ÇUKUROVA UNIVERSITY
FACULTY OF ENGINEERING
COMPUTER ENGINEERING DEPARTMENT

# Lecture 4: Complexity, P, NP, Reducibility

## Complexity

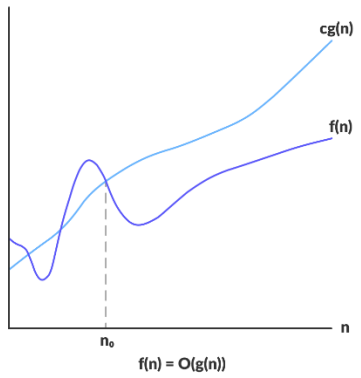
Complexity of an algorithm is a measure of the amount of time, space, and/or communication overhead required by an algorithm for an input of a given size (n).

- Time complexity
- Space complexity
- Communication complexity

## Big oh notation ($O$)

$f(n) \in O(g(n))$ if there exist positive constant C and $n_0$ such that,

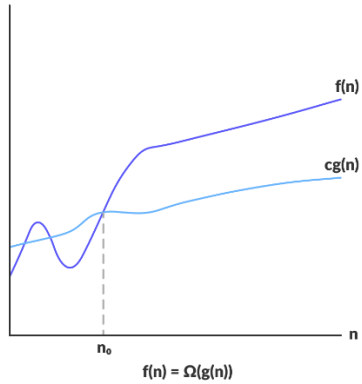
$$0 \leq f(n) \leq C * g(n) \ for\ all\ n \geq n_0$$




## Big Omega ($\Omega$) notation

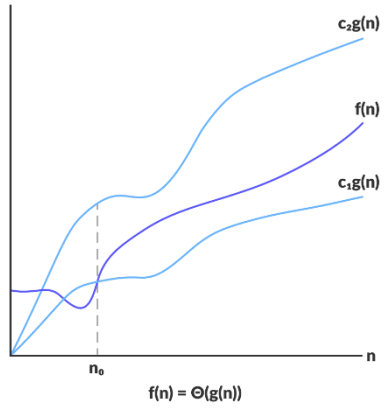
$f(n) \in \Omega(g(n))$ if there exists positive constant C and ($n_0$) such that

