COMP90038
Algorithms and Complexity

Lecture 22: NP-completeness
(with thanks to Harald Søndergaard & Michael Kirley)

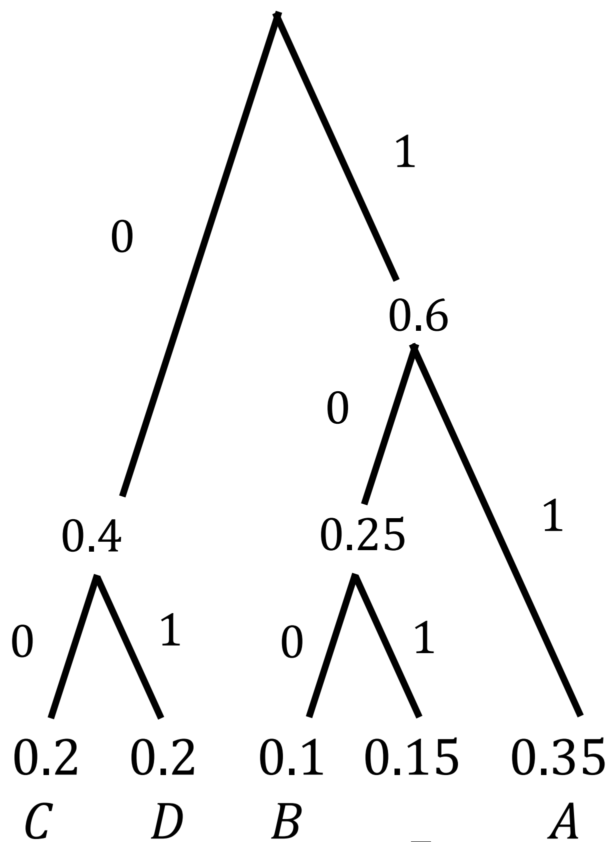
Casey Myers
Casey.Myers@unimelb.edu.au
David Caro Building (Physics) 274

Review from Lecture 21: Huffman Encoding

- Sometimes (for example for common English text) we may know the frequencies of letters fairly well.
- If we don't know about frequencies then we can still count all characters in the given text as a first step.
- But how do we assign codes to the characters once we know their frequencies?
- By repeatedly selecting the two smallest weights and fusing them.
- This is Huffman's algorithm—another example of a greedy method.
- The resulting tree is a Huffman tree.

Review from Lecture 21: Huffman Encoding

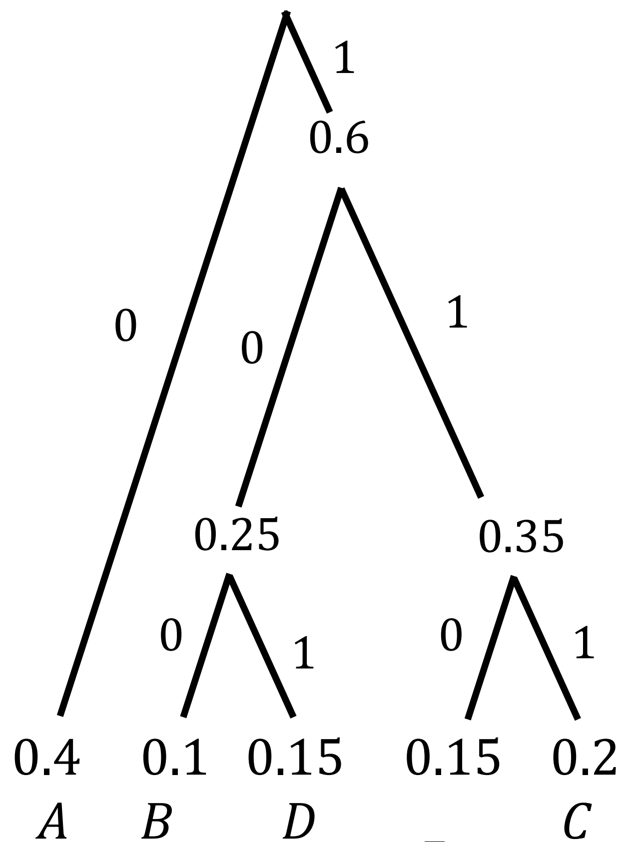
Symbol	A	B	C	D	_
Frequency	0.35	0.1	0.2	0.2	0.15
Codeword	11	100	00	01	101



Review from Lecture 21: Huffman Encoding

Symbol	A	B	C	D	_
Frequency	0.4	0.1	0.2	0.15	0.15
Codeword	0	100	111	101	110

Encode $ABACABAD$:
0100011101000101



Decode 100010111001010:

100	0	101	110	0	101	0
B	A	D	_	A	D	A

Concrete Complexity

- We have been concerned with the analysis of algorithm's running times (best, average, worst cases).

- Our approach has been to give a bound for the asymptotic behaviour of running time, as a function of input size.

- For example, the quicksort algorithm is $O(n^2)$ in the worst case, whereas mergesort is $O(\log n)$.

Abstract Complexity

- Complexity theory instead asks:

  "*What is the inherent difficulty of the problem?*"

- How do we know when we have come up with an algorithm which is optimal (in the asymptotic case).

