

1 2 3 4	1 2 3 4	1 2 3 4	1 2 3 4
1 2 3 4	1 2 3 4	1 2 3 4	1 2 3 4
1 2 3 4	1 2 3 4	1 2 3 4	1 2 3 4
1 2 3 4	1 2 3 4	1 2 3 4	1 2 3 4

`solve(problem: unary constraints)`

- `domains = init_domains()`
- `restrict_domains(problem, domains)`
- initialize `stack` as an empty stack
- while `true`:
  - if `propagate(domains)` reports no conflict:
    - if  $|\text{domains}[\text{spot}]| = 1$  for all `spots`, this is a consistent assignment
    - `orig_domains = copy(domains)`
    - `spot, num = search(domains)`
    - add `(spot, num, orig_domains)` to `stack`
  - otherwise:
    - if `stack` is empty, there is no consistent assignment
    - `domains = backtrack(stack)`

1	1 2 3 4	1 2 3 4	2
1 2 3 4	2	1 2 3 4	1 2 3 4
1 2 3 4	1 2 3 4	2	1 2 3 4
1 2 3 4	1 2 3 4	1 2 3 4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)

1	1 2 3 4	1 2 3 4	2
1 2 3 4	2	1 2 3 4	1 2 3 4
1 2 3 4	1 2 3 4	2	1 2 3 4
1 2 3 4	1 2 3 4	1 2 3 4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	1 2 3 4	1 2 3 4	2
1 2 3 4	2	1 2 3 4	1 2 3 4
1 2 3 4	1 2 3 4	2	1 2 3 4
1 2 3 4	1 2 3 4	1 2 3 4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	2 3 4	2 3 4	2
2 3 4	2	1 2 3 4	1 2 3 4
2 3 4	1 2 3 4	2	1 2 3 4
2 3 4	1 2 3 4	1 2 3 4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	2 3 4	2 3 4	2
2 3 4	2	1 2 3 4	1 2 3 4
2 3 4	1 2 3 4	2	1 2 3 4
2 3 4	1 2 3 4	1 2 3 4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	2 3 4	2 3 4	2
2 3 4	2	1 2 3 4	1 2 3 4
2 3 4	1 2 3 4	2	1 2 3 4
2 3 4	1 2 3 4	1 2 3 4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	2 3 4	2 3 4	2
2 3 4	2	1 2 3 4	1 2 3 4
2 3 4	1 2 3 4	2	1 2 3 4
2 3 4	1 2 3 4	1 2 3 4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict



1	2 3 4	2 3 4	2
2 3 4	2	1 2 3 4	1 2 3 4
2 3 4	1 2 3 4	2	1 2 3 4
2 3 4	1 2 3 4	1 2 3 4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	3 4	3 4	2
2 3 4	2	1 3 4	1 3 4
2 3 4	1 2 3 4	2	1 3 4
2 3 4	1 2 3 4	1 2 3 4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	3 4	3 4	2
2 3 4	2	1 3 4	1 3 4
2 3 4	1 2 3 4	2	1 3 4
2 3 4	1 2 3 4	1 2 3 4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	3 4	3 4	2
2 3 4	2	1 3 4	1 3 4
2 3 4	1 2 3 4	2	1 3 4
2 3 4	1 2 3 4	1 2 3 4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	3 4	3 4	2
2 3 4	2	1 3 4	1 3 4
2 3 4	1 2 3 4	2	1 3 4
2 3 4	1 2 3 4	1 2 3 4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	3 4	3 4	2
3 4	2	1 3 4	1 3 4
2 3 4	1 3 4	2	1 3 4
2 3 4	1 3 4	1 2 3 4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	3 4	3 4	2
3 4	2	1 3 4	1 3 4
2 3 4	1 3 4	2	1 3 4
2 3 4	1 3 4	1 2 3 4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	3 4	3 4	2
3 4	2	1 3 4	1 3 4
2 3 4	1 3 4	2	1 3 4
2 3 4	1 3 4	1 2 3 4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict



1	3 4	3 4	2
3 4	2	1 3 4	1 3 4
2 3 4	1 3 4	2	1 3 4
2 3 4	1 3 4	1 2 3 4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	3 4	3 4	2
3 4	2	1 3 4	1 3 4
2 3 4	1 3 4	2	1 3 4
2 3 4	1 3 4	1 2 3 4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	3 4	3 4	2
3 4	2	1 3 4	1 3 4
2 3 4	1 3 4	2	1 3 4
2 3 4	1 3 4	1 2 3 4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	3 4	3 4	2
3 4	2	1 3 4	1 3 4
2 3 4	1 3 4	2	1 3 4
2 3 4	1 3 4	1 2 3 4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	3 4	3 4	2
3 4	2	1 3 4	1 3 4
3 4	1 3 4	2	1 3 4
2 3 4	1 3 4	1 3 4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	3 4	3 4	2
3 4	2	1 3 4	1 3 4
3 4	1 3 4	2	1 3 4
2 3 4	1 3 4	1 3 4	3

`solve(problem: unary constraints)`

- `domains = init_domains()`
- `restrict_domains(problem, domains)`
- initialize `stack` as an empty stack
- while `true`:
  - if `propagate(domains)` reports no conflict:
    - if `|domains[spot]| = 1` for all `spots`, this is a consistent assignment
    - `orig_domains = copy(domains)`
    - `spot, num = search(domains)`
    - add `(spot, num, orig_domains)` to `stack`
  - otherwise:
    - if `stack` is empty, there is no consistent assignment
    - `domains = backtrack(stack)`



`propagate(domains: tile domains)`

- do until `domains` doesn't change:
  - for each `spot` on the board:
    - if `|domains[spot]| = 1`, remove `spot`'s `num` from its `peers`' domains
    - if the above caused any domain to be empty, there is a conflict

1	3 4	3 4	2
3 4	2	1 3 4	1 3 4
3 4	1 3 4	2	1 3 4
2 3 4	1 3 4	1 3 4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	3 4	3 4	2
3 4	2	1 3 4	1 3 4
3 4	1 3 4	2	1 3 4
2 3 4	1 3 4	1 3 4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict



1	3 4	3 4	2
3 4	2	1 3 4	1 3 4
3 4	1 3 4	2	1 3 4
2 3 4	1 3 4	1 3 4	3

`solve(problem: unary constraints)`

- `domains = init_domains()`
- `restrict_domains(problem, domains)`
- initialize `stack` as an empty stack
- while true:
  - if `propagate(domains)` reports no conflict:
    - if  $|\text{domains}[\text{spot}]| = 1$  for all `spots`, this is a consistent assignment
    - `orig_domains = copy(domains)`
    - `spot, num = search(domains)`
    - add `(spot, num, orig_domains)` to `stack`
  - otherwise:
    - if `stack` is empty, there is no consistent assignment
    - `domains = backtrack(stack)`



`propagate(domains: tile domains)`

- do until `domains` doesn't change:
  - for each `spot` on the board:
    - if  $|\text{domains}[\text{spot}]| = 1$ , remove `spot`'s `num` from its `peers`' domains
    - if the above caused any domain to be empty, there is a conflict

