Discrete Structures | Summary

B Pijnacker (s4106164)

Contents

I Fundamentals **3**

1 Fundamentals **4**

1.1 Sets and subsets 4
1.2 Operations on sets 4
1.3 Sequences 4
1.4 Properties of integers 4
1.5 Matrices 4
1.6 Mathematical Structures 5

II Induction, Counting and Recurrences **6**

2 Logic **7**

2.4 Mathematical Induction 7

3 Counting **8**

3.1 Permutations 8
3.2 Combinations 8
3.3 Pigeonhole Principle 8
3.5 Recurrence Relations 8

III Relations and Digraphs **9**

4 Relations and Digraphs **10**

4.1 Product Sets and Partitions 10
4.2 Relations and Digraphs 10
4.3 Paths in Relations and Digraphs 10
4.4 Properties of Relations 10
4.5 Equivalence Relation 11
4.7 Operations on Relations 11
4.8 Transitive Closure and Warshall's Algorithm 11

IV Functions **12**

5 Functions **13**

5.1 Functions 13
5.2 Functions for Computing Science 13
5.3 Growth of Functions 13
5.4 Permutation Functions 14

V Ordering Relations and Structures **15**

6 Ordering Relations and Structures **16**

6.1 Partially Ordered Sets 16
6.2 Extremal Elements of Partially Ordered Sets 16

6.3 Lattices 16
6.4 Finite Boolean Algebras 17

VI Trees and Graphs **18**

7 Trees **19**
7.1 Trees 19
7.2 Labeled Trees 19
7.3 Tree Searching 19
7.4 Undirected Trees 20
7.5 Minimal Spanning Trees 20

8 Topics in Graph Theory **21**
8.1 Graphs 21
8.2 Euler Paths and Circuits 21
8.3 Hamiltonian Paths and Circuits 21

Part I

Fundamentals

Chapter 1

Fundamentals

1.1 Sets and subsets **page 2**

This is easy and general knowledge by now.

1.2 Operations on sets **page 5**

These are the operations that can be performed on sets:

- **Union:** $A \cup B = \{x \mid x \in A \vee x \in B\}$
- **Intersection:** $A \cap B = \{x \mid x \in A \wedge x \in B\}$
- **Complement:** $A - B = \{x \mid x \in A \wedge x \notin B\}$

  A complement of a set can also exists with respect to the universe U
- **Symmetric Difference:** $A \oplus B = (A - B) \cup (B - A)$

1.3 Sequences **page 13**

A sequence is a list of objects arranged in a definite order. A sequence may be finite, or infinite. A finite sequence has an countable number of elements, an infinite sequence does not.

Sequences can be described by a formula. This can either be a recursive formula, like $a_n = a_{n-1} + 5; a_1 = 3$, or an explicit formula, like $a_n = 3n + 2$.

For any set we can create a characteristic equation that gives us a sequence corresponding to the set. For every value in U, there is a 1 in the sequence if this value is in A, and a 0 if this is not. Formally,

$$f_A(x) = \begin{cases} 1 & x \in A \\ 0 & x \notin A \end{cases}$$

1.4 Properties of integers **page 20**

If you want to study the Greatest Common Divisor and Least Common Multiple just read the book because I'm not going to. It's pages 22-27

1.5 Matrices **page 32**

A matrix is a rectangular array with m rows and n columns. Rows are indexed with i, columns with j. We notate any item as a_{ij}.

A couple special matrices are diagonal matrices, where every item off the main diagonal is 0; zero matrices, where every item is 0; and boolean matrices, where every item is either a 0 or a 1.

The operations on matrices are:

- **Addition:** $A = [a_{ij}]; B = [b_{ij}]; A + B = [a + b_{ij}]$ defined by $a + b_{ij} = a_{ij} + b_{ij}$

- **Product:** $A = [a_{ij}](p \times n); B = [b_{ij}](p \times m); AB = [ab_{ij}]$ defined by $ab_{ij} = a_{i1}b_{1j} + a_{i2}b_{2j} + \cdots + a_{ip}b_{pj}$
- **Transpose:** A^T is A flipped over its main diagonal

The inverse of a matrix A is another matrix B such that AB and BA give you an identity matrix (a diagonal matrix with only 1s on its diagonal).

