CS 231 course note

Chenxuan Wei

May 2022

Contents

1 Fundamentals

1.1 Specificying a problem

1. Problem
   - Input specification: show type of input
   - Output specification: show how this is related to output
2. Recipe
   - Make the problem general
   - From the input
   - From the output
3. instance
   of a problem is a specifica data that satisidies the input specification
4. Solution
   to an instance of a problem satisifies the output specification

1.2 Types of data

1. Grids
   a 2 demetional sequence (start from 0 like a array)

2. Tree Terms

   - leaf: node with no child
   - internal node: node that is not a leaf
   - siblings: nodes with the same parent
   - subtree: atree within a tree consisting oa a node and all it's descendants

3. Graph
   a tree without a root

4. Data for degining problems with size
   input

   - Numbers: constant
   - String: n = length
   - Sets: n = number of element in set
   - Sequences: n = length
   - Grids: r = number of row, c = number of columns
   - Trees: n = number of nodes
   - Graphs: n = number of vertices: m = number of edges

   Output

   - Ordering of data items
   - Categorization of data items
   - Subset of data items

1.3 Type of problem

5. Optimization Problem
   Constructive: find optimal solution
   evaluation: find optimal value

6. Decision problem
   a problem that answer yes/no to a question

7. Search problem
   find a feasible solution that satisified a consition

8. Counting and enumeration problems
   counting: number of solution that satisified a condition
   emueration problem: all solution satisified a condition

1.4 Paradigms

1. Exhaustive search

    - Sketch
        - Generate all possiblities
        - Extract infomation
        - Determine the solution
    - checklist
        - Definition of set of possiblities
        - process for generating all possibilities or next possibility
        - Definition of information to extract
        - Porcess for extracting information
        - process for forming the solution from all

2 Order notation

2.1 Running time

1. Average case:
   the value of f(k) is the sum over all instance I of size k of the probability of I multiplied by the running time of the algorithm on instance I

2. Best case
   the best possible running time in terms of k

3. Worst case
   the worst possible running time

2.2 Categories

1. notation
   only have 1, logn, n, n^2, 2^n
   simple function: only have one term with no coefficient
   log is base 2
   only look at domenant term

2. Recipe
   use a simple function, remove all constant, only have dominant term

3. Different notation

   - θ: upper/lower bound
   - O: upper bound
   - ω: lower bound

2.3 Order notation

1. formal definition of $O()$
   $f(n)$ is in $O(g(n))$ if there is a real constant $c > 0$ and constant $n_0 \geq 1$ such that
   $f(n) \leq cg(n), \forall n \geq n_0$

2. formal definition for $\Omega()$
   $f(n)$ is in $\omega(g(n))$ if there is a real constant $c > 0$ and constant $n_0 \geq 1$ such that
   $f(n) \geq cg(n), \forall n \geq n_0$

3. formal definition for $\theta()$
   $f(n)$ is in $\omega(g(n))$ if there is a real constant $c_1, c_2 > 0$ and constant $n_0 \geq 1$ such that
   $c_2g(n) \geq f(n) \geq c_1g(n), \forall n \geq n_0$

3 Algorithm analysis

3.1 Pseudocode

1. Grammer

    - Variable capitalized, function all capital
    - Simple python list operation is allowed
    - slice of [a:b] can be used
    - use append(L, 4) for use function
    - reserved words are bold
    - Assignment use $< -$
    - "for each .. in" for for loop

3.2 Analysis

1. Constant time operation
    - Assignment
    - use a variable
    - using an arithmetic or boolean operation or a comparison
    - Moving to another line in the program
    - Returning a value using return
2. Recipe for worst-case running time
    - Break each block into blocks
    - Determine a bound on each block individually
    - Retain all dominant costs
    - Use θ if all costs are expressed in θ or use O

3.3 Exhaustive search

1. Algorithm

    - Identify an exhausitive search
    - Create a ES algorithm
    - Analyze the run time
    - Implement it

4 Greedy approach

4.1 Greedy approach

1. Paradigm: greedy algorithms

   (a) Sketch

   - Process data
   - Loop to build up the solution step by step
     Use information to make decision
     make updates to information

   (b) Checklist

   - Process for preprocessing data
   - Definition of steps needed to bulid a solution
   - Definition of information to be used for a decision
   - Definition of criteria for a decision
   - Process for making updates to information

4.2 Paths

1. Dijkstra's algorithm

    - Input
        - K: the set of vertices such that cheapest paths from s are known
        - U: the set of vertices not in K
        - special(v) for all $v \in U$, the cheapest cost of a special path from s to v using only vertices in K
    - Process
        - find the chpeast udge in cut(K): the set of edge such a end is in K, another is in U
        - add to K, repeat

