CS120: Intro. to Algorithms and their Limitations Prof. Salil Vadhan

Lecture 26: Review, Synthesis, and Conclusions

Harvard SEAS - Fall 2021 *Dec. 7, 2021*

1 Announcements

- Final Exam Fri 12/17, 9am-12pm, Science Center D
- There will be T-shirts!
- Review sessions, office hours and practice problems all coming - stay tuned!

Recommended Reading:

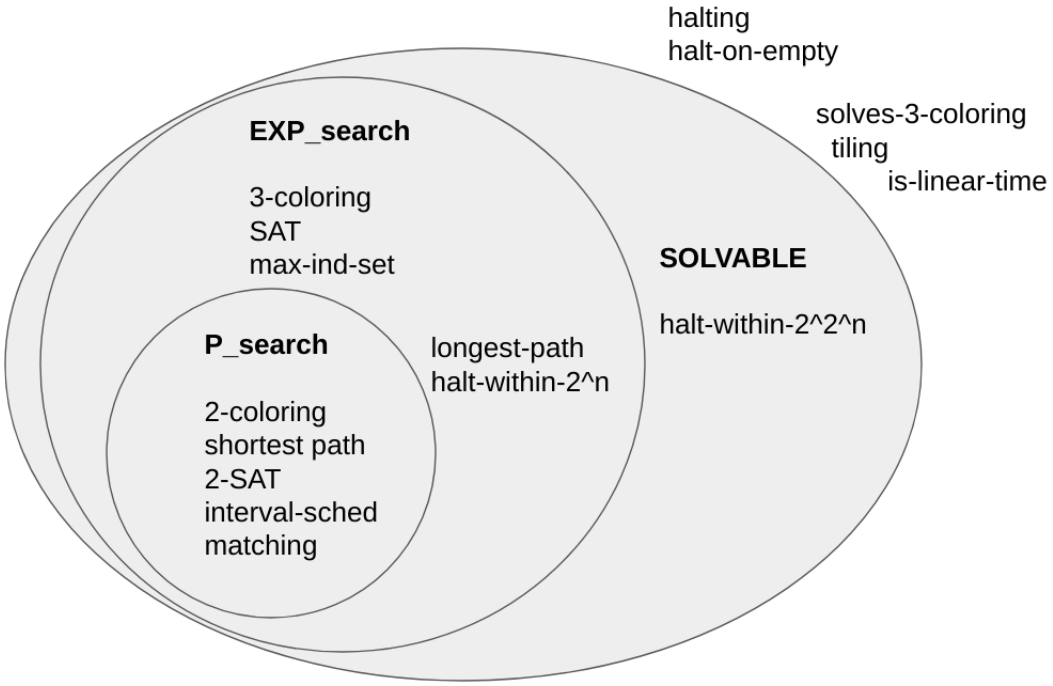
- Roughgarden IV, Epilogue
- MacCormick, Chapter 18

2 An Algorithmicists' Workflow

When confronted with a real-world algorithmic problem (like Web Search, Interval Scheduling, Census Data Releases, Google Maps, Kidney Exchange, Register Allocation, Programming Team, ...), you can tackle it using the skills from cs120 (and future classes) by looping through the following steps:

1. Mathematically model
    - What kinds of input data? Item-key pairs (CS165), graphs and combinatorial structures (AM107, Math 152), logical formulas (Math 141, Phil 144), programs (CS 152, CS 153)
    - What kind of output/queries/objective?
    - Is it a one-shot computational problem or data structure problem?
    - How much detail to retain? How much of the problem to model?
    - Social context and ethics? (CS108, CS126, CS136, CS208, CS226, CS238)
    - What is the right computational model and relevant resources to consider? (CS61, CS121, CS152, CS165)
2. Look for related problems (in class, in the literature, on the web)
    - How much details/structure to incorporate? (Many many cases discussed in class where incorporating a restriction makes the problem easier. For instance, SAT vs 2-SAT, Matching vs Bip. Matching, directed vs undirected graphs, unlimited universe vs bounded universe for sorting, edge weights vs unweighted graphs...)
    - What are the inputs, outputs, and objective function?

- Do we model this as a data structure (repeated input/output rounds) or an algorithm (runs once and returns an answer)
- Reduce to one of the problems we've seen:



3. Try to obtain an algorithm by **reduction to** other problems
   - IntervalScheduling, AreaOfConvexPolygon reducing to Sorting
   - Rotating Paths reducing to Shortest Paths
   - Bipartite Matching reducing to Rotating/Alternating Paths
   - 2-SAT reducing to Shortest Paths
4. Try to apply algorithmic techniques
   - BFS
   - Greedy
   - Exhaustive Search
   - Divide and Conquer (MergeSort, Randomized Selection)
   - Data Structures (Sorted Arrays, BSTs, Hash Tables)
   - Resolution
   - Many more in CS124, CS182, AM121, CS165: dynamic programming, DFS, priority queues, randomized algorithms, approximation algorithms, local search, linear/integer/-convex programming
   - ⋮

5. Try to show hardness/unsolvability by **reduction from** other problems
    - NP-hardness for combinatorial search problems
    - Unsolvability for program verification and logic problems
    - More in CS121, CS127, CS221, Phil144
6. And/or settle for weaker guarantees
    - Find more structure to incorporate in your modelling, or
    - Use algorithms/heuristics that have worst-case exponential time (e.g. SAT or SMT solvers)
    - Settle for approximation algorithms
    - More in CS124, CS152, CS182, AM121

3 Other Takeaways

- Universality
    - Turing-equivalent models
    - (Word-)RAM, Compilers
    - (Extended) Church-Turing Thesis
    - Universal programs
    - Technology-independence
    - More in CS121, CS61, CS152, CS153, Phil 144
- Rigorous mathematical theory
    - Computation and computational problems can be precisely modelled
    - Algorithms can be rigorously analyzed for correctness and efficiency
    - Some problems, including important ones we want to solve, can be *proven* to be impossible or intractable to solve
- There is much we don't know!
    - Can Sorting be done in $O(n)$ time? Matching in $O(n+m)$?
    - P vs. NP
    - Are polynomial-time randomized algorithms more powerful than deterministic polynomial-time algorithms? What about polynomial-time quantum algorithms?
    - Is secure cryptography possible?

4 CS120 Learning Outcomes

From the Syllabus: "By the end of the course, we hope that you will all have the following skills:

- To mathematically abstract computational problems and models of computation
- To design and implement algorithms using a toolkit of algorithmic techniques
- To recognize and formalize inherent limitations of computation
- To rigorously analyze algorithms and their limitations via mathematical proof
- To appreciate the technology-independent mathematical theory of computation as an intellectual endeavor as well as its relationship with the practice of computing."

5 Where to Learn More

- Theory of Computation seminar: http://toc.seas.harvard.edu/
- Many other CS courses, especially x2x. Look at grad (2xx) courses too. (CS120 may serve as a sufficient substitute for CS121/CS124 in some of them.)
- Read more of our textbooks (Roughgarden, MacCormick, CLRS, and the references therein)
- Come talk to me in office hours!