Math 409: Discrete Optimization

Today: Course info & intro to complexity

"Discrete optimization" = finding best solution from some finite (but usually large) set

Ex: min spanning tree, shortest path, max flow
matchings, knapsack, integer programs

§1.1 Algorithms and complexity

FIND DUPLICATES

Input: list $a_1, \dots, a_n \in \mathbb{Z}$

Goal: decide whether some number is repeated

e.g. $5,1,2,4 \rightsquigarrow$ "no" $\quad 5,1,2,1 \rightsquigarrow$ "yes"

Algorithm:

```
for i=1 to n do                          ← "pseudo code"
    for j=i+1 to n do
        if a_i = a_j then return "yes"
return "no"
```

The running time of an algorithm is the number of elementary operations (additions, subtractions, multiplications, comparisons) used.
Usually only care about this up to constant factors

In example: $\binom{n}{2} = \frac{n(n-1)}{2}$ comparisons
(+ for loops + additions $i \to i+1$ ...)

There is a constant $C>0$ so that the algorithm above finishes after at most Cn^2 operations
↳ The algorithm above takes time $O(n^2)$.

Def: For functions f, g from $\mathbb{N} \to \mathbb{R}_{\geq 0}$, we say $f(n) = O(g(n))$ if there is some constant $C>0$ and $n_0 \in \mathbb{N}$ s.t. $f(n) \leq C \cdot g(n)$ for all $n \geq n_0$.

ex. $f(n)$ = max #operations taken by algorithm above to finish on any input $a_1, .., a_n \in \mathbb{N}$
$g(n) = n^2$

Note: there are faster algorithms for this!

An algorithm has polynomial running time if, for some d, it runs in time $O(n^d)$ on every input of length n.

Polynomial time $\approx$ "efficient"

Exponential (or more) time $\approx$ "inefficient"

Subtlety: it only makes sense to say that elementary operations have running time $O(1)$ when the input size of the numbers is $O(1)$ (See notes for more.)

Complexity theory

Complexity class P = problems that have a polynomial time algorithm

Complexity class NP: problems that have a non-deterministic polynomial time algorithm

That is, given a solution, one can verify that it is a solution in polynomial time (in input length)

PARTITION Input: $a_1, \ldots, a_n \in \mathbb{N}$

Goal: Find partition $I \uplus J = \{1, \ldots, n\}$ s.t. $\sum_{i \in I} a_i = \sum_{j \in J} a_j$

e.g. $(a_1, a_2, a_3, a_4, a_5) = (4, 5, 2, 8, 7)$

↳ $\{1,3,5\} \uplus \{2,4\}$ $\quad 4+2+7 = 13 = 5+8$

This problem is in NP. Given $I, J \subseteq \{1,\dots,n\}$ one can check in poly. time that $I \uplus J = \{1,\dots,n\}$ and $\sum_{i \in I} a_i = \sum_{j \in J} a_j$

Big open question: Does P=NP?

The "hardest" problems in NP are in

<u>Complexity class NP complete</u>: problems p s.t. a poly. time alg. for p would give a poly. time alg. for any problem in NP.