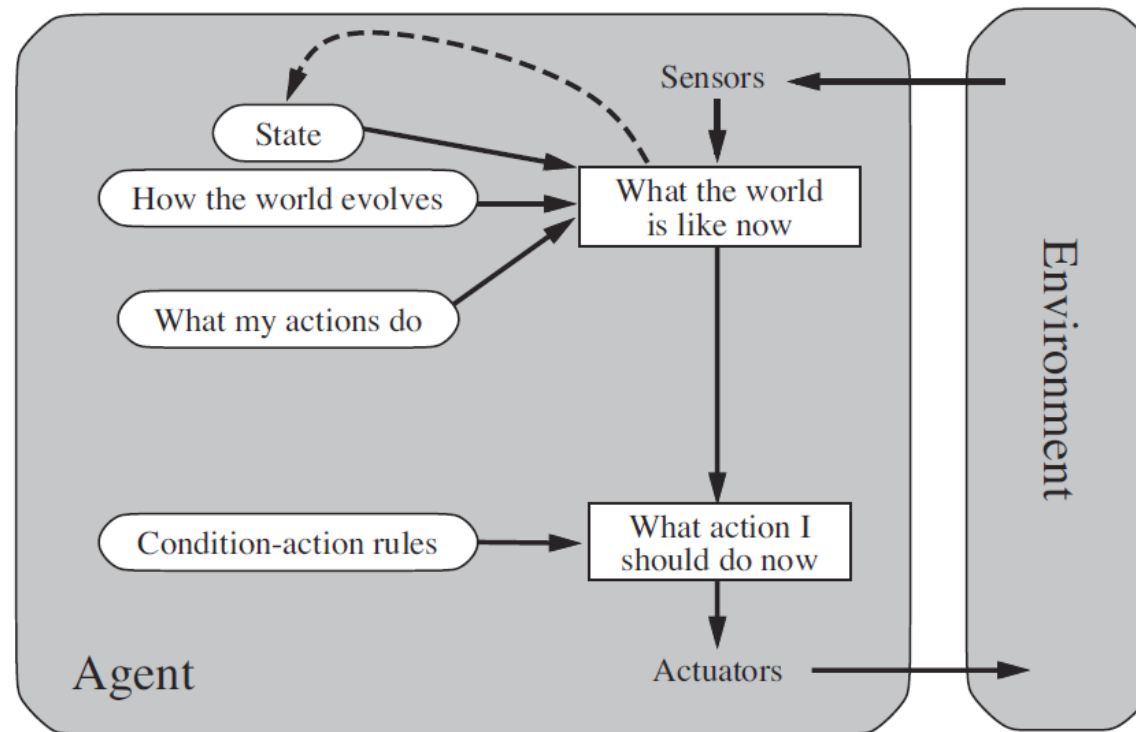


Figure 2.11 A model-based reflex agent.

function MODEL-BASED-REFLEX-AGENT(*percept*) **returns** an action
persistent: *state*, the agent's current conception of the world state
model, a description of how the next state depends on current state and action
rules, a set of condition–action rules
action, the most recent action, initially none

```

state ← UPDATE-STATE(state, action, percept, model)
rule ← RULE-MATCH(state, rules)
action ← rule.ACTION
return action
    
```

Figure 2.12 A model-based reflex agent. It keeps track of the current state of the world, using an internal model. It then chooses an action in the same way as the reflex agent.



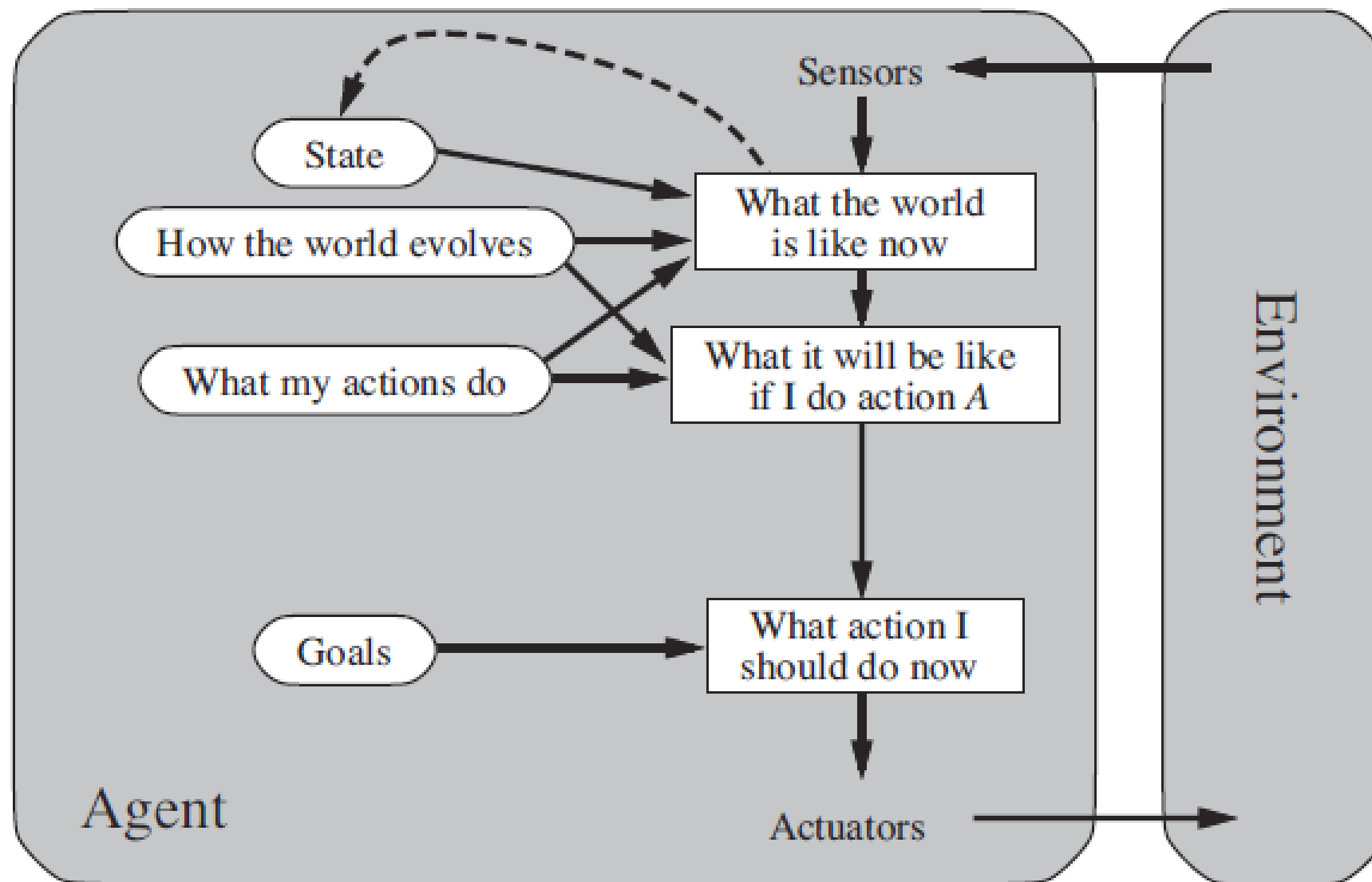


Figure 2.13 A model-based, goal-based agent. It keeps track of the world state as well as a set of goals it is trying to achieve, and chooses an action that will (eventually) lead to the achievement of its goals.

