Certifying Algorithms

Correctnesss of algorithms?

- Formal proof of algorithm correctness
  - only simple problems?
  - implementation ≠ algorithm

- Compare output of two algorithms
  - one algorithm often simple and slow (only small input)

- Assertions / exceptions

- Unit testing
  - systematic testing, random input
Certifying Algorithm

- **Algorithms output proof** \( w \) of **correctness** or **illegal input**
- **Strongly certifying** \( \Rightarrow \) halts on all input; identifies illegal input
- **Certifying** \( \Rightarrow \) halts on all input; illegal input or correct output
- **Weakly certifying** \( \Rightarrow \) halts on valid input; if halts, correct output
- **Motivation**: Ensure correctness of algorithms in the *Library of Efficient Data Types and Algorithms*
Sorting ?

- **Input**: Unsorted array
- **Output**: Input elements in sorted order

- **Checker**:
  - Verify output sorted
  - Verify output = input elements
Greatest Common Divisor - GCD

- **Input**: Positive integers \( a \) and \( b \)
- **Output**: \( g = \gcd(a, b) \)
- **Certificate**: 
  - Integers \( x, y \): where \( g = ax + by \)
- **Checker**: 
  - Check \( g \uparrow a, \ g \uparrow b, \text{ and } g = ax + by \)
  - Sufficient by [MMNP11, Lemma 1]
Bipartite Graph?

- **Input:** Undirected Graph \( G=(V,E) \)
- **Output:** Boolean, is the graph bipartite

- **Certificate:**
  - True: Partition of the vertices, \( V = V_1 \cup V_2 \)
  - False: Odd length cycle

- **Checker:**
  - Verify partition or cycle
Connected Components ?

- **Input**: Undirected graph $G = (V, E)$
- **Output**: Partition of $V$ into the c.c.
- **Certificate**:
  - Each vertex labeled $(i, j)$, where $i$ = component number, $j$ = the nodes number in the component, such that all nodes except one in a c.c. have a neighbor with smaller $j$ (e.g., BFS numbering)
- **Checker**:
  - Edges connect identical $i$
  - Mark non-root nodes ($j$ larger than a neighbor)
  - Check roots different labels
Shortest Path $s \rightarrow t$?

- **Input:** Directed weighted graph $G = (V, E)$, $s, t \in V$
- **Output:** Shortest distance $s \rightarrow t$
- **Certificate:**
  - Distance vector $D$, with distances from $s$ to all nodes
  - Shortest path tree
- **Checker:**
  - Check shortest path tree implies $D$
  - Check that no edge can improve any distance
Planarity Graph?

- **Input**: An undirected graph $G$
- **Output**: Boolean, is $G$ planar
  - Can $G$ be drawn without edges intersecting?
- **Certificate**:
  - Yes = (Combinatorial) Embedding (twin edges, face information)
  - No = $K_{3,3}$ og $K_5$ (Kuratowski subgraphs)
- **Checker**:
  - Yes: Check if $n+f=m+2$, $n=$#nodes, $m=$#edges, $f=$#boundary cycles (sufficient by [MMNS11, Lemma 3])
  - No: Verify Kuratowski subgraphs
Maximum Flow?

- **Input**: Flow network $G$, with capacity constraints $c$
- **Output**: Value of maximum flow

- **Certificate**:
  - Flow along each edge
  - Minimum cut, i.e. partition of the vertices

- **Checker**:
  - Check if valid flow
  - Find capacity of cut
  - Check if cut capacity is equal to value of flow
Dynamic Dictionary

- **Operations**: Insert, Delete, Search, ...

- **Checker / Monitor**: 
  - Check maintains a doubly-linked list of *handles* into dictionary

- Checker identifies wrong queries immediately
Priority Queue

- **Operations**: Insert, DeleteMin ...
- **Checker / Monitor**: (see figure)
  - check element against lower bound on deletion
- **Checker** identifies wrong queries delayed