$$0 \leq C * g(n) \leq f(n) \ for\ all\ n \geq n_0$$



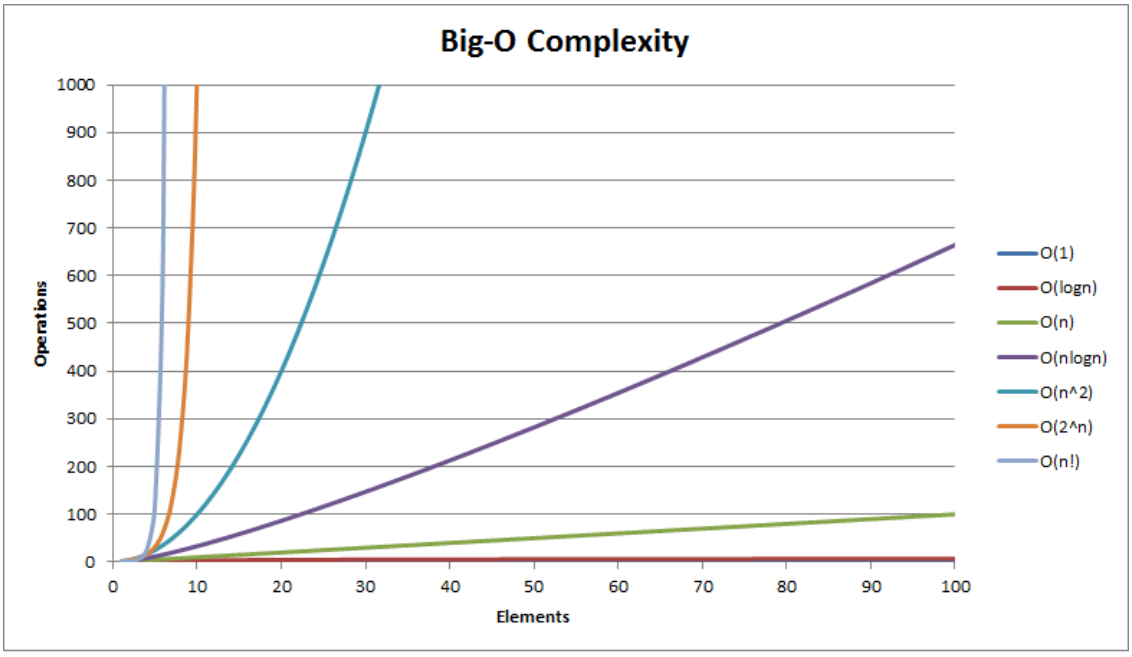
**Big Theta (Θ) notation**

$f \in \Theta(g)$ if **f(n)** is $O(g(n))$ and **f(n)** is $\Omega(g(n))$




The most popular complexity classes:

| Growth | Order |
|---|---|
| Constant | $O(1)$ |
| Logarithmic | $O(log\ n)$ |
| Linear | $O(n)$ |
| Log-linear | $O(nlogn)$ |
| Polynomial | $O(n^k), k \geq 2$ |
| Exponential | $O(d^n), d > 1$ |
| Factorial | $O(n!)$ |



**The complexity class P**

Deterministic TM: $time_T: \sum^* \rightarrow \mathbb{N}$

$time_T(w) =$ Number of configuration transitions when processing $w$

$$f: \mathbb{N}_0 \rightarrow \mathbb{N}_0$$

$$TIME(f) = \{L \subseteq \sum^* \mid \exists det.\ TM\ T\ with\ L = L(T)\ with\ time_T(w) \in O(f(|w|))\}$$

Example: $TIME(n^2)$

$$P = \bigcup_{k \geq 0} TIME(n^k)$$

↓

Class of languages that are decided by *deterministic* Turing machines in polynomial time

**The complexity class NP**

Non-Deterministic TM: $ntime_T: \sum^* \rightarrow \mathbb{N}$

$ntime_T(w) = min($Number of configuration transitions when processing $w)$

$$f: \mathbb{N}_0 \rightarrow \mathbb{N}_0$$

$$NTIME(f) = \{L \subseteq \sum^* \mid \exists non - det.\ TM\ T\ with\ L = L(T)\ with\ ntime_T(w) \in O(f(|w|))\}$$

Example: $NTIME(n^2)$

$$NP = \bigcup_{k \geq 0} NTIME(n^k)$$

↓

Class of languages that are decided by *non-deterministic* Turing machines in polynomial time

**Summary:**

- The class P is the set of all problems that can be solved with a polynomial-time deterministic algorithm (or deterministic Turing machine).
    - Polynomial running time: $O(n^k)$ for fixed k.
    - Also: $O(1)$, $O(n)$, $O(n^2)$, $O(n^3)$, ...,

- All problems in P are solvable polynomially and are therefore considered to be solvable efficiently (i.e. solvable in reasonable time).
  - "Practical" (or “reasonable”) is interpreted very generously
  - $O(n^{100})$ is classified as practicable, as is $O(n^2)$

- NP is the set of all problems that can be solved with a non-deterministic algorithm (or non-deterministic Turing machine) in polynomial time.
- A non-deterministic algorithm may choose (guess) from several possibilities in one calculation step, whereby the algorithm always chooses in such a way that it leads to a positive decision, if there is one.
- All problems that can only be solved with (over)exponential effort (e.g. $O(2^n)$, $O(n!)$, ...) are considered to be not solvable efficiently (i.e. not solvable in a practical time).

| Runtime | 10 | 100 | 1000 | $10^6$ |
|---|---|---|---|---|
| n | 0.01 µsec | 0.1 µsec | 1 µsec | 1 msec |
| $n \log_{10} n$ | 0.01 µsec | 0.2 µsec | 3 µsec | 6 msec |
| $n^2$ | 0.1 µsec | 10 µsec | 1 msec | 16.7 min |
| $n^3$ | 1 µsec | 1 msec | 1 sec | 31.7 years |
| $2^n$ | 1 µsec | $4*10^{13}$ years | ... | ... |
| n! | 3.6 msec | $2.9*10^{141}$ years | ... | ... |

Assumption: Computer performs $10^9$ elementary operations per second.

