Foundations of Computer Science*

October 1, 2014

Disclaimer

These notes have been prepared with the **only** purpose to help me pass the Computer Science qualifiying exam at the University of Illinois at Chicago. They are distributed as they are (including errors, typos, omissions, etc.) to help other students pass this exam (and possibly relieving them from part of the pain associated with such a process). I take **no responsibility** for the material contained in these notes (which means that you can't sue me if you don't pass the qual!) Moreover, this pdf version is distributed together with the original LaTeX (and LyX) sources hoping that someone else will improve and correct them. I mean in absolute no way to violate copyrights and/or take credit stealing the work of others. The ideas contained in these pages are **not mine** but I've just aggregated information scattered all over the internet.

Contents

1 Set Theory (Basic concepts)

1.1 Definitions

- A *set* is an unordered collection of distinct objects.
- The *empty set* $\emptyset$ is defined as $\emptyset = \{\nexists x : x \in \emptyset\}$.
- The *universe set* U is defined as $U = \{\forall x : x \in U\}$.

- The *intersection* of sets A and B is the set $A \cap B = \{x : x \in A \text{ and } x \in B\}$.
- The *union* of sets A and B is the set $A \cup B = \{x : x \in A \text{ or } x \in B\}$.
- The *complement* of set A is the set $\bar{A} = \{x : x \notin A\}$.
- The *difference* of sets A and B is the set $A \backslash B = A \cap \bar{B} = \{x : x \in A \text{ and } x \notin B\}$.
- The *symmetric difference (XOR)* of sets A and B is the set $A \otimes B = (A \backslash B) \cup (B \backslash A)$.

1.2 Properties

- Empty set: $A \cup \emptyset = A$ and $A \cap \emptyset = \emptyset$.
- Universe set: $A \cup U = U$ and $A \cap U = A$.
- Idempotency: $A \cup A = A$ and $A \cap A = A$.
- Cummutative: $A \cup B = B \cup A$ and $A \cap B = B \cap A$.
- Associative: $A \cup (B \cup C) = (A \cup B) \cup C$ and $A \cap (B \cap C) = (A \cap B) \cap C$.
- Distributive: $A \cup (B \cap C) = (A \cup B) \cap (B \cup C)$ and $A \cap (B \cup C) = (A \cap B) \cup (B \cap C)$.
- Absorption: $A \cup (A \cap B) = A$ and $A \cap (A \cup B) = A$.
- DeMorgan: $\overline{A \cup B} = \bar{A} \cap \bar{B}$ and $\overline{A \cap B} = \bar{A} \cup \bar{B}$.
- Negation: $A \cup \bar{A} = U$ and $A \cap \bar{A} = \emptyset$.
- Double negation: $\bar{\bar{A}} = A$.

1.3 Power set and partitions

- We define the *power set* of a set A as the set of all subset of A: $\mathcal{P}(A) = \{X : X \subseteq A\}$. The cardinality of such a set is: $|\mathcal{P}(A)| = 2^{|A|}$
- We say a collection $\alpha = \{A_i\}$ of non-empty sets forms a *partition* of the set A if and only if:
  - $A_i \in \alpha \implies A_i \in \mathcal{P}(A)$
  - the sets are pairwise disjoint, that is, $A_i, A_j \in \alpha$ and $i \neq j \implies A_i \cap A_j = \emptyset$
  - $\bigcup_{A_i \in \alpha} A_i = A$

1.4 Cartesian product

The Cartesian product of two sets A and B, denoted $A \times B$, is the set of all the *ordered* pairs such that the first element of the pair is an element of A and the second is an element of B. More formally $A \times B = \{(a, b) : a \in A \text{ and } b \in B\}$ whose cardinality is $|A \times B| = |A| \cdot |B|$ if A and B are finite.

We denote an *n-fold* Cartesian product over a single set $A^n = A \times A \times \cdots \times A$ whose cardinality is $|A^n| = |A|^n$ if A is finite.

2 Relations

An *n-ary* relation on sets A_1, A_2, ..., A_n is a subset of $A_1 \times A_2 \times \cdots \times A_n$.

A *binary relation* R on two sets A and B is a subset of the Cartesian product $A \times B$.

2.1 Properties of binary relations

- A binary relation $R \subseteq A \times A$ is *reflexive* if aRa for all $a \in A$.
- A relation R is *symmetric* if $aRb \implies bRa$.
- A relation R is *transitive* if $aRb \wedge bRc \implies aRc$.
- A binary relation R on a set A is *antisymmetric* if $aRb \wedge bRa \implies a = b$.