A boolean matrix has some special operations, namely the meet and the join. These are defined as follows.

- (join) $A \vee B = C = [c_{ij}]$ where

$$c_{ij} = \begin{cases} 1 & a_{ij} = 1 \vee b_{ij} = 1 \\ 0 & \text{both are 0} \end{cases}$$

- (meet) $A \wedge B = D = [d_{ij}]$ where

$$d_{ij} = \begin{cases} 1 & \text{both are 1} \\ 0 & a_{ij} = 0 \vee b_{ij} = 0 \end{cases}$$

We also have the Boolean product ($\odot$), which follows the same structure as multiplication, but any entry in the output matrix is 1 if at any comparison a corresponding pair has both entries as a 1. Better explanation on page 38 in the book.

1.6 Mathematical Structures **page 41**

A structure is a collection of objects with defined operations abd accompanying properties.

An important property of a structure is closure. A structure is closed with respect to an operation if the result of the operation is another member of the collection of objects.

There is also some important properties of operations. These include the following

- **Commutativity:** $x \square y = y \square x$
- **Associativity:** $(x \square y) \square z = x \square (y \square z)$
- **Distributivity:** $(x \square y) \triangle z = (x \triangle z) \square (y \triangle z)$

A structure with a binary operator $\square$ may contain a distinguished object e, for which $x \square e = e \square x = x$. We call this e the identity for $\square$. The identity for any operation is unique. For example, the identity for the operator $+$ as defined in mathematics is the number 0.

Part II

Induction, Counting and Recurrences

Chapter 2

Logic

2.4 Mathematical Induction **page 76**

Mathematical induction is a proof technique which can be used to formally prove that statements hold for a given domain of numbers. We first show that $P(k)$ holds for a statement $P(n)$. If from this assumption we can prove that $P(k+1)$ holds, we have proven that $P(n)$ holds for any $n \geq k$. We can do this in a few basic steps:

- ▷ **Basis Step:** Show that $P(k)$ holds.
- ▷ **Induction Step:** Show that if $P(k)$ holds, $P(k+1)$ holds.

A slightly different form of mathematical induction is strong mathematical induction. Here the induction step is to show that $P(n_0) \wedge P(n_0+1) \wedge P(n_0+2) \wedge \cdots \wedge P(k) \Rightarrow P(k+1)$ is a tautology.

Chapter 3

Counting

3.1 Permutations **page 106**

If we want to know how many sequences of r distinct elements can be formed from a set A with n elements, also called the number of permutations of n objects taken r at the time, we calculate ${}_nP_r$.

$${}_nP_r = \frac{n!}{(n-r)!}$$

3.2 Combinations **page 110**

In a permutation, order matters. If we want to know how many r-element subsets we can take of an n-element set A, thus disregarding order, we take the combination of A, taken r at a time. This is ${}_nC_r$.

$${}_nC_r = \frac{n!}{r!(n-r)!}$$

3.3 Pigeonhole Principle **page 114**

If n pigeons are assigned to m pigeonholes, and $m < n$, then at least one pigeonhold contains two or more pigeons. This is called the pigeonhole principle. We can generalize this to the Extended Pigeonhole Principle. The Extended Pigeonhole Principle tells us that if n pigeons are assigned to m pigeonholes, then one of the pigeonholes must contain at least $\lfloor \frac{n-1}{m} \rfloor + 1$ pigeons.

3.5 Recurrence Relations **page 118**

Whenever we want to find an explicit formula for a recursive formula, we call the recursive formula a recurrence relation. We have techniques for finding an explicit formula for a recurrence relation.

Backtracking

For any recurrence relation a_n we can substitute $a_{n-1}, a_{n-2}, \ldots$ until a pattern is clear.

By using a characteristic equation

We can define a recurrence relation as a linear homogeneous relation of degree k if it is of the form $a_n = r_1a_{n-1} + r_2a_{n-2} + \cdots + r_ka_{n-k}$. For this we can create a polynomial characteristic equation of the form $x^k = r_1x^{k-1} + r_2x^{k-2} + \cdots + r_k$. The roots of such an equation can then be used to create an explicit formula. We will give a theorem for degree $k = 2$ only.

If a characteristic equation of degree 2 has 2 distinct roots s_1 and s_2, then $a_n = us_1^n + vs_2^n$ where u, v depend on the initial conditions.