Estimated age of the universe: $18.8*10^9$ years.

# Question: $P =^{?} NP$

It is assumed that P is not equal to NP.

$$DTM_{\sum} = NTM_{\sum}$$

$$EXPTIME = \bigcup_{c>1} TIME(c^n)$$

$$P \subseteq NP \subseteq EXPTIME \subseteq NEXPTIME$$

**Reducibility**

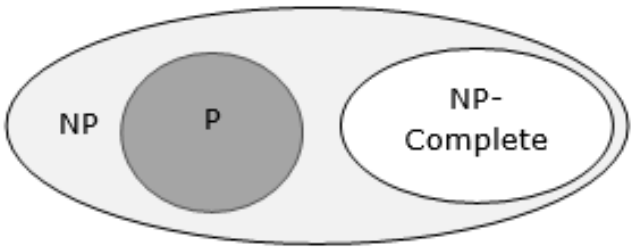
$L_1 \subseteq \sum_1^*$ polynomially reducible to $L_2 \subseteq \sum_2^*$ iff there is a total function $f: \sum_1^* \rightarrow \sum_2^*$ that is computable in polynomial time: $w \in L_1$ iff $f(w) \in L_2$

$L_1 \leq L_2$ *or* $L_1 \leq_{pol} L_2$

- transitive, $L_1 \leq L_2,\ L_2 \leq L_3 \Rightarrow L_1 \leq L_3$
- $L_2 \in P, L_1 \leq L_2 \Rightarrow L_1 \in P$
- $L_2 \in NP, L_1 \leq L_2 \Rightarrow L_1 \in NP$
- $L_2 \notin P, L_1 \leq L_2 \Rightarrow L_1 \notin P$
- $L_2 \notin NP, L_1 \leq L_2 \Rightarrow L_1 \notin NP$

**NP-easy, NP-hard, NP-complete**

- $L \subseteq \sum^*$ is called NP-easy iff $L \in NP$
- $L \subseteq \sum^*$ is called NP-hard iff $L^{\iota} \leq L,\ \forall L^{\iota} \in NP$
- $L \subseteq \sum^*$ is called NP-complete (NPC) iff L is NP-easy and NP-hard.




**Note:** L ∈ NPC. Then P = NP iff L ∈ P

**Methods to show that L is in NPC:**

(1) (i) Show $L \in$ NP

(ii) Show $L^{\prime} \leq L,\ \forall L^{\prime} \in NP$

(2) (i) Show $L \in$ NP

(ii) Show $L^{\prime} \leq L,\ for\ a\ single\ L^{\prime} \in NPC$

$$L^{\prime} \in NPC \rightarrow L^{\prime\prime} \leq L^{\prime} \leq L,\ \ L^{\prime\prime} \in NP$$

**Examples of Reduction:**

$$SAT \in NPC\ (S.Cook)$$

$$SAT \leq 3SAT \leq HAMILTON \leq TSP$$

$$3SAT \leq CLIQUE \leq VERTEX\ COVER$$

$$3SAT \leq MAX2SAT \leq NAESAT \leq 3CG$$

$$3SAT \leq KNAPSACK \leq PARTITION \leq BINPACKING$$

**Concrete NPC Examples**

Many natural problems we would like to solve are in NP. Every problem in NP has an exponential time algorithm.

**Decision problems vs. Optimization problems:**

- A decision problem asks us to check if something is true (possible answers: 'yes' or 'no')
  - Prime numbers
    - Instance: A positive integer n
    - Question: is n prime?

- An optimization problem asks us to find, among all feasible solutions, one that maximizes or minimizes a given objective
  - Single shortest-path problem
    - Instance: Given a weighted graph G, two nodes s and t of G
    - Problem: Find a simple path from s to t of minimum total length

## A. Satisfiability (SAT) Problem

### i. Definition

- A Boolean formula is in Conjunctive Normal Form (CNF) if it is the AND of clauses.
- Each clause is the OR of literals.
- A literal is either a variable or the negation of a variable.
- Problem: Given a Boolean statement in CNF, determine whether it is satisfiable or not.

