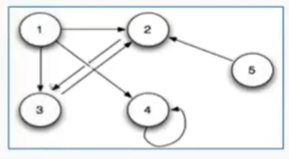
# **Section 1.4: Graphs**

**Definitions**

* A graph consists of a set of vertices (nodes) connected by edges.
  + Can be drawn in different ways
  + Two graphs are equivalent if vertices are adjacent to each other the same
* Any pair of vertices connected by an edge are said to be adjacent.
* A graph is *n*-colorable if there is an assignment of *n* colors to its vertices such that any two adjacent vertices have distinct colors.
* A graph is planar if it can be drawn on a plane so that no edges intersect.
* A directed graph is a graph where each edge points in one direction.
  + This does NOT include a double-edged arrow UNLESS it represents two individual arrows pointing both directions
  + Each vertex on a directed graph has an indegree (# of edges pointing to a vertex) and an outdegree (# of edges originating at that vertex)



* The degree of a vertex is the number of edges that it touches
  + Add two to the degree of a vertex if it has a loop (going to and from itself)
* A graph (V’, E’) is a subgraph of graph (V, E) if V’⊆V and E’⊆E.

**Representing a Graph as Data**

Vertices are a set. Ex. {1,2,3,4,5,6}

Edges are a set of sets. Ex. {{1,2}, {1,5}, {2,5}, {2,3}, {3,4}, {4,5}, {4,6}}

The inner sets are the two vertices that are adjacent to each other.

You may choose to represent this as a pair (V, E): ({1,2,3,4,5,6}, {{1,2}, {1,5}, {2,5}, {2,3}, {3,4}, {4,5}, {4,6}})

**Representing a Directed Graph as Data**

Vertices are a set. Ex. {1,2,3,4,5,6}

Edges are a set of pairs (2-tuples, because order matters!). Ex. {(1,2), (1,4), (1,3), (2,3), (3,2), (4,4), (5,2)} (i.e. Starting at 1, going to 2; starting at 1, going to 4; etc.)

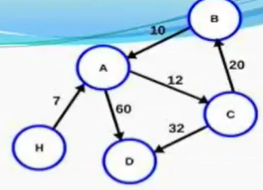
You may choose to represent this as an ordered pair (V, E): ({1,2,3,4,5,6}, {(1,2), (1,4), (1,3), (2,3), (3,2), (4,4), (5,2)})

**Weighted Graphs**

They have information attached to each edge (for example, distance between cities, cost, etc.).

* Now, represent edge as a 3-tuple (source, definition, weight)
* Can represent undirected as directed
* Each directed edge becoming a set of directed edges

Example:



Vertices: {A, B, C, D, H}

Edges: {(A,D,60), (A,C,12), (B,A,10), (C,B,20), (C,D,32), (H,A,7)}

(V, E) = ({A, B, C, D, H}, {(A,D,60), (A,C,12), (B,A,10), (C,B,20), (C,D,32), (H,A,7)})

**Questions**

*Graph or directed graph or neither?*

The process is as follows:

1. Cut out ones that have braces on the outside (all graphs are represented by pairs)
2. Cut out ones that have the first element (vertices) in parens (must have braces)
3. Cut out ones that have the second element (edges) in parens (must have braces)
4. Cut out ones that have the second element (edges) as a set (must be set in a set)
5. Cut out ones w/ vertices not matching the edges (for example, c is in the second element, but is not in the first)
6. If the second element has pairs () within the set, it’s a directed graph
7. If the second element has sets {} within the set, it’s a graph

