BM40A1500 DATA STRUCTURES AND ALGORITHMS

NP-COMPLETENESS

2024

LIMITS OF COMPUTING AND HARD PROBLEMS

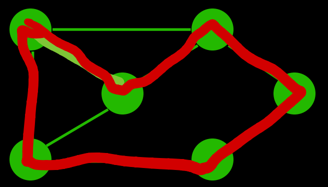
- So far, we have mostly covered algorithms that are efficient
- There are large number of problems for which no efficient algorithms are known.
- An algorithm can be considered efficient if it is **polynomial time**.
  - Algorithms for which the running time is $O(n^k)$, where k is some constant.
  - Note: $O(n^{100})$ algorithm would not be very efficient, however, hardly any algorithms, for which k is very large, exist.
- Exponential time algorithms:
  - Algorithms for which the running time is $\Omega(c^n)$, where $c > 1$ is some constant.
  - Note: $\Omega(1.001^n)$ algorithm would be efficient, however, hardly any such algorithms exist.
  - Problems for which all known algorithms have exponential running time are considered as **hard problems**.
  - Algorithms for hard problems are considered **hard algorithms**.

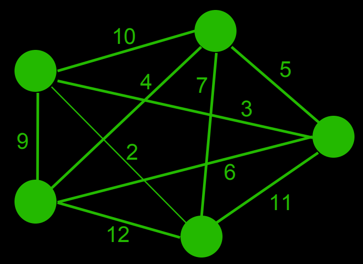
CLASSES P AND NP

- Decision problems:
  - A problem whose output is either “YES” or “NO”.
  - For example, is there a cycle in a graph that visits every vertex exactly once and has a length of X or shorter?
  - In the case of “YES”, the algorithm also outputs the proof: for example, a cycle (sequence of vertices) that fulfills the conditions.
- Class P:
  - Decision problems for which there exist a polynomial time algorithm.
- Class NP:
  - Decision problems for which the solutions (proof) can be verified in polynomial time if the output is ”YES”.
  - For example, given a sequence of vertices, it is quick to check that the graph contains the path, and the length of the path is shorter than X.
  - Note that all the problems in Class P belong to Class NP.

NP-COMPLETENESS

- Decision problem is NP-complete if it is in NP and can be reduced to another NP-complete problem in polynomial time.
- Reduction allows us to solve one problem in terms of another.
  - If Problem A reduces to Problem B and we have an algorithm for Problem B, we can use it to solve Problem A.
- The NP-complete problems are the hardest problems in NP.
- Example:
  - **The Hamiltonian cycle problem**: does a given graph contain a path that visits every vertex exactly once and returns to the starting vertex?
  - **Travelling salesman decision problem**: is there a cycle in a graph that visits every vertex and has a length of X or shorter?
  - The Hamiltonian cycle problem is known to be NP-complete. Is the travelling salesman decision problem NP-complete?





REDUCTION: EXAMPLE

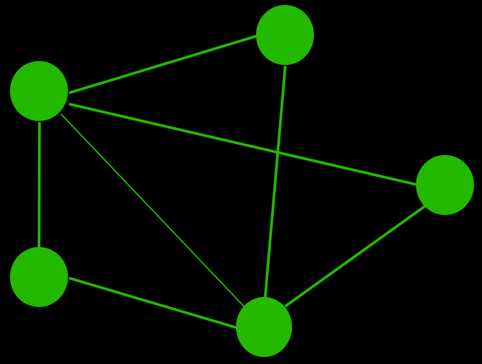
- Does Graph A contain a path that visits every vertex exactly once and returns to the starting vertex (Hamiltonian cycle)?

is the same as asking

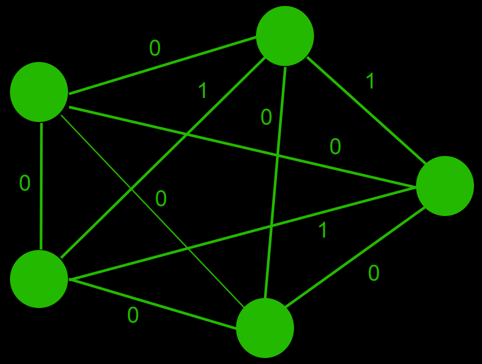
- Does Graph B contain a cycle that visits every vertex exactly once and has a length of 0 (travelling salesman decision problem)?

