Algorithms

Christopher Rodriguez

Colorado State University Global

WC21-CSC200-2: Computer Science Fundamentals

Prof. Chintan Thakkar

2022-02-06

Algorithms

If the essence of computation is the transformation of data representation (Erwig, 2017), *how* that data is transformed is surely the mainstay of the science of computation. This paper dives into the idea of algorithms on an abstract level as the tools we use to manipulate data and solve problems, as well as outlining a few common household tasks with an algorithm and brief explanation. The problems this paper outlines are:

1. Making a Peanut Butter and Jelly Sandwich.
2. Getting Up in the Morning.
3. Doing Your Homework.
4. Driving Home in the Afternoon.

All problem solving uses algorithms, whether they are explicitly referred to as such or not. Understanding their place in the human experience will only enable better solutions to be found.

What is an Algorithm?

Lovelace believed in tempering our approach to science with intuition (Lai, 2017, p.2). By using our inherent creativity as humans to approach the constructs present in an otherwise abstract model---and indeed, to use that creativity to create such models in the first place---we would be able to achieve progress limited only by our collective imagination. And the *way* we can do such things is by understanding, and using, algorithms.

In Lovelace's Poetical Science, there were three main features: Observation, Interpretation, and Integration (Lai, 2017, p.3). If we can draw a thread from Interpretation, where models are built and analysis applied, to Data Structures, then it follows that Integration is the realm of the Algorithm: Applying pragmatic applications to those abstract models, connecting disparate ideas

by presenting data in new ways, and most importantly allowing solutions to become part of a more cohesive whole. Algorithms let us take the data we have and make actual use of it.

In many ways, Algorithms are the verbs of Computer Science: With a clearly defined algorithm, different computers can perform a specific computation in a reliable, repeatable way (Erwig, 2017, p.8). We connect data to build new and useful things by putting that data through a rigorously specific set of computations. Reproducability, reliability, and reasonability in our code is provided almost entirely through the selection and implementation of the correct (for the problem at hand) algorithm.

The following are examples of such algorithms for common, everyday tasks, and brief explanations thereof. These tasks are ``problems" most will be familiar with. Few of us consider such mundane tasks from an abstract-enough viewpoint to outline such an algorithm, but each and every person has an algorithm they use (implicitly) to get these things done.

Sandwiches

We'll first turn our attention to Making a Peanut Butter and Jelly Sandwich.

```
start
   if not (gather ingredients(bread-slice,
                              bread-slice,
                              peanut-butter,
                              jelly,
                              knife,
                              plate)
   then
      abort-and-go-shopping()
   end if
   put bread on plate as bread[0]
   put bread on plate as bread[1]
   while(bread[0].needs-peanut-butter())
     do
        use knife on peanut-butter
        apply knife to bread
     done
   while(bread[1].needs-jelly())
     do
        use knife on jelly
        apply knife to bread
     done
   append bread[1].inverted() to bread[0] as sandwich
   use knife on sandwich as half[0] and half[1]
end
```

This is a fairly straightforward algorithm to consider; It is very rare that any deviation or flow control is needed in such a straightforward (and destructive!) process. We start by gathering all of the materials which we will manufacture into our finished sandwich (2 slices of bread, a jar

of peanut butter, and a jar of jelly) and the tools we'll need to do so correctly (a plate and a knife). For the purposes of this exercise we are assuming these are already existent and defined.

We then move onto the steps. There are only two points at which we will judge whether we are ready to continue or not: After applying the Peanut Butter, and after applying the Jelly. The condition we are going to check for is the amount that has been applied: If more is needed, the step should be repeated until we ``have enough''. Otherwise, it is simply a linear progression of steps as outlined in Figure 1.

It is worth mentioning here that, in our example, we are not considering the sandwich made until it has been sliced into two halves. We will not go into the controversy on this point; Some consider this step optional. That is not the case here.

