[Solution Quiz 3] Question 2

Ragav Sachdeva

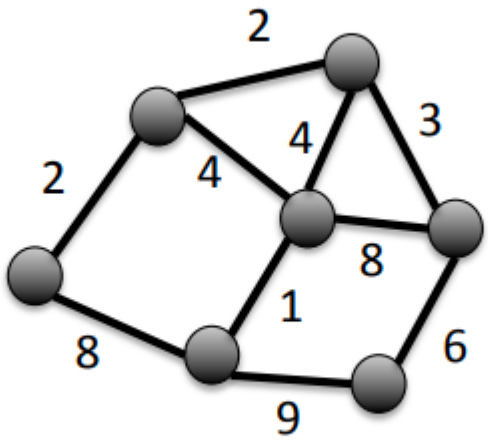
All sections

Hi all,

I marked Q2, Quiz 3 of your submissions and I'd like to provide the solutions and some feedback.

Before I dive into it, let me point out that some of you may have found this quiz to be a bit challenging and it's okay, it was meant to be challenging.

Q2.1 Apply Kruskal's algorithm to the following graph



In your answer show which edges are added and skipped and draw the final spanning tree. (**3 marks**)

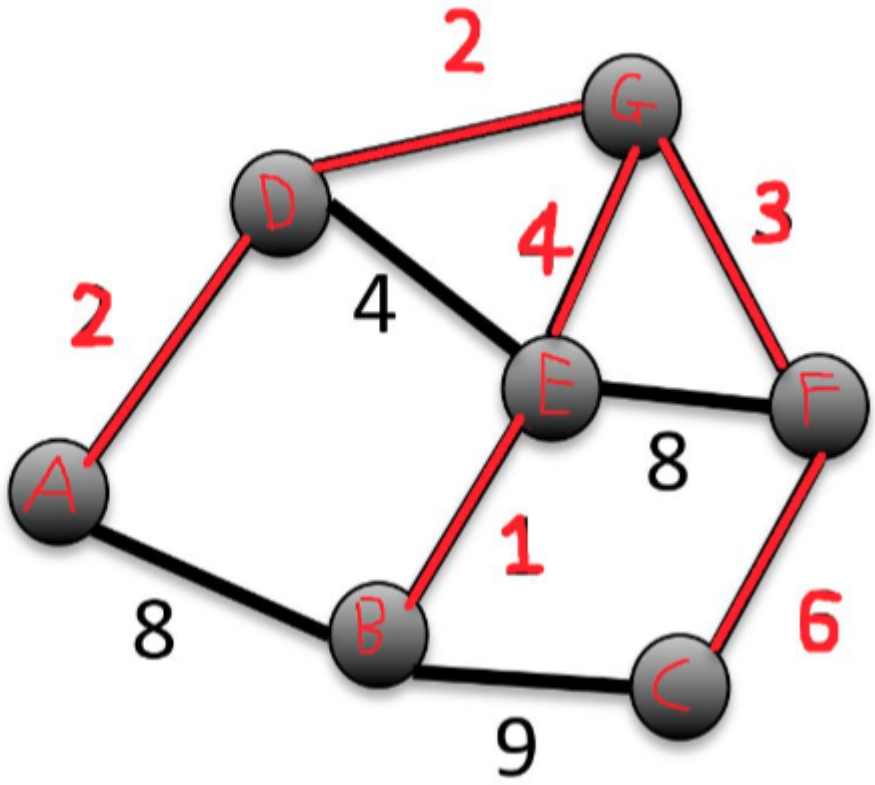
Most student got this correct. Note that the question is worth 3 marks and asked you to do a few things

- apply Kruskal's algorithm (1 mark) - Any evidence of sorting the edges by weight and introducing them 1 by 1 without introducing a cycle will get you the mark
- show which edges are added and skipped (1 mark) - You need to explicitly identify the edges that were added and skipped
- final mst (1 mark) - drawing the correct mst

Solution: (there is another possible solution, you were awarded marks for that too)

Q2.1

Add BE
Add AD
Add DG
Add GF
Add GE
Skip DE
Add FC
Skip EF
Skip AB
Skip BC



Q2.2 Describe the role of the Union–Find data structure in Kruskal's Algorithm.
(1 marks)

