CS39002 Operating Systems Laboratory
Spring 2026

Lab Assignment: 1
Date posted: 09-Jan-2026

System Calls Fork, Wait, and Exec

Let $G = (V, E)$ be a directed graph with $|V| = n$. A Hamiltonian cycle in G is a (simple) directed cycle in G of length n. We know that determining a Hamilotonian cycle in graphs is an NP-complete problem. We can nonetheless run an exhaustive search for Hamiltonian cycles in small graphs. One step of the search algorithm is given below. We assume that the vertices of G are numbered 1, 2, . . . , n.

> Let $v_1, v_2, \ldots, v_c$ be a partially computed path discovered so far in G. We always take $v_1 = 1$.
> If $c = n$, check if v_1 is in the neighbor list of v_n. Return *success* or *failure* accordingly.
> If $c = 0$, then take $v_1 = 1$.
> Otherwise, for each neighbor v_{c+1} of v_c, that is **not** on the partially constructed path, extend the current path to $v_1, v_2, \ldots, v_c, v_{c+1}$, and continue the search. If any of these searches discovers a Hamiltonian cycle, then return *success*.
> If all these neighbors (if any) fail to locate a Hamiltonian cycle, return *failure*.

In this assignment, you write a single multi-process application to carry out the above search. Each process P handles the search (except the continuation part) of the above pseudocode. The process P receives the partially constructed path $v_1, v_2, \ldots, v_c$ via its command-line parameters. You run the program (the root process in the application) without any command-line argument (this corresponds to $c = 0$).

If P finds $c = n$, then it searches for v_1 in the neighborhood of v_n, and terminates with an appropriate exit status indicating *success* or *failure* (see below). Before the termination, Q prints this Hamiltonian cycle.

If $c = 0$ (the root process), P forks a child process, and waits for the child process Q to terminate. If the exit status of Q is *success*, P also terminates with exit status indicating *success*. If the exit status of Q is *failure*, P prints "No Hamiltonian cycle found", and terminates with exit status indicating *failure*.

In the case $0 < c < n$, P carries out a search for the unvisited neighbors v_{c+1} of v_c. For each such neighbor, P forks a child process R, and waits for R to terminate. If R returns *success*, P also returns *success*, and terminates. If R returns *failure*, P continues the search with the next unvisited neighbor v_{c+1}. If the child processes corresponding to all these neighbors return *failure*, P terminates after returning *failure*.

P extends the current path by v_{c+1} for each unvisited neighbor in the adjacency list for v_c. This is done by creating an `argv` array for the child process (Q or R). The child process `exec`'s the same program by passing the augmented `argv` array as the command-line arguments.

Exit status: Follow the convention: `exit(0)` means return *success*, and `exit(1)` means return *failure*.

Input graph *G*: This is stored in a text file `graph.txt`. The first line stores the number n of vertices in G. This is followed by n lines storing the neighbors of the vertices. Recall that we number the vertices as 1, 2, . . . , n. The neighbors of vertex u are specified as follows.

```
u -> n1 n2 ... nk
```

A random graph generator `gengraph.c` is provided to you. Compile and run the program with two arguments: n (the number of vertices in G), and p (the probability with which each edge (u, v) exists in E).

Submit a single C/C++ source file.