Waking Up

For this algorithm, we are actually going to break up the process into a few different, smaller sections---each algorithms in their own right---to make it easier to conceptualize the phases of our process. These are the yellow blocks beneath the normal steps in Figure 2.

```
start
boolean workday = is-today-a-work-day?()
boolean eat-out = will-i-eat-out?()
start lounge:
    if not workday then
        check-email()
        roll-over()
        sleep()
    end if
    leave-bed()
end
get-dressed(underclothes,pants,shirt,hat)
start food-and-drink
    start-tea-water()
    if eat-out then
        order-breakfast()
    else
        make-toast()
    end if
end
start leave
    check-backpack(laptop, notebook)
    pet-cats(scout, psymon)
    leave-apartment()
end
return startday()
end
```

There are three main phases: Lounging Around, Food and Drink, and Leaving the Apartment. How we enter and leave each section is modified slightly by the other big departure from our sandwich algorithm: Decisions. At two crucial moments in our overall algorithm, a choice is made:

First, we decide whether or not we are going into work today. And second, we decide if we want to eat out today.

If it is a work day, unfortunately, the Lounging Around step is cut very short, by simply getting out of bed. If we are eating out, we do not need to make toast---likewise, if we are not eating out, we do not need to order food. These steps are skipped when appropriate.

Finally, there are some steps in the process---like getting dressed, for instance---that are executed no matter what. After all, we can't leave the house if we haven't gotten dressed.

Doing Homework

Homework is a fairly variable process: At times, papers need to be written. At other times, papers need to be read. The algorithm outlined here assumes the function itself is already defined: There is no need to outline *how* a paper gets written. Instead, focus is put on the decision-making process, and iterating over a loop.

```
start
homework-log = gather-all-homework()
done-log = make-empty-stack()
while homework-log.is-not-empty()
    do
        switch pop!(homework-log) as x
            case: x == paper
                  x.write()
                  done-log.push!(x)
            end
            case: x == note
                  x.review()
                  done-log.push!(x)
            end
            case: x == discussion
                  x.respond()
                  done-log.push!(x)
            end
            case: x == article
                  x.study()
                  done-log.push!(x)
            end
            default:
                  x.research()
                  done-log.push!(x)
            end
        end
    done
submit(done-log)
end
```

There are four defined homework functions here: Write Paper, Post Discussion, Study Article, and Review Notes. It is assumed there is a stack of work to do (Assignments), and once it is done, the assignments need to be Submitted. This flow can be seen in Figure 3

While there are still assignments to do, we decide which kind of assignment it is: A paper, an article, a discussion, or a note. Depending on that type, we then move to a specific action, as outlined above. Once that action is complete, we return to pulling another task off of the pile, until it is empty. When it is empty, we submit everything.

Driving Home

The final algorithm to be presented here is that for driving home in the afternoon. This algorithm will highlight the concept of an exception.

```
start
car.enter()
car.start()
begin navigate-home
    car.drive(home)
    if event == accident
        call deal-with-accident()
        return -1
    else if car.drive(home).path ≠ open
        car.stop()
        load(new-path(home))
        check(path)
        navigate-home()
    else if traffic > normal-traffic
        car.speed.lower()
        load(traffic-report)
        set drive-style = defensive
        navigate-home()
    end
end
car.stop()
car.exit()
end
```

Driving, on the surface, seems very straightforward: Get in the car, turn it on, drive home, turn it off, get out. However, the most complicated wrinkle to this problem is the myriad of other people trying to do the same thing, and the additional problems that might therefore be caused.

Limiting the possible problems to a reasonable number: An accident could happen, traffic could build up, and a road could be closed. If any of these things happen, our process will need to change: Traffic requires more defensive, attentive driving. A road closure requires a new route to be chosen. And an accident aborts the concept of driving altogether, leading us towards the entirely separate process of dealing with insurance and possibly worse issues.

How we move between these issues can be seen in Figure 4. Note that the algorithm does *not* always end in success!

