DSC 190

DATA STRUCTURES & ALGORITHMS

Complexity Theory

The quest for efficient algorithms is about finding clever ways to avoid taking exponential time. So far we have seen the most brilliant successes of this quest; now we meet the quest's most embarrassing and persistent failures.

- paraphrased from *Algorithms* by Dasupta, Papadimitriou, Vazirani

Exponential to Polynomial

- Many problems have brute force solutions which take exponential time.
- Example: clustering to maximize separation
- The challenge of algorithm design: find a more efficient solution.

Polynomial Time

- If an algorithm's worst case time complexity is $O(n^k)$ for some k, we say that it runs in **polynomial time**.
  - Example: $\Theta(n \log n)$, since $n \log n = O(n^2)$.

- Polynomial is much faster than exponential for big n.
  - But not necessarily for small n.
  - Example: n^{100} vs 1.0001^n.

- We therefore think of polynomial as "efficient".

Question

- Is every problem solvable in polynomial time?
- **No!** Problem: print all permutations of n numbers.
- **No!** Problem: given $n \times n$ checkerboard and current pieces, determine if red can force a win.

Ok, then...

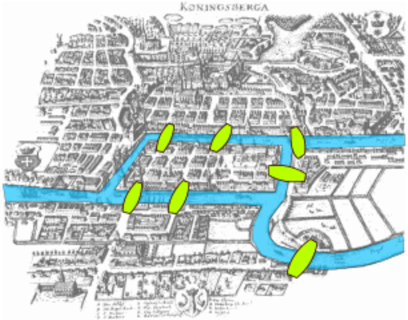
- What problems can be solved in polynomial time?
- What problems can't?
- How can I tell if I have a hard problem?
- Core questions in **computational complexity theory**.

DSC 190

DATA STRUCTURES & ALGORITHMS

Eulerian and Hamiltonian Cycles

Example: Bridges of Königsberg

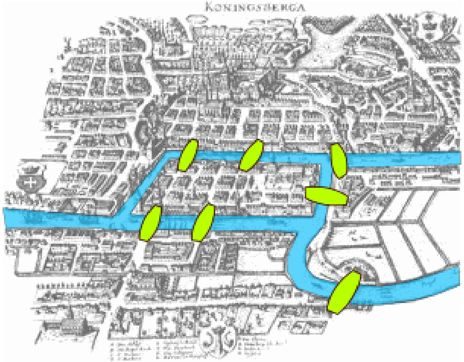


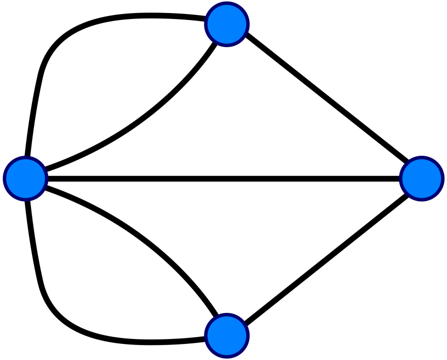
- **Problem**: Is it possible to start and end at same point while crossing each bridge exactly once?

Leonhard Euler

1707 - 1783

Eulerian Cycle

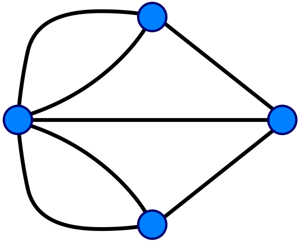




Is there a cycle which uses each edge exactly once?

Necessary conditions

- Graph must be connected.
- Each node must have even degree.
- Answer for Königsberg answer: it is **impossible**.



In General...

- These conditions are **necessary** and **sufficient**.
- A graph has a Eulerian cycle **if and only if**:
  - it is connected;
  - each node has even degree.

Exercise

Can we determine if a graph has an Eulerian cycle in time that is polynomial in the number of nodes?

