1 July

<u>Lies from yesterday's lecture</u>

1. $[f(n) = O(g(n))$ and $g(n) \neq O(f(n))]$ does <u>NOT</u> imply $f(n) = o(g(n))$.

   — However, the converse <u>is</u> true.

2. I omitted a crucial word in the definition of P. The correct definition will be given and explained today:

   <u>Defn.</u> P is the class of <u>decision</u> problems for which there exists a polynomial-time algorithm.

1 July

An optimization problem is three problems [P&S §15.2]

An instance of an optimization problem consists of a set F of feasible solutions and a function $c: F \to \mathbb{R}$ giving the objective value for each feasible solution.

The set F is generally very large, so it is not given explicitly. Instead, a given optimization problem has an associated algorithm $\mathcal{A}_F$ that, given a combinatorial object f and a set of parameters S, determines whether f is an element of F, the set of feasible solutions specified by the parameters in S.

- For example, the parameters in S for a max LP in canonical form are the coefficient matrix A and the right-hand side b, and the algorithm $\mathcal{A}_F$ determines whether a solution x is feasible by testing whether $x \geq 0$ and $Ax \leq b$.

- Note that the same $\mathcal{A}_F$ works for all instances of the problem (the "max LP in canonical form" problem). Different instances just have different parameters S for this same algorithm $\mathcal{A}_F$.

Similarly, the objective function $c: F \to \mathbb{R}$ is generally not given explicitly as a set of pairs of the form $(f, c(f))$ for $f \in F$. Instead, a given optimization problem has an associated algorithm $\mathcal{A}_c$ that, given a feasible solution f and another set of parameters Q, returns the value of $c(f)$.

- For example, for the "max LP in canonical form" problem, the parameters Q are the coefficients c of the objective function, and the algorithm $\mathcal{A}_c$ computes the objective value of a feasible solution x by calculating $c^T x$.

  - Again, the <u>same</u> algorithm $\mathcal{A}_c$ works for all instances of the problem. Different instances just have different parameters Q.

So, for a given optimization, an instance can be specified by a representation of the sets of parameters S and Q.

<u>Note:</u> A frequent and pragmatic assumption is that $\mathcal{A}_F$ and $\mathcal{A}_c$ are polynomial-time algorithms. This is usually a reasonable assumption — for most common problems, testing a solution for feasibility and evaluating the objective function are easy to do.

1 July

Now, a combinatorial optimization problem really has three associated problems:

- Optimization version:

  Given S and Q, find an optimal feasible solution.

- Evaluation version:

  Given S and Q, find the optimal objective value.

- Decision (recognition) version:

  Given S and Q and a value $L \in \mathbb{R}$, does there exist a feasible solution with objective value $\geq L$?

  (This is the question for a maximization problem. For a minimization problem, the question should have "$\leq L$.")

Note that the decision version is a yes-no problem. In general:

Defn. A decision (or recognition) problem is a problem in which the answer for every instance is either "yes" or "no."

Example. Traveling salesman problem.

Optimization version: Given n cities and the cost of each arc (i,j), find a cheapest tour through all cities.

Evaluation version: Given n cities and the cost of each arc (i,j), find the cheapest cost of a tour through all cities.

Decision version: Given n cities and the cost of each arc (i,j), and a value $L \in \mathbb{R}$, does there exist a tour through all cities having cost $\leq L$?

Example. Max clique.

Optimization version: Given a graph $G=(V,E)$, find a clique of maximum cardinality.

Evaluation version: Given a graph $G=(V,E)$, find the greatest number of vertices in a clique.

Decision version: Given a graph $G=(V,E)$ and an integer k, does G contain a clique with at least k vertices?

1 July

Question: If we can solve one of these three versions of the problem, can we solve the others?

- Optimization $\longrightarrow$ evaluation: Easy. If we can solve the optimization version to find an optimal feasible solution, then we can solve the evaluation version by just evaluating the objective value of this optimal solution.

