Homework 4

Costa Stavrianidis

2022-12-03

Nested Set Enumeration

1a Write an algorithm in R or Python that computes the full nested set enumeration values for a tree data structure in an SQL database, according to the E-R model presented in class, and updates the nested set values in the database accordingly.

- **You may use an RDBMS of your choice to connect to. However, we recommend using SQLite, which is widely supported, both for programming languages and OSs, and is often already pre-installed (such as an macOS). You may need to install the SQL database interface libraries and/or packages for your chosen programming language.**
    - **For R and SQLite, you will need the DBI (API for opening and closing a database connection, sending statements, and receiving results) and RSQLite packages, both available from CRAN. For example, the code for connecting to an in-memory database is the following:**
        - library(RSQLite)
        - library(DBI)
        - db <- dbConnect(RSQLite::SQLite(), ":memory:")
        - **Substitute a database file name (SQLite databases are in a single file with extension .db) to connect to an on-disk database.**
    - **For Python and SQLite, there is the sqlite3 package, which implements the Python DB-API. There is also sqlalchemy for a more abstracted interface (that translates API function calls into SQL for you). You can choose either one.**
- **To populate the database for testing queries and list result sets, we will use the the NCBI Taxonomy tree. For SQLite, you can obtain a pre-populated database from a 2017 snapshot by following the directions here (under the "Tree" section). (This will result in almost 1M rows in the Node table.)**
    - **Note that this database already contains pre-computed nested set values; you may use these for testing your queries, but the nested set values your algorithm computes, if implemented correctly, will be different.**
- **Note: As is the case in general for species, phylogenetic, and other trees in biology, when performing nested set enumeration on the NCBI Taxonomy tree there is no significance to the ordering of child nodes.**

Answer the following questions.

- **Query the maximum nested set value from the database after your algorithm completes. How does it compare to the expected value? (Hint: if nested set enumeration starts at 1 and**

increments by 1, the expected maximum nested set value must be a particular integer multiple of the number of nodes visited.)

- **What is the time complexity and memory complexity of your algorithm (in O notation). Is it different from the best possible algorithm (why or why not).**
- **List names and nested set enumeration values for NCBI Taxon IDs (= NODE.IDs) 37570, 451864, and 1940571**

1b Using the nested set enumeration values you computed in part 1a, formulate and run SQL queries for the 4 cardinal recursive queries. In particular:

- **List all ancestors of NODE.ID = 37570 (Noctuoidea)**
- **List all nodes for which NODE.ID = 37570 (Noctuoidea) is an ancestor. (Include the ancestor node.)**
- **Note: If there are many descendants, this query will return as many rows. Give the total number of matching rows, and include the first 20 in your submission.**
- **List the common ancestors of NODE.ID = 9773 (Megaptera novaeangliae i.e. Humpback Whale) and NODE.ID = 9913 (Bos taurus i.e. cow). Order the common ancestors by increasing distance from the two taxa.**
- **State the most recent common ancestor (MRCA) of NODE.ID = 9773 (Megaptera novaeangliae i.e. Humpback Whale) and NODE.ID = 9913 (Bos taurus i.e. cow).**
- **Note: you can limit the number of rows returned by a query using LIMIT N, where N is a positive integer, as the last clause in an SQL query.**

Submission: For this part, submit code and answers in the form of an Rmarkdown or Jupyter Notebook document, plus its respective HTML rendering. Be sure to include outputs where requested.

Creating Node table

```
# Open connection to database
con = sqlite3.connect("ncbi20170203.db")
cur = con.cursor()

# Create Node table
query1 = """CREATE TABLE IF NOT EXISTS Node1 (
    [ID] INTEGER PRIMARY KEY,
    [Name] TEXT,
    [Parent_ID] INTEGER REFERENCES Node1(ID),
    [Left] INTEGER,
    [RIGHT] INTEGER)"""
cur.execute(query1)
```

```
query2 = """INSERT INTO Node1 (ID, Name, Parent_ID, Left, RIGHT)
SELECT n.id, n.name, n.parent, n.left, n.right
FROM   Node AS n, Node AS p
WHERE  p.name = 'Eukaryota'
AND    n.left >= p.left and n.right <= p.right"""
cur.execute(query2)
```

```
cur.execute("ALTER TABLE Node RENAME TO Node_Original")
```

```
cur.execute("ALTER TABLE Node1 RENAME TO Node")
```

```
cur.execute("CREATE INDEX Node_Parent_Idx ON Node(Parent_ID)")
```

```
cur.execute("CREATE INDEX Node_Name_Idx ON Node(Name)")
```

```
cur.execute("CREATE INDEX Node_Left_Idx ON Node(Left)")
```

```
cur.execute("CREATE INDEX Node_Right_Idx ON Node(Right)")
```

