CS381 Homework 0 Problem 3

Connor Couetil

January 22, 2026

1 Exercise 2.2.3

The following is a brute-force algorithm that selects all possible combinations of edges, counting each edge as part of the matching set only if it is disjoint with edges selected later. The base case is when the graph G as one edge, the matching set is one.

Recursive Algorithm

`max-matching`($G = (V, E)$)

1. If $|E|$ is 1
   A. Return 1
2. Let there be an empty set M
3. For each edge e in E
   A. Remove e from G, producing $G' = (V, E')$
   B. If e is disjoint with all edges in G'
      1. Add (1 + max-matching(G')) to M
   C. Else
      1. Add (max-matching(G')) to M
4. Return the largest value in M

2 Exercise 2.2.4

The key observation is that if we select some vertex to be part of the independent set, the other endpoint in all the edges it's a part of cannot be part of that independent set. So we can remove those vertexes and their associated edges from the graph, and recurse until no edges are left, with each recursion iteration choosing one next possible independent vertex. If we do this exhaustively, and bubble up the maximums from each recursive call, we'll find the maximum independent set size for the original graph.

Recursive Algorithm

`max-independent-set`($G = (V, E)$)

1. If $|E|$ is empty
   A. Return $|V|$

2. Let there be an empty set, M
3. For each vertex v in V
   A. Let G' equal G
   B. For each edge $e = (v, v')$ in E containing v
      1. Remove any edges containing v' from G'
      2. Remove the vertex v' from G'
   C. add (1 + max-independent-set(G')) to M
4. Return the largest value in M

3 Exercise 2.2.5

We'll recurse on all values of x greater than x_1, those sub-sequences are the same problem. The approach is depth-first search on all possible orderings of values in the array. There's probably a more efficient approach where you discard inversions and recurse to combine subsequences.

Recursive Algorithm

`longest-increasing-subsequence-including-first(`x_1`, ..., `x_n`)`

1. If n is 1
   A. Return 1
2. Let there be an empty set, M
3. For x_i in x_2, ..., x_n
   A. If $x_i > x_1$
      1. Add (1 + longest-increasing-subsequence-including-first(x_i, ..., x_n)) to M
4. Return the largest value in M

4 Exercise 2.2.6

For each edge s is a part of, if its other endpoint can react t then s can reach t. The base case is if s has an edge with t.

`reachable(`$G = (V, E)$`, `s`, `t`)`

1. For each edge $e = (s, v)$ in E containing s as the start
   A. If v is t
      1. Return true
   B. Return reachable($G = (V, E)$, v, t)
2. Return false

5 Exercise 2.2.7

The partition problem is similar to the subset sum problem. The key observation is that we can reduce the size of the problem by combining two elements into one and recursing. If the elements will be in the same partition in a final solution, then we can combine them into their sum. If the elements will not be in the same partition in the final solution, then we can cancel them out by their difference.

`partition(`x_1`, ..., `x_n`)`

1. If n is 0
   A. Return true
2. If n is 1
   A. Return $x_n == 0$
3. Return partition($x_1 + x_2$, ..., x_n) $\vee$ partition($x_1 - x_2$, ..., x_n)