

David Keil, CSCI 400 Artificial Intelligence
Framingham State University

Mathematics and computer-science background

1. Set theory and logic
2. Arrangements of data
3. Algorithms and interaction
4. Probability and statistics

1. Set theory and logic

\in, \notin	set membership, nonmembership
\subseteq, \subset	subset, proper subset
\cup, \cap	union, intersection
$A \times B$	Cartesian product of sets A and B ; a set of ordered pairs

Strings:

Σ (sigma)	usually a finite alphabet
Σ^*	the set of all strings over Σ
$L \subseteq \Sigma^*$	a language (set of strings)
λ (lambda)	null string, ""

Sets

- A *set* is an aggregation of items, real, imaginary, or abstract, taken as a whole
- Sets may be defined by
 - Enumeration: $A = \{1, 2, 4\}$
 - Membership predicate
 $A = \{x \mid is_odd(x)\}$
- Sets are defined without respect to order:
 $\{1, 2, 4\} = \{2, 1, 4\}$
- A set contains no duplicates:
 $\{1, 2, 4\} = \{1, 2, 4, 2\}$

Set notation

\in	set membership
\notin	nonmembership
\subseteq, \subset	subset, proper subset
\cup, \cap	union, intersection
$A \times B$	Cartesian product of sets A and B ; a set of ordered pairs
$A = \{a, b, c\}$	specification by enumeration
$A = \{x \mid P(x)\}$	specification by predicate (P)

Union, intersection, complement

- *Membership*: “ $x \in A$ ” means x is an element of set A
- *Union* of A and B :

$$(A \cup B) = \{ x \mid x \in A \vee x \in B \}$$
- *Intersection* of A and B :

$$(A \cap B) = \{ x \mid x \in A \wedge x \in B \}$$
- *Relative complement* of A w.r.t. B :

$$(A - B) = \{ x \mid x \in A \wedge x \notin B \}$$
- Generalized union (intersection, \cap):

$$\cup \{ A, B, C \} = A \cup B \cup C$$

Inclusion (subsets)

- $A \subseteq B$ (set A is a subset of set B) means that any member of A is a member of B
- Strict inclusion: $A \subset B$ means $A \subseteq B \wedge A \neq B$
- If $A \subseteq B$ then $B \supseteq A$ (B is a superset of A)
- For every set A , $A \subseteq A$
- The *null set* \emptyset
 - has no members
 - is a subset of all sets
- The *power set* of A , written $\wp(A)$ or 2^A , is the set of all subsets of A

Cartesian product of two sets:

- The set of ordered pairs defined by selecting one member of each

- Example:*

$$\{1,2\} \times \{a,b\} = \{(1,a), (1,b), (2,a), (2,b)\}$$

	a	b
1	(1,a)	(1,b)
2	(2,a)	(2,b)

Relations

- Relation: a subset of a Cartesian product

- Example:*

$R = \{(1,a), (2,b)\}$ is a relation

$$R \subseteq \{1,2\} \times \{a,b\}$$

R	a	b
1	T	F
2	F	T

Example: $>$ is a relation

$>$	1	2	3	4
1	F	F	F	F
2	T	F	F	F
3	T	T	F	F
4	T	T	T	F

- Because $2 > 1$; $3 > 1$; $3 > 2$, etc.
- A relation is a set of ordered pairs:

$\{ (2,1), (3,1), (3,2), (4,1), (4,2), (4,3) \}$

Functions

- *Function*: a relation of which each left-hand member of a tuple is in not more than one tuple (maps to a unique value)
- *Examples*: *EVEN* is a function;
 $>$ is a relation but not a function
- Every function has a
 - *Domain*: set of values mapped from
 - *Range*: set of values mapped to
- Computation of a function takes parameter as input, return value as output

Logic

- Languages of logic have *syntax* (form), *semantics* (truth values of sentences)
- Truth of a sentence with variables is relative to its *model* (set of variable assignments)
- *Entailment*: $\alpha \models \beta$ (in every model where α is true, β is true)
- Example: $\text{not}(\text{breeze in } (1,1)) \models \text{not}(\text{pit in } (1,2))$ and $\text{not}(\text{pit in } (2,2))$
- *Inference* is finding a desired instance of entailment

