Theoretical Concepts of Computer Science
Review Topics

Last Updated: January 23rd, 2023

1 Sets

A **set** represents a collection of items, where each item is called a **member** or **element** of the set.

The most common way to represent a set is by using **list notation**, where the set members are listed one-by-one, and the list is delimited by braces. For example,

$$\{2, 3, 5, 7, 11\}$$

uses list notation to describe the set consisting of all prime numbers that do not exceed 11. Note that the order in which the members are listed does not matter. Indeed the sets $\{2, 3, 5, 7, 11\}$ and $\{3, 11, 5, 2, 7\}$ are identical. Also, each member occurs only *once* in the set, meaning that, e.g. $\{1, 2, 2, 3, 3, 3\} = \{1, 2, 3\}$.

The set having no members is called the **empty set**, and is denoted by $\emptyset$.

Sometimes we need to use **informal list notation** by including one or more instances of an **ellipsis** $\ldots$ to indicate that a pattern is to be continued, either indefinitely or up to some value. The beginning pattern usually consists of one or more members that imply a pattern, followed by an ellipsis, and ending with zero or more members of the set.

Example 1.1. The following are some examples of informal list notation.

1. The set of prime numbers less than 100 may be written as

$$\{2, 3, 5, 7, 11, \ldots, 97\}.$$

2. $\mathcal{N}$ denotes the set of **natural numbers** $\{0, 1, 2, \ldots\}$.

3. $\mathcal{I}$ denotes the set of integers

$$\{0, \pm 1, \pm 2, \ldots\} = \{\ldots, -2, -1, 0, 1, 2, \ldots\}.$$

We may use the **membership symbol** $\in$ to indicate that an item is a member of some set. For example, $-2 \in \mathcal{I}$ asserts that -2 is a member of $\mathcal{I}$. On the other hand, $3.14 \notin \mathcal{I}$ asserts that 3.14 is *not* a member of $\mathcal{I}$ since it is not an integer.

Example 1.2. The following are all true statements.

1. $61 \in \{2, 3, 5, 7, 11, \ldots, 97\}$.
2. $39 \notin \{2, 3, 5, 7, 11, \ldots, 97\}$.
3. $\{3\} \in \{\emptyset, \{1\}, \{3\}, \{1, 2\}, \{2, 3\}, \{1, 2, 3\}\}$
4. $3 \notin \{\emptyset, \{1\}, \{3\}, \{1, 2\}, \{2, 3\}, \{1, 2, 3\}\}$

$\square$

$\subseteq$ $\subset$

1.1 Subsets

Set A is said to be a **subset** of set B iff every member of A is also a member of B. This is symbolically denoted by $A \subseteq B$. Moreover, A is said to be a **proper subset** of B, denoted $A \subset B$, iff it is a subset of B, but there is some member of B who is not a member of A.

Example 1.3. Sets $\{3\}$ and $\{1, 3\}$ are proper subsets of $\{1, 2, 3\}$, while $\{1, 2, 3\}$ is a subset of $\{1, 2, 3\}$, but not a proper subset. $\square$

$\{1,2,3\} \subseteq \{1,2,3\}$

Example 1.4. Neither set $A = \{2, 3, 6, 7\}$ nor set $B = \{2, 3, 5, 7, 11, 13, 17\}$ is a subset of $C = \{2, 3, 5, 7, 11, 13\}$ since $6 \in A$ but $6 \notin C$, and $17 \in B$ but $17 \notin C$.

$A \not\subseteq C$ $\quad$ $B \not\subseteq C$

Example 1.5. $A = \{\{1,2\},\{3,4\},\{3,5\}\}$ is *not* a subset of $B = \{1,2,3,4,5\}$ since, e.g., $\{1,2\} \in A$ but $\{1,2\} \notin B$. This is true since $\{1,2\}$ is a set, and B's members are the natural numbers 1 through 5 which are not sets.

$A \not\subseteq B$

sets numbers

Example 1.6. Given $B = \{2, \{3, 7\}, 3, 4, \{5\}, \{6, 8, 9\}\}$, all of the following statements are true.

a. $\{2, 4\} \subseteq B$. since $2 \in B$ and $4 \in B$

b. $\{3, 7\} \not\subseteq B$. $3 \in B$ but $7 \notin B$

c. $9 \notin B$. 9 is not a member of B

d. $\{5\} \in B$.

