Programmeren IK 2019

Jelle van Assema

Vorige week

- Jupyter Notebook
- Comprehensions
- File IO

Deze week

- Oefenen met alles Python
- Constructieve zoekalgoritmes
- Recursie

Wachtwoord kraken


NL
01-ABC-1

10 x 10 x 26 x 26 x 26 x 10

17,576,000

Een wachtwoord van 10 tekens, bestaande uit:

- Letters (groot en klein)
- Cijfers
- De tekens: ! , .

Een wachtwoord van 10 tekens, bestaande uit:

- Letters (groot en klein)
- Cijfers
- De tekens: ! , .

$$(26 + 26 + 10 + 3)^{10}$$

$$1.3462743 * 10^{18}$$

Een wachtwoord van 10 tekens kraken?

Bij 1 miljard wachtwoorden per seconde:

$(1.3462743 * 10^{18}) / 10^{9} =$

$1.3462743 * 10^{9}$ seconden

373965 uur

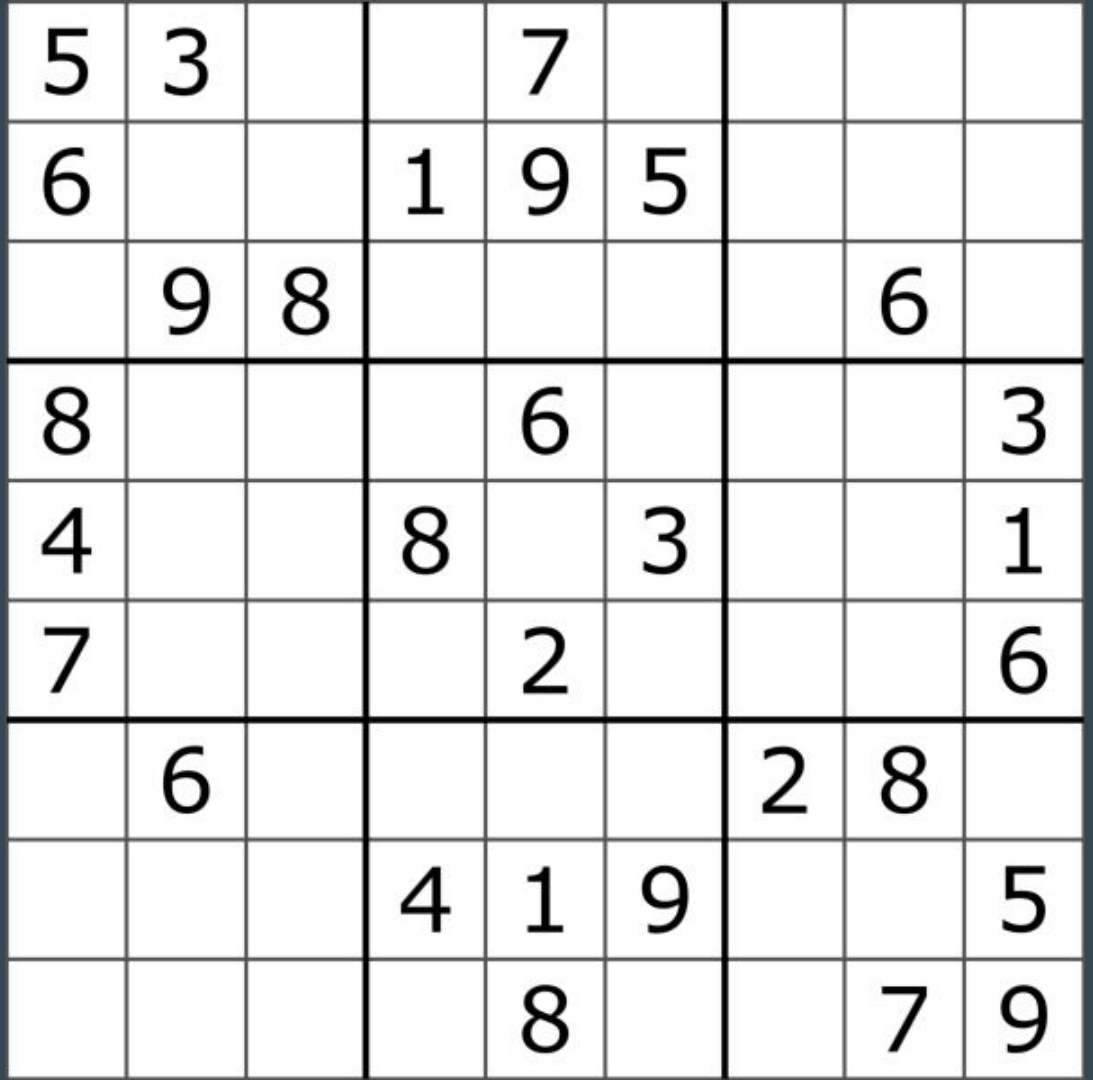
42.69 jaar

Een wachtwoord van 10 tekens kraken?

Wachtwoorden zijn vaak niet willekeurig.

In praktijk:

- Patronen
- Dictionary attacks
- 0000


5 3 7
6 1 9 5
9 8 6
8 6 3
4 8 3 1
7 2 6
6 2 8
4 1 9 5
8 7 9

Sudoku’s

Een bord van 9 x 9, met elk op vakje 9 mogelijkheden

$$9^{81}$$

$$1.9662705 * 10^{77}$$

$$1.9662705 * 10^{77}$$

It is estimated that there are between 10^{78} to 10^{82} atoms in the known, observable universe.

~~$1.9662705 * 10^{77}$~~

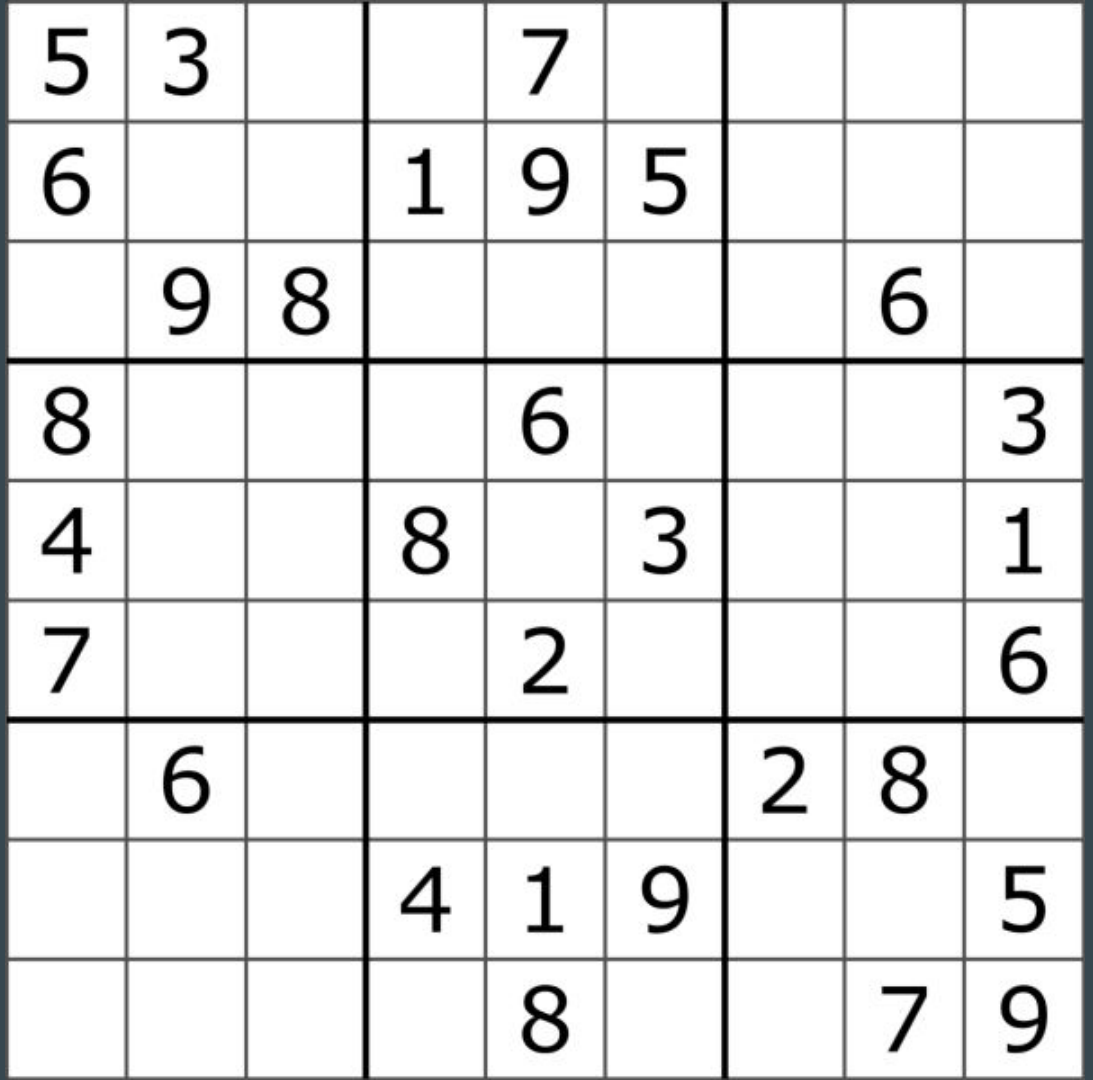
6,670,903,752,021,072,936,960

$/ (2.9475324 * 10^{55})$

Sudoku’s oplossen

Probleemgrootte is maar een fractie van:

$$6,670,903,752,021,072,936,960$$


5 3 7
6 1 9 5
9 8 6
8 6 3
4 8 3 1
7 2 6
6 2 8
4 1 9 5
8 7 9

Sudoku’s oplossen

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

- 51 lege vakjes
- 9^{51} mogelijkheden toch?
- Dat zijn $6.9531773 * 10^{26}$ keer zoveel mogelijkheden dan er sudoku’s bestaan

Sudoku’s oplossen

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

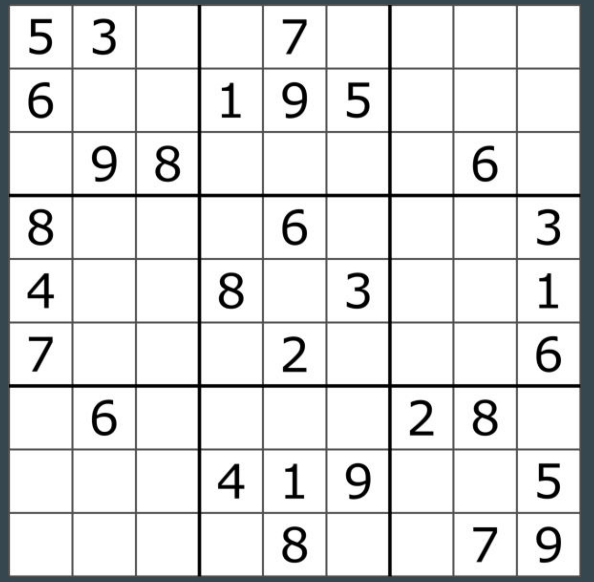
- 51 lege vakjes
- ~~Naïeve manier van oplossen~~
- Manier voor doorzoeken met enkel geldige sudoku’s.

Constructief oplossen

- Een opbouwende manier van gestructureerd zoeken

- ~~Naïeve manier van oplossen~~

- Manier voor doorzoeken met enkel geldige sudoku’s.

