# MIS
# (Make It Sudoku)

20145139 신지훈
20145157 이현수
20155341 조하림

# INTRODUCTION

# PROJECT 목적

- 스도쿠는 미국의 건축가 ‘하워드 간즈’ 가 ‘레온하르트 오일러’의 라틴방진을 기초로 하여 ‘넘버 플레이스라’는 이름으로 만든 수학 퍼즐이다

- 스도쿠는 과거부터 현재까지 감정적으로 성취감을, 신체적으로는 두뇌 자극과 집중력 향상에 도움을 주어 대중들에게 많은 사랑을 받는 게임이다.

- 하지만 이러한 장점과는 다르게 오히려 스도쿠 문제를 풀지 못해서 궁금증과 답답함을 느끼는 대중들도 있을 것이다.

- 우리 팀은 이러한 문제를 해결하고 문제를 openCV를 통한 문제를 이용하여 스도쿠 문제를 자동 입력하고 문제를 풀어주는 프로그램을 개발하려고 한다.

# WHY? OPENCV, GENETIC ALGORITHM

- openCV는 1999년도 인텔이 시작한 프로젝트로써 영상과 같은 실시간 이미지 프로세싱에 중점을 둔 오픈소스 라이브러리다.

- 본 프로젝트는 손글씨로 그린 스도쿠를 인식하기 위해 openCV를 사용함.

- 유전 알고리즘은 자연세계의 진화과정에 기초한 계산 모델로서 존 홀랜드(John Holland)에 의해서 1975년에 개발된 전역 최적화 기법으로, 최적화 문제를 해결하는 기법의 하나이다.

- 다른 알고리즘은 순차적으로 대입하는 것에 비해 유전알고리즘은 적합도가 높은 것을 찾아가기 때문에 알고리즘으로써 좀더 빠른 속도를 기대함.

문제 입력

OpenCV를 이용해
이미지속 스도쿠 문제를
인식함

문제 풀이

유전 알고리즘을 사용하여
인식된 스도쿠 문제를 풀이

# OPENCV

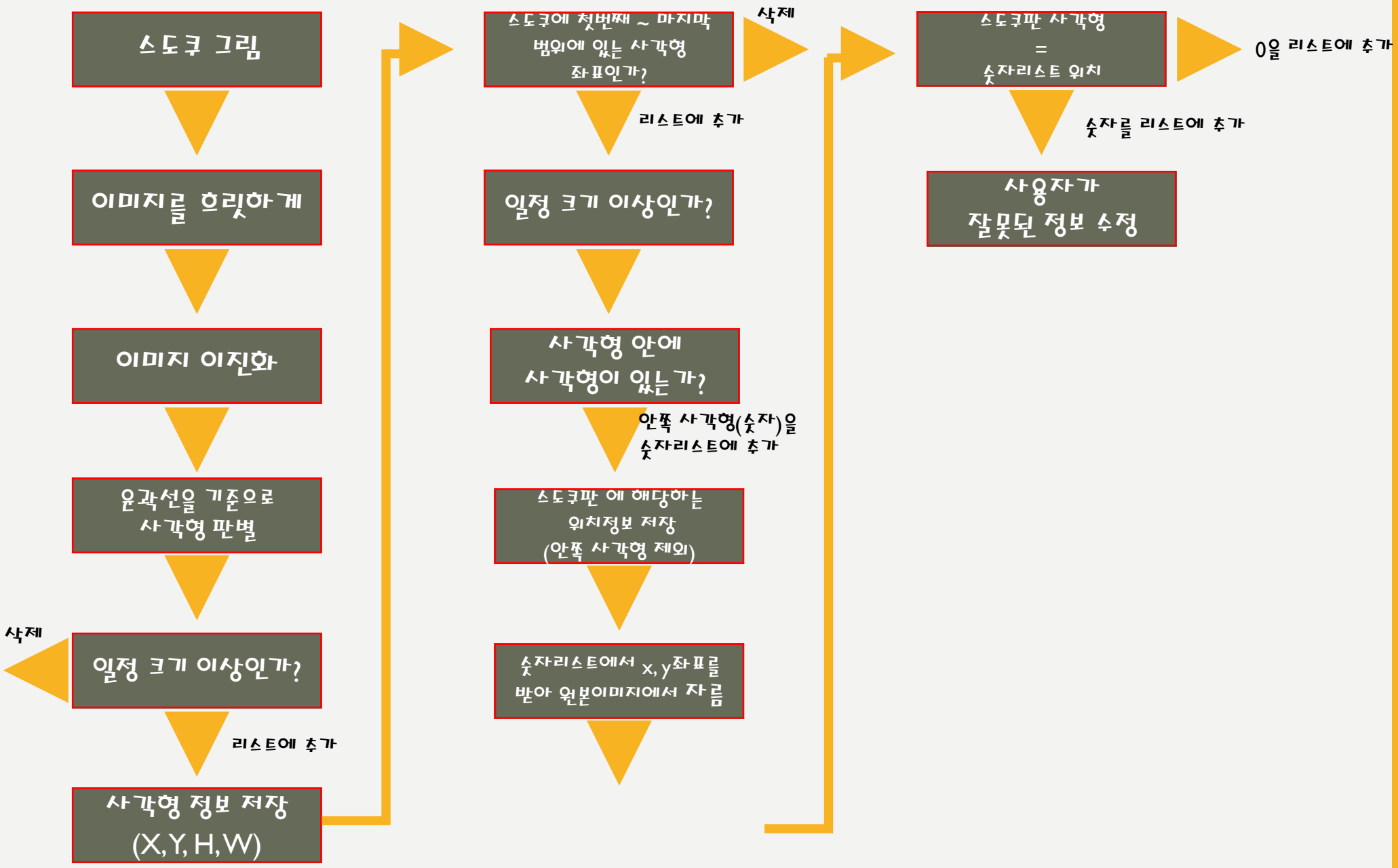
문제 입력

OpenCV를 이용해
이미지속 스도쿠 문제를
인식함

문제 풀이

유전 알고리즘을 사용하여
인식된 스도쿠 문제를 풀이

# openCV 순서도


스도쿠 그림
이미지를 흐릿하게
이미지 이진화
윤곽선을 기준으로 사각형 판별
일정 크기 이상인가?
삭제
리스트에 추가
사각형 정보 저장 (X,Y,H,W)
스도쿠에 첫번째 ~ 마지막 범위에 있는 사각형 좌표인가?
삭제
리스트에 추가
일정 크기 이상인가?
사각형 안에 사각형이 있는가?
안쪽 사각형(숫자)을 숫자리스트에 추가
스도쿠판 에 해당하는 위치정보 저장 (안쪽 사각형 제외)
숫자리스트에서 x, y좌표를 받아 원본이미지에서 자름
스도쿠판 사각형 = 숫자리스트 위치
0을 리스트에 추가
숫자를 리스트에 추가
사용자가 잘못된 정보 수정

```python
def isRectangleinRectangle(possible_contours):
    possible_digit=[]
    cnt=0
    for d1 in possible_contours:
        for d2 in possible_contours:
            if  d2['x']<d1['x']<d2['x']+d2['w'] and d2['y']<d1['y']<d2['y']+d2['h'] and d2['x']<d1['x']+d1['w']<d2['
                d1['idx'] = cnt
                cnt += 1
                possible_digit.append(d1)
                break
    return possible_digit
```

