

Dynamic Programming Project

COT 5407, Summer 2024

Due August 1, 2024

1 Overview

For this project, you will develop an algorithms to evaluate the cost to produce a robot. Designing and implementing these algorithms will require you to model the problem using dynamic programming, then understand and implement your model.

You are only allowed to consult the class slides, the textbook, the TAs, and the professor. **In particular, you are not allowed to use the Internet.** This is a group project. The only people you can work with on this project are your group members. This policy is strictly enforced.

The first requirement for this project will be to make a post on the Meet Your Group discussion (posted in Week 4 module) by July 21. You should, at a minimum, give your name and some form of contact information to your groupmates they can use to get in touch with you (e.g., email or phone number). You are encouraged to make plans for an initial meeting where you can start brainstorming how to solve the project, though this is not required.

In addition to the discussion post and group submission, you will also evaluate your teammates' cooperation and contribution. These evaluations will form a major part of your grade on this project, so be sure that you respond to messages promptly, communicate effectively, and contribute substantially to your group's solution. You will submit the peer evaluations to another assignment on Canvas, labeled "Dynamic Programming Project (peer evaluation)."

A word of warning: this project is team-based, but it is quite extensive and a nontrivial task. You are highly encouraged to start working on (and start asking questions about) this project early; teams who wait to start until the week before the due date may find themselves unable to complete it in time.

2 Problem Description

For this project, you will be given an input file describing how to assemble a robot known as an omnidroid, and all of the intermediate products required to construct them, and you will need to compute the total cost required to make these robots.

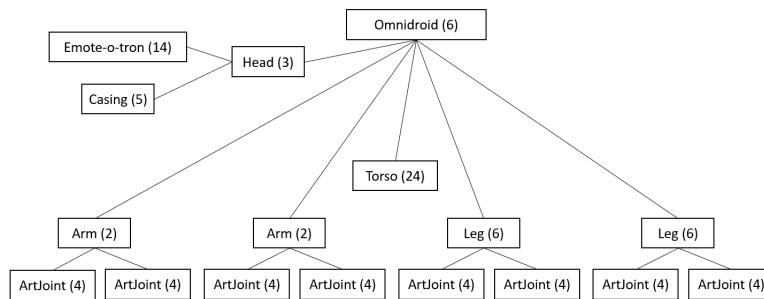
3 Constructing omnidroids

The schematic for building an omnidroid will include a list of parts that need to be combined to assemble the omnidroid as well as the number of sprockets required to attach that part.

Sprockets represent the main material component of robots, and the cost for an omnidroid should be computed in terms of sprockets used. Parts for omnidroids come in two varieties, basic and intermediate. Basic parts are constructed by combining together some number of sprockets in the correct configuration, whereas intermediate parts are constructed via their own assembly process, where other basic and/or intermediate parts are attached to the construct via sprockets.

For example, omnidroid model T9001 is constructed by combining a head assembly, torso as-assembly, and two arm assemblies, and two leg assemblies using 6 sprockets. Head assemblies are constructed by combining an Emote-o-tron computer (14 sprockets) and skull casing (5 sprockets) using 3 sprockets. The torso is a basic part constructed from 24 sprockets. Arms and legs are built from two articulating joint assemblies (4 sprocket each), using 2 sprockets for arms and 6 sprockets for legs. In total, each T9001 omnidroid is built from 100 sprockets.

A diagram describing how to build the omnidroid appears below:



4 Input format

Your program should read its input from the file input.txt, in the following format.

The first line of the file will have two positive integers n and m , indicating the number of parts in the assembly and the number of assembly dependencies, respectively. The next m lines will have two nonnegative integers i and j , representing that part i is used in the assembly of part j . For convenience, the parts are numbered 0 to $n - 1$ instead of being identified by names like “Emote-o-tron”, with part $n - 1$ representing the finished omnidroid. Every intermediate part will appear as the second entry in at least one line, while basic parts will not appear as the second entry in a line. The n lines following the part dependencies will have a single integer indicating how many sprockets are used to create an omnidroid part outside of any other parts used in the construction (in order, part 0 to part $n - 1$).