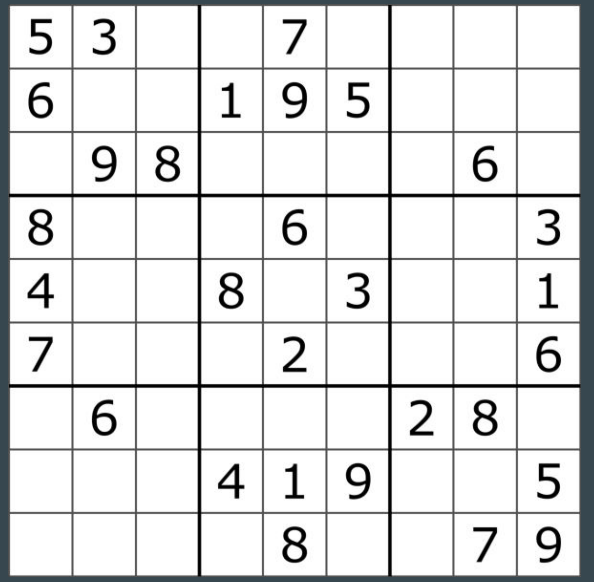


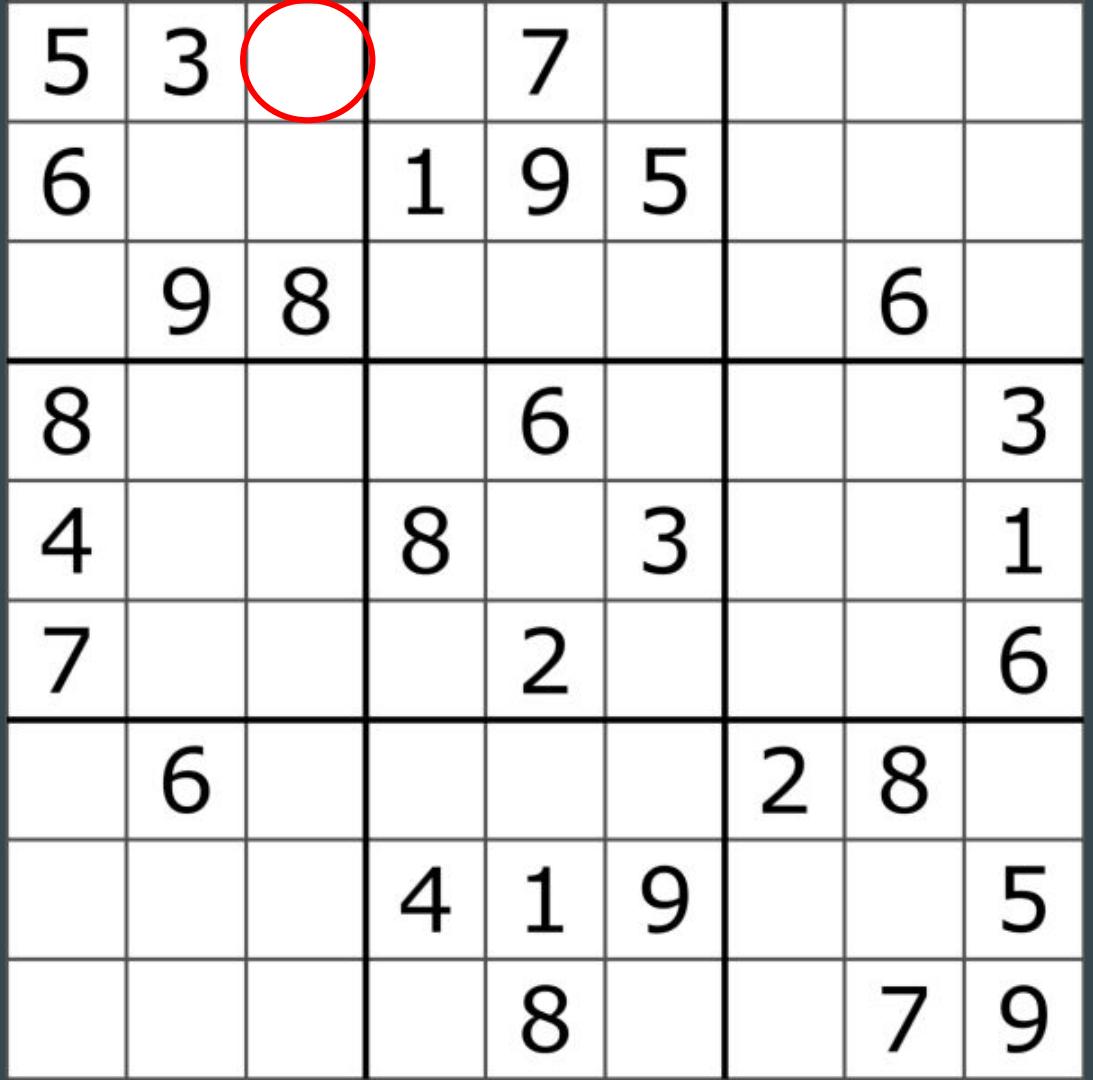
Constructief oplossen

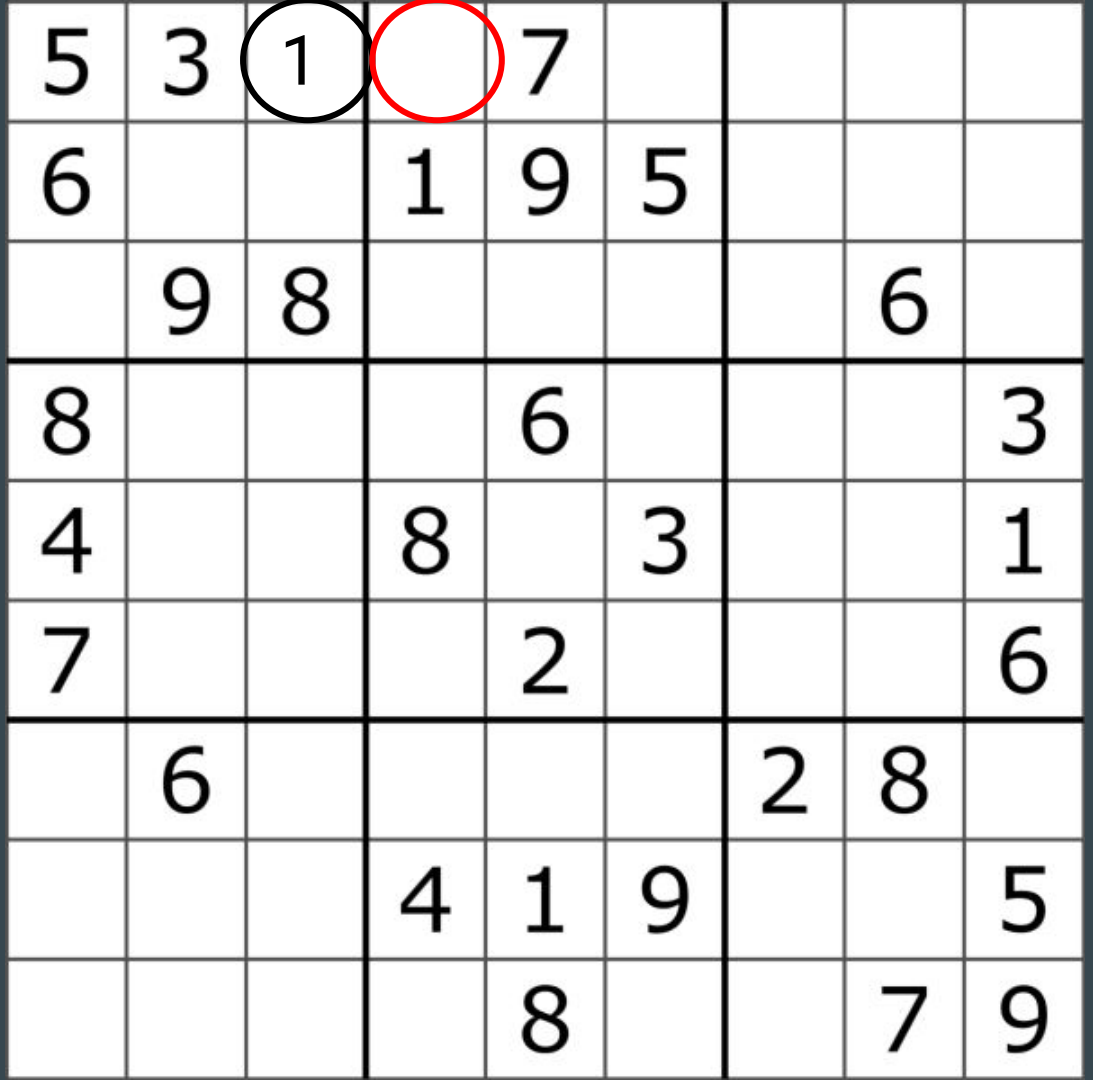
- Een opbouwende manier van gestructureerd zoeken

- ~~Naïeve manier van oplossen~~

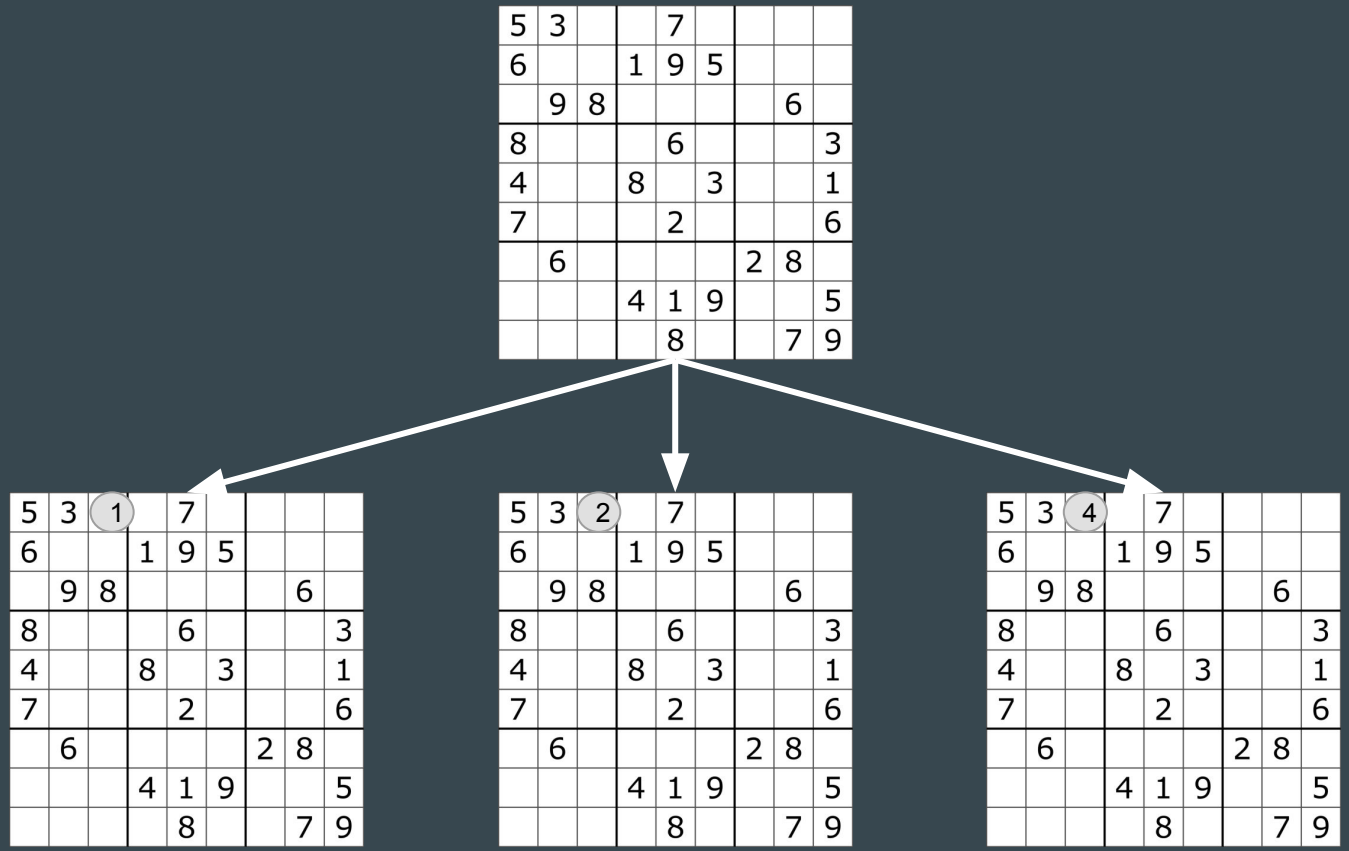
- Manier voor doorzoeken met enkel geldige sudoku’s.