- Evaluation $\longrightarrow$ decision: Also easy. If we can solve the evaluation version to find the optimal objective value z^*, then we can solve the decision version by just comparing z^* to L.

- Decision $\longrightarrow$ evaluation: Suppose that we have a way to solve the decision version.

  - Assumption: The optimal objective value is an integer, and for a given instance x all possible objective values are contained in an interval $[\underline{z}, \overline{z}]$ for which $\underline{z}$ and $\overline{z}$ are easily computable from x and such that $\overline{z} - \underline{z} \leq 2^{p(|x|)}$ for some polynomial $p(n)$.

  - This is a reasonable assumption for many problems. For instance, the shortest path problem: if all arc

weights are integers, then the optimal objective value will be an integer. If arc weights are rational numbers, they can be multiplied by their least common denominator to get integer arc weights. We can take
$\underline{z}$ = (sum of all negative arc weights),
$\overline{z}$ = (sum of all positive arc weights),
and it can be shown that $\overline{z} - \underline{z} \leq 2^{p(|x|)}$ for some polynomial $p(n)$, as long as $|x|$ also counts the number of bits used to represent the arc weights.

Under this assumption, we can solve the evaluation version by solving polynomially many instances of the decision version through a technique called <u>binary search</u>. (Next page.)

- <u>Evaluation $\longrightarrow$ optimization</u>: Sometimes. For example, in the shortest path problem, we can solve the evaluation version n times to label each node with its shortest distance from s, and then use these labels to identify admissible arcs. But no general technique is known that works for all problems.

1 July

<u>Binary search</u> for solving the evaluation version of a maximization problem using a solver for the decision version.

1. Initialize $m := \lfloor \underline{z} \rfloor$, $M := \lceil \overline{z} \rceil$.
   - Comment: We will maintain the invariant that the optimal objective value is in the interval $[m, M]$.

2. If $M < m$, return INFEASIBLE.
   If $M = m$, return m. $\leftarrow$ Oops, actually need to solve decision problem one more time to determine whether a feasible solution exists.

3. Let $L := \left\lceil \frac{m+M}{2} \right\rceil$. (Approximate midpoint of $[m, M]$.)
   Use the solver for the decision version to determine whether there exists a feasible solution with objective value $\geq L$.
   - If so, set $m := L$. (Opt.obj.val. is in $[L, M]$.)
   - If not, set $M := L-1$. (Opt.obj.val. is in $[m, L-1]$.)

4. Go to step 2.

<u>Note:</u> For a minimization problem, change step 3:

3. Let $L := \left\lfloor \frac{m+M}{2} \right\rfloor$. (Floor this time, not ceiling.)
   Use the solver for the decision version to determine whether there exists a feas. soln. with obj.val. $\leq L$.
   - If so, set $M := L$. (Opt.obj.val. is in $[m, L]$.)
   - If not, set $m := L+1$. (Opt.obj.val. is in $[L+1, M]$.)

Observe that each time step 3 is executed the interval $[m, M]$ is cut in half (well, ± 1). The search terminates when $M = m$. So the number of times step 3 is executed is $O(\log_2(\overline{z} - \underline{z}))$. By assumption, $\overline{z} - \underline{z} \leq 2^{p(|x|)}$, so $\log_2(\overline{z} - \underline{z}) \leq p(|x|)$, So the number of times the decision version is solved is $O(p(|x|))$, which is polynomial in $|x|$.

1 July

The classes P and NP [P&S §15.3]

Defn. The class P is the class of decision problems for which there exists a polynomial-time algorithm.

Examples. The decision versions of almost all the problems we discussed on and before June 22 are in P:

- Linear programming. (The simplex algorithm is not known to run in polynomial time in the worst case, but there are other algorithms for LP that do run in polynomial time, e.g., the ellipsoid algorithm. See P&S §8.6–8.7 for details.)
- Critical path.
- Transportation problem.
- Shortest path.
- Minimum spanning tree.
- Max flow.
- Bipartite matching.
  - Also matching in general graphs: see P&S §10.4–10.5.