해당 가로,세로 좌표 중에 x,y가 있는지 검사 검사

```
def isRectangle(contours_dict):
    impossible=[]
    cnt=0
    for d1 in contours_dict:
        for d2 in contours_dict:
            if(d2['w']-(d2['w']/2)<d1['w']<d2['w']+(d2['w']/2) or d2['h']-(d2['h']/2)<d1['h']<d2['h']+(d2['h']/2)):
                cnt+=1
        if(cnt<4 or d1['x']==0 or d1['y']==0):
            impossible.append(d1)
        cnt=0
    return impossible
```

해당 크기와 비슷한 사각형이 있으면 cnt+1

x,y 가 0이거나 비슷한 개수가 4개 이하이면 불가능

```python
def deleteRectangleInRectangle(contours_dict,impossible):
    possible_contours=[]
    out=False
    for d1 in contours_dict:
        for d2 in impossible:
            if(d1['x']==d2['x'] and d1['y']==d2['y']):
                out=True
        if(out==False):
            possible_contours.append(d1)
        out=False
    return possible_contours
```

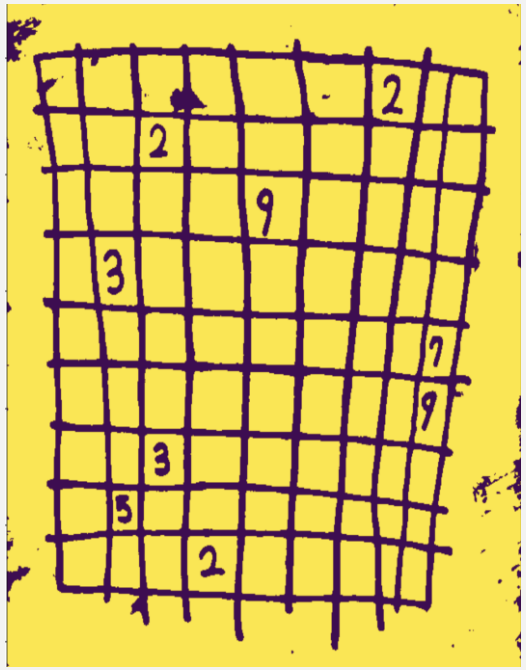
사각형 리스트를 지우는 함수

```
img=cv2.imread('F_test.jpeg')
height,width,channel=img.shape
gray=cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
img_blurred=cv2.GaussianBlur(gray,ksize=(21,21),sigmaX=0)
img_thresh=cv2.adaptiveThreshold(
    img_blurred,
    maxValue=255,
    adaptiveMethod=cv2.THRESH_BINARY,
    thresholdType=cv2.THRESH_BINARY,
    blockSize=211,
    C=2
)
plt.figure(figsize=(15,16))
plt.imshow(img_thresh)
```

파일을 읽어들임

이미지를 이진화함

이미지를 흐릿하게 하는 함수



윤곽선을 찾음

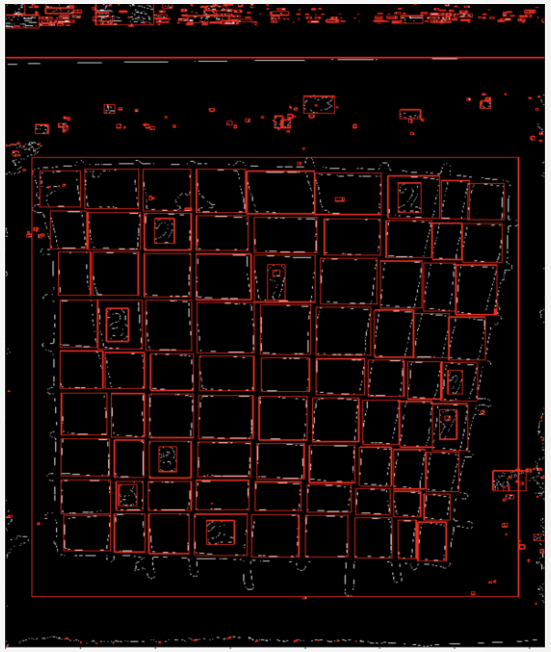
```
contours,_=cv2.findContours(img_thresh,cv2.RETR_LIST,cv2.CHAIN_APPROX_SIMPLE)
temp_result = np.zeros((height, width, channel), dtype=np.uint8)
cv2.drawContours(temp_result,contours,-1,(255,255,255))
plt.figure(figsize=(18,16))
plt.imshow(temp_result,cmap='gray')
len(contours)
```

이미지 출력

```python
contours_dict=[]
cnt=0
temp_result=np.zeros((height,width,channel),dtype=np.uint8)
cv2.drawContours(temp_result,contours,-1,(255,255,255))
for contour in contours:
    x,y,w,h=cv2.boundingRect(contour)
    cv2.rectangle(temp_result,(x,y),(x+w,y+h),(255,0,0),2)
    if w>(width/500) and h>(height/50):
        contours_dict.append({
            'contour':contour,
            'x':x,
            'y':y,
            'w':w,
            'h':h,
            'cx':x+(w/2),
            'cy':y+(h/2)
        })
plt.figure(figsize=(100,160))
plt.imshow(temp_result,cmap='gray')
```

