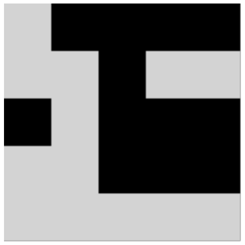
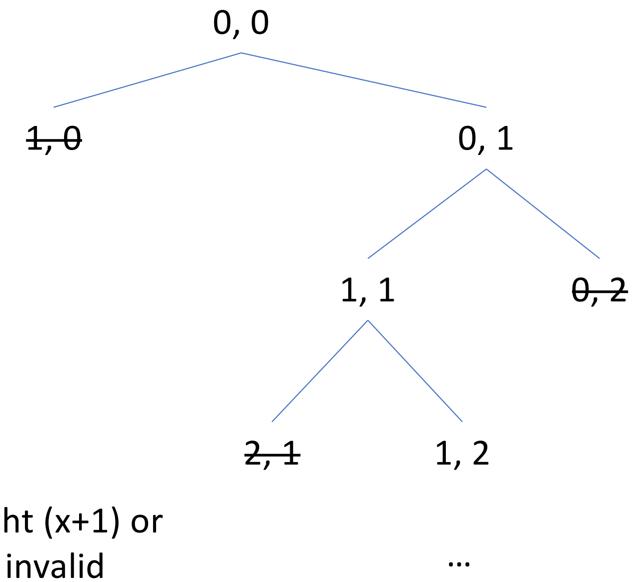
Lecture 21

5 maze functions in 2+ lectures (+ 2 more for later)

	result	Non-functional requirement	sr/tr	path?	visited?	tandem WLs	rsf
l21:	true or false	must terminate	sr				
	true or false	visit each pos only once	tr				
l22:	first path	small mazes	sr				
	shortest path Combination chooses shortest path	small mazes	sr				
	first path length	large mazes	tr				
l23:	shortest path	large mazes	tr				
	shortest path length	large mazes	tr				

Tree of x,y positions moving through this maze





At each step it is only possible to move right (x+1) or down (y+1). But sometimes those may be invalid because they run into a wall or off the edge of the maze.

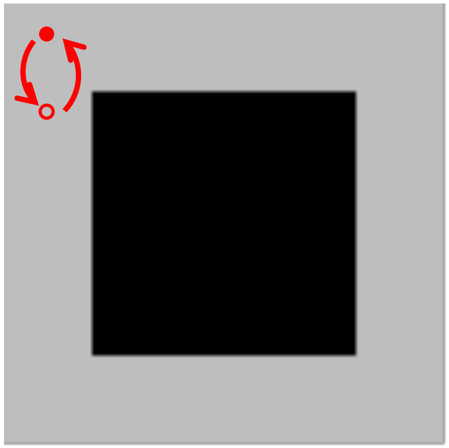
Do not assume each position can have only one valid next position. In general it is an arbitrary-arity tree.

This maze is solveable, so will eventually reach 4, 4. Yay!

```
(define M4
  (list 0 0 0 0 0
        0 W W W 0
        0 W 0 0 0
        0 W 0 W W
        W W 0 0 0))
```

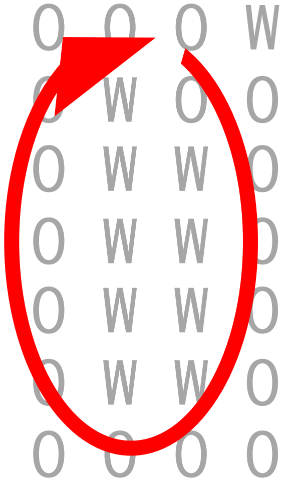
must move left

need to be able to move up down left right



```
(define M7
  (list O O O O O O O O O O
        W W O W W O W W W O
        O O O W W O W O O O
        O W O O W O W O W W
        O W W O W W W O O O
        O W W O O O W W W O
        O W W O W O W O O O
        O W W O O O W O W W
        O O O O W W W O O O
        W W W W W O O W W O))
```

```
(define M7
  (list O O O O O O O O O O
        W W O W W O W W W O
        O O O W W O W O O O
        O W O O W O W O W W
        O W W O W W W O O O
        O W W O O O W W W O
        O W W O W O W O O O
        O W W O O O W O W W
        O O O O W W W O O O
        W W W W W O O W W O))
```



