Basrah University
Faculty of Education for Pure Sciences
Computer Department
A.L. Hend Muslim Jasim

Sets and operations

A set is a group of objects represented as a unit. Sets may contain any type of objects, including numbers, symbols, and even other sets. The objects in a set are called elements or members. Sets may be described formally in several ways. One way is by listing a set's elements inside braces. Thus, the set **S = {7, 21, 57},**

contains the elements 7, 21, and 57. The symbols ∈ and ∉ denote set membership and nonmembership. We write 7 ∈ {7, 21, 57} and 8 ∉ {7, 21, 57}. For two sets A and B, we say that A is a subset of B, written $A \subseteq B$, if every member of A also is a member of B. We say that A is a proper subset of B, written $A \subsetneq B$, if A is a subset of B and not equal to B.

The order of describing a set doesn't matter, nor does repetition of its members. We get the same set S by writing {57, 7, 7, 7, 21}. If we do want to take the number of occurrences of members into account, we call the group a multiset instead of a set. Thus {7} and {7, 7} are different as multisets but identical as sets.

An infinite set contains infinitely many elements. We cannot write a list of all the elements of an infinite set, so we sometimes use the "... ." notation to mean "continue the sequence forever." Thus we write the set of natural numbers N as $\{1, 2, 3, \ldots\}$. The set of ***integers*** Z is written as $\{\ldots, -2, -1, 0, 1, 2, \ldots\}$.

The set with zero members is called the ***empty set*** and is written ∅. A set with one member is sometimes called a ***singleton set***, and a set with two members is called an ***unordered pair***.

When we want to describe a set containing elements according to some rule, we write {*n* | rule about *n*}. Thus $\{n \mid n = m^2$ for some $m \in \mathrm{N}\}$ means the set of perfect squares.

If we have two sets *A* and *B*, the ***union*** of *A* and *B*, written $A \cup B$, is the set we get by combining all the elements in *A* and *B* into a single set. The ***intersection*** of *A* and *B*, written $A \cap B$, is the set of elements that are in both *A* and *B*. The ***complement*** of *A*, written $\overline{A}$, is the set of all elements under consideration that are *not* in *A*.

As is often the case in mathematics, a picture helps clarify a concept. For sets, we use a type of picture called a ***Venn diagram***. It represents sets as regions enclosed by circular lines. Let the set START-t be the set of all English words that start with the letter "t". For example, in the figure, the circle represents the set START-t. Several

members of this set are represented as points inside the circle.

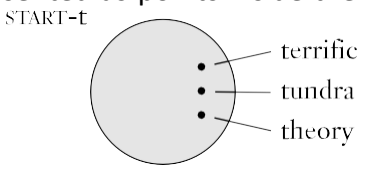

FIGURE 1 Venn diagram for the set of English words starting with "t"

Similarly, we represent the set END-z of English words that end with "z" in the following figure.

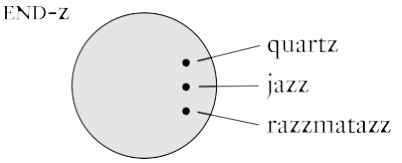

FIGURE 2 Venn diagram for the set of English words ending with "z"

To represent both sets in the same Venn diagram, we must draw them so that they overlap, indicating that they share some elements, as shown in the following figure. For example, the word *topaz* is in both sets. The figure also contains a circle for the set START-j. It doesn't overlap the circle for START-t because no word lies in both sets.

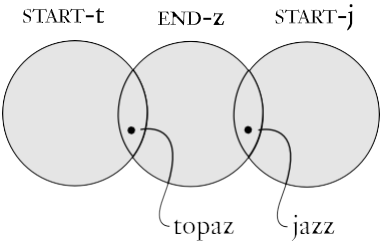

FIGURE 3 Overlapping circles indicate common elements

The next two Venn diagrams depict the union and intersection of sets A and B.