잡음 제거

사각형의 x,y,w,h,중앙 위치값 저장



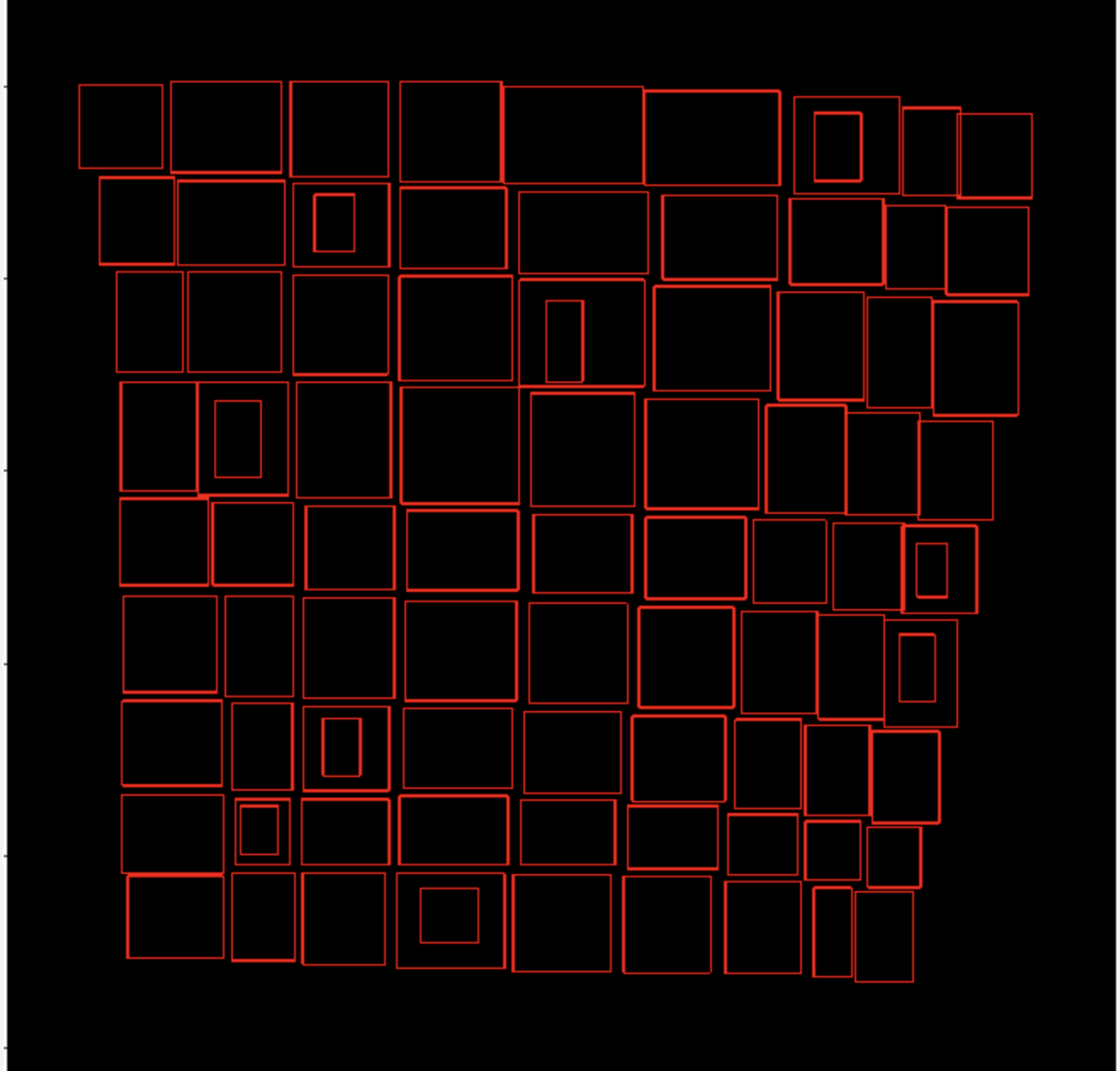
```
temp=[]
temp_result=[]
allcontours=[]
for d1 in contours_dict:
    for d2 in contours_dict:
        if(d2['w']-(d2['w']/4)<d1['w']<d2['w']+(d2['w']/4) and d2['h']-(d2['h']/2)<d1['h']<d2['h']+(d2['h']/2)):
            cnt+=1
        if cnt>40:
            temp.append(d1)
    cnt=0
pointH1=0
pointH2=0
largeX=0
largeY=0
smallX=temp[0]['x']
smallY=temp[0]['y']
for d1 in temp:
    if(d1['x']>largeX):
        largeX=d1['x']
    if(d1['y']>largeY):
        largeY=d1['y']
        pointH2=d1['h']
    if(d1['x']<smallX):
        smallX=d1['x']
    if(d1['y']<smallY):
        pointH1=d1['h']
        smallY=d1['y']
for d1 in contours_dict:
    if(smallX-int(pointH2/2)<=d1['x']<=largeX+int(pointH2/2) and (smallY-int(pointH1/5))<=d1['y']<=largeY+(pointH2/5)):
        allcontours.append(d1)

temp_result=np.zeros((height,width,channel),dtype=np.uint8)
#cv2.rectangle(temp_result, pt1=(smallX,(smallY-int(pointH1/5))), pt2=(largeX+int(pointH2/2),int(largeY+(pointH2/2))), color=(0, 0, 25.
for d in allcontours:
    cv2.rectangle(temp_result, pt1=(d['x'], d['y']), pt2=(d['x']+d['w'], d['y']+d['h']), color=(255, 0, 0), thickness=2)

plt.figure(figsize=(100, 60))
plt.imshow(temp_result, cmap='gray')
```

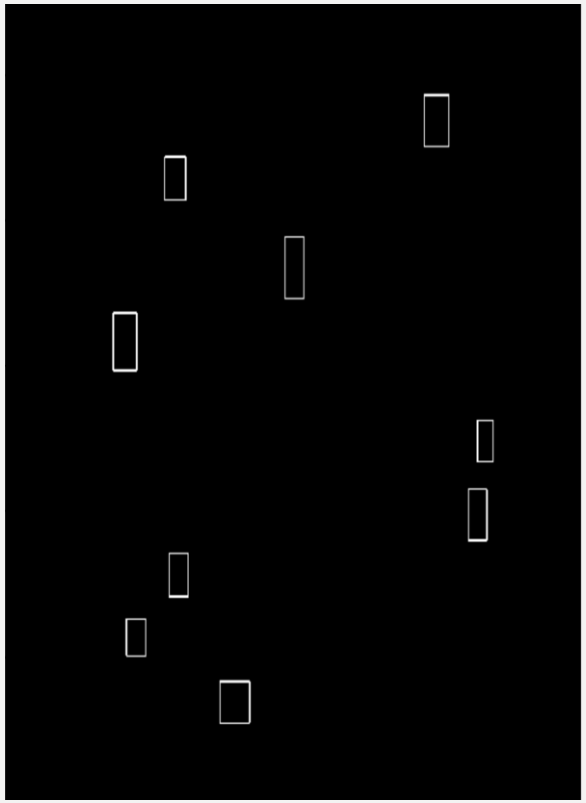
스도쿠 사각형의 개수는 총 81개 하지만
사람이 손으로 그리므로 40개가 있으면 스도쿠
사각형으로 인정

첫번째와 마지막 x,y 추출 그 안에 있는 x,y는
스도쿠로 인정!