* ({a,b}, {a,b}) Neither
* {{a,b}, {a,b}} Neither
* ({a,b}, {{a,b}}) Graph
* ({a,b}, (a,b)) Neither
* {{a,b}, (a,b)} Neither
* ({a,b,c}, (a,a), (a,c)) Neither
* ({a,b,c}, ((a,a), (a,c))) Neither
* ({a,b,c}, {(a,a), (a,c)}) Directed graph
* ({a,b}, {{a,b}, {a,a}}) Graph
* {{a,b}, {{a,b}, {a,a}}} Neither
* ({a,b}, {(a,b), (a,a)}) Directed graph
* ({a,b}, ({a,b}, {a,a})) Neither
* ({a,b}, ∅) Graph or directed Notes: No edges in the graph (∅)
* {{a,b}, ∅} Neither
* (∅, ∅) Graph or directed Note: No edges or vertices in graph
* {∅, ∅} Neither

**What’s wrong with these?**

*All of them have a,b in the vertices, but have c in the edges.*

* ({a,b}, {(a,b), (a,c)}) This one is fine except for the *c*.
* ({a,b}, {{(a,b)}, (a,c)}) The (a,b) is contained within a set, which is weird.
* ({a,b}, {{(a,b), (a,c)}}) The pairs are within a set within a set.
* ({a,b}, {((a,b), (a,c))}) The pairs are within another pair.

**Path in a Graph**

Informal: Sequence of vertices you travel through Ex. (4,5,2,3,4,6), a tuple (order matters)

Formal: A path from x0 to xn is a sequence of edges that you denote by a sequence of vertices x0, x1, …, xn such that there is an edge from x*i* - 1 to x*i* for .

**Cycles**

A cycle is a path whose beginning and ending vertices are equal and in which no edge occurs more than once. A graph with no cycles is called acyclic.

The length of the path x0, …, x*n* is the number *n* of edges.

You can have a path and a cycle on (un)directed graphs.

# 

# **Section 1.3: Strings & Languages**

**Definitions**

* A string is a finite ordered sequence of 0 or more symbols that are placed next to each other in juxtaposition. (A bunch of characters, or symbols, placed next to each other.)
  + Ex. Strings over alphabet A include a, bbm, aabb, abos, obama, usa, rowan
  + Underscores also count as symbols in strings
  + How are you are three strings for the purposes of this class (spaces matter)
  + “Lambda” represents the empty string, it is a string over any alphabet ()
  + String length is the number of elements in that string (Notation: | |)
    - Ex. | obama | = 5, || = 0
* An alphabet is a finite set of symbols that you can use to make strings. This includes letters and emojis. Notation:
  + Ex. Let A = {a,b,m,n,o,r,s,u,w}. A is an alphabet.
  + There may be an infinite number of strings that can come from an alphabet, but the resulting string should always be finite.
* The set of natural numbers (N) means {0,1,2,3,4,5,6,...}
* The set of integers (Z) means {...-3,-2,-1,0,1,2,3,...}

**Problems from the book: Page 58 #8**: Write down all possible strings of length 2 over the set A = {a,b,c}.

Basically, you have to find all the combinations possible of strings of length 2.

aa, ab, ac, ba, bb, bc, ca, cb, cc

**String Concatenation**

You can concatenate two strings by placing them next to each other, creating a new string. For example,

Concatenate obama && USA == obamausa

Concatenate rowan && ^ == rowan

Concatenate rowan && ☺ == rowan☺

**Definition: Language**

A language is a set of strings. This is NOT finite. It can be of infinite length or even empty. For example,

{a,b,ab,aabb,ba} is a language over the alphabet {a,b}

{^} is also a language over the alphabet {a,b}

{a,^} is also a language over the alphabet {a,b}

∅ is also a language over the alphabet {a,b}

For the last example, the empty set has NO elements (think { }) while the lambda has 1 (an empty string).

**Definitions of A\***

If A is an alphabet, then A\* is a special set which is the set of all strings over the alphabet A. For example, suppose A = {a,b}.

1. Is A\* a language? (*Should be able to figure out the answer without knowing what’s in A\*.*) Yes because a language means a set of strings, and A\* by definition is a set of strings.
2. How big is A\*? There’s an infinite number of strings over the alphabet.
3. What is A\*? (*Process: Always start out with the lambda, or the empty string. Then ask yourself what are the elements of the string of length 1, 2, etc.*) {^,a,b,aa,ab,ba,bb,...}

**Notation: sn**

If *s* is a symbol, then sn is *n* copies of *s*. Suppose *s* = ☺.

* s0 = ^
* s2 = ☺☺
* s4 = ☺☺☺☺

**Using set notation to represent languages**

The set of all symbols a*n*such that *n* is in the set of natural numbers.

{^, a, aa, aaa, aaaa, …}

The set of all symbols a concatenated with b*n* such that *n* is in the set of

natural numbers.

{a, ab, abb, abbb, …}

The set of all symbols a*n* concatenated with b*n* such that *n* is in the set of

natural numbers. (*Note: The number of a’s and b’s must match since they*

*have the same variable as its exponent, or* n.)

{^, ab, aabb, aaabbb, aaaabbbb, …}

The set of all symbols a*n* concatenated with b*m* such that *n* and *m*

are in the set of natural numbers. (*Note: Notice how* a *and* b *have*

*different exponent variables this time. Each string has 0 or more*

*a’s and 0 or more b’s, and the a’s come before the b’s.*)

{^, a, b, aa, ab, bb, aaa, aab, …}

The set of all symbols a*n* such that *n* is on the set of natural

numbers unioned with the set of all symbols b*n* such that *n*

is on the set of natural numbers. (*Note:* *You never mix up*

*the number of a’s and the number of b’s here because*

*they’re not concatenated.* a0 *and* b0 *both produce lambdas,*

*but it’s redundant to put both in.*)

{^, a, b, aa, bb, …}

**The Product of Two Languages**

If L and M are languages, then the product of them is another language (i.e. Take one string from L, one string from M, concatenate): LM = }

For example, suppose L = {^, abb, b} and M = {a, b}. LM == {a, b, abba, abbb, ba, bb}

* Note that concatenation of a string and a lambda is just the string.
* The answer is all possible combinations of the two sets’ strings, but ordered (L first).

Question: Does LM always equal ML for any pair of languages? Use the same L and M definitions from the previous example. No, ML == {a, b, aabb, babb, ab, bb}.

Question: Are there any circumstances where LM can equal ML?

* Easy: L == M
* Also: L or M == ∅
* Also: L or M == {^}

**Problem from the book: Page 60, #10**: Use your wits to solve each of the following language equations for the unknown language (AKA solve for L):

* L{a, b} = {a, baa, b, bab}
  + Firstly, | L | must be 2 because a and b are by themselves (the strings with length 1) and the strings with length 3 require another string to be combined with the symbols a and b from the given language. This also can be found by counting the number of strings in the final result (4) and dividing it by the number of strings in the given language (2) to get 2.
  + Secondly, ^L because strings a and b are present in the product, meaning they were concatenated with lambdas.
  + Thus, L = {^, ba}.
* L{^, a} = {^, a, b, ab, ba, aba}
  + Firstly, there has to be a lambda in L because the only way to get lambda as part of the product of languages is if you concatenate 2 lambdas.
  + Secondly, there must be a “b” in L because there is no b present in the given language.
  + Thirdly, there must be an “ab” in L because there is no way to get that string in the order of the concatenation. L is always the first part of the resulting string because that is the order it was multiplied. Plus, it makes sense when you concatenate it with “a” to get the last resulting string.
  + Thus, L = {^, b, ab}.
* L{b, ^, ab} = {abb, ab, abab, bab, b, bb}
  + Firstly, there must be a “b” in L because in the resulting set of strings, there is a “b” by itself. This is NOT a result of the “b” in the given language concatenated with a lambda, or else the lambda in the given language and the supposed other lambda would produce a lambda in the resulting set. It also makes sense due to the resulting string “bb”.
  + Secondly, there must be an “ab” in L because of similar reasons as the last bullet point. Also notice “abab” in the resulting set.
  + Thus, L = {ab, b}.

