**CS3230: Design and Analysis of Algorithms (Spring 2020)**

**Homework Set #3**

**OUT: 30-MAR-2020** **DUE: 24-APR-2020 (6pm)**

**IMPORTANT: Write your NAME, Matric No, Tut. Gp in your Answer Sheet.**

- Start each of the problem on a *separate* page.
- Make sure your name and matric number is on each sheet.
- Write legibly. If we cannot read what you write, we cannot give points. In case you CANNOT write legibly, please type out your answers and print out hard copy.
- To hand the homework in, please submit a .pdf file of your typed/scanned version of your assignment to the submissions box on Luminus. It may be handwritten and scanned but please make sure your handwriting is legible and the copy has sufficiently high resolution.

**IMPORTANT:** You are advised to start on the homework *early*. For some problems, you need to let it incubate in your head before the solution will come to you. Also, start early so that there is time to consult the teaching staff during office hours. Some students might need some pointers regarding writing the proofs, others may need pointers on writing out the algorithm (idea, example, pseudo-code), while others will need to understand more *deeply* the material covered in class before they apply them to solving the homework problems. Use the office hours for these purposes!

It is always a good idea to email the teaching staff before going for office hours or if you need to schedule other timings to meet. Do it in advance.

**HOW TO "Give an Algorithm"**
When asked to "give an algorithm" to solve a problem, your write-up should *NOT* be just the code. (*This will receive very low marks.*) Instead, it should be a short essay. The first paragraph should summarize the problem and what your results are. The body of the essay should consist of the following:

- A description of the algorithm in *English* and, if helpful, pseudo-code.
- One worked *example* or *diagram* to show more precisely how your algorithm works.
- A *proof* (or indication) of the correctness of the algorithm.
- an *analysis* of the running time of the algorithm.

Remember, your goal is to communicate. Full credit will be given only to correct solutions which are described clearly. Convoluted and obtuse descriptions will receive low marks.

In CS3230, you learn to develop high-level abstractions when describing algorithms. Try not to speak in ML/AL (machine/assembly language) or for "`(j=0; j<n; j++) do`". Instead give names to your sets (of objects or things or data structures), talk about Depth-First Search, Binary Search, traverse the graph, sort the set, use a priority queue, etc. You are no longer in CS1010, CS1020, CS2010 or CS2020. Speak with greater sophistication, and at a higher level of abstraction.

(**Note:** Each problem is worth 10 marks.)

# Background Story

Good news everyone! The Swedish robot from $\pi$KEA is here with the supercollider that Professor Farnsworth ordered. He's pretty excited to plug it in and power it up so that he can run new experiments. But, his lab in a mess (he doesn't clean up very often) and basically he need to reroute power from sources to sinks. He's hired Mr. Nodle, a research assistant, to help him out here.

Unfortunately, after a few months of running around the labyrinth that is Professor Farnsworth's lab, it's probably safe to say that Nodle is not going to be able to solve this by himself. While he's busy crying in a corner about how he's misplaced the trust of the Professor, you could really do him a favor by trying to show him that in fact, this problem is in fact, very difficult, so he doesn't have to feel so bad after all[1].

# Problem Statement

We will call the problem you're trying to solve `PLUGIT`. The input is given as an $m \times n$ grid of tiles, where each tile can be one of the following:

1. An empty tile
2. A socket (powered)
3. A socket (unpowered)
4. A broadcast node
5. Wall(s)

(**Note:** Wires are not part of the input.)

Now the goal is to be able to power all sockets (broadcast nodes need not be powered). The section below details the physics of Professor Farnsworth's lab.

**Definition 0.1** *`PLUGIT` Given such a grid with tilings, the problem is to output* ***yes*** *if there exists a routing by using the directed wires (that follows the rules) to power all sockets, and* ***no*** *otherwise.*

**Goal of assignment:** Show that `PLUGIT` is NP-complete.

# Some Details on The Mechanics of the Problem

This section pretty much explains the rules of what's going on, or the mechanics of what is allowed and what isn't (i.e. the physics of Professor Farnsworth's lab). So for the below examples, here are the icons that we will use:

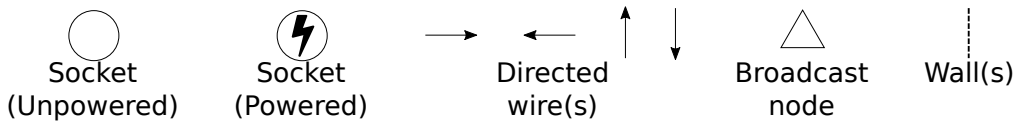


Figure 1: The icons that we will use for the examples on the grid.