1	3 4	3 4	2
3 4	2	1 3 4	1 3 4
3 4	1 3 4	2	1 3 4
2 3 4	1 3 4	1 3 4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	3 4	3 4	2
3 4	2	1 3 4	1 3 4
3 4	1 3 4	2	1 3 4
2 3 4	1 3 4	1 3 4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	3 4	3 4	2
3 4	2	1 3 4	1 4
3 4	1 3 4	2	1 4
2 4	1 4	1 4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	3 4	3 4	2
3 4	2	1 3 4	1 4
3 4	1 3 4	2	1 4
2 4	1 4	1 4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	3 4	3 4	2
3 4	2	1 3 4	1 4
3 4	1 3 4	2	1 4
2 4	1 4	1 4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	3 4	3 4	2
3 4	2	1 3 4	1 4
3 4	1 3 4	2	1 4
2 4	1 4	1 4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	3 4	3 4	2
3 4	2	1 3 4	1 4
3 4	1 3 4	2	1 4
2 4	1 4	1 4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict



1	3 4	3 4	2
3 4	2	1 3 4	1 4
3 4	1 3 4	2	1 4
2 4	1 4	1 4	3

`solve(problem: unary constraints)`

- `domains = init_domains()`
- `restrict_domains(problem, domains)`
- initialize `stack` as an empty stack
- while `true`:
  - if `propagate(domains)` reports no conflict:
    - if  $|\text{domains}[\text{spot}]| = 1$  for all `spots`, this is a consistent assignment
    - `orig_domains = copy(domains)`
    - `spot, num = search(domains)`
    - add `(spot, num, orig_domains)` to `stack`
  - otherwise:
    - if `stack` is empty, there is no consistent assignment
    - `domains = backtrack(stack)`



`propagate(domains: tile domains)`

- do until `domains` doesn't change:
  - for each `spot` on the board:
    - if  $|\text{domains}[\text{spot}]| = 1$ , remove `spot`'s `num` from its `peers`' domains
    - if the above caused any domain to be empty, there is a conflict

1	3 4	3 4	2
3 4	2	1 3 4	1 4
3 4	1 3 4	2	1 4
2 4	1 4	1 4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	3 4	3 4	2
3 4	2	1 3 4	1 4
3 4	1 3 4	2	1 4
2 4	1 4	1 4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	3 4	3 4	2
3 4	2	1 3 4	1 4
3 4	1 3 4	2	1 4
2 4	1 4	1 4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	3 4	3 4	2
3 4	2	1 3 4	1 4
3 4	1 3 4	2	1 4
2 4	1 4	1 4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	3 4	3 4	2
3 4	2	1 3 4	1 4
3 4	1 3 4	2	1 4
2 4	1 4	1 4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	3 4	3 4	2
3 4	2	1 3 4	1 4
3 4	1 3 4	2	1 4
2 4	1 4	1 4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	3 4	3 4	2
3 4	2	1 3 4	1 4
3 4	1 3 4	2	1 4
2 4	1 4	1 4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict



1	3 4	3 4	2
3 4	2	1 3 4	1 4
3 4	1 3 4	2	1 4
2 4	1 4	1 4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	3 4	3 4	2
3 4	2	1 3 4	1 4
3 4	1 3 4	2	1 4
2 4	1 4	1 4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	3 4	3 4	2
3 4	2	1 3 4	1 4
3 4	1 3 4	2	1 4
2 4	1 4	1 4	3

`solve(problem: unary constraints)`

- `domains = init_domains()`
- `restrict_domains(problem, domains)`
- initialize `stack` as an empty stack
- while `true`:
  - if `propagate(domains)` reports no conflict:
    - if  $|\text{domains}[\text{spot}]| = 1$  for all `spots`, this is a consistent assignment
    - `orig_domains = copy(domains)`
    - `spot, num = search(domains)`
    - add `(spot, num, orig_domains)` to `stack`
  - otherwise:
    - if `stack` is empty, there is no consistent assignment
    - `domains = backtrack(stack)`



`propagate(domains: tile domains)`

- do until `domains` doesn't change:
  - for each `spot` on the board:
    - if  $|\text{domains}[\text{spot}]| = 1$ , remove `spot`'s `num` from its `peers`' domains
    - if the above caused any domain to be empty, there is a conflict

1	3 4	3 4	2
3 4	2	1 3 4	1 4
3 4	1 3 4	2	1 4
2 4	1 4	1 4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	3 4	3 4	2
3 4	2	1 3 4	1 4
3 4	1 3 4	2	1 4
2 4	1 4	1 4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	3 4	3 4	2
3 4	2	1 3 4	1 4
3 4	1 3 4	2	1 4
2 4	1 4	1 4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	3 4	3 4	2
3 4	2	1 3 4	1 4
3 4	1 3 4	2	1 4
2 4	1 4	1 4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	3 4	3 4	2
3 4	2	1 3 4	1 4
3 4	1 3 4	2	1 4
2 4	1 4	1 4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict



1	3 4	3 4	2
3 4	2	1 3 4	1 4
3 4	1 3 4	2	1 4
2 4	1 4	1 4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	3 4	3 4	2
3 4	2	1 3 4	1 4
3 4	1 3 4	2	1 4
2 4	1 4	1 4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	3 4	3 4	2
3 4	2	1 3 4	1 4
3 4	1 3 4	2	1 4
2 4	1 4	1 4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	3 4	3 4	2
3 4	2	1 3 4	1 4
3 4	1 3 4	2	1 4
2 4	1 4	1 4	3

`solve(problem: unary constraints)`

- `domains = init_domains()`
- `restrict_domains(problem, domains)`
- initialize `stack` as an empty stack
- while true:
  - if `propagate(domains)` reports no conflict:
    - if  $|\text{domains}[\text{spot}]| = 1$  for all `spots`, this is a consistent assignment
    - `orig_domains = copy(domains)`
    - `spot, num = search(domains)`
    - add `(spot, num, orig_domains)` to `stack`
  - otherwise:
    - if `stack` is empty, there is no consistent assignment
    - `domains = backtrack(stack)`



`propagate(domains: tile domains)`

- do until `domains` doesn't change:
  - for each `spot` on the board:
    - if  $|\text{domains}[\text{spot}]| = 1$ , remove `spot`'s `num` from its `peers`' domains
    - if the above caused any domain to be empty, there is a conflict

1	3 4	3 4	2
3 4	2	1 3 4	1 4
3 4	1 3 4	2	1 4
2 4	1 4	1 4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	3 4	3 4	2
3 4	2	1 3 4	1 4
3 4	1 3 4	2	1 4
2 4	1 4	1 4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	3 4	3 4	2
3 4	2	1 3 4	1 4
3 4	1 3 4	2	1 4
2 4	1 4	1 4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)

```
search(domains: tile domains)
```

- spot = an unassigned spot (according to domains)
- num = a possible num for spot (according to domains)
- restrict the domain of spot to num.