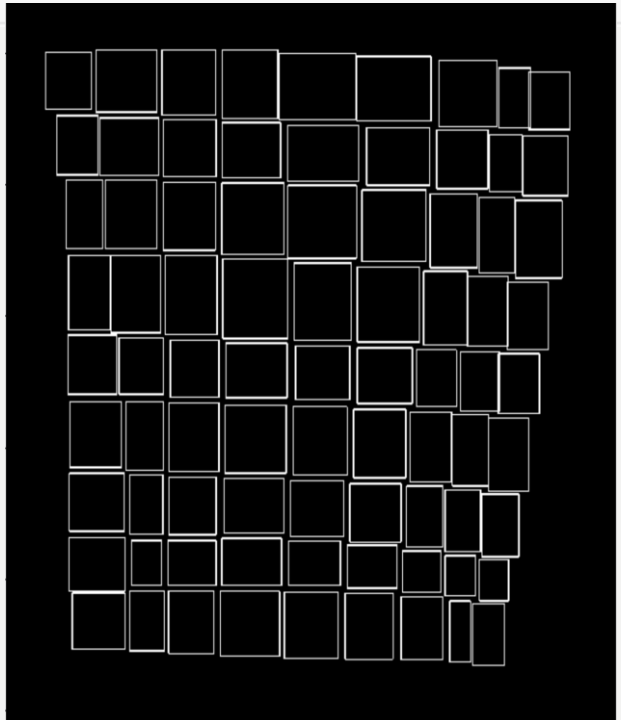
```
digit=isRectangleinRectangle(allcontours)
digit=deleteRectangleInRectangle(digit,isRectangleinRectangle(digit))
temp_result = np.zeros((height, width, channel), dtype=np.uint8)

for d in digit:
    cv2.rectangle(temp_result, pt1=(d['x'], d['y']), pt2=(d['x']+d['w'], d['y']+d['h']), color=(255, 255, 255), thickness=2)
plt.figure(figsize=(80, 60))
plt.imshow(temp_result, cmap='gray')
```



사각형 안에 사각형이 있는지 확인
있으면 바깥쪽 사각형 제거

```
#사각형 추출
allcontours=deleteRectangleInRectangle(allcontours,digit)
temp_result = np.zeros((height, width, channel), dtype=np.uint8)
for d in allcontours:
    cv2.rectangle(temp_result, pt1=(d['x'], d['y']), pt2=(d['x']+d['w'], d['y']+d['h']), color=(255, 255, 255), thic
plt.figure(figsize=(15, 16))
plt.imshow(temp_result, cmap='gray')
```



안쪽 사각형 제거 함수

```
listarr=[]
digit=sortRectangle(digit)
for i in range(len(digit)):
    img_cropped=cv2.getRectSubPix(
        img,
        patchSize=(int(digit[i]['w']),int(digit[i]['h'])),
        center=(int(digit[i]['cx']),int(digit[i]['cy']))
    )
    char=pytesseract.image_to_string(img_cropped,lang='kor',config='--psm 7 --oem 0 -c tessedit_char_whitelist=12345
    if(char!='' and 0<int(char[0])<10):
        listarr.append(char[0])
    else:
        listarr.append(-1)
```

숫자 정렬 현재 리스트에는 숫자 이미지가 순서대로 정렬 되어있지 않음

이미지를 자르는 함수

이미지를 인식하여 숫자로 변환하는 함수

```
listAnswer=[]
isblank=True
cnt=0
allcontours=sortRectangle(allcontours)


for d1 in allcontours:
    for d2 in digit:
        if  d1['x']<d2['x']<d1['x']+d1['w'] and d1['y']<d2['y']<d1['y']+d1['h']:
            listAnswer.append(int(listarr[cnt]))
            cnt=cnt+1
            isblank=False
    if(isblank==True):
        listAnswer.append(0)
    isblank=True
```

```
row=[]
sudoku=[]
for i in range(9):
    for j in range(i*9,(i*9)+9):
        row.append(listAnswer[j])
    sudoku.append(row)
    row=[]
```

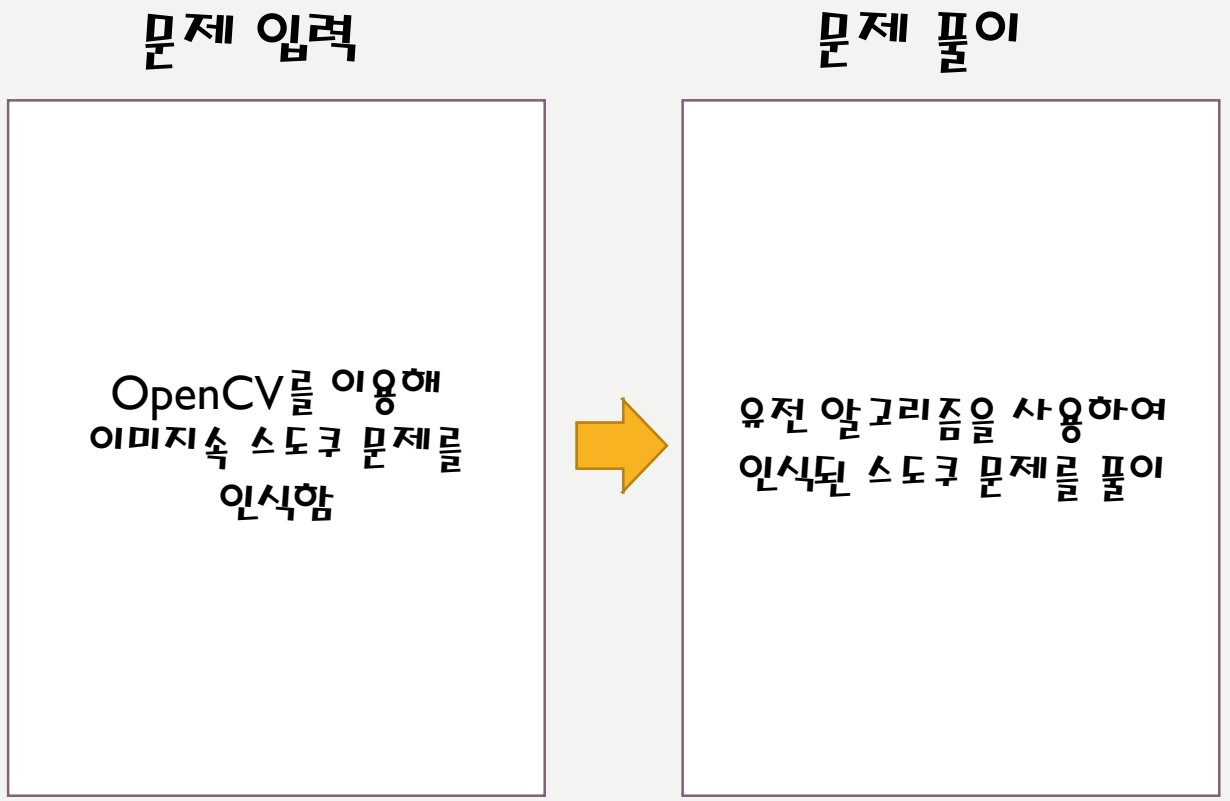
9*9 리스트를 만듦.