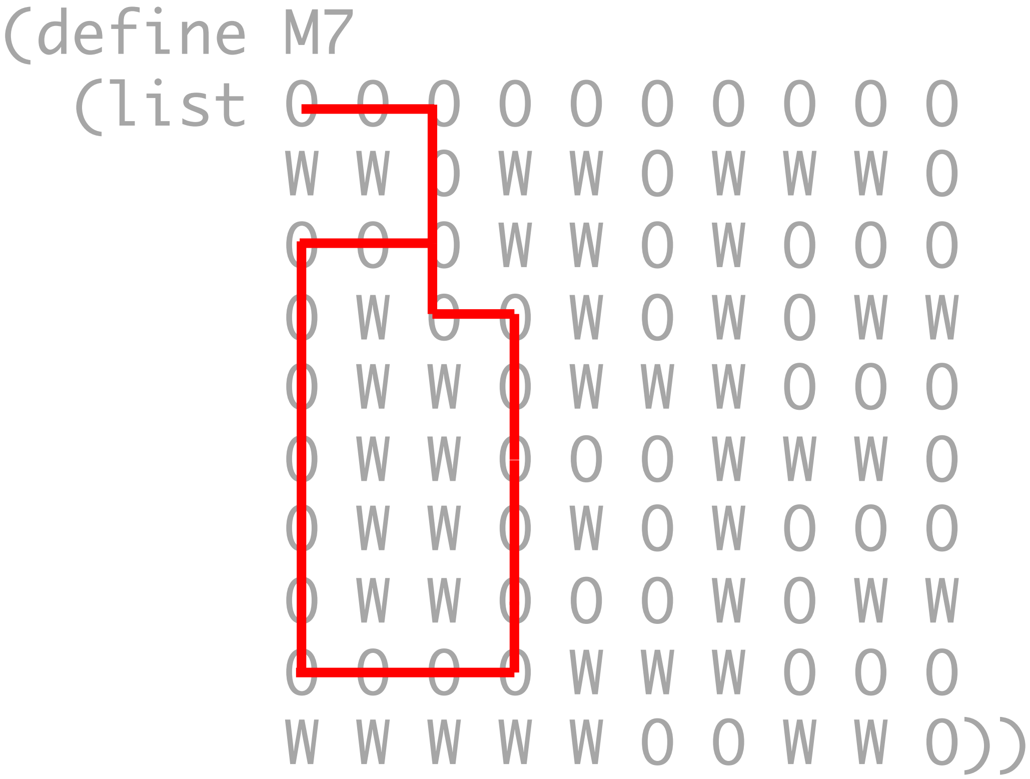
cycle

represent path to current position

use a path accumulator

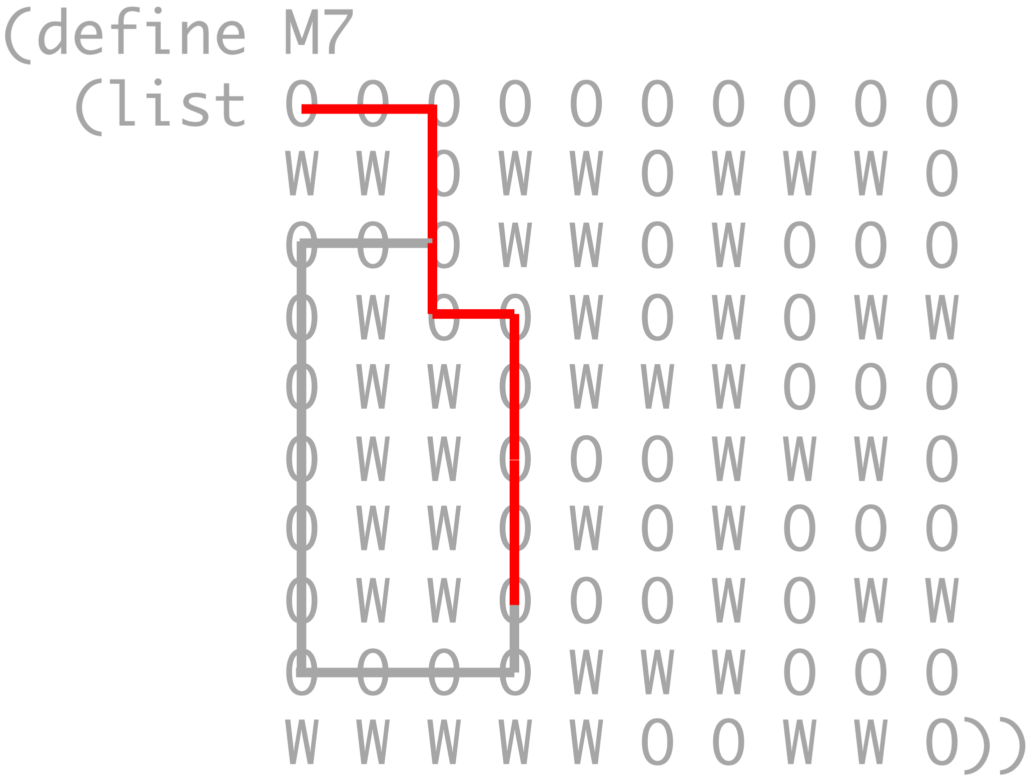
fail when a cycle is detected

```
;; structural recursion, with path accumulator

;; trivial:   reaches lower right, previously seen position
;; reduction: move up, down, left, right if possible
;; argument:  maze is finite, so moving will eventually
;;            reach trivial case or run out of moves

;; path is (listof Pos); positions on this path through data
(define (solve/p p path)
  (cond [(solved? p) true]
        [(member p path) false]
        [else
         (solve/lop (next-ps p)