If a characteristic equation has a single root s, then the explicit formula is $a_n = us^n + vns^n$

Part III

Relations and Digraphs

Chapter 4

Relations and Digraphs

4.1 Product Sets and Partitions **page 141**

An ordered pair a, b is a listing of the objects a and b in a prescribed order. It is a sequence of length 2. The product of 2 nonempty sets A and B is the product set containing all ordered pairs a, b with $a \in A$ and $b \in B$. The cardinality of the set $A \times B$, $|A \times B| = |A| \cdot |B|$.

A partition of a nonempty set A is a collection P such that each element of A belongs to exactly one of the sets in P.

4.2 Relations and Digraphs **page 145**

A relation between two sets of objects maps the eleents of set A to the elements of set B. This is often done using a formula that says, for example, $a\,R\,b \iff a < b$. A relation may be represented as a set of ordered pairs. A relation R from A to B is a subset of $A \times B$. A relation can also map a set to itself, in that case we have $R \subseteq A \times A$. The domain of R, $(Dom(R))$ is the set of elements in A that relates to an element in B.

A relation may also be represented by a matrix. In this case we have an $m \times n$ matrix for $|A| = m$ and $|B| = n$. A 1 in the matrix at m_{ij} tells us that $(a_i, b_j) \in R$. A 0 tells us that $(a_i, b_j) \notin R$.

If R is a relation on A, we can also represent the relation as a digraph. Here we draw all elements of A as vertices of the digraph, and relations are directed edges. A relation that goes both ways can be drawn as an undirected edge. Any element of A has an in-degree and an out-degree, these being the amount of ingoing edges and the amount of outgoing edges, respectively.

4.3 Paths in Relations and Digraphs **page 153**

Suppose R is a relation on A. A path in R is a finite sequence that follows the edges of a path from a to b. The length of a path is the amount of edges the path crosses.

We can also define the relation R^∞, this being the relation where there is a path from any element to any other element. We call this the connectivity relation.

If we have a matrix representing the relation called M_R, we can create the matrix for M_{R^n} as follows.

$$M_{R^n} = M_R \odot M_R \odot \cdots \odot M_R \qquad (n \text{ factors})$$

4.4 Properties of Relations **page 159**

Reflexivity

A relation R on A is reflexive if $(a, a) \in R \;\forall\; a \in A$. The relation is irreflexive if $(a, a) \notin R \;\forall\; a \in A$.

Symmetry

A relation R on A is symmetric if $(a, b) \in R \Rightarrow (b, a) \in R$. It is asymmetric if $(a, b) \in R \Rightarrow (b, a) \notin R$. The relation is antisymmetric if $((a, b) \in R \wedge (b, a) \in R) \Rightarrow a = b$.

In a symmetric relation, the digraph contains only undirected edges, as every relation goes both ways.

Transitivity

A relation R on A is transitive if $((a,b) \in R \wedge (b,c) \in R) \Rightarrow (a,c) \in R$.

Connectivity

A relation R on A is connected if for any $a \in A$ there exists a path to $b \in A$.

4.5 Equivalence Relation **page 166**

A relation R on A is called an equivalence relation if it is relexive, symmetric, and transitive.

If you have a partition P of set A, and the relation R on A is defined as

$$a \; R \; b \iff a \text{ and } b \text{ are members of the same block}$$

then R is an equivalence relation on A.

I don't understand theorem 2 on page 168 so you can go read that yourself.

4.7 Operations on Relations **page 177**

If we have relation R on A, we can obtain the complementary relation $\overline{R}$ by saying that $(a,b) \in \overline{R} \iff (a,b) \notin R$. If we have a matrix representing R, we can obtain the matrix for $\overline{R}$ by 'flipping all the bits' in the matrix.

We can also obtain the inverse relation R^{-1} defined by $(b,a) \in R^{-1} \iff (a,b) \in R$. It is clear to see that $(R^{-1})^{-1} = R$. If we have a matrix M_R for R, we can obtain the matrix for R^{-1} by taking $(M_R)^T$.

Closure

We can obtain closures of a relation by adding the ordered pairs needed for the relation to have the property. A closure of a relation should be the smallest possible superset of the set that has the wanted property.

