

Homework 8: P/NP, quarter wrap-up

Version 2: We corrected some information in the due dates section that incorrectly reflected a prior quarter.

Due Date: This homework is due **Friday** March 8th. You will also be able to resubmit two problems with every homework; the resubmission for this week is due on **Monday** March 11th.

Collaboration: You are allowed (and encouraged!) to discuss these problems at a high level with others. But, please read the [full collaboration policy](#) to ensure you are keeping your discussions at the right level, and remember to cite any collaborators.

Problems to Submit: We will count your 2 best mechanical question and your 5 best long-form questions. We strongly recommend you skim all the problems in a section before attempting them. While we try to make problems in a given section of approximately equal difficulty, you may find some to be easier than others.

Note that we set the grade guarantees expecting you to have 1 mechanical and 3 long-form problems per assignment, so submitting only that would let you “keep pace,” but as this is the last assignment, we wanted to give you a chance to catch up from earlier in the quarter.

Directions: Unless otherwise noted, you are allowed to use algorithms described in class (or prerequisite courses) as though you had library implementations of them. For example, you can say “run the BFS-based 2-coloring algorithm from class on the graph G ” or “run the bipartite checking algorithm from lecture 5 on G .” We also have a list of data structures and algorithms you can use from 373 [here](#).

Since this is a course about efficient algorithms, algorithms that are slower than necessary are unlikely to get “E” scores, but (unless otherwise noted) we do not care about constant factors. In general, an algorithm that is correct but (mildly) slower than optimal will get an equal or higher score to an algorithm that is fundamentally incorrect, but fast.

Mechanical Problems

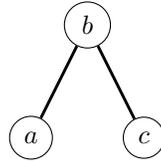
1. True or False

For each of the following, you should answer either “known to be true”, “known to be false”, or “unknown” (“unknown” will be correct if and only if answering the question would resolve P vs. NP). Additionally, add 1-2 sentences of explanation. We’ll give ‘E’ for getting 4/5 (with explanation) and ‘S’ for 3/5 (with explanation).

- (a) All other problems in NP can be solved in quadratic time if there’s some problem in NP that has a quadratic time solution.
- (b) NP-problems are a subset of NP-complete problems (that is, if A is NP, then A is also NP-complete).
- (c) If we take a problem B we already know is NP-hard, and reduce from B to A , then we show A is NP-hard.
- (d) NP problems are a subset of P problems (that is, if A is in NP, then A is also in P).
- (e) We know that the shortest path problem can be solved with Dijkstra’s algorithm. The shortest path problem is in NP.

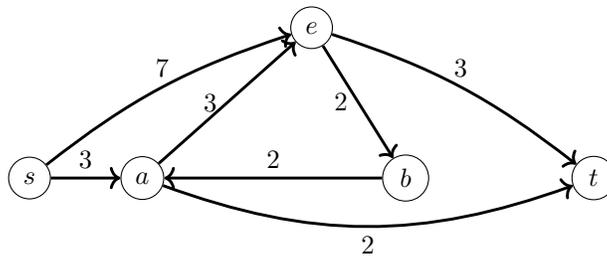
2. Mechanical Reduction

Given the graph below, transform this 3-coloring instance to a 3-SAT instance. Your solution should be a list of constraints on each 3-SAT variable. Make sure you include both edge constraints and consistency constraints. Refer to slide 40-48 in the [reduction lecture](#) for how to get started.



3. Linear Flow-Programming

You are given the following graph where s is the source vertex, t is the sink vertex, and each edge is labeled with its capacity.



- (a) Formulate a linear program for finding max flow on *this particular graph*. For this problem, we want you to give it in as much explicit detail as possible, so **do not use sigma/summation notation**. Describe what decision variables you will use, what your objective function is, and all constraints that you employ. Briefly explain the purpose of each. (Refer to [Lecture 18, Slide 56](#) for how to get started.)

Long-Form Problems

4. Written: Reduction I

On Homework 6, you had a problem of assigning labels 0, 1, 2 to employees at a company based on their direct reporting (tree) structure. (Make sure to remember this is an **undirected** graph)

Your algorithm you wrote worked so well, other companies are asking you how you did it and want to buy your code. There's only one catch, other companies have more complicated (not necessarily tree) reporting structures. Some people might have more than one supervisor, for example. But that should be an easy fix — you can handle that right?