An example of relation that is neither symmetric nor antisymmetric is x "divides" y on $\mathbb{Z}$: 6 divides 3 doesn't imply 3 divides 6; 3 divides -3 and -3 divides 3 doesn't imply -3=3!

2.2 Types of binary relations

2.2.1 Equivalence relation

A binary relation that is reflexive, symmetric and transitive is an *equivalence relation*.

If R is an equivalence relation on a set A, then for $a \in A$, the *equivalence class* of a is the set $[a] = \{b \in A : aRb\}$, that is, the set of all elements equivalent to a.

The equivalence classes of any equivalence relation R on a set A form a partition of A, and any partition of A determines an equivalence relation on A for which the sets in the partition are the equivalence classes.

2.2.2 Partial order

A binary relation that is reflexive, antisymmetric and transitive is a *partial order*.

A set on which a partial order relation is defined is called a *partially ordered set*. (poset)

In a poset A, there may be no single "maximum" element a such that bRa for all $b \in A$. Instead, the set may contain several *maximal* elements a such that for no $b \in A$, where $b \neq a$, is it the case that aRb.

2.2.3 Total relation

A binary relation R on a set A is a *total relation* if $\forall a, b \in A$, $aRb \vee bRa$, that is, if every pairing of elements of A is related by R.

2.2.4 Total order

A binary relation that is both a partial order and a total relation is a *total order*.

3 Functions

Given two sets A and B, a function f is a binary relation on A and B such that $\forall a \in A, \exists! b \in B : (a, b) \in f$.

The set A is called the *domain* of f and the set B in called the *codomain* of f.

Intuitively f assigns an element of B to *each* element of A. No element of A is assigned two different elements of B, but the same element of B can be assigned to two different elements of A.

If $f : A \to B$ is a function the *image* of a set $A' \subseteq A$ under f is defined by $f(A') = \{b \in B : b = f(a) \, for \, some \, a \in A'\}$. The *range* of f is the image of its domain, that is, $f(A)$.

3.1 Types of functions

- A function is a *surjection* if its range is its codomain. A surjection $f : A \to B$ is sometimes described as *mapping* A *onto* B and $|B| = |f(A)| \leq |A|$.
- A function $f : A \to B$ is an *injection* if distinct arguments to f produce distinct values, that is, $a \neq a' \implies f(a) \neq f(a')$. An injection is sometimes called a *one-to-one* function and $|B| \geq |f(A)| = |A|$.
- A function $f : A \to B$ is a *bijection* if it's injective and surjective. A bijection is sometimes called a *one-to-one correspondence* or, when $B = A$, *permutation* and $|B| = |f(A)| = |A|$. When a function f is bijective we define its inverse f^{-1} as $f^{-1}(b) = a \iff f(a) = b$.

3.2 Counting functions

- The set of all (possible) functions from A to B is written B^A and $|B^A| = |B|^{|A|}$. This is obtained thinking that, for each element $a \in A$, $f(a)$ can be any value $b \in B$. Therefore we have $|B|$ ways of choosing b for each value of a which leads to $|B^A| = \underbrace{|B| \cdot |B| \cdot \dots \cdot |B|}_{|A|\, times} = |B|^{|A|}$.

3.3 Boolean Functions

A Boolean function is a function $f : \times^n\{0,1\} \to \{0,1\}$. In general the number of such Boolean functions is 2^{2^n}.

Boolean functions are strictly related to Propositional Logic: see section 7 for more details.

4 Basic counting and combinatorics

4.1 Combinatorics

4.1.1 Counting subsets

- How many *ordered subsets* of cardinality k can be obtained from a set of cardinality n? (*permutation*s)

$$n(n-1)\dots(n-k) = \frac{n!}{(n-k)!}$$

- What happens when $n = k$? (*permutation*s)

$$\frac{n!}{0!} = n! = k!$$

- What if the *permutation* is circular?

$$(n-1)!$$

- How many *subsets* of cardinality k can be obtained from a set of cardinality n? (*combinations*)

$$\frac{n!}{k!(n-k)!} = \binom{n}{k} = \binom{n}{n-k}$$

4.1.2 Counting multisets (picking from "infinite" buckets)

- How many *ordered multisets* of cardinality k can be obtained from a set of cardinality n? (or given k holes and n types, how many different ways do I have to distribute the types?)

$$n^k$$

- How many *multisets* of cardinality k can be obtained from a set of cardinality n? (or given k holes and n types, how many different ways do I have to put them in non considering permutations?)