Sample Output

A Hamiltonian graph

```
8
1 -> 2 3 4
2 -> 7
3 -> 2 5 6 8
4 -> 2 3 8
5 -> 2 3 7 8
6 -> 5 7
7 -> 6 8
8 -> 1 4
```

```
*** Process 6129:
*** Process 6130: 1
*** Process 6131: 1 2
*** Process 6132: 1 2 7
*** Process 6133: 1 2 7 6
*** Process 6134: 1 2 7 6 5
*** Process 6135: 1 2 7 6 5 3
*** Process 6136: 1 2 7 6 5 3 8
*** Process 6137: 1 2 7 6 5 3 8 4
*** Process 6138: 1 2 7 6 5 8
*** Process 6139: 1 2 7 6 5 8 4
*** Process 6140: 1 2 7 6 5 8 4 3
*** Process 6141: 1 2 7 8
*** Process 6142: 1 2 7 8 4
*** Process 6143: 1 2 7 8 4 3
*** Process 6144: 1 2 7 8 4 3 5
*** Process 6145: 1 2 7 8 4 3 6
*** Process 6146: 1 2 7 8 4 3 6 5
*** Process 6147: 1 3
*** Process 6148: 1 3 2
*** Process 6149: 1 3 2 7
*** Process 6150: 1 3 2 7 6
*** Process 6151: 1 3 2 7 6 5
*** Process 6152: 1 3 2 7 6 5 8
*** Process 6153: 1 3 2 7 6 5 8 4
*** Process 6154: 1 3 2 7 8
*** Process 6155: 1 3 2 7 8 4
*** Process 6156: 1 3 5
*** Process 6157: 1 3 5 2
*** Process 6158: 1 3 5 2 7
*** Process 6159: 1 3 5 2 7 6
*** Process 6160: 1 3 5 2 7 8
*** Process 6161: 1 3 5 2 7 8 4
*** Process 6162: 1 3 5 7
*** Process 6163: 1 3 5 7 6
*** Process 6164: 1 3 5 7 8
*** Process 6165: 1 3 5 7 8 4
*** Process 6166: 1 3 5 7 8 4 2
*** Process 6167: 1 3 5 8
*** Process 6168: 1 3 5 8 4
*** Process 6169: 1 3 5 8 4 2
*** Process 6170: 1 3 5 8 4 2 7
*** Process 6171: 1 3 5 8 4 2 7 6
*** Process 6172: 1 3 6
*** Process 6173: 1 3 6 5
*** Process 6174: 1 3 6 5 2
*** Process 6175: 1 3 6 5 2 7
*** Process 6176: 1 3 6 5 2 7 8
*** Process 6177: 1 3 6 5 2 7 8 4
*** Process 6178: 1 3 6 5 7
*** Process 6179: 1 3 6 5 7 8
*** Process 6180: 1 3 6 5 7 8 4
*** Process 6181: 1 3 6 5 7 8 4 2
*** Process 6182: 1 3 6 5 8
*** Process 6183: 1 3 6 5 8 4
*** Process 6184: 1 3 6 5 8 4 2
*** Process 6185: 1 3 6 5 8 4 2 7
*** Process 6186: 1 3 6 7
*** Process 6187: 1 3 6 7 8
*** Process 6188: 1 3 6 7 8 4
*** Process 6189: 1 3 6 7 8 4 2
*** Process 6190: 1 3 8
*** Process 6191: 1 3 8 4
*** Process 6192: 1 3 8 4 2
*** Process 6193: 1 3 8 4 2 7
*** Process 6194: 1 3 8 4 2 7 6
*** Process 6195: 1 3 8 4 2 7 6 5
*** Process 6196: 1 4
*** Process 6197: 1 4 2
*** Process 6198: 1 4 2 7
*** Process 6199: 1 4 2 7 6
*** Process 6200: 1 4 2 7 6 5
*** Process 6201: 1 4 2 7 6 5 3
*** Process 6202: 1 4 2 7 6 5 3 8

Hamiltonian cycle found: 1 4 2 7 6 5 3 8 1
```

