Introduction to Algorithms and Data Structures

Lecture 24: P and NP

Mary Cryan

School of Informatics
University of Edinburgh

"Polynomial-time"

We have seen a large pool of algorithms in this course:

- Insertion sort - worst-case running-time $\Theta(n^2)$
- Mergesort - worst-case running-time $\Theta(n \cdot \lg(n))$
- Breadth-First search - worst-case running time $\Theta(m + n)$ (where n is the number of nodes and m the number of edges)
- Edit distance - worst-case running-time $\Theta(m \cdot n)$
- All-pairs Shortest-Paths - worst-case running-time $\Theta(n^3)$

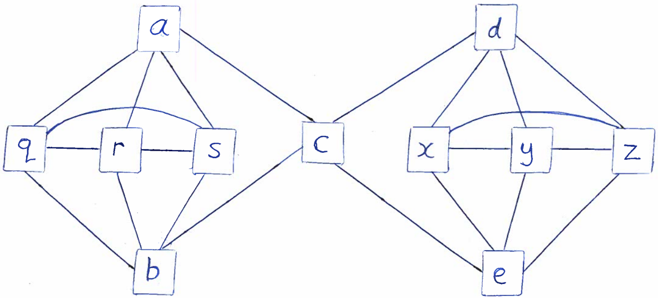
These worst-case running-time (asymptotic) functions all fit into the category of polynomial-time.

> *We say that a computational problem is "polynomial time" if we have a deterministic algorithm A solves the problem (is correct for every instance of the problem) and there is some fixed $r \in \mathbb{R}$ such that for every* instance $\mathcal{I}$, *the algorithm runs in time at most $O(|\mathcal{I}|^r)$.*

Cycles in Graphs

We return to the world of (undirected) graphs $G = (V, E)$.

- An Euler tour (ET) of a given graph is a cycle in the graph which traverses every edge *exactly once* (though may visit vertices more than once).
- A Hamiltonian cycle (HC) of a graph is a simple cycle of the graph which visits every node *exactly once*.



Cycles in Graphs

Consider the problem of testing whether a given graph has an ET/HC.

- For Euler tours, Euler proved (as a generalisation of the Königsberg Bridge Problem) that any connected graph which only has even-degree vertices has an Euler tour.
  - Connectedness can be checked in $\Theta(n+m)$ time (DFS, lecture 15) ...
  - We can check the even-ness of all vertex degrees in $O(m+n)$ time ...
  - So testing for presence of an Euler tour is "polynomial-time" ("in P")
- The Hamilton Cycle problem is believed to be NP-complete - and to have *no* polynomial-time algorithm.

Verifying versus Finding

We think (mainly) about "Decision problems" where we ask questions like

- "Does this graph have an Euler tour?"
- "Does this graph have a Hamiltonian Cycle?"

  (ie, "Does this graph have a simple cycle of length n?")
- "Is the edit distance between these two sequences less than 5"?

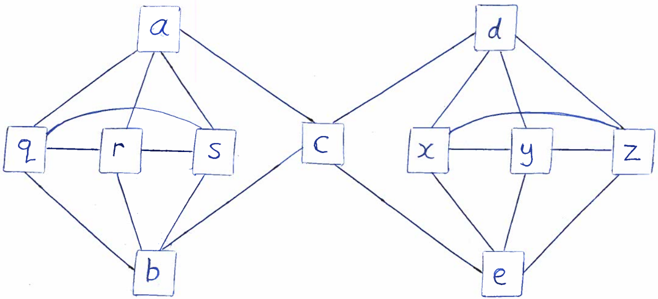
Can often re-cast an optimisation problem as a decision problem ... eg, decision version of edit distance above.

NP: “Guess and Check”

The complexity class NP is essentially a “Guess and Check” model.
It includes any decision problem for which:

- A solution can be written down succinctly in terms of the given input.
- We can check whether a “guessed” solution is correct in polynomial time.
- We do not worry about how we found/guessed the solution (the “magic”)

Guessing and Checking



- Guess a HC, for example: $a, q, r, s, b, c, d, x, y, z, e$.
  - Need to check each pair of vertices have an edge between (yes), plus that $(e, a) \in E$ (no), plus each vertex visited exactly once (yes).
  - Check details proposed HC against graph adjacency list/matrix.
- Similarly could guess and check a proposed ET, for example: $(a, q), (q, b), (b, s), (s, r), (r, q), (q, s), (s, a), (a, r), (r, b), (r, c), (c, d),$ $(d, y), (y, x), (x, z), (z, y), (y, e), (e, x), (x, d), (d, z), (z, e), (e, c), (c, a)$

Polynomial-time problems

All the algorithmic problems we have considered in this course have formal definitions, and inputs in a prescribed form.