A lot of the answers I read basically described what Union-Find data structure is rather than its *role* in Kruskal's algorithm. If you really want to go down the path of explaining what union-find is and describing each of its function you *can* do it but you'd also need

to explain why/how it is required to solve MST problem using Kruskal's algorithm. If you simply say "find operation looks up the set each node belongs to and if they are different, the union part joins the two sets" it doesn't tell me the relevance to Kruskal's (you don't **need** sets and unions etc. for Kruskal's, it is just one implementation of it). You'd therefore need to describe how in Kruskal's the spanning tree starts with no edges (which is equivalent to n disjoint sets) then you need to introduce edges one by one (which is accomplished using union) as long as no cycle is introduced (which you can check using find).

However, the key role/advantage of Union-Find is that fact that it allows for ***efficient* cycle detection** in an un-directed graph (which is directly relevant to the algorithm). It is important to realise that the idea of disjoint sets (that need to be "merged") is not fundamental to Kruskal's algorithm; it is an outcome using Union-Find itself. Furthermore, efficiency is important. You can totally implement Kruskal's using the following approach

- sort all the edges in ascending order
- Initialise an adjacency matrix to represent the spanning tree (initially no edges)
- Add edges one by one if no cycle is introduced (which can be checked by performing a depth-first search based algorithm)

but it is incredibly inefficient to use as DFS will take linear time to check for cycle (every single time).

Q2.3 Briefly describe how the *find* operation is implemented in the Union–Find data structure used in Kruskal's algorithm. **(1 marks)**

find does **not** "check if two vertices are in the same subset". We actually ***use*** find() to accomplish that. The job of *find* is merely to tell is which set a particular node belongs to. However, you need to provide/describe the implementation (and not just tell me what *find* **does**).

The implementation I was looking for was basically something that describes the **implementation of find function on this website (https://www.geeksforgeeks.org/union-find/)** in words (or something along the lines of). Code/pseudo code were also accepted.

A lot of students described it in a very theoretical sense where you have an array R that stores the "set name" for each node and you essentially just return it. While this is true, in a practical sense it would imply that every time you do a union of two sets, you'd need to iterate over all the elements in the smaller set and update their "set name" in the R array (which is incredibly inefficient). In practice you want to do something like in the link I attached above (also see **this (http://www.geeksforgeeks.org/union-find-algorithm-set-2-union-by-rank/)** - which is an optimisation on top).

Having said that, I am still pretty torn about the "expected" answer. I gave full/partial marks where I deemed fit but if you think you have been unfairly marked for this question, email me with a *justification* and I'll happily re-consider.

Q2.4 What does it mean to say that a problem is NP? Give an example of a problem that is NP. **(2 marks)**

Most people got this correct. I accepted the formal mathematical definition or even general hand wavy idea of what an NP problem is. Basically, as long as you had something along the lines of:

- NP is the set of decision problems for which the problem instances, where the answer is "yes", have proofs verifiable in polynomial time by a **deterministic Turing machine; (https://en.wikipedia.org/wiki/Deterministic_Turing_machine)** or
- The set of decision problems *solvable* in polynomial time by a **non-deterministic Turing machine (https://en.wikipedia.org/wiki/Non-deterministic_Turing_machine)**

Even if you didn't mention Turing machine and loosely said "the solution can be verified in polynomial time" you got the mark.

A common mistake I noticed was that a lot of students confused NP with Non-Polynomial and said "no polynomial time solution is known but given a solution we can check it in polynomial time". **You should know that every problem in P is also in NP** i.e. for a problem to be in NP we **may or may not** have a polynomial time solution. The only rule the definition proposes is on verifying the candidate solution.

Definition got you 1 mark. And giving an example got you another mark.

A lot of students said TSP is in NP. This is in fact not true. Think about it.. if I give you a tour for a TSP, would you be able to guarantee me in Polynomial time that it is in fact the shortest? No, you'd need to compare it with every other possible tour to make sure it is indeed the shortest. Having said that the decision-version of the TSP problem is in indeed in NP, where you don't find the shortest solution but a solution where the total distance is less than k. In this case, if I give you a candidate solution, you can quickly add up all the numbers and tell me if it is less than K and hence valid or not. I gave everyone the benefit of the doubt and gave you the full mark even if you only said TSP (assuming you meant the decision version). However, if you described the TSP problem in the solution and use the "shortest" version, that was incorrect.