$\{3, 7\} \in B$

$\{5\} \not\subseteq B$

$\{\{5\}\} \subseteq B$

The **power set** of a set S, denoted $\mathcal{P}(S)$ is the set of all subsets of S. Note that if $|S| = n < \infty$, then $|\mathcal{P}(S)| = 2^n$, since, for each of the n members of S, one has a binary choice as to whether or not to add the member to the subset. This makes a total of

$$\underbrace{2 \times 2 \times \cdots \times 2}_{n \text{ times}} = 2^n$$

different possible subsets.

Example 1.7. For $S = \{1, 2, 3\}$ we have

000

$$\mathcal{P}(S) = \{\emptyset, \{1\}, \{2\}, \{3\}, \{1,2\}, \{1,3\}, \{2,3\}, \{1,2,3\}\}.$$

$\{1\} \to 100$ $\qquad 2 \times 2 \times 2 = 8$

$\{2\} \to 010$

$\{3\} \to 001$

$\vdots$

$\{1,2,3\} \to 111$

2 Graphs

A **Graph** is a pair of sets V, E, where

Graph $G = (V, E)$ V is a set of **vertices**, also called **nodes**, while E is a set whose members are pairs of vertices and are called **edges**. Each edge may be written as a tuple of the form (u, v), where $u, v \in V$.

Adjacency and Incidence If $e = (u, v)$ is an edge, then we say that u is **adjacent** to v, and that e is **incident** with u and v.

Order $|V| = n$ is called the **order** of G.

Size $|E| = m$ is called the **size** of G.

Path A **path** P of length k in a graph is a sequence of vertices $P = v_0, v_1, \ldots, v_k$, such that $(v_i, v_{i+1}) \in E$ for every $0 \leq i \leq k - 1$.

Simple Path $P = v_0, v_1, \ldots, v_k$ and the vertices $v_0, v_1, \ldots, v_k$ are all distinct.

Cycle A **cycle** is a path that begins and ends at the same vertex.

Geometrical Representation obtained by representing each vertex as a figure (usually a circle) on a two-dimensional plane, and each edge $e = (u, v)$ as a smooth arcs that connects vertex u with vertex v.

Degree The **degree** of a vertex v, denoted as $\deg(v)$, equals the number of edges that are incident with v. Note: loop edges are counted twice.

Example 2.1. Let $G = (V, E)$, where

$$V = \{SD, SB, SF, LA, SJ, OAK\}$$

are cities in California, and

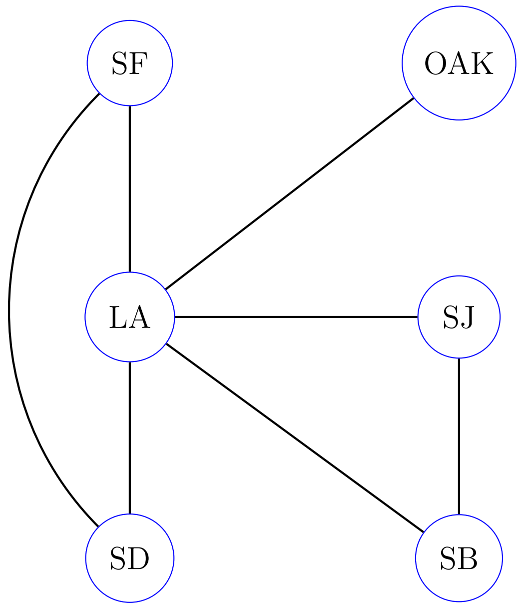
$$E = \{(SD, LA), (SD, SF), (LA, SB), (LA, SF), (LA, SJ), (LA, OAK), (SB, SJ)\}$$

are edges, each of which represents the existence of one or more flights between two cities. Figure 1 shows a graphical representation of G. G has order 6 and size 7.