Difficult Problems

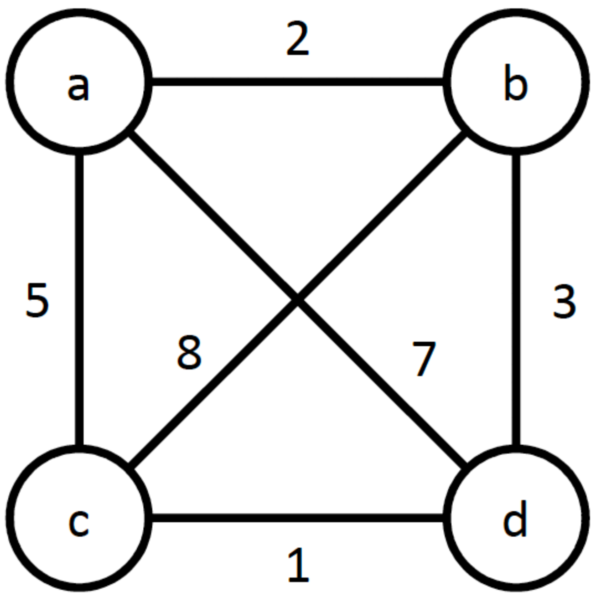
- Which problems are difficult to solve?

- The travelling Salesman problem can be solved through brute force for very small instances

  – One solution is: $a - b - d - c - a$

- However, it becomes very difficult as the number of nodes and connections increase.

  – The solution can be checked to determine if it is a good solution or not.



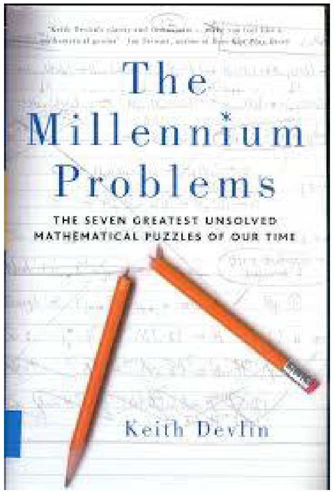
Does P=NP?

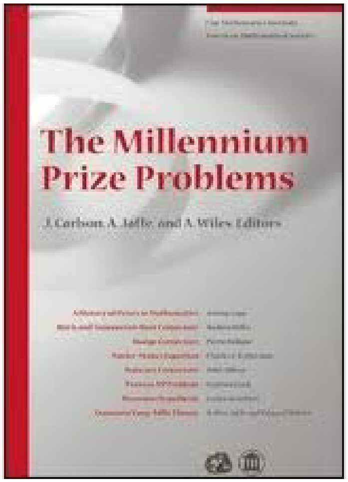
- The “P versus NP” problem comes from computational complexity theory.

- P means with polynomial time complexity
  - That is, algorithms that have $O(\text{poly}(n))$
  - Sorting is a type of polynomial time problem

- NP means mon-deterministic polynomial
  - The answer can be checked in polynomial time, but cannot find the answer in polynomial time for large n.
  - The TSP problem is an NP problem.

- This is the most important question in Computer Science





Algorithmic Problems

THE UNIVERSITY OF MELBOURNE

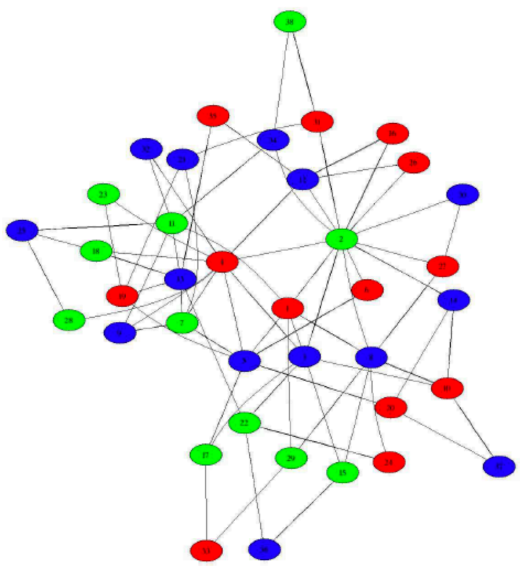
- When we talk about a problem in computer science, we almost always mean a family of instances of a general problem.

- An algorithm for the problem has to work for all possible instances (inputs).

- Example: The sorting problem—an instance is a sequence of items.

- Example: The graph *k*-colouring problem—an instance is a graph.

- Example: Equation solving problems—an instance is a set of, say, linear equations.



Easy and Hard Problems

- A path in a graph G is simple if it visits each node of G at most once.

- Consider this problem for undirected graphs G:

  SPATH: Given G and two nodes a and b in G, is there a simple path from a to b of length at most k?

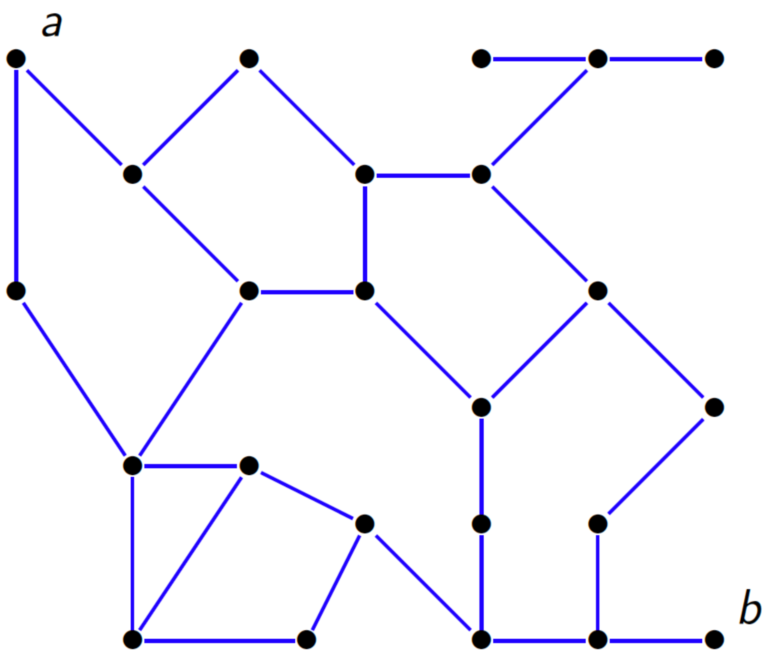
- And this problem:

  LPATH: Given G and two nodes a and b in G, is there a simple path from a to b of length at least k?