Propositional logic

- Propositional logic (propositional calculus) is a Boolean algebra with the values *True* and *False* and the operations \neg , \wedge , \vee , \rightarrow
- It is a *language of assertions* that each evaluate to *true* or *false*, including
 - *true, false*
 - Symbols (p, q, r, \dots) that may stand for assertions such as “roses are red”
 - Negation: $\neg p$
 - Conjunction: $p \wedge q$
 - Disjunction: $p \vee q$
 - Implication: $p \rightarrow q \Leftrightarrow \neg q \vee p$

Semantics of propositional logic

- Let p, q , be formulas
- Wherever p is true, $\neg p$ is false
- $(p \wedge q)$ is true when both p and q are true
- $(p \vee q)$ is true when either p or q or both are true
- $(p \rightarrow q)$ is true when p is false or q is true
- $(p \Leftrightarrow q)$ or $(p \equiv q)$ or $(p \text{ iff } q)$ is true when p and q have the same values under all interpretations (truth assignments)

Interpretations

- An interpretation of a set of formulas is an assignment of truth values to the symbols
- *Example:* An interpretation of $(p \wedge \neg(p \vee q))$ is $p = \text{true}, q = \text{false}$. Under this interpretation, $(p \wedge \neg(p \vee q))$ is false
- A formula is *satisfiable* if it has an interpretation under which it is true

Truth tables

- A truth table lists the values of a formula under all possible interpretations

- *Example:*

p	q	$(p \wedge \neg q)$
f	f	f
f	t	f
t	f	t
t	t	f

- Formulas with the same truth table are *equivalent*:
for formulas ϕ and φ ,
 $(\phi \Leftrightarrow \varphi)$ or $(\phi \equiv \varphi)$ or $(\phi \text{ iff } \varphi)$

Problems in propositional logic

- *Evaluate*: given a particular set of variable assignments and a formula, evaluate formula
- *Satisfiability (SAT)*: Given a formula ϕ , does a set of variable assignments exist that satisfies ϕ (makes ϕ true)?
- *Validity*: does ϕ hold under *all* variable assignments?
- *Examples*: $(p \wedge \neg p)$ is not satisfiable;
 $(p \vee \neg p)$ is satisfiable and valid;
 $p \wedge q$ evaluates to *true* if p, q are *true*

Some inference rules

- *Modus Ponens*: $(p \wedge (p \rightarrow q)) \rightarrow q$
- *Modus Tollens*: $((p \rightarrow q) \wedge \neg q) \rightarrow \neg p$
- $p \rightarrow q$ is defined as $\neg p \vee q$
- *Note*: if p is false, then $p \rightarrow q$ holds vacuously for any q , whether true or false

3. Arrangements of data

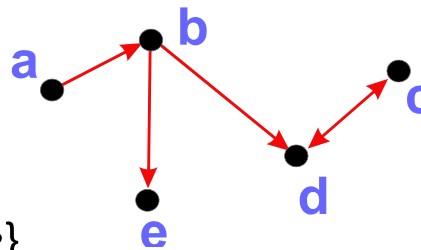
- Linear structures
 - Array (random access)
 - Linked list (sequential access)
- Restricted access
 - Stack (LIFO, push/pop)
 - Queue (FIFO, enqueue/dequeue)
 - Priority queue (insert, extract-min)
- Graph $G = \langle V, E \rangle$
 - Directed, undirected
 - Sparse, dense
 - Weighted
- Tree (connected acyclic graph)
 - Root, parent, child, leaf

Collections

- *Set* (distinct, unordered)
- *Multiset* (unordered)
- *Array*
- *Dictionary* (search, insertion, deletion)
- *List* (ordered)
- *Special-access* (stack, queue, priority queue)
- *Graph* (set of vertices, set of edges)
- Possible implementations of collections:
arrays, linked lists, trees

Relations and directed graphs

Graph: a set of vertices V and a set of edges E connecting the vertices. (E is a relation on V .)