Figure 2 shows a simple path of length 4. Figure 3 shows a cycle of length 3. Let's verify the Handshaking theorem.

$$\deg(\text{SF}) + \deg(\text{LA}) + \deg(\text{SD}) + \deg(\text{OAK}) + \deg(\text{SJ}) + \deg(\text{SB}) =$$

$$2 + 5 + 2 + 1 + 2 + 2 = 14 = 2 \cdot 7 = 2|E|.$$



$|V| = 6$

$|E| = 7$

Figure 1: Graphical Representation of G

$\deg(SF) + \deg(LA) + \deg(SD) + \deg(oak) + \deg(SJ) + \deg(SB) = 2 + 5 + 2 + 1 + 2 + 2 = 14 = 2|E|$

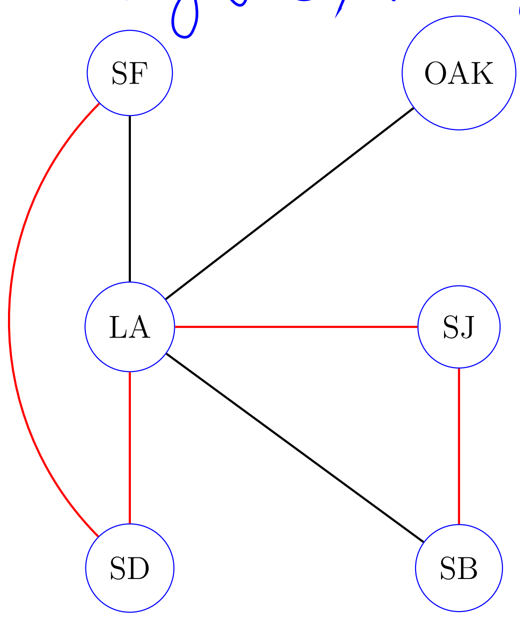


Figure 2: Simple path (in red) $P =$ SF,SD,LA,SJ,SB of length 4

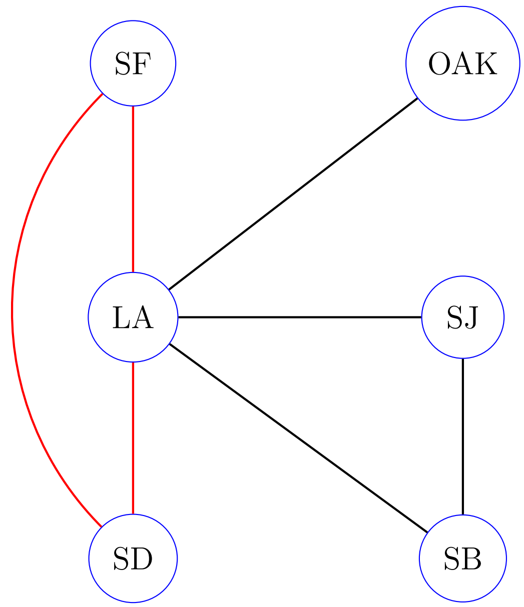


Figure 3: Cycle (in red) $C =$ SF,SD,LA,SF of length 3

3 Computational Problems

Informally, when we think of a problem, we think of a situation that needs to be resolved. Moreover, in computer science we think of a computing problem as not just one situation, but rather a collection of situations that share a common underlying theme. Each situation is referred to as a **problem instance**, and represents a concrete example of the general problem.

Definition 3.1. A **functional problem** is one for which, for each problem instance x, there is a unique solution to x. Letting I denote the set of problem instances, and S the set of possible solutions, we may think of a functional problem as a function $\mathrm{sol} : I \to S$, such that, for each problem instance $x \in I$, $\mathrm{sol}(x)$ equals its solution. Not surprisingly, we call function `sol` the **solution function** for the given problem.

Definition 3.2. A **decision problem** is a functional problem for which $S = \{0, 1\}$. This means that sol is a **predicate function** since its codomain is $\{0, 1\}$, and it is also called the **indicator function** for the given problem. In other words, we may think of each problem instance x as either having or not having some property and $\mathrm{sol}(x)$ indicates whether or not x has the property. Finally, we call x a **positive** (respectively, **negative**) instance iff $\mathrm{sol}(x) = 1$ (respectively, $\mathrm{sol}(x) = 0$).

Definition 3.3. A **discrete computational problem** is a functional problem such that, for both the set I of instances and the set S of solutions, each member of the set can be represented as a word (i.e. a sequence of characters) over some finite alphabet Σ. Thus, we may think of I as a subset of words, i.e. a **language**, over alphabet Σ.