1	3 4	3 4	2
3 4	2	1 3 4	1 4
3 4	4	2	1 4
2 4	1 4	1 4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict



1	3 4	3 4	2
3 4	2	1 3 4	1 4
3 4	4	2	1 4
2 4	1 4	1 4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	3 4	3 4	2
3 4	2	1 3 4	1 4
3 4	4	2	1 4
2 4	1 4	1 4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
- for each spot on the board:
  - if |domains[spot]| = 1, remove spot's num from its peers' domains
  - if the above caused any domain to be empty, there is a conflict

1	3 4	3 4	2
3 4	2	1 3 4	1 4
3 4	4	2	1 4
2 4	1 4	1 4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	3 4	3 4	2
3 4	2	1 3 4	1 4
3 4	4	2	1 4
2 4	1 4	1 4	3

`solve(problem: unary constraints)`

- `domains = init_domains()`
- `restrict_domains(problem, domains)`
- initialize `stack` as an empty stack
- while `true`:
  - if `propagate(domains)` reports no conflict:
    - if `|domains[spot]| = 1` for all `spots`, this is a consistent assignment
    - `orig_domains = copy(domains)`
    - `spot, num = search(domains)`
    - add `(spot, num, orig_domains)` to `stack`
  - otherwise:
    - if `stack` is empty, there is no consistent assignment
    - `domains = backtrack(stack)`



`propagate(domains: tile domains)`

- do until `domains` doesn't change:
  - for each `spot` on the board:
    - if `|domains[spot]| = 1`, remove `spot`'s `num` from its `peers`' domains
    - if the above caused any domain to be empty, there is a conflict

1	3 4	3 4	2
3 4	2	1 3 4	1 4
3 4	4	2	1 4
2 4	1 4	1 4	3

`solve(problem: unary constraints)`

- `domains = init_domains()`
- `restrict_domains(problem, domains)`
- initialize `stack` as an empty stack
- while `true`:
  - if `propagate(domains)` reports no conflict:
    - if `|domains[spot]| = 1` for all `spots`, this is a consistent assignment
    - `orig_domains = copy(domains)`
    - `spot, num = search(domains)`
    - add `(spot, num, orig_domains)` to `stack`
  - otherwise:
    - if `stack` is empty, there is no consistent assignment
    - `domains = backtrack(stack)`



`propagate(domains: tile domains)`

- do until `domains` doesn't change:
  - for each `spot` on the board:
    - if `|domains[spot]| = 1`, remove `spot`'s `num` from its `peers`' domains
    - if the above caused any domain to be empty, there is a conflict

1	3 4	3 4	2
3 4	2	1 3 4	1 4
3 4	4	2	1 4
2 4	1 4	1 4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	3 4	3 4	2
3 4	2	1 3 4	1 4
3 4	4	2	1 4
2 4	1 4	1 4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	3 4	3 4	2
3 4	2	1 3 4	1 4
3 4	4	2	1 4
2 4	1 4	1 4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict



1	3 4	3 4	2
3 4	2	1 3 4	1 4
3 4	4	2	1 4
2 4	1 4	1 4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	3 4	3 4	2
3 4	2	1 3 4	1 4
3 4	4	2	1 4
2 4	1 4	1 4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	3 4	3 4	2
3 4	2	1 3 4	1 4
3 4	4	2	1 4
2 4	1 4	1 4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	3 4	3 4	2
3 4	2	1 3 4	1 4
3 4	4	2	1 4
2 4	1 4	1 4	3

`solve(problem: unary constraints)`

- `domains = init_domains()`
- `restrict_domains(problem, domains)`
- initialize `stack` as an empty stack
- while `true`:
  - if `propagate(domains)` reports no conflict:
    - if `|domains[spot]| = 1` for all `spots`, this is a consistent assignment
    - `orig_domains = copy(domains)`
    - `spot, num = search(domains)`
    - add `(spot, num, orig_domains)` to `stack`
  - otherwise:
    - if `stack` is empty, there is no consistent assignment
    - `domains = backtrack(stack)`



`propagate(domains: tile domains)`

- do until `domains` doesn't change:
  - for each `spot` on the board:
    - if `|domains[spot]| = 1`, remove `spot`'s `num` from its `peers`' domains
    - if the above caused any domain to be empty, there is a conflict

1	3 4	3 4	2
3 4	2	1 3 4	1 4
3 4	4	2	1 4
2 4	1 4	1 4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	3 4	3 4	2
3 4	2	1 3 4	1 4
3 4	4	2	1 4
2 4	1 4	1 4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	3 4	3 4	2
3 4	2	1 3 4	1 4
3 4	4	2	1 4
2 4	1 4	1 4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	3 4	3 4	2
3 4	2	1 3 4	1 4
3 4	4	2	1 4
2 4	1 4	1 4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict



1	3	3 4	2
3 4	2	1 3 4	1 4
3	4	2	1
2	1	1 4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	3	3 4	2
3 4	2	1 3 4	1 4
3	4	2	1
2	1	1 4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	3	3 4	2
3 4	2	1 3 4	1 4
3	4	2	1
2	1	1 4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	3	3 4	2
3 4	2	1 3 4	1 4
3	4	2	1
2	1	1 4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	3	3 4	2
3 4	2	1 3 4	1 4
3	4	2	1
2	1	1 4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	3	3 4	2
3 4	2	1 3 4	1 4
3	4	2	1
2	1	1 4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	3	3 4	2
3 4	2	1 3 4	4
3	4	2	1
2	1	4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	3	3 4	2
3 4	2	1 3 4	4
3	4	2	1
2	1	4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict



1	3	3 4	2
3 4	2	1 3 4	4
3	4	2	1
2	1	4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	3	3 4	2
3 4	2	1 3 4	4
3	4	2	1
2	1	4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	3	3 4	2
3 4	2	1 3 4	4
3	4	2	1
2	1	4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	3	3 4	2
3 4	2	1 3 4	4
3	4	2	1
2	1	4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	3	3 4	2
3 4	2	1 3 4	4
3	4	2	1
2	1	4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	3	3 4	2
3 4	2	1 3 4	4
3	4	2	1
2	1	4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	3	3 4	2
3 4	2	1 3 4	4
3	4	2	1
2	1	4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	3	3	2
3 4	2	1 3	4
3	4	2	1
2	1	4	3

`solve(problem: unary constraints)`

- `domains = init_domains()`
- `restrict_domains(problem, domains)`
- initialize `stack` as an empty stack
- while `true`:
  - if `propagate(domains)` reports no conflict:
    - if `|domains[spot]| = 1` for all `spots`, this is a consistent assignment
    - `orig_domains = copy(domains)`
    - `spot, num = search(domains)`
    - add `(spot, num, orig_domains)` to `stack`
  - otherwise:
    - if `stack` is empty, there is no consistent assignment
    - `domains = backtrack(stack)`



`propagate(domains: tile domains)`

- do until `domains` doesn't change:
  - for each `spot` on the board:
    - if `|domains[spot]| = 1`, remove `spot's num` from its `peers' domains`
    - if the above caused any domain to be empty, there is a conflict



1	3	3	2
3 4	2	1 3	4
3	4	2	1
2	1	4	3

`solve(problem: unary constraints)`

- `domains = init_domains()`
- `restrict_domains(problem, domains)`
- initialize `stack` as an empty stack
- while `true`:
  - if `propagate(domains)` reports no conflict:
    - if `|domains[spot]| = 1` for all `spots`, this is a consistent assignment
    - `orig_domains = copy(domains)`
    - `spot, num = search(domains)`
    - add `(spot, num, orig_domains)` to `stack`
  - otherwise:
    - if `stack` is empty, there is no consistent assignment
    - `domains = backtrack(stack)`



