CS 341 – Algorithms

Lecture 1 – Course Introduction

11 May 2021

Today's Plan

1. Course information/administration
2. Course overview
3. Time complexity and computation model
4. 3-SUM ?

Course Information

Course homepage: https://cs.uwaterloo.ca/~lapchi/cs341/.

You can find the course outline, course notes, slides, and homework there.

We have a piazza page for Q&A: https://piazza.com/uwaterloo.ca/spring2021/cs341.

You can also see the course homepage of my past offerings in Winter 2016 and Spring 2017.

Course Requirements

Homework 50%

- 5 problem sets, each 10%
- one programming problem in each homework

Take home midterm 20%

June 28 (Mon) posted 9am , collected 9am June 29

Take home final exam 30%

Please find the tentative schedule in the course outline (e.g. midterm ~June 28, HW1 due ~May 31).

Academic honesty is very important.

Online Lectures

We have online live lectures via Zoom, probably on Wed and Fri 11:10am-12:40pm (TBA).

The lectures will be recorded and posted on YouTube afterwards.

- You are encouraged to show your video so that I could get some visual response.
- You are encouraged to interrupt me to ask questions directly.
- You are also encouraged to ask and answer questions in chat. I will check from time to time.

References

Course notes will be provided and usually posted the day before the lecture.

Slides will be provided and the unannotated version will be posted the day before the lecture,
and the annotated version will be posted after the lecture.

We will mostly use the problems discussed in the following three reference books.

- [DPV] Algorithms, by Dasgupta, Papadimitriou, and Vazirani, McGraw-Hill.
- [KT] Algorithm Design, by Kleinberg and Tardos, Pearson.
- [CLRS] Introduction to Algorithms, by Cormen, Leiserson, Rivest and Stein, MIT Press.

Course Resources and Support

- Lectures (live on Zoom and recorded videos).
- Course notes.
- Slides.
- Homework and supplementary exercises.
- Tutorials (live on Zoom and recorded videos). Worked out sample problems. Mon
- Office hours (live on Zoom): time TBA, after lectures. Wed-Fri 1:15-2:15
- TA office hours (live on Zoom): time TBA. Tue, Thu one on evening
- Piazza Q&A.

Hope these can support and accommodate different styles of learning.

Today’s Plan

1. Course information/administration
2. Course overview
3. Time complexity and computation model
4. 3-SUM

Syllabus

The main focus is on the design and analysis of algorithms. fast algorithms

We will also study the theory of NP-completeness in the end. P vs NP problem

The tentative schedule is:

1. divide and conquer (~4 lectures).
2. graph algorithms (~4 lectures). BFS, DFS
3. greedy algorithms (~3 lectures). MST X
4. dynamic programming (~4 lectures).
5. bipartite matching (~3 lectures). local search (linear programming)
6. NP-completeness (~4 lectures).

Course Style

There are a few steps to develop an efficient algorithm useful in practice. ↙ formulate problem

- Understand the structures and mathematical properties of the problem. e.g. nice recurrence
- Use these observations to design an algorithm. Prove correctness and analyze time complexity.
- Efficient implementation, with the use of good data structures.

induction, contradiction

MATH 239

↗

This course is theoretically oriented.

- ○ We will focus on the first two steps, and spend most of the time in writing **mathematical proofs**.
- ○ Standard data structures will be enough (e.g. queue, stack, heap, balanced search tree, etc).
- ○ The programming problems are to practice your implementation skills.

Two Classical Problems

We are given a graph $G = (V, E)$ with a non-negative cost c_e on each edge.

The **traveling salesman problem** askes us to find a minimum cost tour visiting every vertex at least once.

The **Chinese postman problem** askes us to find a minimum cost tour visiting every edge at least once.

TSP: · naive, by trying all $n!$ orderings $1\ 2\ 3 \ldots n$, $1\ 3\ 2 \ldots n$ $\geq n! \approx n^n$

- dynamic programming ✓ $O(2^n)$ - best known
- NP-complete ✓, Probably not exist polynomial time algo for TSP
- approximation algorithms (466)

Chinese postman problem: $O(n^4)$ poly exact algo

graph matching (bipartite matching ✓)

Learning Outcome

- Know basic techniques and well-known algorithms well.
- Have the skills to design new algorithms for simple problems.
- Have the skills to prove correctness and analyze time complexity of an algorithm.
- Use reductions to solve problems and to prove hardness.

recursion

Today’s Plan

1. Course information/administration
2. Course overview
3. Time complexity and computation model
4. 3-SUM

Time Complexity

Roughly speaking, we count the number of operations that the algorithm requires.

But instead of counting precisely, we use the asymptotic time complexity to analyze algorithms.