숫자가 있는 사각형이면 해당 위치에 숫자를 넣음

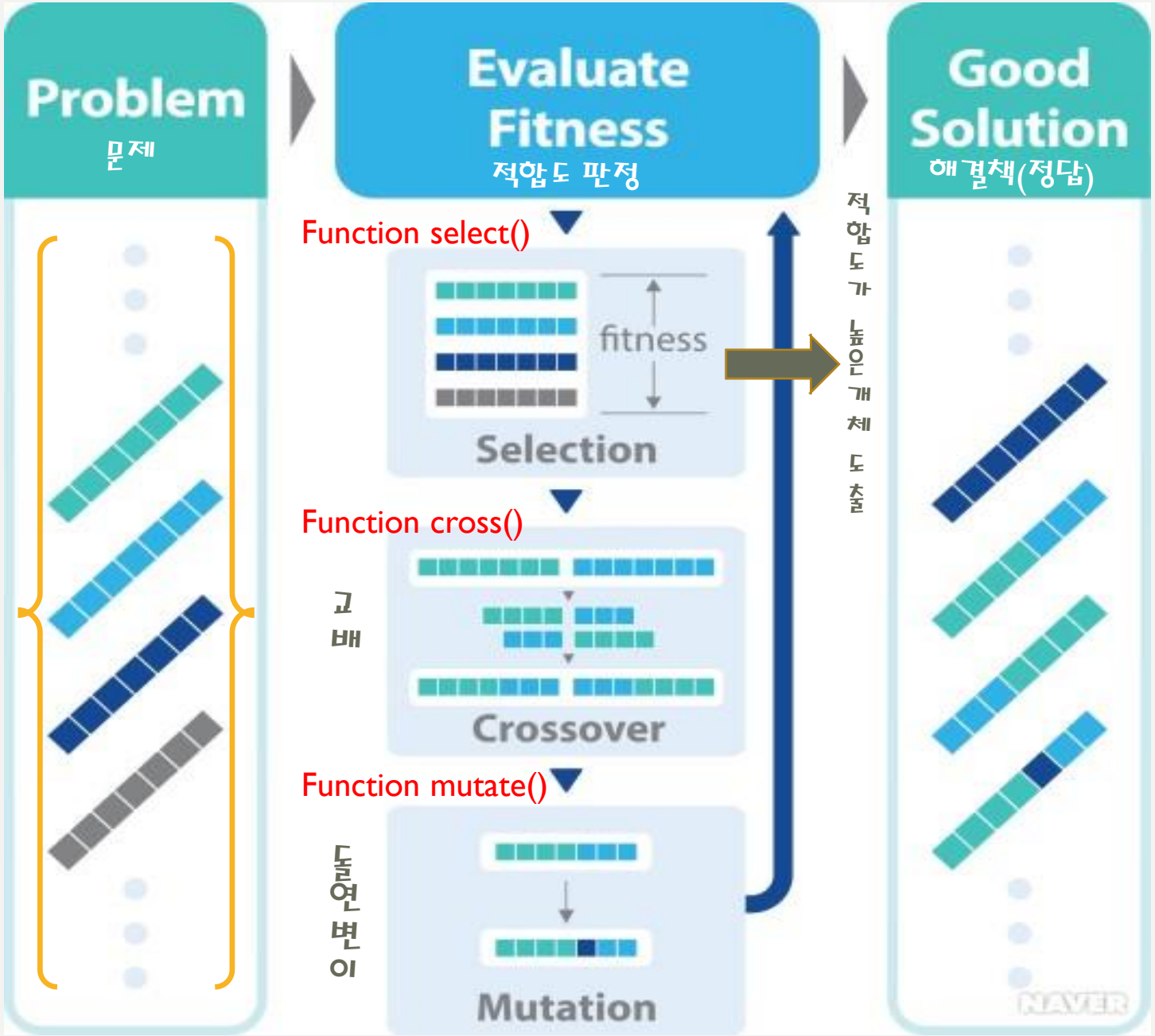
```
print("잘못 인식한 부분을 수정해 주세요!! 종료:Q")
for i in range(9):
    print("")
    for j in range(9):
        print(sudoku[i][j],",",end='')
while(True):
    column=input("행입력 종료 Q: ")
    if(column=='Q' or column=='q'):
        break
    else:
        column=int(column)
        row=int(input("열입력: "))
        value=int(input("숫자입력: "))
        sudoku[column-1][row-1]=value
    print("\n==============")
    for i in range(9):
        print("")
        for j in range(9):
            print(sudoku[i][j],",",end='')
solve(sudoku)
```