- If you had a large graph G, which of the two problems would you rather have to solve?

Easy and Hard Problems

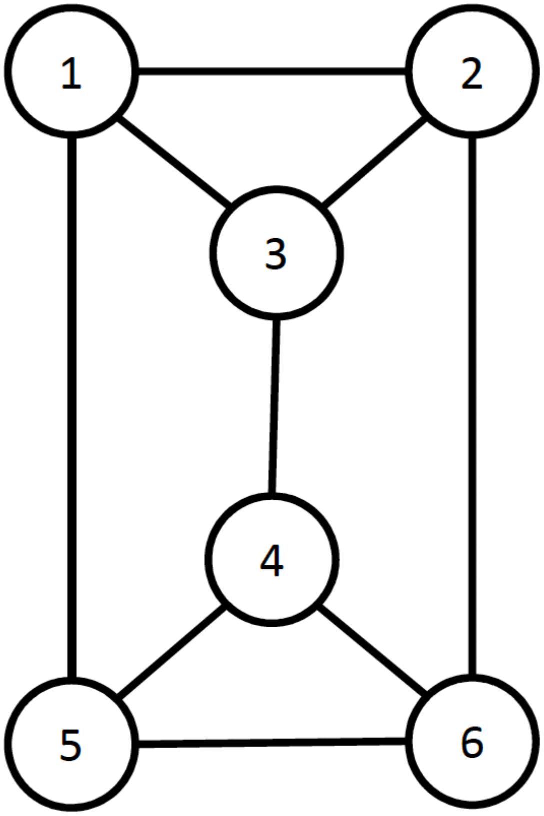
- There are fast algorithms to solve SPATH.
- Nobody know of a fast algorithm for LPATH.
- It is likely that the LPATH problem cannot be solved in polynomial time (But we do not know for sure).



Easy and Hard Problems

THE UNIVERSITY OF MELBOURNE

- Two other related problems:
  - EUL: The Eulerian tour problem: in a graph is there a path which visits each edge of the graph once, returning to the origin?
  - HAM: The Hamiltonian tour problem: in a graph, is there a path which visits each node of the graph once, returning to the origin?



- Is the Eulerian tour problem P?
  - We just need to know whether the edge distribution is even.
- Is the Hamiltonian tour problem P?
  - No. As the number of nodes increases, the runtime becomes exponential.

Easy and Hard Problems

- Try to rank these problems according to inherent difficulty:
  - SAT: Given a propositional formula φ, is φ satisfiable?
  - SUBSET-SUM: Given a set S of positive integers and a positive integer t, is there a subset of S that adds up to t?
  - 3COL: Given a graph G, is it possible to colour the nodes of G using only three colours, so that no edge connects two nodes of the same colour?

Easy and Hard Problems

- Try to rank these problems according to inherent difficulty:
  - SAT: Given a propositional formula φ, is φ satisfiable?
  - SUBSET-SUM: Given a set S of positive integers and a positive integer t, is there a subset of S that adds up to t?
  - 3COL: Given a graph G, is it possible to colour the nodes of G using only three colours, so that no edge connects two nodes of the same colour?

AND		
p	**q**	$\boldsymbol{p \wedge q}$
F	F	F
F	T	F
T	F	F
T	T	T

Then $OR = p \vee q$, etc.

Satisfiable problem: $\varphi = x_1 \vee \neg x_2 \wedge x_3 \vee \neg x_4$

Can we find an assignment of values (F or T) to the variables x_i that result in $\varphi == T$?

Polynomial-Time Verifiability

- SAT, SUBSET-SUM, 3COL, LPATH and HAM share an interesting property.
- If someone claims to have a solution (a "yes instance") then we can quickly test their claim.
- In other words, these problems seem hard to solve, but the solutions allow for efficient verification. They are polynomial-time verifiable.
- That property is shared by a very large number of interesting problems in planning, scheduling, design, information retrieval, networks, games, …
- To understand this concept, we need to talk about Turing Machines.

Turing Machines

- Turing Machines are an abstract model of a computer.

- Despite their simplicity they appear to have the same computational power as any other computing device.

  - That is, any function that can be implemented in C, Java, Python, etc. can be implemented in a Turing Machine.

- Moreover, a Turing Machine is able to simulate any other Turing Machine.

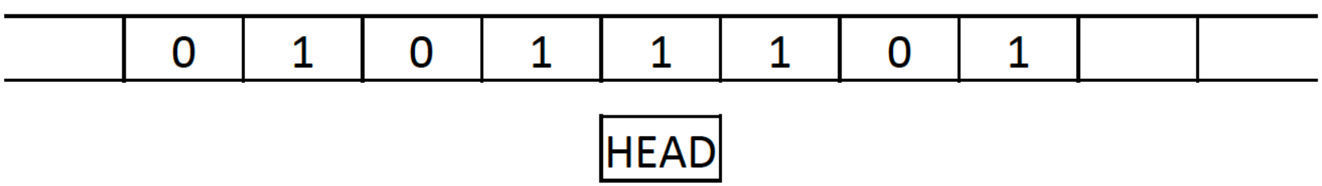
  - This is known as the universality property.