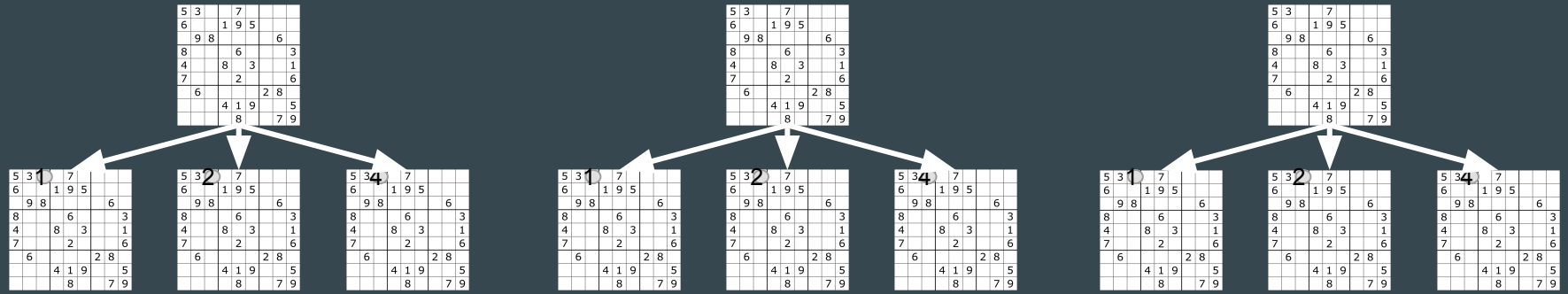

5 3 7
6 1 9 5
9 8 6
8 6 3
4 8 3 1
7 2 6
6 2 8
4 1 9 5
8 7 9


5 3 1 7
6 1 9 5
9 8 6
8 6 3
4 8 3 1
7 2 6
6 2 8
4 1 9 5
8 7 9

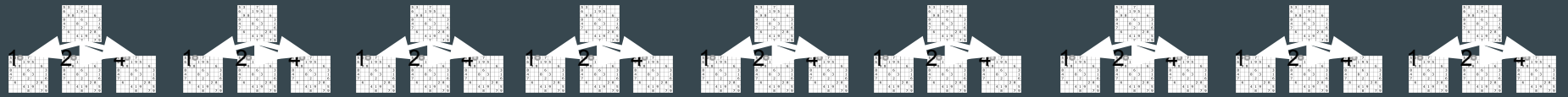
Breadth-First Search (BFS)



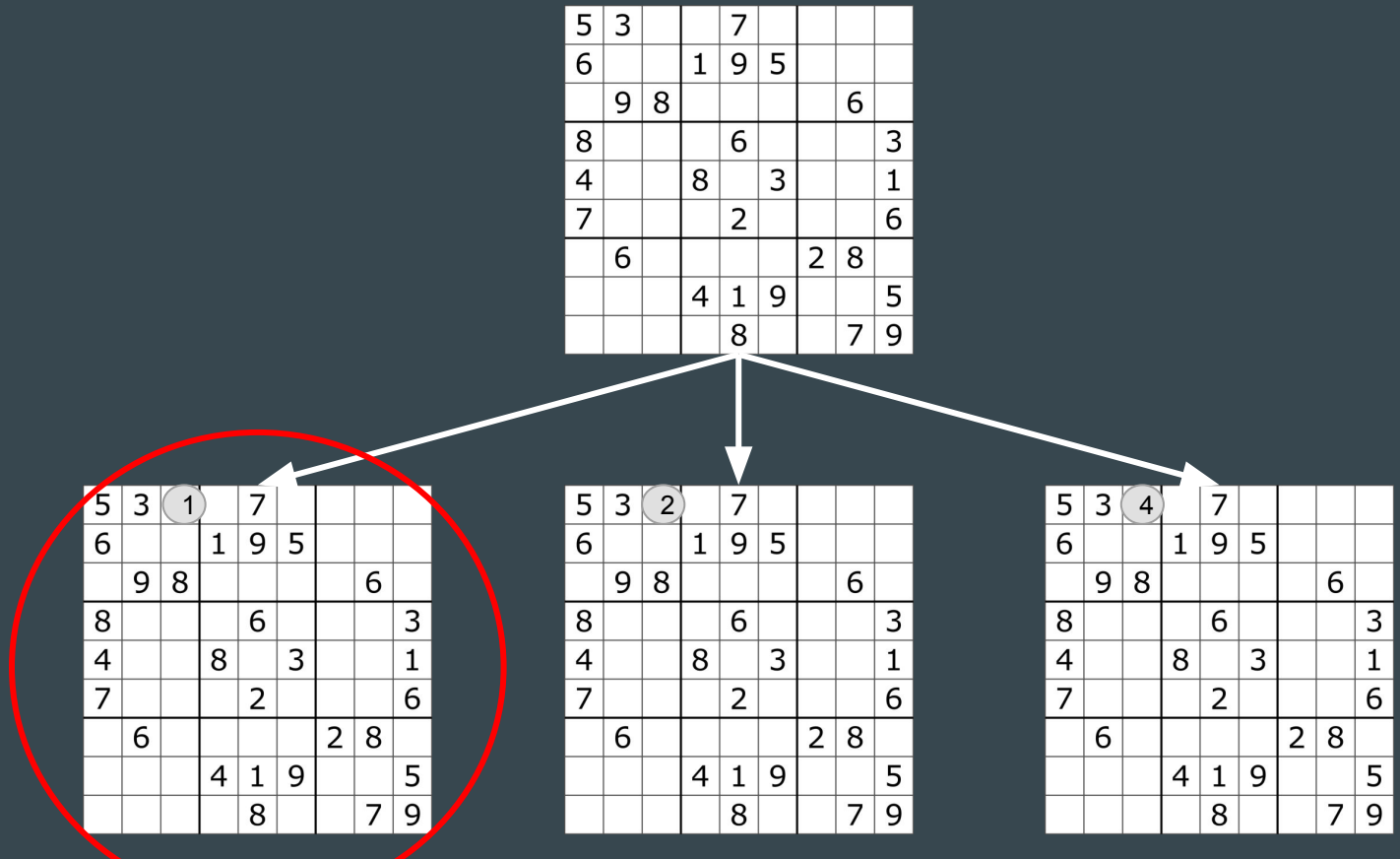
Breadth-First Search (BFS)



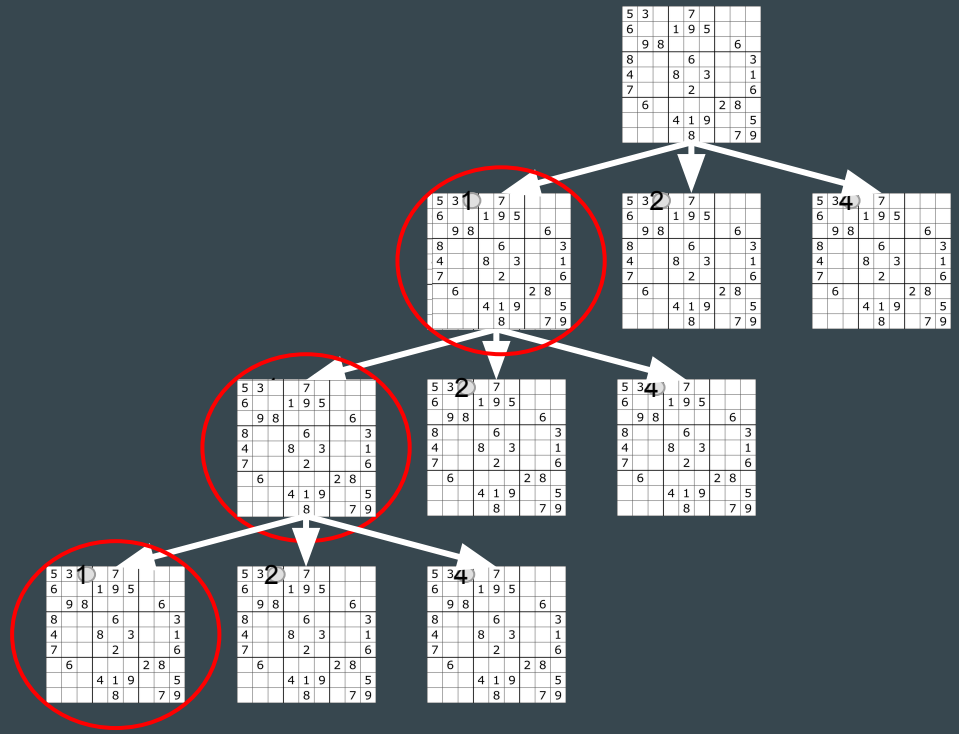
Breadth-First Search (BFS)



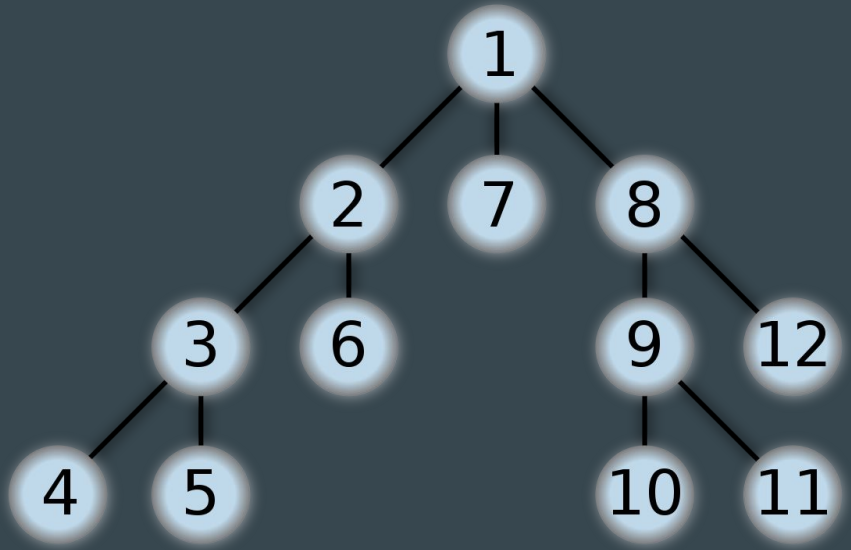
Depth-First Search (DFS)