$$V = \{a, b, c, d, e\}$$

$$E = \{(a, b), (b, e), (b, d), (c, d), (d, c)\}$$

Graphs

- A *graph* is a set of vertices and a set of directed or undirected edges
- Cycles, connectivity, and trees
- Graphs model communication relations, state transitions, precedence in time
- Algorithms exist to find paths, minimum-weight paths, minimal spanning trees
- *Implementations*: 2D array or array of linked lists

Applications of graphs

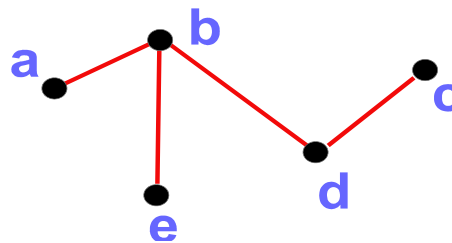
- Network topologies:
star, linear bus, ring
- Network routing algorithms
- Scheduling (e.g., a prerequisites diagram)
- Finite automata (vertices are states; edges are transitions between states)
- Trees to represent hierarchies of all kinds

Edges in a digraph

- An edge in a directed graph relates a vertex to another vertex
- *Example:* Who has sent email to whom? What intersections are connected by streets?
- A *relation* associates elements of sets, such as senders and recipients of email
- [pic]

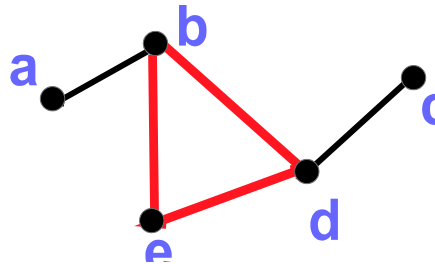
Paths

- *Path:* A sequence of pairwise adjacent vertices whose edges connect two vertices in a graph
- Path from a to d : (a,b,d)



Cycle:

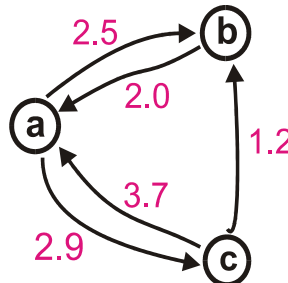
- A set of edges that form a path from one vertex to itself without repeating an edge
- Cycle below: (b, e, d)
- A graph with no cycles is *acyclic*



Weighted graphs

A weighted graph has an adjacency matrix of $(\text{reals} \cup \{\infty\})$

	a	b	c
a	0	2.5	2.9
b	2.0	0	∞
c	3.7	1.2	0

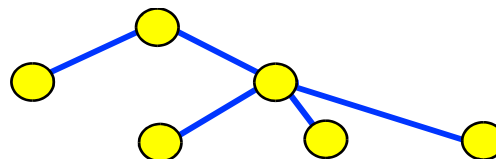


Weight can reflect a distance or cost

A tree is hierarchic

- Examples:
 - Family tree
 - Organizational hierarchy
 - Module hierarchy chart
 - Taxonomy
 - Parse tree
 - Disk directory
- A tree has a *root* at top, *leaves* at bottom
- A nonroot node has one *parent* node
- A nonleaf has one or more *child* nodes
- Every tree node is the root of a subtree

Tree: a connected acyclic graph



- Exactly one path joins any pair of vertices
- Removing an edge disconnects the tree
- Adding any edge creates a cycle
- Cardinalities of edges and vertices:
 $|E| = |V| - 1$

Parse trees

```

graph TD
    expression[expression] --- term1[term]
    expression --- op[addition operator]
    expression --- term2[term]
    term1 --- factor1[factor]
    factor1 --- 2[2]
    op --- plus[+]
    term2 --- factor2[factor]
    term2 --- op2[multiplication operator]
    term2 --- factor3[factor]
    factor2 --- 5[5]
    op2 --- asterisk[*]
    factor3 --- 3[3]
    
```

- Compilers build a program structure from tokens
- Root may be *program*
- Lexemes are leaves of tree
- Nonterminal syntax elements (e.g., *expression*, *factor*) are internal tree nodes or the root

D. Keil Special Topics: Artificial Intelligence 1/12 29

Height of a complete binary tree with n nodes is $O(\log_2 n)$

```

graph TD
    A(( )) --- B(( ))
    A --- C(( ))
    B --- D(( ))
    B --- E(( ))
    C --- F(( ))
    C --- G(( ))
    
```