• 2. 请写出基于目标的Agent的伪代码Agent程序

```
function GOAL-BASED-AGENT(percept) returns an action
  persistent: state, the agent's current conception of the world state
               model, a description of how the next state depends on current state and action
               goal, a description of the desired goal state
               plan, a sequence of actions to take, initially empty
               action, the most recent action, initially none

  state ← UPDATE-STATE(state, action, percept, model)
  if GOAL-ACHIEVED(state, goal) then return a null action
  if plan is empty then
    plan ← PLAN(state, goal, model)
    action ← FIRST(plan)
    plan ← REST(plan)
  return action
```


● Sudoku

1. 问题描述

状态：一个已填若干数字的数独

初始状态：任意状态都可以作为初始状态

动作：选择一个空格，选择一个不与同行同列同九宫格冲突的1~9数字填入

转移模型：填入数字后得到空格数少1的新数独

目标测试：所有空格均填满

路径成本：无



●2. 伪代码（深搜）

state = 初始棋盘

function dfs():

 pos = 当前state中第一个空格位置

 if pos不存在:

 print 当前棋盘

 return true

 for i = 1 to 9 do

 if 在pos位置填i冲突: continue

 在pos位置填入i

 if dfs(): return true

 将pos位置清0

 return false



● 3. 算法性能分析

完备性：如果问题有解，该算法能找到一组解（深搜能够遍历完搜索树）

最优性：此问题的解没有最优性

时间复杂度：遍历完搜索树（上界）为 $O(9^n)$ ，其中 n 为空格个数，实际只会遍历很少的节点就能找到一个解

空间复杂度： $O(n)$

【搜索问题的时间复杂度一般是指指数级别的】

【空间复杂度：需要记录状态】



- The missionaries and cannibals

1. 问题描述

状态：河两岸的野人和传教士所有安全的数量（均只有一类人或者野人数量不超过传教士），以及小船在河的某一岸

初始状态：一岸有 m 个野人和 m 个传教士，且小船在这一岸

动作：小船从小船所在的一岸运送 p 个野人和 q 个传教士到另一岸，且运送后两岸的人数、船上人数均安全（只有一类人或野人数量不超过传教士）

转移模型：一次运送后小船到达另一岸，且两岸的人数相应变化

目标测试： m 个野人和 m 个传教士均抵达对岸

路径成本：小船每运送一次花费代价1，路径成本为总的运送次数

2. 算法伪代码

```
function bfs():
```

```
    init_state = {左岸m个野人, m个传教士, 船在左岸}
```

```
    queue = an empty FIFO queue
```

```
    queue.push(init_state)
```

```
    cost[init_state] = 0
```

```
    while not empty(queue):
```

```
        node = queue.pop()
```

```
        for each boat configuration: config={船上p个野人, q个传教士} with  $p+q$ 
```

```
<= n:
```

```
            if 船上不安全 or 运送后两岸不安全: continue
```

```
            child = next_state(node, config)
```

```
            if visited[child]: continue
```

```
            visited[child] = true
```

```
            cost[child] = cost[node] + 1
```

```
            if child == {右岸m个野人, m个传教士, 船在右岸}: return cost[child]
```

```
            queue.push(child)
```

```
return -1
```



3. 算法性能分析

完备性：如果问题有解，该算法能找到一组解（宽搜能够遍历完搜索树）

最优性：该算法能找到最优解（宽搜能够找到最优解）

时间复杂度：遍历完搜索树（上界）为 $O(m^2n^2)$ ，实际只会遍历很浅的深度就能找到最优解

空间复杂度： $O(m^2)$




```
double get_dir()
{
    int r = rand() % 10000;
    double dir = r * 2 * pi / 10000.;
    return dir;
}

double distance(double x, double y)
{
    double res = 0;
    for (int i = 0; i < n; ++i)
    {
        double dx = x - coors[i][0];
        double dy = y - coors[i][1];
        res += sqrt(dx*dx + dy*dy);
    }
    return res;
}
```

```
srand(131); // try different seeds
double cur_x = (left + right) / 2;
double cur_y = (up + down) / 2;
double cur_d = distance(cur_x, cur_y);
double t = 1e5;
int iter = 128;

while(t > 0.0001)
{
    for (int i = 0; i < iter; ++i)
    {
        double dir = get_dir();
        double dx = t * cos(dir);
        double dy = t * sin(dir);
        double new_x = cur_x + dx;
        double new_y = cur_y + dy;
        double new_d = distance(new_x, new_y);
```



```

if(new_d < cur_d)
{
    cur_x = new_x;
    cur_y = new_y;
    cur_d = new_d;
}
else
{
    double delta_E = cur_d - new_d;
    double p = exp(delta_E / t);
    if (rand() % 10000 < p * 10000)
    {
        cur_x = new_x;
        cur_y = new_y;
        cur_d = new_d;
    }
}
}
t *= 0.99;
}