Problem 1a

```
# Create node class
class node:
    def __init__(self, query):
        self.ID = query[0]
        self.name = query[1]
        self.parentID = query[2]
        self.left = query[3]
        self.right = query[4]
        self.children = []


# Select root node
cur.execute("SELECT * FROM Node WHERE Name = 'Eukaryota'")
```

```
## <sqlite3.Cursor object at 0x108d6ac70>
```

```
root = node(cur.fetchone())

# Create function to calculate enumeration values and update in database
def nestedSet(root: node, nestedNumber: int):
    root.left = nestedNumber
    cur.execute("UPDATE Node SET Left = {} WHERE ID = {}".format(root.left, root.ID))
    cur.execute("SELECT * FROM Node WHERE Parent_ID = {}".format(root.ID))
    for row in cur.fetchall():
        root.children.append(node(row))
    for i in root.children:
        nestedNumber = nestedSet(i, nestedNumber + 1)
    root.right = nestedNumber + 1
    cur.execute("UPDATE Node SET RIGHT = {} WHERE ID = {}".format(root.right, root.ID
))
    return root.right


# Run function on root node, incrementing from 1
nestedSet(root, 1)
```

```
## 1802156
```

Question 1 - Maximum Nested Set Value

```
# Query maximum nested set value from Node table
cur.execute("SELECT MAX(RIGHT) FROM Node")
```

```
## <sqlite3.Cursor object at 0x108d6ac70>
```

```
print(cur.fetchone()[0])

# Expected maximum nested set value (should be double the amount of nodes)
```

```
## 1802156
```

```
cur.execute("SELECT COUNT(*) FROM Node")
```

```
## <sqlite3.Cursor object at 0x108d6ac70>
```

```
print(cur.fetchone()[0] * 2)
```

```
## 1802156
```

The calculated maximum nested set value and expected maximum are equivalent.

Question 2 - Time/Space Complexity of Algorithm

Assuming the SQL queries are constant, the time complexity of the algorithm is O(N) overall. For each node we traverse to, we create a node object for it and then add all of its children to an instance variable of a list of nodes. Adding all of these children to the instance variable is O(N) complexity (appending to a list is amortized constant time). Then, we loop through the child nodes and make our recursive call for each one. We repeat this same process for every child of every child and so on and so forth, but we visit each node once. These are two O(N) operations done separately, so we add them together to get O(2N) -> O(N) overall. This has the same overall time complexity of a recursive depth first, which should only have to touch every node once, and is the best possible algorithm for this problem.

Space complexity is also O(N). If we assume (in the absolute worst case scenario) that every node in the tree has one parent and one child, we would have to store N nodes at once for a DFS algorithm.

Question 3 - Listing names and nested set values

```
cur.execute("SELECT * FROM Node WHERE ID = 37570")
```

```
## <sqlite3.Cursor object at 0x108d6ac70>
```

```
node1 = node(cur.fetchone())
print(
    "For Node ID 37570, the Name is {}, the Left Enumeration value is \
{}, and the Right Enumeration value is {}.".format(
        node1.name, node1.left, node1.right
    )
)
```

```
## For Node ID 37570, the Name is Noctuoidea, the Left Enumeration value is 1291407,
and the Right Enumeration value is 1318266.
```

```
cur.execute("SELECT * FROM Node WHERE ID = 451864")
```

```
## <sqlite3.Cursor object at 0x108d6ac70>
```

```
node2 = node(cur.fetchone())
print(
    "For Node ID 451864, the Name is {}, the Left Enumeration value is \
{}, and the Right Enumeration value is {}.".format(
        node2.name, node2.left, node2.right
    )
)
```

```
## For Node ID 451864, the Name is Dikarya, the Left Enumeration value is 413696, and
the Right Enumeration value is 661523.
```

```
cur.execute("SELECT * FROM Node WHERE ID = 1940571")
```

```
## <sqlite3.Cursor object at 0x108d6ac70>
```

```
node3 = node(cur.fetchone())
print(
    "For Node ID 1940571, the Name is {}, the Left Enumeration value is \
{}, and the Right Enumeration value is {}.".format(
        node3.name, node3.left, node3.right
    )
)
```

```
## For Node ID 1940571, the Name is Gyrodactylus botnicus, the Left Enumeration value
is 704285, and the Right Enumeration value is 704286.
```

Problem 2a

Question 1 - List all ancestors of NODE.ID = 37570 (Noctuoidea)

```
query1 = """SELECT p.id, p.name
FROM   Node AS n, Node AS p
WHERE  n.name = 'Noctuoidea'
AND    n.left >= p.left and n.right <= p.right"""
cur.execute(query1)
```

```
## <sqlite3.Cursor object at 0x108d6ac70>
```

```
print(cur.fetchall()[1:])
```

```
## [(104431, 'Obtectomera'), (37567, 'Ditrysia'), (41197, 'Heteroneura'), (41196, 'Ne
olepidoptera'), (41191, 'Glossata'), (7088, 'Lepidoptera'), (85604, 'Amphiesmenoptera
'), (33392, 'Endopterygota'), (33340, 'Neoptera'), (7496, 'Pterygota'), (85512, 'Dico
ndylia'), (50557, 'Insecta'), (6960, 'Hexapoda'), (197562, 'Pancrustacea'), (197563,
'Mandibulata'), (6656, 'Arthropoda'), (88770, 'Panarthropoda'), (1206794, 'Ecdysozoa'
), (33317, 'Protostomia'), (33213, 'Bilateria'), (6072, 'Eumetazoa'), (33208, 'Metazo
a'), (33154, 'Opisthokonta'), (2759, 'Eukaryota')]
```

