Solutions to Homework 7, CS 173A (Fall 2018)

Homework 7 asked you to show that the construction problems Maximum Independent Set, Minimum Vertex Coloring, and Maximum Clique could all be solved in polynomial time on path graphs.

Recall that a path graph $P_n = (V, E)$ has vertex set $V = \{v_1, v_2, \ldots, v_n\}$ and edge set $E = \{(v_i, v_{i+1}) | i = 1, 2, \ldots, n-1\}$. We will assume that the input is a path graph with n vertices, matching this specification. (We will also not check first that the input satisfies this...we just assume it.)

You are asked to come up with a polynomial time algorithm for the construction problems stated above. What this means, in terms of a list of things you need to do, is:

- State the algorithm (clearly). Most importantly, the algorithm should be correct - you should be confident about it. Even if you don't know how to prove it's correct.
- Justify the running time being polynomial. Any polynomial is fine. Also, it's not important to provide a detailed analysis - handwaving is really okay, as long as you say something reasonable.
- Explain why the algorithm returns a "feasible" solution (e.g., a clique when a maximum clique is desired, a proper vertex coloring when a minimum vertex coloring that is proper is desired, and an independent set when a maximum independent set is desired).
- Explain why the output from the algorithm is the best possible. So, among all possible feasible solutions, what your algorithm returns is the best – biggest independent set, biggest clique, and a coloring that uses the smallest number of colors.

Use simple pseudocode. If necessary, explain how to read it.

Also, realize that there are many ways to solve these problems - different algorithms will work, and different proofs will work. It's not at all important that you do your algorithm in any particular way nor that you prove your algorithm correct in any particular way. But be simple in how you write your algorithm so that your audience will understand the algorithm and believe it's correct. And that it's polynomial time. Try to be simple in how you explain why it's correct. Think about implementing the algorithm!

1 Algorithm and Proof for Minimum Vertex Coloring

Note that in my pseudocode, I will use $x := y$ for the assignment operator, and $x = y$ for the boolean test of equality.

1.1 Algorithm

Given the input P_n, assign color red and blue as follows: we assign red to all v_i where i is odd and we assign blue to all v_i where i is even. The output (i.e., the coloring) is an array $\mathit{color}[1 \dots n]$, so that $\mathit{color}[i]$ is the color assigned to v_i. Equivalently, we return the array $\mathit{color}[1 \dots n]$ constructed using the following (dynamic programming) code:

```
• $\mathit{color}[1] :=$ red
• for $i := 2$ up to $n$ do:
    – if $\mathit{color}[i-1] =$ red then $\mathit{color}[i] :=$ blue else $\mathit{color}[i] :=$ red.
• return array $\mathit{color}$
```

1.2 Proof

We need to prove the following:

1. The algorithm takes polynomial time.
2. The algorithm produces a proper vertex coloring for P_n.
3. There is no proper vertex coloring that uses fewer colors.

The running time to run the algorithm on input P_n is clearly $O(n)$.

To prove that the algorithm produces a proper coloring requires we show that for all edges $(v_i, v_j) \in E$, $\mathit{color}[i] \neq \mathit{color}[j]$. Remember that all vertices with odd indices get one color and all vertices with even indices get the other color. Now note that all edges in P_n are between vertices v_i and v_{i+1} (for some i), so that all edges are between vertices that have indices with different parities. Hence, no two adjacent vertices get the same color.

To prove that there is no proper coloring with fewer colors, we note that the algorithm uses only one color if $n = 1$ and otherwise uses two colors. It's very clear that P_1 can be colored with one color, and that P_n cannot be colored with only one color if $n > 1$ (since the edge (v_1, v_2) requires two colors).

Therefore, the algorithm provably solves the Minimum Vertex Coloring construction problem and does so in polynomial time, when the input is a path graph.

2 Algorithm and Proof for Maximum Clique

2.1 Algorithm

For the same input, we have the following algorithm:

```
• If $n = 1$, return $\{v_1\}$
• If $n \geq 2$, return $\{v_1, v_2\}$.
```

2.2 Proof

We need to prove the following:

1. The algorithm takes polynomial time.
2. The algorithm produces a clique for P_n.
3. There is no larger clique in P_n.

It's really clear that the algorithm takes $O(n)$ time - we just need to check the number of vertices is equal to 1 or more than 1.

(In case you're curious – in fact, one could argue that the running time is $O(1)$, since we can stop processing the input as soon as we know it has two or more vertices. This is a somewhat tricky point... don't worry if you find this confusing.)

To show that the algorithm returns a clique on all n, note that for $n = 1$ we return a single vertex (vacuously a clique) and for $n \geq 2$ we return a pair of vertices that are adjacent. Hence, we are returning a clique on all inputs. (Note to students; I would not have minded if you just said "it's obvious"...because it really is.)