```

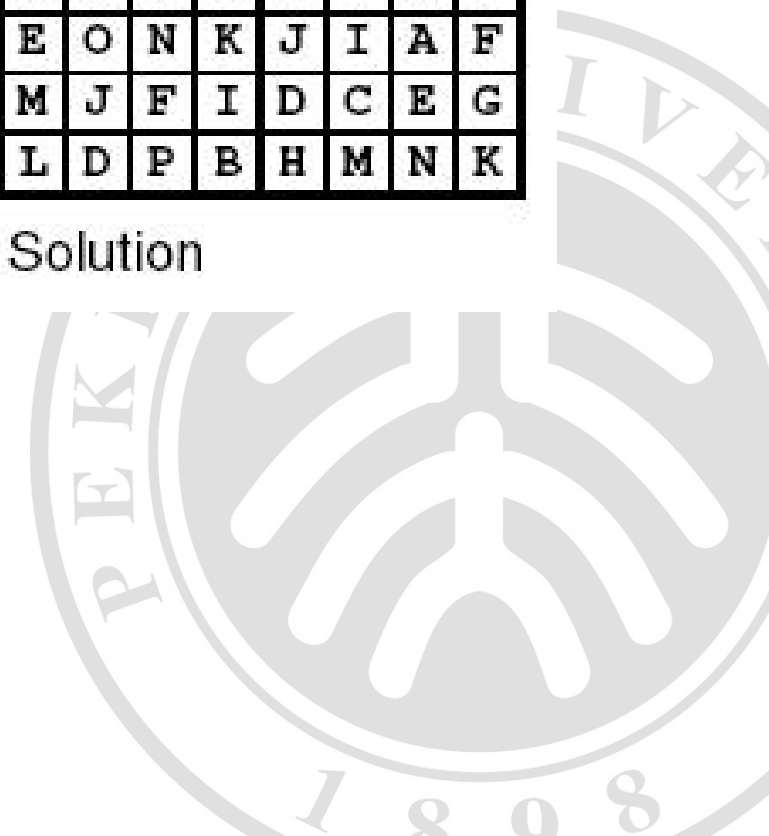

		A					C						O		I
	J			A		B		P		C	G	F		H	
		D			F		I		E					P	
	G		E	L		H					M		J		
				E					C			G			
	I			K		G	A		B				E		J
D		G	P			J		F					A		
	E				C		B			D	P			O	
E			F		M			D			L		K		A
	C									O		I		L	
H		P		C			F		A			B			
			G		O	D				J					H
K				J					H		A		P		L
		B			P			E			K			A	
	H			B			K			F	I		C		
		F				C			D			H		N	

a) Sudoku grid

F	P	A	H	M	J	E	C	N	L	B	D	K	O	G	I
O	J	M	I	A	N	B	D	P	K	C	G	F	L	H	E
L	N	D	K	G	F	O	I	J	E	A	H	M	B	P	C
B	G	C	E	L	K	H	P	O	F	I	M	A	J	D	N
M	F	H	B	E	L	P	O	A	C	K	J	G	N	I	D
C	I	L	N	K	D	G	A	H	B	M	O	P	E	F	J
D	O	G	P	I	H	J	M	F	N	L	E	C	A	K	B
J	E	K	A	F	C	N	B	G	I	D	P	L	H	O	M
E	B	O	F	P	M	I	J	D	G	H	L	N	K	C	A
N	C	J	D	H	B	A	E	K	M	O	F	I	G	L	P
H	M	P	L	C	G	K	F	I	A	E	N	B	D	J	O
A	K	I	G	N	O	D	L	B	P	J	C	E	F	M	H
K	D	E	M	J	I	F	N	C	H	G	A	O	P	B	L
G	L	B	C	D	P	M	H	E	O	N	K	J	I	A	F
P	H	N	O	B	A	L	K	M	J	F	I	D	C	E	G
I	A	F	J	O	E	C	G	L	D	P	B	H	M	N	K

b) Solution

Figure 1. Sudoku



- 0-1 矩阵精确覆盖问题
 - 找到矩阵中的某几行，选择这几行之后能够让1覆盖所有的列

$$\begin{pmatrix} 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \end{pmatrix}$$

1, 4, 5



- 回溯法：
- 1. (尝试) 选择第一行
- 2. 把与第一行中的1冲突的行（紫色）去掉
 - 选中第一行之后这些紫色行不能再选了

$$\begin{pmatrix}
 0 & 0 & 1 & 0 & 1 & 1 & 0 \\
 1 & 0 & 0 & 1 & 0 & 0 & 1 \\
 0 & 1 & 1 & 0 & 0 & 1 & 0 \\
 1 & 0 & 0 & 1 & 0 & 0 & 0 \\
 0 & 1 & 0 & 0 & 0 & 0 & 1 \\
 0 & 0 & 0 & 1 & 1 & 0 & 1
 \end{pmatrix}$$



- 回溯法：
- 3. 剩下的行（黑色）中，重复同样的操作
 - A. 选择第一行
 - B. 删除掉占用的其它行
- 有解？ 算法结束。无解？ 选错了行，回溯。

$$\begin{pmatrix} 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \end{pmatrix}$$

$$\begin{pmatrix} 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{pmatrix}$$



- 精确覆盖问题如何对应数独问题？
- 互相制约的选择问题：
 - 1、 每个格子只能填一个数字
 - 2、 每行1-9的这9个数字都得填一遍（也就意味着每个数字只能填一遍）
 - 3、 每列1-9的这9个数字都得填一遍
 - 4、 每宫1-9的这9个数字都得填一遍



- 把矩阵的每个列都定义成一个约束条件。
- 每个格子只能填一个数字
 - 第1列定义成: $(1, 1)$ 填了一个数字
 - 第2列定义成: $(1, 2)$ 填了一个数字
 -
 - 第9列定义成: $(1, 9)$ 填了一个数字
 - 第10列定义成: $(2, 1)$ 填了一个数字
 -
 - 第18列定义成: $(2, 9)$ 填了一个数字
 -
 - 第81列定义成: $(9, 9)$ 填了一个数字
- **每一列都是要在精确覆盖问题中满足的**



- 每行1-9的这9个数字都得填一遍
 - 第82列定义成：在第1行填了数字1
 - 第83列定义成：在第1行填了数字2
 -
 - 第90列定义成：在第1行填了数字9
 - 第91列定义成：在第2行填了数字1
 -
 - 第99列定义成：在第2行填了数字9
 -
 - 第162列定义成：在第9行填了数字9
- **每一列都是要在精确覆盖问题中满足的**



- 每列1-9这9个数字都得填一遍
 - 第163列定义成：在第1列填了数字1
 - 第164列定义成：在第1列填了数字2
 -
 - 第171列定义成：在第1列填了数字9
 - 第172列定义成：在第2列填了数字1
 -
 - 第180列定义成：在第2列填了数字9
 -
 - 第243列定义成：在第9列填了数字9
- **每一列都是要在精确覆盖问题中满足的**



- 第244列定义成：在第1宫填了数字1
- 第245列定义成：在第1宫填了数字2
-
- 第252列定义成：在第1宫填了数字9
- 第253列定义成：在第2宫填了数字1
-
- 第261列定义成：在第2宫填了数字9
-
- 第324列定义成：在第9宫填了数字9



- 现在我们定义了列，但是还没有矩阵
- 我们需要很多行来满足这些列
- 那么都有哪些可以用的行呢？



- 已经有的数字：
- 把 (4, 2) 中填的是7，解释成4个约束条件
 - 1、在 (4, 2) 中填了一个数字。
 - 2、在第4行填了数字7
 - 3、在第2列填了数字7
 - 4、在第4宫填了数字7 (坐标 (X, Y) 到宫N的公式为： $N = \text{INT}((X-1)/3) \times 3 + \text{INT}((Y-1)/3) + 1$)