Given two functions $f(n)$, $g(n)$, we say

- (upper bound, big-O) $g(n) = O(f(n))$ if $\lim_{n\to\infty} \frac{g(n)}{f(n)} \le c$ for some constant c (independent of n).
- (lower bound, big-Ω) $g(n) = \Omega(f(n))$ if $\lim_{n\to\infty} \frac{g(n)}{f(n)} \ge c$ for some constant c.
- (same order, big-Θ) $g(n) = \Theta(f(n))$ if $\lim_{n\to\infty} \frac{g(n)}{f(n)} = c$ for some constant c.
- (loose upper bound, small-o) $g(n) = o(f(n))$ if $\lim_{n\to\infty} \frac{g(n)}{f(n)} = 0$
- (loose lower bound, small-ω) $g(n) = \omega(f(n))$ if $\lim_{n\to\infty} \frac{g(n)}{f(n)} = \infty$

Examples and Exercises

$100n^2 \quad = O(n^2) \quad , \quad \Theta(n^2)$

$2n^3 + \frac{n^3}{\log n} \quad = O(n^3) \quad , \quad \Theta(n^3)$

$n \log n \quad = o(n^2) \quad , \quad = \omega(n)$

$\frac{n^3}{\log \log n} \quad = o(n^3) \quad = \omega(n^{2.99})$

$2^n \quad = o(n^n)$

$2^{\sqrt{n}}$ vs $n^{\log n}$ $\quad 2^{\sqrt{n}} \quad n = 2^{\log_2 n} \Rightarrow n^{\log_2 n} = 2^{(\log_2 n)^2}$

$\sqrt{n}$ vs $2^{\sqrt{\log n}}$ $\quad n = 2^{\log_2 n} \quad \sqrt{n} = \left(2^{\log_2 n}\right)^{\frac{1}{2}} = 2^{\frac{\log_2 n}{2}}$ vs $2^{\sqrt{\log n}}$

$2^{n^{1.1}}$ vs $n!$ $\quad$ Stirling's approx $\quad n! \approx \sqrt{2\pi n}\left(\frac{n}{e}\right)^n \le n^{n+1} = 2^{\boxed{(n+1)\log_2 n}}$

Worst Case Time Complexity

We say an algorithm has time complexity $O\big(f(n)\big)$ if it requires at most $O\big(f(n)\big)$ primitive operations for **all inputs of size $\boldsymbol{n}$** (e.g. n bits, n numbers, n vertices, etc).

$2^{2^{100}}n^2$ vs n^3?

$O(n^2)$ vs $O(n^3)$ former is faster only if $n \geq 2^{2^{100}}$

- it happens, not often, could be improved.
- when n is large enough, former is faster

Polynomial Time Algorithms

"Good" algorithms: worst case time complexity $O(poly(n))$.

n^3 n^{100}

TSP $O(2^n)$

- better than brute-force

- first step

Computation Model

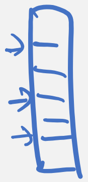
Word-RAM model

- Access an arbitrary position of an array in constant time.
- Each word operation (e.g. addition, multiplication, read/write) can be done in constant time.

e.g. - graph problem on n vertices - use $\log_2 n$ bits to label vertices

assume $\log_2 n$ bits fit into a word $2^{\log_2 n} = n$

- binary search on n elements can be done in $O(\log n)$



Non-trivial example: computing determinant.

$\begin{pmatrix} c \end{pmatrix} \xrightarrow{\text{Gaussian}} \begin{pmatrix} x & & & \\ & x & & \\ & & x & \\ 0 & & & x \end{pmatrix}$

Gaussian elimination $\Theta(n^3)$

$\log_2 n$ bits

intermediate number $\geq n$ bits

bit-complexity

Be aware of what we are assuming!

Today's Plan

1. Course information/administration
2. Course overview
3. Time complexity and computation model
4. 3-SUM

1. TA office hours Tue / Thu
2. Tutorials Mon
3. Video recordings zoom recording
   remove them
   after a week

3-SUM

Input: n numbers $a_1, a_2, \dots, a_n$, and a target number c

Output: indices i, j, k such that $a_i + a_j + a_k = c$, or report that such triples do not exist.

naive: try all triples $O(n^3)$

variants:
- i, j, k distinct
- i, j, k could be equal

3-SUM: Algorithm 2

Observe that $a_i + a_j + a_k = c$ can be rewritten as $c - a_i - a_j = a_k$.

Idea: Enumerate all pairs a_i, a_j and check whether $c - a_i - a_j = a_k$ for some k.

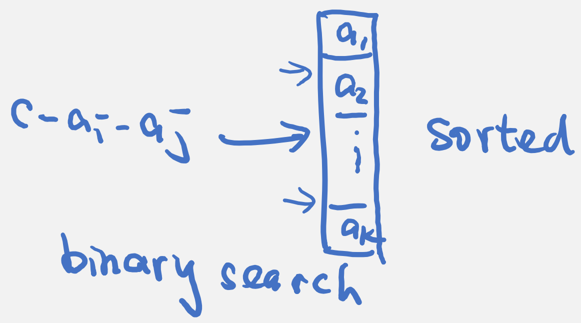
sort $a_1, \dots, a_n$

for $1 \le i \le n$. for $1 \le j \le n$

$\exists k$ s.t. $\underbrace{c - a_i - a_j} = a_k$?