`propagate(domains: tile domains)`

- do until `domains` doesn't change:
  - for each `spot` on the board:
    - if `|domains[spot]| = 1`, remove `spot`'s `num` from its `peers`' domains
    - if the above caused any domain to be empty, there is a conflict

1	3	3	2
3 4	2	1 3	4
3	4	2	1
2	1	4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	3	3	2
3 4	2	1 3	4
3	4	2	1
2	1	4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	3	3	2
3 4	2	1 3	4
3	4	2	1
2	1	4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	3	3	2
3 4	2	1 3	4
3	4	2	1
2	1	4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	3	3	2
3 4	2	1 3	4
3	4	2	1
2	1	4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	3	3	2
3 4	2	1 3	4
3	4	2	1
2	1	4	3

`solve(problem: unary constraints)`

- `domains = init_domains()`
- `restrict_domains(problem, domains)`
- initialize `stack` as an empty stack
- while true:
  - if `propagate(domains)` reports no conflict:
    - if `|domains[spot]| = 1` for all `spots`, this is a consistent assignment
    - `orig_domains = copy(domains)`
    - `spot, num = search(domains)`
    - add `(spot, num, orig_domains)` to `stack`
  - otherwise:
    - if `stack` is empty, there is no consistent assignment
    - `domains = backtrack(stack)`



`propagate(domains: tile domains)`

- do until `domains` doesn't change:
  - for each `spot` on the board:
    - if `|domains[spot]| = 1`, remove `spot's num` from its `peers' domains`
    - if the above caused any domain to be empty, there is a conflict

1	3	3	2
3 4	2	1 3	4
3	4	2	1
2	1	4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict



1	3		2
3 4	2	1 3	4
3	4	2	1
2	1	4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	3 4	3 4	2
3 4	2	1 3 4	1 4
3 4	1 3 4	2	1 4
2 4	1 4	1 4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
backtrack(stack: history stack, domains: domains)
```

- pop (spot, num, orig\_domains) from stack
- replace domains with orig\_domains
- remove num from the domain of spot

1	3 4	3 4	2
3 4	2	1 3 4	1 4
3 4	1 3	2	1 4
2 4	1 4	1 4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
backtrack(stack: history stack, domains: domains)
```

- pop (spot, num, orig\_domains) from stack
- replace domains with orig\_domains
- remove num from the domain of spot

1	3 4	3 4	2
3 4	2	1 3 4	1 4
3 4	1 3	2	1 4
2 4	1 4	1 4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	3 4	3 4	2
3 4	2	1 3 4	1 4
3 4	1 3	2	1 4
2 4	1 4	1 4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	3 4	3 4	2
3 4	2	1 3 4	1 4
3 4	1 3	2	1 4
2 4	1 4	1 4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	3 4	3 4	2
3 4	2	1 3 4	1 4
3 4	1 3	2	1 4
2 4	1 4	1 4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	3 4	3 4	2
3 4	2	1 3 4	1 4
3 4	1 3	2	1 4
2 4	1 4	1 4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict



1	3 4	3 4	2
3 4	2	1 3 4	1 4
3 4	1 3	2	1 4
2 4	1 4	1 4	3

`solve(problem: unary constraints)`

- `domains = init_domains()`
- `restrict_domains(problem, domains)`
- initialize `stack` as an empty stack
- while true:
  - if `propagate(domains)` reports no conflict:
    - if `|domains[spot]| = 1` for all `spots`, this is a consistent assignment
    - `orig_domains = copy(domains)`
    - `spot, num = search(domains)`
    - add `(spot, num, orig_domains)` to `stack`
  - otherwise:
    - if `stack` is empty, there is no consistent assignment
    - `domains = backtrack(stack)`



`propagate(domains: tile domains)`

- do until `domains` doesn't change:
  - for each `spot` on the board:
    - if `|domains[spot]| = 1`, remove `spot's num` from its `peers' domains`
    - if the above caused any domain to be empty, there is a conflict

1	3 4	3 4	2
3 4	2	1 3 4	1 4
3 4	1 3	2	1 4
2 4	1 4	1 4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	3 4	3 4	2
3 4	2	1 3 4	1 4
3 4	1 3	2	1 4
2 4	1 4	1 4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	3 4	3 4	2
3 4	2	1 3 4	1 4
3 4	1 3	2	1 4
2 4	1 4	1 4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	3 4	3 4	2
3 4	2	1 3 4	1 4
3 4	1 3	2	1 4
2 4	1 4	1 4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	3 4	3 4	2
3 4	2	1 3 4	1 4
3 4	1 3	2	1 4
2 4	1 4	1 4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	3 4	3 4	2
3 4	2	1 3 4	1 4
3 4	1 3	2	1 4
2 4	1 4	1 4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	3 4	3 4	2
3 4	2	1 3 4	1 4
3 4	1 3	2	1 4
2 4	1 4	1 4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict



1	3 4	3 4	2
3 4	2	1 3 4	1 4
3 4	1 3	2	1 4
2 4	1 4	1 4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	3 4	3 4	2
3 4	2	1 3 4	1 4
3 4 1 3	2	1 4	3
2 4	1 4	1 4	3

`solve(problem: unary constraints)`

- `domains = init_domains()`
- `restrict_domains(problem, domains)`
- initialize `stack` as an empty stack
- while `true`:
  - if `propagate(domains)` reports no conflict:
    - if  $|\text{domains}[\text{spot}]| = 1$  for all `spots`, this is a consistent assignment
    - `orig_domains = copy(domains)`
    - `spot, num = search(domains)`
    - add `(spot, num, orig_domains)` to `stack`
  - otherwise:
    - if `stack` is empty, there is no consistent assignment
    - `domains = backtrack(stack)`



`propagate(domains: tile domains)`

- do until `domains` doesn't change:
  - for each `spot` on the board:
    - if  $|\text{domains}[\text{spot}]| = 1$ , remove `spot`'s `num` from its `peers`' domains
    - if the above caused any domain to be empty, there is a conflict

1	3 4	3 4	2
3 4	2	1 3 4	1 4
3 4	1 3	2	1 4
2 4	1 4	1 4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	3 4	3 4	2
3 4	2	1 3 4	1 4
3 4	1 3	2	1 4
2 4	1 4	1 4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	3 4	3 4	2
3 4	2	1 3 4	1 4
3 4	1 3	2	1 4
2 4	1 4	1 4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	3 4	3 4	2
3 4	2	1 3 4	1 4
3 4	1 3	2	1 4
2 4	1 4	1 4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	3 4	3 4	2
3 4	2	1 3 4	1 4
3 4	1 3	2	1 4
2 4	1 4	1 4	3