4.3 MST

1. Spanning tree
   a tree connecting all vertices in a connected graph

2. MST
   Input: a connected graph G with positive edge weights
   Output: a subset A of E(G) that forms a tree all of V(G)

3. Optimal substructure Property
   a problem satisified this when a optimal solution to an instance of a problem can be formed from optimal solution of a smaller instance

4. Greedy choice property:
   a greedy algorithm satisfies this property when a optimal solution consistent with each greedy choice made

5 Divide and conquer

5.1 Twenty questions

1. Recursive
   at least a base case and a recursive case

5.2 Design

1. Paradiam divide-and-conquer:

    - Sketch
        - Solve base case directly
        - Deivide into smaller instances
        - Conquer recursive cases using recursive calls
        - combine results on smaller instances
    - checklist
        - Definition of smaller instances
        - definition of base cases
        - Process for dividing the instance

5.3 Interation method

1. How to use it
   First, converge T(n) = T(n-k) + C
   then find a k that can change T(n-k) to base case
   sub k into first equation
   then only n let in RHS

5.4 Master method

1. Master method
   in form of $T(n) = aT(n//b) + f(n)$
   let x = $n^{log_b a}$
   compare f(n) and x
   use which one is bigger, if equal, $T(n) \in \theta(xlogn)$

5.5 Substitution

1. Guess a upper boudn for T(n)
2. subsiteu guess T on smaller values
3. simplify the right hand side to prove the bound
4. check that the bound holds for the base cases

6 Dyanmic programming

6.1 Matrix-chain multiplication

1. Definition of Problem
   Input: a sequence of interger d_i
   Output: A parenthesization of the matrices M_0 to minimize the number of multplication

6.2 Dynamic programming

1. Sketch

   - Create a table
   - Loop over table entries
     fill in base casese
     fill in non-base cases
   - get solution

2. Checklist

   - Definition of solution interm of solutions to smaller instances
   - Definition of information to store in each table entry
   - Definition of basecases and there values
   - Definition of the shape of the table or tables need to store the solution to the smaller instances
   - Process for get the solution from table

6.3 Matrix-chain revisted

1. Optimal substructure recipe

   - Using the optimal solution O for an arbitrary instance I, construc one or more smaller instances
   - Decompose O into pieces
   - Show that if any piece O' is not an optimal solution for a smaller instance I'

2. shape of the grid

   - triangle may occur when $i < j$ or $M[i, j] = M[j, i]$
   - we may use smaller demension if the entries depend on only a few rows

3. Order of evaluation
   check the formula, if there is a i+ or j+, we can't use increate in value of that

6.4 Longest common subsequence

1. The question
   Input Sequence X and Y
   Output: A sequence Z that is a subsequence of both X and Y and maximum length

7 Hardness of problems

7.1 Complexity

1. Upper and lower bound
   upper bound of $O(f(n))$ on a problem means there is an algorithm correctly solving the problem that worst case running thime is $O(f(n))$
   lower bound $\omega(g(n))$ on a problem means any problem that correctly solve the problem must use $\omega(g(n))$ as worst case

2. Polynomial size
   $\sum c_i n^i$

3. Complexity class P
   P conain decision problems only
   it incluse all **Polynimial-time algorightm**
   if a problem is tractable if it is in P

7.2 Decition Trees

1. Key operation
   is a type of step that can be used to represent the other types of steps

2. Comparison-based algorithm
   is when key operation is a comparison step

3. Decision trees
   A Model of computation for comparision-based algorithms

4. Information theory lower bound/decision tree lower bound
   A tree of height H have at most 2^h leavs
   if a problem have l possible outputs, then any comparison=based algotithm have worst case $\omega(logl)$

7.3 Twenty question

1. adversary
   A way to get the worst case input
   answer questions to keep the set of possibilities as big as possible

2. Strategy
   a procedure that produces an answer to each question by the algorithm

   - the answer is consistent with previously-given answers,
   - there is at least one input consistent with all the answers given,
   - there are no limits on time to compute an answer, and
   - there are no limits on amount of extra information to store

   but there is no knowledge of the algorithm or future question

3. Recipe

   (a) Specify an adversary strategy
   (b) Determine a number of steps T that any correct algorithm must take
   (c) Show that after T-1 steps of any algorithm, there will be at least two inputs consistent with the answers given by the adversary, and that they yield different outputs.