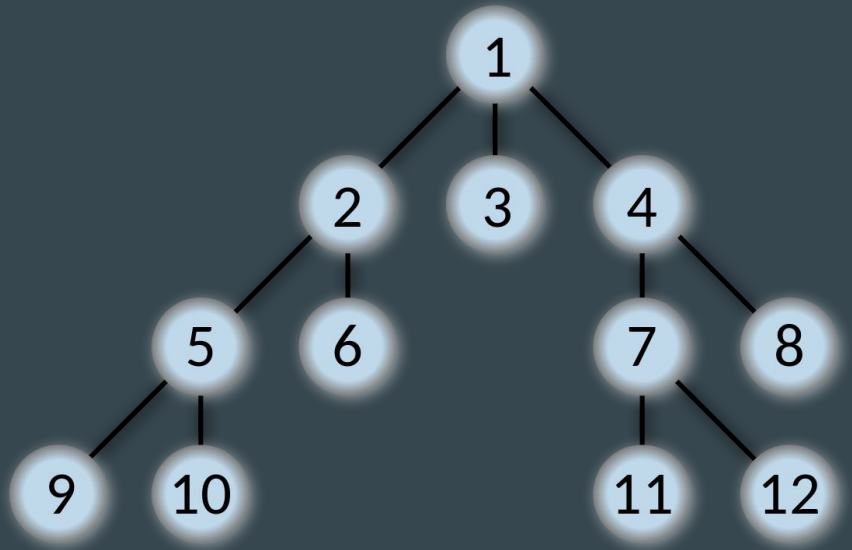
Depth-First Search (DFS)



DFS



BFS



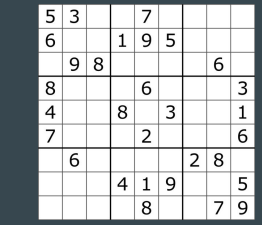
Wachtwoord kraken

DFS algoritme

```
1  function DFS(V)
2      let S be a stack
3      S.push(V)
4      while S is not empty
5          V = S.pop()
6          for all candidates C from V do
7              let W be a copy of V
8              apply C to W
9              if W is a solution do
10                 return W
11             S.push(W)
```

DFS algoritme

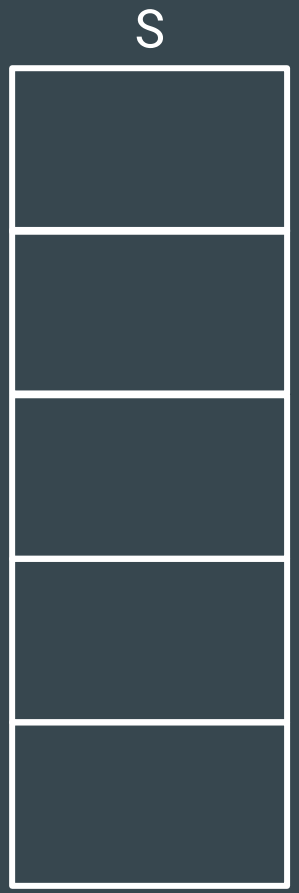
V




```
1   function DFS(V)
2       let S be a stack
3       S.push(V)
4       while S is not empty
5           V = S.pop()
6           for all candidates C from V do
7               let W be a copy of V
8               apply C to W
9               if W is a solution do
10                  return W
11              S.push(W)
```

DFS algoritme

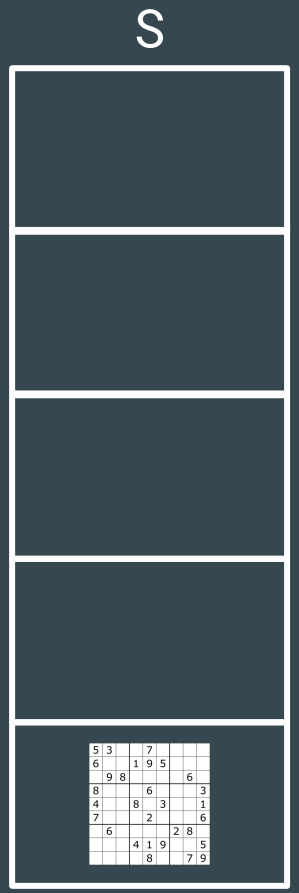
```
1   function DFS(V)
2       let S be a stack
3       S.push(V)
4       while S is not empty
5           V = S.pop()
6           for all candidates C from V do
7               let W be a copy of V
8               apply C to W
9               if W is a solution do
10                  return W
11              S.push(W)
```

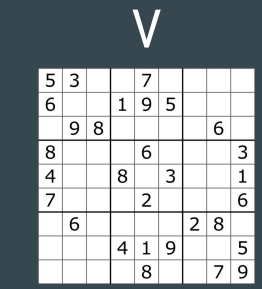


5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

DFS algoritme

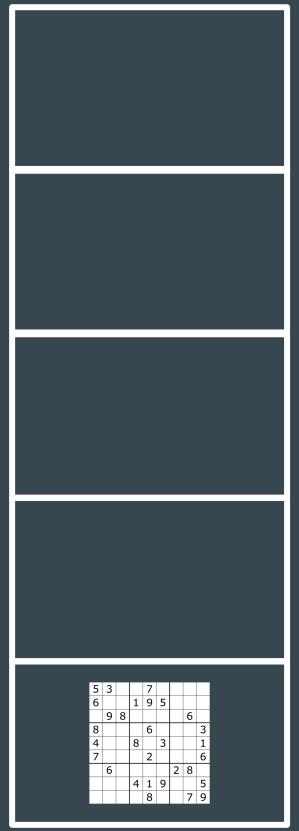
```
1   function DFS(V)
2       let S be a stack
3       S.push(V)
4       while S is not empty
5           V = S.pop()
6           for all candidates C from V do
7               let W be a copy of V
8               apply C to W
9               if W is a solution do
10                  return W
11              S.push(W)
```

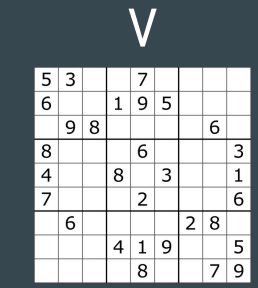




DFS algoritme

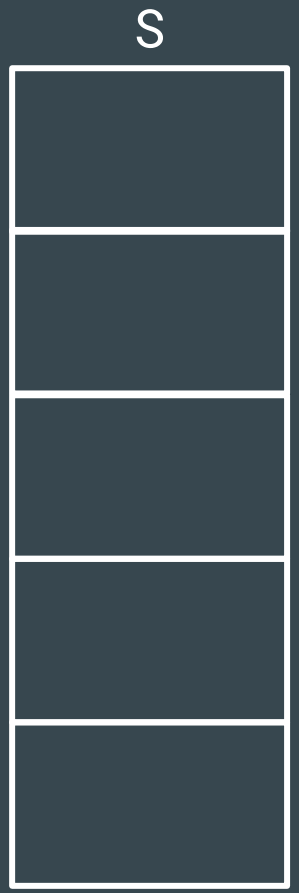
```
1   function DFS(V)
2       let S be a stack
3       S.push(V)
4       while S is not empty
5           V = S.pop()
6           for all candidates C from V do
7               let W be a copy of V
8               apply C to W
9               if W is a solution do
10                  return W
11              S.push(W)
```

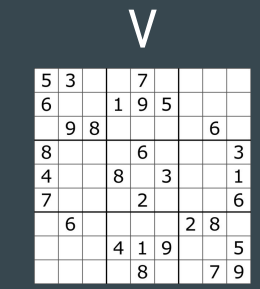




DFS algoritme

```
1   function DFS(V)
2       let S be a stack
3       S.push(V)
4       while S is not empty
5           V = S.pop()
6           for all candidates C from V do
7               let W be a copy of V
8               apply C to W
9               if W is a solution do
10                  return W
11              S.push(W)
```



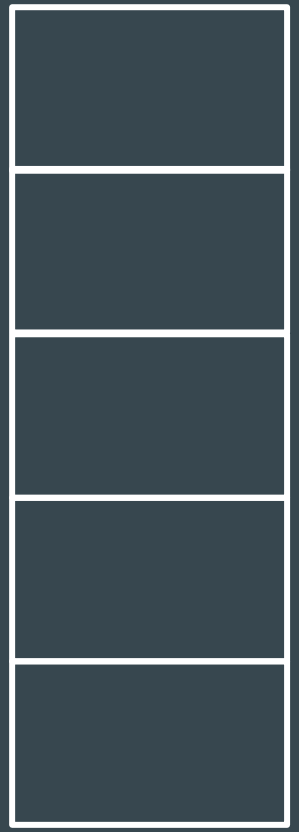


5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

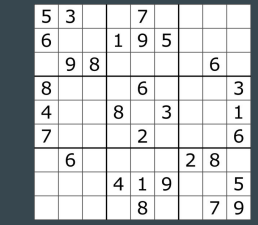
DFS algoritme

```
1   function DFS(V)
2       let S be a stack
3       S.push(V)
4       while S is not empty
5           V = S.pop()
6           for all candidates C from V do
7               let W be a copy of V
8               apply C to W
9               if W is a solution do
10                  return W
11              S.push(W)
```

S



V



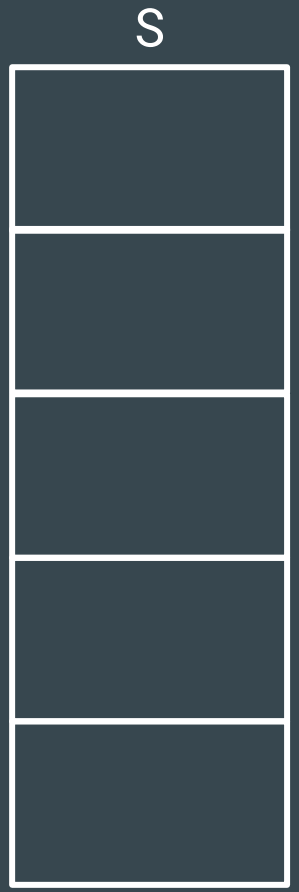
5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

C

1

DFS algoritme

```
1   function DFS(V)
2       let S be a stack
3       S.push(V)
4       while S is not empty
5           V = S.pop()
6           for all candidates C from V do
7               let W be a copy of V
8               apply C to W
9               if W is a solution do
10                  return W
11              S.push(W)
```



V

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

W

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

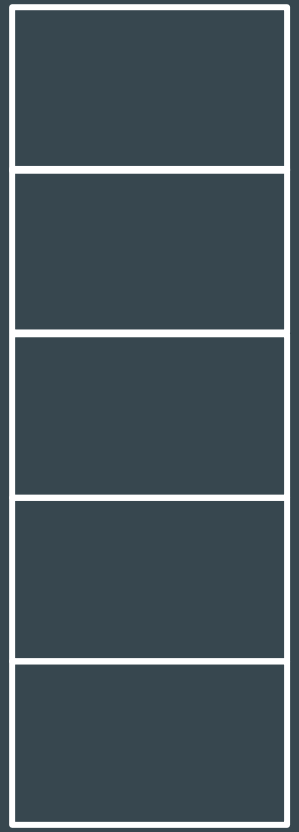
C

1

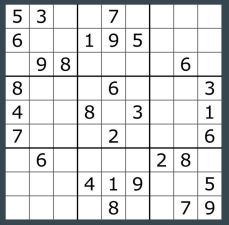
DFS algoritme

```
1   function DFS(V)
2       let S be a stack
3       S.push(V)
4       while S is not empty
5           V = S.pop()
6           for all candidates C from V do
7               let W be a copy of V
8               apply C to W
9               if W is a solution do
10                  return W
11              S.push(W)
```