### ii. Example

- A Boolean formula is satisfiable iff there is an assignment of 0s and 1s to the variables that makes the entire formula in CNF evaluate to 1 (true); otherwise, it is unsatisfiable.
  - The statement $x_1 \wedge (x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_3)$ is satisfiable with $x_1$=true, $x_2$= true, and $x_3$ = false.
- Unsatisfiable:
  - $(x_1 \vee x_2) \wedge (x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_2) \wedge (\neg x_1 \vee \neg x_2)$

- Decision: Is there an assignment of 0s and 1s to the variables that makes the entire formula in CNF evaluate to 1 (true)
- Optimization: See MAX-SAT problem described in (C).

### iii. Application

- SAT problem has applications in model checking, automated theorem proving, software verification etc.

## B. 3SAT

- 3SAT is a special form of SAT. The statement should be in CNF and each clause may contain a maximum of 3 literals.
- Often used in proofs of NP-completeness of other problems

## C. The maximum satisfiability problem (MAX-SAT)

### i. Definition

- The MaxSAT problem is the optimization version of SAT.
- The idea behind this formalism is that sometimes not all the constraints of a problem can be satisfied, and we try to satisfy the maximum number of them.
- MAXSAT is the problem of determining the maximum number of clauses, of a given Boolean formula in CNF, that can be made true by an assignment of truth values to the variables of the formula.
- If the clauses are restricted to have at most 2 literals, as in 2-satisfiability, we get the MAX-2SAT problem.
- If they are restricted to at most 3 literals per clause, as in 3-satisfiability, we get the MAX-3SAT problem.

### ii. Example

$$(x_1 \vee x_2) \wedge (x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_2) \wedge (\neg x_1 \vee \neg x_2)$$

- The above CNF formula is not satisfiable: no matter which truth values are assigned to its two variables, at least one of its four clauses will be false.
- However, it is possible to assign truth values in such a way as to make three out of four clauses true; indeed, every truth assignment will do this.
- Therefore, if this formula is given as an instance of the MAX-SAT problem, the solution to the problem is the number three.

## D. NAE3SAT

### i. Definition

- Like 3SAT, an instance of the problem consists of a collection of Boolean variables and a collection of clauses, each of which combines three variables or negations of variables.
- However, unlike 3SAT, which requires each clause to have at least one true Boolean value, NAE3SAT requires that the three values in each clause are not all equal to each other (in other words, at least one is true, and at least one is false).
- In other words: There must be a satisfying assignment of 0s and 1s to the variables of the formula, with the conditions
  - no clauses have all three literals equal in truth value
  - the entire formula in CNF will be evaluated to 1 (true)

### ii. Example

- Take

$$\begin{aligned}\phi &= (\neg x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_2 \vee \neg x_3) \\ &\wedge (x_1 \vee x_2 \vee x_3)\end{aligned}$$

  as an example.

- Then $\{x_1 = \texttt{true}, x_2 = \texttt{false}, x_3 = \texttt{false}\}$ NAE-satisfies $\phi$ because

$$\begin{aligned}&(\texttt{false} \vee \texttt{true} \vee \texttt{true}) \wedge (\texttt{false} \vee \texttt{false} \vee \texttt{true}) \\ \wedge\ &(\texttt{true} \vee \texttt{false} \vee \texttt{false}).\end{aligned}$$

### iii. Application

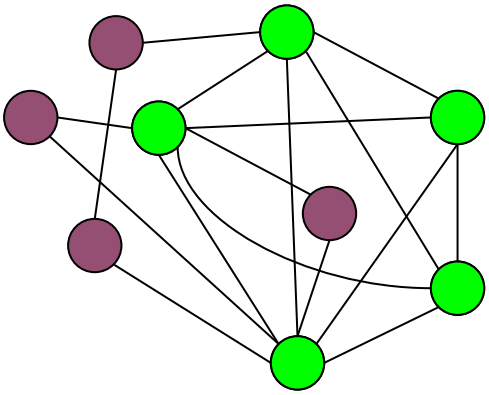
- Often used in proofs of NP-completeness of other problems

## E. Clique

### i. Definition

- Input: A graph G=(V, E).
- Problem: To decide if there is a set of nodes $C=\{v_1,...,v_k\} \subseteq V$, s.t for any $u,v \in C$: $(u,v) \in E$.