The problem is now the following:

Input: A number n of employees (they will be numbered $0, \dots, n - 1$).

A list of pairs (i, j) to indicate employee i and j cannot have their reports due on the same week (you could get **any** list of pairs, not just a list representing a tree).

Output: The minimum number of 0 labels that are required in a labeling that has no listed pairs with the same label, or ∞ if it's not possible to label.

For example, on input

$n = 3$

$(0, 1), (1, 2), (0, 2)$

The correct output is 1

on input

$n = 4$

$(0, 1), (0, 2), (0, 3), (1, 2), (1, 3), (2, 3)$

The correct output is ∞ .

- (a) Give a reduction that shows the problem is now NP-hard.
Remember you reduce FROM the known NP-complete problem (this one always feels backward to Robbie, be careful!)

- (b) Explain why your reduction is correct. Remember to handle both directions!
We have 1-2 sentences per direction.

- (c) Did we prove $P=NP$ on homework 6? Explain why or why not. (Our explanation is 2 sentences)

5. Written: Reduction II

A **Burr Path**¹ is a path that visits at least $1/2$ of the vertices in a graph.

The **Burr Path Problem** is the following:

Input: A directed, unweighted graph on n vertices

Output:

True if there is a path in the graph that visits at least $\lceil n/2 \rceil$ of the vertices

False otherwise

For example:

- If $V = \{a, b, c, d, e, f, g\}$, $E = \{(a, b), (c, d), (c, e), (f, g)\}$, then return False (no path uses $\lceil 7/2 \rceil = 4$ different vertices)
- On a 9 vertex cycle, the answer is True, any five consecutive vertices (and the edges between them) form a Burr Path.

Burr Path is NP-complete.

Someone suggests the following reduction from Hamiltonian Path:

“Don’t change the graph at all, just take the output of the hypothetical Burr Path solver. If there is a Burr Path, there must be a Hamiltonian Path too.”

The reduction is not valid.

- (a) Give a graph on which the reduction fails, and explain why it is incorrect (our explanation is 2 sentences).
- (b) Give a correct reduction from Hamiltonian Path to Burr Path.
- (c) Explain why your reduction is correct. Remember to handle both directions! Our explanations are 2-3 sentences each.

¹Like a Hamiltonian Path, but not as good. Hamiltonian Paths aren’t named after Alexander Hamilton, but this is the best name we could think of.

6. Written: Approximation

For an undirected graph $G = (V, E)$, a **matching** is a subset E' of E such that no two edges in E' share an endpoint. Consider the following LP for solving the maximum matching problem, where each edge e in E has a corresponding decision variable x_e :

$$\begin{aligned} \text{maximize:} & \quad \sum_{e: \text{edge in } E} x_e \\ \text{subject to:} & \quad \sum_{e: u \text{ is an endpoint of } e} x_e \leq 1 && \text{(for each vertex } u \text{ in } V) \\ & \quad 0 \leq x_e \leq 1 && \text{(for each edge } e \text{ in } E) \end{aligned}$$

Your output might be fractional, but you can assume that every variable is set to 0, 1, or $1/2$ (as opposed to e.g. $1/3$)². We'll round to an integer matching, while ensuring that if the LP finds a (fractional) matching of value k (that is, the objective value is k), you find a (true) matching of at least $2k/3$ edges.

- (a) Describe how to produce an integer matching. (Your algorithm should run in polynomial time, but do not worry about getting the best possible efficiency; we won't deduct for slow polynomial time algorithms.)

Hint: consider the subgraph $H = (V, F)$, where F consists of only the edges e in E for which $x_e = 1/2$ initially.

- Is there an upper bound on the degree of all vertices in H ?
- Each of the connected components in H are one of two possible kinds of graphs. (What are the 2 kinds?)

- (b) Argue that your algorithm produces a “real” matching, i.e. that the subset of edges E' that you've proposed indeed satisfies the definition of matching.

Hint 1: it holds that any two edges in (V, E') don't share an endpoint if and only if every vertex has degree 1 or fewer (with respect to E').

Hint 2: you might find it helpful to break into cases based on how many edges in H are touching a given vertex.

- (c) Argue that your algorithm is polynomial time. (1–2 sentences)

- (d) Let x_e denote the values of the variables set by the LP originally, and let x'_e denote the values after performing the procedure in part (a). Argue that your true matching has at least $2k/3$ edges, where $k = \sum_e x_e$.

