Project 2: Dynamic programming

CS 590, Spring 2025

Due April 21, 2025

1 Overview

For this project, you will develop an algorithm to solve a problem related to the board game *On Tour*. Designing and implementing this solution will require you to model the problem using dynamic programming, then understand and implement your model.

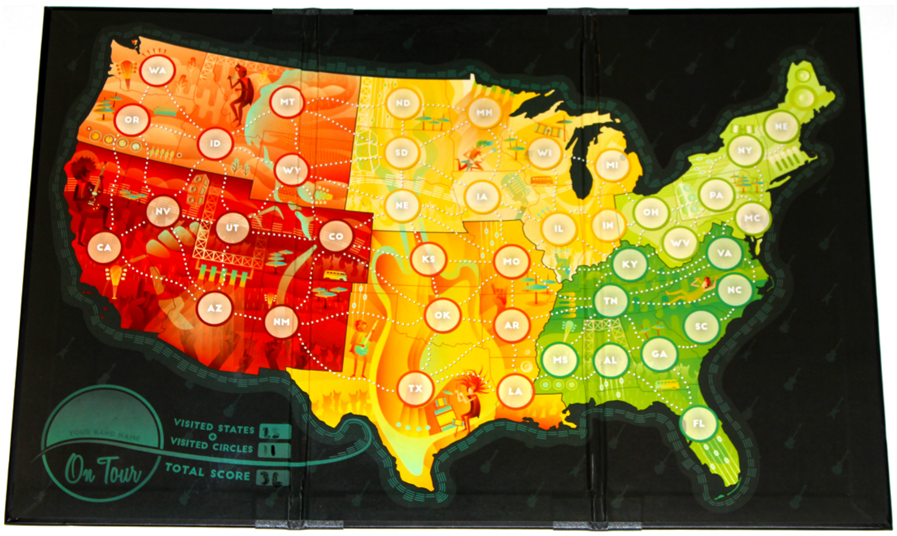
You are only allowed to consult the class slides, the textbook, the CAs, and the professor. In particular, you are not allowed to use the Internet. This is a group project. The only people you can work with on this project are your group members. This policy is strictly enforced.

In addition to the 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. Details for your team evaluations are in Section 5.2. You will submit the peer evaluations to another assignment on Canvas, labelled "Project 2 (individual)."

A word of warning: this project is team-based, but it is 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.

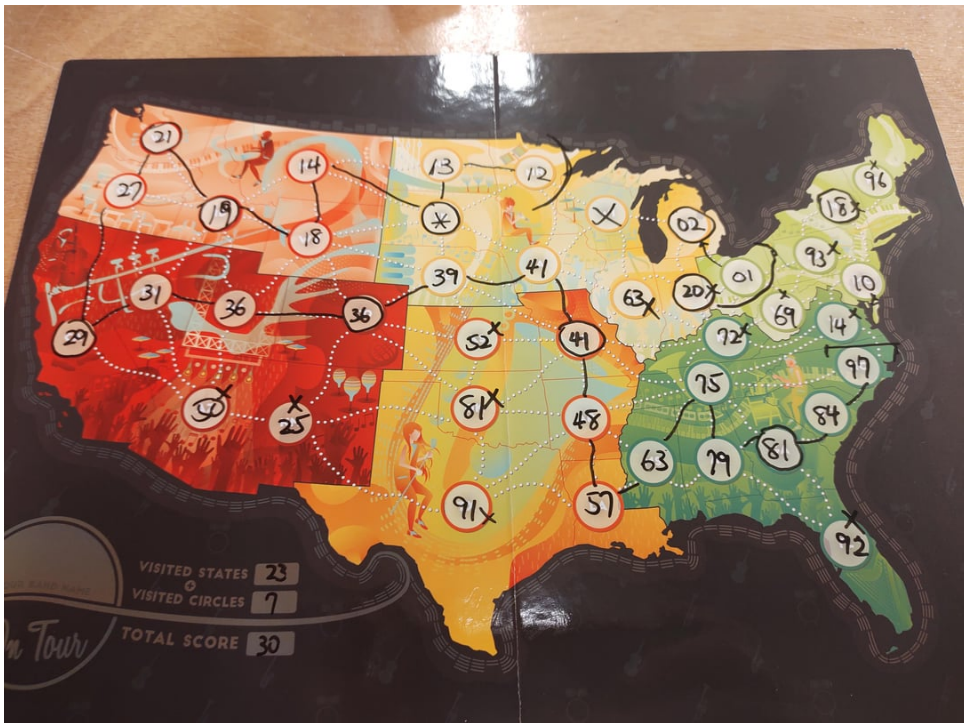
2 Problem Description

In the board game *On Tour*, players are given a map with various spaces that they will fill with numbers over the course of the game. One of the maps for the game consists of the US states, shown below:



At the end of the game, players will score points based on the longest "tour" they can form in which all of the numbers on the tour increase or stay the same. In addition, players have a chance to circle some of the spaces on the board, which makes these locations worth twice as many points as other locations. The player with the greatest number of points (total number of spaces connected, with an extra point for each circled location) wins the game.