                    (cons p path))]))

(define (solve/lop lop path)
  (cond [(empty? lop) false]
        [else
         (local [(define try (solve/p (first lop) path))]
           (if (not (false? try))
               try
               (solve/lop (rest lop) path)))]))
```



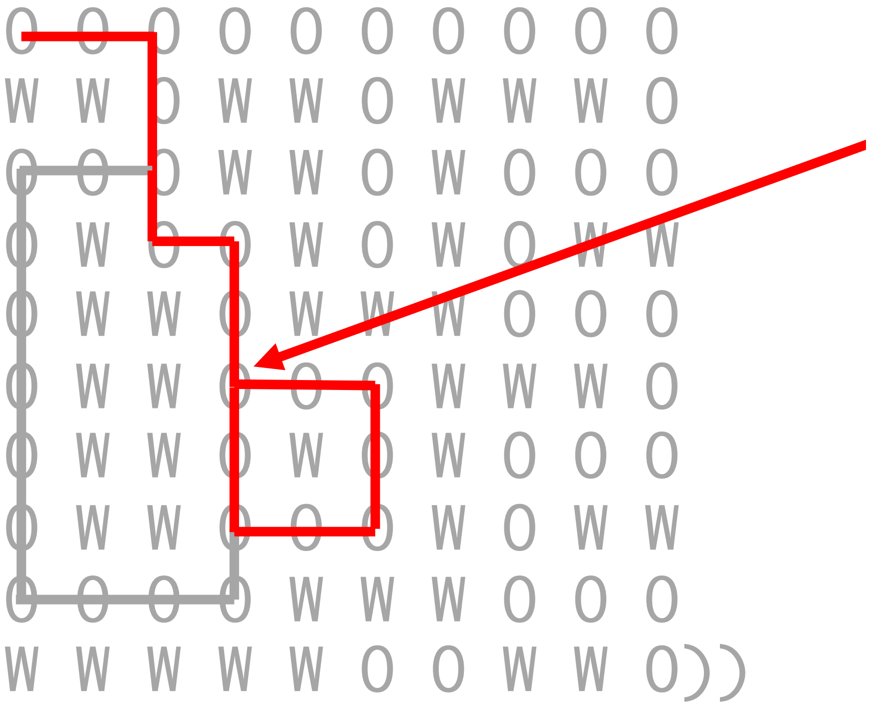
```
(define M7
  (list O O O O O O O O O O
        W W O W W O W W W O
        O O O W W O W O O O
        O W O O W O W O W W
        O W W O W W W O O O
        O W W O O O W W W O
        O W W O W O W O O O
        O W W O O O W O W W
        O O O O W W W O O O
        W W W W W O O W W O))
```