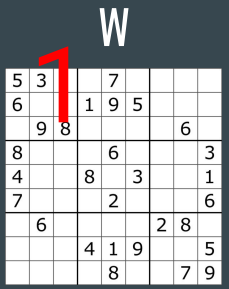
S



V



W



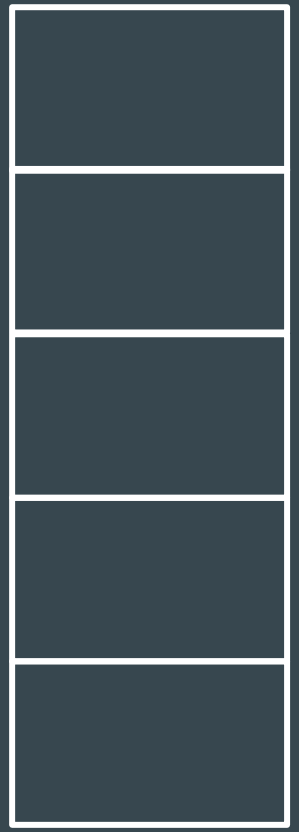
C

1

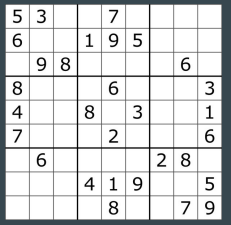
DFS algoritme

```
1   function DFS(V)
2       let S be a stack
3       S.push(V)
4       while S is not empty
5           V = S.pop()
6           for all candidates C from V do
7               let W be a copy of V
8               apply C to W
9               if W is a solution do
10                  return W
11              S.push(W)
```

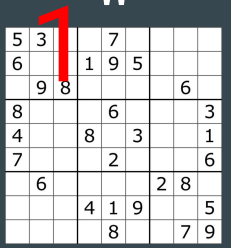
S



V



W

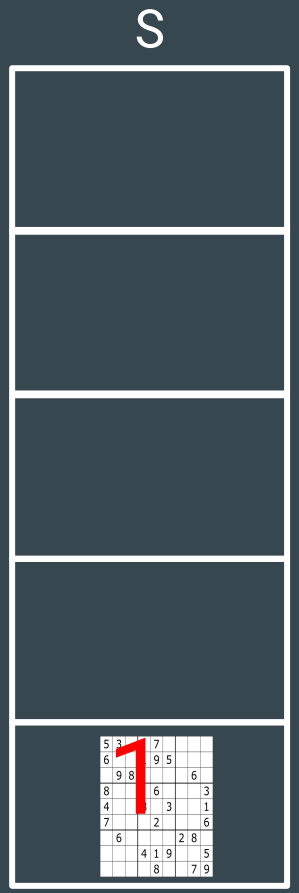


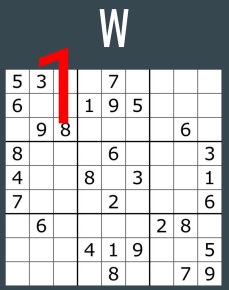
C

1

DFS algoritme

```
1   function DFS(V)
2       let S be a stack
3       S.push(V)
4       while S is not empty
5           V = S.pop()
6           for all candidates C from V do
7               let W be a copy of V
8               apply C to W
9               if W is a solution do
10                  return W
11              S.push(W)
```

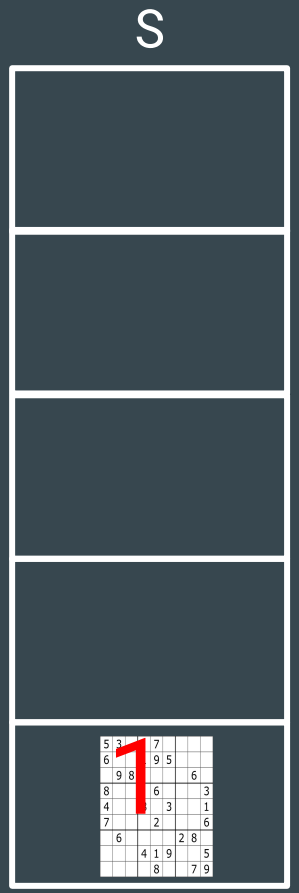


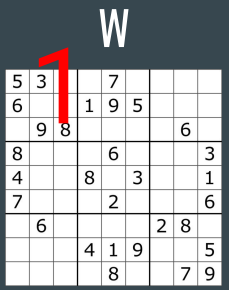


C
1

DFS algoritme

```
1   function DFS(V)
2       let S be a stack
3       S.push(V)
4       while S is not empty
5           V = S.pop()
6           for all candidates C from V do
7               let W be a copy of V
8               apply C to W
9               if W is a solution do
10                  return W
11              S.push(W)
```

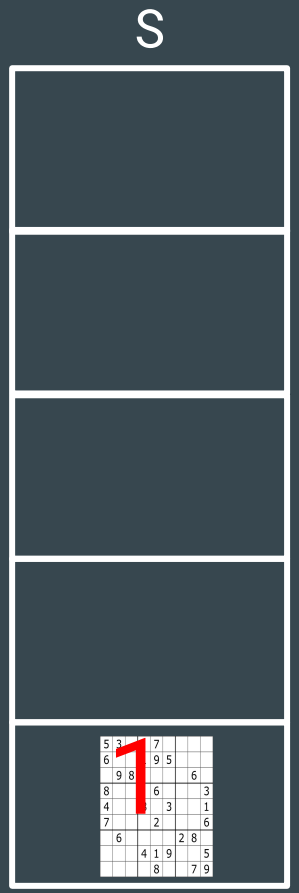




C
2

DFS algoritme

```
1   function DFS(V)
2       let S be a stack
3       S.push(V)
4       while S is not empty
5           V = S.pop()
6           for all candidates C from V do
7               let W be a copy of V
8               apply C to W
9               if W is a solution do
10                  return W
11              S.push(W)
```



V

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

W

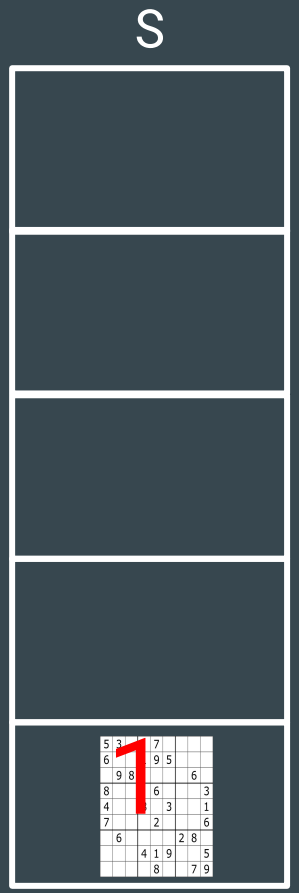
5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

C

2

DFS algoritme

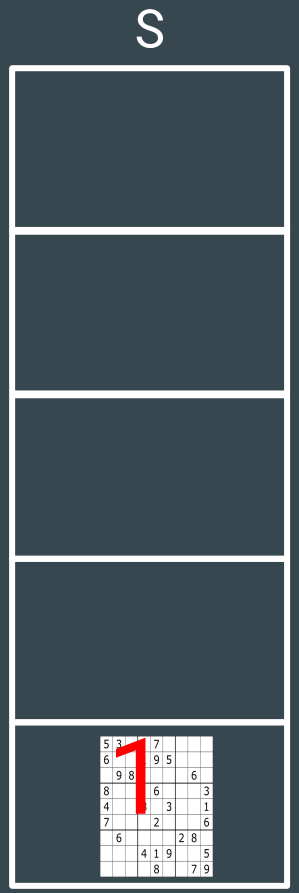
```
1   function DFS(V)
2       let S be a stack
3       S.push(V)
4       while S is not empty
5           V = S.pop()
6           for all candidates C from V do
7               let W be a copy of V
8               apply C to W
9               if W is a solution do
10                  return W
11              S.push(W)
```



C
2

DFS algoritme

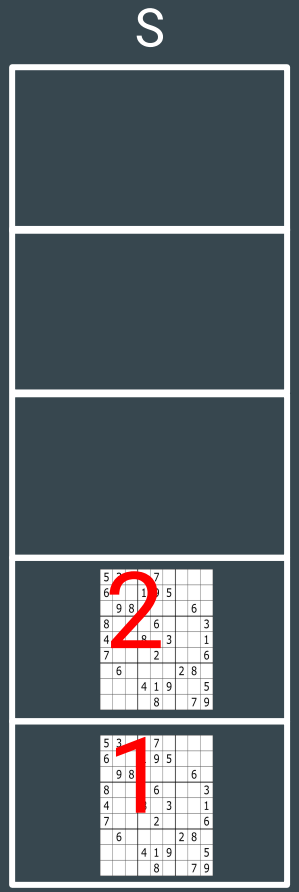
```
1   function DFS(V)
2       let S be a stack
3       S.push(V)
4       while S is not empty
5           V = S.pop()
6           for all candidates C from V do
7               let W be a copy of V
8               apply C to W
9               if W is a solution do
10                  return W
11              S.push(W)
```



C
2

DFS algoritme

```
1   function DFS(V)
2       let S be a stack
3       S.push(V)
4       while S is not empty
5           V = S.pop()
6           for all candidates C from V do
7               let W be a copy of V
8               apply C to W
9               if W is a solution do
10                  return W
11              S.push(W)
```



V

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

W

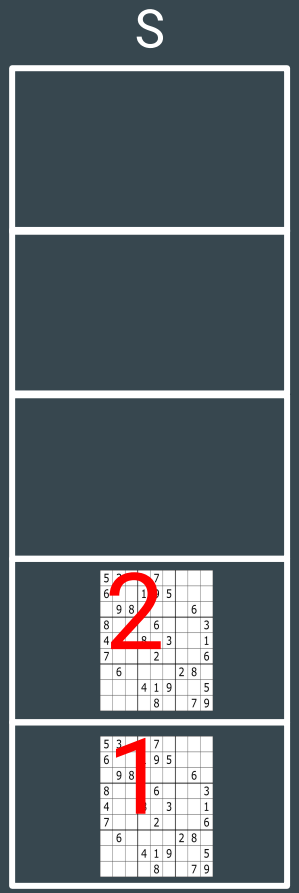
5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

C

2

DFS algoritme

```
1   function DFS(V)
2       let S be a stack
3       S.push(V)
4       while S is not empty
5           V = S.pop()
6           for all candidates C from V do
7               let W be a copy of V
8               apply C to W
9               if W is a solution do
10                  return W
11              S.push(W)
```

