6.046 Review Topics


2. Randomized algorithms: Quicksort, Quickselect, deterministic select.

3. Hashing: Perfect hashing, universal hashing.


5. Amortized analysis: dynamic tables, aggregate/accounting/potential method

6. Skip lists

7. Computational geometry: Range searching (Kd-tree).
   - Construction algorithm:
     (a) Choose x or y coordinate (alternate)
     (b) Choose the median of the coordinate; this defines a horizontal or vertical line
     (c) Recurse on both sides
   - Size: $O(n)$
   - Depth: $O(\log(n))$
   - Construction time: $O(n \log n)$
   - Query time: $O(P + \sqrt{n})$.

8. Greedy approach: MST
   - Greedy-choice property: A locally optimal choice is globally optimal.
   - Prim’s algorithm: Growing MST from 1 vertex. Always pick the vertex that is closest to the current MST, add the minimum weight edge that joins the new vertex with the current MST to the MST. Repeat. $O(E + V \log(V))$
   - Kruskal’s algorithm: Sort all edges in order from min to max, add an edge into the MST if it does not create a cycle. $O(E \log(V))$

9. Dynamic programming:
   - Hallmarks:
     (a) Optimal sub-structure.
     (b) Overlapping sub-problems.
   - Longest common subsequence, parsimony, parsing context-free grammar
10. Shortest paths:

(a) Single source:

- Dijkstra $O(E + V \log(V))$ - only works with non-negative weight.
- Bellman-Ford $O(VE)$ - works with general weight.

(b) All pairs shortest paths:

- Floyd-Warshall: $O(V^3)$
- Min-plus Product: $O(V^3 \log(V))$.

11. Network flows:

(a) Max flow: Given a flow network $G$, find a flow of maximum value on $G$. Example here.

(b) Max flow - min cut theorem: The following are equivalent:

i. $f$ is a maximum flow.
ii. $G_f$ (the residual network) contains no augmenting paths.
iii. $|f| = c(S, T)$ for some cut $(S, T)$ of $G$.

- Ford-Fulkerson algorithm: Lec notes.

(c) Multi-sources multi-sinks flows: Create a universal source and a universal sink. Connect the universal source with all given sources and the capacity of the edges are the supplies of corresponding source. Connect all given sinks with the universal sink, the capacity of the edges are the demands.

(d) Bipartite matching: Similar to the multi source multi sink example, use supply=demand=1.

12. Linear programming:

(a) Maximizing/Minimizing a linear combination of variables, subject to linear constraints. Real-valued LPs solvable in poly-time. Integer-constrained LPs not necessarily.

(b) Network flow, shortest paths can be expressed as LPs. Multi-commodity flow solvable with LPs (see recitation notes) - no other poly-time algorithm known.

(c) LP in 2 dimensions (2 variables, $m$ constraints). Incremental algorithm takes $O(m^2)$ worst case, $O(m)$ expected time.

13. Complexity:

(a) P= decision problems solvable in polynomial time. NP = decision problems with solutions verifiable in polynomial time.

(b) NP-hard = everything in NP can be reduced to it in polynomial time. NP-complete = in NP and NP-hard.
(c) We say A is polynomial-time reducible to B (abbreviated \( A \leq B \)) if there is a polynomial-time transformation that takes instances of A to instances of B, in such a way that YES-instances go to YES-instances, and NO-instances go to NO-instances.

(d) We assumed that SAT was NP-complete. Then we showed that SAT \( \leq \) CLIQUE \( \leq \) INDEPENDENT-SET \( \leq \) VERTEX-COVER.

(e) SUBSET-SUM, KNAPSACK, and GRAPH-3-COLORABILITY also NP-complete (though we didn’t see all the proofs. The latter was done in recitation.)

14. Randomized Complexity:

(a) Randomized Algorithm takes regular input + uniformly random bits.

(b) RP= class of problems with a randomized algorithm that accepts YES-instances with probability \( > \frac{1}{2} \) (over choice of random bits), and always rejects NO-instances.

co-RP = class of problems with a randomized algorithm that always accepts YES-instances, and rejects NO-instance with probability \( > \frac{1}{2} \).

(c) Example co-RP problem: Matrix Product Checking.
Given \( A, B, C \) check if \( A \times B = C \). Choose random binary vector \( x \), check if \( ABx = Cx \). Algorithm takes \( O(n^2) \) time. We showed it’s in co-RP.

15. Approximation algorithms:

(a) for a minimization problem, a \( \rho \)-approximation algorithm produces an answer that’s at most \( \rho \) times the optimal answer.

(b) 2-approximation algorithm for Traveling Salesman Problem (TSP): traverse the MST.

(c) \( \log(n) \)-approximation for Set Cover: choose greedily.

(d) (in recitation) 2-approximation algs for vertex cover and min-weight vertex cover.

16. Parallel Algorithms

(a) Sync/Spawn code constructs

(b) Work \( T_1 \)= total computation done. Critical Path \( T_\infty \)= length of longest computation path.

(c) Greedy scheduling on \( P \) processors give computation time \( \leq T_1/P + T_\infty \)

(d) Example: Parallel merge sort. Parallel-merge subroutine has \( PM_\infty = \Theta(\log^2 n) \), and \( PM_1 = \Theta(n) \). Using that in parallel-merge-sort achieves \( T_\infty = \Theta(\log^3 n) \) and \( T_1 = \Theta(n \log n) \).

17. Cryptography

(a) Diffie-Hellman key exchange

(b) RSA