注意，有数字的格子一定是不互相冲突的，所以都能选上



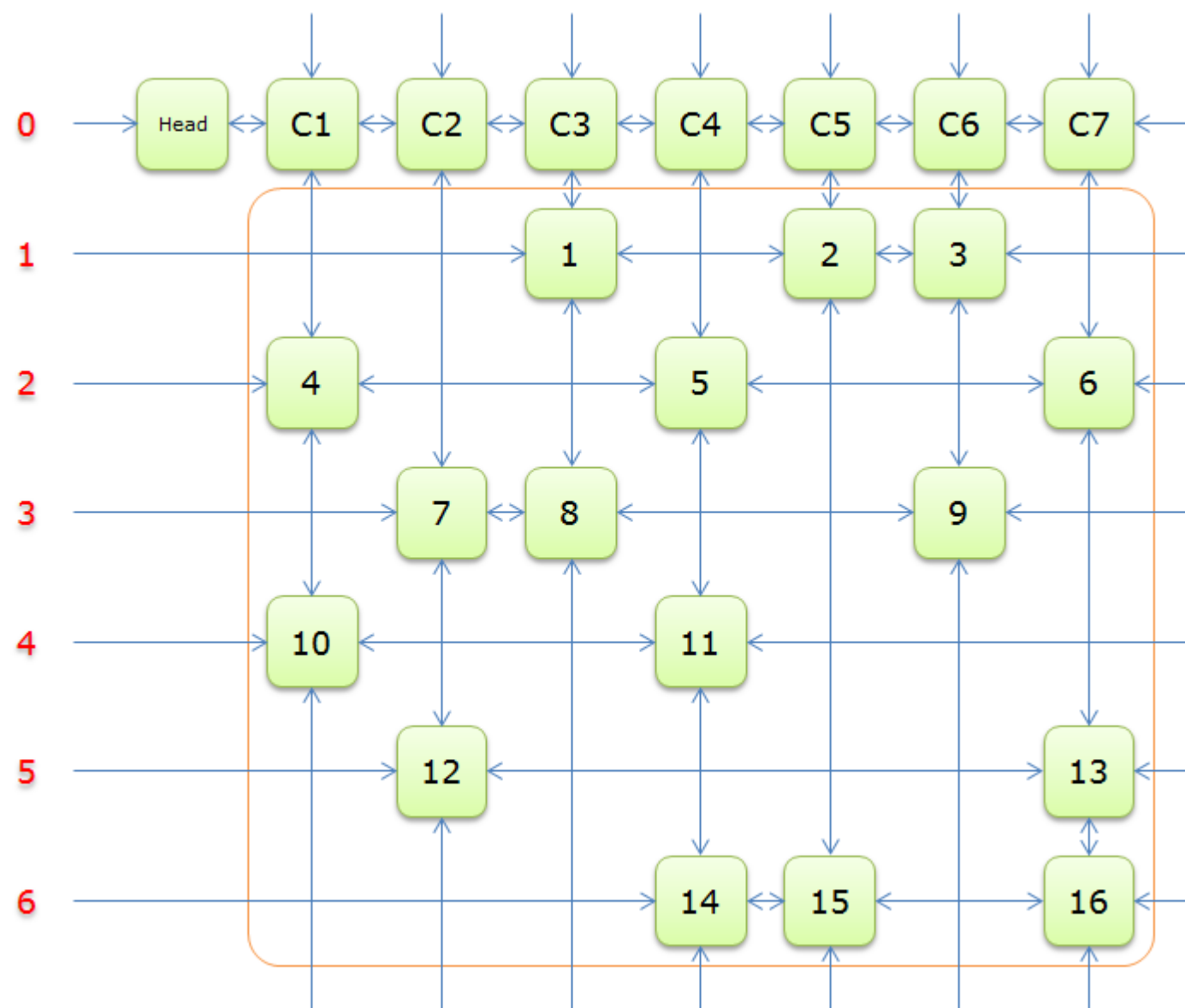
- 没数字的格子
- 把 (5, 8) 中没有数字转换成
 - (5, 8) 中填的是1, 转成矩阵的一行就是, 第44、118、226、289列是1, 其余列是0。
 - (5, 8) 中填的是2, 转成矩阵的一行就是, 第44、119、227、290列是1, 其余列是0。
 - (5, 8) 中填的是3, 转成矩阵的一行就是, 第44、120、228、291列是1, 其余列是0。
 - (5, 8) 中填的是4, 转成矩阵的一行就是, 第44、121、229、292列是1, 其余列是0。
 - (5, 8) 中填的是5, 转成矩阵的一行就是, 第44、122、230、293列是1, 其余列是0。
 - (5, 8) 中填的是6, 转成矩阵的一行就是, 第44、123、231、294列是1, 其余列是0。
 - (5, 8) 中填的是7, 转成矩阵的一行就是, 第44、124、232、295列是1, 其余列是0。
 - (5, 8) 中填的是8, 转成矩阵的一行就是, 第44、125、233、296列是1, 其余列是0。
 - (5, 8) 中填的是9, 转成矩阵的一行就是, 第44、126、234、297列是1, 其余列是0。
- **没有数字的格子对应9行, 这9行是互相冲突的, 并且可能会和已有的行冲突**
- **只要满足所有列, 就是填好的数独**
- **只要没有冲突, 填法就是合法的**



