B → target routing array (grid)

s → source terminal

t → destination/target terminal

P → routing path for net (s, t)

Algorithm LEE-ROUTER (B, s, t, P)
    input: B, s, t
    output: P
begin
    $plist = s$; // "present" list, source of wave exp.
    $nlist = \phi$; // "next" list (of grid elements)
    $temp = 1$; // label number used during wave exp.
    $path\_exists$ = FALSE;
    while $plist \neq \phi$ **do** // until wave exp. is possible
        for each vertex v_i **in** $plist$ **do**
            for each vertex v_j **neighboring** v_i **do**
            if $B[v_j]$ = UNBLOCKED **then**
                $L[v_j] = temp$; // label grid element (1, 2, 3, ...)
                INSERT(v_j, $nlist$); // add to list for future wave exp.
                if $v_j = t$ **then**
                    $path\_exists$ = TRUE;
                    exit while;
        $temp = temp + 1$; // increment label #
        $plist = nlist$; // new source(s) for wave exp.
        $nlist = \phi$;
    if $path\_exists$ = TRUE **then** RETRACE (L, P); → pick one routing path (we may have one or more avail.)
    else path does not exist; // declare failure
end.

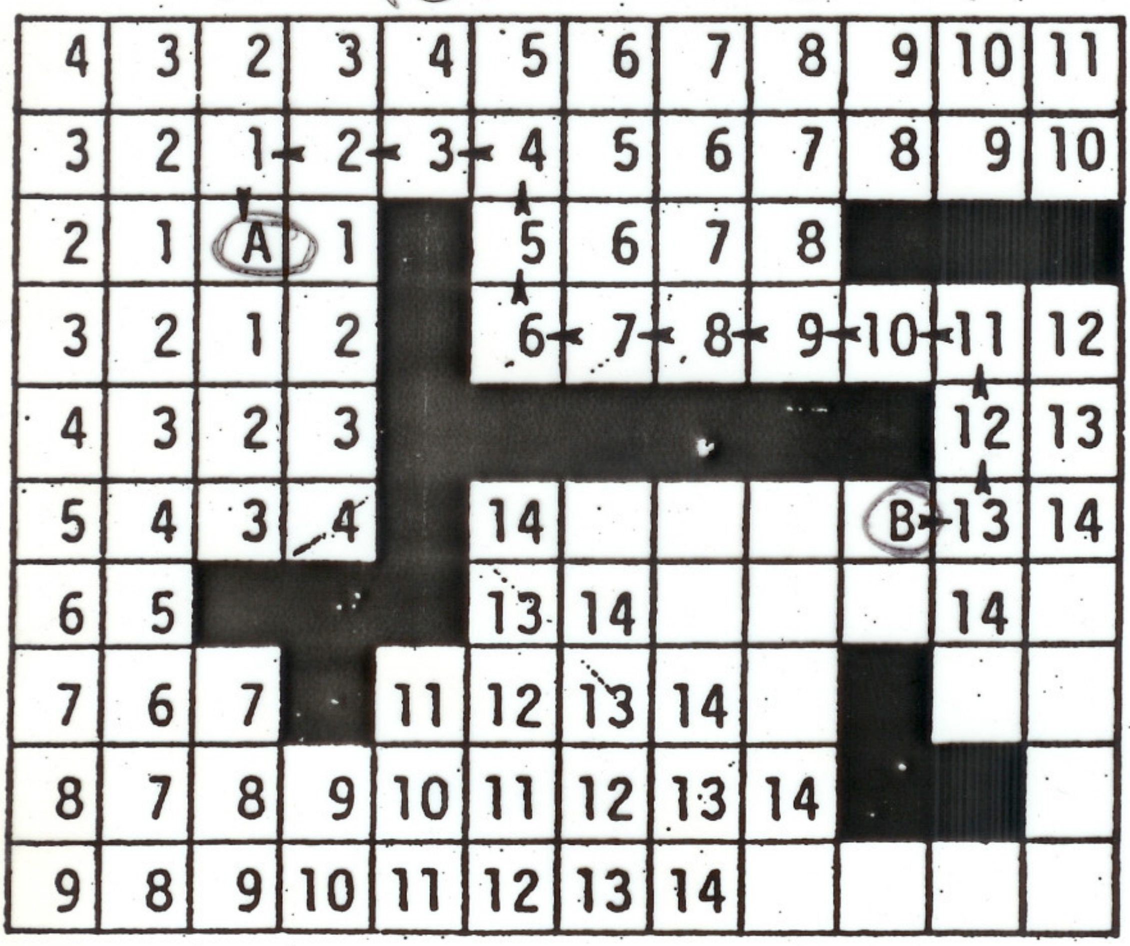
Figure 8.16: Algorithm LEE-ROUTER.

+ guaranteed to find a connection if it exists
+ guaranteed to find shortest path
− expensive in terms of run time & memory req. ; used as last step (cleanup) (last resort)

- Wave expansion & traceback for two-terminal net whose:
  source is A
  destination is B

- For many nets there is a choice of multiple traceback paths. Pick one according to other criteria
  → minimize # of bends
  → take least congested path (keep routing resources available for other nets to be routed)

- Last few nets are the hardest to route (in practice)
  ↳ especially in "fixed" targets such as FPGAs



Time & space complexit $O(h \times w)$ where $h \times w$ is grid size
→ expensive for large arrays
→ Stacks?