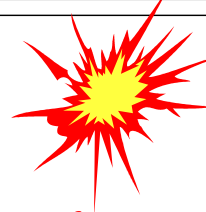
- ...because size n doubles with each level added
- Or, $2^{\text{height}-1} \leq n \leq 2^{\text{height}} - 1$

D. Keil Special Topics: Artificial Intelligence 1/12 30

4. Algorithms and interaction

Algorithm:

A precise plan
to solve a *functional*
problem in a finite number of steps



- Program designs use algorithms that convert pre-specified input to output
- Algorithms compute functions from *input* to *output*
- We may associate algorithms with the computation of Turing machines and programs

Algorithms compute functions

- Methods, procedures, and some high-level-language “functions” *implement* algorithms
- We distinguish functions from the algorithms that *compute* them
- Some functions are *uncomputable*
- All computable functions are *recursively definable*



Defining “algorithm”

- *Not algorithms:*
 - Online “algorithms” (non-functionality)
 - Probabilistic algorithms (nondeterminism)
 - Anytime algorithms (termination on cue)
 - Event loop (interleaved input/output)
- An *algorithmic procedure* is equivalent to
 - A run of a C++ or Java program with all input specified at beginning and all output at end
 - Computation of a Turing machine that always halts
 - Computation of a Random Access Machine
 - Derivation of a μ -recursive function

Problem: find rightmost zero

Last0(A)

// Precond: $|A| > 0$ and A contains a 0

$y \leftarrow -1$

$i \leftarrow 1$

while $i \leq |A|$

 If $A[i] = 0$

$y \leftarrow i$

$i \leftarrow i + 1$

return y

// Post: y stores location of rightmost 0 in A

Largest(A)

```
y ← A[1]
i ← 2
While i ≤ |A|
  if A[ i ] > y
    y ← A[ i ]
  i ← i + 1
return y
```

Linear search

Search(A, x)

```
y = false
for i ← 1 to |A|
  if A[ i ] = x
    y = true
return y
```

Product (x, y)

```
result ← 0
for i ← 1 to x
    result ← result + y
return result
```

Constraint satisfaction problems

Problem:

- A set x of variables x_1, x_2, \dots, x_n
- A set C of constraints C_1, C_2, \dots, C_m , that each specifies acceptable combinations of values for a certain subset of x
- A variable assignment to x that does not violate any of C is *consistent* or legal
- A variable assignment to all of x is *complete*
- *Solution:* a complete consistent variable assignment

General form of CSPs and solutions

- *Advantage* of formulating problems as CSPs: standard state representations enable generic transition functions, goal tests, heuristics
- *Form of state*: set of partial variable assignments
- *Initial state*: no assignments
- *Successor function*: assigns value to one variable while violating no constraint
- *Goal test*: variable assignments are complete
- Order-independence of variable assignments makes *backtracking* helpful

Optimization problems

Examples:

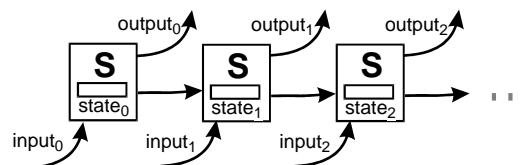
- *Closest pair*
Given a set of ordered pairs, points on a Cartesian plane, find two points that are closest together
- *Shortest path*: For vertices u, v , in *weighted* graph G , find *shortest* path from u to v
- *Bin packing*: Find a packing that minimizes the number of bins

Interactive problems

- A *service* (solving a sequential-interactive problem) is a series of responses to inputs
- A *protocol* defines rules for providing a service
- *Examples:*
 - Search engine
 - DBMS responses to database queries
 - Web server response to clicks
- *Note:* Response may reflect state of server, so that a service is not a series of algorithm executions generating output from input

Interactive computation

- Feature of computing today: Computation as an ongoing *service*, not assumed to terminate
- Must solve problems whose inputs cannot be completely specified a priori
- Dynamic input and output *during* computation
- *Persistence of state* between interaction steps
- *Environment* is an active partner in computation



