```
#lang racket
(require racket/trace)

(define (add_prefix pre lst)
  (map (lambda (x) (cons pre x)) lst))

;; The algorithm. O(n).
(define (hamiltonian_cycle_on_cube n)
  (cond
    [(<= n 1) '((0) (1))]
    [else
     (let* ((inner_cube (hamiltonian_cycle_on_cube (- n 1)))
            (zerolist (add_prefix 0 inner_cube))
            (onelist (add_prefix 1 (reverse inner_cube))))
       (append zerolist onelist))]))

(trace hamiltonian_cycle_on_cube)
(hamiltonian_cycle_on_cube 3)
;>(hamiltonian_cycle_on_cube 3)
;> (hamiltonian_cycle_on_cube 2)
;> >(hamiltonian_cycle_on_cube 1)
;< <'((0) (1))
;< '((0 0) (0 1) (1 1) (1 0))
;<'((0 0 0) (0 0 1) (0 1 1) (0 1 0) (1 1 0) (1 1 1) (1 0 1) (1 0 0))
;'((0 0 0) (0 0 1) (0 1 1) (0 1 0) (1 1 0) (1 1 1) (1 0 1) (1 0 0))


;; check if it's correct
(define (equal_but_one vert1 vert2 diffs)
  (cond
    [(empty? vert1) (= diffs 1)]
    [(= (car vert1) (car vert2))
     (equal_but_one (cdr vert1) (cdr vert2) diffs)]
    [else (equal_but_one (cdr vert1) (cdr vert2) (+ 1 diffs))]))

(define (ham-check-aux lst)
  (cond
    [(< (length lst) 2) #t]
    [else (let* ((vert1 (car lst))
                 (vert2 (cadr lst)))
            (and (equal_but_one vert1 vert2 0) (ham-check-aux (cdr lst))))]))

(define (ham-check n lst)
  (let ((len (length lst)))
    (and
     ;; correct number of visited vertices
     (= len (expt 2 n))
     ;; no duplicates
```

```
      (= len (length (remove-duplicates lst)))
      ;; All adjacent vertices differ by exactly one.
      (ham-check-aux lst))))

;(ham-check 1 (hamiltonian_cycle_on_cube 1))
;(ham-check 2 (hamiltonian_cycle_on_cube 2))
;(ham-check 4 (hamiltonian_cycle_on_cube 4))
;(ham-check 5 (hamiltonian_cycle_on_cube 5))
;(ham-check 10 (hamiltonian_cycle_on_cube 10))
;#t
;#t
;#t
;#t
;#t
```