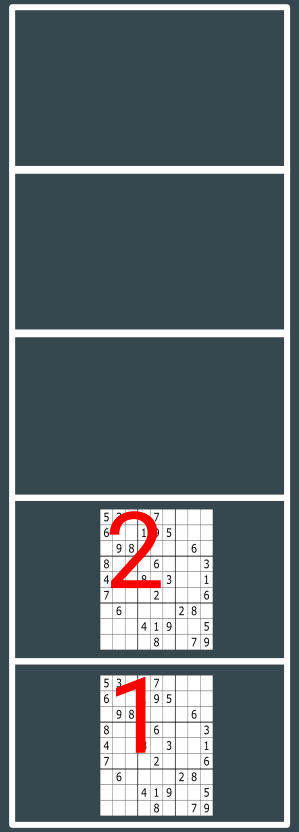


C
4

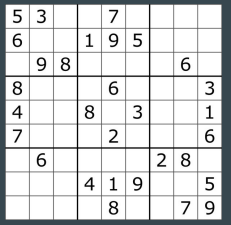
DFS algoritme

```
1   function DFS(V)
2       let S be a stack
3       S.push(V)
4       while S is not empty
5           V = S.pop()
6           for all candidates C from V do
7               let W be a copy of V
8               apply C to W
9               if W is a solution do
10                  return W
11              S.push(W)
```

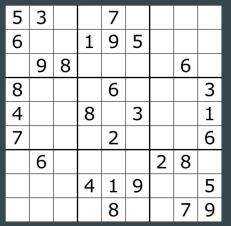
S



V



W

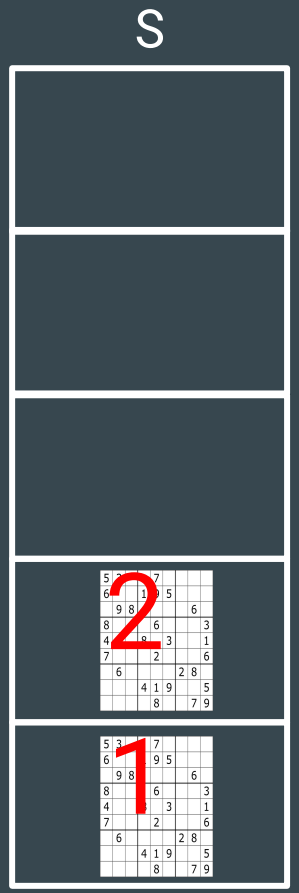


C

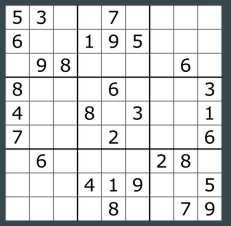
4

DFS algoritme

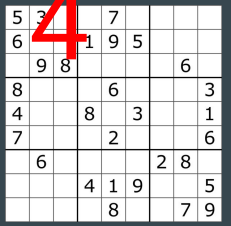
```
1   function DFS(V)
2       let S be a stack
3       S.push(V)
4       while S is not empty
5           V = S.pop()
6           for all candidates C from V do
7               let W be a copy of V
8               apply C to W
9               if W is a solution do
10                  return W
11              S.push(W)
```



V



W

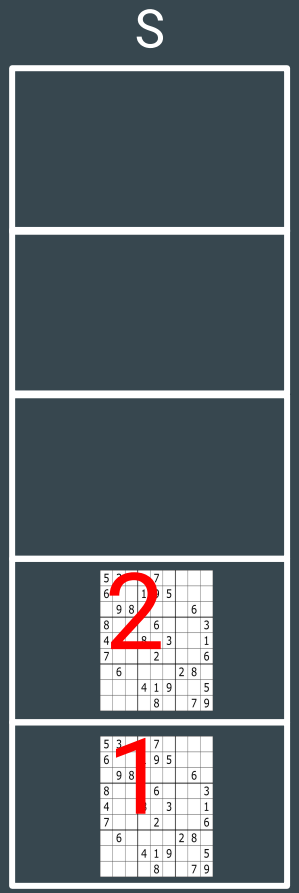


C

4

DFS algoritme

```
1   function DFS(V)
2       let S be a stack
3       S.push(V)
4       while S is not empty
5           V = S.pop()
6           for all candidates C from V do
7               let W be a copy of V
8               apply C to W
9               if W is a solution do
10                  return W
11              S.push(W)
```



V

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

W

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

4

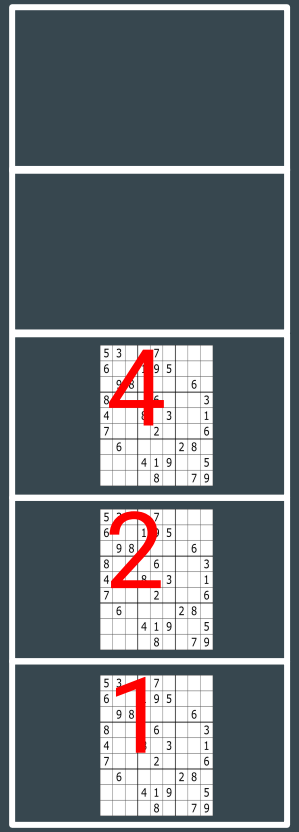
C

4

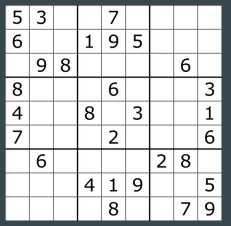
DFS algoritme

```
1   function DFS(V)
2       let S be a stack
3       S.push(V)
4       while S is not empty
5           V = S.pop()
6           for all candidates C from V do
7               let W be a copy of V
8               apply C to W
9               if W is a solution do
10                  return W
11              S.push(W)
```

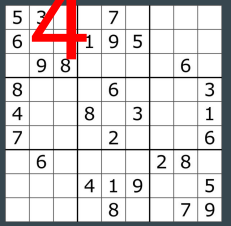
S



V



W



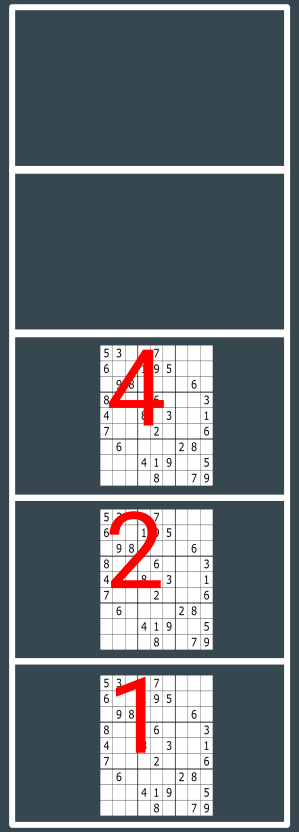
C

4

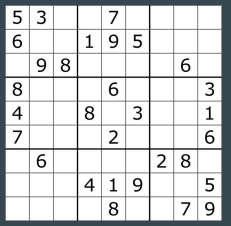
DFS algoritme

```
1   function DFS(V)
2       let S be a stack
3       S.push(V)
4       while S is not empty
5           V = S.pop()
6           for all candidates C from V do
7               let W be a copy of V
8               apply C to W
9               if W is a solution do
10                  return W
11              S.push(W)
```

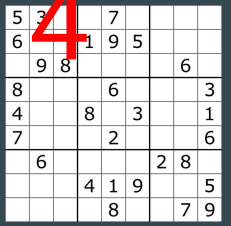
S



V



W

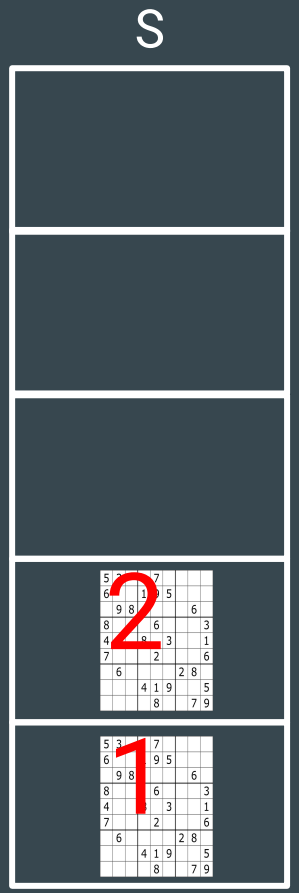


C

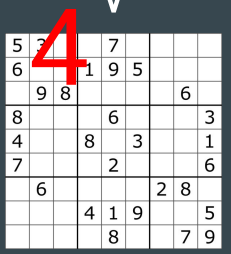
4

DFS algoritme

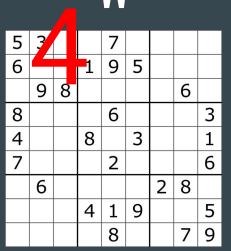
```
1   function DFS(V)
2       let S be a stack
3       S.push(V)
4       while S is not empty
5           V = S.pop()
6           for all candidates C from V do
7               let W be a copy of V
8               apply C to W
9               if W is a solution do
10                  return W
11              S.push(W)
```



V



W

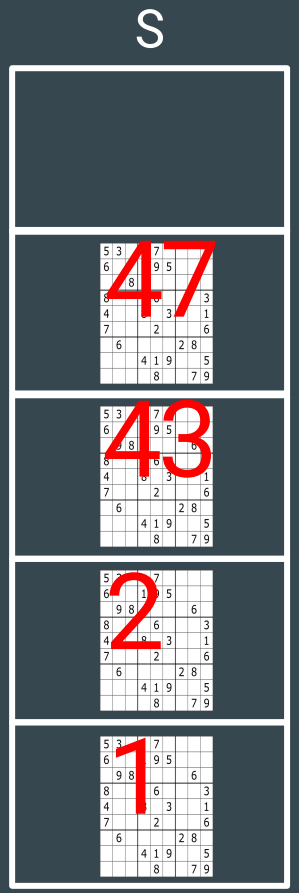


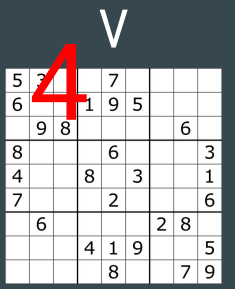
C

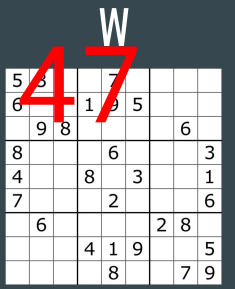
4

DFS algoritme

```
1   function DFS(V)
2       let S be a stack
3       S.push(V)
4       while S is not empty
5           V = S.pop()
6           for all candidates C from V do
7               let W be a copy of V
8               apply C to W
9               if W is a solution do
10                  return W
11              S.push(W)
```





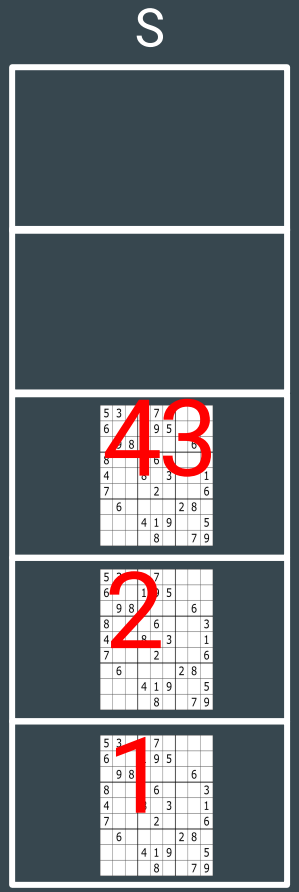


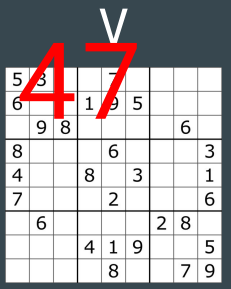
C

7

DFS algoritme

```
1   function DFS(V)
2       let S be a stack
3       S.push(V)
4       while S is not empty
5           V = S.pop()
6           for all candidates C from V do
7               let W be a copy of V
8               apply C to W
9               if W is a solution do
10                  return W
11              S.push(W)
```

