CS-3510-02 Algorithms, Spring 2025 — Mid-term 2 take home part

Hayden Hawley — Collaborator(s):

Academic Integrity Requirement: You must write your solution in your own words. Do not copy from external sources, classmates, or generative AI tools.

Problem 1 (10 pts)

An independent set I in an undirected graph $G = (V, E)$ is a subset $I \subseteq V$ of vertices such that no two vertices in I are joined by an edge of E. Consider the following greedy algorithm to try to find a maximum size independent set which is based on the general idea that choosing vertices with small degree to be in I will rule out fewer other vertices:

Algorithm 1 GreedyIndependentSet

```
I ← ∅
while G is not empty do
    Choose a vertex v of smallest degree in G        ▷ Not counting deleted edges
    I ← I ∪ {v}
    Delete v and all its neighbors and their connected edges from G
                                  ▷ None of the neighbors can be included since v is included
end while
return I
```

Prove that this algorithm is incorrect by counterexample. Specifically, give an example of a graph on which this algorithm does not produce a largest size independent set. Show both the independent set that the algorithm finds and a larger independent set.

Solution:

The greedy algorithm does **not** always find the biggest possible independent set. Consider this graph:

```
A   D
 \ /
  B
 / \
C   E
```

Edges: A–B, B–C, B–D, B–E

- Outer nodes (A, C, D, E) all have degree 1; B has degree 4.
- A greedy algorithm might pick A first (lowest degree).
- It then removes A and B.
- We're left with C, D, and E. Picking C removes it, leaving D and E.
- Final independent set might be $\{A, C\}$ (size 2).

- But the maximum independent set is $\{A, D, E\}$ (size 3).

Therefore, the greedy algorithm does not always find the best possible result.

Problem 2 (10 pts)

Suppose A is an array of n integers that is a strictly decreasing sequence, followed by a strictly increasing sequence such as $[12, 9, 8, 6, 3, 4, 7, 9, 11]$. Give an $O(\log n)$ algorithm to find the minimum element of the array. Justify your algorithm is correct.

Problem 3 (10 pts)

Consider a graph that is a path, where the vertices are $v_1, v_2, \ldots, v_n$, with edges between v_i and v_{i+1}. Suppose that each node v_i has an associated weight w_i. Give an algorithm that takes an n-vertex path with weights and returns an independent set of maximum total weight. The runtime of the algorithm should be polynomial in n. Justify your algorithm is correct.