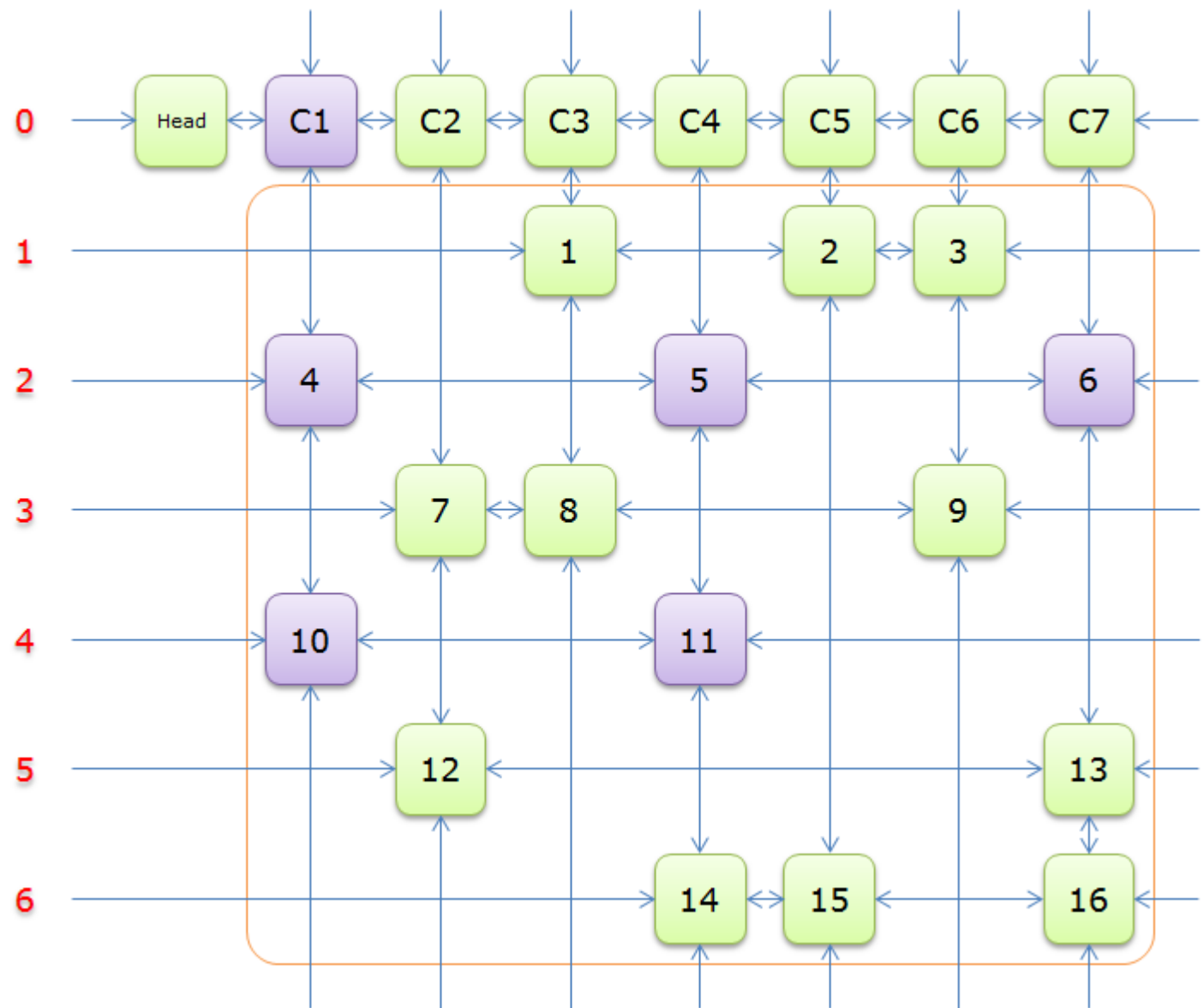
- 求解精确覆盖问题
 - 其核心瓶颈是对行/列的删除和回溯
 - 使用Dancing Links数据结构
 - 十字交叉循环双向链表



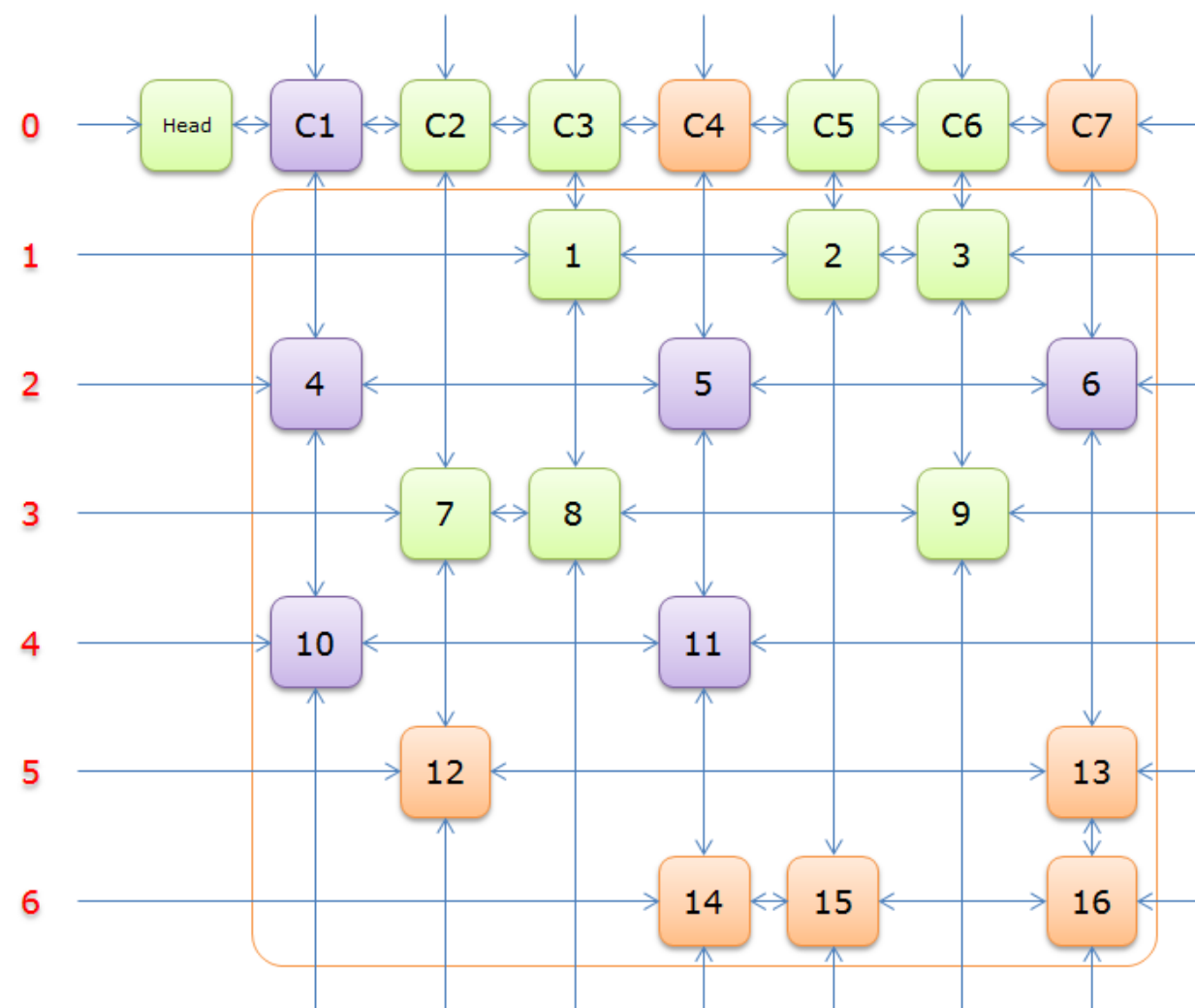
- Head, 指向列标链表
- 访问C7R5 (13号) 元素?
 - 找到 head
 - 按顺序找到第7列
 - 按顺序找到第5行
- 找到所有第3列的元素?
- 找到所有第2行的元素?