$$\binom{n+k-1}{n-1} = \binom{n+k-1}{k} = \left(\binom{n}{k}\right)$$

- Given k holes and n types (or k indistinguishable objects and n bins) the number of different ways to distribute them so that there is at least one in each bin is

$$\binom{k-1}{n-1}$$

4.1.3 Multinomial coeffiecients (picking from "finite" buckets)

- How many multisets of cardinality n can be obtained from a multiset of cardinality n where k_i are the multiplicities of each of the distinct elements? (*multiset permutation*)

$$\frac{n!}{k_1!k_{2!}\dots k_m!} = \binom{n}{k_1, k_2 \dots, k_m}$$

4.2 Combinatorial principles

4.2.1 Rule of sum

If A and B are two *disjoint finite* sets, then $|A \cup B| = |A| + |B|$. Which means that the number of ways to choose one element from one of two disjoint sets is the sum of the cardinalitied of he two sets.

4.2.2 Rule of product

If A and B are two *finite* sets, then $|A \times B| = |A| \cdot |B|$. Which means that the number of ways to choose an ordered pair is the number of ways to choose the first element times the the number of ways to choose the second element.

4.2.3 Inclusion-exclusion principle

The inclusion-exclusion principle relates the size of the union of multiple sets, the size of each set, and the size of each possible intersection of the sets. It could be thought of as the generalization of the rule of sum.

In it's simplest version (for two sets) it states that if A and B are two sets, then

$$|A \cup B| = |A| + |B| - |A \cap B|$$

from which we can conclude that $|A \cup B| \leq |A| + |B|$. If A and B are disjoint we have $|A \cup B| = |A| + |B|$ and if $A \subseteq B$ then $|A| \leq |B|$.

4.2.4 Recursive formula for binomial coefficients

Binomial coefficients can be recursively defined as

$$\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}$$

with, as initial value

$$\binom{n}{0} = 1 \qquad \forall n \in \mathbb{N}$$
$$\binom{0}{k} = 0 \qquad \forall k > 0$$

4.2.5 Generating function for binomial coefficients (binomial expansion)

The generating function can be written as

$$(x+y)^n = \sum_{k=0}^{n} \binom{n}{k} x^k y^{n-k}$$

and, substituting $y = 1$ we obtain

$$(1+x)^n = \sum_{k=0}^{n} \binom{n}{k} x^k$$

and, in it's most particular case, substituting $x = y = 1$

$$2^n = \sum_{k=0}^{n} \binom{n}{k}$$

5 Proof Techniques

5.1 Mathematical Induction

Used to prove that a certain statement $P(n)$ holds for every value of $n \geq b$, where $n, b \in \mathbb{N}$.

5.1.1 Weak

1. *Base case*: show that $P(b)$ is true. (usually $b = 0$ or $b = 1$)
2. *Inductive step*:
   (a) *Inductive hypothesis*: assume $P(n)$ holds for some $n \geq b$.
   (b) Show that $P(n+1)$ holds. Usually this involves substituting $n+1$ in $P(n)$ to find $P(n+1)$ and show that we can obtain $P(n+1)$ from $P(n)$ using the "recursive definition" of the statement we are trying to prove.

5.1.2 Strong

1. *Base case*: Same as weak induction.
2. *Inductive step*:
   (a) *Inductive hypothesis*: assume $P(i)$ holds $\forall n \in \mathbb{N} : b \leq i \leq n$.
   (b) Same as weak induction.

5.1.3 Structural

Generalization of mathematical induction used to prove that a statement $P(x)$ holds on any instance x of some sort of recursively-defined structure S (like lists or trees).

1. Define a well-founded partial order on the structure ("sublist" for lists and "subtree" for trees).
2. *Base case*: Show that P holds for all the minimal structures (empty list, tree root).
3. *Inductive step*:
   (a) *Inductive hypothesis*: assume that P holds for the immediate substructures of a certain structure S.
   (b) Show that P must hold for S. Usually this involves showing how S can be obtained from the immediate substructures applying some sort of recursive rule.

Since every property must be proved separately, there may exist more than one base case, and/or more than one inductive case, depending on how the function or structure was constructed.

Example Let F be the set of fully parenthesized expressions in x defined recursively as follows:

1. $\{x\} \in F$, $\{0\} \in F$, $\{1\} \in F$;
2. If $e_1, e_2 \in F$ then $\{\text{'['}e_1\text{'+'}e_2\text{']'}\} \in F$;
3. If $e_1, e_2 \in F$ then $\{\text{'['}e_1\text{'}*\text{'}e_2\text{']'}\} \in F$;
4. If $e_1 \in F$ then $\{\text{'-['}e_1\text{']'}\} \in F$.

Prove that every $e \in F$ has the same number of left and right parentheses.

***

Define $P(e) ::=$ "*expression e has the same number of left and right parentheses*".

