Cheat Sheet INF237

Algorithms engineering – Tommy Odland – Last edit: June 2, 2018

▷Practial programming and Python

- Some modules from the *Python standard library* are indispensable in programming contests:
  - `bisect` for working with binary search.
  - `defaultdict` and `deque` from `collections`.
  - `heapq` for working with priority queues.
  - `itertools` for permutations, combinations, etc.
  - `reduce` from `functools` is sometimes useful.
  - See also `math`, `string`, `operator` and `random`.
  - Learn how to use the built-in data structures.
- Implementing a naive algorithm and testing against it on random inputs is often wise.
- If Python 3 is too slow, try Python 2. Use the `%timeit` and `%lprun` IPython functions to time code and identify slow parts. Use Java if all fails.

▷Sliding, searching and sorting

- Comparison based sorting runs in $\mathcal{O}(n \log n)$ time.
- Maintaining a sorted list can speed up algorithms, since insertion is $\mathcal{O}(\log n)$ and retrieval is $\mathcal{O}(\log n)$. This advice has broad applicability.
- For problems "find minimal k such that $P(k)$", it might be possible to use binary search to find k if "Is $P(k)$ possible?" can be answered quickly.

▷Graph algorithms

- BFS and DFS are used to explore graphs. Both run in $\mathcal{O}(V+E)$ time. DFS uses a stack, BFS uses a queue. May be implemented recursively too.
- A *minimal spanning tree* may be found quickly using Krusal's algorithm or Prim's algorithm.
  - Prim's algorithm uses a priority queue.
  - Kruskal's algorithm uses a *disjoint-set data structure* to check if vertices are in the same set.
- Dijkstra's algorithm finds the single source shortest path to every other vertex in a graph.
  - A more general algorithm is the A^* search.

▷Dynamic programming

- Dynamic programming is used for problems with overlapping sub-problems, i.e. problems that obey the *principle of optimality.*
- The recipe is to (1) find a recursive formula, (2) identify the base cases (initial conditions) and (3) solve the problems such that no problem is solved twice. Either bottom-up to top-down using cache.
- Typically one creates a *DP table*, an n-dimensional table of function values. Dynamic programming is also possible over other data structures, such as trees and more general graphs.

▷Exponential time algorithms

- Some algorithms run in exponential time, such as graph coloring.
- Approximate algorithms may often be used if finding the best answer is not necessary.
- If the exact answer is needed, use *backtracking* in an implicit tree to prune infeasible solutions as soon as possible to reduce run time in practice.

▷Computational geometry

- The area of a *convex polygon* may be found in $\mathcal{O}(n)$ time using the shoelace formula.
- Given a set of n points in $\mathbb{R}^2$, their convex polygon may be found in $\mathcal{O}(n \log n)$ time using *Graham's scan*, which orders the point by angle and examines them in a linear fashion using a stack.
- Angles, line crossings and so forth may be tricky due to floating point arithmetic and special cases, but typically relatively straight forward.

▷Number theory

- The *Sieve of Eratosthenes* generates primes $\leq n$ in $\mathcal{O}(n \log \log n)$ time, since $1/2 + 1/3 + 1/5 + \cdots + 1/n = \log \log n$. It naively requires $\mathcal{O}(n)$ space.
- To verify if a number is prime, one option is to check numbers from 1 to $\sqrt{n}$, or use a precomputed prime list if available.

▷Segment trees

- Given an array A, computing sums $A[i \dots j]$ has $\mathcal{O}(1)$ update and $\mathcal{O}(n)$ query. A cumulative sum has $\mathcal{O}(n)$ update and $\mathcal{O}(1)$ query.
- A *segment tree* balances the requirements above for $\mathcal{O}(n \log n)$ update and $\mathcal{O}(n \log n)$ query. It may also be used for other functions apart from sums.

▷References

- Halim, Steven. *Competitive Programming*, 3rd Edition, 2013.
- Dasgupta, Sanjoy. *Algorithms.* McGraw Hill, 2008.