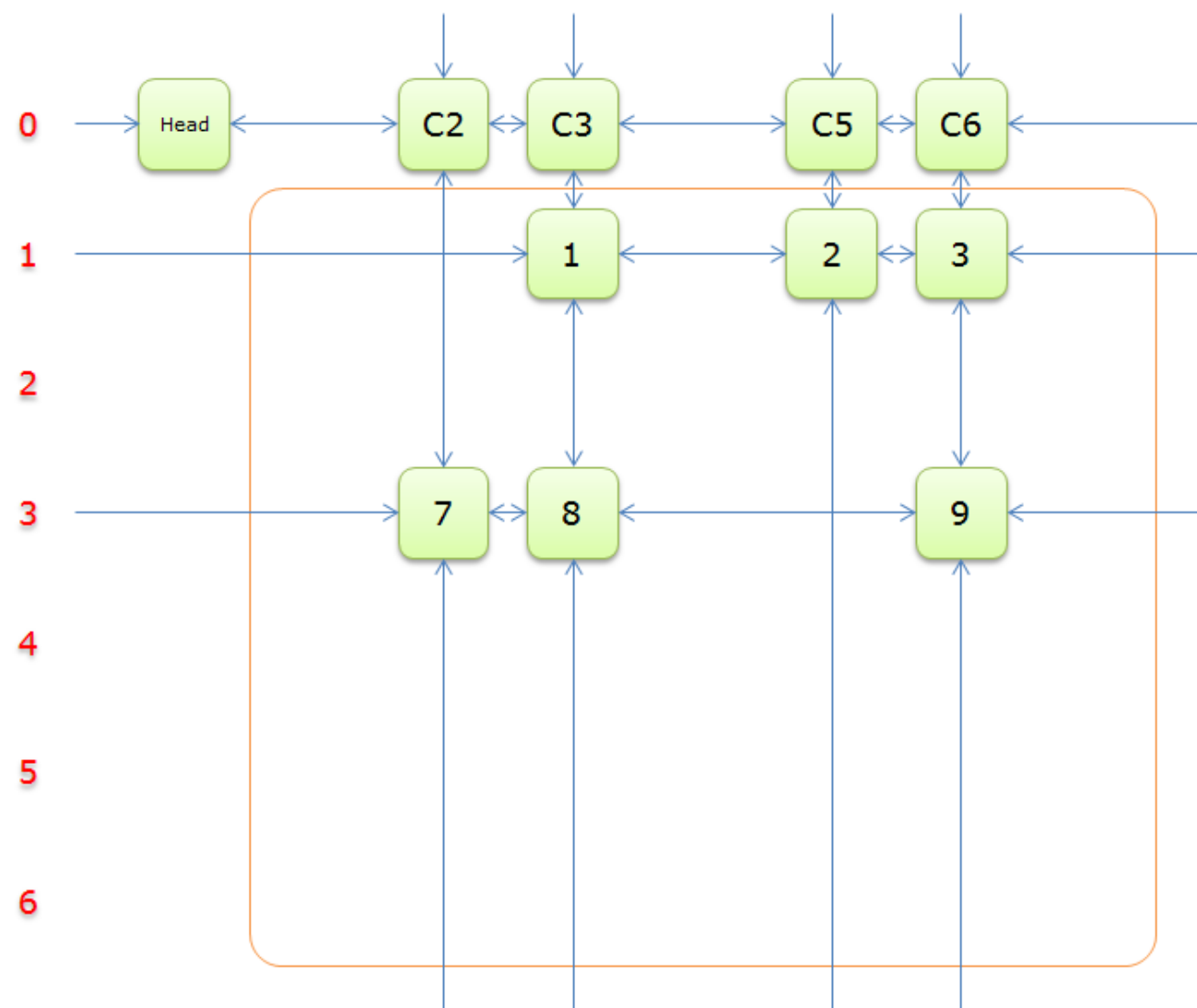
- 1. 满足第1列
- 选择第2行 (如果不行再回来选择第3行)



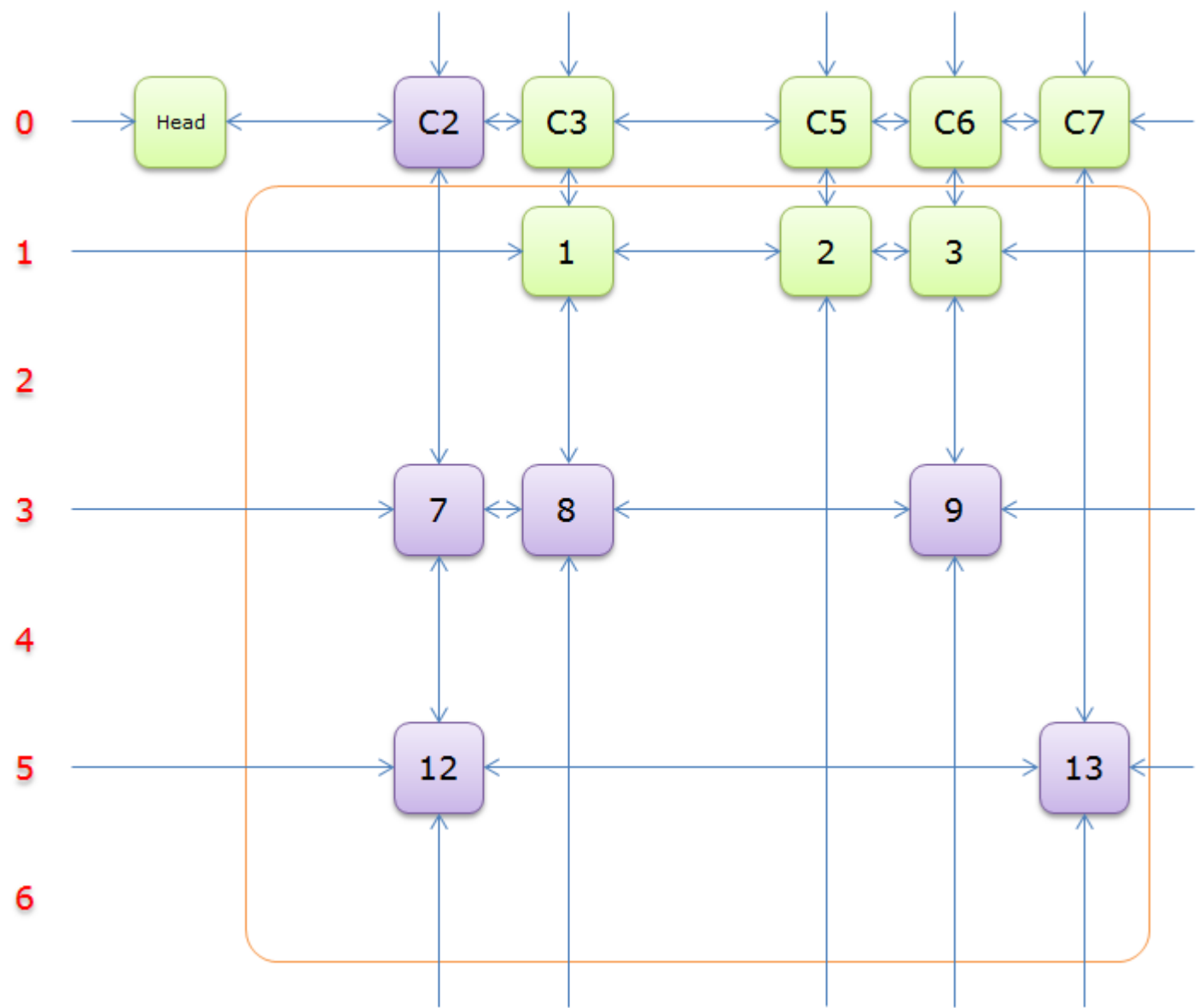
- 2. 选择了第2行的时候
 - C4, C7都被满足了 (标记下次不再需要满足, 橙色)
 - C4, C7对应有1的行 (5, 6) 都不能要了
 - 第4行当然也不能要了
 - 不能要的行
 - 在**列链表**中删去连接 (15)
 - 方法: 找到C4
 - C4找到14 (不用删)
 - 14找到对应的行
 - 找到15, 让2直接连到C5
 - 回溯: 加回来C4
 - C4找到14, 找回15
 - 15还连着2和C5, 重建连接