Q2.5 Briefly describe how a problem can be shown to be NP-Hard. **(2 marks)**

Wikipedia defines NP-Hard problems as:

"A problem H is NP-hard when every problem L in NP can be reduced in polynomial time to H; that is, **assuming** a solution for H takes 1 unit time, H's **solution** can be used to solve L in polynomial time"

However things can get a bit hairy depending on your choice of words e.g. if you state H as a decision problem, then you should use "polynomial-time many-one reduction" instead of just "polynomial time reduction", on the other hand if you are defining NP-Hard in terms of NP-complete problems, then it doesn't restrict them to decision problems etc. etc.

The good thing is that the question didn't ask for a definition. Phew. Just how you could **show/prove** a given problem, say Q, to be NP-Hard.

Some of the answers I got were on the right track where the students said:
1. Find a known problem, B, that is in NP-Hard or NP-complete
2. Show problem B can be used to solve the original problem Q in polynomial time (if yes, then Q is NP-Hard).

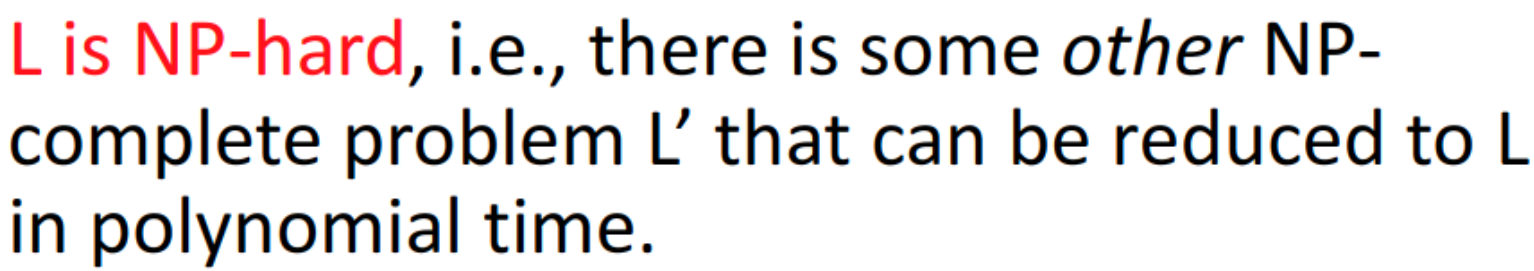
What I don't understand with this answer is "using" *problem* B to *solve* problem A. Using a problem itself to solve another problem has very little meaning (as far as I know), you should either use

- problem to problem parallel: Map your *problem* Q to a known NP-Hard *problem* B (where the mapping is a polynomial reduction function)
- solution to solution parallel: Draw a relationship between B's *solution* and Q's *solution*

and not

- problem to solution parallel: where you suggest Q's *solution* can be found using *problem* B (and not B's solution) in polynomial time (this isn't wrong per say, just a bit incomplete)

Anyway, the answer I was looking for is along of the lines of this (screenshot taken from the lecture slides, replace L with Q and L' with B):



Here is a forum (https://cs.stackexchange.com/questions/19228/np-hard-proof-polynomial-time-reduction) that explains Proof of NP-Hardness in much more detail than I do.

Q2.6 Is the minimum spanning tree problem in P? Please explain your answer.
(1 marks)

Practically everyone got this correct. The answer is yes. We have known algorithms (e.g. Kruskal's) that are bounded by polynomial functions and therefore, MST problem is in P.

If you have any concerns regarding your mark for Question 2, please reach out to me directly at a1720657@student.adelaide.edu.au (mailto:a1720657@student.adelaide.edu.au). For your email subject use "[NP-Hard is hard] (subject of the email...)" so that I know you've read this announcement in its entirety (otherwise, my [automatic] reply to you will be to read the announcement).

Search entries or author | Unread

Write a reply...