**ii. Example**



- Decision: Given an undirected G=(V, E) and integer k, does G have a clique of size k.

- Optimization: Find the maximum number of nodes that form a clique.

**iii. Application**

- Consider a social network, where the graph's vertices represent people, and the graph's edges represent mutual acquaintance. Then, a clique represents a subset of people who all know each other, and algorithms for finding cliques can be used to discover these groups of mutual friends.
- Along with its applications in social networks, the clique problem also has many applications in bioinformatics and computational chemistry.

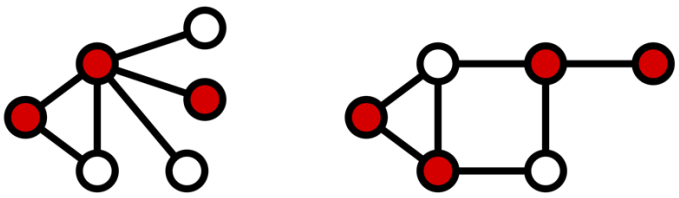
## F. Vertex Cover

**i. Definition:**

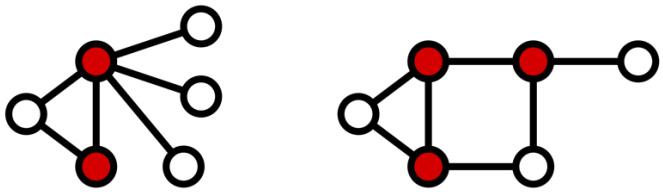
- A **vertex cover** (sometimes **node cover**) of a graph is a set of vertices that includes at least one endpoint of every edge of the graph.
- A vertex cover of G= (V, E) is $V' \subseteq V$ such that every edge in E is incident to some $v \in V'$.

**ii. Examples**

- Examples of vertex covers:



- Examples of minimum vertex covers:



- Decision: Given an undirected G=(V, E) and integer k, does G have a vertex cover with k vertices?
- Optimization: Given an undirected G=(V, E), what is the minimum number of vertex cover nodes?

### iii. Applications:

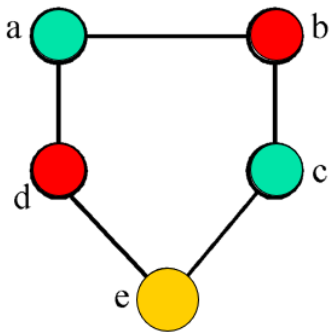
- A commercial establishment interested in installing the fewest possible closed circuit cameras covering all hallways (edges) connecting all rooms (nodes) on a floor might model the objective as a vertex cover minimization problem.
- Similarly, consider the problem of placing guards in a museum where corridors in the museum correspond to edges and the task is to place a minimum number of guards so that there is at least one guard at the end of each corridor.

## G. Graph Coloring Problem

### i. Definition

- A coloring of a graph G = (V, E) is a function f : V $\rightarrow$ { 1, 2, 3,…, k } such that if (u, v) $\in$ E, then f(u)≠f(v).
- The graph coloring problem is to determine if G has a coloring for k.

### ii. Example




3-colorable
f(a)=1, f(b)=2, f(c)=1
f(d)=2, f(e)=3

- Decision: Given an undirected G = (V, E) and integer k, determine if G has a coloring for k?
- Optimization: Given an undirected G = (V, E), what is the minimum number of colors satisfying the graph coloring criteria?

### iii. Applications

- Mobile Radio Frequency Assignment: When frequencies are assigned to towers, frequencies assigned to all towers at the same location must be different. How to assign frequencies with this constraint? What is the minimum number of frequencies needed? This problem is also an instance of graph coloring problem where every tower represents a vertex and an edge between two towers represents that they are in range of each other
- Map Coloring: Geographical maps of countries or states where two neighbor (adjacent) cities cannot be assigned same color. Four colors are sufficient to color any map (See Four Color Theorem)

## H. Partition problem

### i. Definition

- Given a set of positive numbers A = { $a_1$,$a_2$,…,$a_n$ }, determine if ∃ a partition P, s.t. $\sum_{i \in P} a_i = \sum_{i \in P} a_i$

### ii. Example

- A = {3, 6, 1, 9, 4, 11}
- Partition: {3, 1, 9, 4} and {6, 11}

- Decision: Given n positive numbers. Can the n numbers be divided in two clusters whereas the sums of the numbers in the two clusters are the same?

