CS41 Homework 9

This homework is due at 11:59PM on Monday, November 8. Write your solution using LaTeX. Submit this homework using **github** as **.tex** file; the code should be in a file called **pretty-print.py**. This is a **partnered homework**. You should primarily be discussing problems with your homework partner.

It's ok to discuss approaches at a high level with others. However, you should not reveal specific details of a solution, nor should you show your written solution to anyone else. The only exception to this rule is work you've done with a lab teammate *while in lab*. In this case, note (in your **homework submission poll**) who you've worked with and what parts were solved during lab.

1. **Pretty-printing** (based on KT 6.6) Suppose we have a paragraph of text, and we want to print it neatly on a page. The paragraph consists of a list of words $w_1, w_2, \ldots, w_n$; each word w_i has length ℓ_i. The maximum line length is M. (Assume that $\ell_i \leq M$ for all i.) We assume we have a fixed-width font and ignore issues of punctuation and hyphenation.

   Consider a line containing words $w_i, w_{i+1}, \ldots, w_j$, and using only one space between words. Because the words must fit within the maximum line length, we know that:

   $$\text{length of this line } = (\ell_i + 1) + (\ell_{i+1} + 1) + \cdots + (\ell_{j-1} + 1) + \ell_j \leq M$$

   The "slack" space on a line is the number of spaces remaining at the right margin, so for this line it is the value:

   $$\text{slack of this line} = M - \Big((\ell_i + 1) + (\ell_{i+1} + 1) + \cdots + (\ell_{j-1} + 1) + \ell_j\Big)$$

   The penalty is the sum over all the lines (including the last) of the *squares* of the slack of all lines in the paragraph.

   (a) Describe and analyze a dynamic programming algorithm to find the best way to print a paragraph, where "best" means "with smallest penalty". Include a recursive definition of the optimal value that motivates your algorithm.

   (b) Implement your algorithm in the file `pretty-print.py`; it should print an optimal division of words into lines. The input should be a number (M, the maximum line length) and a file containing some words; you should assume that a "word" is any contiguous sequence of characters not including whitespace. The program should print the paragraph, split into lines appropriately, followed by the numerical value of the penalty.

   Note: You should feel free to create your own input files and test your program on a variety of different inputs. Some example inputs and outputs (for comparison/testing) can be found in `/home/fontes/public/cs41/text-wrapping`. Your program should be able to handle quite large inputs — e.g. one example is the full text of Herman Mellville's Moby Dick, with $M = 40$ (note that this might take a few minutes to run, but should not crash).

For example, consider the input in `demoText`.[1]
With maximum line length 25, we can call the program by running
```
> python pretty-print.py 25 demoText
```
and get the output:

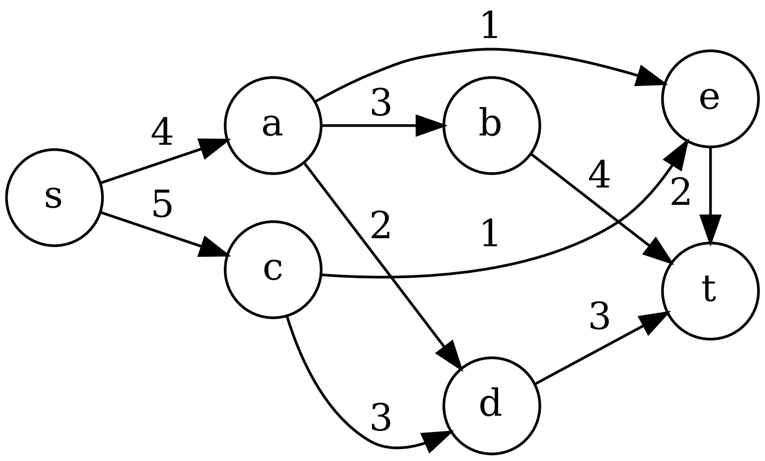
```
Not far from here, by
a white sun, behind a
green star, lived the
Steelypips, illustrious,
industrious, and they
hadn’t a care: no spats
in their vats, no rules,
no schools, no gloom,
no evil influence of the
moon, no trouble from
matter or antimatter-for
they had a machine, a
dream of a machine, with
springs and gears and
perfect in every respect.
Penalty: 137
```

With maximum line length 75, the output should look like:

```
Not far from here, by a white sun, behind a green star, lived the
Steelypips, illustrious, industrious, and they hadn’t a care: no spats
in their vats, no rules, no schools, no gloom, no evil influence of the
moon, no trouble from matter or antimatter-for they had a machine, a
dream of a machine, with springs and gears and perfect in every respect.
Penalty: 199
```

[1] This is a line from Stanislaw Lem’s *The Cyberiad.*

2. **Maximum flow.** Find the maximum flow from s to t and the minimum cut between s and t in the network below. Show the residual network at intermediate steps as you build the flow. (You can include your intermediate residual flow as figures/images, or you can explicitly write out all of the residual graph (all edges and all capacities) at each step.)



3. **Changing edge capacities.** Consider the following claim:

   Claim 1. *Let G be an arbitrary flow network, with a source s, sink t, and a positive integer capacity c_e on every edge e. Let (A, B) be a minimum $s - t$ cut with respect to these edge capacities $\{c_e : e \in E\}$. Now, suppose we add 1 to every edge capacity. Then, (A, B) is still a minimum $s - t$ cut with respect to these new capacities $\{1 + c_e : e \in E\}$.*

   Answer whether you think the claim is TRUE or FALSE. If you answer TRUE, give a short explanation why it is true. If you answer FALSE, give a counterexample showing the claim is false.

4. **(extra challenge) Flow variants.** In the standard flow problem, we get an input $G = (V, E)$ a directed graph and edge capacities $c_e \geq 0$ limiting how much flow can pass along an edge. Consider the following two variants of the maximum flow problem.

   (a) It might be that each junction where water pipes meet is limited in how much water it can handle (no matter how much the pipes can carry). In this case, we want to add *vertex* capacities to our problem. The input is a directed G (with source s and sink $t \in V$), edge capacities $c_e \geq 0$, and vertex capacities $c_v \geq 0$ describing the upper limit of flow which can pass through that vertex. Give a polynomial-time algorithm to find the maximum $s \rightsquigarrow t$ flow in a network with both edge and vertex capacities. (Hint: reduce this problem to the standard network flow problem, then use Ford-Fulkerson to solve it.)

(b) It might be that there are multiple sources and multiple sinks in our flow network. In this case, the input is a directed G, a list of sources $\{s_1, \ldots, s_x\} \subset V$, a list of sinks $\{t_1, \ldots, t_y\} \subset V$, and edge capacities $c_e \geq 0$.

Give a polynomial-time algorithm to find the maximum flow in a network with multiple sources and multiple sinks. (Hint: reduce this problem to the standard network flow problem, then use Ford-Fulkerson to solve it.)