**Note:** Wires are not part of the input.

[1] Or, you could choose not to, but then he would have to give you 0 marks for the assignment he's set for your CS3230.

## Mechanic 1: Powering Sockets

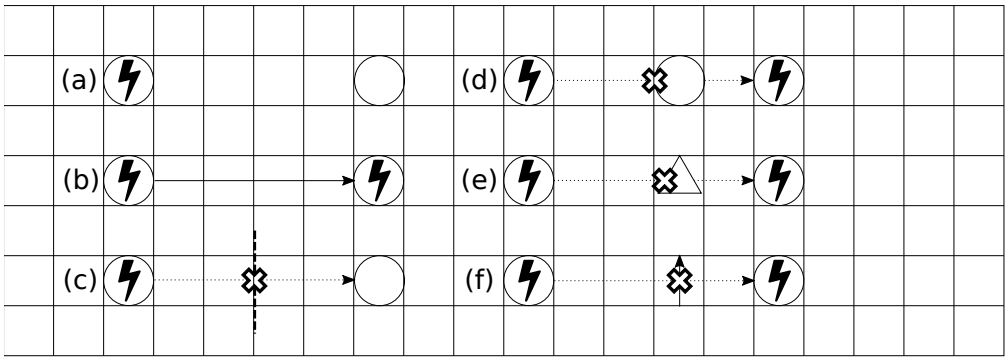


Figure 2: (a) Before directing power; (b) After directing power; (c, d, e, f) An impossible case

You can draw a directed wire from a powered socket into an unpowered one. Each socket can only have at most one incoming wire, and at most one outgoing wire. Note that you can only direct power in one of the four directions: up, down, left, or right. The wires also cannot cross the (c) walls, (d) sockets, (e) broadcast nodes, or (f) other wires.

## Mechanic 2: Redirecting Power

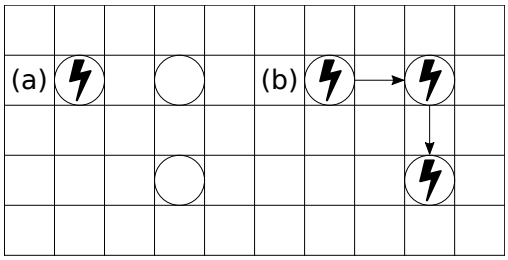


Figure 3: (a) Before redirecting power; (b) After redirecting power

After a node has been powered, you can again direct power to any other node. Again, you can only direct power in one of the four directions: up, down, left, or right.

## Mechanic 3: Broadcast Nodes

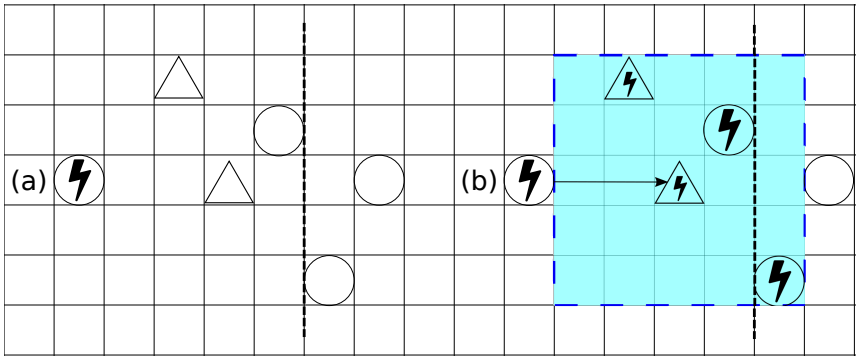


Figure 4: (a) Before powering broadcast node; (b) After powering broadcast node

After a broadcast node has been powered, anything within a $5 \times 5$ grid, with the broadcast node as the centre (highlighted in blue above) will also be powered, even if they are behind walls. In the example above, notice that the rightmost socket remains unpowered because it is outside of the blue region.