- Optimization: We are looking for a division of these numbers into two clusters so that the difference between the sums of the numbers in the two clusters is as small as possible.

### iii. Application:

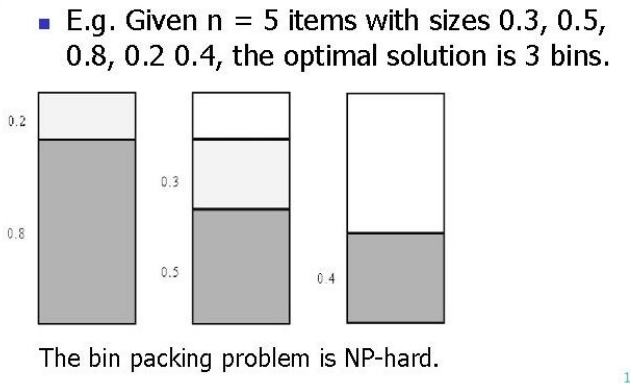
- Number partitioning has applications in areas like public key encryption and task scheduling.

## İ. Bin Packing Problem

### i. Definition

- Objects of different sizes should be stored in as few containers as possible. The containers are the same size.
- Decide whether n objects with weights $0 < w_i ...., w_n < 1$ can be packed in at most k containers.
- No container may hold objects with a total weight> 1.

### ii. Example




- Decision: Given n and k. Can the n objects be distributed among the k containers in such a way that none of the containers overflows?

- Optimization: Find a mapping between items' weights and containers that minimizes the number of containers.

### iii. Application

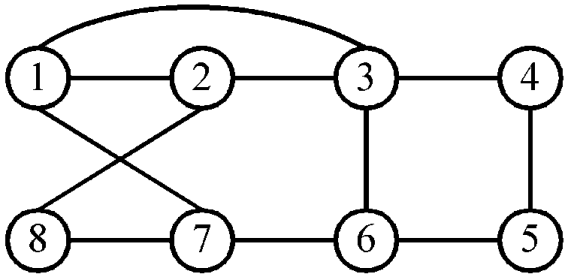
- Binpacking problem has applications in packaging industry.

## J. Hamiltonian cycle problem (HCP)

### i. Definition

- A Hamiltonian cycle is a round trip path along n edges of G which visits every vertex once and returns to its starting vertex.

### ii. Example




Hamiltonian cycle: 1, 2, 8, 7, 6, 5, 4, 3, 1.

- Decision: Given a graph G(V, E), the problem is to determine if the graph contains a Hamiltonian cycle consisting of all the vertices belonging to V?

- Optimization: See Travelling Salesman Problem described in (K).

### iii. Application

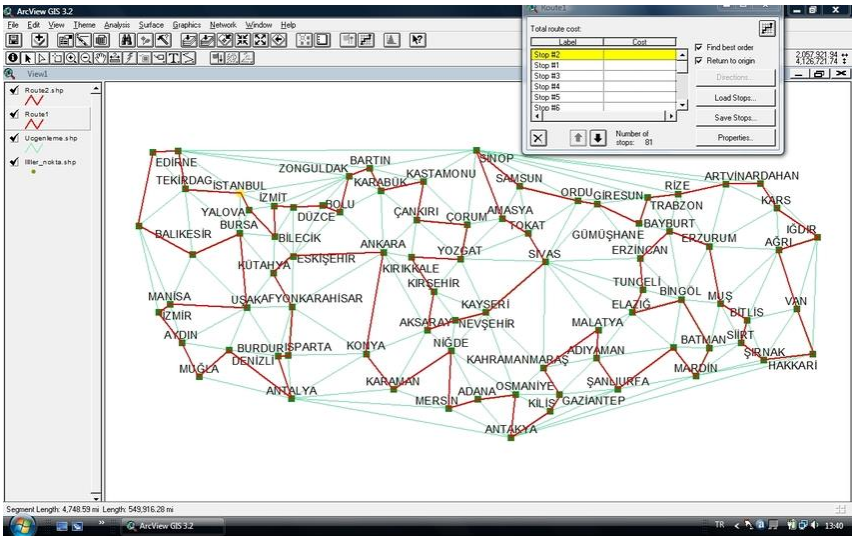
- HCP has many applications such as the choice of travel routes and network topology.