Base case:

- $e = 0, e = 1, e = x$ has no parentheses.

Inductive step:

- Assume $P(e_1)$ and $P(e_2)$ and prove $P([e_1 + e_2])$;
- $P(e)$ implies that e has k_e left parentheses and k_e right parentheses for some $k_e \in \mathbb{N}$.
- Therefore $[e_1 + e_2]$ will have $k_{e_1} + k_{e_2} + 1$ right parenthese and $k_{e_1} + k_{e_2} + 1$ left parentheses.

To properly conclude the demonstration we need to repeat the inductive step on properties 3 and 4.

□

5.2 Proof by contradiction

To prove a statement P holds by contradiction we need to:

1. Assume P doesn't hold. $(\neg P)$
2. Show that this leads to some contradiction. Usually this involves showing that:
   (a) Either $\neg P \implies \bot$
   (b) or equivalently $\neg P \implies (Q \wedge \neg Q)$

this technique is often used to prove correctness of greedy algorithms.

6 Summations and infinite series

6.1 Properties

The value of a *finite* series is always well defined, and we can add its terms in any order.

If the limit $\lim_{n\to\infty} \sum_{k=0}^{n} a_k$ does not exists the series diverges; otherwise, it converges. The terms of an infinite series cannot always be added in any order: we can change such an order only if the series is *absolutely convergent*, that is a series for which $\sum_{k=0}^{n} |a_k|$ also converges.

6.1.1 Linearity

All finite series are linear, which means

$$\sum_{k=1}^{n}(ca_k + b_k) = c\sum_{k=1}^{n} a_k + \sum_{k=1}^{n} b_k$$

where c is any real number. This also applies to infinite convergent series. This property becomes really handy when manipulating and transforming summations incorporating other linear operators (integration, differentiation, logarithm,...) and in particular asymptotic notation.

6.1.2 Products

We can convert a formula with a product to a formula with a summation using the identity

$$\lg\left(\prod_{k=1}^{n} a_k\right) = \sum_{k=1}^{n} \lg a_k$$

6.2 Common series

Some closed form solutions that are particularly handy.

6.2.1 Arithmetic series

$$\sum_{k=0}^{n} k = \frac{n(n+1)}{2}$$

$$\sum_{k=0}^{n} k^2 = \frac{n(n+1)(2n+1)}{6}$$

$$\sum_{k=0}^{n} k^3 = \frac{n^2(n+1)^2}{4}$$

6.2.2 Geometric series

For real $x \neq 1$, we have

$$\sum_{k=0}^{n} x^k = \frac{x^{n+1} - 1}{x - 1}$$

A particular case of the previous is $\sum_{k=0}^{n} 2^k = 2^{n+1} - 1$ when $x = 2$.
When the summation is infinite and $|x| < 1$, we have the infinite decreasing series

$$\sum_{k=0}^{\infty} x^k = \frac{1}{1 - x}$$

of which a particular case is $\sum_{k=0}^{\infty} \frac{1}{2^k} = 2$ when $x = 2$.

6.2.3 Harmonic series

For positive integers n, the nth harmonic number is

$$H_n = 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n} = \sum_{k=1}^{n} \frac{1}{k} = \ln n + O(1)$$

The infinite series $\sum_{k=1}^{\infty} \frac{1}{k}$ it's an example of divergent series.

6.2.4 Telescoping series

$$\sum_{k=1}^{n} (a_k - a_{k-1}) = a_n - a_0$$

$$\sum_{k=0}^{n-1} (a_k - a_{k+1}) = a_0 - a_n$$

Note that sometime it is possible to rewrite a particular series as a telescoping series. As an example $\sum_{k=1}^{n-1} \frac{1}{k(k+1)} = \sum_{k=1}^{n-1} \left(\frac{1}{k} - \frac{1}{k+1} \right) = 1 - \frac{1}{n}$.

6.2.5 Other common series

$$\sum_{k=0}^{n} 1 = n$$

This may seem trivial but it's very handy to manipulate and transform summations.
An example application of the previous is this is the series

$$\sum_{k=1}^{n} \lg n = \lg n \cdot \sum_{k=1}^{n} 1 = n \lg n$$

7 Propositional Calculus

It's the study of Boolean functions, where 1 plays the role of "true" and 0 the role of "false".

7.1 Definitions

Statement variable: a Boolean variable

Statement form: a particular form of a Boolean function (e.g. $p \wedge (\sim q \vee r)$)

Equivalence of two statement forms: two statements forms are *logically equivalent* if they have the same truth table

Tautology (Contradiction): A statement form that represents the constant 1 function is called *tautology* and is represented with the symbol $\top$. A statement form that represents the constant 0 function is called *contradiction* and is represented with the symbol $\bot$.

