## *Algorithm* (**X** 🡪 **Y**)

|  |  |  |
| --- | --- | --- |
| 0 | - *find\_nearest\_neighbor()* on the *starting node*. If it’s the *terminal node*, return (a small chain). |  |
| 0 | - *find\_node\_by\_src()* on the *starting node* and count the results. If NULL, return w/error NOGRAPH. |  |
| 0 | - *create\_pathpage()* for *that many found nodes*, and *append\_pathpage(i, starting node)*. [Append the *starting node* to each new *pathpage*]  - *append\_pathpage(i, x) < i = index; x = found nodes >* [Append each node found above]  - *append\_visited(starting node)* [Add the *starting node* to the *visited set*.] |  |
| 1 | - *find\_unvisited\_pathpage\_terminators()*. [Search through *pathpages* for terminators that are **not** in the *visited set*.]  - If nothing comes up, search every path *terminator* for their **neighbors** which are **not** in the *visited set*.  *- Clone* these *pathpages,* and append the *node(s)* found to the new *pathpages*. **Goto** step 1. |  |
| 2 | - For each *non-visited terminating node* found, run *find\_node\_by\_src()*, and exclude resulting nodes that are in the *visited set*.  - If the only nodes returned are in the visited set, consider the result *NULL*.  - Add its *terminator* to the *visited set*. If *NULL* **and** the *pathpage terminator* is **not** the *terminal node*, *kill* the applicable *pathpage*.  - **Otherwise**, count the total number of possible nodes-to-be-visited from the new nodes.  - Add the *terminators* for the *pathpages*-to-be-copied to the *visited set*.  - *Copy* the applicable *pathpages* by how many non-visited nodes were returned.  - Append the nodes found above to the new *pathpages*, so long as the new nodes are **adjacent** to their respective *terminator.* If not, *kill* that *pathpage*.  - If the node you just appended was the *terminal node*, mark it ‘*OK*’ (re-*activate* that *pathpage*).  - Increment + check the global *loop counter*. If it’s exceedingly high (≥ *nodes2*), either the graph or the algorithm is broken. Return w/error EPATHFAIL. |  |
| 3 | - *all\_nodes\_are\_visited()*?If not, **goto** step 1.  - Check ~~living~~ *pathpage* terminators. If the last element is the *terminal node*, mark *OK*. **Or**, if *any* neighbor is the *terminal node*, append the *terminal node*, and mark OK.  - Kill any *living* *pathpages* that contain loops/repeated nodes. Add their *terminators* back to the *visited set*. (remove ‘*OK’* if applicable).  - If any living *pathpages* remain that do **not** contain the *terminal node*:  - Remove their *terminators* **and** the neighbors of those *terminators* **and** the *terminal node* from the *visited set*. Do **not** remove the *starting node*. **Goto** step 1.  ~~- Kill other paths~~ **~~and~~** ~~kill paths that do not follow sequentially. Find + kill loops again.~~  - *find\_shortest\_pathpage()* for each remaining living *pathpage* + apply tie-breaking rules. Return the winning *pathpage*. |  |

## *Visited Set*

|  |
| --- |
|  |

## *Pathpages*

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |