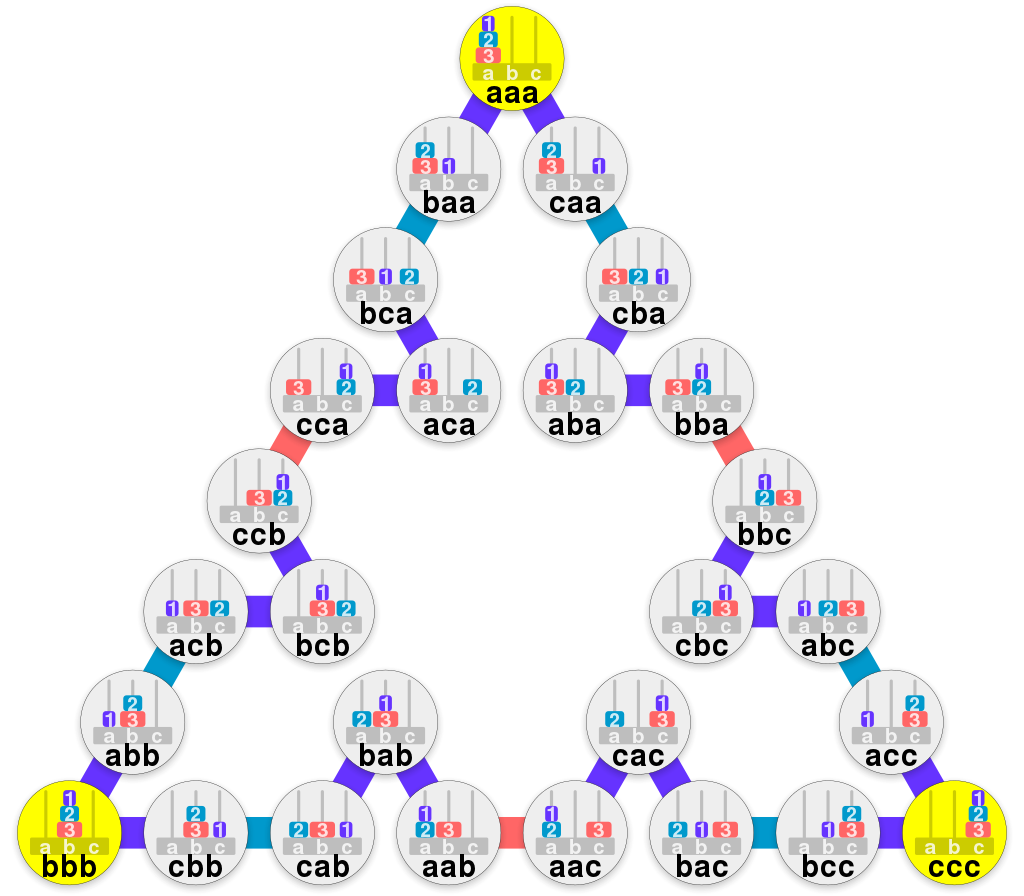
Team Project: Using Hill Climb algorithm to solve Towers of Hanoi problem.



Used this graph from wikipedia to create a graph that is understandable in clisp

(setq hanoi '(

(a (b 7) (c 6))

(b (a 7) (c 6) (h 8))

(c (a 7) (b 7) (f 5))

(d (e 4) (f 5) (g 6))

(e (d 5) (f 5) (w 3))

(f (c 6) (d 5) (e 4))

(g (d 5) (h 8) (i 7))

(h (b 8) (g 6) (i 7))

(i (g 6) (h 8) (r 7))

(j (k 5) (l 6) (s 3))

(k (j 4) (l 6) (q 6))

(l (j 5) (k 6) (o 6))

(m (n 7) (o 6) (p 7))

(n (m 7) (o 6))

(o (l 6) (m 7) (n 7))

(p (m 7) (q 6) (r 7))

(q (k 6) (p 7) (r 7))

(r (i 7) (p 7) (q 6))

(s (j 4) (ts 2) (u 3))

(ts (s 3) (u 3) (z 1))

(u (s 3) (ts 2) (x 3))

(v (w 3) (x 3) (y 1))

(w (e 4) (v 2) (x 3))

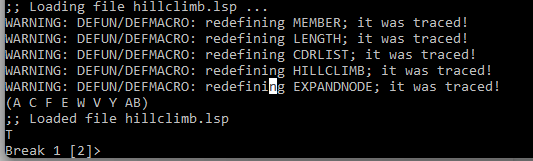
(x (u 3) (v 2) (w 3))

(y (v 2) (z 1) (ab 0))

(z (ts 2) (y 2) (ab 0))

(ab (y 1) (z 1)))

)



Running returns the path of

A -> C -> F -> E -> W -> V -> Y -> AB

Which in terms of the original graph goes

aaa -> caa -> cba -> bba -> bbc -> abc -> acc -> ccc

This is actually the optimal solution in this case, changing the weights it would be possible to path find to any state desired.

Rest of the code used

(setq temp hanoi)

(defun member (A L)

(cond

((NULL L) '())

((EQ (car L) A) T)

(T (member A (cdr L)))))

(defun length (L)

(do ((M L)(sum 0))

((NULL M) sum)

(setq M (cdr M))

(setq sum(+ sum 1))))

(defun cdrList (start graph)

(cond

((> (length graph) 2)

(cond

((eq (car(car graph)) start) graph)

((cdrList start (cdr graph))))

(cdr graph))))

(defun hillClimb (start target graph &optional path)

(cond

((eq start target)

(push start path)

(write (nreverse path)))

((eq (car(car graph)) start)

(push start path)

(setq start (expandNode start target (cdr(car graph)) path))

(hillClimb start target (cdrList start temp) path))

((hillClimb start target (cdrList start graph) path))))

(defun expandNode (node target connections path)

(setq shortest 9999)

(do ((M connections))

((NULL M) nextNode)

(cond

((member target (car M))

(setq nextNode target)

(setq M (cdr M)))

((member (car(car M)) path) (setq M (cdr M)))

((cond

((< (car(cdr(car M))) shortest)

(setq shortest (car(cdr(car M))))

(setq nextNode (car(car M)))

(setq M (cdr M)))

((setq M (cdr M))))))))