Turing Machines

- A Turing Machine are has an infinite tape through which it takes its input and perform its computations.

- It can:
  - Both read from and write to the tape,
  - Move either left or right over the tape.

- Whether the Turing Machine reads, write or moves left or right depends on a control sequence.

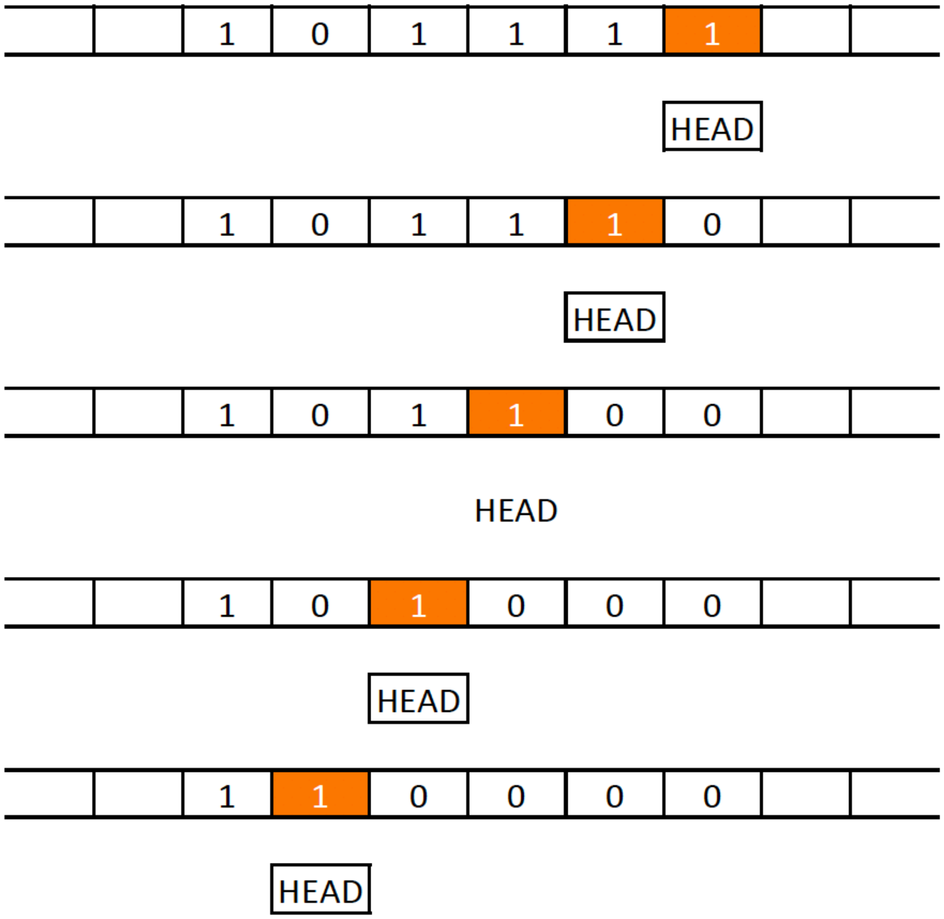


- The tape is unbounded in both directions.

- A Turing machine may fail to halt.

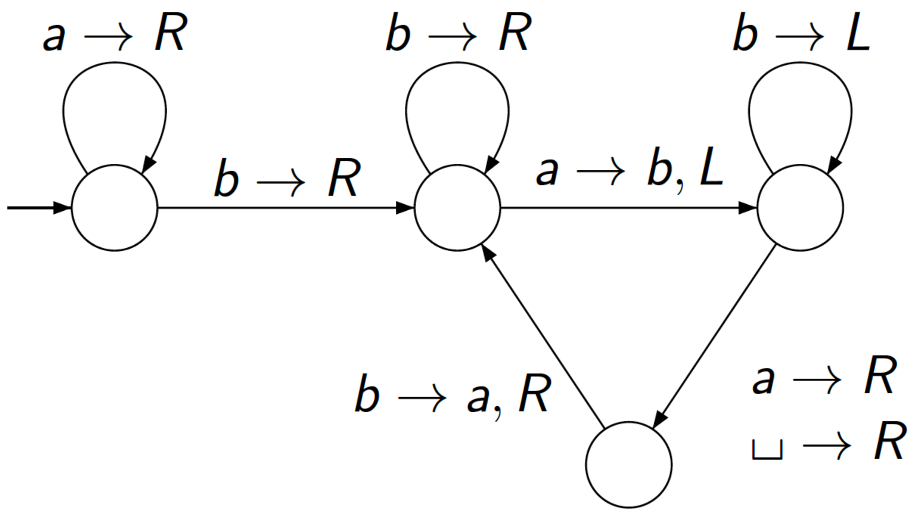
Turing Machines: Example

- Let the control sequence be:
  - If read **1**, write **0**, go **LEFT**
  - If read **0**, write **1**, **HALT**
  - If read _, write **1**, **HALT**.
- Alphabet is { _, **0**, **1** }
- The input will be $47_{10} = 101111_2$
- The output is $48_{10} = 11000_2$
- The rules add one to a number.



Turing Machines: Complex Example

- This (halting) Turing machine sorts a sequence of as and bs.
- The tape alphabet is $\{ \sqcup, \boldsymbol{a}, \boldsymbol{b} \}$



Turing Machines: Complex Example

- This (halting) Turing machine sorts a sequence of as and bs.