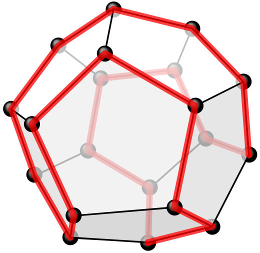
Answer

- We can check if it is connected in $\Theta(V + E)$ time.
- Compute every node's degree in $\Theta(V)$ time with adjacency list.
- Total: $\Theta(V + E) = O(V^2)$. **Yes!**

Gaming in the 19th Century

I have found that some young persons have been much amused by trying a new mathematical game which the Icosian furnishes [...]

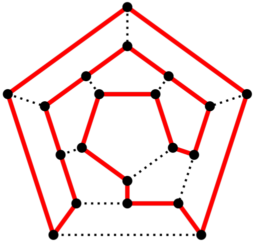
- W.R. Hamilton, 1856



Hamiltonian Cycles

- A **Hamiltonian cycle** is a cycle which visits each *node* exactly once (except the starting node).

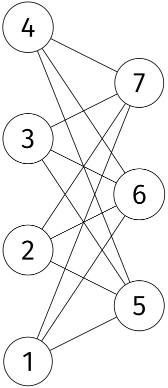
- Game: find a Hamiltonian cycle on the graph below:



Exercise

Can we determine whether a general graph has a Hamiltonian cycle in polynomial time?

Some cases are easy



In General

- Could brute-force.
- How many possible cycles are there?

Hamiltonian Cycles are Difficult

- This is a **very difficult** problem.
- No polynomial algorithm is known for general graphs.
- In special cases, there may be a fast solution. But in general, worst case is hard.

Note

- Determining if a graph has a Hamiltonian cycle is **hard**.
- But if we're given a "hint" (i.e., $(v_1, v_2, \ldots, v_n)$ is possibly a Hamiltonian cycle), we can check it very quickly!
- Hard to solve; but easy to verify "hints".

Similar Problems

- Eulerian: polynomial algorithm, **“easy”**.
- Hamiltonian: no polynomial algorithm known, **“hard”**.

Main Idea

Computer science is littered with pairs of similar problems where one easy and the other very hard.

DSC 190

DATA STRUCTURES & ALGORITHMS

Shortest and Longest Paths

Problem: SHORTPATH

- **Input:** Graph[1] G, source u, dest. v, number k.
- **Problem:** is there a path from u to v of length $\leq k$?
- **Solution:** BFS or Dijkstra/Bellman-Ford in polynomial time.
- **Easy!**

[1]Weighted with no negative cycles, or unweighted.

Problem: LongPath

- **Input:** Graph[2] G, source u, dest. v, number k.
- **Problem:** is there a **simple** path from u to v of length $\geq k$?
- Naïve solution: try all $V!$ path candidates.

[2] Weighted or unweighted.

Long Paths

- There is no known polynomial algorithm for this problem.
- It is a **hard problem**.
- But given a “hint” (a possible long path), we can verify it very quickly!

DSC 190

DATA STRUCTURES & ALGORITHMS

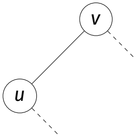
Reductions

Reductions

- HAMILTONIAN and LONGPATH are related.
- We can “convert” HAMILTONIAN into LONGPATH in polynomial time.
- We say that HAMILTONIAN **reduces** to LONGPATH.

Reduction

- Suppose we have an algorithm for LONGPATH.
- We can use it to solve HAMILTONIAN as follows:



- Pick arbitrary node u.
- For each neighbor v of u:
  - Create graph G' by copying G, deleting (u, v)
  - Use algorithm to check if a simple path of length $\geq |V| - 1$ from u to v exists in G'.
  - If yes, then there is a Hamiltonian cycle.

Reductions

- If Problem A reduces[3] to Problem B, it means “we can solve A by solving B”.
- Best possible time for A ≤ best possible time for B + polynomial
- “A is no harder than B”
- “B is at least as hard as A”

[3]We’ll assume reduction takes polynomial time.