Period 1 –
up to cycle
detection



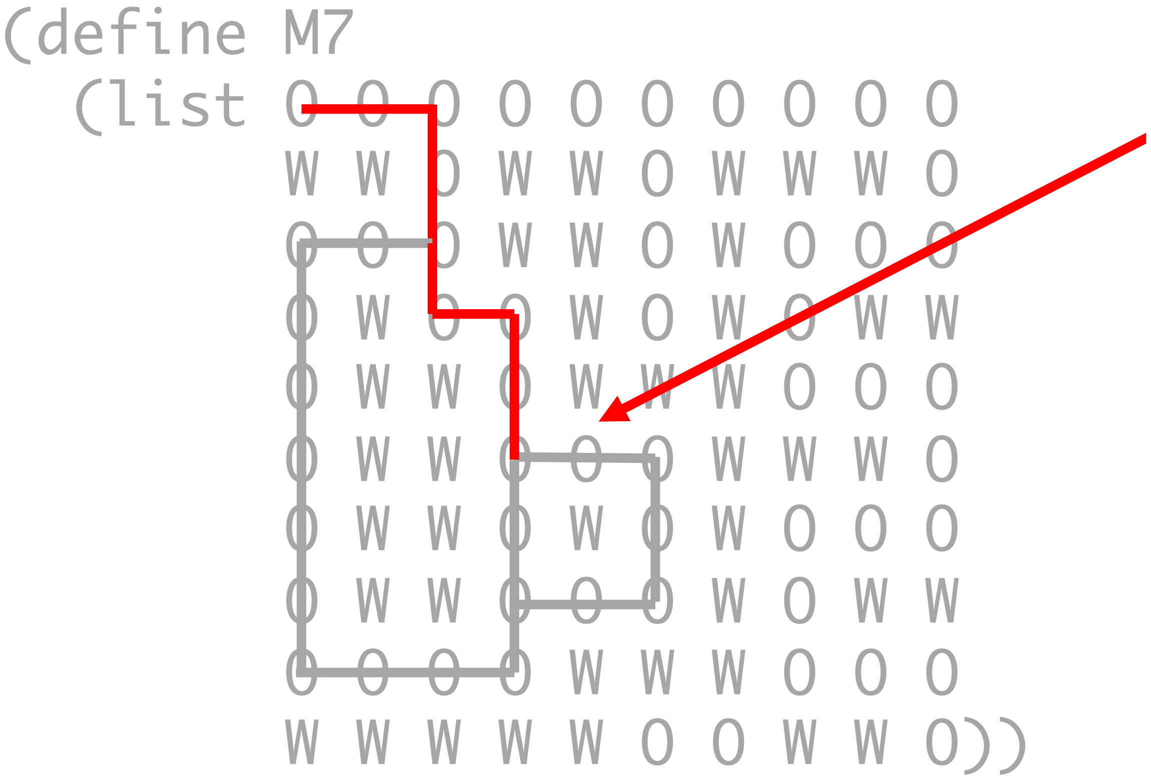
```
(define M7
  (list O O O O O O O O O O
        W W O W W O W W W O
        O O O W W O W O O O
        O W O O W O W O W W
        O W W O W W W O O O
        O W W O O O W W W O
        O W W O W O W O O O
        O W W O O O W O W W
        O O O O W W W O O O
        W W W W W O O W W O))
```

Period 2 – backtrack
(grey is visited, but not path; there is grey under the red)



```
(define M7
  (list 0 0 0 0 0 0 0 0 0 0
        W W 0 W W 0 W W W 0
        0 0 0 W W 0 W 0 0 0
        0 W 0 0 W 0 W 0 W W
        0 W W 0 W W W 0 0 0
        0 W W 0 0 0 W W W 0
        0 W W 0 W 0 W 0 0 0
        0 W W 0 0 0 W 0 W W
        0 0 0 0 W W W 0 0 0
        W W W W W 0 0 W W 0))
```