- The tape alphabet is { _, $\boldsymbol{a}$, $\boldsymbol{b}$ }

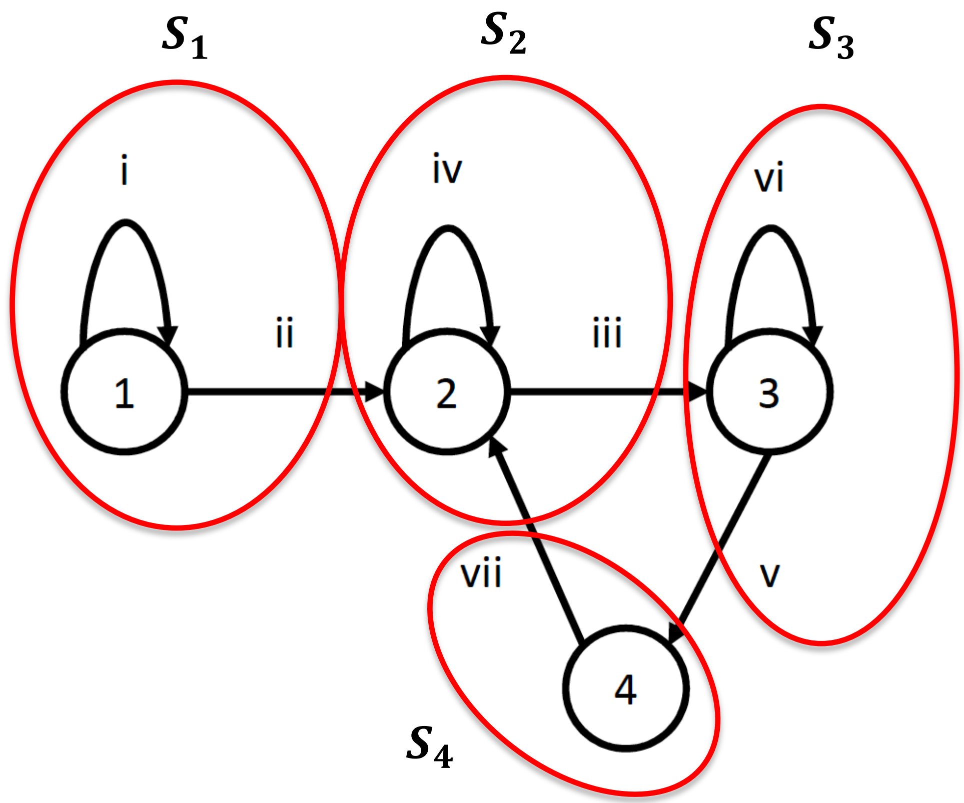
- We will develop a state automaton:

  i. If $\boldsymbol{S_1}$ and $\boldsymbol{a}$, go **RIGHT**, stay in $\boldsymbol{S_1}$
  ii. If $\boldsymbol{S_1}$ and $\boldsymbol{b}$, go **RIGHT**, go to $\boldsymbol{S_2}$

  iii. If $\boldsymbol{S_2}$ and $\boldsymbol{a}$, write $\boldsymbol{b}$, go **LEFT**, go to $\boldsymbol{S_3}$
  iv. If $\boldsymbol{S_2}$ and $\boldsymbol{b}$, go **RIGHT**, stay in $\boldsymbol{S_2}$

  v. If $\boldsymbol{S_3}$ and $\boldsymbol{a}$ or _, go **RIGHT**, go to $\boldsymbol{S_4}$
  vi. If $\boldsymbol{S_3}$ and $\boldsymbol{b}$, go **LEFT**, stay in $\boldsymbol{S_3}$

  vii. If $\boldsymbol{S_4}$ and $\boldsymbol{b}$, write $\boldsymbol{a}$, go **RIGHT**, go to $\boldsymbol{S_2}$



Turing Machines: Complex Example

- This (halting) Turing machine sorts a sequence of as and bs.

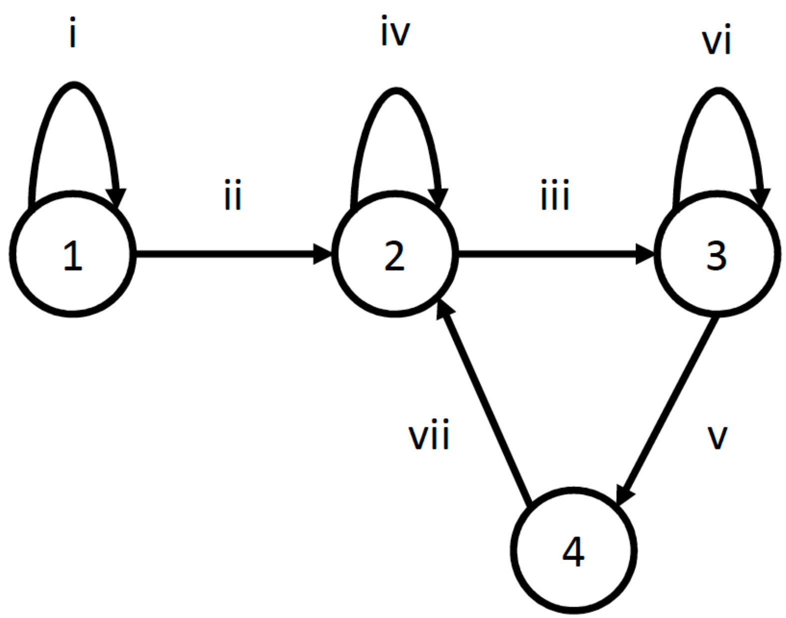
- The tape alphabet is $\{\ \_, \boldsymbol{a}, \boldsymbol{b}\ \}$

