CS 170 Efficient Algorithms and Intractable Problems
Spring 2018 A. Chiesa and U. Vazirani

Problem Statement

You live in a foreign country which has been partitioned by the lords of the land into several kingdoms. You are one such lord and wish to grow your domain and become emperor of the entire country. As a battle-hardened war veteran, you know that war is costly and want to complete your conquest as quickly as possible. To aid in your planning, your very talented spies have managed to determine the exact durations of time (in days) that it will take to move your army from any kingdom to any neighboring kingdom and also the time to conquer any given kingdom once your army arrives there. Note that, since you are one of the more feared lords of the land who rules with an iron fist, an invasion by your army strikes fear into the hearts of this countries' citizens; as a result, for any kingdom you conquer, all neighboring kingdoms will surrender and join your empire rather than risk being the next country on your invasion list. Starting at your kingdom, can you figure out the optimal war campaign that conquers the country in the least amount of time possible?

Formally, you can view the problem as a connected, undirected graph. You have a set K of kingdoms, and a function $N(k)$ which returns the set of neighboring kingdoms of kingdom k, for any $k \in K$. You also know the travel costs c_{ij} to go from any kingdom $i \in K$ to any kingdom $j \in N(i)$, and the conquering costs c_{ii} for any $i \in K$. Given the starting kingdom s you must determine a tour $T = \{s, k_1, k_2, k_3, \ldots, k_m, s\}$ through the country, along with which kingdoms along T that you decide to explicitly conquer (let's call this set of conquered territories C), such that the entire country is absorbed by your empire in the minimum time possible. Note that after a kingdom has surrendered, it can still be conquered to get the neighboring kingdoms to surrender. This can be stated as the following optimization problem:

$$\min \sum_{\substack{(i,j)\in T \\ j\in N(i)}} c_{ij} + \sum_{i\in C} c_{ii}$$

$$\text{s.t. } \forall\, k \in K, \text{ either } k \in C \text{ or } N(k) \in C$$

Input Format

The first line of your input should contain a single integer, $|K|$, which equals the number of kingdoms. The second line should contain a list of $|K|$ distinct names, separated by spaces. These are the names of your kingdoms. The names must be alphanumeric, can contain up to 20 characters, and can contain only characters A-Z, a-z, and 0-9. The third line must contain the name of the kingdom which will be your starting point.

The next $|K|$ lines should contain an adjacency matrix representation of your graph. In particular, line $3+i$ should contain a space-separated list of $|K|$ numbers/characters. If there is no connection to the ith and jth kingdoms on the map, then the jth entry in the list will be the lowercase 'x'. If the jth entry in the list is a number, c_{ij}, and $i \neq j$, then kingdoms i and j are connected, and c_{ij} if represents the distance between them. For $i = j$, c_{ii} represents the conquering cost of kingdom i. The c_{ij} must be strictly positive integers or floating-point numbers. They must be less than 2 billion, and floats must have less than 5 decimal places. It must be the case that $c_{ij} = c_{ji}$, since the graph is undirected. In other words, your adjacency matrix must be symmetric.

Your graph must be connected. If it is not connected, it is impossible for you to conquer all kingdoms. The edge weights of your graph must also obey the metric property. This means that any triangle in the graph must satisfy the triangle inequality.

Sample input:

```
3
Kanto Johto Hoenn
Johto
1 6 x
6 2 0.5
x 0.5 3
```

Output Format

The output file corresponding to an input must have the same name, except with the extension replaced by ".out". For example, the output file for "1.in" would be named "1.out".

Your output should consist of 2 lines. The first line should be a space-separated list of kingdom names which represents the tour taken in your solution. These names should be in the order in which they are visited in your tour. Vertices may be repeated (because it is a tour). The first and last names should be the starting kingdom specified in the corresponding input file (since you start at that kingdom and return to it)

The second line should be a space-separated list of the kingdoms that you choose to conquer. The order of the kingdoms does not matter in this line. These names must be a subset of the names in the tour on line 1.

Sample output:

```
Johto Kanto Johto
Kanto
```

Note: A common mistake is to save your input/output files as '.in.txt' or '.out.txt' instead of '.in' or '.out'. You can make sure your files are saved in the right format using the following commands:

```
ls -a # Lists the file with their extension.
# If the extension is something different from .in or .out
mv [OLD_NAME_HERE] [name].in
mv [OLD_NAME_HERE] [name].out
```

Timeline

Phase 1 (Due April 13th, 11:59 pm)

Overview

You are allowed to form groups of up to 3 people for the project. This is a hard limit. You must stick with the same group throughout Phase 1 and Phase 2. While you are allowed to do the project alone or with 2 people, we highly encourage you to find a group of 3.

Each group must submit three input files, along with your solution to each of those instances. You will also submit a "design doc" detailing how you plan to approach writing a solver. You must submit exactly 1 input file and 1 output file for each of the following categories:

- At most 50 kingdoms
- At most 100 kingdoms
- At most 200 kingdoms

Your design doc should be at least 200 words, and no more than 500 words long. It should be a high-level description of algorithms/approaches that you have tried or plan to try, as well as your reasoning for what led you to these approaches, and why you think they will work.