## K. Travelling Salesman Problem

### i. Definition

- This is harder than the Hamilton circuit problem.
- Instead of just finding a circuit, we also want to find the circuit that has the lowest cost in a weighted graph.

### ii. Example



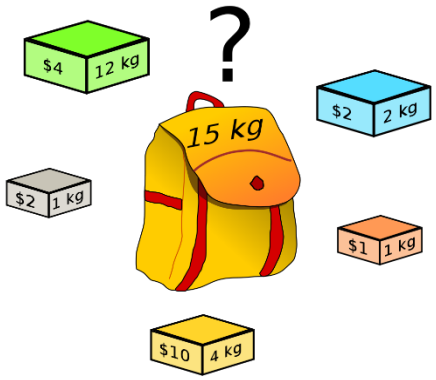
- Optimization: Given a complete (undirected) graph G = (V, E) with integer edge weights c(e) ≥ 0, find a Hamiltonian cycle of minimum cost.

## L. Knapsack

### i. Definition

- Given n items and a "knapsack" that can carry weight up to W kg
- Object i weighs $w_i > 0$ kg and has value $v_i > 0$.
- Goal: fill knapsack so as to maximize total value.

### ii. Example




$v_i$ / $w_i$

| Item | Value | Weight |
|---|---|---|
| 1 | 1 | 1 |
| 2 | 6 | 2 |
| 3 | 18 | 5 |
| 4 | 22 | 6 |
| 5 | 28 | 7 |

**W = 11**

Greedy = 25: {5, 2, 1}
OPT value = 40: {3, 4}

- Decision: Can a value of at least V be achieved without exceeding the weight W?

- Optimization: Given n items, each with a weight and a value, determine the number of each item to include in a collection so that the total weight is less than or equal to a given limit W and the total value is as large as possible.

### iii. Application

- The most obvious application of the Knapsack Problem is in selecting what items go into a shipping box. Determining what are the most valuable items that will fit without exceeding the predetermined weight limit on an order-by-order basis not only reduces shipping costs, it reduces the amount of packing material used and potentially the overall number of boxes.

## M. Multiprocessor Scheduling

### i. Definition

- Input: *m* identical machines, *n* jobs with *i*th job having processing time $t_i$

- Goal: Schedule jobs to machines such that
  - jobs run uninterrupted on a machine
  - a machine processes only one job a time
  - makespan or maximum load on any machine is minimized

Let $A(i)$ be the set of jobs assigned to machine $i$. The load on $i$ is

$$\sum_{i \in A(i)} t_j$$

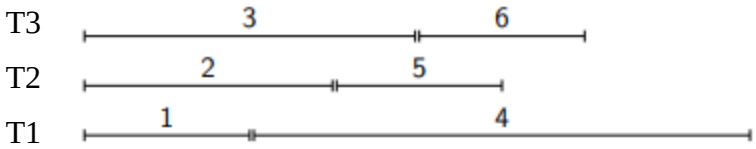
The makespan of $A$ is $T = max_i T_i$

### ii. Example

- Consider 6 jobs whose processing times is given as follows:

| Jobs | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| $t_i$ | 2 | 3 | 4 | 6 | 2 | 2 |

- Consider the following job schedule on 3 machines




- The loads are: T1 = 8, T2 = 5, and T3 = 6. So makespan of schedule is 8.

- Decision: Is there a distribution of the *n* jobs (with their runtimes) over the *m* machines such that they are processed in a total time of at most *D* (D = deadline).

- Optimization: Given n jobs with their runtimes, and m processors; find a schedule with minimum finish (makespan) time.

### iii. Application

- The applications of this problem are most strongly associated with the scheduling of computational tasks in a multiprocessor environment.
- Scheduling of processes by operating systems