- We will develop a state automaton:
  i. If $\boldsymbol{S_1}$ and $\boldsymbol{a}$, go **RIGHT**, stay in $\boldsymbol{S_1}$
  ii. If $\boldsymbol{S_1}$ and $\boldsymbol{b}$, go **RIGHT**, go to $\boldsymbol{S_2}$

  iii. If $\boldsymbol{S_2}$ and $\boldsymbol{a}$, write $\boldsymbol{b}$, go **LEFT**, go to $\boldsymbol{S_3}$
  iv. If $\boldsymbol{S_2}$ and $\boldsymbol{b}$, go **RIGHT**, stay in $\boldsymbol{S_2}$

  v. If $\boldsymbol{S_3}$ and $\boldsymbol{a}$ or _, go **RIGHT**, go to $\boldsymbol{S_4}$
  vi. If $\boldsymbol{S_3}$ and $\boldsymbol{b}$, go **LEFT**, stay in $\boldsymbol{S_3}$

  vii. If $\boldsymbol{S_4}$ and $\boldsymbol{b}$, write $\boldsymbol{a}$, go **RIGHT**, go to $\boldsymbol{S_2}$



Turing Machines: Complex Example

- This (halting) Turing machine sorts a sequence of as and bs.

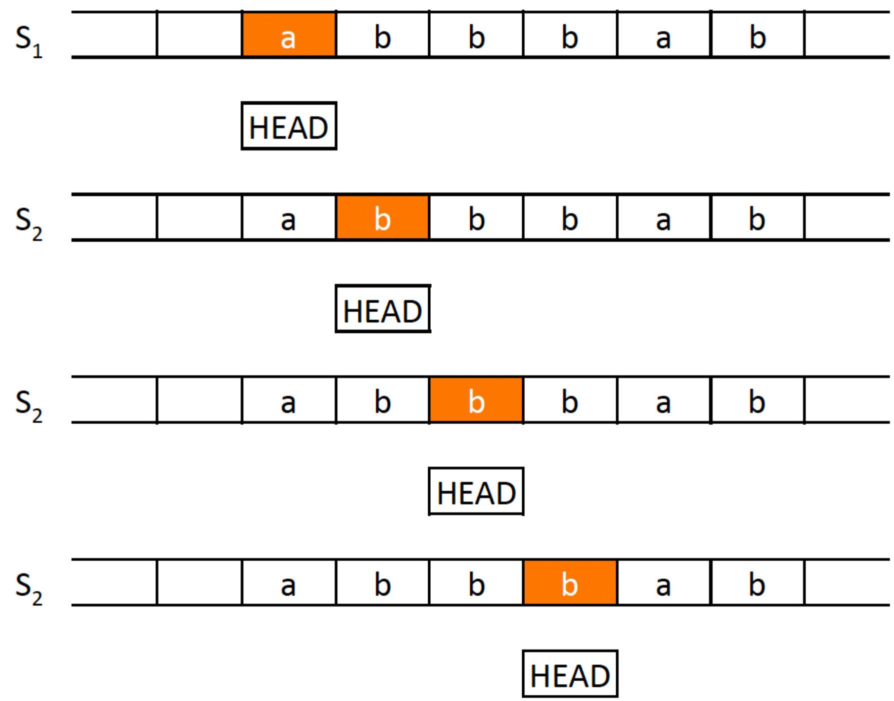
- The tape alphabet is $\{ \_, \boldsymbol{a}, \boldsymbol{b} \}$

- We will develop a state automaton:
  1. If $\boldsymbol{S_1}$ and $\boldsymbol{a}$, go **RIGHT**, stay in $\boldsymbol{S_1}$
  2. If $\boldsymbol{S_1}$ and $\boldsymbol{b}$, go **RIGHT**, go to $\boldsymbol{S_2}$

  3. If $\boldsymbol{S_2}$ and $\boldsymbol{a}$, write $\boldsymbol{b}$, go **LEFT**, go to $\boldsymbol{S_3}$
  4. If $\boldsymbol{S_2}$ and $\boldsymbol{b}$, go **RIGHT**, stay in $\boldsymbol{S_2}$

  5. If $\boldsymbol{S_3}$ and $\boldsymbol{a}$ or _, go **RIGHT**, go to $\boldsymbol{S_4}$
  6. If $\boldsymbol{S_3}$ and $\boldsymbol{b}$, go **LEFT**, stay in $\boldsymbol{S_3}$

  7. If $\boldsymbol{S_4}$ and $\boldsymbol{b}$, write $\boldsymbol{a}$, go **RIGHT**, go to $\boldsymbol{S_2}$



Turing Machines: Complex Example

- This (halting) Turing machine sorts a sequence of as and bs.