`solve(problem: unary constraints)`

- `domains = init_domains()`
- `restrict_domains(problem, domains)`
- initialize `stack` as an empty stack
- while `true`:
  - if `propagate(domains)` reports no conflict:
    - if  $|\text{domains}[\text{spot}]| = 1$  for all `spots`, this is a consistent assignment
    - `orig_domains = copy(domains)`
    - `spot, num = search(domains)`
    - add `(spot, num, orig_domains)` to `stack`
  - otherwise:
    - if `stack` is empty, there is no consistent assignment
    - `domains = backtrack(stack)`



`propagate(domains: tile domains)`

- do until `domains` doesn't change:
  - for each `spot` on the board:
    - if  $|\text{domains}[\text{spot}]| = 1$ , remove `spot`'s `num` from its `peers`' domains
    - if the above caused any domain to be empty, there is a conflict

1	3 4	3 4	2
3 4	2	1 3 4	1 4
3 4	1 3	2	1 4
2 4	1 4	1 4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict



1	3 4	3 4	2
3 4	2	1 3 4	1 4
3 4	1 3	2	1 4
2 4	1 4	1 4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	3 4	3 4	2
3 4	2	1 3 4	1 4
3 4	1 3	2	1 4
2 4	1 4	1 4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	3 4	3 4	2
3 4	2	1 3 4	1 4
3 4	1 3	2	1 4
2 4	1 4	1 4	3

`solve(problem: unary constraints)`

- `domains = init_domains()`
- `restrict_domains(problem, domains)`
- initialize `stack` as an empty stack
- while `true`:
  - if `propagate(domains)` reports no conflict:
    - if `|domains[spot]| = 1` for all `spots`, this is a consistent assignment
    - `orig_domains = copy(domains)`
    - `spot, num = search(domains)`
    - add `(spot, num, orig_domains)` to `stack`
  - otherwise:
    - if `stack` is empty, there is no consistent assignment
    - `domains = backtrack(stack)`



`propagate(domains: tile domains)`

- do until `domains` doesn't change:
  - for each `spot` on the board:
    - if `|domains[spot]| = 1`, remove `spot`'s `num` from its `peers`' domains
    - if the above caused any domain to be empty, there is a conflict

1	3 4	3 4	2
3 4	2	1 3 4	1 4
3 4	1 3	2	1 4
2 4	1 4	1 4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	3 4	3 4	2
3 4	2	1 3 4	1 4
3 4	1 3	2	1 4
2 4	1 4	1 4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	3 4	3 4	2
3 4	2	1 3 4	1 4
3 4	1 3	2	1 4
2 4	1 4	1 4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)

```
search(domains: tile domains)
```

- spot = an unassigned spot (according to domains)
- num = a possible num for spot (according to domains)
- restrict the domain of spot to num.

1	3 4	3 4	2
3 4	2	1 3 4	1 4
3 4	3	2	1 4
2 4	1 4	1 4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	3 4	3 4	2
3 4	2	1 3 4	1 4
3 4	3	2	1 4
2 4	1 4	1 4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict



1	3 4	3 4	2
3 4	2	1 3 4	1 4
3 4	3	2	1 4
2 4	1 4	1 4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	3 4	3 4	2
3 4	2	1 3 4	1 4
3 4	3	2	1 4
2 4	1 4	1 4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	3 4	3 4	2
3 4	2	1 3 4	1 4
3 4	3	2	1 4
2 4	1 4	1 4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	3 4	3 4	2
3 4	2	1 3 4	1 4
3 4	3	2	1 4
2 4	1 4	1 4	3

`solve(problem: unary constraints)`

- `domains = init_domains()`
- `restrict_domains(problem, domains)`
- initialize `stack` as an empty stack
- while `true`:
  - if `propagate(domains)` reports no conflict:
    - if  $|\text{domains}[\text{spot}]| = 1$  for all `spots`, this is a consistent assignment
    - `orig_domains = copy(domains)`
    - `spot, num = search(domains)`
    - add `(spot, num, orig_domains)` to `stack`
  - otherwise:
    - if `stack` is empty, there is no consistent assignment
    - `domains = backtrack(stack)`



`propagate(domains: tile domains)`

- do until `domains` doesn't change:
  - for each `spot` on the board:
    - if  $|\text{domains}[\text{spot}]| = 1$ , remove `spot`'s `num` from its `peers`' domains
    - if the above caused any domain to be empty, there is a conflict

1	3 4	3 4	2
3 4	2	1 3 4	1 4
3 4	3	2	1 4
2 4	1 4	1 4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	3 4	3 4	2
3 4	2	1 3 4	1 4
3 4	3	2	1 4
2 4	1 4	1 4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	3 4	3 4	2
3 4	2	1 3 4	1 4
3 4	3	2	1 4
2 4	1 4	1 4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	3 4	3 4	2
3 4	2	1 3 4	1 4
3 4	3	2	1 4
2 4	1 4	1 4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict



1	3 4	3 4	2
3 4	2	1 3 4	1 4
3 4	3	2	1 4
2 4	1 4	1 4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	3 4	3 4	2
3 4	2	1 3 4	1 4
3 4	3	2	1 4
2 4	1 4	1 4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	3 4	3 4	2
3 4	2	1 3 4	1 4
3 4	3	2	1 4
2 4	1 4	1 4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	3 4	3 4	2
3 4	2	1 3 4	1 4
3 4	3	2	1 4
2 4	1 4	1 4	3

`solve(problem: unary constraints)`

- `domains = init_domains()`
- `restrict_domains(problem, domains)`
- initialize `stack` as an empty stack
- while `true`:
  - if `propagate(domains)` reports no conflict:
    - if `|domains[spot]| = 1` for all `spots`, this is a consistent assignment
    - `orig_domains = copy(domains)`
    - `spot, num = search(domains)`
    - add `(spot, num, orig_domains)` to `stack`
  - otherwise:
    - if `stack` is empty, there is no consistent assignment
    - `domains = backtrack(stack)`



`propagate(domains: tile domains)`

- do until `domains` doesn't change:
  - for each `spot` on the board:
    - if `|domains[spot]| = 1`, remove `spot`'s `num` from its `peers`' domains
    - if the above caused any domain to be empty, there is a conflict

1	3 4	3 4	2
3 4	2	1 3 4	1 4
3 4	3	2	1 4
2 4	1 4	1 4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	3 4	3 4	2
3 4	2	1 3 4	1 4
3 4	3	2	1 4
2 4	1 4	1 4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	3 4	3 4	2
3 4	2	1 3 4	1 4
3 4	3	2	1 4
2 4	1 4	1 4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	4	3 4	2
3 4	2	1 3 4	1 4
4	3	2	1 4
2 4	1 4	1 4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict



1	4	3 4	2
3 4	2	1 3 4	1 4
4	3	2	1 4
2 4	1 4	1 4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	4	3 4	2
3 4	2	1 3 4	1 4
4	3	2	1 4
2 4	1 4	1 4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	4	3 4	2
3 4	2	1 3 4	1 4
4	3	2	1 4
2 4	1 4	1 4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	4	3 4	2
3 4	2	1 3 4	1 4
4	3	2	1 4
2 4	1 4	1 4	3