We will collect all of the input files and release them to you all after the Phase 1 due date. The difficulty of your input files will determine part of your grade for this phase.

Some starting points for solver ideas include:

- Develop heuristics for some greedy algorithms. How could you solve this problem by choosing greedily?
- Figuring out (Google!) what problems this problem is similar to and reducing this problem to another problem.
- Reading about approximation algorithms for those other problems
- Finding solvers to those other problems (you have access to all freely available libraries!)
- Trying to do a variant of local search on this problem (you can read about it online or in the textbook.)

Feel free to use some of these ideas in your preliminary report, but make sure you actually explain concretely what you intend to do. For instance, if you intend to use some greedy heuristics, come up with some heuristics to test.

Submission

The three input files you submit should be named 50.in, 100.in, and 200.in. The respective output files should be named 50.out, 100.out, and 200.out. If your files do not satisfy these requirements, your input is invalid, and you will not receive any credit for this portion of the project. You will submit the design doc on Gradescope. **Only one member of your team must submit the design doc, and that member must add the other members on Gradescope.**

Phase 2 (Due April 30th, 11:59 pm)

Overview

Design an algorithm and run it on the entire pool of input files. We have released starter code to parse input files in the file solver.py, posted on Piazza and the website (you do not have to use the starter code). You may use any

programming language you wish. However, we must be able to replicate your results by running your code. **You may only use free services** (academic licenses included). This includes things like free AWS credits for students. If you use AWS or any other cloud computing service, you must cite exactly how many hours you used it for in your project report. We should be able to replicate the results that you cite. If you use any non-standard libraries, you need to explicitly cite which you used, and explain why you chose to use those libraries. If you choose to use the instructional machines, **you may only use one at a time**. We will be strict about enforcing this, and will be sending out a Google form for students to self-report people who they see using multiple instructional machines. Anybody caught using multiple instructional machines will receive a zero for this part of the project.

The reason we have to enforce this rule is because CS 170 students always end up overloading the instructional machines in the week before the project is due, and this makes them unavailable to other students. We want to make sure that you are not inconveniencing other students with your project work.

Submission

Put your output files in a zip folder and submit on Gradescope. The output file for an input named "example.in" should be named "example.out". Please make sure you include your teammates in your Gradescope submission.

The auto-grader will score your output files immediately (although it may take a while to run). We will maintain a live leaderboard on Gradescope showing how well your solutions perform compared to the rest of the class. Your team name will be used to display your team's results. **Please keep team names civil and PG13.**

You will also submit your code as well as a **project report** in Phase 2. The report should be a summary of how your algorithm works, what other methods you tried along the way, what computing resources you used (e.g. AWS, instructional machines, etc.). Your report should be at least 400 words, and no more than 1000 words long. We should be able to replicate all your results using the code you submit as well as your project report. More details on how to submit your code and project report will be released closer to the Phase 2 deadline.

We strongly suggest you start early.

Grading

Phase 1 is worth 20% of your project grade. You will get 5% for submitting a design doc that demonstrates thoughtful analysis of the problem. The difficulty of each file itself will determine the other 15% of your phase I grade. For each file, the grading follows as below:

- 5% (full credit): at least 80% of student submissions score less than your solution.
- 4%: 60-80% of submissions score less than your solution.
- 3%: 40-60% of submissions score less than your solution.
- 2%: 20-40% of submissions score less than your solution.
- 1%: 0-20% of submissions score less than your solution.

Phase 2 is worth 80% of your project grade. Your outputs will be graded based on the ratio of your score on that input to the best score of any team on that input. We will also have a staff baseline solution. **The top 5% of teams in Phase 2 will receive extra credit.**

Late submissions will receive a 20% penalty on that phase for each day it is late. The penalty is measured as a percentage of the total score. So if you submit Phase 1 one day late, you lose 20% of the total points you can get for Phase 1. If you submit it 2 days late, you lost 40% of the total points, etc. Lateness is measured by Gradescope submission, and immediately, by the second. For instance, a submission on 12:01:00AM is considered a day late.

Academic Honesty

Here are some rules to keep in mind while doing the project:

1. No sharing of any files (input files or output files), in any form.
2. No sharing of code, in any form.
3. Informing others about available libraries is encouraged in the spirit of the project. You are encouraged to do this openly, on Piazza.
4. Informing others about available research papers that are potentially relevant is encouraged in the spirit of the project. You are encouraged to do this openly, on Piazza.
5. Informing others about possible reductions is fine, but treat it like a homework question – you are not to give any substantial guidance on how to actually think about formulating the reduction to anyone not in your team.
6. As a general rule of thumb: don't discuss things in such detail that after the discussion, the other teams write code that produces effectively the same outputs as you. This should be a general guideline about how deep your discussions should be.
7. If in doubt, ask us. Ignorance is not an excuse for over-collaborating.

If you observe a team cheating, or have any reason to suspect that someone is not playing by the rules, please report it here.

As a final word: remember that this project is going to be curved overall. Overcollaborating is, regardless of academic dishonesty, not a good idea because you don't want to give away key insights that your group has made.