행 수정 및 스도쿠 문제 풀이

# GENETIC ALGORITHM

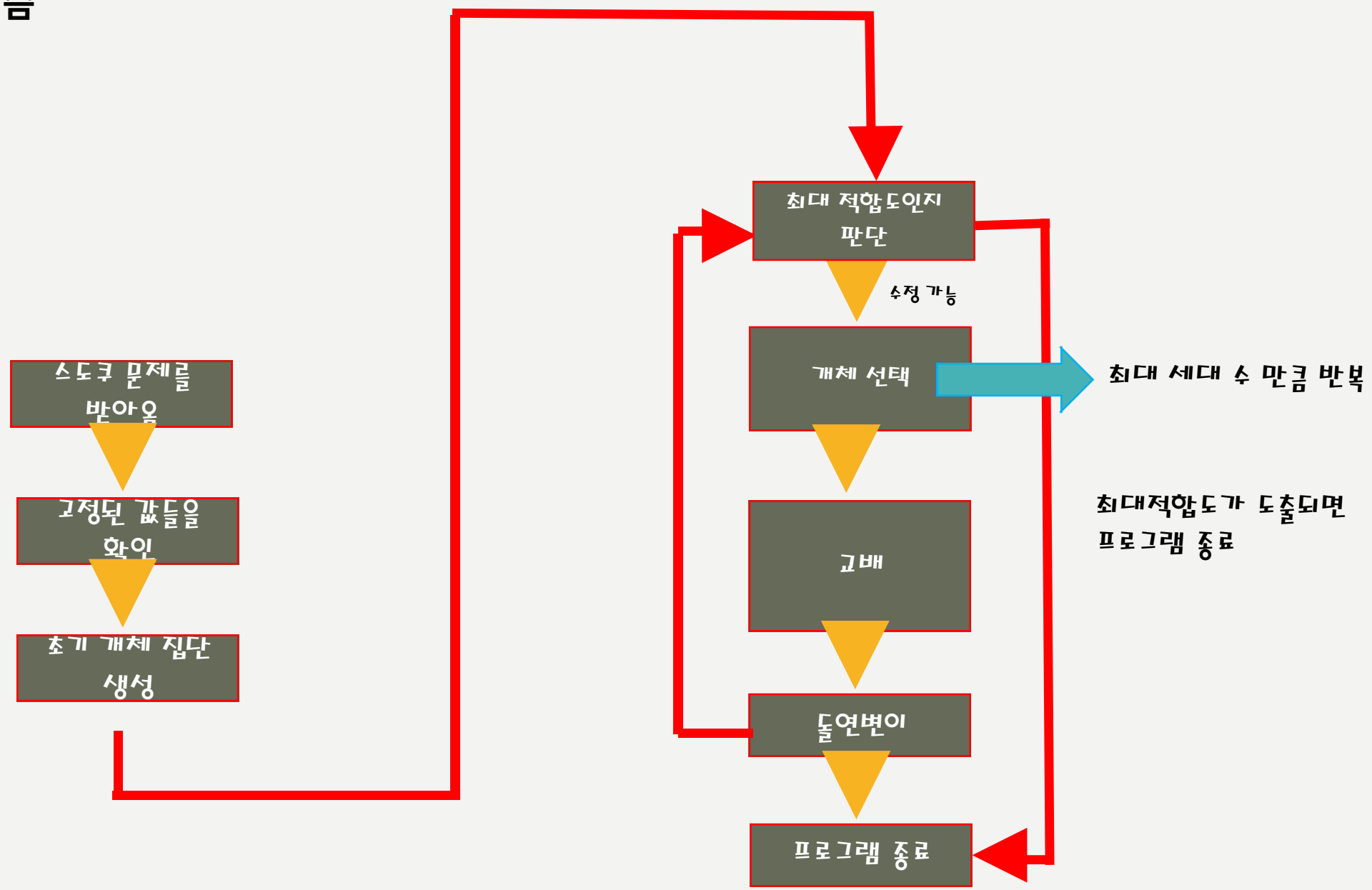

문제 입력
OpenCV를 이용해
이미지속 스도쿠 문제를
인식함
문제 풀이
유전 알고리즘을 사용하여
인식된 스도쿠 문제를 풀이

Function genetic()




다양한 개체 생성

# 유전알고리즘 순서도


스도쿠 문제를 받아옴
고정된 값들을 확인
초기 개체 집단 생성
최대 적합도인지 판단
수정 가능
개체 선택
교배
돌연변이
프로그램 종료
최대 세대 수 만큼 반복
최대적합도가 도출되면 프로그램 종료

# 실행문

```
if __name__ == "__main__":

    popul = 10
    selection = 0.5
    mutation = 0.05
    max_gen = 300000

    ready()
    genetic(popul,selection,mutation,max_gen)
    print_highest()
```

스도쿠 문제안에 먼저 채워져 있는 숫자들의 위치를 기억함

```
#ready for problem to put fixed values.
def ready():
    global value
    global fixed
    for i in range (9):
        for j in range (9):
            if(value[i][j] != 0):
                fixed[i][j] = True
            else:
                fixed[i][j] = False
```

# 적합도를 계산해서 반환하는 함수

```
def get_fitness(value):
    fitness = 0
    for i in range(9):
        for j in range(9):
            fitness += copy.deepcopy(checkElementScore(value,i,j))
    return fitness
```

```python
def genetic(popul,selection,mutation,max_gen):
    global values
    global value
    global high_fitness
    global times
    global gens
    global n
    global start
    fitness_values = []
    start = t.time()
    gen = 0
    #First Generation
    for i in range(popul):
        fitness_values.append(copy.deepcopy(0))
        values.append(copy.deepcopy(put_element(copy.deepcopy(value))))
    for i in range(popul):
        fitness_values[i] = copy.deepcopy(get_fitness(copy.deepcopy(values[i])))
    #After first generation. Loop
    while(gen != max_gen):
        #answerFound
        if(high_fitness==243):

            time = t.time()-start
            print(time)
            times += time
            print(gen+1," Generation")
            print("Complete!")
            gens += gen
            n += 1
            return
        #getFitness
        for i in range(popul):
            fitness_values[i] = copy.deepcopy(get_fitness(copy.deepcopy(values[i])))
        #select
        select(copy.deepcopy(selection),popul,copy.deepcopy(fitness_values))
        #cross
        cross(popul,selection)
        #mutate
        for i in range(popul):
            values[i] = copy.deepcopy(mutate(values[i],mutation))

        gen+= 1
    if(high_fitness!=243):
        print("Cannot find the answer.")
```

적합도 저장할 공간을 초기화

초기 개체들을 생성

최대 세대 수 만큼 반복

정답이 도출되면 함수 종료

개체들의 적합도를 저장

개체들의 적합도 저장

적합도가 높은 개체 선정 단계

돌연변이 단계

주어진 세대 수 동안 반복해도 답을 찾지 못함

```
def select(selection,popul,fitness_values):
    global values
    global high_fitness
    global high_value
    selected=int(popul*selection)
    insf = 0
    insv = values[0]
    for i in range(popul): 적합도가 높은 개체를 리스트 앞쪽으로 이동시킴
        for j in  range(popul):
            if(fitness_values[i] < fitness_values[j]):
                insf = copy.deepcopy(fitness_values[j])
                fitness_values[j] = copy.deepcopy(fitness_values[i])
                fitness_values[j] = copy.deepcopy(insf)
                insv = copy.deepcopy(values[j])
                values[j] = copy.deepcopy(values[i])
                values[j] = copy.deepcopy(insv)
    for i in range(popul - selected): 선정되지 못한 개수만큼 저장된 리스트에서 제외시킴
        values.pop()
    if(high_fitness < fitness_values[0]):
        high_value = copy.deepcopy(values[0])
        high_fitness = copy.deepcopy(fitness_values[0])
```

```
def cross(popul,selection):
    global values
    selected = int(popul*selection)
    inst = []
    while(len(inst) != popul):
        valueo = copy.deepcopy(values[random.randint(0,len(values)-1)])
        valuet = copy.deepcopy(values[random.randint(0,len(values)-1)])
        for i in range(9):
            for j in range(9):
                ins = copy.deepcopy(valueo[i][j])
                valueo[i][j] = copy.deepcopy(valuet[i][j])
                if(not checkElementFlag(valueo,i,j)):
                    valueo[i][j] = copy.deepcopy(ins)
        inst.append(copy.deepcopy(valueo))
    values = copy.deepcopy(inst)
```

