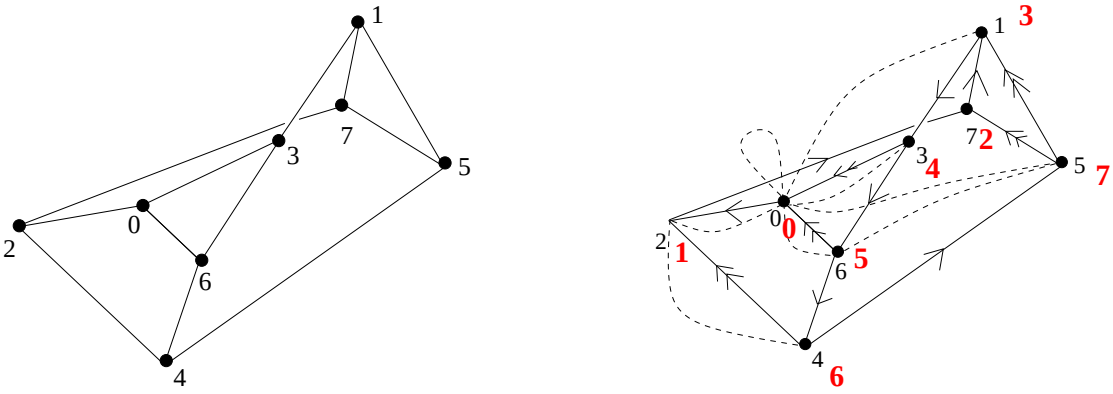

Figure 1: Graph and palm-tree version (almost). Black numbers, original vertex labelling. Edges with arrows: tree edges. Red numbers: preorder rank. Edges with chevrons: back edges. `Highpt-1` function: dotted lines. `Highpt-2` omitted; `highpt-2[u]=u` for all u.

22 Hopcroft-Tarjan algorithm

The Hopcroft-Tarjan algorithm takes a graph G and first uses depth-first search to evaluate certain features of the graph:

- Parent links
- Preorder ranks
- `highpt_1` function, same as `highpt` in the bcc algorithm, and
- `highpt_2` function, defined as follows:

  `highpt_2[u]` is, if it exists, the highest ancestor v of u (i.e., the ancestor with lowest preorder rank) which is connected to a descendant w of u, so (w, v) is a back edge, ***and*** v is strictly below `highpt_1[u]`. If no such v exists then `highpt_2[u]` takes the default value u.

- $\phi(u, v)$ defined for tree and back edges (u, v) as follows:
  - $\phi(u, v) = 2\times$ preorder rank of v if (u, v) is a back edge.
  - Let h_1 and h_2 be the high points of v[1]
    $\phi(u, v) = 2\times$ preorder rank of h_1 if $h_2 = v$, and
  - $\phi(u, v) = 1 + 2\times$ preorder rank of h_1, if $h_2 \neq v$.

The *palm-tree* form of a graph is the directed graph with directed edges for tree edges and back edges and none for their inverse. Also, the out-edges from each node are arranged in a particular order (not a cyclic order), so that if $(u, v), (u, w)$ are successive out-edges from u in this arrangement, then $\phi(u, v) \leq \phi(u, w)$.

The Hopcroft-Tarjan algorithm operates on the palm-tree graph as follows.

[1] Correcting an error; high points of v, not u.

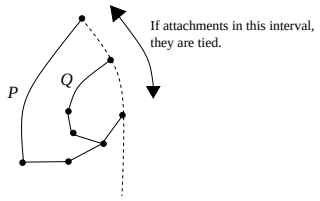

Figure 2: Q is tied to P

- There is a procedure `extract-path(u)` which begins at u, takes the next available out-edge (u, v) (if any), where this out-edge has minimum ϕ-value among all remaining out-edges out of u, deletes (u, v) from the palm-tree graph: it is the first edge in the path. Then it selects the first available edge (v, w) out of v, and so on, until an out-edge (x, y) has been extracted where y is `highpt_1[u]`.

- The first path extracted from 0 (the first vertex in preorder) is a simple cycle leading back to 0.