Example 3.4. Consider the problem called `Prime`, where a problem instance is a positive integer $n \geq 2$, and the solution we seek is an answer, `yes` or `no` (or 1 or 0), to the question of whether n is a prime number, i.e. a number that is only divisible by 1 and itself. Then `Prime` is a decision problem as well as a discrete computational problem, since each integer may be written as a word over $\Sigma = \{0, 1, \ldots, 9\}$ while 0 and 1 are single-bit words over $\Sigma = \{0, 1\}$. $\square$

Example 3.5. Consider the problem called `Sort`, where a problem instance is an array a of integers, and the solution we seek is another array b whose members are the same members of a, but in sorted order. `Sort` is certainly a functional problem since, for every integer array a, there is exactly one array, call it $\mathrm{sort}(a)$, whose members are the same as those of a and written in sorted order. Finally, `Sort` is a discrete computational problem since every integer array may be written as a word over the alphabet

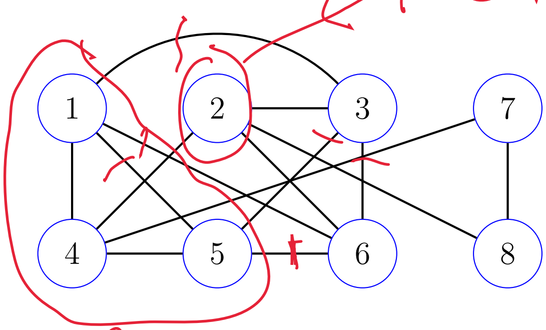
$$\Sigma = \{0, 1, \ldots, 9, [,], ``, "\},$$

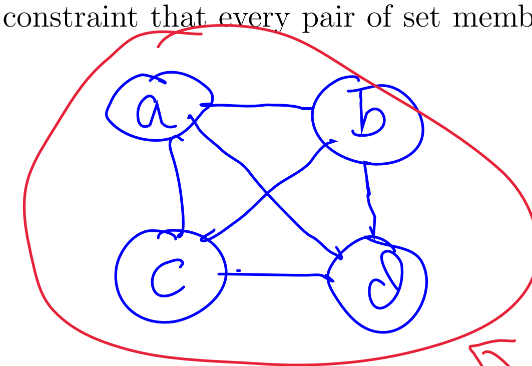
where the left and right bracket symbols are symbols of the alphabet, while the double quotes surrounding the comma are used as delimiters and are *not* part of the "comma" symbol. □

Generally speaking, theoretical computer science prefers to build theoretical frameworks that are exclusive to decision problems because i) it greatly simplifies the analysis, and ii) in most cases a non-decision problem can be framed as a decision problem without losing the meaning and complexity that is inherent in the problem. However, in this course we will also encounter optimizationn problems.

Example 3.6. Consider the problem called `Clique`, where a problem instance is a simple graph $G = (V, E)$, and the solution we seek is a subset $C \subseteq V$ of vertices of maximum size such that, for every $u \in C$ and $v \in C$, $(u, v) \in E$. In other words, every pair of vertices in C must represent an edge in G. Problem `Clique` is called an **optimization** problem, since it calls for finding a structure (in this case a subset) that optimizes (in this case maximizes) an objective function (the objective is to make the set as large as possible) subject to the constraint that every pair of set members must form an edge in G.

1-Clique





3-Clique

4-Clique

4-Clique : $\{1, 3, 5, 6\}$

3.1 Size Parameters

One of the main computing objectives with respect to a given problem is to develop systems and algorithms that are capable of efficiently solving the problem, i.e., efficiently computing the problem's solution function. The most common way to measure efficiency of an algorithm is to examine the number of steps and the amount of memory required by the algorithm as a function of the size of a problem instance. The official **size** of a problem instance is defined as the number of bits needed to store the instance in computer memory. However, this definition can seem both ambiguous and cumbersome to work with in practice. For this reason, we use one or more **size parameters** to indicate the size of a given instance.

The following are examples of size parameters for the problems introduced in this lecture.

`Prime` Size parameter n represents the number of bits of the input integer that is being tested for primality.

`Sort` Size parameter n represents the size (i.e. number of integer members) of the array a that needs sorting. In addition, we may optionally use k to represent the maximum number of bits needed to represent an integer member of a.

`Clique` Size parameter n denotes the **order** (number of vertices) of graph G, while m denotes the **size** (number of edges) of G.