Composition

If we have three sets A, B, C and two relations R_1, R_2. We can obtain the relation $R_2 \circ R_1$ from A to C by first applying R_1, then R_2.

4.8 Transitive Closure and Warshall's Algorithm **page 187**

If we have a relation R, then R^∞ is the transitive closure of the set. To obtain a transitive closure we can use Warshall's Algorithm.

1. Start at column 1, row 1. For the column obtain a set A containing all elements where there is a 1 in the column. For the row obtain a set B containing all elements where there is a 1 in the row.
2. Obtain the set $Q = A \times B$.
3. Add the ordered pairs in Q to the matrix.
4. Continue with the next row/column until no more changes occur.

Sidenote: I think this is correct but if it isn't please tell me.

Part IV

Functions

Chapter 5

Functions

5.1 Functions **page 207**

A function is a special type of relation. A function f from nonempty sets A to B $(f : A \to B)$. For any $a \in \text{Dom}(f)$, there exists a single corresponding element $f(a) \in B$. We can see a function as a mapping, because any input element has a single corresponding output element.

If we have two functions, $f : A \to B$ and $g : B \to C$, we can create the composite function $(g \circ f : A \to C)$ by first applying f from A to B, and then applying g to the obtained set B to obtain C.

Special types of functions

For all these functions, let $f : A \to B$ where neither $A = \varnothing$ or $B = \varnothing$.

- ▷ f is **everywhere defined** if $\text{Dom}(f) = A$
- ▷ f is **onto** if $\text{Ran}(f) = B$
- ▷ f is **one-to-one** if there do not exist 2 elements a_1 and a_2 such that $f(a_1) = f(a_2)$.

  A one-to-one function is often called a bijection, because not only does every $a \in \text{Dom}(f)$ have a unique $b \in \text{Ran}(f)$, every b also has a unique corresponding a.
- ▷ f is **invertable** if its inverse relation f^{-1} is also a function.

  If f is one-to-one; so is f^{-1}.
- ▷ The **identity function on A** is defined by $1_A(a) = a$

  $(f^{-1} \circ f) = 1_A$

5.2 Functions for Computing Science **page 216**

If we have a set A which is a subset of $U = \{u_1, u_2, u_3, \ldots, u_n\}$, we can construct a characteristic equation for A as

$$f_A(u_i) = \begin{cases} 1 & u_i \in A \\ 0 & u_i \notin A \end{cases}$$

I don't we need to do hashing function and fuzzy sets/logic because we never did that during the lectures and I can't imagine it'll be on the exam. It's on page 217-223 if you want to study it yourself.

5.3 Growth of Functions **page 226**

A way to compare growth of two function is by using Big $\mathcal{O}$. We can show a function f to be in $O(g)$ by showing that $|f(n)| \leq c \cdot |g(n)|$ for all $n \geq k$. The $\mathcal{O}$-class of a function is an absolute upper bound for the function, because the function never gets bigger than its class. We can define a relation Θ where ${}_a\Theta_b \iff O(f) = O(g)$.

Θ-classes are often used to describe the average running time of an algorithm

5.4 Permutation Functions **page 231**

A permutation is a bijection (one-to-one function) of a set A onto itself. If we have $A = \{a_1, a_2, \ldots, a_n\}$ and p is a bijection on A, we list the values of A and corresponding values as

$$p = \begin{pmatrix} a_1 & a_2 & \ldots & a_n \\ p(a_1) & p(a_2) & \ldots & p(a_n) \end{pmatrix}$$

The composition of two permutations, is another permutation. This is refered to as the product of these permutations. If A is a set of n elements, there exists $n!$ permutations of A.

Part V

Ordering Relations and Structures

Chapter 6

Ordering Relations and Structures

6.1 Partially Ordered Sets **page 248**

A relation R on A is a partial order if R is reflexive, antisymmetric, and transitive. The set A together with R, denoted as (A, R) is called a partially ordered set, or a poset.

If $(A, \leq)$ is a poset, elements $a, b \in A$ are said to be comparable if $a \leq b$ or $b \leq a$. Not every two elements of a poset need to be comparable. $\mathbb{Z}^+, |$ is a partial order, but $2, 7 \in \mathbb{Z}^+$ are not comparable. If every pair of elements in A is comparable, we say that A is a linear order.