`solve(problem: unary constraints)`

- `domains = init_domains()`
- `restrict_domains(problem, domains)`
- initialize `stack` as an empty stack
- while `true`:
  - if `propagate(domains)` reports no conflict:
    - if  $|\text{domains}[\text{spot}]| = 1$  for all `spots`, this is a consistent assignment
    - `orig_domains = copy(domains)`
    - `spot, num = search(domains)`
    - add `(spot, num, orig_domains)` to `stack`
  - otherwise:
    - if `stack` is empty, there is no consistent assignment
    - `domains = backtrack(stack)`



`propagate(domains: tile domains)`

- do until `domains` doesn't change:
  - for each `spot` on the board:
    - if  $|\text{domains}[\text{spot}]| = 1$ , remove `spot`'s `num` from its `peers`' domains
    - if the above caused any domain to be empty, there is a conflict

1	4	3 4	2
3 4	2	1 3 4	1 4
4	3	2	1 4
2 4	1 4	1 4	3

`solve(problem: unary constraints)`

- `domains = init_domains()`
- `restrict_domains(problem, domains)`
- initialize `stack` as an empty stack
- while `true`:
  - if `propagate(domains)` reports no conflict:
    - if `|domains[spot]| = 1` for all `spots`, this is a consistent assignment
    - `orig_domains = copy(domains)`
    - `spot, num = search(domains)`
    - add `(spot, num, orig_domains)` to `stack`
  - otherwise:
    - if `stack` is empty, there is no consistent assignment
    - `domains = backtrack(stack)`



`propagate(domains: tile domains)`

- do until `domains` doesn't change:
  - for each `spot` on the board:
    - if `|domains[spot]| = 1`, remove `spot`'s `num` from its `peers`' domains
    - if the above caused any domain to be empty, there is a conflict

1	4	3 4	2
3 4	2	1 3 4	1 4
4	3	2	1 4
2 4	1 4	1 4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	4	3 4	2
3 4	2	1 3 4	1 4
4	3	2	1 4
2 4	1 4	1 4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	4	3 4	2
3 4	2	1 3 4	1 4
4	3	2	1 4
2 4	1 4	1 4	3

`solve(problem: unary constraints)`

- `domains = init_domains()`
- `restrict_domains(problem, domains)`
- initialize `stack` as an empty stack
- while `true`:
  - if `propagate(domains)` reports no conflict:
    - if  $|\text{domains}[\text{spot}]| = 1$  for all `spots`, this is a consistent assignment
    - `orig_domains = copy(domains)`
    - `spot, num = search(domains)`
    - add `(spot, num, orig_domains)` to `stack`
  - otherwise:
    - if `stack` is empty, there is no consistent assignment
    - `domains = backtrack(stack)`



`propagate(domains: tile domains)`

- do until `domains` doesn't change:
  - for each `spot` on the board:
    - if  $|\text{domains}[\text{spot}]| = 1$ , remove `spot`'s `num` from its `peers`' domains
    - if the above caused any domain to be empty, there is a conflict



1	4	3 4	2
3 4	2	1 3 4	1 4
4	3	2	1 4
2 4	1 4	1 4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	4	3 4	2
3 4	2	1 3 4	1 4
4	3	2	1 4
2 4	1 4	1 4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	4	3 4	2
3 4	2	1 3 4	1 4
4	3	2	1 4
2 4	1 4	1 4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	4	3 4	2
3 4	2	1 3 4	1 4
4	3	2	1 4
2 4	1 4	1 4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	4	3 4	2
3 4	2	1 3 4	1 4
4	3	2	1 4
2 4	1 4	1 4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	4	3 4	2
3 4	2	1 3 4	1 4
4	3	2	1 4
2 4	1 4	1 4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	4	3 4	2
3 4	2	1 3 4	1 4
4	3	2	1 4
2 4	1 4	1 4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	4	3	2
3	2	1 3 4	1 4
4	3	2	1 4
2 4	1	1 4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict



1	4	3	2
3	2	1 3 4	1 4
4	3	2	1 4
2 4	1	1 4	3

`solve(problem: unary constraints)`

- `domains = init_domains()`
- `restrict_domains(problem, domains)`
- initialize `stack` as an empty stack
- while `true`:
  - if `propagate(domains)` reports no conflict:
    - if `|domains[spot]| = 1` for all `spots`, this is a consistent assignment
    - `orig_domains = copy(domains)`
    - `spot, num = search(domains)`
    - add `(spot, num, orig_domains)` to `stack`
  - otherwise:
    - if `stack` is empty, there is no consistent assignment
    - `domains = backtrack(stack)`



`propagate(domains: tile domains)`

- do until `domains` doesn't change:
  - for each `spot` on the board:
    - if `|domains[spot]| = 1`, remove `spot`'s `num` from its `peers`' domains
    - if the above caused any domain to be empty, there is a conflict

1	4	3	2
3	2	1 3 4	1 4
4	3	2	1 4
2 4	1	1 4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	4	3	2
3	2	1 4	1 4
4	3	2	1 4
2 4 1		1 4	3

`solve(problem: unary constraints)`

- `domains = init_domains()`
- `restrict_domains(problem, domains)`
- initialize `stack` as an empty stack
- while `true`:
  - if `propagate(domains)` reports no conflict:
    - if `|domains[spot]| = 1` for all `spots`, this is a consistent assignment
    - `orig_domains = copy(domains)`
    - `spot, num = search(domains)`
    - add `(spot, num, orig_domains)` to `stack`
  - otherwise:
    - if `stack` is empty, there is no consistent assignment
    - `domains = backtrack(stack)`



`propagate(domains: tile domains)`

- do until `domains` doesn't change:
- for each `spot` on the board:
  - if `|domains[spot]| = 1`, remove `spot's num` from its `peers' domains`
  - if the above caused any domain to be empty, there is a conflict

1	4	3	2
3	2	1 4	1 4
4	3	2	1 4
2 4	1	1 4	3

`solve(problem: unary constraints)`

- `domains = init_domains()`
- `restrict_domains(problem, domains)`
- initialize `stack` as an empty stack
- while true:
  - if `propagate(domains)` reports no conflict:
    - if  $|\text{domains}[\text{spot}]| = 1$  for all `spots`, this is a consistent assignment
    - `orig_domains = copy(domains)`
    - `spot, num = search(domains)`
    - add `(spot, num, orig_domains)` to `stack`
  - otherwise:
    - if `stack` is empty, there is no consistent assignment
    - `domains = backtrack(stack)`



`propagate(domains: tile domains)`

- do until `domains` doesn't change:
  - for each `spot` on the board:
    - if  $|\text{domains}[\text{spot}]| = 1$ , remove `spot`'s `num` from its `peers`' domains
    - if the above caused any domain to be empty, there is a conflict

1	4	3	2
3	2	1 4	1 4
4	3	2	1 4
2 4	1	1 4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	4	3	2
3	2	1 4	1 4
4	3	2	1 4
2 4	1	1 4	3

`solve(problem: unary constraints)`

- `domains = init_domains()`
- `restrict_domains(problem, domains)`
- initialize `stack` as an empty stack
- while `true`:
  - if `propagate(domains)` reports no conflict:
    - if `|domains[spot]| = 1` for all `spots`, this is a consistent assignment
    - `orig_domains = copy(domains)`
    - `spot, num = search(domains)`
    - add `(spot, num, orig_domains)` to `stack`
  - otherwise:
    - if `stack` is empty, there is no consistent assignment
    - `domains = backtrack(stack)`