**Product of a language with itself**

Before: If s is a *symbol*, then is *n* copies of s.

Now: If L is a *language*, then is L multiplied by itself *n* times.

The special case is that is {^}.

For example, suppose L = {a, bb}. What is ?

First, figure out what is: {a, bb}{a, bb} = {aa, abb, bba, bbbb}.

Then multiply the result with itself again: {aa, abb, bba, bbbb}{a, bb} = {aaa, aabb, abba, abbbb, bbaa, bbabb, bbbba, bbbbbb}

**Trick Questions**

Let L = {ab, cc}, M = {^}, N = ∅, and P = {a, ^}.

* Are L, M, and N all languages? Yes; remember, a language is a set of strings. L has strings, M has a lambda, which is an empty string contained in a set, and N is an empty set with nothing in it. This can be considered a set of strings.
* What is LM? Since M is just a lambda, LM = {ab, cc}.
* What is LN? Remember for the product of two languages, take one string from both languages and concatenate it. However, there is NO string in N, so there’s no string to concatenate with, thus resulting in LN = {∅}.
* WHat is MN? See the previous explanation for more details. MN = {∅}.
* What is ? First, what is ? {a, ^}{a, ^} == {aa, a, ^}. Then, multiply the resulting set by P: {aa, a, ^}{a, ^} == {aaa, aa, a, ^}.

**Problem from the book: Page 60, #9:** Let L = {^, abb, b} and M = {bba, ab, a}.

* ML = {bba, ab, a}{^, abb, b} == {bba, bbaabb, bbab, ab, ababb, abb, a, aabb}.
* = L = {^, abb, b}

**Closure of a Language: L\***

Before: A was an alphabet; A\* is the set of all strings over an alphabet

Now: L is a language; L\* = {} (the set of all possible concatenations of strings from L).

For example,

* Suppose L is the language {a}; what is L\*? L\* = {^, a, aa, aaa, aaaa, …}

(*Note:* *This is the same as {a}\* if you treat this as an alphabet.*)

* Suppose L is the language {a, b}; what is L\*? L\* = {^, a, b, aa, ab, ba, bb, aaa, aab, …}

(*Note: Notice the ^ is the , a and b are , and the others are a product of the other possible concatenations.*)

* Suppose L is the language {^}; what is L\*? L\* = {^}

(*Note: The only possible concatenation of lambdas is the lambda itself.*)

* Suppose L is the language ∅; what is L\*? L\* = {^}

(*Note: Remember that any language is equal to this. In addition, multiplying an empty set with itself will result in an empty set (nothing) so there are no other possible concatenations.*)

**Positive Closure of a Language: L+**

Before: L is a language; L\* = {}

Now: L is a language; L+ **=** {}; same as L\* but don’t include the 0

For example,

* Suppose L is the language {a}; what is L+? L+ = {a, aa, aaa, aaaa, …}
* Suppose L is the language {^, a}; what is L+? L+ = {^, a, aa, aaa, aaaa, …}

(*Note: This DOES have lambda in it because L1 includes lambda.*)

* Suppose L is the language {a, b}, what is L+? L+ = {aa, ab, ba, bb, aaa, aab, aba, abb, …}
* Suppose L is the language {^}, what is L+? L+ = {^}

(*Note: Remember, this is because L1 includes lambda.*)

* Suppose L is the language ∅; what is L+? L+ = ∅

(*Note: Every concatenation of the empty set IS the empty set. The union of all of those is still the empty set.*)

**From Book: Properties of Closure (for Languages)**

*Make sure they make sense to you! LOOK INTO THEM MORE LATER*

{^}\* = {^}

∅\* = {^}

^ iff L+ = L\*

You know that if lambda is in L, then lambda is definitely in L+ because it contains L1 and lambda is definitely in that. The only difference between L+ and L\* is the potential absence of a lambda.