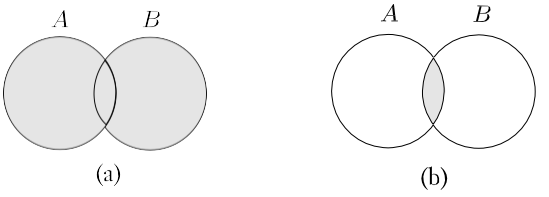

FIGURE 4 Diagrams for (a) A U B and (b) A ∩ B

SEQUENCES AND TUPLES

A ***sequence*** of objects is a list of these objects in some order. We usually designate a sequence by writing the list within parentheses. For example, the sequence 7, 21, 57 would be written (7, 21, 57).

The order doesn't matter in a set, but in a sequence it does. Hence (7, 21, 57) is not the same as (57, 7, 21). Similarly, repetition does matter in a sequence, but it doesn't matter in a set. Thus (7, 7, 21, 57) is different from both of the other sequences, whereas the set {7, 21, 57} is identical to the set {7, 7, 21, 57}.

As with sets, sequences may be finite or infinite. Finite sequences often are called ***tuples***. A sequence with k elements is a ***k-tuple***. Thus (7, 21, 57) is a 3-tuple. A 2-tuple is also called an ***ordered pair***.

Sets and sequences may appear as elements of other sets and sequences. For example, the ***power set*** of A is the set of all subsets of A. If A is the set {0, 1}, the power set of A is the set {∅, {0}, {1}, {0, 1}}. The set of all ordered pairs whose elements are 0s and 1s is {(0, 0), (0, 1), (1, 0), (1, 1)}.

If A and B are two sets, the ***Cartesian product*** or ***cross product*** of A and B, written $A \times B$, is the set of all ordered pairs wherein the first element is a member of A and the second element is a member of B.

EXAMPLE 1

If $A = \{1, 2\}$ and $B = \{x, y, z\}$,

$$A \times B = \{(1, x), (1, y), (1, z), (2, x), (2, y), (2, z)\}.$$

We can also take the Cartesian product of k sets

$A_1, A_2, \ldots, A_k$, written $A_1 \times A_2 \times \cdots \times A_k$.

It is the set consisting of all k-tuples $(a_1, a_2, \ldots, a_k)$ where $a_i \in A_i$.

EXAMPLE 2

If A and B are as in Example 1,

$$A \times B \times A = \{(1, x, 1), (1, x, 2), (1, y, 1), (1, y, 2), (1, z, 1), (1, z, 2),$$
$$(2, x, 1), (2, x, 2), (2, y, 1), (2, y, 2), (2, z, 1), (2, z, 2)\}.$$

If we have the Cartesian product of a set with itself, we use the shorthand

$$\overbrace{A \times A \times \ldots . A}^{K} = A^k$$

EXAMPLE 3

The set N^2 equals N $\times$ N. It consists of all ordered pairs of natural numbers. We also may write it as $\{(i, j) \mid i, j \geq 1\}$.

FUNCTIONS AND RELATIONS

Functions are central to mathematics. A ***function*** is an object that sets up an input–output relationship. A function takes an input and produces an output. In every function, the same input always produces the same output. If *f* is a function whose output value is *b* when the input value is *a*, we write ***f*(*a*) = *b***.

A function also is called a ***mapping***, and, if $f(a) = b$, we say that *f* maps *a* to *b*. For example, the absolute value function *abs* take a number *x* as input and returns *x.*

if *x* is positive and −*x* if *x* is negative. Thus $abs(2) = abs(-2) = 2$. Addition is another example of a function, written *add*.

The input to the addition function is an ordered pair of numbers, and the output is the sum of those numbers.

The set of possible inputs to the function is called its ***domain***. The outputs of a function come from a set called its ***range***. The notation for saying that *f* is a function with domain *D* and range *R* is $\boldsymbol{f: D \longrightarrow R}$.

In the case of the function *abs*, if we are working with integers, the domain and the range are Z, so we write $abs : Z \longrightarrow Z$.

In the case of the addition function for integers, the domain is the set of pairs of integers $Z \times Z$ and the range is Z, so we write $add : Z \times Z \longrightarrow Z$.

Note that a function may not necessarily use all the elements of the specified range. The function *abs* never takes on the value −1 even though $-1 \in Z$. A function that does use all the elements of the range is said to be ***onto*** the range.