A non-Hamiltonian graph

```
8
1 -> 4 5 8
2 -> 3 4 8
3 -> 4 6
4 -> 5 8
5 -> 1 2 6 7
6 -> 3 4 7
7 -> 3 4
8 -> 5
```

```
*** Process 8812:
*** Process 8813: 1
*** Process 8814: 1 4
*** Process 8815: 1 4 5
*** Process 8816: 1 4 5 2
*** Process 8817: 1 4 5 2 3
*** Process 8818: 1 4 5 2 3 6
*** Process 8819: 1 4 5 2 3 6 7
*** Process 8820: 1 4 5 2 8
*** Process 8821: 1 4 5 6
*** Process 8822: 1 4 5 6 3
*** Process 8823: 1 4 5 6 7
*** Process 8824: 1 4 5 6 7 3
*** Process 8825: 1 4 5 7
*** Process 8826: 1 4 5 7 3
*** Process 8827: 1 4 5 7 3 6
*** Process 8828: 1 4 8
*** Process 8829: 1 4 8 5
*** Process 8830: 1 4 8 5 2
*** Process 8831: 1 4 8 5 2 3
*** Process 8832: 1 4 8 5 2 3 6
*** Process 8833: 1 4 8 5 2 3 6 7
*** Process 8834: 1 4 8 5 6
*** Process 8835: 1 4 8 5 6 3
*** Process 8836: 1 4 8 5 6 7
*** Process 8837: 1 4 8 5 6 7 3
*** Process 8838: 1 4 8 5 7
*** Process 8839: 1 4 8 5 7 3
*** Process 8840: 1 4 8 5 7 3 6
*** Process 8841: 1 5
*** Process 8842: 1 5 2
*** Process 8843: 1 5 2 3
*** Process 8844: 1 5 2 3 4
*** Process 8845: 1 5 2 3 4 8
*** Process 8846: 1 5 2 3 6
*** Process 8847: 1 5 2 3 6 4
*** Process 8848: 1 5 2 3 6 4 8
*** Process 8849: 1 5 2 3 6 7
*** Process 8850: 1 5 2 3 6 7 4
*** Process 8851: 1 5 2 3 6 7 4 8
*** Process 8852: 1 5 2 4
*** Process 8853: 1 5 2 4 8
*** Process 8854: 1 5 2 8
*** Process 8855: 1 5 6
*** Process 8856: 1 5 6 3
*** Process 8857: 1 5 6 3 4
*** Process 8858: 1 5 6 3 4 8
*** Process 8859: 1 5 6 4
*** Process 8860: 1 5 6 4 8
*** Process 8861: 1 5 6 7
*** Process 8862: 1 5 6 7 3
*** Process 8863: 1 5 6 7 3 4
*** Process 8864: 1 5 6 7 3 4 8
*** Process 8865: 1 5 6 7 4
*** Process 8866: 1 5 6 7 4 8
*** Process 8867: 1 5 7
*** Process 8868: 1 5 7 3
*** Process 8869: 1 5 7 3 4
*** Process 8870: 1 5 7 3 4 8
*** Process 8871: 1 5 7 3 6
*** Process 8872: 1 5 7 3 6 4
*** Process 8873: 1 5 7 3 6 4 8
*** Process 8874: 1 5 7 4
*** Process 8875: 1 5 7 4 8
*** Process 8876: 1 8
*** Process 8877: 1 8 5
*** Process 8878: 1 8 5 2
*** Process 8879: 1 8 5 2 3
*** Process 8880: 1 8 5 2 3 4
*** Process 8881: 1 8 5 2 3 6
*** Process 8882: 1 8 5 2 3 6 4
*** Process 8883: 1 8 5 2 3 6 7
*** Process 8884: 1 8 5 2 3 6 7 4
*** Process 8885: 1 8 5 2 4
*** Process 8886: 1 8 5 6
*** Process 8887: 1 8 5 6 3
*** Process 8888: 1 8 5 6 3 4
*** Process 8889: 1 8 5 6 4
*** Process 8890: 1 8 5 6 7
*** Process 8891: 1 8 5 6 7 3
*** Process 8892: 1 8 5 6 7 3 4
*** Process 8893: 1 8 5 6 7 4
*** Process 8894: 1 8 5 7
*** Process 8895: 1 8 5 7 3
*** Process 8896: 1 8 5 7 3 4
*** Process 8897: 1 8 5 7 3 6
*** Process 8898: 1 8 5 7 3 6 4
*** Process 8899: 1 8 5 7 4

No Hamiltonian cycle found
```