Hint 1: Break up both sums along the connected components of (V, F) , then show that each term of $\sum_e x'_e$ is greater-than-or-equal to $2/3$ of the corresponding term in $\sum_e x_e$. Think something along the lines of:

$$\begin{aligned} \sum_{e \text{ in } E} x_e &= \sum_{e \text{ not in } F} x_e + \sum_{\substack{P: P \text{ is 1st kind} \\ \text{of connected comp.}}} \sum_{e \text{ in } P} x_e + \sum_{\substack{C: C \text{ is 2st kind} \\ \text{of connected comp.}}} \sum_{e \text{ in } C} x_e \\ & \quad \sum_{e \text{ in } C} x'_e \geq \frac{2}{3} \sum_{e \text{ in } C} x_e && \text{(want to show)} \end{aligned}$$

Hint 2: $\frac{n/2}{\lfloor n/2 \rfloor} \leq \frac{3}{2}$ for every positive integer n ; what might n be, and what is the significance of $n/2$ and $\lfloor n/2 \rfloor$?

- (e) Give an example of an input graph G where a maximum value of the LP really is more than the value of any matching.

Hint: your graph must not be bipartite for this to be possible.

²For this particular type of LP, there are some LP solvers which can do that automatically. You don't need to worry about why (look up “basic feasible solution” if you're really curious); for designing the algorithm you just need to know that this is possible

7. Written: Review Subarray DP

Recall that a **contiguous subarray** is all of the elements in an array between indices i and j , inclusive (and if $j < i$ we define it to be the empty array).

Call a subarray **nearly contiguous** if it is contiguous (i.e. contains all elements between indices i and j for some i, j) or if it contains all but one of the elements between i and j .

For example, in the array $[0, 1, 2, 3, 4]$,

- $[0, 2, 3]$ is nearly contiguous (from 0 to 3, skipping 1), sum is 5
- $[0, 1, 2, 3]$ is nearly contiguous (because it is contiguous from 0 to 3), sum is 6.
- $[2, 4]$ is nearly contiguous (from 2 to 4, skipping 3), sum is 6.
- $[3]$ is nearly contiguous (because it is contiguous from 3 to 3), sum is 3.
- $[\]$ is nearly contiguous (because it is contiguous from 1 to 0), sum is 0.
- $[0, 2, 4]$ is **not** nearly contiguous (because you'd have to remove two elements).

The sum of a nearly contiguous subarray is the sum of the included elements.

Given `int[] A`, your task is to return the maximum sum of a nearly contiguous subarray.

For example, on input $[10, 9, -3, 4, -100, -20, 15, -5, 9]$ your algorithm should return 24 (corresponding to $i = 6, j = 8$ and skipping the -5).

- Define one or more recurrences to solve this problem.
- Give English descriptions (1-2 sentences each should suffice) for what your recurrences calculate. Be sure to mention what any parameter(s) mean.
- How do you calculate your final overall answer (e.g. what parameters input to which of your recurrences do you check).
- What memoization structure(s) would you use?
- What would the running time of your algorithm be (you do **not** have to write the code). Justify in 1-3 sentences.

8. Written: Plotting

One of your TAs has become an architect for King County and has designed a new infrastructure development plan for Seattle. They need your help to write an algorithm: you are given a $k \times k$ grid representing plots of land and k infrastructure projects labeled from 1 to k . The challenge is to allocate the projects to the plots of land without violating any of the conditions, which are given in the form of two 2D integer arrays `rowConds` and `colConds`:

- `rowConds[i] = [a, b]` says that the project labeled with number a must be in a row that is strictly above the row at which the project with label b appears.
- Similarly, `colConds[i] = [c, d]` says that the project labeled with number c must be in a column that is strictly to the left of the column at which the project with label d is placed.

Example 1:

Sample Input:

$k = 3$ (you are given 3 by 3 plots of land and your projects are labeled 1, 2, and 3)

`rowConds = [[1, 2], [3, 2]]`

`colConds = [[2, 1], [3, 2]]`

Correct Output Grid: `[[3, 0, 0], [0, 0, 1], [0, 2, 0]]`

3		
		1
	2	

Explanation: Notice that in the output, project 3 is in row 0, project 1 is in row 1, and project 2 is in row 2, so this placement satisfies all the row conditions: project 1 should be placed above project 2 in the grid and project 3 should be placed above project 2 in the grid.