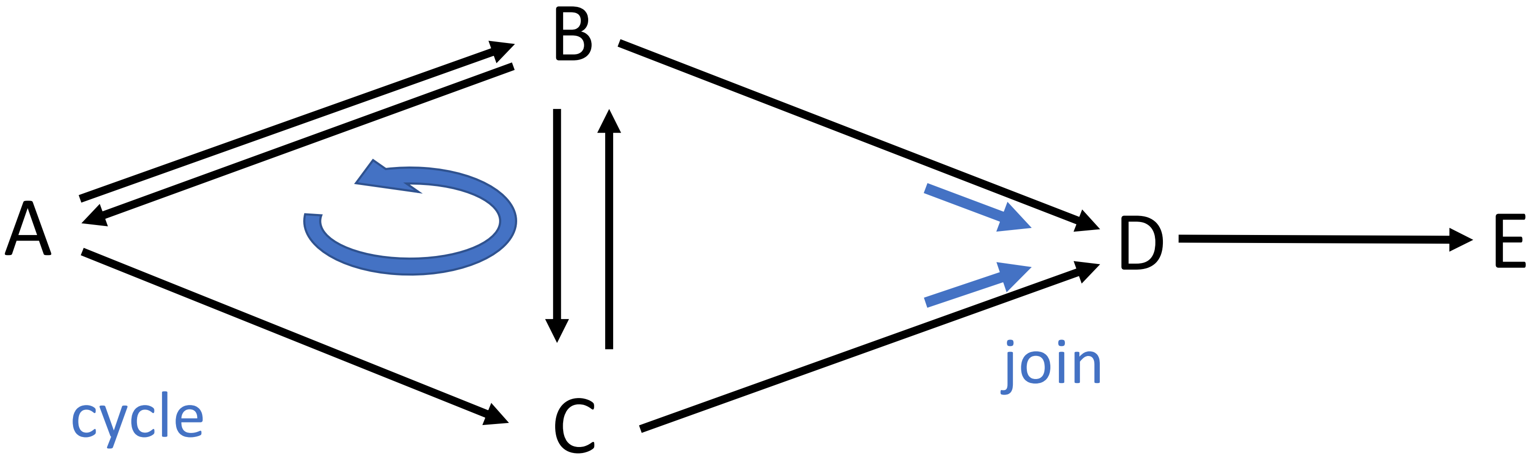
Period 3 –
right branch and
follow until detect
cycle



```
(define M7
  (list O O O O O O O O O O
        W W O W W O W W W O
        O O O W W O W O O O
        O W O O W O W O W W
        O W W O W W W O O O
        O W W O O O W W W O
        O W W O W O W O O O
        O W W O O O W O W W
        O O O O W W W O O O
        W W W W W O O W W O))
```

Period 4 – right branch and immediately hit join

Directed Graphs



```
;; tail recursion, with visited accumulator

;; trivial:   reaches lower right, previously seen position
;; reduction: move up, down, left, right if possible
;; argument:  maze is finite, so moving will eventually
;;            reach trivial case or run out of moves

;; p-wl is    (listof Pos); worklist
;; visited is (listof Pos); every position ever visited
(define (solve/p p p-wl visited)
  (cond [(solved? p) true]
        [(member p visited) (solve/lop p-wl visited)]
        [else
         (solve/lop (append (next-ps p) p-wl)
                    (cons p visited))]))

(define (solve/lop p-wl visited)
  (cond [(empty? p-wl) false]
        [else
         (solve/p (first p-wl) (rest p-wl) visited)]))
```

5 maze functions in 2 days (+ 2 more for later)

	result	**Non-functional requirement**	**sr/tr**	**path?**	**visited?**	**tandem WLs**	**rsf**
l21:	true or false	must terminate	sr	Y	n/a	n/a	n/a
	true or false	visit each pos only once	tr	N	Y	N	N
l22:	first path	small mazes	sr				
	shortest path Combination chooses shortest path	small mazes	sr				
	first path length	large mazes	tr				
l23:	shortest path	large mazes					
	shortest path length	large mazes					

