COT4400 - Group 11
Project Report

1. How can you break down a problem instance of the omnidroid construction problem into one or more smaller instances? You may use sprocket[t] to represent the number of sprockets used for part t, and you may use req[t] and use[t] to represent the collection of all parts required to build part t and all parts that part t is used to build, respectively. Your answer should include how the solution to the original problem is constructed from the subproblems.

   The omnidroid construction problem requires subproblems to be solved in order to build the final solution. The omnidroid itself can be considered to be built from the bottom up where each intermediate component is constructed using other components and then also as parts to the whole omnidroid (an example being the arm assembly being built using two articulated joints and then being used in the construction of the omnidroid itself). From this we see that in order to count for the total number of sprockets, recursion in the form of a depth-first search would be a likely solution as you must check each component for dependencies, if any, before totaling the count for each subproblem.

2. What are the base cases of the omnidroid construction problem?

   If the construction of the omnidroid were composed of only a basic component or none. Solution at index of m would be equal to the number of sprockets required for one component.

3. How can you break down a problem instance of a robotomaton construction problem into one or more smaller instances? You should assume that you are given sprockets and previous arrays that indicate the number of sprockets required for each stage of construction and the number of previous stages used to construct a particular part. Your answer should include how the solution to the subproblems are combined together to solve the original problem.

   The problem is separated and handled stage by stage where each line of the input is giving us a new solution and when the value in the second column is >= 1 we recurse backwards and obtain the solutions for however many recursions/stages to apply to the current stage.

4. What are the base cases of the robotomaton construction problem?

   Case 1 (Nested cases): When N (Number of stages) - 1 = 0 and P (Previous stages) = 0

5. What data structure would you use to recognize repeated problems for each problem (two answers)? You should describe both the abstract data structures, as well as their implementations.

   Our solution to finding the output here would ideally be using a 2D array with an adjacency list or map to navigate through the inputs and sum them up according to their stages. The 2D array based map or adjacency list would be used to store the values and to look up the values for subproblems during recursion.

   What we actually did for our solution is use multiple 1D arrays to store the solutions and dependencies (such as i and j) and look up values for subproblems as we recursively went through the function.

6. Give pseudocode for a memoized dynamic programming algorithm to calculate the sprockets needed to construct an omnidroid.

```
Input: n, m
Output: number of sprockets in omnidroid construction
Algorithm: SprocketHelper
soln = Array(n, m)
Initialize soln to 0
Return omnidroid(n, m)

Input: n, m, i, j, parts, soln(n, m)
Output: soln(n, m)
Algorithm: Omnidroid
if soln(n, m) = 0
    if m = 0
        soln(j) += i(parts)
    else
        omnidroid(n, m-1)
        soln(j) += i(parts)
```

7. What is the worst-case time complexity of your memoized algorithm for the omnidroid construction problem?

   The worst–case time complexity of our omnidroid problem would be $O(n + m)$ because we have to assign the n values into the solution array, in the helper function, then in the omnidroid function go through all m values to store the values into specific arrays.

8. Give pseudocode for a memoized dynamic programming algorithm to calculate the sprockets needed to construct a robotomaton.

```
Input: stages, inputFile, i, j
Output: reading values into i and j
Algorithm: RobotomatonReader
while count < stages
        scan line for values into x and y
        push x and y into i and j respectively
        count++

Input: stages, i, j
Output: soln(stages)
Algorithm: RobotomatonHelper
soln = Array(stages)
Initialize soln to 0;
return Robotomaton(stages)

Input: stages, i, j, soln(stages)
Output: soln(stages)
Algorithm: Robotomaton
if soln(stages) = 0
        if stages = 0
                soln(stages) = i(stages)
        else if y = 0
                Robotomaton(stages-1)
                soln(stages) += i(stages)
        else if y = 1
                Robotomaton(stages-1)
                soln(stages) +=  i(stages) + robotomaton(stages-1)
        else
                robotomaton(stages-1)
                soln(stages) += i(stages)
                Add the previous j stages to soln(stages)
```

9. Give pseudocode for an iterative algorithm to calculate the sprockets needed to construct a robotomaton. This algorithm does not need to have a reduced space complexity, but it should have asymptotically optimal time complexity.

```
for( int a = 0; a < soln.size; a++)
        count += soln[a];
```

```
For each iteration of the soln array:
    Add into a count variable
```