Design and Analysis of Algorithms

Wenjie Li

October 24, 2019

1 Lec 1

1.1 Correctness

1. Total correctness and partial correctness
2. $3x + 1$ problem
3. Fibonacci number algorithms
4. Pay attention to the **-1** index when writing the Fibonacci algorithm

   Fibonacci(n)
   $x = 1$ and $y = 0$
   for $i = 1$ to n **do**:
     $y = x + y$;
     $x = y - x$;
   end for
   print y
5. Multiplication algorithm
6. The correctness of a program = The correctness of an algorithm

2 Lec 2

2.1 Complexity

1. Size of input: Number n, graph G
2. Complexity of the array method for computing Fibonacci number F(n)
3. Is the Fibonacci computing method polynomial time?
4. Can we do better than the time in (3)?

2.2 Divide and conquer

1. Algorithm and its complexity of finding (min, max) in an array
2. Mergesort. Algorithm and complexity
3. Master Thm.
4. Compare $T_1(n) = 3T_1(\frac{n}{2})$, $T_2(n) = 5T_2(\frac{n}{3})$
5. $T(n) \leq T(\frac{4}{5}n) + T(\frac{1}{5}n) + O(n)$

6. Divide and conquer for multiplication. Algorithm and its complexity.
7. Strassen's matrix multiplication. Complexity
8. Fibonacci number. Complexity. Repeated squaring.
9. Find the skyline algorithm
10. Find a closest pair of points among n points P in the plane
11. Definition (Steps) of divide and conquer

3 Lec 3

3.1 Graphs

1. Given a graph G, what are its order, its size and the input size
2. Lower bound for sorting n numbers.
3. BFS, DFS and how they choose frontier edge.
4. Shortest path for undirected graph. How to define the $N^i(s)$ recursively.

3.2 Euler graph

1. Definition of (u, v) walk, a closed walk.
2. Definition of (u, v) trail, a closed trail
3. Definition of (u, v) path, and cycle
4. Definition of Euler tour
5. State Euler Theorem in digraphs and undirected graphs.
6. Algorithm for merging closed trails.
7. Algorithm for avoiding bridges
8. Correctness for avoiding bridges

4 Lec 4

4.1 Time Stamping

1. Time stamping for DFS, algorithm
2. DFS forest. Ancestor and Descendent, Parent and Child.
3. Edge types in digraphs, in undirected graphs
4. How to determine edge types? The ordering of pre/post numbers.$(_u(_v)_v)_u$, $(_v(_u)_u)_v$, $(_v(_v)_u)_u$

4.2 DAG

1. Directed acyclic graph(dag). How to determine if a graph is a dag
2. Topological sort in dags. Definition.
3. Algorithms for topological sort: 1. Repeatedly delete vertices 2. sort vertices w.r.t. post number. Time complexity for two algorithms.

4.3 Components in a graph

1. Cut vertex and Bridge. Definition
2. An edge is a bridge iff
3. Time complexity for finding cut vertices and bridges
4. In a dfs tree T, (1) a root is a cut vertex iff . (2) A non-root vertex is a cut vertex of G iff
5. Define $\alpha(v)$ to be the pre number of the oldest ancestor of v that can be reached through a back edge from a descendant of v.
6. A non root vertex v is a cut vertex iff v has a child v' such that $\alpha(v') \geq pre(v)$
7. Definition of Block
8. Definition of Strongly connected component(scc)
9. Meta-graph
10. Sink scc and how to find sink scc
11. Highest post number
12. Reverse graph time complexity
13. Decomposition algorithm.

5 Lec 5

5.1 MST

1. Definition of MST
2. Easy deletion algorithm to look for MST
3. Definition of a cut
4. Greedy MST algorithm. Blue rule and red rule.
5. Correctness of Greedy MST, how to prove (1) All edges will be colored (2) There is a MST with all blue edges but no red edges.
6. Kruskal's algorithm, Prim's algorithm
7. Boruvka's algorithm and the condition to make it correct.
8. Improving a spanning tree.
9. Implementation of Kruskal's algorithm
10. Time complexity of Kruskal's algolrithm

6 Lec 6

6.1 Union and find

1. Definition of rank
2. Union by rank
3. Path compression
4. The rank of a root increases only when
5. A node of rank k has at least 2^k descendants, and therefore the maximum rank is $logn$ which implies that Find(x) takes at most $O(logn)$ time.