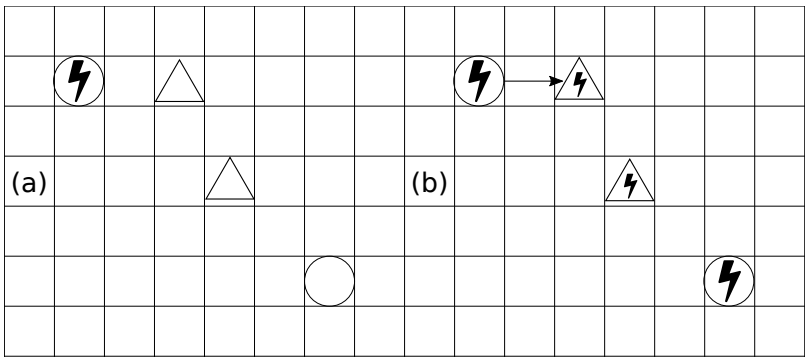


Figure 5: (a) Before powering broadcast node; (b) After powering broadcast node

Note that any other broadcast node that is within the area of effect will also be powered, and they in turn will power anything within their own area of effect, as shown above.

## Mechanic 4: Some More Restrictions

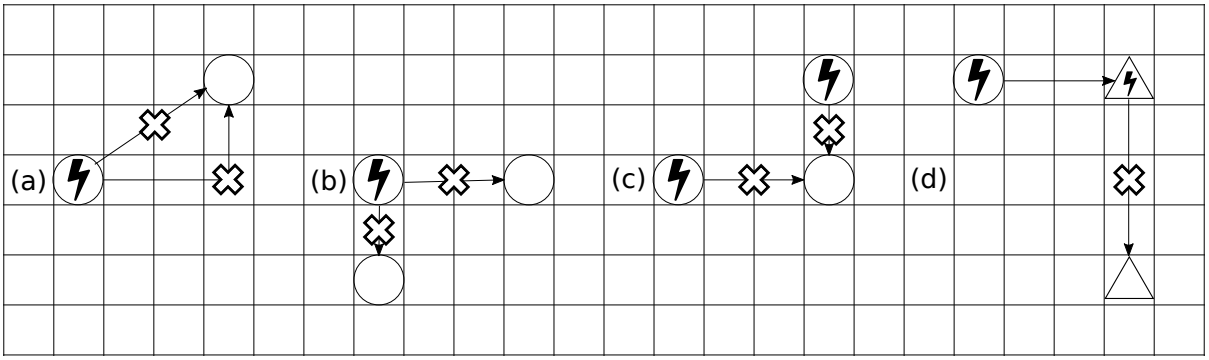


Figure 6: Some more illegal moves

You are not allowed to draw directed wires that are not axis aligned nor are you allowed to bend wires. You also cannot use a single powered socket to power more than 1 unpowered socket, and you cannot use two different powered sockets to power the same socket. Wires cannot cross other wires and and broadcast nodes, and sockets. Lastly, wires cannot come out of broadcast nodes.

## A few examples

Below is an example of a valid instance, where $(a)$ is the input, and $(b)$ shows why it is solvable.

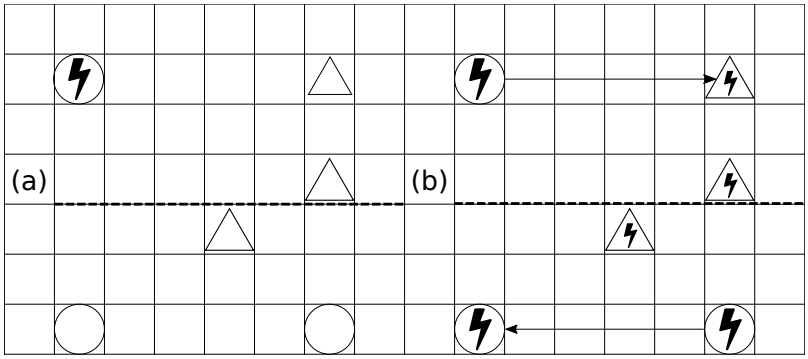


Figure 7: A valid instance of `PLUGIT`

On the other hand, here is an example of an instance that it not solvable.