But there are also many important decision problems that are not known to be in P: e.g., the decision versions of IP, SAT, knapsack, integer factorization, TSP, max clique, Hamiltonian circuit, chromatic number, vertex cover, subset sum.

Defn. A verifier for a decision problem A is an algorithm that, given an instance x of A and an additional input called a certificate, will output "yes" only if the answer to x is "yes."

- If the verifier outputs "yes," we say that the verifier has verified or accepted the instance–certificate pair.
- Note that this definition says "only if," not "if and only if." A verifier is allowed to output "no" even if the answer to x is "yes."
- If the output of the verifier is "yes," then we are guaranteed that the answer to x is "yes."

Think of a certificate as a claimed proof that the answer to x is "yes," and think of the verifier as a proof-checker. If the verifier checks the proof and finds that it is valid, then we are guaranteed that the answer to x is "yes." But the verifier may also reject the claimed proof as invalid; that doesn't necessarily mean that the answer to x is "no" — it might just be a flawed proof.

1 July

<u>Example.</u> COMPOSITENESS.
(Common convention is to write the names of decision problems in all capital letters.)

<u>Instance</u>: A positive integer n.

Question: Is n composite?

Certificate: Two positive integers p and q.

Verifier: Outputs "yes" iff $p \geq 2$, $q \geq 2$, and $pq = n$. Outputs "no" otherwise.

<u>Observe</u>: If the verifier outputs "yes" for an instance-certificate pair, then we are guaranteed that the answer to the instance is "yes" (because the certificate gives two nontrivial factors of n).

But the verifier may output "no" even if the answer to the instance is "yes," if the certificate is an invalid proof. For example, if $n = 15$, the certificate $p = 2$, $q = 7$ will cause the verifier to output "no," because that is an invalid proof of the compositeness of n.

<u>Defn.</u> The class NP is the class of decision problems for which there exists a verifier and, for each instance x whose answer is "yes," there exists a corresponding certificate of polynomial size (in $|x|$) that the verifier will accept in polynomial time (in $|x|$).

<u>Example.</u> COMPOSITENESS is in NP.

- For a "yes" instance n encoded in $O(\log n)$ bits, the size of a corresponding certificate is $O((\log n)^2)$, which is polynomial in the size of the input.

- Exercise: Verification can be done in $O((\log n)^2)$ time.

<u>Example.</u> Every problem in P is also in NP.

Certificate: Nothing.

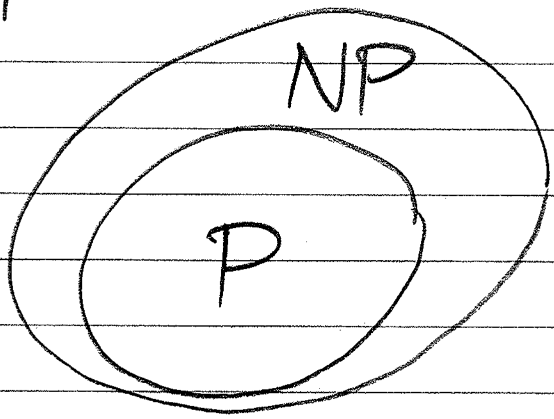
Verifier: Solves the instance (in polynomial time) and outputs the answer.

1 July

Notes.

⚠ — NP does **NOT** mean "not P"!

Every problem in P is also in NP.



(NP actually stands for "nondeterministic polynomial time," because there is an alternative, equivalent definition in terms of a model of computation called "nondeterministic Turing machines.")

— The definition of NP does not require proofs for "no" instances, only for "yes" instances.

— We know that $P \subseteq NP$. But it is <u>unknown</u> whether $P = NP$. This is the "P vs. NP problem," and it is a major open question. Most researchers believe $P \neq NP$, but this has not been proven.