For example, if we label the 8 parts used the omnidroid T9001 as: 0 articulated joint, 1 arm assembly, 2 leg assembly, 3 torso, 4 Emote-o-tron, 5 skull casing, 6 head assembly, and 7 omnidroid, the input for T9001 would be as follows:

```

8 12
0 1
0 1
0 2
0 2
4 6
5 6
1 7
1 7

```

2 7
2 7
3 7
6 7
4
2
6
24
14
5
3
6

Note: as in this example, the same part may be used multiple times in the construction of another part. Also (not shown in this example), parts with larger ID values may be used as inputs to parts with smaller ID values. You may assume the assembly instructions will never be impossible to satisfy (e.g., part 1 is constructed from part 2, which is constructed from part 1), and the overall omnidroid (ID $n - 1$) will not be used to build any other part.

4.1 Output

Your program should write its output to the file `output.txt`. Your program should output the total number of sprockets needed to build the omnidroid. For example, if the input file described omnidroid T9001, the output file should contain:

100

Implementation note: when solving these robot construction problems, the number of sprockets used to construct a robot may exceed 2^{32} , but it will be less than 2^{64} . All of the numbers in the input files will be smaller than 2^{32} .

5 Project report

In your project report, you should include brief answers to 5 questions. Note that you must use dynamic programming to solve this problem; other solutions will not receive substantial credit.

1. How you can 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.
2. What is the base cases of the omnidroid construction problem?
3. What data structure would you use to recognize repeated problems for omnidroids? You should describe both the abstract data structure, as well as its implementation.
4. Give pseudocode for a memoized dynamic programming algorithm to calculate the sprockets needed to construct an omnidroid.
5. What is the worst-case time complexity of your memoized algorithm?

6 Coding your solutions

In addition to the report, you should implement a dynamic programming algorithms that can find the number of sprockets used in omnidroids. Your code may be iterative or recursive, but both solutions must use dynamic programming. Also, you may code your solution in C++ or Python, but it must compile and run in a Linux environment. If you are using C++ and compiling your code cannot be accomplished by the command

```
g++ -o robot *.cpp
```

you should include a Makefile that is capable of compiling the code via the make command.

Whether you use C++ or Python, your source code may be split into any number of files. In addition, your code will not be tested on input (e.g., omnidroid requirements that include invalid part IDs or omnidroids that cannot be constructed).

7 Submission

Your submission for this project will be in two parts, the group submission and your individual peer evaluations.

7.1 Group submission

The submission for your group should be a zip archive containing 1) your report (described in Section 5) as a PDF document, and 2) your code (described in Section 6). If your code requires more than a simple command to compile and run then you must also provide a Makefile and/or shell script. You should submit this zip archive to the “Dynamic Programming Project (group)” assignment on Canvas.

Be aware that your project report and code will be checked for plagiarism.

7.2 Teamwork evaluation

The second part of your project grade will be determined by a peer evaluation. Your peer evaluation should be a text file that includes 1) the names of all of your teammates (including yourself), 2) the team member responsibilities, 3) whether or not your teammates were cooperative, 4) a numeric rating indicating the proportional amount of effort each of you put into the project, and 5) other issues we should be aware of when evaluating your and your teammates’ relative contribution. The numeric ratings must be integers that sum to 60.

It’s important that you be honest in your evaluation of your peers. In addition to letting your team members whether they do (or do not) need to work on their teamwork and communication skills, we will also evaluate your group submission in light of your team evaluations. For example, a team in which one member refused to contribute would be assessed differently than a team with three functioning members.

You should submit your peer evaluation to the “Dynamic Programming Project (peer evaluation)” assignment on Canvas.

8 Grading

Group discussion post	5 points
Report	35 points
Questions 1 and 4	10 each
Questions 2, 3, and 5	5 each
Code	30 points
Compiles	5
Uses correct input and output format	5
Computes correct answer	15
Good coding style	5
Teamwork	30 points

Note that if your algorithms are inefficient, you may lose points for both your pseudocode and your submission. Also, in extreme cases, the teamwork portion of your grade may become negative or greater than 30. In particular, if you do not contribute to your group's solution at all, you can expect to receive an overall grade of 0 on the project.