We may describe a specific function in several ways. One way is with a procedure for computing an output from a specified input. Another way is with a table that lists

all possible inputs and gives the output for each input.

EXAMPLE 4

Consider the function f : {0, 1, 2, 3, 4}–→ {0, 1, 2, 3, 4}.

n	f (n)
0	1
1	2
2	3
3	4
4	0

This function adds 1 to its input and then outputs the result modulo 5. A number modulo m is the remainder after division by m.

For example, the minute hand on a clock face counts modulo 60. When we do modular arithmetic, we define Zm = {0, 1, 2,...,m − 1}. With this notation, the aforementioned function f has the form f : Z5 –→ Z5.

GRAPHS

An ***undirected graph***, or simply a ***graph***, is a set of points with lines connecting some of the points. The points are called ***nodes*** or ***vertices***, and the lines are called ***edges***, as shown in the following figure.

FIGURE 5 Examples of graphs

The number of edges at a particular node is the ***degree*** of that node. In Figure 5(a), all the nodes have degree 2. In Figure 5(b), all the nodes have degree 3. No more than one edge is allowed between any two nodes. We may allow an edge from a node to itself, called a ***self-loop***, depending on the situation.

In a graph *G* that contains nodes *i* and *j*, the pair (*i, j*) represents the edge that connects *i* and *j*. The order of *i* and *j* doesn't matter in an undirected graph, so the pairs (*i, j*) and (*j, i*) represent the same edge. Sometimes we describe undirected edges with unordered pairs using set notation as in {*i, j*}. If *V* is the set of nodes of *G*

and E is the set of edges, we say $G = (V, E)$. We can describe a graph with a diagram or more formally by specifying V and E. For example, a formal description of the graph in Figure 5(a) is
$(\{1, 2, 3, 4, 5\}, \{(1, 2), (2, 3), (3, 4), (4, 5), (5, 1)\})$,and a formal description of the graph in Figure 5(b) is $(\{1, 2, 3, 4\}, \{(1, 2), (1, 3), (1, 4), (2, 3), (2, 4), (3, 4)\})$.

Graphs frequently are used to represent data. Nodes might be cities and edges the connecting highways, or nodes might be people and edges the friendships between them. Sometimes, for convenience, we label the nodes and/or edges of a graph, which then is called a ***labeled graph***. Figure 6 depicts a graph whose nodes are cities and whose edges are labeled with the dollar cost of the cheapest nonstop airfare for travel between those cities if flying nonstop between them is possible.

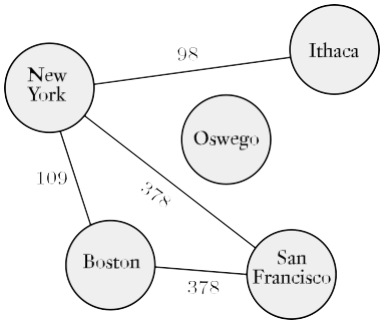

FIGURE 6 Cheapest nonstop airfares between various cities

We say that graph G is a ***subgraph*** of graph H if the nodes of G are a subset of the nodes of H, and the edges of G are the edges of H on the corresponding nodes. The following figure shows a graph H and a subgraph G.

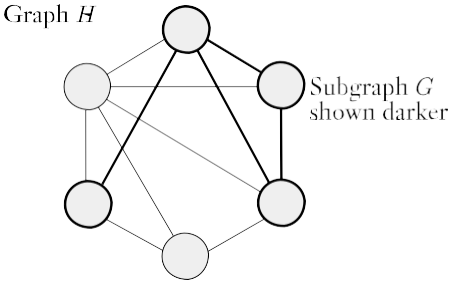

FIGURE 7Graph *G* (shown darker) is a subgraph of *H*

A ***path*** in a graph is a sequence of nodes connected by edges. A ***simple path*** is a

path that doesn't repeat any nodes. A graph is ***connected*** if every two nodes have a path between them. A path is a ***cycle*** if it starts and ends in the same node. A ***simple cycle*** is one that contains at least three nodes and repeats only the first and last nodes. A graph is a ***tree*** if it is connected and has no simple cycles, as shown in Figure 8 A tree may contain a specially designated node called the ***root***. The nodes of degree 1 in a tree, other than the root, are called the ***leaves*** of the tree.