Lecture 22

5 maze functions in 2 days (+ 2 more for later)

	result	**Non-functional requirement**	**sr/tr**	**path?**	**visited?**	**tandem WLs**	**rsf**
l21:	true or false	must terminate	sr	Y	n/a	n/a	n/a
	true or false	visit each pos only once	tr	N	Y	N	N
l22:	first path	small mazes	sr				
	shortest path Combination chooses shortest path	small mazes	sr				
	first path length	large mazes	tr				
l23:	shortest path	large mazes					
	shortest path length	large mazes					

produce path (sr)

```
(define M7
  (list O O O O O O O O O O
        W W O W W O W W W O
        O O O W W O W O O O
        O W O O W O W O W W
        O W W O W W W O O O
        O W W O O O W W W O
        O W W O W O W O O O
        O W W O O O W O W W
        O O O O W W W O O O
        W W W W W O O W W O))
```

```
;; trivial:   reaches lower right, previously seen position
;; reduction: move up, down, left, right if possible
;; argument:  maze is finite, so moving will eventually
;;            reach trivial case or run out of moves

;; path is (listof Pos); positions before p on this path through data
;;                       in reverse order
(define (solve/p p path)
  (cond [(solved? p) (reverse (cons p path))]
        [(member p path) false]
        [else
         (solve/lop (next-ps p)
                    (cons p path))]))

(define (solve/lop lop path)
  (cond [(empty? lop) false]
        [else
         (local [(define try (solve/p (first lop) path))]
           (if (not (false? try))
               try
               (solve/lop (rest lop) path)))]))
```

5 maze functions in 2 days (+ 2 more for later)

	result	**Non-functional requirement**	**sr/tr**	**path?**	**visited?**	**tandem WLs**	**rsf**
l21:	true or false	must terminate	sr	Y	n/a	n/a	n/a
	true or false	visit each pos only once	tr	N	Y	N	N
l22:	first path	small mazes	sr	Y	n/a	n/a	n/a
	shortest path Combination chooses shortest path	small mazes	sr				
	first path length	large mazes	tr				
l23:	shortest path	large mazes	tr				
	shortest path length	large mazes	tr				

produce shortest path (sr)

```
(define M7
  (list O O O O O O O O O O
        W W O W W O W W W O
        O O O W W O W O O O
        O W O O W O W O W W
        O W W O W W W O O O
        O W W O O O W W W O
        O W W O W O W O O O
        O W W O O O W O W W
        O O O O W W W O O O
        W W W W W O O W W O))
```

```
;; path is (listof Pos); positions before p on this path through data
;;                        in reverse order
(define (solve/p p path)
  (cond [(solved? p) (reverse (cons p path))]  ;(append path (list p))
        [(member p path) false]
        [else
         (solve/lop (next-ps p)
                    (cons p path))]))          ;(append path (list p))

(define (solve/lop lop path)
  (cond [(empty? lop) false]
        [else
         ;; this is the combination position where we can compare
         ;; two paths...
         (local [(define try1 (solve/p (first lop) path))
                 (define try2 (solve/lop (rest lop) path))]
           ;; (@template-origin 2-one-of)
           ;;      t2          false         (listof Pos)
           ;; t1
           ;;
           ;; false            t2            t2
           ;;
           ;; (listof Pos)     t1            (<shorter> t1 t2)
           ;;
           (cond [(false? try1) try2]
                 [(false? try2) try1]
                 [else
                  (if (<= (length try1) (length try2))
                      try1
                      try2)]))]))
```

5 maze functions in 2 days (+ 2 more for later)

	result	**Non-functional requirement**	**sr/tr**	**path?**	**visited?**	**tandem WLs**	**rsf**
l21:	true or false	must terminate	sr	Y	n/a	n/a	n/a
	true or false	visit each pos only once	tr	N	Y	N	N
l22:	first path	small mazes	sr	Y	n/a	n/a	n/a
	shortest path Combination chooses shortest path	small mazes	sr	Y	n/a	n/a	n/a
	first path length	large mazes	tr				
l23:	shortest path	large mazes	tr				
	shortest path length	large mazes	tr				