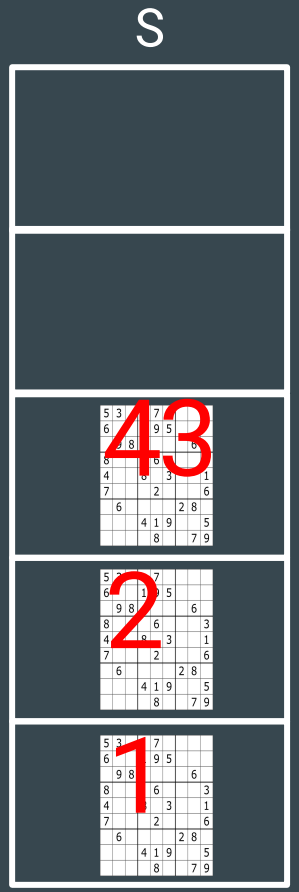


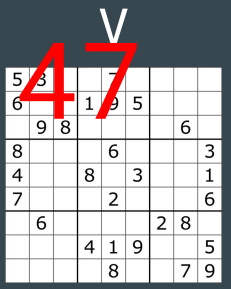


C
7

DFS algoritme

```
1   function DFS(V)
2       let S be a stack
3       S.push(V)
4       while S is not empty
5           V = S.pop()
6           for all candidates C from V do
7               let W be a copy of V
8               apply C to W
9               if W is a solution do
10                  return W
11              S.push(W)
```

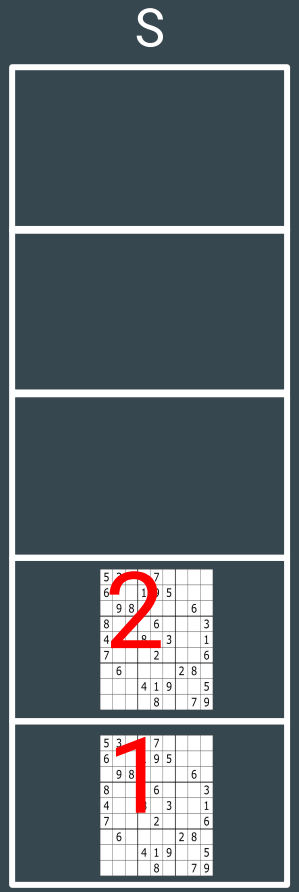


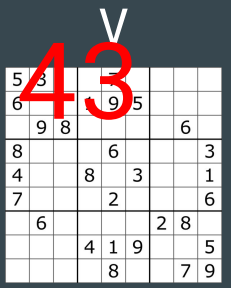


C
7

DFS algoritme

```
1   function DFS(V)
2       let S be a stack
3       S.push(V)
4       while S is not empty
5           V = S.pop()
6           for all candidates C from V do
7               let W be a copy of V
8               apply C to W
9               if W is a solution do
10                  return W
11              S.push(W)
```

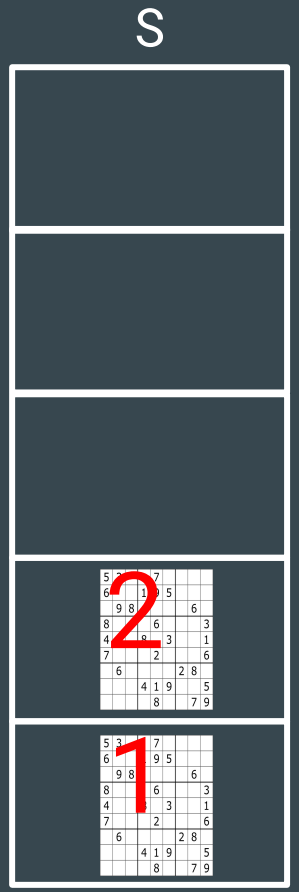


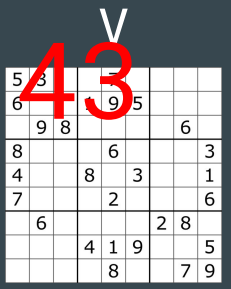


C
7

DFS algoritme

```
1   function DFS(V)
2       let S be a stack
3       S.push(V)
4       while S is not empty
5           V = S.pop()
6           for all candidates C from V do
7               let W be a copy of V
8               apply C to W
9               if W is a solution do
10                  return W
11              S.push(W)
```

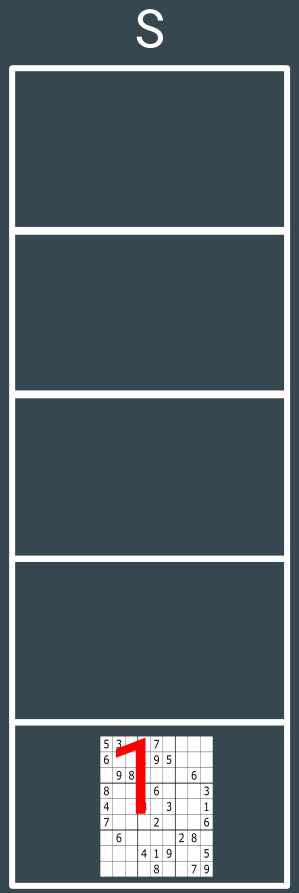




C
7

DFS algoritme

```
1   function DFS(V)
2       let S be a stack
3       S.push(V)
4       while S is not empty
5           V = S.pop()
6           for all candidates C from V do
7               let W be a copy of V
8               apply C to W
9               if W is a solution do
10                  return W
11              S.push(W)
```



C
7

DFS algoritme

- Eerst de diepte in

- Kom je niet verder, backtracken!

Recursie

- Een functie die zichzelf aanroept

```
def forever():
    print(“hello”)
    forever()
```

Recursie

- Basecase
- Reductiestap

Is een getal even?

Is een getal even?

```
0 is een even getal => true


1 is een oneven getal => false
```

Is een getal even?

```
0 is een even getal => true


1 is een oneven getal => false


is_even(n) = is_even(n - 2)
```

Recursie, is een getal even?

```
def even(n):
    if n == 0:
        return True
    if n == 1:
        return False
    return even(n - 2)
```

Recursie, binary search

Recursie, binary search

```
naald gevonden => true

niks meer te doorzoeken => false
```

Recursie, binary search

naald gevonden => true

niks meer te doorzoeken => false

needle < midden? doorzoek linkerhelft

needle > midden? Doorzoek rechterhelft

Recursie, binary search

```
def binary_search(numbers, n):
    if len(numbers) == 0:
        return False
    mid = len(numbers) // 2
    if numbers[mid] == n:
        return True
    ...
```

Recursie, binary search

```
def binary_search(numbers, n):
    if len(numbers) == 0:
        return False
    mid = len(numbers) // 2
    if numbers[mid] == n:
        return True
    if numbers[mid] > n:
        return binary_search(numbers[:n])
    else:
        return binary_search(numbers[n + 1:])
```

Recursie, wachtwoord kraken

Recursie, wachtwoord kraken

```
guess is gelijk aan wachtwoord? => True

lengte van gok is max_lengte? => False
```

Recursie, wachtwoord kraken

```
guess is gelijk aan wachtwoord? => True

lengte van gok is max_lengte? => False


voor elke optie:

   crack(guess + optie)
```

Recursie, sudoku

Recursie, sudoku

```
Alles ingevuld => opgelost

Geen optie meer te verkennen => onopgelost
```

Recursie, sudoku

Alles ingevuld => opgelost

Geen optie meer te verkennen => onopgelost

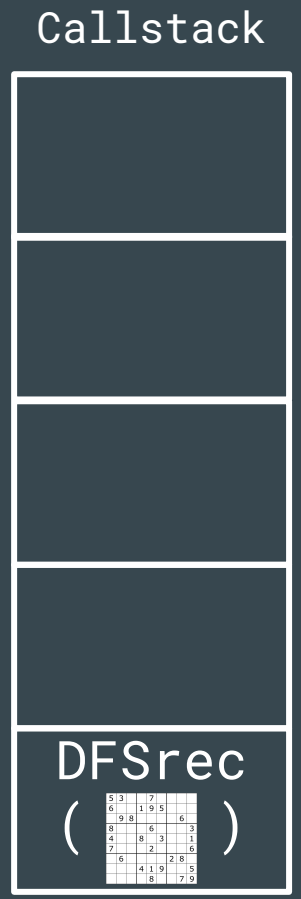
Vul een vakje in
- Ga van een sudoku met n lege vakjes, naar n - 1 lege vakjes

DFS recursief

```
1  function DFS-recursive(V)
2      if V is solved
3          return V
4
5      for all candidates C from V do
6          apply C to V
7          DFS-recursive(V)
8          if V is solved
9              return V
10         undo C to V
```

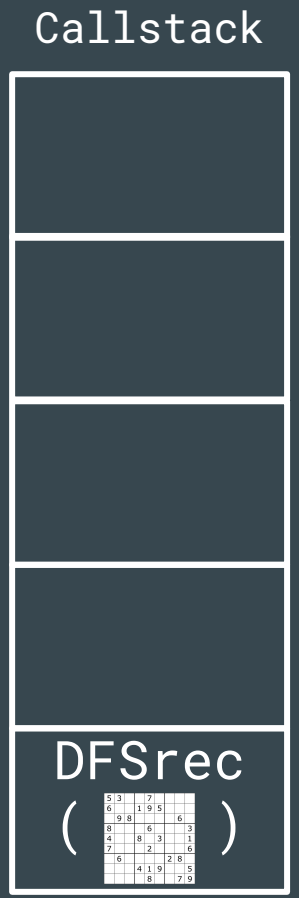
DFS recursief

```
1  function DFS-recursive(V)
2      if V is solved
3          return V
4
5      for all candidates C from V do
6          apply C to V
7          DFS-recursive(V)
8          if V is solved
9              return V
10         undo C to V
```



DFS recursief

```
1   function DFS-recursive(V)
2       if V is solved
3           return V
4
5       for all candidates C from V do
6           apply C to V
7           DFS-recursive(V)
8           if V is solved
9               return V
10          undo C to V
```



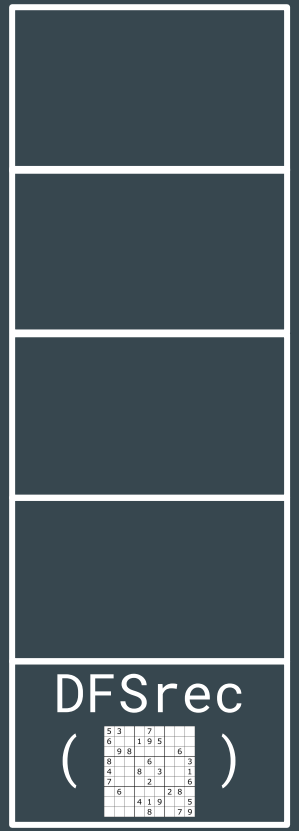
V

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

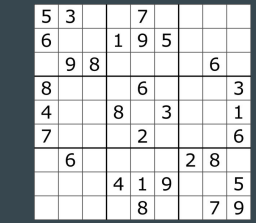
DFS recursief

```
1   function DFS-recursive(V)
2       if V is solved
3           return V
4
5       for all candidates C from V do
6           apply C to V
7           DFS-recursive(V)
8           if V is solved
9               return V
10          undo C to V
```