On the other hand, if you know that L+ = L\*, then you know lambda is in L+ because the only way you know lambda is in L+ is if lambda is in L (because L1 is basically L).

L\* = L\*L\*

L\* = (L\*)\*

Let L and M both be languages:

(L\*M\*)\* = (LM)\*

Test this out yourself with L = {a} and M = {b} that this is true.

(L\*M\*)\* = (L\*M\*)\*

**Problem from book: Page 60, Problem 10**: Let L and M be 2 languages. For each of the following languages, describe the general structure of a string *x* by writing it as a concatenation of strings that are either in L or M. (i.e. The result of the concatenations is string *x*. What is *x*?)

LM\*: Either *x* is just a string from L or *x* is a string from L concatenated with one or more strings from M // Either or where and M.

For the first possibility, remember, M\* means the set of all concatenations on M, which most definitely includes ^ (lambda) because is always lambda. If *x* is an element of L, the concatenation of *x* and lambda, the resulting string is *x*.

For the second possibility, *x* is some string where the first part is a part of language L and all the other parts are elements of M.

{LM)\*: Either *x* is the empty string or *x* is the concatenation of strings from LM. // *x* = ^ or *x* = ...where .

For the first possibility, when you take the star (\*) of anything, you know that lambda is in there.

If there are any intersections within L and M, then *x* is the set of strings where each string is an element of that intersection (and the 1 to *n* is the definition of the star since it’s all possible concatenations forever).

(LM)\*: Either *x* is lambda (empty string) or *x* is the set of strings of the concatenation of things from L and M stuck together. // *x* = ^ or *x* = ()...() where and .

For the first possibility, when you take the star (\*) of anything, you know that lambda is in there.

Secondly, the star indicates that the concatenation continued. (Remember, the formal answer is the one that is needed.) Each pair consists of something from L and something from M.

**STUFF NEVER USED AS SYMBOLS (ELEMENTS IN ALPHABETS):**

| 𝚲 | Capital lambda is a special symbol which will always mean the empty string |
| --- | --- |
| ∅ | This is a special symbol which will always represent the empty set (it’s just a shorthand for two curly braces: { } |
| Groups of English Letters | We use *individual* English letters as symbols all the time - here’s an example of an alphabet with 3 symbols: {a, b, c}.  We will never squish multiple English letters together to represent a single symbol - so, for example, the following is definitely NOT an alphabet:  {a, bc}  Because “bc” is a string of two symbols together. |
| Normal set and tuple symbols | Things like parentheses, curly braces, and commas will always mean what you think they mean - we’ll never say “oh, let’s talk about the alphabet that has just one symbol - a comma - in it” - that would get too confusing. |

**Some questions**

Let A = q, r, s, qr, rs, qs.

1. Could A be an alphabet? No, there is no curly braces
2. Could A be a string? No, because the commas are in there, and commas can’t be strings
3. Could A be a language? No, there is no curly braces

Let B = aeiou.

1. Could B be an alphabet? No, there are no curly braces
2. Could B be a string? Yes, because it is a finite sequence of symbols
3. Could B be a language? No, there are no curly braces

Let C = {q, r, s, qr, rs, qs}.

1. Could C be an alphabet? No, because qr, rs, and qs are two symbols squished together, and it is not a set of symbols
2. Could C be a string? No, because it has curly braces
3. Could C be a language? Yes, because it is a set and each element is a string

Let D = {aeiou}.

1. Could D be an alphabet? No, because aeiou is a bunch of symbols squished together
2. Could D be a string? No, because it has curly braces
3. Could D be a language? Yes, because it is a set and has a string

| Rowan Rocks |’s size is 11 (1 string)

| {Rowan, Rocks} |’s size is 2 (2 strings)

| {R, o, w, a, n, R, o, c, k, s} | 8 (duplicates are R and O)