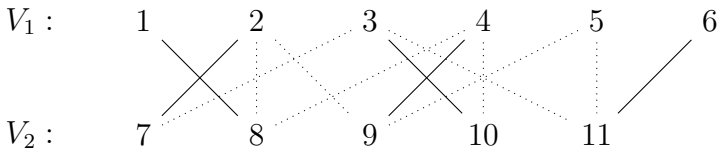
CSCI 3104 Spring 2018 Problem Set 10
Alison Ostlund (05/22)
Worked with: Kyle Ward, Hannah Haines, Kyle Rose, Jason Lubrano

---

Problem 1

(15 pts total) A matching in a graph G is a subset $E_M \subseteq E(G)$ of edges such that each vertex touches at most one of the edges in E_M. Recall that a bipartite graph is a graph G on two sets of vertices, V_1 and V_2, such that every edge has one endpoint in V_1 and one endpoint in V_2. We sometimes write $G = (V_1, V_2; E)$ for this situation. For example:



The edges in the above example consist of all the lines, whether solid or dotted; the solid lines form a matching.
The bipartite maximum matching problem is to find a matching in a given bipartite graph G, which has the maximum number of edges among all matchings in G.

(a) *Prove that a maximum matching in a bipartite graph $G = (V_1, V_2; E)$ has size at most* $\min\{|V_1|, |V_2|\}$.

Each vertex in V_1 has only one edge, with a corresponding vertex in V_2 where V_1 and V_2 are subsets G. This is due to the definition of Matching. The max edges in a matching can only be as large as the max number of edges in the smaller subset of vertices's. In the case that every vertex in the smaller subset has an edge, when adding another edge there would be 2 edges, then the definition of Matching would not hold, thus the graph G' would no longer be a matching graph. This shows that the max edges in a matching can only be as large as the max number of edges in the smaller subset of vertices's. Adding another edge to the matching would require another vertex in the smaller subset which does not already have an edge, which is not the case here.

(b) *Show how you can use an algorithm for max-flow to solve bipartite maximum matching on undirected simple bipartite graphs. That is, give an algorithm which, given an undirected simple bipartite graph $G = (V_1, V_2; E)$:*

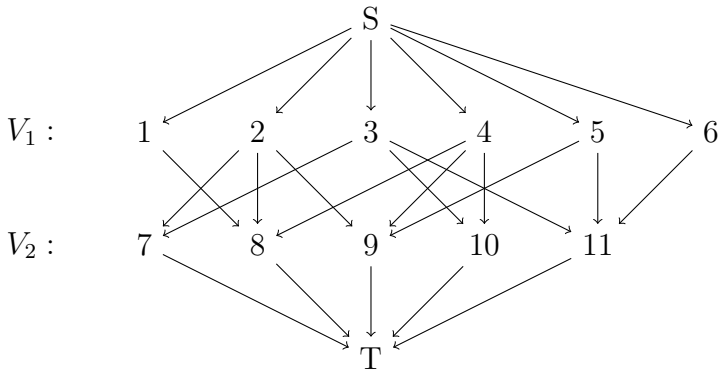
(1) *constructs a directed, weighted graph G' (which need not be bipartite) with weights $w : E(G') \to \mathbb{R}$ as well as two vertices $s; t \in V(G')$,*

(2) *solves max-flow for* $(G', w), s, t$*, and*

(3) *uses the solution for max-flow to find the maximum matching in* G*. Your algorithm may use any* `max-flow` *algorithm as a subroutine.*

```
Add S and T to the graph
Connect S and T to V1 and V2 using directed arrows so that
(S -> V1, V1-> V2, V2 -> T)
Add weights of 1 to the edges
Construct a residual capacity graph, no need for reduced graph since there
are no back edges
Solve for Max-Flow using residual graph
Extract Max-Flow and remove all edges from S and T
Remove all residual edges and edges with not in Max-Flow
```

(c) *Show the weighted graph constructed by your algorithm on the example bipartite graph above.*



All edges in the graph have a weight of 1

Problem 2

(20 pts total) In the review session for his Deep Wizarding class, Dumbledore reminds everyone that the logical definition of NP requires that the number of bits in the witness w is polynomial in the number of bits of the input n. That is, $|w| = poly(n)$. With a smile, he says that in beginner wizarding, witnesses are usually only logarithmic in size, i.e., $|w| = O(\log n)$.

(a) *Because you are a model student, Dumbledore asks you to prove, in front of the whole class, that any such property is in the complexity class P.*

Since we are using binary in this we need to represent a number n in any base. So would need $base^{size}$ solutions to check against the witness. Therefore with binary representation, it would be 2^{size}. Since the size in this case is $\log(n)$,there is $2^{\log_2(n)}$ solutions to check against the witness.
$2^{\log_2(n)} = n$.
Thus, if we have n solutions to check the witness against, the problem can be solved in O(n) time, which is polynomial.