- We can solve the Hamiltonian Cycle problem by using any algorithm for the travelling salesman problem.
  - The Hamiltonian cycle problem is NP-complete
  - Therefore, the travelling salesman decision problem is NP-complete

Graph A



Graph B



NP-COMPLETENESS

- Examples of NP-complete decision problems:
  - Does the the graph have a cycle that visit every vertex exactly once (Hamiltonian cycle)?
  - Given a set of numbers, can we select a subset that sums up to (exactly) X?
  - Can we color a graph (every vertex) with three colors, so that the adjacent vertices have always different color?
- No efficient (polynomial time and deterministic) algorithm is known for NP-complete problems.
- If we find a polynomial time algorithm for one NP-complete problem, we can use it to solve any NP-complete problem efficiently.
  - This would mean that P = NP
  - Commonly believed that P ≠ NP, but this has not been proved.

OPTIMIZATION PROBLEMS

- For example:
  - Traveling salesman problem: what is the shortest route (path) that visit each city (vertex) once and returns to the origin city?
- Traveling salesman decision problem:
  - Is there a route that visits each city once returning to the origin city and is shorter than X?
  - NP-complete
- We can use an algorithm that solves the decision problem to solve the optimization problem:
  - Let us assume Algorithm A that solves the decision problem.
  - Search for the smallest value of X for which Algorithm A returns “YES” (e.g., using the binary search).
- Since the optimization problem is at least as difficult as the NP-complete decision problems, it is called NP-hard.

NP-HARD / NP-COMPLETE PROBLEMS

- NP-hard optimization problems are very common.

- Examples:
  - Finding an optimal route for a container ship.
  - Packing a container ships in optimal way.
  - Scheduling teaching events by minimizing conflicts.
  - Allocation of taxis for customers.
  - Designing the smallest possible crossword puzzle from a set of words.
  - Selecting seats for wedding guests so that guest knowing each other are as close as possible.

ALTERNATIVE DEFINITION

- Class NP:
  - Problems that can be solved in polynomial time using a non-deterministic algorithm.
    - Algorithm that can check all possible solutions in parallel to determine which is correct/optimal.
    - For example, testing all the possible routes and selecting the shortest.
- With this definition of the class NP, also the NP-hard optimization problems are considered as NP-complete.
- Due to the close connection between the optimization problems and corresponding decisions problems, the difference in definitions is not very significant.
- However, it is good to note that different books use slightly different definitions.

COPING NP-HARD PROBLEMS

- If the size (N) of the problem is small, we can use brute-force
  - Test all the possible solutions (combinations, permutations, subsets, etc.) using backtracking.
- If the size of the problem increases, we can still find optimal solution using
  - Branch-and-bound (see Week 7 material) or
  - Dynamic programming (see Week 8 material).
  - These are still exponential time algorithms when applied to NP-hard problems.
- When the problem size is too big for exact algorithms, we need to settle for approximation algorithms.
  - An algorithm that finds a good, but not necessarily the optimal solution.
  - Greedy approach
  - Heuristics
  - Probabilistic algorithms
  - etc.

APPROXIMATION ALGORITHMS

$\Theta(2^n)$

- Example: Given a set of integers, divide them into two subsets in such a way that the differences of the subset sums is as small as possible. $|S_1 - S_2| = 0$

  [4, 2, 9, 3, 8, 5, 3, 18] $\rightarrow$ [3, 5, 18] : 26 [4, 2, 9, 8, 3] : 26

- Heuristic 1: go trough the integers one-by-one and place the integer to the subset that currently has a smaller sum.

$\Theta(n)$ [4 3 8 3] : 18 [2 9 5 18] : 34 $|S_1 - S_2| = 16$

- Heuristic 2: sort the integers from largest to smallest and then apply Heuristic 1.

$\Theta(n \log n)$ [18 4 3 2] : 27 [9 8 5 3] : 25 $|S_1 - S_2| = 2$

- Randomization: randomize the order of integers and apply Heuristic 1. Repeat multiple times and select the best solution.
  - The more repetitions we do, the more likely we are to get lucky and find an optimal or close-to-optimal solution.

$\Theta(mn)$


LUT
University