Product sets

If $(A, \leq)$ and $(B, \leq)$ are posets, then $(A \times B, \leq)$ is a poset with

$$(a, b) \leq (a', b') \qquad \text{if} \qquad a \leq a' \in A \wedge b \leq b' \in B$$

This poset $(A \times B, \leq)$ is called the product partial order. We also have the partial order $(A \times B, \prec)$. This is called lexicographic order and is defined as

$$(a, b) \prec (a', b') \qquad \text{if} \qquad a < a' \vee (a = a' \wedge b \leq b')$$

Isomorphism

If we have a poset $(A, \leq)$ and a one-to-one function $f : A \to A'$, this function is an isomorphism from (A, leq) to $(A', \leq')$.

6.2 Extremal Elements of Partially Ordered Sets **page 258**

In this section we consider $(A, \leq)$

An element $a \in A$ is called a **maximal element** if there is no element $c \in A$ such that $a < c$. An element $b \in A$ is called a **minimal element** if there is no element $d \in A$ such that $d < b$.

An element is called a **greatest element** if $x \leq a \; \forall \; x \in A$. An element is called a **least element** if $a \leq x \; \forall \; x \in A$. Not all posets have least or greatest elements.

Consider a subset B from poset A. An element $a \in A$ is a **upper bound** of B if $b \leq a \; \forall \; b \in B$. An element $a \in A$ is a **lower bound** of B if $a \leq b \; \forall \; b \in B$. An element is a **least upper bound** if there is no other upper bound that relates to it. An element is a **greatest lower bound** if every other lower bound relates to it. The exception being bounds that have the same 'height' in a Hasse diagram.

6.3 Lattices **page 263**

A lattice is a poset $(L, \leq)$ in which every subset consisting of two elements has a least upper bound and a greatest lower bound.

If we have two lattices $(L_1, \leq)$ and $(L_2, \leq)$, then $(L_1 \times L_2, \leq)$ is also a lattice. L_1 and L_2 are sublattices of $L_1 \times L_2$.

If two lattices are isomorphic as posets then we say they are isomorphic lattices.

A lattice is distributive if $a \wedge (b \vee c) = (a \wedge b) \vee (a \wedge c)$ or $a \vee (b \wedge c) = (a \vee b) \wedge (a \vee c)$. Lattices are only nondistributive if the lattices pictured in Figure 44 (page 269) of the book are sublattices of the lattice in question.

A complement of an element $a \in A$ is a', where $a \wedge a' = 0$ and $a \vee a' = I$. A lattice L is complemented if it is bounded and if every element in L has a complement. If L is also distributive, any complement is unique.

6.4 Finite Boolean Algebras **page 273**

If a set S has n elements, all subsets of S can be represented by n-length sequences of 0s and 1s. If we do this for the lattice $(\mathcal{P}(S), \subseteq)$ where S has n elements, we obtain the lattice B_n. Any lattice that is isomorphic with B_n with $n \geq 0$ is a Boolean algebra.

Any formula involving $\cup$ and $\cap$ that hold for arbitrary subsets of S will continue to hold for elements of a Boolean algebra if $\cup$ is substituted for $\vee$ and $\cap$ with $\wedge$.

Part VI

Trees and Graphs

Chapter 7

Trees

7.1 Trees **page 309**

A tree (T, v_0) is a relation T on a set A with v_0 as a root. Any other element $v \in A$ is refered to as a vertex in T.

A tree consists of levels, from level 0, the root, to level n. Vertices that exists on the same level are called siblings. Vertices that are included in the subtree where the current vertex is the root, are descendants of that vertex. Vertices also have a parent and may have children. A node without children is called a leaf.

If every vertex in a tree has at most n offspring, the tree is an n-ary tree. If every vertex has n offspring, the tree is a complete n-tree.

7.2 Labeled Trees **page 313**

We can label the vertices of a tree to indicate a relation between the labels. For instance, we can express a algebraic expression as a tree. We can search through the tree to get the original order back.

7.3 Tree Searching **page 318**

Searching, or walking, a tree can be done in multiple different ways. In the three algorithms below, we only look at binary trees, with a left subtree and a right subtree per vertex.

Preorder Search

1. Visit v;
2. Apply this algorithm to $(T(v_L), v_L)$ if it exists;
3. Apply this algorithm to $(T(v_R), v_R)$ if it exists.

