COMP4019 - Lab Session 4 – Red-Black Trees; Dynamic Programming

Xavier Carpent & Ian Knight

October 28, 2021

Red-Black Trees

Implement a Red-Black tree data structure, with only the `insert` and `traverse` operations (deleting and other operations do not need to be implemented at this stage).

Measuring Complexity Experimentally

In order to confirm the complexity of data structure procedures and algorithms, it is often useful to experimentally measure the efficiency of their implementation.

There are several ways to go about it. For instance, have a global "operation count" that is incremented whenever a time-consuming operation is performed (e.g. every time a recursive function is called, inside loops, etc.) and measured when the procedure is terminated (this is a form of "code instrumentation"). Alternatively, the CPU time can be measured (example in python below):

```python
import time

start = time.time()
do_stuff(data)
end = time.time()

print(end - start)
```

Try to measure the efficiency of your implementation of the `traverse` (easier) and `insert` (trickier) operations. Some important points to consider:

- How many measurements do you need for statistical significance?
- Static input versus random input
- For instrumentation: when to increment the counter?
- For timing: what to measure? what other programs are running?

Observe the results. Plotting (you can for instance use a plotting library in your favorite programming language, use excel, or an online tool to plot sequences) may help interpret the results graphically. Do they confirm your expectations? Experiment and comment.

Dynamic Programming - Regex Matching

Regular Expressions are a useful tool in many applications. They are used to determine whether or not an input string `s` matches a *pattern* string `p`. Some examples of patterns:

- `[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}` matches email addresses;
- `<xml>(.*?)</xml>` matches the content of a (non-empty) `xml` tag;
- `(0?[1-9]|[12][0-9]|3[01])/(0?[1-9]|1[0-2])/\d{4}` matches dates.

Consider the simplified problem where the pattern may only contain letters `[a-z]`, the `?` character which matches any letter, and the `*` character, which matches any number (including 0) of letters.

1. Dynamic programming could be a useful paradigm to implement this problem... why? (*DP is useful when the problem has two specific properties*).
2. Implement the problem in your favorite language. Determine its time complexity. How much memory does your implementation require?