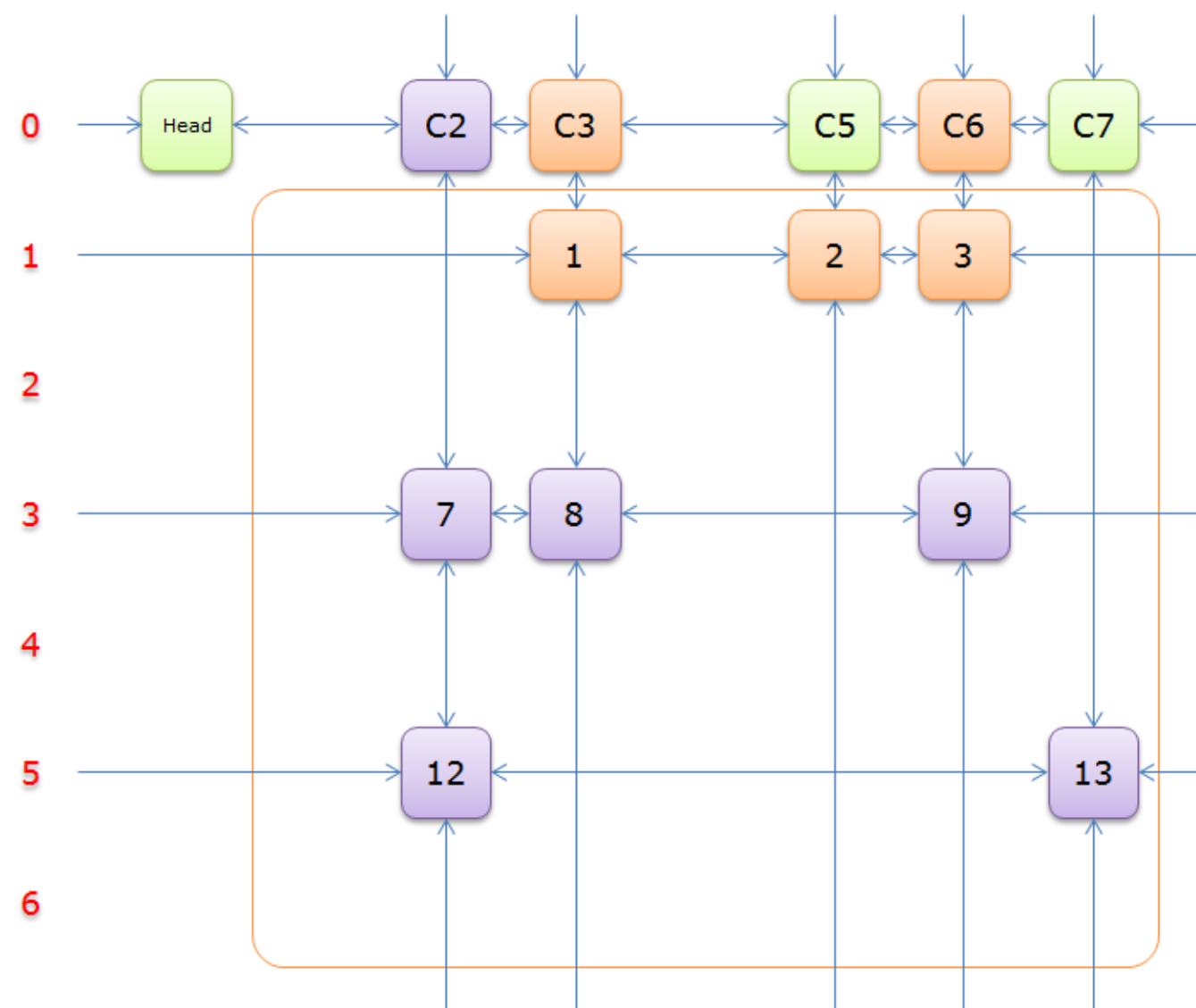
- 删除后目前的情况，递归求解
 - 对于C2来说，12去哪里了？
 - 12只是用C2找不到了
 - 12用第5行还是找得到的
 - 上一层的栈知道我们消去了第【2, 4】【5, 6】行，想要回溯的时候再按照行恢复原状即可。
 - C7怎么找回？
 - C7只是被标记了不要的，其链接信息还是在的
 - 15号块怎么找回？
 - 上一层栈知道我们消去了行6，可以找回