↑ by binary search

idea:



time: $\overset{\text{sorting}}{O(n \log n)} + O(n^2 \underbrace{\log n}_{\text{binary search}}) = O(n^2 \log n)$

3-SUM: Algorithm 3

Note that $a_i + a_j + a_k = c$ can also be rewritten as $a_i + a_j = c - a_k$.

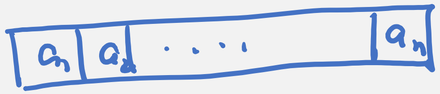
Idea: Enumerate all a_k and check whether $a_i + a_j = c - a_k$ for some pair i, j.

sort $a_1 \ldots\ldots a_n$

for $1 \le k \le n$

$\exists$ i,j. s.t. $a_i + a_j = \widetilde{c - a_k}$?

$b = c - a_k$



$\exists$ i,j s.t $a_i + a_j = b$?

if we can solve 2-SUM in $O(n)$ time.

then 3-SUM can be solved in $O(n^2)$ time.

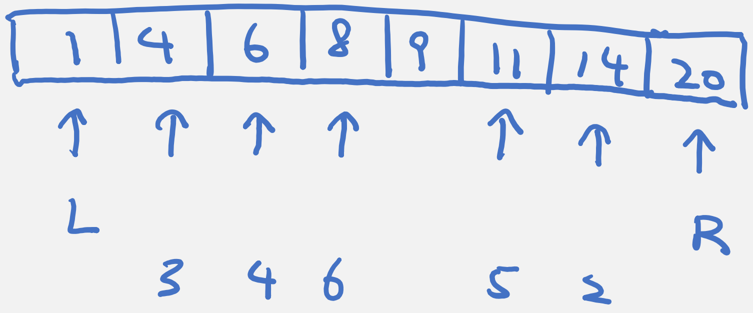
We reduce the 3-SUM problem to n instances of the 2-SUM problem.

2-SUM: Algorithm

Input: n numbers $a_1 \le a_2 \le \cdots \le a_n$, and a target number b.

Output: indices i, j such that $a_i + a_j = b$, or report that such pairs do not exist.

$b=19$



```
initially $L=1$, $R=n$
while $L \le R$
                                          return $L, R$.
                                            ↑
        if $a_L+a_R=b$, then return "YES"
        else if $a_L+a_R>b$, then decrease $R$ by 1
        else if $a_L+a_R<b$, then increase $L$ by 1.
return "NO"
```

Time complexity: each iteration $R-L$ decreases by 1

$\Rightarrow$ at most n iterations $\Rightarrow$ $O(n)$ time

2-SUM: Proof of Correctness

10 marks

$O(n^3)$ ≤ 1 mark

$O(n^2 \log n)$ ≥ 7 marks

$O(n^2)$ algo but without proofs ≥ 7 marks

$O(n^2)$ correct proof, 10 marks.

If this question was in the exam...

- If no such pairs exist, we won't find them
- If such a pair i, j exists, then the algo will find it

⇒ by contradiction, suppose i, j exists but $\underbrace{\text{the algo missed it}}_{L > R}$.

$\exists$ a time s.t. $L = i$ or $R = j$

Consider the first time that it happens.

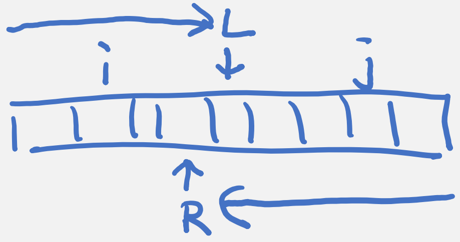
WLOG assume $L = i$

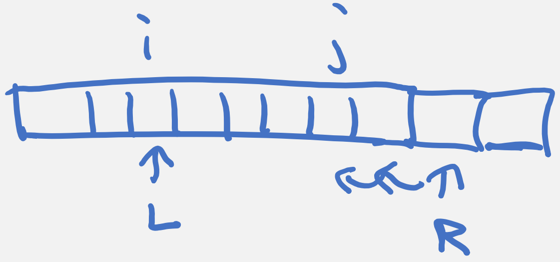
$\Rightarrow R > j$

at that iteration $a_L + a_R = a_i + a_R > a_i + a_j = b$

⇒ the algo would decrease R by 1

⇒ R would move to j $\square$





Literature (Optional)

Conjecture: $O(n^2)$ algorithm is optimal for 3-SUM.

Researchers started to use this conjecture to prove hardness for other problems!

if problem A can be solved in $O(n^{1.99})$ time

then 3-SUM can be solved in $O(n^{1.99})$ time as well

but this is impossible, so problem A cannot be solved in $O(n^{1.99})$ time

$O(n^2/(\log n/\log\log n)^{\frac{2}{3}})$ for 3SUM, so $o(n^2)$.

open: $O(n^{1.99})$ is possible?