Now let's look at column conditions. In our output grid project 3 is in column 0, project 2 is in column 1, and project 1 is in column 2, therefore it also satisfies all the column conditions: project 2 is placed to the left of 1, project 3 is placed to the left of project 2. Similarly, `[[0, 0, 1], [3, 0, 0], [0, 2, 0]]` here is also a valid output

Example 2:

Sample Input:

$k = 4$

`rowConds = [[4, 3], [1, 4], [1, 2]]`

`colConds = [[1, 4], [2, 1], [2, 4], [3, 1], [3, 2], [3, 4]]`

Correct Output Grid: `[[0, 0, 1, 0], [0, 2, 0, 0], [0, 0, 0, 4], [3, 0, 0, 0]]`

		1	
	2		
			4
3			

Example 3:

Sample Input:

$k = 3$

`rowConds = [[1, 2], [2, 3], [3, 1]]`

`colConds = [[3, 2]]`

Correct Output : []

Explanation: There is no placement that can satisfy all of the row conditions. The first and second row conditions contradicts the third row condition. If project 2 is below project 1 and project 3 is below project 2 then project 3 cannot be above project 1.

- (a) Describe the graph(s) you want to use for this problem. Explain what your nodes and edges are going to be for each graph. (*Hint: we used 2 graphs; one for rows, and the other for columns*)
- (b) Using the graph(s) you created, what kind of property do we need your graph(s) to have in order to have a valid placement that can satisfy every row and column condition?
- (c) Design an algorithm to output one valid placement given the input. Write pseudocode (or English descriptions) for your algorithm.
- (d) What is the running time of your algorithm?

9. Programming: Make A Verifier

In this problem, you'll write a verifier for 3-SAT. (Take a look at [Lecture 22, Slide 11](#) if you need a reminder of what 3-SAT is.)

Specifically, you'll be given the following inputs:

- A boolean array X of length n , where $X[i] = x_{i+1}$ for $0 \leq i < n$
(This array encodes a proposed setting of the variables.)
- A 2d boolean array Y with m rows and 3 columns, where $Y[i][j] = y_{i+1, j+1}$.
- A 2d int array Z with m rows and 3 columns, where $1 \leq Z[i][j] = z_{i+1, j+1} \leq n$ for each element $Z[i][j]$.
(Together, Y and Z encode the constraints.)

You'll be given some proposed setting of the Boolean variables x_1, x_2, \dots, x_n , and your task is to verify whether this setting of the variables will satisfy all m of the constraints, where each constraint is of the form

$$x_{z_{i,1}} == y_{i,1} \ || \ x_{z_{i,2}} == y_{i,2} \ || \ x_{z_{i,3}} == y_{i,3}$$

for $1 \leq i \leq m$. In the constraints of the form above, each $1 \leq z_{i,j} \leq n$ determines the subscript for a boolean variable. ($1 \leq j \leq 3$)

Your task is to write an efficient verifier for the problem. In other words, you should return `True` if all constraints are satisfied with the variable setting given `False` otherwise. Note that you're only writing a verifier, not a solver. 3-SAT is NP-complete, so a solver would be very inefficient. We just want an efficient verifier (is the proposed variable setting good or not?).

In order to help you translate between the abstract problem description and the code, the following Java-style pseudocode gives an example of a concrete instance of the problem for $n = 4$ and $m = 5$:

```
final boolean T = true;
final boolean F = false;

boolean[] X = {T, F, T, T};

boolean[][] Y = {{F, T, F},
                 {F, T, T},
                 {T, T, T},
                 {T, F, T},
                 {F, T, F}};

int[][] Z = {{1, 2, 3},
             {1, 2, 3},
             {3, 1, 4},
             {4, 1, 3},
             {4, 3, 2}};

/* We want to check whether the constraint

(x_1 == F || x_2 == T || x_3 = F) &&
(x_1 == F || x_2 == T || x_3 = T) &&
(x_3 == T || x_1 == T || x_4 = T) &&
(x_4 == T || x_1 == F || x_3 = T) &&
(x_4 == F || x_3 == T || x_2 = F)

is satisfied, given that the proposed
setting of the boolean variables is:

x_1 = T, x_2 = F, x_3 = T, x_4 = T */
```