Conclusion

Algorithms are the verbs of Computer Science. They are *what we do* that makes computation---that is, the transformation of our data and how it is represented---happen. Without algorithms, we could not solve even the simplest of problems. And algorithms are everywhere in our daily lives.

References

Erwig, M. (2017, August 11). *Once upon an algorithm: How stories explain computing*. MIT Press. https://doi.org/10.7551/mitpress/10786.001.0001

Lai, A. (2017). Ada Lovelace: `poetical scientist'. *OR/MS Today*, *44*(1). Retrieved January 23, 2022, from https://www.informs.org/ORMS-Today/Public-Articles/February-Volume-44-Number-1/Ada-Lovelace-poetical-scientist

List of Figures

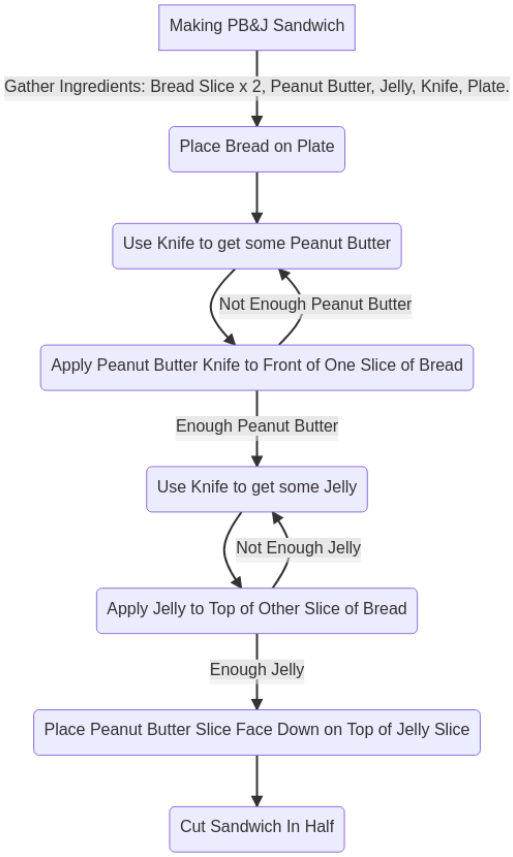

Figure 1: Making a Peanut Butter and Jelly Sandwich

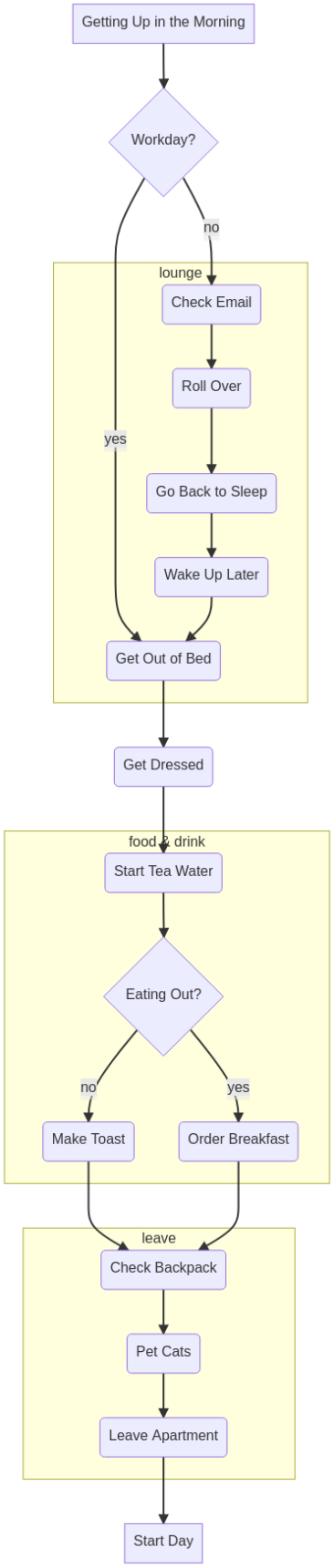

Figure 2: Getting Up for the Day

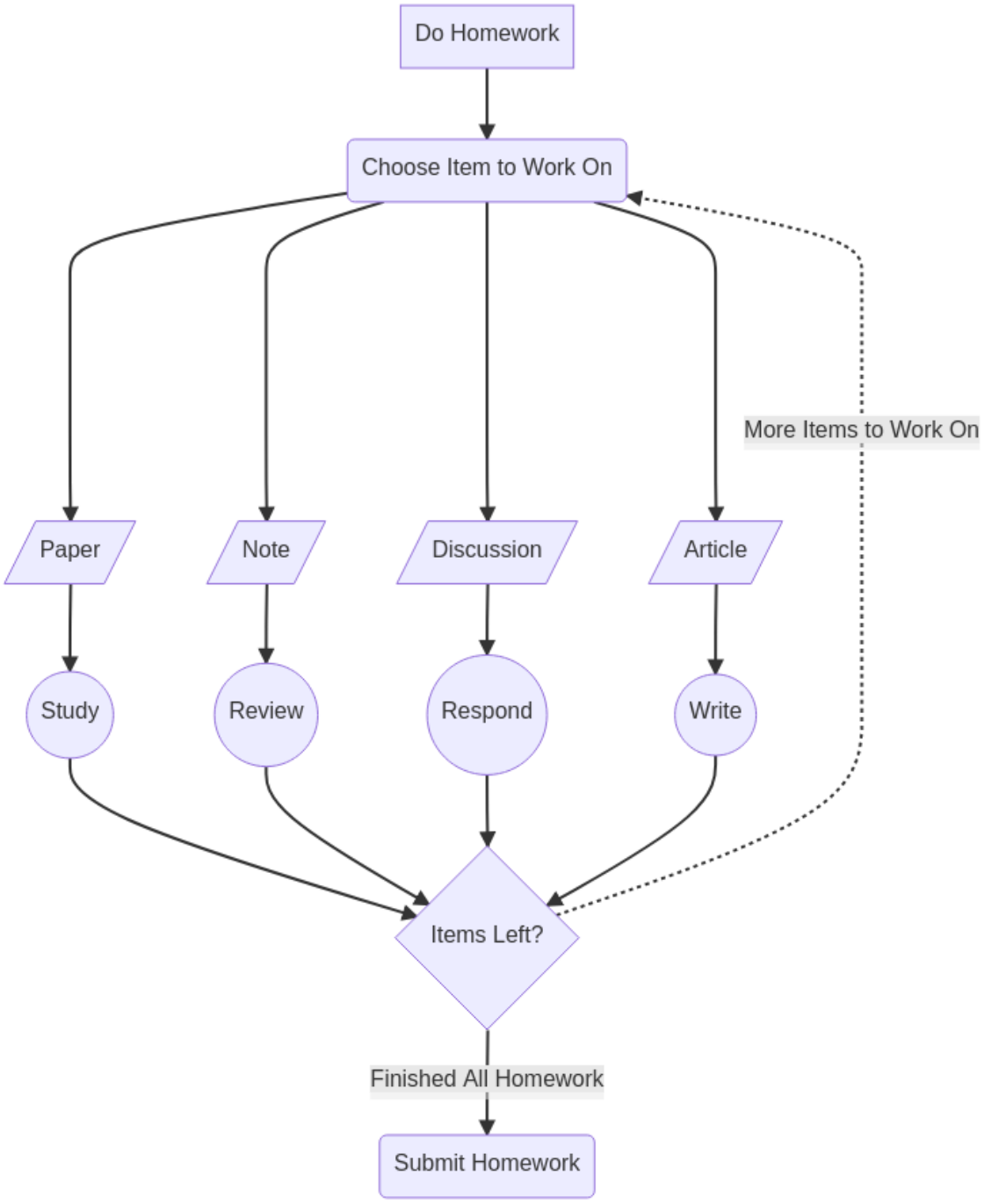

Figure 3: Doing Homework

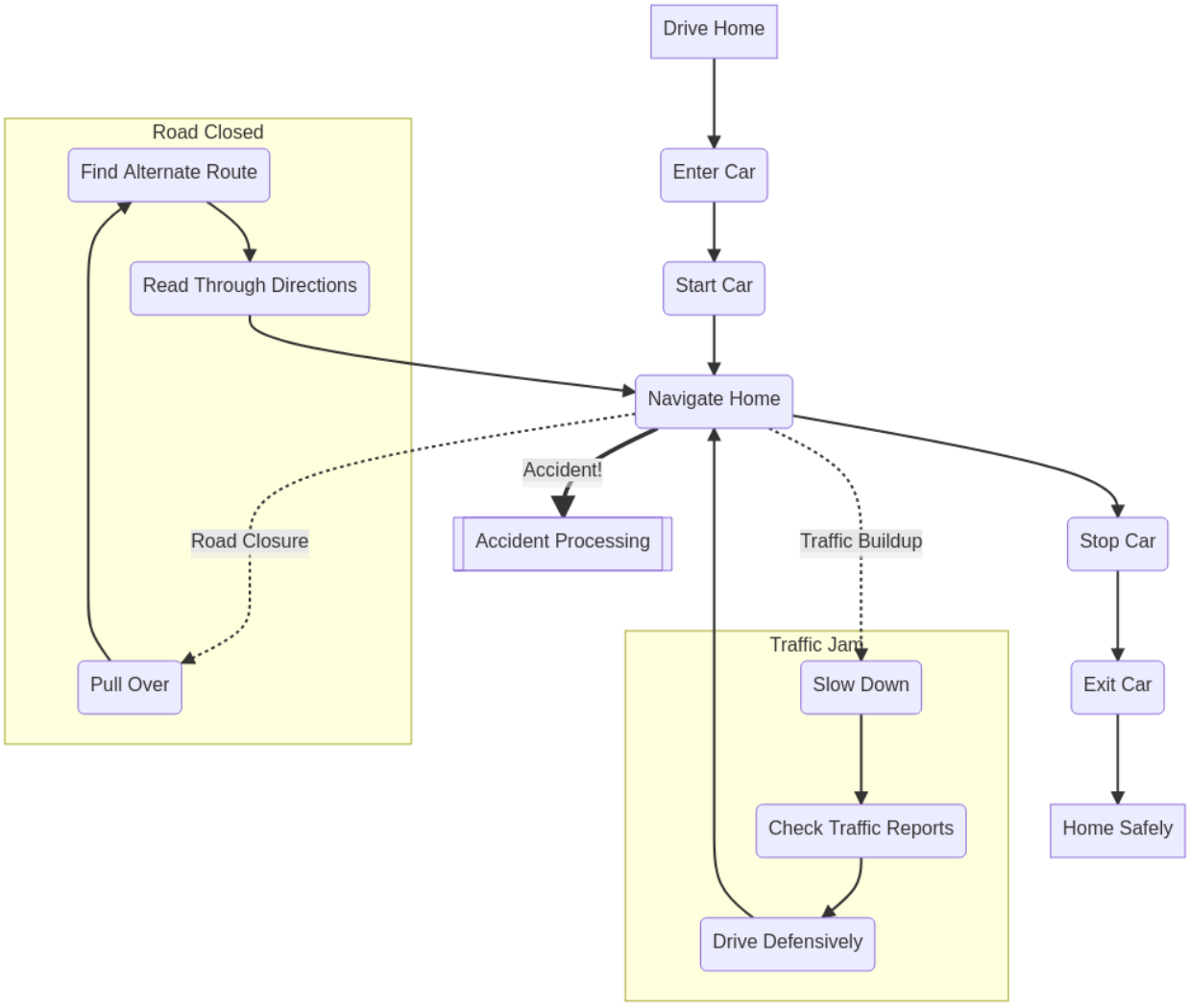

Figure 4: Driving Home in the Afternoon