7.4 Reductions

1. Definition
   Problem A can be reduce to b if we can solve A using a algorithm for B at most a polynomial number of time and at most polynomial extra time
   We say A is reducible to B

2. Equivalent
   if they are both reduceble to each other, then they are equivalent

7.5 NP

1. polynomial-time verification algorithm for decision problem
   is a polynomial-time algorithm that has 2 input
   a instance of the problem and extra information
   produce yes if the input is yes-instance

2. NP
   is a class of decision porblem that can be verified in polynomial time

3. Recipe for membership in NP

   - Give a certificate for each yes-instance and show that its size is at most polynomial.
   - Give a verification algorithm and show that its worst-case running time is at most polynomial.
   - Show that the algorithm answers "Yes" for any yes-instance and its certificate.
   - Show that the algorithm is not fooled by false certificates for any no-instances.

7.6 NP-hardness

1. NP hard
   if every X $\in$ NP is reducible to Y, then Y is NP-hard

2. NP complete
   if Y is in Np and it is NP hard, it is complete

3. Cecipe for NP complete

   - Prove Z is in NP.
   - Select Y that is known to be NP-complete.
   - Give an algorithm to compute a function f mapping each instance of Y to an instance of Z (it needn't map to all of Z)
   - Prove that if x is a yes-instance for Y then f(x) is a yes-instance for Z.
   - Prove that if f(x) is a yes-instance for Z then x is a yes-instance for Y.
   - Prove that the algorithm computing f runs in at most polynomial time.

Compromising on time

7.7 Backtracking

1. Partial solution
   A partial solution represent a set of all solution that consistent a smiliar part of information
   extending a partical solution: is the process to divide groups to form a larget partial solution if possible

2. Canadidate
   A partial solution is a canadidate if is in the correct form to be a solution or witness

3. Paradigm: Backtracking's sketch

   - Start with the initial partial solution at the root
   - Initialize extra information
   - At the current node:
     - Stop if the partial solution is a candidate
     - Stop if the partical solution cannot be extended
     - Recursively process each child
     - update the extra information
     - update output
   - Backtrack

4. Paradigm: Backtracking's checklist

   - Definition of a partial solution
   - Definition of a partial solution at the root
   - Definition of extra information
   - Process for the root
   - Process for a non-root
   - Process for determining that a partial solution is a candidate
   - Process for determining that a partial solution cannot be extended
   - Process for extending a partial solution
   - Process for updating the extra information
   - Process for determining the output

7.8 Branch-and-bound

1. Paradigm: Branch and bound's sketch
    - Start with the initial partial solution at the root
    - Initialize extra information
    - At the current node:
        - Stop if the partial solution is a candidate
        - Stop if the partial solution cannot be extended
        - Stop if the bound is worse than the best-so-far
        - Recursively process each child in order
        - Update the extra information
        - Update the output
    - Backtrack
2. Checklist
    - Definition of partial solution
    - Definition of partial solution at the root
    - Definition of extra information
    - Definition of a bounding function
    - Process for the root
    - Process for a non-root
    - Process for determining that a partial solution is a candidate
    - Process for determining that a partial solution cannot be extended
    - Process for extending a partial solution
    - Process for updating the extra information
    - Process for determining the output

8 Compromising on correctness

8.1 Approximation

1. Proving the algorithm is not correct

   - Choose a instance x
   - show that x is an instance of the problem the algorithm is supposed to solve
   - show that produce y
   - find a z that z is better than y

2. Approximation algorithm
   is an algorithm for optimization problems with guarrantess a approximate solution will be produces

3. Ratio bound R(n)
   let A be approximate solution, O be optimal solurion, max(A/O, O/A) $\leq$ R(n)

4. Disproving raio bound

   - Choose a instance x
   - show that x is an instance of the problem the algorithm is supposed to slove
   - show max(A/O, O/A) $>$ R(n)

   if ratio bound is $\theta(f(n)$, show max(A/O, O/A) $>$ g(n), $g(n) \notin \theta(f(n))$

5. Complexity class APX
   the set of problem that have constant ratio bound

8.2 Heuristics

1. Hill climibing
   is to start with a feasible solurion and keep make small imporve ments until no improvement can be made

9 Chenging the rules

9.1 Special instances

1. pseudo-polynomial time
   Polynomial time in the largest integer in the input

2. Weakly Np-complete
   if there is a known pseudo-polynomial algorithm that solve the problem

3. Strong NP-complete
   if it can be proved that it can not solved by a pseudo-polynomial time algorithm unless P=NP

9.2 Las Vegas algorithms

1. Algorithm
   Guaranteed to give the correct answer in expected polynomial time

9.3 Monte Carlo algorithms

1. Algorithm
   Guaranteed to give a answer in poly time, with high probability to give a correct answer

9.4 Online

1. offline algorithm
   start conputation after get all inputs

2. online algorithm
   start computation before get all inputs

3. Competitive ratios
   When using online algorithm, we have a ratio c
   if the cost is at most c times the cost of the best offline algorithm for any input

10 More to explore

10.1 Other models of computation