- We assume the input is described non-wastefully, ie integers should be represented by *binary numbers*, or maybe *decimal numbers*, but *not* in unary format.
- In fact, even graphs, edges can be described succinctly in binary format.
- Such a sensible representation is called an encoding.

Definition

A computational problem Q is "polynomial time" if there is some fixed $r \in \mathbb{R}$, and some deterministic algorithm A, which returns a correct solution for every instance $\mathcal{I}$ in time at most $O(|\mathcal{I}|^r)$.

Decision problems

Definition

A computational problem Q is a decision problem if it can be described in terms of a collection of potential solutions S, where $Q(\mathcal{I}) = 1$ if there is a solution in S which solves the instance $\mathcal{I}$ and $Q(\mathcal{I}) = 0$ otherwise.

We will often consider decision problems as *languages* (over alphabet $\{0, 1\}$) where $\mathcal{I} \in L_Q \Leftrightarrow Q(\mathcal{I}) = 1$.

Definition

The complexity class P is the class of decision problems Q for which there is a polynomial-time algorithm to compute Q exactly on all input instances.

Informally, we will often include (non-decision) problems (like edit distance, sorting) in the class P.

The complexity class NP

Definition

Consider a decision problem Q wrt its collection of potential solutions S. We say that a two-parameter algorithm A is a verifier for Q iff for all instances $\mathcal{I}$ of Q

$$\text{There is some } y \in S \text{ such that } A(\mathcal{I}, y) = 1 \quad \Leftrightarrow \quad Q(\mathcal{I}) = 1$$

- y is the "guess"
- Sometimes the (successful) solutions of S are called certificates.
- For the Hamilton Cycle problem, the solutions/certificates would be permutations of the vertices of the graph.
- This verifier corresponds to the informal "checker" of our introduction.
- But we say do not say (and will not be drawn ...) on where the guess comes from. That's "magic".

The complexity class NP

Definition

The complexity class NP is the class of decision problems Q (wrt a collection of potential solutions/certificates S) for which there is a verifier $A = A(\mathcal{I}, y)$ which runs in time polynomial in the size $|\mathcal{I}|$ of the instance.

(note this requires that the solution/certificate y is polynomial in $|\mathcal{I}|$ also)

We do not concern ourselves with how the solution/certificate is "guessed".

- This is the power of the model NP, it allows us to capture decision problems which (we believe) have no polynomial-time algorithms to solve them.
- Think about Hamilton Cycle - if we really were "guessing" the solution, we would be considering $n!$ different permutations, which is exponential in the size of the graph.
- For a problem in P (like the Euler Tours problem) the guess can be the empty string.

Reductions between (decision) problems

If I could solve problem Q in polynomial-time, then I would also be able to solve problem R in polynomial-time.

Definition

A problem R can be reduced to the problem Q if there is a polynomial-time computable function $f : \{0,1\}^* \rightarrow \{0,1\}^*$ such that for all instances $\mathcal{I}$ of R

$$R(\mathcal{I}) = 1 \quad \Leftrightarrow \quad Q(f(\mathcal{I})) = 1$$

- Means that R is no harder (at least in the sense of polynomial-time computation) than Q. And that Q is "at least as hard" as R.
- We write $R \leq_{\mathrm{P}} Q$.

NP-completeness

No (NP) problem is any harder than me.

Definition

A decision problem Q is said to be NP-complete if it belongs to the class NP, and it is also the case that for every problem R in NP, $R \leq_{\mathrm{P}} Q$.

Do these even exist?

NP-completeness

No (NP) problem is any harder than me.

Definition

A decision problem Q is said to be NP-complete if it belongs to the class NP, and it is also the case that for every problem R in NP, $R \leq_{\mathrm{P}} Q$.

Do these even exist?

Yes they do! In lecture 24, we will discuss the intrinsic NP-complete problem $\textsc{Sat}$, and show how to reduce other computational problems to $\textsc{Sat}$.

Reading

Reading:

- The Intro to Chapter 34, and Section 34.1 of [CLRS] introduce the concepts of *polynomial-time*, *decision problems*, and the complexity class P.
- Section 34.2 discusses the concept of polynomial-time verifiability and the complexity class NP.
- Section 34.3 introduces the concept of reducibility.

For Lecture 24:

- Section 34.3 covers reducibility and the NP-Completeness of Circuit Satisfiability.