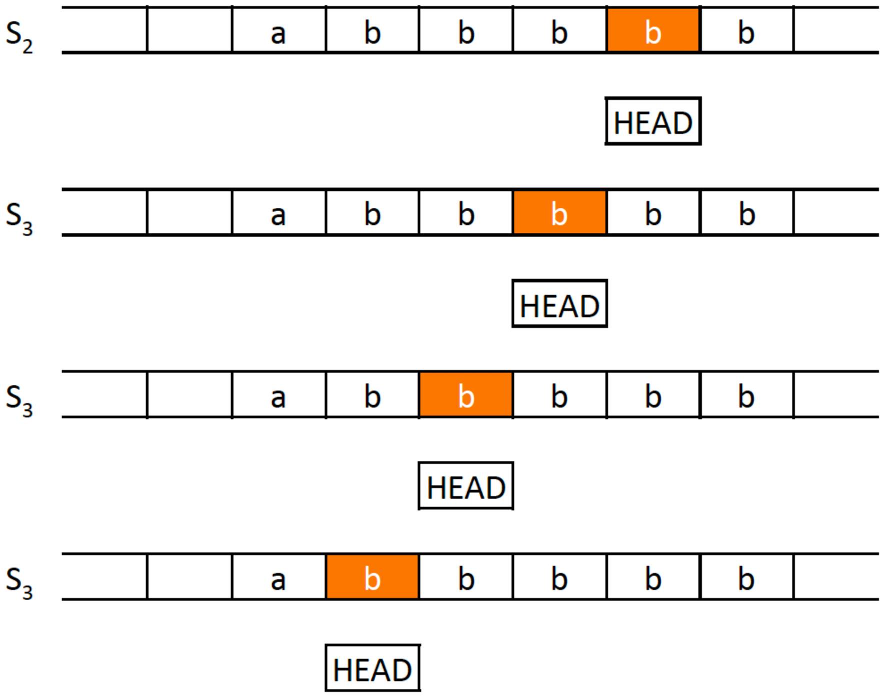
- The tape alphabet is $\{\ \_, \boldsymbol{a}, \boldsymbol{b}\ \}$

- We will develop a state automaton:
  i. If $\boldsymbol{S_1}$ and $\boldsymbol{a}$, go **RIGHT**, stay in $\boldsymbol{S_1}$
  ii. If $\boldsymbol{S_1}$ and $\boldsymbol{b}$, go **RIGHT**, go to $\boldsymbol{S_2}$

  iii. If $\boldsymbol{S_2}$ and $\boldsymbol{a}$, write $\boldsymbol{b}$, go **LEFT**, go to $\boldsymbol{S_3}$
  iv. If $\boldsymbol{S_2}$ and $\boldsymbol{b}$, go **RIGHT**, stay in $\boldsymbol{S_2}$

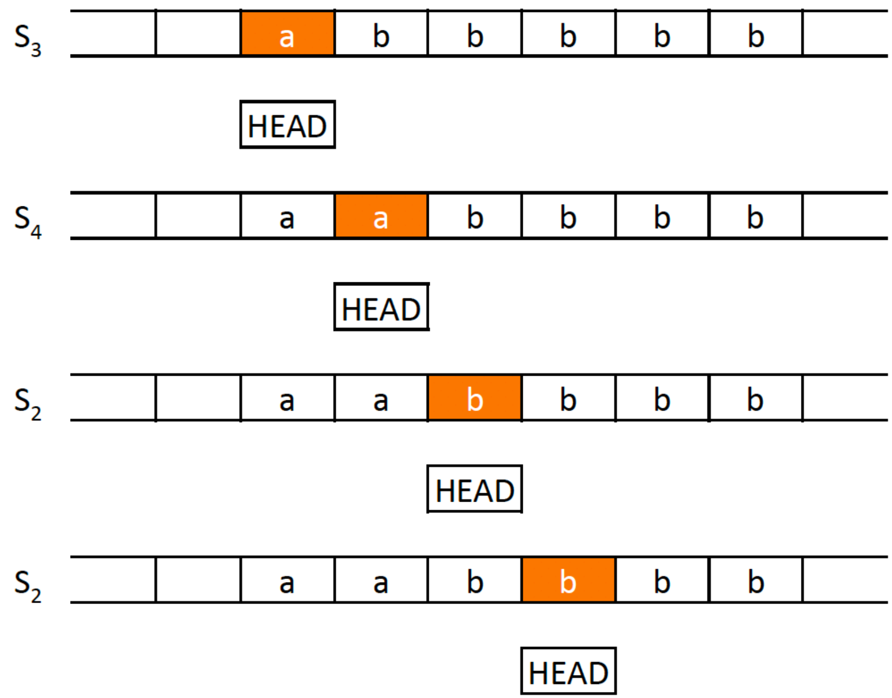
  v. If $\boldsymbol{S_3}$ and $\boldsymbol{a}$ or _, go **RIGHT**, go to $\boldsymbol{S_4}$
  vi. If $\boldsymbol{S_3}$ and $\boldsymbol{b}$, go **LEFT**, stay in $\boldsymbol{S_3}$

  vii. If $\boldsymbol{S_4}$ and $\boldsymbol{b}$, write $\boldsymbol{a}$, go **RIGHT**, go to $\boldsymbol{S_2}$



Turing Machines: Complex Example

- This (halting) Turing machine sorts a sequence of as and bs.
- The tape alphabet is $\{\ \_, \boldsymbol{a}, \boldsymbol{b}\ \}$
- We will develop a state automaton:
  - i. If $\boldsymbol{S_1}$ and $\boldsymbol{a}$, go **RIGHT**, stay in $\boldsymbol{S_1}$
  - ii. If $\boldsymbol{S_1}$ and $\boldsymbol{b}$, go **RIGHT**, go to $\boldsymbol{S_2}$
  - iii. If $\boldsymbol{S_2}$ and $\boldsymbol{a}$, write $\boldsymbol{b}$, go **LEFT**, go to $\boldsymbol{S_3}$
  - iv. If $\boldsymbol{S_2}$ and $\boldsymbol{b}$, go **RIGHT**, stay in $\boldsymbol{S_2}$
  - v. If $\boldsymbol{S_3}$ and $\boldsymbol{a}$ or _, go **RIGHT**, go to $\boldsymbol{S_4}$
  - vi. If $\boldsymbol{S_3}$ and $\boldsymbol{b}$, go **LEFT**, stay in $\boldsymbol{S_3}$
  - vii. If $\boldsymbol{S_4}$ and $\boldsymbol{b}$, write $\boldsymbol{a}$, go **RIGHT**, go to $\boldsymbol{S_2}$



Non-Deterministic Turing Machines

- From now onwards we will assume that a Turing Machine will be used to implement decision procedures: algorithms with a YES/NO answer.