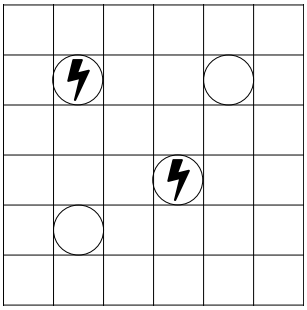

Figure 8: An invalid instance of `PLUGIT`

### Just a side remark

This problem is actually inspired by a phone game called `Transmission`, which is available on both the *Google play store* and the *Apple app store*! Feel free to download it and give it a try. But **as a disclaimer**: Mr Nodle, and Professor Farnsworth are not affiliated with that app in any way. Furthermore, *the rules for the app differ a little compared to the rules/mechanics presented here (there are far more rules there).* **So please follow the rules presented in this document rather than the rules in the app**. (But I will say that you can modify the reduction here to cater to the rules in the app as well, but that would be too much work for one assignment, so if you finish this assignment early, you can think about how to extend this reduction to account for the rules in the phone game instead.)

## Question 1: Proving NP-hardness

Show that `PLUGIT` is NP-hard. You may do this by performing a reduction from any of the NP-Hard problems presented during the lectures or tutorials.

To be precise, the list of problems you may use includes:

1. 3-SAT (If you are new to reductions, we suggest you use this one. This is also what the TAs used.)
2. Travelling Salesman Problem
3. Independent Set
4. Vertex Cover
5. Hamiltonian Cycle
6. Max-Clique

7. Any other problems shown during the tutorials or lectures.

**Note:** The direction of the reductions matter! Make sure you reduce from an NP-hard problem to `PLUGIT`. A full solution constitutes giving a correct reduction which maps (1) all YES instances of the input to YES instances of `PLUGIT` and (2) all NO instances of the input to NO instances of `PLUGIT`. (3) The reduction must also run in polynomial time. Partial marks may be given depending on how these 3 criteria were fulfilled.

## Question 2: Proving in NP

All is not lost, however. It turns out that Mr. Nodle has a good friend Mr. Govond, who just so happens to propose that it is indeed possible to power every socket. However, Mr. Nodle is usually quite skeptical about Mr. Govond's propositions. Mr. Govond should be able to provide a polynomial-sized certificate that it is indeed possible, and Mr. Nodle should be able to verify that certificate, in polynomial time (with respect to the size of the input).

Show that `PLUGIT` is in NP. You can do this by showing that there exists a polynomial time verification algorithm that for every valid instance of `PLUGIT`, along with a polynomially sized certificate format is able to output **yes**, and **no** if the certificate is invalid.

**Note:** A full solution here constitutes of: (1) Specifying a correct certificate format that such that (2) all YES instances are verifiable in polynomial time.

Questions henceforth are unrelated to the previous two.

## Question 3

Say there are $n + m$ friends, as it turns out, the first $n$ people owe money to the other $m$ people, where the set of people who owe and the people who are owed are disjoint. Now, they wish to settle the matter with as few transactions as possible. In other words: Given a set of values $A = [a_1, a_2, \ldots, a_n]$, and $B = [b_1, b_2, \ldots, b_m]$, and an integer $k$, where $A$ is the set of the amount of money that the first $n$ people owe, and $B$ the set of the amount of people the next $m$ people are owed, decide if the first $n$ people can pay at most whatever they owe, the other $m$ people can recieve whatever they are owed, within $k$ transactions. You are guaranteed the total amounts in $A$ and $B$ are the same.

For example, if $A = [8, 10]$ and $B = [5, 5, 8]$ and $k = 3$ the answer is **true**. To see this, note that the first person who owes money, can pay the third person who is owed money. And then the second person who owes money and pay the first and second person who is owed money. This is 3 transactions at most.

Show that this problem is NP-complete. You are allowed to assume the following problems are NP-complete for you to do reductions.

1. Partition (If you are new to reductions, we suggest you use this one. This is also what the TAs used.)
2. 3-SAT
3. Travelling Salesman Problem
4. Independent Set
5. Vertex Cover
6. Hamiltonian Cycle
7. Max-Clique
8. Any other problems shown during the tutorials or lectures.

## Question 4

Consider the optimization version of the problem stated in Question 3, i.e., to find the minimum number of transactions. Design a polynomial time approximation algorithm with approximation ratio at most 2. (Clearly mention the running time of your algorithm and calculate the approximation ratio.)