Callstack

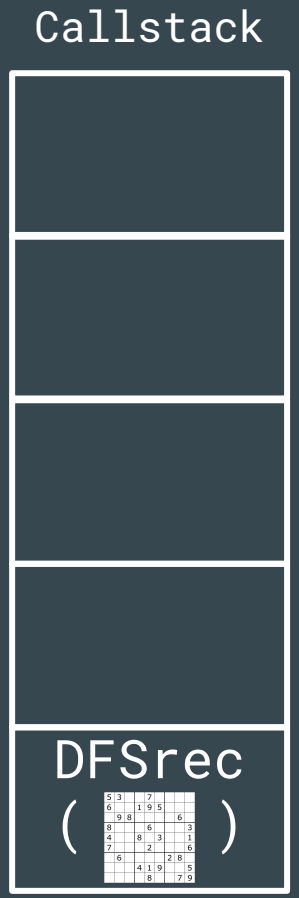


V



DFS recursief

```
1   function DFS-recursive(V)
2       if V is solved
3           return V
4
5       for all candidates C from V do
6           apply C to V
7           DFS-recursive(V)
8           if V is solved
9               return V
10          undo C to V
```



V

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

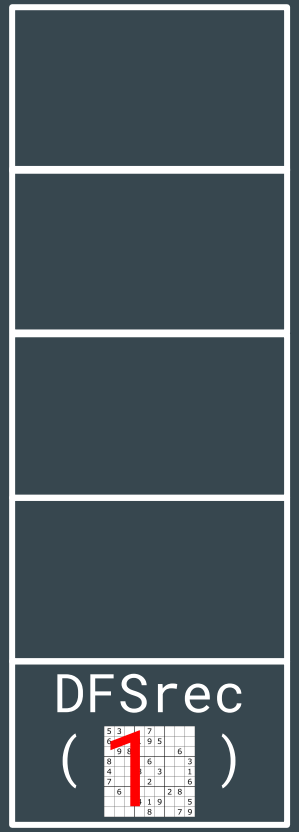
C

1

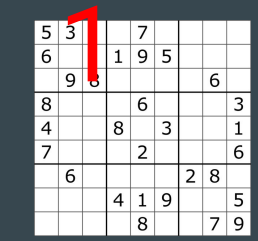
DFS recursief

```
1   function DFS-recursive(V)
2       if V is solved
3           return V
4
5       for all candidates C from V do
6           apply C to V
7           DFS-recursive(V)
8           if V is solved
9               return V
10          undo C to V
```

Callstack



V



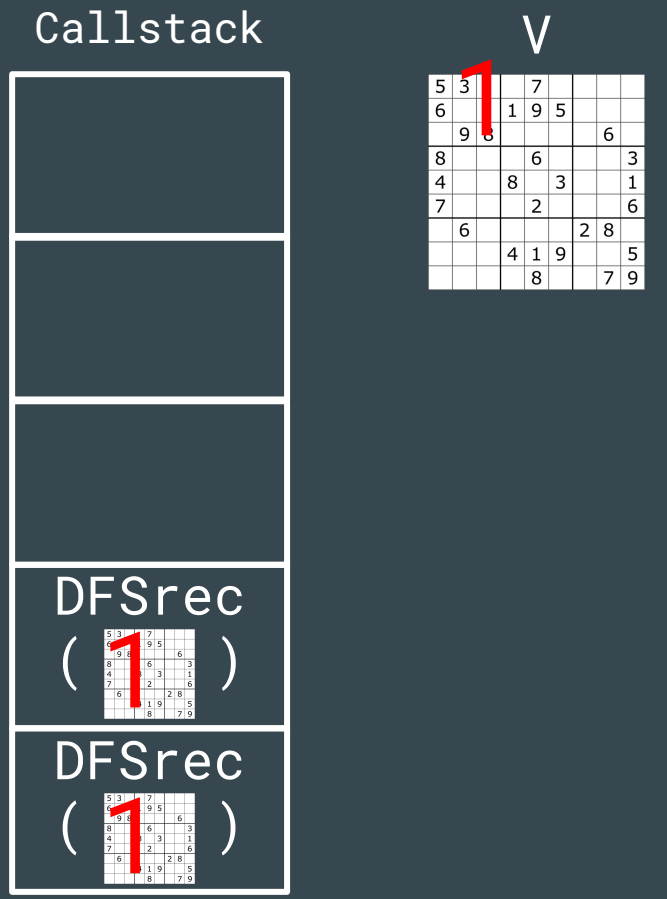
C
1

DFS recursief

```
1   function DFS-recursive(V)
2       if V is solved
3           return V
4
5       for all candidates C from V do
6           apply C to V
7           DFS-recursive(V)
8           if V is solved
9               return V
10          undo C to V
```

DFS recursief

```
1   function DFS-recursive(V)
2       if V is solved
3           return V
4
5       for all candidates C from V do
6           apply C to V
7           DFS-recursive(V)
8           if V is solved
9               return V
10          undo C to V
```



DFS recursief

```
1   function DFS-recursive(V)
2       if V is solved
3           return V
4
5       for all candidates C from V do
6           apply C to V
7           DFS-recursive(V)
8           if V is solved
9               return V
10          undo C to V
```

DFS recursief

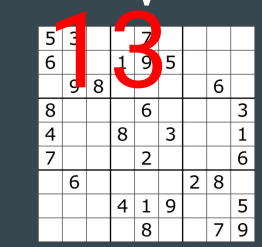
```
1   function DFS-recursive(V)
2       if V is solved
3           return V
4
5       for all candidates C from V do
6           apply C to V
7           DFS-recursive(V)
8           if V is solved
9               return V
10          undo C to V
```

DFS recursief

```
1   function DFS-recursive(V)
2       if V is solved
3           return V
4
5       for all candidates C from V do
6           apply C to V
7           DFS-recursive(V)
8           if V is solved
9               return V
10          undo C to V
```

Callstack

V

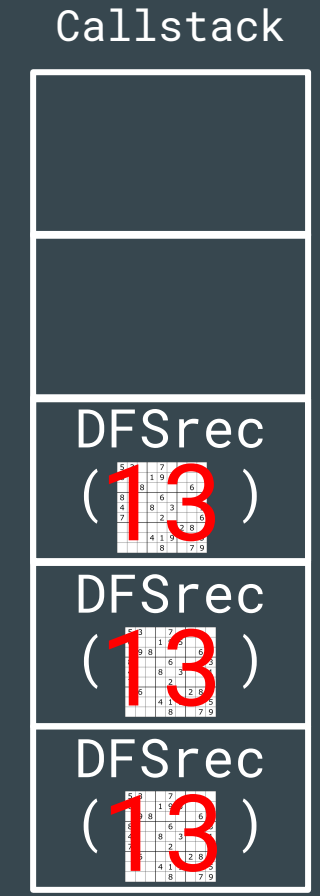


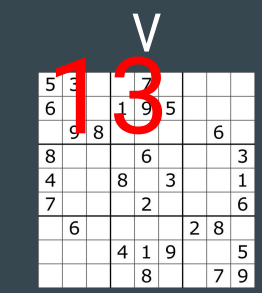
C

3

DFS recursief

```
1   function DFS-recursive(V)
2       if V is solved
3           return V
4
5       for all candidates C from V do
6           apply C to V
7           DFS-recursive(V)
8           if V is solved
9               return V
10          undo C to V
```





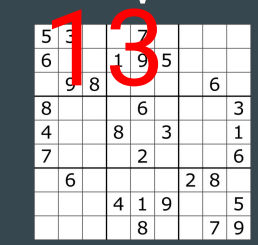
C
3

DFS recursief

```
1   function DFS-recursive(V)
2       if V is solved
3           return V
4
5       for all candidates C from V do
6           apply C to V
7           DFS-recursive(V)
8           if V is solved
9               return V
10          undo C to V
```

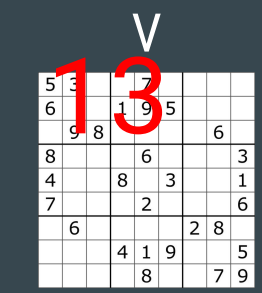
Callstack

V



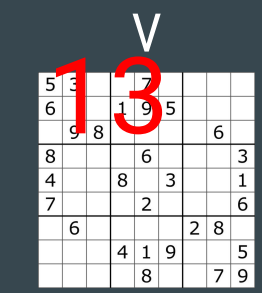
DFS recursief

```
1   function DFS-recursive(V)
2       if V is solved
3           return V
4
5       for all candidates C from V do
6           apply C to V
7           DFS-recursive(V)
8           if V is solved
9               return V
10          undo C to V
```



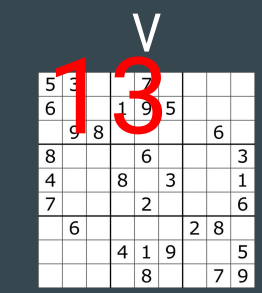
DFS recursief

```
1   function DFS-recursive(V)
2       if V is solved
3           return V
4
5       for all candidates C from V do
6           apply C to V
7           DFS-recursive(V)
8           if V is solved
9               return V
10          undo C to V
```



DFS recursief

```
1   function DFS-recursive(V)
2       if V is solved
3           return V
4
5       for all candidates C from V do
6           apply C to V
7           DFS-recursive(V)
8           if V is solved
9               return V
10          undo C to V
```



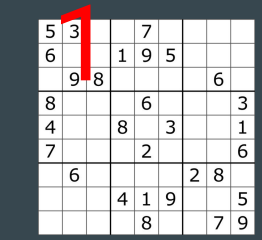
C
3

DFS recursief

```
1   function DFS-recursive(V)
2       if V is solved
3           return V
4
5       for all candidates C from V do
6           apply C to V
7           DFS-recursive(V)
8           if V is solved
9               return V
10          undo C to V
```

Callstack

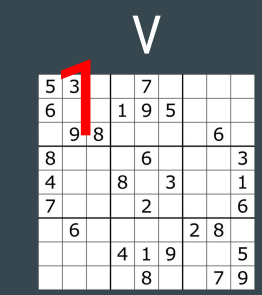
V



C
3

DFS recursief

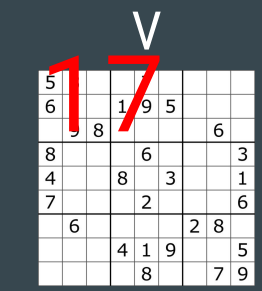
```
1   function DFS-recursive(V)
2       if V is solved
3           return V
4
5       for all candidates C from V do
6           apply C to V
7           DFS-recursive(V)
8           if V is solved
9               return V
10          undo C to V
```



C
7

DFS recursief

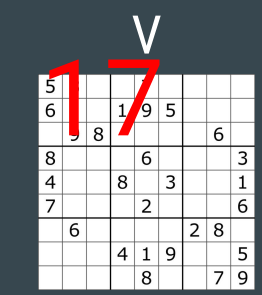
```
1   function DFS-recursive(V)
2       if V is solved
3           return V
4
5       for all candidates C from V do
6           apply C to V
7           DFS-recursive(V)
8           if V is solved
9               return V
10          undo C to V
```



C
7

DFS recursief

```
1   function DFS-recursive(V)
2       if V is solved
3           return V
4
5       for all candidates C from V do
6           apply C to V
7           DFS-recursive(V)
8           if V is solved
9               return V
10          undo C to V
```



C
7

DFS algoritme

- Verschillende implementaties mogelijk

- Hacker editie: iteratief DFS met generators

Deeltentamen 2