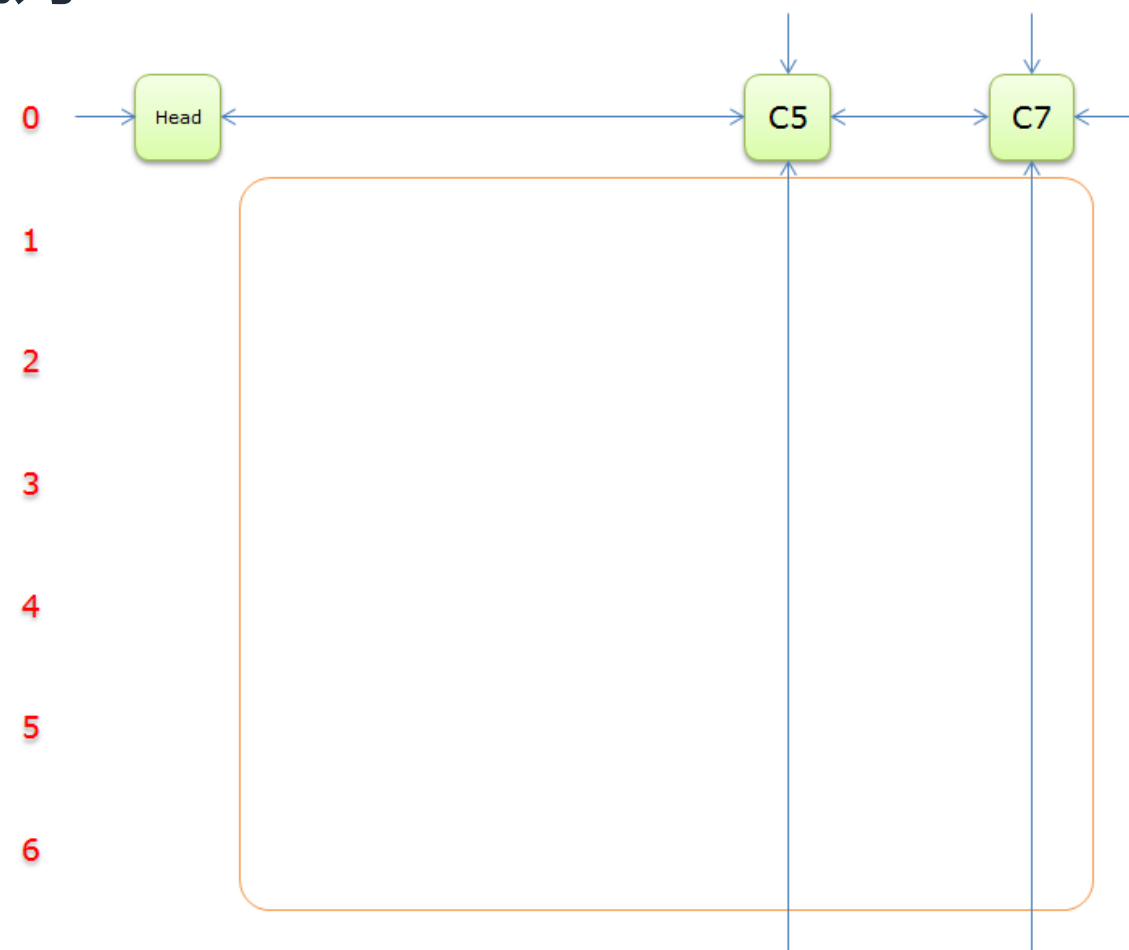
- 下一步，满足C2



- 满足C2会引起8, 9对应的列(C3,C6)被标记为橙色 (不需要)
- 同时第5行也会被移除
- 因为1不能要, 所以一整行都不能要了



- 还有2列无法满足！回溯！重新选择。
 - 刚才最后一刻删了行1，把它恢复出来，3重新指回C6
 - 刚才解决的是C6 C3，把它们标记回来
 - 刚才选择了第3行，现在把它取消选择
 - 第三行第一个是7，下面是12，把12这一行恢复回来
 - 总之按照你取消的顺序再退回来就好
- 高效实现稀疏状态下的行列操作！
 - 不用每次都for整个行整个列



```

hog = cv2.HOGDescriptor(
    (28,28), (14,14), (7,7), (7,7), 9, 1, 4., 0, 2, 0, 64
)
ftrain = []
for item in train_data:
    ftrain.append(hog.compute(item))
train_data = np.array(ftrain)

ftest = []
for item in test_data:
    ftest.append(hog.compute(item))
test_data = np.array(ftest)

clf = svm.LinearSVC()
X = train_data.reshape((60000, -1))
Y = train_label
clf.fit(X, Y)
x = test_data.reshape((10000, -1))
y = test_label
print('Linear SVM score %.3f' % (clf.score(x, y))) # 0.986

```


• OpenCV is a C++ Library

Implementation of HOG (Histogram of Oriented Gradients) descriptor and object detector. [More...](#)

```
#include "objdetect.hpp"
```

Public Types

```
enum { L2Hys = 0 }
```

```
enum { DEFAULT_NLEVELS = 64 }
```

Public Member Functions

```
HOGDescriptor ()
```

Creates the HOG descriptor and detector with default params. [More...](#)

```
HOGDescriptor (Size _winSize, Size _blockSize, Size _blockStride, Size _cellSize, int _nbins, int _derivAperture=1, double _winSigma=-1, int  
_histogramNormType=HOGDescriptor::L2Hys, double _L2HysThreshold=0.2, bool _gammaCorrection=false, int  
_nlevels=HOGDescriptor::DEFAULT_NLEVELS, bool _signedGradient=false)
```

```
HOGDescriptor (const String &filename)
```

```
HOGDescriptor (const HOGDescriptor &d)
```

```
virtual ~HOGDescriptor ()
```

Default destructor. [More...](#)

```
bool checkDetectorSize () const
```

Checks if detector size equal to descriptor size. [More...](#)

```

hog = cv2.HOGDescriptor(
    (28,28), (14,14), (7,7), (7,7), 9, 1, 4., 0, 2, 0, 64
)
ftrain = []
for item in train_data:
    ftrain.append(hog.compute(item))
train_data = np.array(ftrain)

ftest = []
for item in test_data:
    ftest.append(hog.compute(item))
test_data = np.array(ftest)

clf = svm.LinearSVC()
X = train_data.reshape((60000, -1))
Y = train_label
clf.fit(X, Y)
x = test_data.reshape((10000, -1))
y = test_label
print('Linear SVM score %.3f' % (clf.score(x, y))) # 0.986

```

- 使用人工神经网络进行MNIST训练和测试
 - 1. 使用与 Assignment #2 相同的数据
 - 2. 基本要求：
 - a. 使用三层全连接层组成的网络
 - b. 使用包含卷积的神经网络
 - 3. 提交：
 - 代码
 - 报告（包含网络结构和测试结果）
 - 4. 时间
 - 三周，截止时间：4月25日