first path length (tr)

```
(define M7
  (list O O O O O O O O O O
        W W O W W O W W W O
        O O O W W O W O O O
        O W O O W O W O W W
        O W W O W W W O O O
        O W W O O O W W W O
        O W W O W O W O O O
        O W W O O O W O W W
        O O O O W W W O O O
        W W W W W O O W W O))
```

```
;; trivial:   reaches lower right, previously seen position
;; reduction: move up, down, left, right if possible
;; argument:  maze is finite, so moving will eventually
;;            reach trivial case or run out of moves

;; tail recursion, with visited accumulator, tandem worklists
;; p-wl    is (listof Pos);     position (node) worklist
;; c-wl    is (listof Natural); count worklist
;; INVARIANT: p-wl and c-wl always have same length, the
;;            elements of the two work lists correspond
;;            with each other - the nth element of c-wl
;;            is the number of steps in the path in the
;;            maze to reach the nth element of p-wl
;; visited is (listof Pos); every position ever visited
(define (solve/p p c p-wl c-wl visited)
  (cond [(solved? p) (add1 c)]
        [(member p visited) (solve/lop p-wl c-wl visited)]
        [else
         (solve/lop
          (append                    (next-ps p)            p-wl)
          (append (make-list (length (next-ps p)) (add1 c)) c-wl)
          (cons p visited))]))

(define (solve/lop p-wl c-wl visited)
  (cond [(empty? p-wl) false]
        [else
         (solve/p (first p-wl)
                  (first c-wl)
                  (rest p-wl)
                  (rest c-wl)
                  visited)]))
```

5 maze functions in 2 days (+ 2 more for later)

	result	Non-functional requirement	sr/tr	path?	visited?	tandem WLs	rsf
l21:	true or false	must terminate	sr	Y	n/a	n/a	n/a
	true or false	visit each pos only once	tr	N	Y	N	N
l22:	first path	small mazes	sr	Y	n/p	n/a	= path
	shortest path Combination chooses shortest path	small mazes	sr	Y	n/p	n/a	N
	first path length	large mazes	tr	N	Y	Y, path length	N
l23:	shortest path	large mazes	tr				
	shortest path length	large mazes	tr				

Lecture 23

```
(define M6
  (list O O O O O O O O O O
        W W O W W O W W W O
        O O O W W O W O O O
        O W O O W O W O W W
        O W W O W O W O O O
        O W W O W O W W W O
        O W W O W O W O O O
        O W W O O O W O W W
        O O O O W W W O O O
        W W W W W O O W W O))
```

```
(define M6
  (list O O O O O O O O O O
        W W O W W O W W W O
        O O O W W O W O O O
        O W O O W O W O W W
        O W W O W O W O O O
        O W W O W O W W W O
        O W W O W O W O O O
        O W W O O O W O W W
        O O O O W W W O O O
        W W W W W O O W W O))
```

```
(define M6
  (list O O O O O O O O O O
        W W O W W O W W W O
        O O O W W O W O O O
        O W O O W O W O W W
        O W W O W O W O O O
        O W W O W O W W W O
        O W W O W O W O O O
        O W W O O O W O W W
        O O O O W W W O O O
        W W W W W O O W W O))
```

```
;TR version
;; p-wl    is (listof Pos); Position (primary) worklist
;; path-wl is (listof (listof Pos)); path worklist
;; rsf     is (listof Pos) shortest path so far
;; NOTE: rsf is initialized to empty, which is shorter than any
;;       actual possible path
;; INVARIANT: p-wl and path-wl have same length, the nth
;;            element of path-wl is the path in the maze
;;            to the nth element of the p-wl
(define (solve/p p path p-wl path-wl rsf)
  (cond [(solved? p)
         (solve/lop p-wl path-wl (choose rsf (cons p path)))]
        [(member p path)
         (solve/lop p-wl path-wl rsf)]
        [else
         (solve/lop (append (next-ps p)
                            p-wl)
                    (append (make-list (length (next-ps p))
                                       (cons p path))
                            path-wl)
                    rsf)]))

(define (solve/lop p-wl path-wl rsf)
  (cond [(empty? p-wl) (if (empty? rsf) false (reverse rsf))]
        [else
         (solve/p (first p-wl)
                  (first path-wl)
                  (rest p-wl)
                  (rest path-wl)
                  rsf)]))
```