- The form of paths made by `extract-path` is $(u_1, u_2), \ldots, (u_{k-2}, u_{k-1}), (u_{k-1}, u_k)$ where the first $k - 1$ edges are tree edges and the last is a back edge, $u_1 = u$ and u_k is `highpt-1[u]`.

- Attempt layout on such a path first attempts to place the path correctly with respect to the previously laid out paths. This may fail because of interlacing conflicts, and the graph is declared nonplanar.

- Then if $k > 2$ it attempts to lay out paths emanating from u_{k-1}, u_{k-2}, *recursively*, and so on; extracting and arranging all paths from u_{k-1}, then from u_{k-2} and so on up to u_2.

(22.1) Definition *Given such a path P, the paths extracted from $u_{k-1}, \ldots, u_2$ are considered to have P as* parent. *If one of these paths, Q, has its highest attachment strictly higher than u_1, it is* tied to P. *This restricts the ways in which Q can be laid out.*

Here is more data about the palm-tree illustrated above.

```
8 24
0 3 2 3 6
1 3 3 5 7
2 3 0 7 4
3 3 0 6 1
4 3 6 2 5
5 3 1 7 4
6 3 0 4 3
7 3 1 5 2
tables
   graph extras, 8 vertices
   vertex    parent  pre_rank post_rank highpt_1 highpt_2
        0        -1         0         7        0        0
```

```
        1         7         3         4         0         2
        2         0         1         6         0         2
        3         1         4         3         0         2
        4         6         6         1         2         7
        5         4         7         0         7         1
        6         3         5         2         0         2
        7         2         2         5         0         2
tables, preorder
graph extras, 8 vertices, by pre_rank
vertex  parent  pre_rank  highpt_1  highpt_2
     0      -1         0         0         0
     1       0         1         0         1
     2       1         2         0         1
     3       2         3         0         1
     4       3         4         0         1
     5       4         5         0         1
     6       5         6         1         2
     7       6         7         2         3
pre_rank 0 3 1 4 6 7 5 2
phi function
 u  v  phi(u,v) u  v  phi(u,v) u  v  phi(u,v) u  v  phi(u,v)
 0  2       0    1  3       1    2  7       1    3  0       0
 3  6       1    4  2       2    4  5       5    5  7       4
 5  1       6    6  0       0    6  4       3    7  1       1

palmtree
8 12
0 1 2
1 1 3
2 1 7
3 2 0 6
4 2 2 5
5 2 7 1
6 2 0 4
7 1 1
path index 0 0:0 2:1 7:2 1:3 3:4 0:0
path index 1 3:4 6:5 0:0
path index 2 6:5 4:6 2:1
path index 3 4:6 5:7 7:2
path index 4 5:7 1:3
```

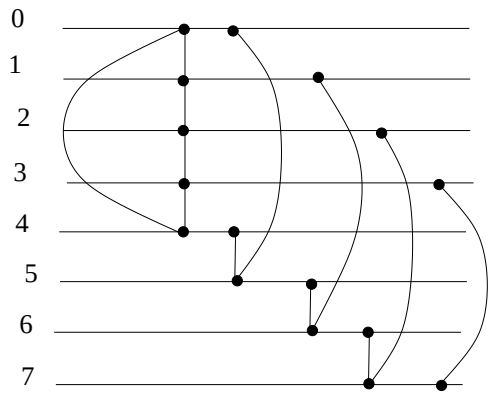

Figure 3: Illustrating the paths extracted

22.1 Attempt_layout in detail

(22.2) Definition *A* block *is a maximal collection of paths with the property that, given any one of the paths in the block, if it is placed to the left or right of its parent path then the placement (left or right of its parent path) of every path in the block is fully determined.*

A block has two sides, those on left and those on the right. The sides of a block can be swapped.

The sides of a block are separated into two lists of paths, active and inactive. The inactive paths belong to the block but they are not involved in any more interlacing tests.

(22.3) Proposition *Let P be a path introduced by* `attempt_layout`*, and let Q be an active path stored in one of the existing blocks. Then P and Q are part of interlacing bridges if and only if the highest attachment of P is above that of Q.*

There is no interlacing if Q is inactive.