정해진 개체 수만큼 교배해서 새로운 개체들을 만듦

상위 적합도를 가진 개체 중 두 개체를 가져옴

두 개체를 비교해서 교체할 수 있는 값이 있으면 교체함

새로 교배 된 개체들을 저장함

```
def mutate(value,mutation):
    mut_val = int(mutation*81)
    blank = 0
    same = {}
    while(mut_val != 0):
        row = copy.deepcopy(random.randint(0,80))
        column = copy.deepcopy(random.randint(0,8))
        if(same.get(row) != column and not(fixed[row%9][column])):
            if(len(getposElement(value,row%9,column)) > 1 ):
                value[row%9][column] = copy.deepcopy(getposElement(value,row%9,column)[copy.deepcopy(random.randrange(0,len(getposElement(value,row%9,c
                same[row] = column
                mut_val -= 1
            else:
                value[row%9][column] = copy.deepcopy(0)
                same[row] = column
                mut_val -= 1
                blank += 1
    for i in range(9):
        for j in range(9):
            if(value[i][j]==0):
                if(len(getposElement(value,i,j)) > 0 ):
                    value[i][j] = copy.deepcopy(getposElement(value,i,j)[copy.deepcopy(random.randrange(0,len(getposElement(value,i,j)),1))])
                    blank -= 1
            if(blank == 0):
                return value
    return value
```

돌연변이 시켜야 하는 칸의 개수

위치를 랜덤으로 받아 옴

현재 값 외에 변경가능한 값이 있다면 값을 변경함

한번 접근한 위치를 표시

현재 값 외에 변경가능한 값이 없다면 빈칸(0) 으로 변경

빈칸으로 변경된 만큼 카운트를 함

빈칸으로 변경된 만큼 숫자를 다시 넣어 줌

```python
def checkElementFlag(value,i,j):
    rowDif = i%3
    colDif = j%3
    flag = True
    #Check column element
    for column in range (9):
        if(value[i][j]==value[i][column]):
            if(column != j):
                flag = False
    #Check row element
    for row in range (9):
        if(value[i][j]==value[row][j]):
            if(row != i):
                flag = False
    #Check square element
    for row in range (3):
        for column in range(3):
            if(value[i][j]==value[i-rowDif+row][j-colDif+column]):
                if(i-rowDif+row != i and j-colDif+column != j):
                    flag = False
    return flag
```

해당 위치에 조건에 맞는 값이 들어갔는지 확인하는 함수

행에 자신과 같은 숫자가 있는지 확인함

열에 자신과 같은 숫자가 있는지 확인함

자신이 속한 구역에 자신과 같은 숫자가 있는지 확인함

```python
def checkElementScore(value,i,j):
    if(value[i][j] == 0):
        return 0
    rowDif = i%3
    colDif = j%3
    score = 0
    flag = True
    #Check column element
    for column in range (9):
        if(value[i][j]==value[i][column]):
            if(column != j):
                flag = False
    if(flag):
        score += 1
    flag = True
    #Check row element
    for row in range (9):
        if(value[i][j]==value[row][j]):
            if(row != i):
                flag = False
    if(flag):
        score += 1
    flag = True
    #Check square element
    for row in range (3):
        for column in range(3):
            if(value[i][j]==value[i-rowDif+row][j-colDif+column]):
                if(i-rowDif+row != i and j-colDif+column != j):
                    flag = False
    if(flag):
        score += 1
    flag = True
    return score
```

행에 자신과 같은 숫자가 있는지 확인함
같은 숫자가 없으면 1점

열에 자신과 같은 숫자가 있는지 확인함
같은 숫자가 없으면 1점

자신이 속한 구역에 자신과 같은 숫자가 있는지 확인함
같은 숫자가 없으면 1점

최고 3점, 최저 0점이 반환됨

```python
def getposElement(value,i,j):
    posv = []
    for k in range(1,10,1):
        value[i][j] = k
        if(checkElementFlag(value,i,j)):
            posv.append(k)
        value[i][j] = 0
    return posv


def put_element(value):
    cnt = 81
    same = {}

    while(cnt != 0):
        row = copy.deepcopy(random.randint(0,8))
        column = copy.deepcopy(random.randint(0,8))
        if(same.get(row) != column and not(fixed[row][column])):
            ins = copy.deepcopy(value[row][column])
            value[row][column] = copy.deepcopy(random.randint(1,9))
            if(not checkElementFlag(value,row,column)):
                value[row][column] = copy.deepcopy(ins)
                same[row] = column
            else:
                same[row] = column
                cnt -= 1
    return value
```