- One variant of the Turing Machine has a powerful guessing capability: If different moves are available the machine will favour a move that ultimately leads to a 'yes' answer.

- Adding this kind of non-determinism to the capabilities of Turing Machines does not change what Turing machine can compute, but it may have an impact of efficiency.

- What a non-deterministic Turing Machine can compute in polynomial time corresponds exactly to the class of polynomial-time verifiable problems.

Non-Deterministic Turing Machines

- What a non-deterministic Turing Machine can compute in polynomial time corresponds exactly to the class of polynomial-time verifiable problems.

- P is class of problems solvable in polynomial time by a deterministic Turing Machine.

- NP is the class of problems solvable in polynomial time by a non-deterministic Turing Machine.

- Clearly $P \subseteq NP$. Is $P = NP$?

Problem Reduction

- The main tool used to determine the class of a problem is reducibility.

- A decision problem P is said to be polynomially reducible to a decision problem Q, if there exists a function t that transforms instances of P to instances of Q such that:
  - t maps all yes instances of P to yes instances of Q and all no instances of P to no instances of Q
  - t is computable by a polynomial time algorithm.

- A decision problem D is said to be NP-complete if:
  - it belongs to class NP
  - Every problem in NP is polynomially reducible to D.

NP-Complete Problems

- SAT, SUBSET-SUM, 3COL, LPATH and HAM, as well as thousands of other interesting and practically relevant problems have been shown to be NP-complete.
- This explains the continuing interest in NP-completeness and related concepts from complexity theory.
- For such problems, we do not know of solutions that are essentially better than exhaustive search.
- There are many other interesting complexity classes, including space complexity classes and probabilistic classes.

Open Problems

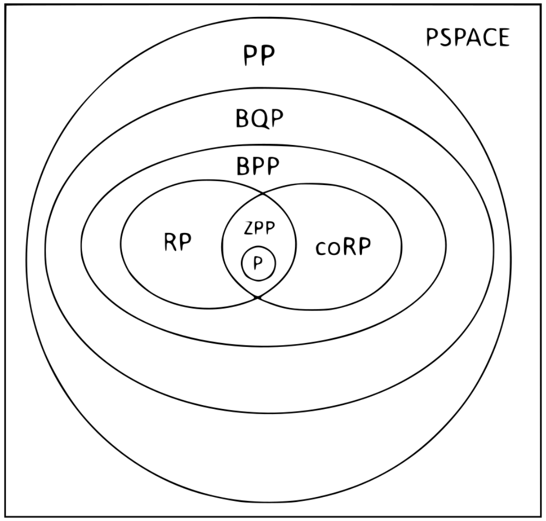
- We do not know whether P=NP, and there are many other unsolved problems.

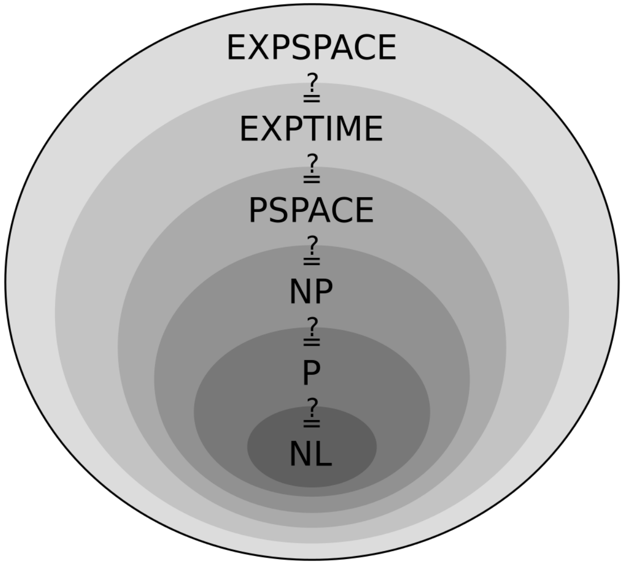
- We know that $P \subseteq EXPTIME$, that is, there are problems that can be solved in exponential time but provably not in polynomial time.

- We also know:

$$P \subseteq RP \subseteq NP \subseteq PSPACE \subseteq NPSPACE \subseteq EXPTIME$$

- But which if these inclusions are strict? (Some must be; all could be…)





Dealing with NP-Completeness

- Pseudo-polynomial problems (SUBSET-SUM and KNAPSACK are in this class): Unless you have really large data, don’t worry; for reasonably sized sets and numbers, the bad behaviour will not have kicked in yet.

- Clever engineering to push the boundary slowly: SAT solvers.

- Approximation algorithms: Settle for less than perfection.

- Live happily with intractability: Sometimes the bad instances never turn up in practice.

Coming Up Next Week

- Special topic lecture on quantum computing
- Subject review