Relative Difficulty

- If Problem A reduces to Problem B, we say B is **at least as hard** as A.
- Example: HAMILTONIAN reduces to LONGPATH. LONGPATH is at least as hard as HAMILTONIAN.

DSC 190

Data Structures & Algorithms

$P \stackrel{?}{=} NP$

Decision Problems

- All of today's problems are **decision problems**.
  - Output: yes or no.
  - Example: Does the graph have an Euler cycle?

P

- Some problems have polynomial time algorithms.
  - SHORTPATH, EULER
- The set of decision problems that can be solved in polynomial time is called **P**.
- Example: SHORTPATH and EULER are in P.

NP

- The set of decision problems with “hints” that can be verified in polynomial time is called **NP**.
- All of today’s problems are in NP.
    - All problems in P are also in NP.
- Example: SHORTPATH, EULER, HAMILTONIAN, LONGPATH are all in NP.

P ⊂ NP

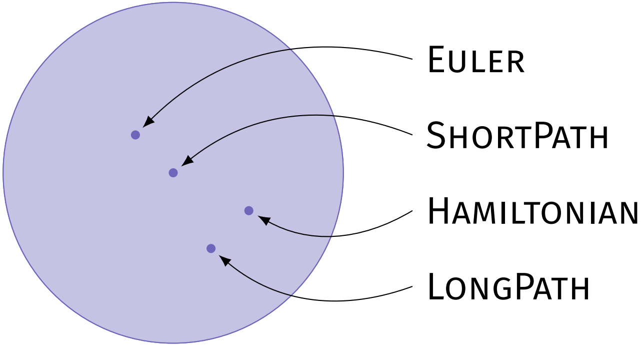
- P is a subset of NP.
- It seems like some problems in NP aren't in P.
  - Example: HAMILTONIAN, LONGPATH.
- We don't know polynomial time algorithms for these problems.
- But that doesn't such an algorithm is impossible!

P = NP?

- Are there problems in NP that aren't in P?
  - That is, is P ≠ NP?
- Or is any problem in NP also in P?
  - That is, is P = NP?

P ≠ NP

P = NP



P = NP?

- Is P = NP?
- **No one knows!**
- Biggest open problem in Math/CS.[4]
- Most think P $\neq$ NP.

[4] If you solve it, you'll be rich and famous.

What if P = NP?

- Possibly Earth-shattering.
  - Almost all cryptography instantly becomes obsolete;
  - Logistical problems solved exactly, quickly;
  - *Mathematicians* become obsolete.

- But maybe not...
  - Proof could be non-constructive.
  - Or, constructive but really inefficient. E.g., $\Theta(n^{10000})$

DSC 190

DATA STRUCTURES & ALGORITHMS

NP-Completeness

Problem: 3-SAT

- Suppose `x_1,...,x_n` are boolean variables (`True`,`False`)
- A **3-clause** is a combination made by `or`-ing and possibly negating three variables:
  - `x_1 or x_5 or (not x_7)`
  - `(not x_1) or (not x_2) or (not x_4)`

Problem: 3-SAT

- **Given:** m clauses over n boolean variables.
- **Problem:** Is there an assignment of `x_1`, …, `x_n` which makes all clauses true simultaneously?
- No polynomial time algorithm is known.
- But it is easy to verify a solution, given a hint.
  - 3-SAT is in NP.

Cook's Theorem

Every problem in NP is polynomial-time reducible to 3-SAT.

- ...including Hamiltonian, long path, etc.
- 3-SAT is at least as hard as every problem in NP.
- "hardest problem in NP"

Cook's Theorem (Corollary)

- If 3-SAT is solvable in polynomial time, then all problems in NP are solvable in polynomial time.
  - ...including Hamiltonian, long path, etc.

NP-Completeness

- We say that a problem is **NP-complete** if:
  - it is in NP;
  - every problem in NP is reducible to it.