Dynamic environments

- The function-optimization problem defined earlier is *static*, assumes f will be the same later as now
- Interaction is often in *dynamic* real-world environments, such as climate or ecosystem
- This increases the difficulty of verifying designs of reactive systems
- A separate, but related, category of environment is *stochastic* (imperfectly predictable)

Multi-stream-interactive problems

- A *mission* is a problem requiring services to multiple input streams, possibly under time constraints (quality of service)
- The sources of multiple input streams may interact among themselves
- *Streams* (connections) may be subject to creation and destruction
- Part of a mission may involve *scalability* in real time

5. Probability and statistics

Possibility trees

- A series of events that each has a finite number n of alternative outcomes may be diagrammed by a *possibility tree*, which is n -ary
- *Theorem* (instance of the Multiplication Rule): a series of k events, each with n possible outcomes, has n^k distinct paths from root to leaf of its possibility tree
- *Example*: a four-digit PIN number with 36 possibilities for each character has 36^4 possible values

Counting elements of disjoint sets

- For a finite set A partitioned as A_1, A_2, \dots, A_k ,
 $|A| = \sum_{i=1 \text{ to } k} |A_i|$
- *Theorem*: for finite disjoint sets A, B , with $B \subseteq A$, $|A - B| = |A| - |B|$
- *Proof*:
 - if $B \subseteq A$, then $B \cap A - B = \emptyset$ (partition of B)
 - so $|B| + |A - B| = |A|$
 - hence $|A - B| = |A| - |B|$
- Also, $|A \cup B| = |A| + |B| - |A \cap B|$

Pigeonhole principle

- (Intuition) If n pigeons enter m pigeon holes, and if $n > m$, then at least one hole must have at least two pigeons
- (Formal) *Theorem*: If $|A| > |B|$ then $f: A \rightarrow B$ cannot be injective; i.e., $(\exists a, b \in A, a \neq b) f(a) = f(b)$
- *Example*: at least two people in Framingham have the same last-four, because there are 10K last-4s and more than 10K persons in Framingham
- *Corollary*: Any function from an infinite set to a finite one is non-injective

Permutations and combinations

- *Set*: A non-duplicating collection of items, not defined by ordering
- *Sequence*: An aggregate defined by ordering; possibly with duplication
- *Permutations*: The possible orderings of elements of a set
- *Combinations*: The unordered subsets of a set
- Our interest is to *count* permutations and combinations in order to determine probabilities

Permutations

- *Definition:* Orderings of n objects taken k at a time, without repetition; there are $(n!)$ permutations for n objects
- *Example:* There are $5! = 120$ ways to order the letters A, B, C, D, E
- *k-permutations* ($P(n,k)$): Orderings of n objects taken k at a time; there are $(n! / (n - k)!)$ k -permutations of n objects
- *Example:* there are $P(6, 3) = 120$ different ways to throw a die such that only 1, 2, or 3 show

Combinations

- *Definition:* the number of ways to select from k objects at a time, taken from a set of n objects, without order or repetition
- $C(n, k) = n! / ((n - k)! k!)$
- *Example:* There are $C(36, 6)$ ways to play the lottery where 6 numbers are chosen out of 36
- $C(n, k)$ is also written $\binom{n}{k}$ (“ n choose k ”)
- Note that $n! = n (n - 1) (n - 2) \times \dots \times 2$

Binomial coefficients

- $C(n, k)$ is also called a *binomial coefficient*, is computed by Pascal's Triangle, and is defined by the following recurrence, called *Pascal's formula*:

$$C(n, k) = \begin{cases} 1 & \text{if } k = 0 \text{ or } k = n \\ C(n-1, k-1) + C(n-1, k) & \text{otherwise} \end{cases}$$

- *Binomial theorem*: $(a + b)^n = \sum_{k=0..n} C(n, k) a^{n-k} b^k$

References

Susanna Epp. *Discrete Mathematics with Applications*. Brooks/Cole, 2011.

George Luger. *Artificial Intelligence*. Addison Wesley, 2005.

Stuart Russell and Peter Norvig. *AI: A Modern Approach, 2nd ed.* Prentice Hall, 2003.