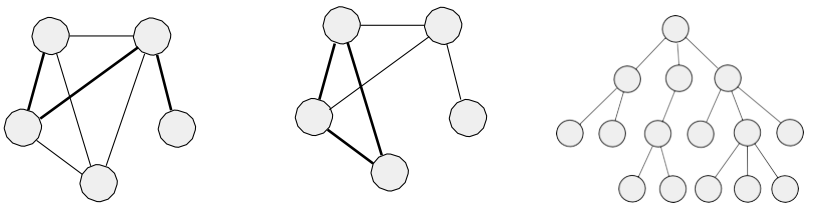

FIGURE 8 (a) A path in a graph, (b) a cycle in a graph, and (c) a tree

A ***directed graph*** has arrows instead of lines, as shown in the following figure. The number of arrows pointing from a particular node is the ***outdegree*** of that node, and the number of arrows pointing to a particular node is the ***indegree***.

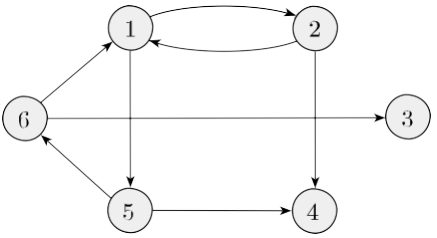

FIGURE 9 A directed graph

In a directed graph, we represent an edge from *i* to *j* as a pair (*i,j*). The formal description of a directed graph *G* is (*V, E*), where *V* is the set of nodes and *E* is the set of edges. The formal description of the graph in Figure 9 is ({1,2,3,4,5,6}, {(1,2), (1,5), (2,1), (2,4), (5,4), (5,6), (6,1), (6,3)}).

A path in which all the arrows point in the same direction as its steps is called a ***directed path***. A directed graph is ***strongly connected*** if a directed path connects every two nodes. Directed graphs are a handy way of depicting binary relations. If *R* is a binary relation whose domain is $D \times D$, a labeled graph $G = (D, E)$ represents *R*,

where $E = \{(x, y) \mid xRy\}$.

STRINGS AND LANGUAGES

Strings of characters are fundamental building blocks in computer science. The alphabet over which the strings are defined may vary with the application. For our purposes, we define an ***alphabet*** to be any nonempty finite set. The members of the alphabet are the ***symbols*** of the alphabet. We generally use capital Greek letters Σ and Γ to designate alphabets and a typewriter font for symbols from an alphabet. The following are a few examples of alphabets.

$\Sigma 1 = \{0, 1\}$

$\Sigma 2 = \{a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z\}$

$\Gamma = \{0, 1, x, y, z\}$

A string over an alphabet is a finite sequence of symbols from that alphabet, usually written next to one another and not separated by commas. If Σ1 = {0,1}, then 01001 is a string over Σ1. If Σ2 = {a, b, c,..., z}, then abracadabra is a string over Σ2. If w is a string over Σ, the length of w, written |w|, is the number of symbols that it contains. The string of length zero is called the empty string and is written ε. The empty string plays the role of 0 in a number system. If w has length n.

we can write w = w1w2 ⋯ wn where each wi ∈ Σ. The reverse of w, written wR, is the string obtained by writing w in the opposite order (i.e., wnwn−1 ⋯ w1). String z is a substring of w if z appears consecutively within w. For example, cad is a substring of abracadabra.

If we have string x of length m and string y of length n. the concatenation of x and y, written xy, is the string obtained by appending y to the end of x, as in x1 ⋯ xmy1 ⋯ yn. To concatenate a string with itself many times, we use the superscript notation x^k to mean

$$\underbrace{xx \cdots x}_{K}.$$

The lexicographic order of strings is the same as the familiar dictionary order. We'll occasionally use a modified lexicographic order, called **shortlex order** or simply **string order**, that is identical to lexicographic order, except that shorter strings precede longer strings. Thus, the string ordering of all strings over the alphabet {0,1} is

(ε, 0, 1, 00, 01, 10, 11, 000,.. .). Say that string x is a **prefix** of string y if a string z exists where xz = y, and that x is a **proper prefix** of y if in addition x ≠ y. A language is a set

of strings. A language is **prefix-free** if no member is a proper prefix of another member.