6. Time complexity of a sequence of m Union, Find, and Make-Set operations (n of which are Make-Set operations in time $O(m\alpha(m,n))$
7. Ackermann function definition
8. Application: Connectedness queries in dynamic graphs

6.2 Greedy Algorithms

1. Activity selection problem and the algorithm
2. For the activity selection problem, possible selection criteria are to choose an activity such that it
   (1) The maximum number of compatible activities
   (2) The minimum amount of time for the activity
   (3) The minimum start time
   (4) **The minimum finish time = latest start time**
3. Correctness of activity selection algorithm
4. General scheme of greedy algorithms

6.3 Huffman code

1. Prefix code
2. Huffman encoding algorithm
3. Time complexity for Huffman encoding. Data structure:
4. Correctness of Huffman encoding

7 Lec 7

7.1 Shortest path

1. Dijkstra's algorithm and its limitation
2. Correctness of Dijkstra's algorithm (a) At any time, $d(v) = \delta(v)$ for every vertex $v \in S$ (b) At any time, $\delta(x)$ equals the length of a shortest (s, x)-path whose internal vertices all reside inside S
3. Implementation of Dijkstra's algorithm and time complexity
4. Negative cycle
5. Bellman-Ford algorithm. Complexity
6. Shortest paths in DAG. Complexity
7. Applications (1). Currency trading
   (2). Nested boxes
   (3). System of difference constraints.

8 Lec 8

8.1 Dynamic Programming

1. Dynamic programming is a clever use of doing search, better than recurrence
2. Every DP algorithm has an underlying DAG

3. Shortest paths in a DAG. Let $v_1, v_2, ..., v_n$ be a topological sort of the vertices of G, and assume $s = v_1$. Then every edge of G goes from left to right. For vertex v_i, let $d(v_i)$ denote the distance from s to v_i. Then

$$d(v_i) = mind(v_j) + w(v_i v_j) : v_j v_i \in E$$

We can compute distances $d(v_i)$ from s to all vertices by a single pass from left to right.

4. Longest increasing subsequence

5. Subset Sum. Find a subset S' from a set S of n positive integers such that the sum of integers in S' exactly equals a given integer N

6. Time complexity of Subset sum, polynomial?

7. Key steps in designing a DP algorithm

8.2 Application of Dynamic Programming

1. Shortest paths with $\leq k$ edges.
2. Weighted interval scheduling.
3. Minimum weight vertex cover in trees. How to define $\alpha(T)$ and $\beta(T)$

9 Lec 9

9.1 NP-complete

1. The theory of NP-completeness provide very strong evidence that:
2. Polynomial reduction or polynomial transformation.
3. P = NP
4. Decision problem consists of two parts.
5. An instance of a problem
6. A problem is a decision problem if its solution to any instance is either yes or no
7. A decision problem if its solution to any instance is either yes or no, and formally, a decision problem Π consists of a set D_π of instance and a subset $Y_\pi \subset D_\pi$ of yes-instances
8. Optimization and search versions for a problem
9. Encoding scheme of input
10. Class P = { Π: decision problem Π which admits a polynomial-time algorithm }
11. A polynomial reduction from Π_1 to Π_2

9.2 Examples

1. Reduction from Vertex Cover to Boolean Formula. Definition of Hamming weight
2. Reduction from Hamiltonian Cycle(HC) to Hamiltonian Path(HP)
3. Class NP = { Π: Decision problem Π admits a polynomial time verification algorithm. }
4. A verification algorithm is used to verify a "proof" of a yes-instance I.
5. A verification algorithm for a problem Π has two arguments. The ordinary input I and a string $c(I)$, called a certificate
6. Perfect Matching Problem. Tutte's theorem
7. Class NPC: $\{\Pi : \Pi \in$ NP, and for every problem $\Pi' \in$ NP, $\Pi' \leq_P \Pi$ }

8. Problems in NPC are NP-Complete, and are intractable.
9. NP-complete problems. P = NP
10. NP-hardness

10 Lec 10

10.1 Basic NP-complete problems

1. 3-SAT
2. Vertex Cover of size **at most** k (VC)
3. Clique of size **at least** k
4. How to reduce 3SAT to Vertex Cover
5. Hamiltonian Cycle (HC)
6. Exact Cover By 3-Sets (X3C).
7. Partition
8. Other related NP-Complete problems.
   (1) Independent set of size **at least** k
   (2) Hamiltonian Path
   (3) 3-D matching problem

10.2 Proving NP-completeness

1. Local Replacement from Vertex Cover to Dominating Set
2. Independent Set and Clique problem and Vertex Cover
3. DAG Deletion
4. Feedback Vertex Set
5. 3-Satisfiability
6. Partition into triangles
7. Subset to Vertex Cover
8. Restrictions
   1. Longest Path
   2. Dense Induced Subgraph
   3. Degree Constrained Spanning Tree
   4. Hitting Set
   5. Multiple Components

11 Lec 11

11.1 Approximation Algorithm

1. Definition of Approximation Ratio
2. 2-Approximation algorithm for Vertex Cover
3. No bounded approximation ratio algorithm of Vertex Cover
4. Travelling Salesman Problem (TSP) .

5. 2-Approximation algorithm for TSP

12 Lec 12

12.1 FPT Algorithm

1. Parameterized complexity
2. k-Vertex Cover can be solved in $O(1.2738^k + kn)$
3. Fixed-parameter tractability
4. $O^*(f(k))$ and $O(f(k)|I|^c$
5. W[k]-hard
6. Bounded Search Tree