(b) *Well done, Dumbledore says. Now, explain why the logical definition of NP implies that any problem in NP can be solved by an exponential time algorithm.*

The runtime for any solvable problem is $O(n^n)$, which is the longest time you can take to solve a problem that is exponential. NP means that a problem can be solved, not necessarily knowing in what time, and be verified as polynomial time. Since the problem can be solved an verified we can say that it had to take at most exponential time. Therefore, any NP problem can be solved by an exponential algorithm.

(c) *Dumbledore then asks the class: "So, is NP a good formalization of the notion of problems that can be solved by brute force? Discuss." Give arguments for both possible answers.*

All NP problems can be solved by brute force, in exponential time. Since the problem is NP it is also verifiable as polynomial time,P. Which you can argue that because of this, NP encompasses brute force problems.
There is a subset of problems that can be solved by brute force. These subsets although cannot be verified in polynomial time and therefore are not NP problems. This case shows that NP is not a good formalization of problems that be solved using a brute force approach. This is due to the fact that you are including problems that aren't NP problems.

Problem 3

(30 pts total) The Order of the Phoenix is trying to arrange to watch all the corridors in Hogwarts, to look out for any Death Eaters. Professor McGonagall has developed a new spell, Multi-Directional Sight, which allows a person to get a 360-degree view of where they are currently standing. Thus, if they are able to place a member of the Order at every intersection of hallways, they'll be able to monitor all hallways. In order not to spare any personnel, they want to place as few people as possible at intersections, while still being able to monitor every hallway. (And they really need to monitor every hallway, since Death Eaters could use Apparition to teleport into an arbitrary hallway in the middle of the school.) Call a subset S of intersections "safe," if, by placing a member of the Order at each intersection in S, every hallway is watched.

(a) *Formulate the above as an optimization problem on a graph. Argue that your formulation is an accurate reflection of the problem. In your formulation, show that the following problem is in NP: Given a graph G and an integer k, decide whether there a safe subset of size $\leq k$.*

Using the intersections as vertices's and the edges as hallways, the min number of watchers needed can be found using a min-vertex cover algorithm. This will use the algorithm to find the smallest possible number of vertices's are needed for a given graph such that all edges are marked as "touched" or in this case safe. This is a NP problem because it is a decision problem. It has to go through each vertex to check for safe edges and for any unmarked edges, which makes the problem non-deterministic. This goes through each vertex to check for current safe hallways and potential safe edges which ensures that safe edges are ¡= k (number of intersections).

(b) *Consider the following greedy algorithm to find a safe subset:*

```
S = empty
mark all hallways unwatched
while there is an unwatched intersection
    pick any unwatched hallway; let u,v be its endpoints
    add u to S
    for all hallways h with u as one of its endpoints
        mark h watched
    end
end
```

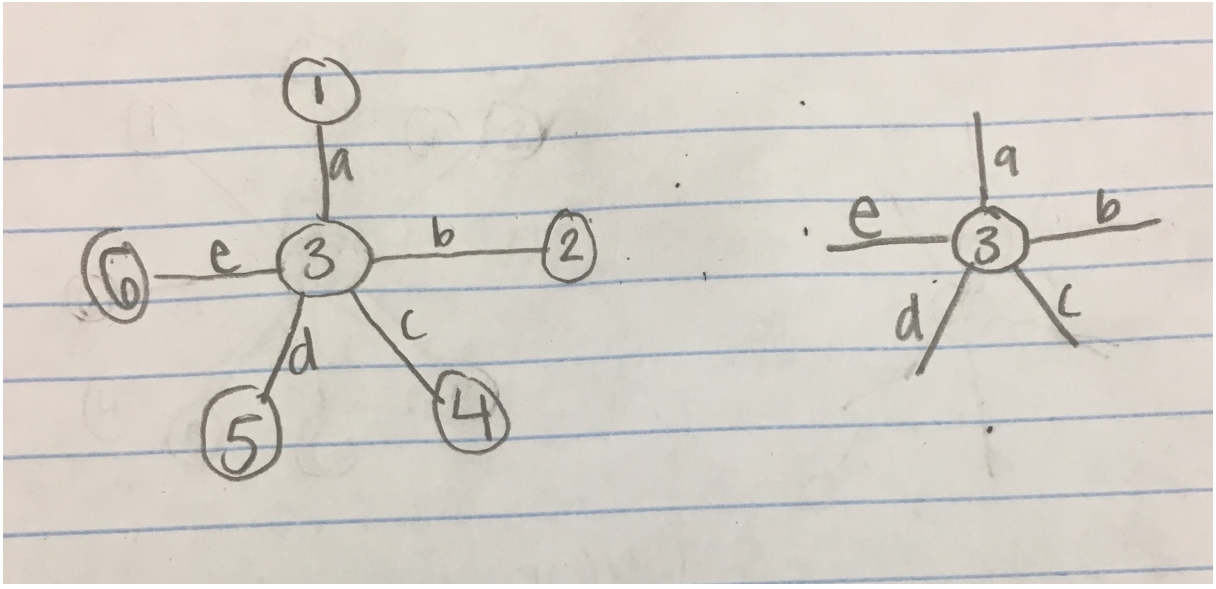
Although this algorithm need not find the minimum number of people needed to cover all hallways, prove that it always outputs a safe set, and prove that it always runs in polynomial time.