BOOLEAN LOGIC

Boolean logic is a mathematical system built around the two values TRUE and FALSE. Though originally conceived of as pure mathematics, this system is now considered to be the foundation of digital electronics and computer design. The values TRUE and FALSE are called the ***Boolean values*** and are often represented by the values 1 and 0. We use Boolean values in situations with two possibilities, such as a wire that may have a high or a low voltage, a proposition that may be true or false, or a question that may be answered yes or no. We can manipulate Boolean values with the ***Boolean operations***. The simplest Boolean operation is the ***negation*** or **NOT** operation, designated with the symbol ¬. The negation of a Boolean value is the opposite value. Thus ¬0 = 1 and ¬1 = 0. We designate the ***conjunction*** or **AND** operation with the symbol ∧. The conjunction of two Boolean values is 1 if both of those values are 1. The ***disjunction*** or **OR** operation is designated with the symbol ∨.

The disjunction of two Boolean values is 1 if either of those values is 1. We summarize this information as follows.

0 ∧ 0 = 0	0 ∨ 0 = 0	¬0 = 1
0 ∧ 1 = 0	0 ∨ 1 = 1	¬1 = 0
1 ∧ 0 = 0	1 ∨ 0 = 1	
1 ∧ 1 = 1	1 ∨ 1 = 1	

We use Boolean operations for combining simple statements into more complex Boolean expressions, just as we use the arithmetic operations + and × to construct complex arithmetic expressions. For example, if P is the Boolean value representing the truth of the statement “the sun is shining” and Q represents the truth of the statement “today is Monday”, we may write P ∧ Q to represent the truth value of the statement “the sun is shining and today is Monday” and similarly for P ∨ Q with and replaced by or. The values P and Q are called the operands of the operation.

Several other Boolean operations occasionally appear. The eXclusive-OR (XOR), operation is designated by the ⊕ symbol and is 1 if either but not both of its two operands is 1. The equality operation, written with the symbol ↔, is 1 if both of its operands have the same value. Finally, the implication operation is designated by the symbol → and is 0 if its first operand is 1 and its second operand is 0; otherwise, → is 1. We summarize this information as follows.

$0 \oplus 0 = 0$ $0 \leftrightarrow 0 = 1$ $0 \rightarrow 0 = 1$

$0 \oplus 1 = 1$ $0 \leftrightarrow 1 = 0$ $0 \rightarrow 1 = 1$

$1 \oplus 0 = 1$ $1 \leftrightarrow 0 = 0$ $1 \rightarrow 0 = 0$

$1 \oplus 1 = 0$ $1 \leftrightarrow 1 = 1$ $1 \rightarrow 1 = 1$

We can establish various relationships among these operations. In fact, we can express all Boolean operations in terms of the AND and NOT operations, as the following identities show. The two expressions in each row are equivalent. Each row expresses the operation in the left-hand column in terms of operations above it and AND and NOT.

$P \vee Q$ $\neg(\neg P \wedge \neg Q)$

$P \rightarrow Q$ $\neg P \vee Q$

$P \leftrightarrow Q (P \rightarrow Q) \wedge (Q \rightarrow P)$

$P \oplus Q$ $\neg(P \leftrightarrow Q)$

The distributive law for AND and OR comes in handy when we manipulate Boolean expressions. It is similar to the distributive law for addition and multi- plication, which states that $a \times (b + c) = (a \times b) + (a \times c)$. The Boolean version comes in two forms:

$P \wedge (Q \vee R)$ equals $(P \wedge Q) \vee (P \wedge R)$, and its dual

$P \vee (Q \wedge R)$ equals $(P \vee Q) \wedge (P \vee R)$.