Question 2 - List all nodes for which NODE.ID = 37570 (Noctuoidea) is an ancestor. (Include the ancestor node.)

```
query2 = """SELECT n.id, n.name
FROM   Node AS n, Node AS p
WHERE  p.name = 'Noctuoidea'
AND    n.left >= p.left and n.right <= p.right"""
cur.execute(query2)
```

```
## <sqlite3.Cursor object at 0x108d6ac70>
```

```
print("The total number of matching rows is {}.".format(len(cur.fetchall())))

# Printing the first 20 rows, including the ancestor
```

```
## The total number of matching rows is 13430.
```

```
cur.execute(query2)
```

```
## <sqlite3.Cursor object at 0x108d6ac70>
```

```
ancestor = cur.fetchall()[-1]
cur.execute(query2)
```

```
## <sqlite3.Cursor object at 0x108d6ac70>
```

```
all = cur.fetchall()[0:19]
all.insert(0, ancestor)
print(all)
```

```
## [(37570, 'Noctuoidea'), (62934, 'Acronicta sp. near modica Mitter 166'), (95176, '
Acronicta sp. near pruni Mitter 18'), (214017, 'Acronicta dactylina'), (214018, 'Acro
nicta hasta'), (214019, 'Acronicta lepusculina'), (214020, 'Acronicta morula'), (2140
23, 'Acronicta impressa'), (663890, 'Acronicta hastulifera'), (685348, 'Acronicta tri
tona'), (688356, 'Acronicta americana'), (688357, 'Acronicta fragilis'), (688358, 'Ac
ronicta funeralis'), (688359, 'Acronicta innotata'), (688360, 'Acronicta interrupta')
, (688361, 'Acronicta retardata'), (688362, 'Acronicta superans'), (688363, 'Acronict
a tristis'), (688467, 'Acronicta afflicta'), (688468, 'Acronicta haesitata')]
```

Question 3 - List the common ancestors of NODE.ID = 9773 (Megaptera novaeangliae i.e. Humpback Whale) and NODE.ID = 9913 (Bos taurus i.e. cow). Order the common ancestors by increasing distance from the two taxa

```
query3 = """SELECT p.id, p.name
FROM   Node AS n1, Node AS n2, Node AS p
WHERE  n1.name = 'Megaptera novaeangliae' and n2.name = 'Bos taurus'
AND    n1.left >= p.left and n1.right <= p.right
AND    n2.left >= p.left and n2.right <= p.right
ORDER BY p.right ASC"""
cur.execute(query3)
```

```
## <sqlite3.Cursor object at 0x108d6ac70>
```

```
print(cur.fetchall())
```

```
## [(91561, 'Cetartiodactyla'), (314145, 'Laurasiatheria'), (1437010, 'Boreoeutheria'
), (9347, 'Eutheria'), (32525, 'Theria'), (40674, 'Mammalia'), (32524, 'Amniota'), (3
2523, 'Tetrapoda'), (1338369, 'Dipnotetrapodomorpha'), (8287, 'Sarcopterygii'), (1175
71, 'Euteleostomi'), (117570, 'Teleostomi'), (7776, 'Gnathostomata'), (7742, 'Vertebr
ata'), (89593, 'Craniata'), (7711, 'Chordata'), (33511, 'Deuterostomia'), (33213, 'Bi
lateria'), (6072, 'Eumetazoa'), (33208, 'Metazoa'), (33154, 'Opisthokonta'), (2759, '
Eukaryota')]
```

Question 4 - State the most recent common ancestor (MRCA) of NODE.ID = 9773 (Megaptera novaeangliae i.e. Humpback Whale) and NODE.ID = 9913 (Bos taurus i.e. cow)

```
query4 = """SELECT p.id, p.name
FROM   Node AS n1, Node AS n2, Node AS p
WHERE  n1.name = 'Megaptera novaeangliae' and n2.name = 'Bos taurus'
AND    n1.left >= p.left and n1.right <= p.right
AND    n2.left >= p.left and n2.right <= p.right
ORDER BY p.right ASC LIMIT 1"""
cur.execute(query4)
```

```
## <sqlite3.Cursor object at 0x108d6ac70>
```

```
print(cur.fetchone())
```

```
## (91561, 'Cetartiodactyla')
```