`propagate(domains: tile domains)`

- do until `domains` doesn't change:
- for each `spot` on the board:
  - if `|domains[spot]| = 1`, remove `spot's num` from its `peers' domains`
  - if the above caused any domain to be empty, there is a conflict

1	4	3	2
3	2	1 4	1 4
4	3	2	1 4
2 4	1	1 4	3

`solve(problem: unary constraints)`

- `domains = init_domains()`
- `restrict_domains(problem, domains)`
- initialize `stack` as an empty stack
- while `true`:
  - if `propagate(domains)` reports no conflict:
    - if  $|\text{domains}[\text{spot}]| = 1$  for all `spots`, this is a consistent assignment
    - `orig_domains = copy(domains)`
    - `spot, num = search(domains)`
    - add `(spot, num, orig_domains)` to `stack`
  - otherwise:
    - if `stack` is empty, there is no consistent assignment
    - `domains = backtrack(stack)`



`propagate(domains: tile domains)`

- do until `domains` doesn't change:
  - for each `spot` on the board:
    - if  $|\text{domains}[\text{spot}]| = 1$ , remove `spot`'s `num` from its `peers`' domains
    - if the above caused any domain to be empty, there is a conflict

1	4	3	2
3	2	1 4	1 4
4	3	2	1 4
2 4	1	1 4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict



1	4	3	2
3	2	1 4	1 4
4	3	2	1 4
2 4	1	1 4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	4	3	2
3	2	1 4	1 4
4	3	2	1 4
2 4	1	1 4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	4	3	2
3	2	1 4	1 4
4	3	2	1 4
2 4	1	1 4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	4	3	2
3	2	1 4	1 4
4	3	2	1 4
2 4	1	1 4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	4	3	2
3	2	1 4	1 4
4	3	2	1 4
2 4 1		1 4	3

`solve(problem: unary constraints)`

- `domains = init_domains()`
- `restrict_domains(problem, domains)`
- initialize `stack` as an empty stack
- while `true`:
  - if `propagate(domains)` reports no conflict:
    - if  $|\text{domains}[\text{spot}]| = 1$  for all `spots`, this is a consistent assignment
    - `orig_domains = copy(domains)`
    - `spot, num = search(domains)`
    - add `(spot, num, orig_domains)` to `stack`
  - otherwise:
    - if `stack` is empty, there is no consistent assignment
    - `domains = backtrack(stack)`



`propagate(domains: tile domains)`

- do until `domains` doesn't change:
  - for each `spot` on the board:
    - if  $|\text{domains}[\text{spot}]| = 1$ , remove `spot`'s `num` from its `peers`' domains
    - if the above caused any domain to be empty, there is a conflict

1	4	3	2
3	2	1 4	1 4
4	3	2	1 4
2 4	1	1 4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	4	3	2
3	2	1 4	1 4
4	3	2	1 4
2 4	1	1 4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	4	3	2
3	2	1 4	1 4
4 3	2	1 4	1 4
2 4	1		3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict



1	4	3	2
3	2	1 4	1 4
4	3	2	1
2	1	1 4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	4	3	2
3	2	1 4	1 4
4	3	2	1
2	1	1 4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	4	3	2
3	2	1 4	1 4
4	3	2	1
2	1	1 4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	4	3	2
3	2	1 4	1 4
4	3	2	1
2	1	1 4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	4	3	2
3	2	1 4	1 4
4	3	2	1
2	1	1 4	3

`solve(problem: unary constraints)`

- `domains = init_domains()`
- `restrict_domains(problem, domains)`
- initialize `stack` as an empty stack
- while `true`:
  - if `propagate(domains)` reports no conflict:
    - if  $|\text{domains}[\text{spot}]| = 1$  for all `spots`, this is a consistent assignment
    - `orig_domains = copy(domains)`
    - `spot, num = search(domains)`
    - add `(spot, num, orig_domains)` to `stack`
  - otherwise:
    - if `stack` is empty, there is no consistent assignment
    - `domains = backtrack(stack)`



`propagate(domains: tile domains)`

- do until `domains` doesn't change:
  - for each `spot` on the board:
    - if  $|\text{domains}[\text{spot}]| = 1$ , remove `spot`'s `num` from its `peers`' domains
    - if the above caused any domain to be empty, there is a conflict

1	4	3	2
3	2	1 4	1 4
4	3	2	1
2	1	1 4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	4	3	2
3	2	1 4	1 4
4	3	2	1
2	1	1 4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	4	3	2
3	2	1 4	1 4
4	3	2	1
2	1	1 4	3

`solve(problem: unary constraints)`

- `domains = init_domains()`
- `restrict_domains(problem, domains)`
- initialize `stack` as an empty stack
- while `true`:
  - if `propagate(domains)` reports no conflict:
    - if  $|\text{domains}[\text{spot}]| = 1$  for all `spots`, this is a consistent assignment
    - `orig_domains = copy(domains)`
    - `spot, num = search(domains)`
    - add `(spot, num, orig_domains)` to `stack`
  - otherwise:
    - if `stack` is empty, there is no consistent assignment
    - `domains = backtrack(stack)`



`propagate(domains: tile domains)`

- do until `domains` doesn't change:
  - for each `spot` on the board:
    - if  $|\text{domains}[\text{spot}]| = 1$ , remove `spot`'s `num` from its `peers`' domains
    - if the above caused any domain to be empty, there is a conflict



1	4	3	2
3	2	1 4	1 4
4	3	2	1
2	1	1 4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	4	3	2
3	2	1 4	4
4	3	2	1
2	1	4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	4	3	2
3	2	1 4	4
4	3	2	1
2	1	4	3

`solve(problem: unary constraints)`

- `domains = init_domains()`
- `restrict_domains(problem, domains)`
- initialize `stack` as an empty stack
- while `true`:
  - if `propagate(domains)` reports no conflict:
    - if  $|\text{domains}[\text{spot}]| = 1$  for all `spots`, this is a consistent assignment
    - `orig_domains = copy(domains)`
    - `spot, num = search(domains)`
    - add `(spot, num, orig_domains)` to `stack`
  - otherwise:
    - if `stack` is empty, there is no consistent assignment
    - `domains = backtrack(stack)`



`propagate(domains: tile domains)`

- do until `domains` doesn't change:
  - for each `spot` on the board:
    - if  $|\text{domains}[\text{spot}]| = 1$ , remove `spot`'s `num` from its `peers`' domains
    - if the above caused any domain to be empty, there is a conflict

1	4	3	2
3	2	1 4	4
4	3	2	1
2	1	4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	4	3	2
3	2	1 4	4
4	3	2	1
2	1	4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	4	3	2
3	2	1 4	4
4	3	2	1
2	1	4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	4	3	2
3	2	1 4	4
4	3	2	1
2	1	4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	4	3	2
3	2	1 4	4
4	3	2	1
2	1	4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict



1	4	3	2
3	2	1 4	4
4	3	2	1
2	1	4	3

`solve(problem: unary constraints)`

- `domains = init_domains()`
- `restrict_domains(problem, domains)`
- initialize `stack` as an empty stack
- while true:
  - if `propagate(domains)` reports no conflict:
    - if  $|\text{domains}[\text{spot}]| = 1$  for all `spots`, this is a consistent assignment
    - `orig_domains = copy(domains)`
    - `spot, num = search(domains)`
    - add `(spot, num, orig_domains)` to `stack`
  - otherwise:
    - if `stack` is empty, there is no consistent assignment
    - `domains = backtrack(stack)`



`propagate(domains: tile domains)`

- do until `domains` doesn't change:
  - for each `spot` on the board:
    - if  $|\text{domains}[\text{spot}]| = 1$ , remove `spot`'s `num` from its `peers`' domains
    - if the above caused any domain to be empty, there is a conflict

1	4	3	2
3	2	1 4	4
4	3	2	1
2	1	4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	4	3	2
3	2	1	4
4	3	2	1
2	1	4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	4	3	2
3	2	1	4
4	3	2	1
2	1	4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	4	3	2
3	2	1	4
4	3	2	1
2	1	4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	4	3	2
3	2	1	4
4	3	2	1
2	1	4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	4	3	2
3	2	1	4
4	3	2	1
2	1	4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	4	3	2
3	2	1	4
4	3	2	1
2	1	4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict



1	4	3	2
3	2	1	4
4	3	2	1
2	1	4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	4	3	2
3	2	1	4
4	3	2	1
2	1	4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	4	3	2
3	2	1	4
4	3	2	1
2	1	4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	4	3	2
3	2	1	4
4	3	2	1
2	1	4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	4	3	2
3	2	1	4
4	3	2	1
2	1	4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	4	3	2
3	2	1	4
4	3	2	1
2	1	4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	4	3	2
3	2	1	4
4	3	2	1
2	1	4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	4	3	2
3	2	1	4
4	3	2	1
2	1	4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict



1	4	3	2
3	2	1	4
4	3	2	1
2	1	4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	4	3	2
3	2	1	4
4	3	2	1
2	1	4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	4	3	2
3	2	1	4
4	3	2	1
2	1	4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	4	3	2
3	2	1	4
4	3	2	1
2	1	4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	4	3	2
3	2	1	4
4	3	2	1
2	1	4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	4	3	2
3	2	1	4
4	3	2	1
2	1	4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	4	3	2
3	2	1	4
4	3	2	1
2	1	4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	4	3	2
3	2	1	4
4	3	2	1
2	1	4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict



1	4	3	2
3	2	1	4
4	3	2	1
2	1	4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	4	3	2
3	2	1	4
4	3	2	1
2	1	4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	4	3	2
3	2	1	4
4	3	2	1
2	1	4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	4	3	2
3	2	1	4
4	3	2	1
2	1	4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	4	3	2
3	2	1	4
4	3	2	1
2	1	4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	4	3	2
3	2	1	4
4	3	2	1
2	1	4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	4	3	2
3	2	1	4
4	3	2	1
2	1	4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	4	3	2
3	2	1	4
4	3	2	1
2	1	4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict



1	4	3	2
3	2	1	4
4	3	2	1
2	1	4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	4	3	2
3	2	1	4
4	3	2	1
2	1	4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	4	3	2
3	2	1	4
4	3	2	1
2	1	4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	4	3	2
3	2	1	4
4	3	2	1
2	1	4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	4	3	2
3	2	1	4
4	3	2	1
2	1	4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	4	3	2
3	2	1	4
4	3	2	1
2	1	4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	4	3	2
3	2	1	4
4	3	2	1
2	1	4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	4	3	2
3	2	1	4
4	3	2	1
2	1	4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict



1	4	3	2
3	2	1	4
4	3	2	1
2	1	4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	4	3	2
3	2	1	4
4	3	2	1
2	1	4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	4	3	2
3	2	1	4
4	3	2	1
2	1	4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	4	3	2
3	2	1	4
4	3	2	1
2	1	4	3

```
solve(problem: unary constraints)
```

- `domains = init_domains()`
- `restrict_domains(problem, domains)`
- initialize `stack` as an empty stack
- while `true`:
  - if `propagate(domains)` reports no conflict:
    - if `|domains[spot]| = 1` for all `spots`, this is a consistent assignment
    - `orig_domains = copy(domains)`
    - `spot, num = search(domains)`
    - add `(spot, num, orig_domains)` to `stack`
  - otherwise:
    - if `stack` is empty, there is no consistent assignment
    - `domains = backtrack(stack)`



```
propagate(domains: tile domains)
```

- do until `domains` doesn't change:
- for each `spot` on the board:
  - if `|domains[spot]| = 1`, remove `spot's num` from its `peers' domains`
  - if the above caused any domain to be empty, there is a conflict

1	4	3	2
3	2	1	4
4	3	2	1
2	1	4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	4	3	2
3	2	1	4
4	3	2	1
2	1	4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	4	3	2
3	2	1	4
4	3	2	1
2	1	4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	4	3	2
3	2	1	4
4	3	2	1
2	1	4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict



1	4	3	2
3	2	1	4
4	3	2	1
2	1	4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	4	3	2
3	2	1	4
4	3	2	1
2	1	4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	4	3	2
3	2	1	4
4	3	2	1
2	1	4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	4	3	2
3	2	1	4
4	3	2	1
2	1	4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	4	3	2
3	2	1	4
4	3	2	1
2	1	4	3

```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict

1	4	3	2
3	2	1	4
4	3	2	1
2	1	4	3

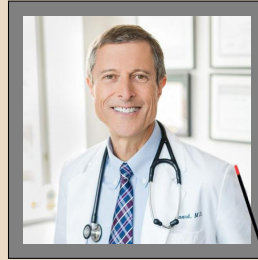
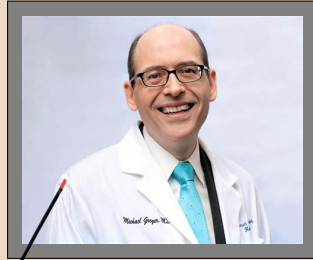
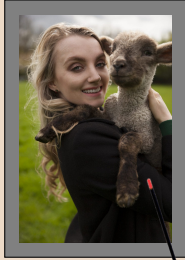
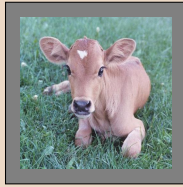
```
solve(problem: unary constraints)
```

- domains = init\_domains()
- restrict\_domains(problem, domains)
- initialize stack as an empty stack
- while true:
  - if propagate(domains) reports no conflict:
    - if |domains[spot]| = 1 for all spots, this is a consistent assignment
    - orig\_domains = copy(domains)
    - spot, num = search(domains)
    - add (spot, num, orig\_domains) to stack
  - otherwise:
    - if stack is empty, there is no consistent assignment
    - domains = backtrack(stack)



```
propagate(domains: tile domains)
```

- do until domains doesn't change:
  - for each spot on the board:
    - if |domains[spot]| = 1, remove spot's num from its peers' domains
    - if the above caused any domain to be empty, there is a conflict



SUPREME BENEVOLENT