Now, we need to show that there is no bigger clique than what the algorithm returns. This is obviously true for P_1. We will prove that this is true by contradiction for $n \geq 2$. So, suppose P_n has a clique X that is bigger than what the algorithm returns. Thus, for $n \geq 2$, this would mean X has at least three vertices. Let v_i, v_j, v_k be any three of the vertices in X. Without loss of generality let $i < j < k$. Then $k \geq i + 2$. But then $(v_i, v_k) \notin E$, contradicting the assumption that X is a clique and contains both v_i and v_k. Hence, the largest clique in P_n has only two vertices for $n \geq 2$ and only one vertex for $n = 1$.

Therefore, the algorithm provably returns a largest clique in P_n for all n, and runs in $O(n)$ time.

3 Algorithm and Proof for Maximum Independent Set

3.1 Algorithm

The algorithm is:

- Return $\{v_i | 1 \leq i \leq n, \text{i is odd}\}$.

For example, on P_5 we would return $\{v_1, v_3, v_5\}$. Note also that the algorithm returns a set with $\lceil \frac{n}{2} \rceil$ vertices on input P_n. Equivalently, the algorithm returns a set with m vertices when $n = 2m$ and a set with $m+1$ vertices when $n = 2m+1$.

3.2 Proof

We need to prove the following:

1. The algorithm takes polynomial time.
2. The algorithm produces a maximum independent set for P_n.
3. There is no larger independent set in P_n.

It's easy to see that the algorithm uses $O(n)$ time.

To see that the algorithm produces an independent set, note that every edge connects vertices v_i and v_{i+1} where i is some integer between 1 and $n-1$. Therefore, no two vertices in the output from the algorithm are adjacent, and hence the output is an independent set.

The last part is to prove that P_n does not have a larger independent set. We prove this separately for the two cases (where n is even or where n is odd).

- Case: n is even. Hence, $n = 2m$ and the algorithm returns a set with m vertices. Partition the set of vertices into m disjoint subsets, $\{v_1, v_2\}$, $\{v_3, v_4\}, \ldots \{v_{n-1}, v_n\}$. Note that each of these subsets contains two vertices that are adjacent to each other. Therefore, every independent set in P_n can have at most one of the vertices from any one of these sets. Hence every independent set can have at most m vertices, and so the algorithm outputs a largest independent set when n is even.

  We could also have proven this by contradiction. Suppose that P_n has an independent set X with at least $m+1$ vertices. The Pigeonhole Principle says you cannot put k pigeons into $k' < k$ pigeonholes without putting at least two pigeons into the same pigeonhole. By the Pigeonhold Principle, X must have both vertices from at least one of these sets. Hence, X cannot be an independent set.

- Case: n is odd, so $n = 2m + 1$. Recall that the algorithm returns a set with $m + 1$ vertices. Also, the algorithm includes v_n, since n is odd. Now, partition the set of vertices into $m + 1$ sets, $\{v_1, v_2\}, \{v_3, v_4\}$, ..., $\{v_{n-2}, v_{n-1}\}$, $\{v_n\}$. Note that the first m of these sets have two vertices each (and both are adjacent to each other) and the last set has a single vertex. Therefore, every independent set can have at most one element from each of the $m+1$ subsets, and so can have at most $m+1$ vertices.

  Again, we could have proven this by contradiction, and said suppose P_n has an independent set X that has more than $m + 1$ vertices. Then by the Pigeonhole principle, X has both of the vertices from at least one of these sets, and hence cannot be an independent set.

Thus, we have proven that the algorithm produces a maximum independent set for every path graph, and does this in polynomial time.

4 Thinking about other problems

Now that you've done this, think about the next problems.

For each of the following construction problems, and for each of the different types of graphs, see if you can find a polynomial time algorithm to solve the problem. Then think about how to prove your algorithm is correct. Remember the steps: (a) describe the algorithm, (b) establish that the algorithm produces a feasible solution, and (c) convince your reader that the output of your algorithm is the best possible (biggest or smallest, depending on the problem).

- Construction problems:
  - Minimum Vertex Cover
  - Maximum Independent Set
  - Maximum Clique
  - Minimum Vertex Coloring
  - Minimum Dominating Set
- Graphs:
  - K_n
  - $K_{n,m}$
  - P_n
  - W_n

Also, try all those construction problems on *trees*. This is easy for some of the problems but not for all. (For example, it's easy to solve Maximum Clique...but not as obvious how to solve Maximum Independent Set.) To start with, just try to come up with an algorithm... try it out on a bunch of trees, without worrying about how to prove it's correct. Worth thinking about.