CS 170 Spring 2015 — Algorithms — Christos Papadimitriou & Prasad Raghavendra

Final Project

Problem Description

Remember the traveling salesman path problem (TSP)? You are given a graph with edge weights, and you are asked to find the best *TSP path*, that is, overall cheapest route that visits all cities exactly once. (Unlike a *tour*, a TSP path doesn't have to return to its start city).

In the **non-partisan travelling senator problem (NPTSP)**, each city also has a color, either RED or BLUE, and **the path cannot visit more than three cities of the same color consecutively.** You still want the cheapest such path that visits every city exactly once.

More specifically, you are given a complete, undirected, weighted graph on N cities. The cities are labeled from 1 to N. The graph is given to you in adjacency matrix format, that is you will recieve an $N \times N$ matrix, where the entry in the ith row and jth column represents the edge weight between city i and city j. You are also given a string of length N representing the colors of the cities. The ith character of the string is R if the city is red, and B otherwise. You may also assume that the number of red and blue cities are exactly the same.

Constraints

- The number of vertices will be an even integer between 1 and 50, inclusive.
- The edge weights will be an integer between 0 and 100, inclusive.

Submission instructions

Phase 1 (Input files and coding.) Due 05/01 6pm

Generate **three** hard instances of NPTSP. Make sure to follow the input format exactly, or your file will be discarded. The input format is described in the following section. Don't forget to start coding during this phase.

You will submit your instance files through glookup. In your submission, include your three files, labeled TEAMNAME1.in, TEAMNAME2.in, TEAMNAME3.in. Also, include a README file where the first line is your team name, and the following up to 4 lines are the student ids of people in your group.

Phase 2 (Output files + Source code) Due 05/08 6pm

On May 4th, around noon, we will release the input files generated by each group that are valid. We will post this as a zip file on piazza. After we release all the input files you will run your code, and submit your output file and your code. Make sure you only have a single output file (see output format specification for more details). The input files will be named "i.in", where i is the number of the instance. You can expect around 400-600 input files. See the output specifications for more details on how to format your output file.

In your submission, include your output file, as well a README file where the first line is your team name, and the following up to 4 lines are the student ids of people in your group. This README should be exactly the same as the one you submitted in the first phase.

Miscellaneous Rules and Reminders

- You may work in teams of at most 4.
- You may use any language and/or resources (including supercomputers and cloud computing, if you are so inclined). Your job is to find good solutions to the hard examples of the other groups, and to fool through your examples the code of other groups.
- You may use any publicly available libraries and research papers, but please make sure to cite your sources if you decide to copy something or use a published method.
- If your input file does not conform to the specifications, you will not get any credit for it. Make sure you carefully read the next sections for more details.
- Similarly, if your output does not conform to the specifications, you will not get credit for that instance.
- It is allowed, even encouraged, to hard code answers to the particular input files in your solutions (just make sure other teams won't be able to find your good solution!).
- Starter code for reading and writing to files has been included at the end of this pdf.
- The staff is unable to give very involved technical support with specifics on implementation. Choose a language that you are comfortable debugging in. We will help, however, with running through general approaches and strategies if you'd like.

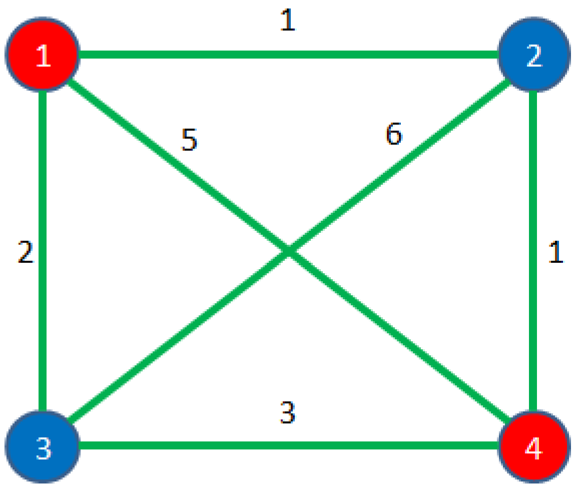
Input format

There will be multiple files. This format describes the contents of one file.
Line 1: One integer N, representing the number of cities
Line $2\ldots N+1$: Line $i+1$ will contain N space separated integers, between 0 and 100. The jth number on this line represents the distance from city i to city j. Since this is a symmetric distance matrix, the ith number on this line should be zero, and the jth number on this line should be equal to the ith number on the $j+1$th line.
Line $N+2$: A string of length N, where each character is either R or B. This string will contain an equal number of Rs and Bs.
We will check inputs and discard invalid ones for no credit. You may assume all input files will follow this format exactly, so there's no need to check these conditions explicitly

Example input

```
4
0 1 2 5
1 0 6 1
2 6 0 3
5 1 3 0
RBBR
```

Here, we have a graph on 4 vertices. It looks like this:



Output format

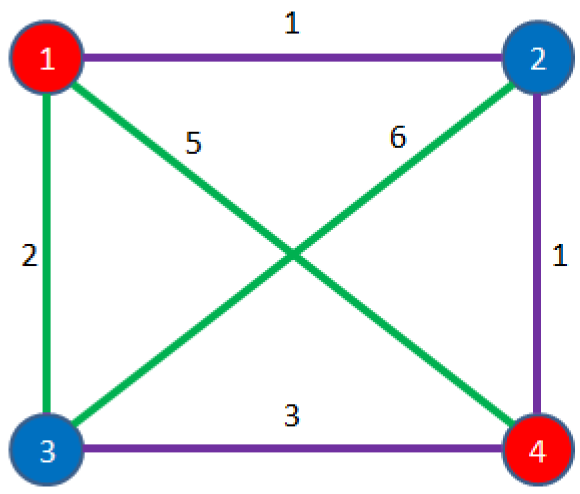
You will submit **one** output file. Each line represents the solution for an individual input file.
Line i: This line corresponds to the answer to the ith input file. You should have N space-separated integers on this line. The integers on this line should be a permutation of $\{1,\ldots,N\}$. The ith integer on the line represents the ith city you will visit on your tour.

Example Output

(Note that this file would have more lines if we had more input files)

```
3 4 2 1
```

Here, the purple edges represent our tour.



We can see that the above tour never visits more than 3 of the same color consecutively. The cost of the tour is the sum of all edge weights, which is 3+1+1=5.

Scoring details

We will score only your output files automatically. We will not automatically run your source code, but we may have a look at it. Your score will reflect two things: (1) how well you did on the examples submitted by others, and (2) how badly the other teams did on your bad examples.

Prizes

The first place team will get treated to dinner with the staff.

Starter code

We've attached some code in popular languages to get you started if you need help to read in and read out to files (only Python, Java, and C/C++). Remember, you can do this in any language you like, and you are free to ignore this code if you wish.

Validators and Scorers

We have also attached some code that will validate your input files, as well as score your output files against a single input. Please make sure to run this validator as a sanity check against your inputs. You may assume that if the validator completes successfully locally, there will not be any problems when you submit your instances.

Helpful starting tips

These are optional, but will help you think about various approaches:

1. How would you solve the unrestricted TSP? You may want to look up some of the basic techniques for doing this (there are too many things to look up, so be very selective. The textbook contains a brief discussion in Chapter 9).
2. What makes the NPTSP different/harder? What type of inputs do you think your algorithm will struggle on?
3. Up to what input sizes that can you try to solve exactly? how?