This algorithm will always produce a safe set because of the while loop. The algorithm will not exit the while loop if there are any unwatched hallways. The algorithm will add one vertex at a time marking its edges as safe until there are no unsafe hallways. This way the while loop will not terminate until there is no unwatched hallways, therefore it will always produce a safe set. The algorithm will go through and check every edge to see if it is "safe", so that produces a runtime of O(E). Once the algorithm adds a vertex to the set, it will need to check all of its corresponding edges, giving a run time of O(E) for that. These together produce a total runtime for the algorithm to be O(E^2). Since $O(E^2) \leq O(n^k)$ where k is a constant and n is the number of vertices's. Therefore $O(E^2)$ is polynomial time.

(c) *Note that, in order to be polynomial-time, an algorithm for this problem cannot simply try all possible subsets of intersections. Prove why not.*

The number of all possible subsets of n intersections would be n^n. If the algorithm tried all of the possible subsets of the n intersections, it would have to go through n^n cases, which would take exponential time. Therefore it would not run in polynomial-time.

(d) *Give an example where the algorithm from (b) outputs a safe set that is strictly larger than the smallest one. In other words, give a graph G, give a list of vertices in the order in which they are picked by the algorithm, and a safe set in G which is strictly smaller than the safe set output by the algorithm.*




```
Graph 1: (left graph)
vertex (1,3) is picked
   vertex 1 is added to S
   edge a in between (1, 3) is marked as safe
Algorithm moves on and picks vertex (3,2)
```

```
   vertex 3 is added to S
   edges b,c,d,e between vertices's (3,2), (3,4), (3,5), and (3,6) is marked
   safe
Algorithm terminates because all edges are marked safe and will return S
{1,3}

Graph 2: (right graph)
Vertex (3,1) is picked
   vertex 3 is added to S
   edges a, b,c,d,e between vertices's (3,1),(3,2), (3,4), (3,5), and (3,6)
   is marked safe
All edges are marked safe and will return S = {3}

Graph 1 has a safe set S that is greater than Graph 2's safe set S, therefore
graph 2 has an safe set that is strictly smaller than using the algorithm to
get graph 1.
```

(e) *Consider the following algorithm:*

```
S = empty
mark all hallways unwatched
while there is an unwatched hallway
    pick any unwatched hallway; let u,v be its endpoints
    add u,v to S
    for all hallways h with u or v one of their endpoints
        mark h watched
    end
end
```

Although this algorithm need not find the minimum number of people needed to cover all hallways, prove that it always outputs a safe set, and prove that it always runs in polynomial time.