Postorder Search

1. Apply this algorithm to $(T(v_L), v_L)$ if it exists;
2. Apply this algorithm to $(T(v_R), v_R)$ if it exists.
3. Visit v;

Inorder Search

1. Apply this algorithm to $(T(v_L), v_L)$ if it exists;
2. Visit v;
3. Apply this algorithm to $(T(v_R), v_R)$ if it exists.

To search non-binary trees, we can represent a non-binary tree as a binary tree. Here we take the original root as the root of our tree. The child of this node will be the left-most node of the next level. From here, all siblings are put as a right subtree, all descendants are put as a left subtree. Look at Figure 16 (page 323) of the book for a good example.

7.4 Undirected Trees **page 326**

An undirected tree is the symmetric closure of a tree, where all edges are made bidirectional. If we have symmetric relation R and p is a path in R. We can say p is simple if no two edges of p correspond to the same undirected edge. If the path start and ends on the same vertex, and p is simple, p is a simple cycle.

Spanning Trees

If R is a symmetric, connected relation on A, then T is a spanning tree for R if it contains exactly the same vertices as R and if it can be obtained by deleting some edges in R.

7.5 Minimal Spanning Trees **page 333**

If we have a weighed graph, we may want to obtain a spanning tree where only the edges with lowest weight remain. This is called a minimal spanning tree. We can obtain a minimal spanning tree with Prim's Algorithm or with Kruskal's Algorithm

Prim's Algorithm

1. Choose a vertex in the graph. Color it black;
2. As long as there remain non-black vertices:
   (a) Let e be an edge of minimal weight connecting a black vertex with a non-black vertex;
   (b) Color e red and the corresponding vertex black.

For the matrix version of this algorithm, look in the book at page 336.

Kruskal's Algorithm

1. Color an edge of minimal weight red;
2. For any non-red edges e of minimal weight, color e red and the corresponding vertices black, provided that this does not create a cycle;
3. Repeat step 2 until a minimal spanning tree is obtained.

Chapter 8

Topics in Graph Theory

8.1 Graphs **page 350**

A graph G consists of vertices, edges, and a function γ that maps edges to vertices. The degree of a vertex in a graph is the number of edges that vertex is connected to. A loop contributes 2 to the degree of a vertex. A vertex with degree 0 is called an isolated vertex. A pair of vertices connected by an edge are called adjacent vertices.

A graph is connected if there is a path from any vertex to any other vertex in the graph. If the graph is disconnected, the free-floating pieces are called components of the graph. A graph where there is a *direct* path from any vertex to any other vertex is called a complete graph. A complete graph of n vertices has degree $n - 1$ for every vertex.

Because paths are fun or something, we're going to make a path in a graph. The path is going to begin and end on the same vertex, which we call a circuit. If no vertex appears more than once in a vertex sequence (except of course the first/last one), we call it a simple circuit.

Subgraphs and Quotient Graphs

A subgraph of a graph is another graph where some edges and/or vertices are missing. The subgraph must be able to be formed by removing edges and vertices from the original graph.

If we have a partition R of the set of vertices V where our graph G was constructed from, we can construct a quotient graph G^R. All vertices that are in the same block of the partition are combined into one vertex. Edges are connected between the blocks in the same way they were in the original graph, now just all going to a single node.

8.2 Euler Paths and Circuits **page 355**

A path in a graph is an Euler path if it includes every edge exactly once. It is an Euler circuit if it also starts and ends on the same vertex. An Euler path is not always possible. If a graph has more than two odd-degree vertices, there can be no Euler path. A Euler path must start at one odd-degree vertex and end at the other. If there is any vertex of odd degree, there can not be an Euler circuit. An algorithm to construct Euler circuits is Fleury's algorithm.

Fleury's Algorithm

1. Select any edge e_1 that is not a bridge in G. Remove this edge from the graph and let G_1 be the new subgraph;

2. Continue deleting edges until you complete the circuit. Be sure to not delete any bridges.

8.3 Hamiltonian Paths and Circuits **page 362**

A Hamiltonian path is a path that visits every vertex exactly once. A Hamiltonian circuit is a Hamiltonian path that begins and ends on the same vertex.

There does not exist an algorithm to find a Hamiltonian circuit for a given graph.