- HAMILTONIAN, LONGPATH, 3-SAT are all NP-complete.

- NP-complete problems are the “hardest” in NP.

Equivalence

- In some sense, NP-complete problems are equivalent to one another.
- E.g., a fast algorithm for HAMILTONIAN gives a fast algorithm for 3-SAT, LONGPATH, and all problems in NP.

Who cares?

- Complexity theory is a fascinating piece of science.
- But it’s practically useful, too, for recognizing hard problems when you stumble upon them.

DSC 190

DATA STRUCTURES & ALGORITHMS

Hard Optimization Problems

Hard Optimization problems

- NP-completeness refers to **decision problems**.
- What about optimization problems?
- We can typically state a similar decision problem.
- If that decision problem is hard, then optimization is at least as hard.

Problem: bin packing

- Optimization problem:
  - **Given:** bin size B, n objects of size $\alpha_1, \dots, \alpha_n$..
  - **Problem:** find minimum number of bins k that can contain all n objects.
- Decision problem version:
  - **Given:** bin size B, n objects of size $\alpha_1, \dots, \alpha_n$, integer k.
  - **Problem:** is it possible to pack all n objects into k bins?
- Decision problem is NP-complete, reduces to optimization problem.

Example: traveling salesperson

- Optimization problem:
  - **Given:** set of n cities, distances between each.
  - **Problem:** find shortest Hamiltonian cycle.
- Decision problem:
  - **Given:** set of n cities, distance between each, length ℓ.
  - **Problem:** is there a Hamiltonian cycle of length $\leq \ell$?
- Decision problem is NP-complete, reduces to optimization problem.

NP-complete problems in machine learning

- Many machine learning problems are NP-complete.
- Examples:
  - Finding a linear decision boundary to minimize misclassifications in non-separable regime.
  - Minimizing k-means objective.

So now what?

- Just because a problem is NP-Hard, doesn't mean you should give up.
- Usually, an approximation algorithm is fast, "good enough".
- Some problems are even hard to *approximate*.

Summary

- Not every problem can be solved efficiently.
- Computer scientists are able to categorize these problems.

DSC 190

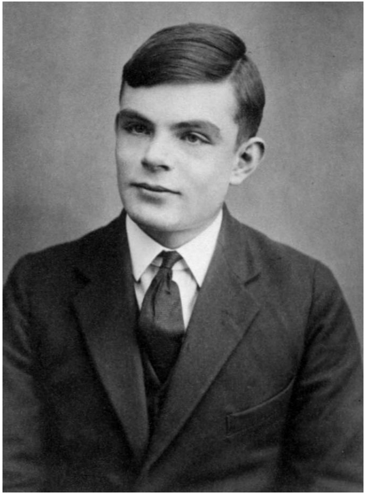
Data Structures & Algorithms

The Halting Problem

Really hard problems

- Some decision problems are harder than others.
- That is, it takes more time to solve them.
- Given enough time, all decision problems can be solved, right?

Alan Turing



1912-1954

Turing's Halting Problem

- **Given:** a function `f` and an input `x`.
- **Problem:** does `f(x)` halt, or run forever?
- Algorithm must work for all functions/inputs!

Turing's Argument

- Turing says: no such algorithm can exist.

- Suppose there is a function `halts(f, x)`:
  - Returns `True` if `f(x)` halts.
  - Returns `False` if `f(x)` loops forever.

Turing's Argument

- Consider `evil_function`.
  - If it halts, it doesn't.
  - If it doesn't halt, it does.

- Contradicts claim that `halt` works.

```
def evil_function(f):
    if halts(f, f):
        # loop forever
    else:
        return
```

Undecidability

- The halting problem is **undecidable**.
- Fact of the universe: there can be no algorithm for solving it which works on all functions/inputs.
- All of these problems are undecidable:
  - Does the program terminate?
  - Does this line of code ever run?
  - Does this function compute what its specification says?

The End