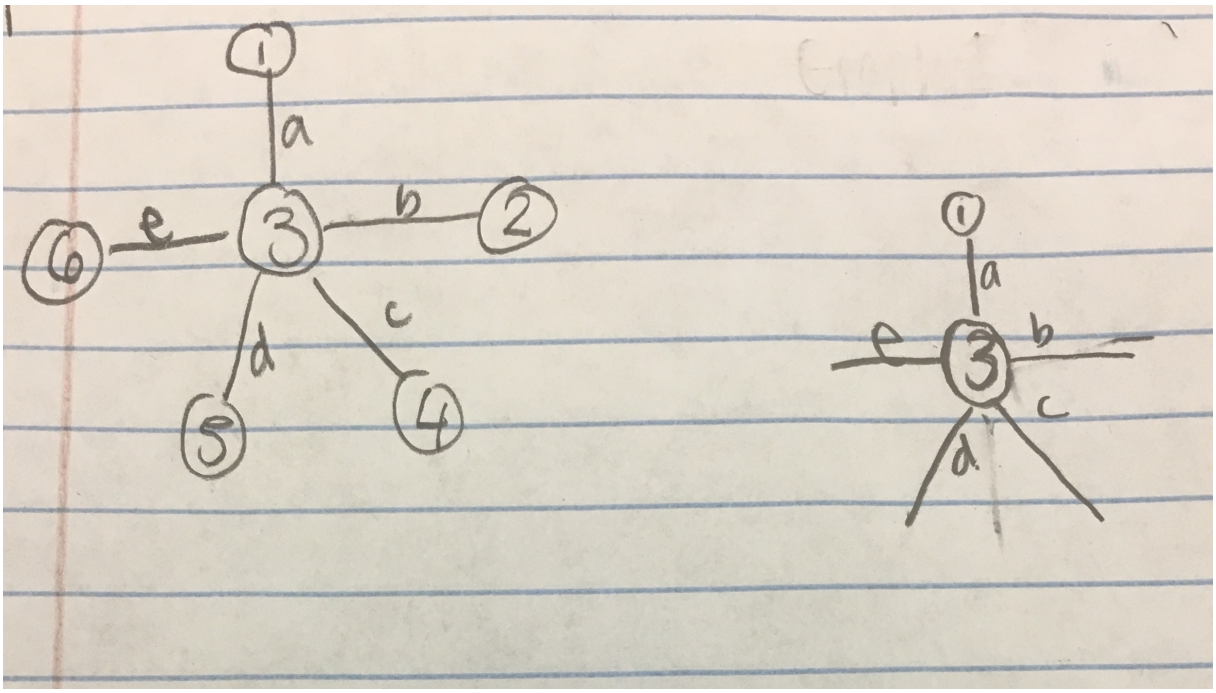
This algorithm is essentially the same as the one in (b). This one differs though because instead of adding one vertex at a time to the safe set, this one adds two at a time. This allows the hallway between both vertices's or intersections, to be marked as safe. This algorithm will not terminate until there are no unwatched hallways left, there always producing a safe set. For the running time, the algorithm will check every edge to see if it is "safe", producing a runtime of O(E). Then the algorithm adds the intersections to the set, it will need to check all their corresponding edges, giving a run time of O(2E) for that. These together produce a total runtime for the algorithm to be O(E^2), which is polynomial.

(f) *In any safe set of intersections, each hallway is watched by at least one member of the Order. Use this to show that the algorithm from (e) always outputs a safe set*

whose size is no more than twice the size of the smallest safe set. Note: you don't need to know what the smallest safe set is to prove this! All you need is the fact stated here.
This is called a "2-approximation algorithm," because it is guaranteed to output a solution that is no worse than a factor of 2 times an optimal solution.

The algorithm adds both intersections of a hallway at once every time. So the algorithm is basically adding twice the number of intersections it actually needs to watch a hallway. Then the algorithm will check the hallway between both intersections as "safe" therefore the algorithm will not be checking that certain hallway again. This implies that the algorithm can never add more than twice the minimum number of intersections needed.

(g) *Does the algorithm from (b) always produce a safe set no bigger than that produced by the algorithm in (e)? If so, give a proof; if not, give a counterexample. A counterexample here consists of a graph, and for each algorithm, the list of vertices it chooses in the order it chooses them, such that the safe set output by algorithm (b) is at least as large as the safe set output by algorithm (e). If you are unable to give either a proof or a counterexample, then for partial credit give a plausible intuitive argument for your answer.*




```
Graph 1 using alg b (left):
vertex (1,3) is picked
   vertex 1 is added to S
   edge a is marked as safe
Algorithm moves on and picks vertex (2,3)
```

```
   vertex 2 is added to S
   edge b is marked as safe
Algorithm moves on and picks vertex (4,3)
   vertex 4 is added to S
   edge c is marked as safe
"   "
... repeats for vertices's 5 and 6 marking edges d and e as safe
All edges marked as safe
Produces a safe set S = {1,2,4,5,6}

Graph 2 alg e (right):
Vertex (1,3) is picked
   vertex 1 and 3 is added to S
   edge a,b,c,d,and e is marked as safe
All edges marked as safe
Produces a safe set S = {1,3}


In this case, Graph 1 using algorithm (3b) produces a safe set that is much larger
```

(h) *Compare the greedy algorithm from (e) with the greedy algorithm from (b). Show which runs faster asymptotically? Which of these two algorithms would you rather use to solve the Order of the Phoenix's problem and why?*

The run time for both algorithms is the same. They both run in $O(E^2)$ time, proven by the proofs in above questions. To solve the Order of the Phoenix's problem however, I choose the second algorithm (3e). This is because it adds two intersections at once. It is guaranteed to be no more than twice the minimum of people needed. Where as (3b) doesn't guarantee this, so (3e) will produce a smaller number a watchers needed than produced with the algorithm in (3b).

(i) *This problem is, in fact, NP-complete. Why does the 2-approximation polynomial-time algorithm from (e) not show that P=NP ?*

For the 2-approximation polynomial-time algorithm, if you were to use a brute force approach to run through this algorithm, you would end up with a time of $O(2^n)$. Also the algorithm doesn't always return the optimal solution.This and the exponential time,implies that P is not encompassed in NP, therefore P is not equal to NP.