An example of a completed game (with the scoring path highlighted) appears below:



This player created a path of length 23 that passed through 7 circled states, for a total of 30 points. Your algorithm should be able to take in information representing a completed game board and output the maximum possible score for the game as well as the path that achieves this score. In the example above, 30 points is indeed the highest score that this player could make.

In the game, some spaces may be marked as wild cards or unusable ($*$ or X above). It turns out that if we allow wildcard spaces or adjacent spaces with the same number, this problem may potentially require an exponential amount of time to solve. Therefore, you only need to solve the restriction of this problem that does not allow wildcards or equal values for adjacent spaces. We can edit the example above to meet these restrictions by modifying the value of South Dakota ($*$) to 14, Montana (14) to 15, Colorado (circled 36) to 37, and Missouri (circled 41) to 42. In addition, you may assume that any unusable spaces (Wisconsin, in this example) are omitted.

3 Project report

In your project report, you should include brief answers to 7 questions. Your answer to questions 1–7 should focus on the problem of calculating the maximum score possible *when starting on a specific space*. You should be able to use this problem formulation to answer the question of the maximum possible score for a given board.

Note that you must use dynamic programming to solve these problems; other solutions will not receive significant credit. In addition, there is an optional bonus question that asks how you would tackle the unrestricted version of this problem.

1. How you can break down a large problem instance into one or more smaller instances? Your answer should include how the solution to the original problem is constructed from the subproblems and why this breakdown makes sense.
2. What are the base cases of this problem, and what are their solutions?
3. What data structure would you use to store the partial solutions to this problem? Justify your answer.
4. Give pseudocode for an algorithm that uses memoization to compute the maximum score.
5. What is the time complexity of your memoized algorithm?
6. Give pseudocode for an iterative algorithm for this problem.
7. Describe how you could modify your algorithm to identify the maximum-scoring tour, not just the maximum possible score.

Bonus Describe (briefly) how you would modify your algorithm to account for adjacent equal values and wildcards. There is likely no algorithm that is guaranteed to solve this problem in polynomial time, so just focus on solving the problem correctly rather than quickly.

4 Coding your solutions

In addition to the report, you should implement an *iterative* dynamic programming algorithm that can solve the *On Tour* scoring problem. Your code may be in C++ or Java. If you choose to implement your code in Java, the class with the `main()` function should be called OnTourScorer. Using either language, your source code may be split into any number of files.

4.1 Input format

Your program should read its input from the file `input.txt`, in the following format. The first line of the file will contain a positive integer n representing the total number of spaces on the board (in the case of the US map, 40, as many of the northeastern states plus Alaska and Hawaii don't have spaces on the board).

The remaining n lines of the file will each contain information about one of the spaces on the board. Each line will begin with the numeric value filled in this space, followed by the number of points this space is worth (1 or 2 in the original game, though we can imagine variants where a space may be worth even more). These two will be followed by the number of neighbors for this space, d, as well as d integers representing the spaces to which it is connected. Each space on the board is identified by a positive integer 1 up to n, where space 1 is the first space described in `input.txt` (the second line of the file), while space n is the last. All of the values on the same line will be separated by spaces.

For example, in the example above, the player wrote 21 in the space representing Washington (WA) and circled it (making it worth 2 points). If this space is space 1, while Oregon (OR) and Idaho (ID) are spaces 6 and 7 respectively, the line representing Washington in this example would be:

```
21 2 2 6 7
```

Note that all of these numbers will be positive integers. There are no spaces worth 0 or negative points, and there are no isolated spaces on the map. In addition, while multiple spaces may have the same value, none of the spaces with the same value will be adjacent.

4.2 Output format

Your program should write its output to the file `output.txt`, in two lines. The first line should contain a single integer with the maximum possible score, while the second line should contain the values representing the maximum-scoring tour.

You have been provided with sample files that represent the example game given above, as well as a much smaller example with only 9 spaces on the board.

4.3 Hint

When initially tackling this problem, it may seem impossible to develop a recursive algorithm that doesn't result in an infinite recursion. The problem is more tractable if you restructure the connections between the spaces and/or "renumber" the spaces. If you do either of these, you should describe the modified network and/or how you mapped the original spaces to more useful ID values in your answer to question 1 in the report (see Section 3).

You may also wish to reference the functions $v(i)$ and $p(i)$ in your answer to represent the value written in space i and the number of points it is worth, respectively.

5 Submission

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

5.1 Group submission

The submission for your group should be a zip archive containing 1) your report (described in Section 3) as a PDF document, and 2) your code (described in Section 4). 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 "Project 2 (group)" assignment on Canvas.

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

5.2 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 "Project 2 (peer eval)" assignment on Canvas.

6 Grading

Report	**55 points**
Questions 1, 6, and 7	10 each
Questions 2–5 and bonus	5 each
Code	**20 points**
Compiles and is correct	15
Good coding style	5
Teamwork	**30 points**

Note that if your algorithm is 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 a total grade of 0 regardless of how well your teammates do on the report or code.