현재위치에 삽입 가능한 숫자가 있는지 확인하고 숫자들을 반환함

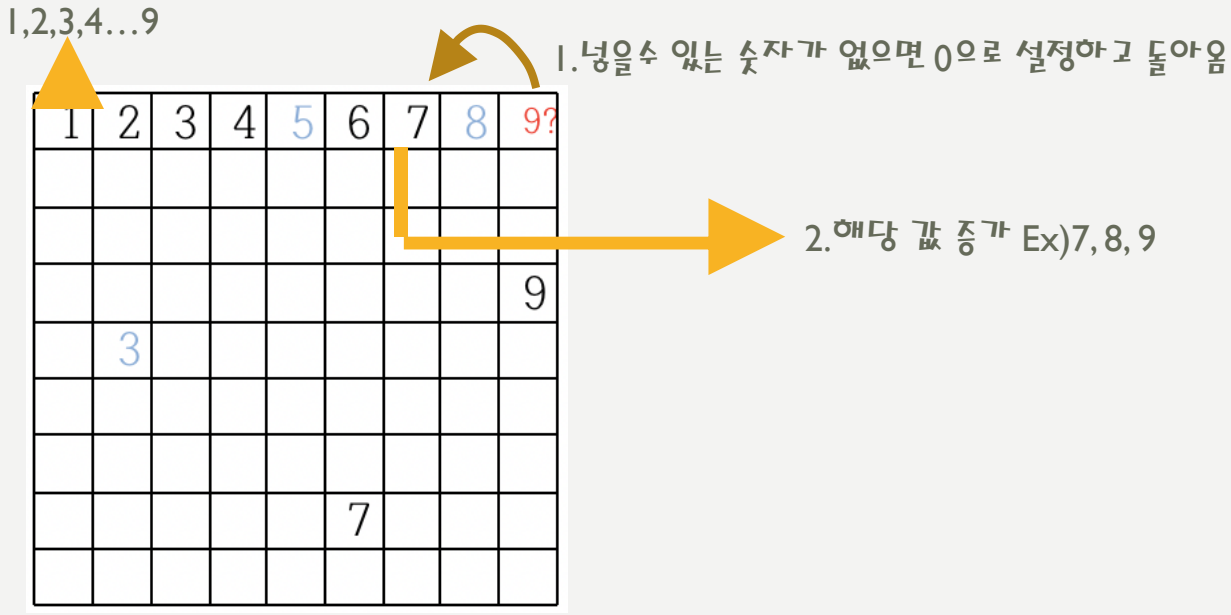
초기 개체 생성 함수

# BACKTRACKING

# 백트레킹 알고리즘?

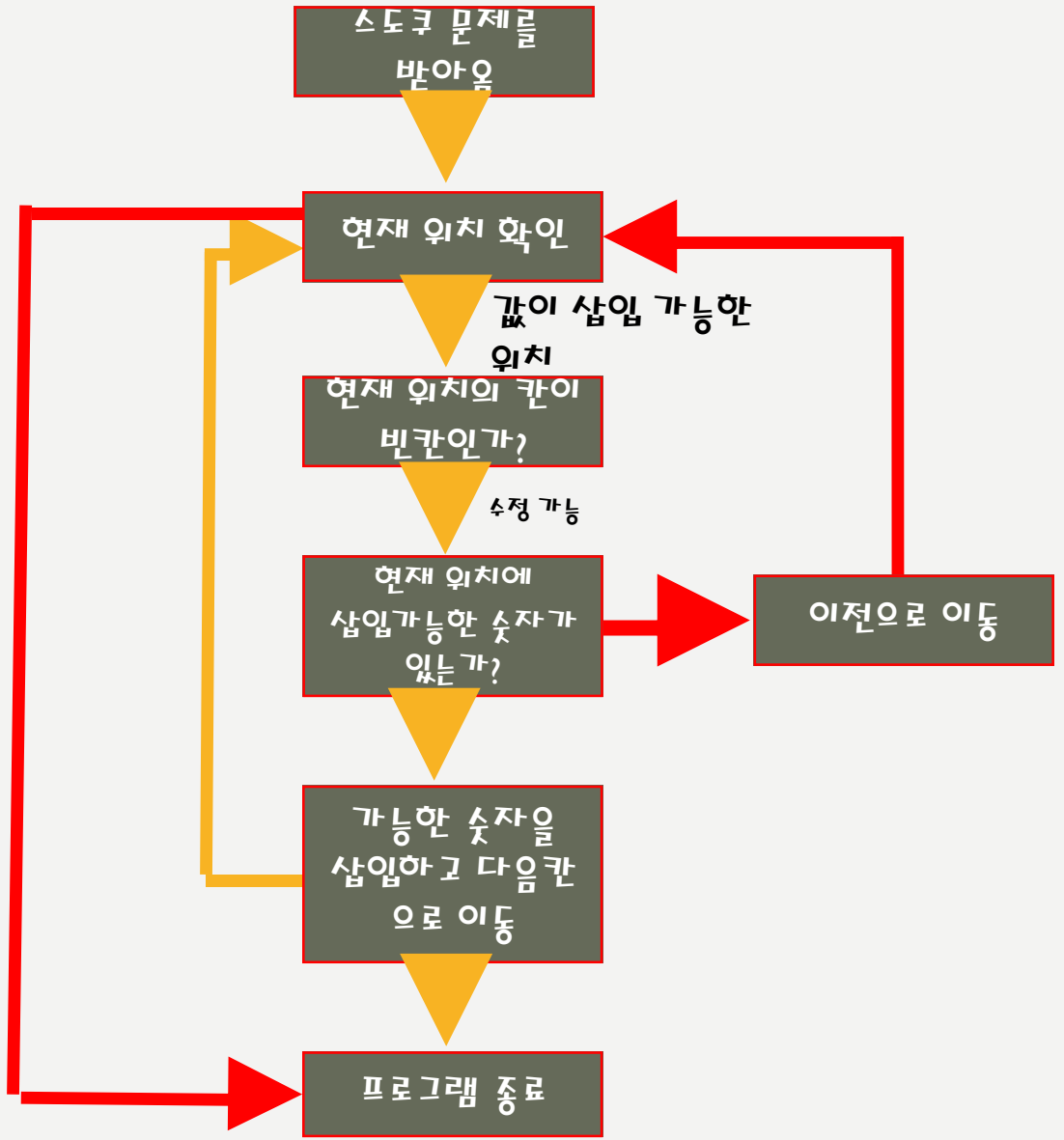
-모든 경우의 수를 전부 고려하는 알고리즘.

-일종의 트리 탐색 알고리즘이라고 봐도 된다.



# 백트레킹 순서도

첫번째 위치로 다시
돌아오거나
마지막 칸까지 채워지면
종료



```
def solve(v):
    global value
    global fixed
    value = copy.deepcopy(v)
    ready()
    posElement=[]
    i=0
    j=0
    print("")
    while(i<9):
        try:
            try:
                if(len(posElement[(i*9)+j])>=0):
                    pass
            except:
                posElement.append(getposElement(value,i,j))
            if(len(posElement[(i*9)+j])>0):
                if(not fixed[i][j]):
                    value[i][j] = copy.deepcopy(posElement[(i*9)+j].pop())
                else:
                    raise ValueError
        except:
            while(True):
                if(i == 0 and j == 0) :
                    print("Unsolvable")
                    return
                elif(j == 0 and i > 0):
                    posElement.pop()
                    if(not fixed[i][j]):
                        value[i][j] = 0
                    i -= 1
                    j = 8
                elif(j>0):
                    posElement.pop()
                    if(not fixed[i][j]):
                        value[i][j] = 0
                    j -= 1
                if(j == -1):
                    j = 1
                if(not fixed[i][j]):
                    value[i][j] = 0
                    j -= 1
                    break
        j += 1
        if(j==9):
            i += 1
            j = 0
    print_value()
```

백트레킹 알고리즘

스도쿠가 다 채워졌으면 종료

값이 있는지 없는지 확인

존재하지 않을 때 실행

posElement에 값을 넣어 줌

고정된 값인지 확인

가능한 리스트에서 해당 원소를 pop함

뒤로 돌아감

첫번째 칸으로 돌아오면 종료

해당 열에 0번째 행이면

이전 열의 마지막 행으로 돌아간다.

해당 행이 1이상이면

해당 위치를 방문하지 않은 상태로 변경함

수정 가능한 원소면

0으로 변경

이전 행으로 돌아감

행을 증가시킴

# 해결해야할 점

- openCV의 GaussianBlur, adaptiveThreshold 값 최적화
  - ➡ 현재 이미지 밝기, 잡음, 선 굵기에 따라 값을 변경 해야 됨.
- 스도쿠판 인식 문제
  - ➡ 이미지에 노이즈가 너무 많으면 결과에 노이즈가 포함됨
- 이미지 ➡숫자 변환 할 때 인식률이 떨어짐
- 유전알고리즘을 이용하면 가중치가 높은 값이 정답에 맞지 않으면 다른 알고리즘보다 좋지않은 결과가 나옴

# THE END