7.2 Logic and sets

Given a set P, let the corresponding lower case letters p stand for the statement that $x \in P$. Substituting:

- $\neg$ with $\sim$
- $\cup$ with $\vee$
- $\cap$ with $\wedge$
- U with $\top$
- $\emptyset$ with $\bot$

we can use the same set properties listed in section 1.2, that is there is a correspondence between set and logic notation.

7.3 Implication

We define $p \implies q$ to be the Boolean function, called implication, with the following truth table

p	q	$p \implies q$
0	0	1
0	1	1
1	0	0
1	1	1

It is possible to write an equivalent statement form using only $(\sim, \vee)$, This and several other facts about implication are summarized in the following table. $\sim q \implies \sim p$ is called the *contrapositive*, $q \implies p$ is called the *converse* and$\sim p \implies \sim q$ is called the *inverse.*

p	q	$p \implies q$	$\sim p \vee q$	$\sim (p \implies q)$	$p \wedge \sim q$	$\sim q \implies \sim p$	$q \implies p$	$\sim p \implies \sim q$
0	0	1	1	0	0	1	1	1
0	1	1	1	0	0	1	0	0
1	0	0	0	1	1	0	1	1
1	1	1	1	0	0	1	1	1

The $\subseteq$ operator is the equivalent of $\implies$ in the set notation as it can be shown with a simple Venn diagram.

7.3.1 Double implication

We define $p \iff q$ as the Boolean function, called *double implication*, with the following truth table

p	q	$p \iff q$	$(p \implies q) \wedge (q \implies p)$
0	0	1	1
0	1	0	0
1	0	0	0
1	1	1	1

7.3.2 Equivalence in English language

- $p \implies q$ is normally written as "q *if* p". The *if* part in "q if and only if p" where p is the *sufficient* condition for q.
- $p \impliedby q$ is normally written as "q *only if* p". The *only if* part in "q if and only if p" where p is the *necessary* condition for q.

8 First Order Logic (Predicate Calculus)

A *predicate* is any function whose codomain are statements that are either true or false. There are two things to be careful about:

- the codomain are statements *not* the truth value of the statements,
- the domain is arbitrary.

The truth set of a predicate S with domain D is the set of those $x \in D$ for which $S(x)$ is true. It is written $\{x|S(x)\}$.

8.1 Quantifiers

- The phrase "for all" is called universal quantifier and is written $\forall$. "$\forall x \in D, S(x)$" is equivalent to saying that the truth set of $S(x)$ contains the set D.
- The phrase "for some" is called existential quantifier and is written $\exists$. "$\exists x \in D, S(x)$" is equivalent to saying that the truth set $S(x)$ contains at least one element of the set D.

8.2 Algebraic rules for predicate logic

In a formula, if a predicate doens't contain a varible that is bound to a quantifier, then it can be moved in and out of the scope of the quantifier without changing the truth value of the formula.

8.2.1 Negating quantifiers

- $\forall x \in D, P(x) \iff \sim (\exists x \in D, \sim P(x))$ which can be written also as $\sim (\forall x \in D, P(x)) \iff \exists x \in D, \sim P(x)$
- $\exists x \in D, P(x) \iff \sim (\forall x \in D, \sim P(x))$ which can be written also as $\sim (\exists x \in D, P(x)) \iff \forall x \in D, \sim P(x)$

Whenevere there are more than one quantifier we have to negate the most external one and then the second most external and so on until we reach a quantifier-free part. At this point we can apply the same rules of predicate logic.

8.2.2 Moving quantifiers

- $\forall x \in D, (P(x) \wedge Q(x)) \iff (\forall x \in D, P(x)) \wedge (\forall x \in D, Q(x))$. We can't do this if we have $\exists$.
- $\exists x \in D, (P(x) \vee Q(x)) \iff (\exists x \in D, P(x)) \vee (\exists x \in D, Q(x))$. We can't do this if we have $\forall$.

As long as quantifiers are not involved the same rules of predicate logic can be used.

8.3 Standardized forms

Any formula can be expressed using only $(\sim, \wedge)$ or $(\sim, \vee)$. This leads to two standardized forms:

- *Conjunctive Normal Form (CNF)*: A formula is in CNF if it is a conjunction of clauses, where a clause is a disjunction of literals, where a literal and its complement cannot appear in the same clause.
- *Disjunctive Normal Form (DNF)*: A formula is in DNF if it is a disjunction of clauses, where a clause is a conjunction of literals.

A formula that is the CNF equivalent of a formula with n clauses will have, in general, $O(2^n)$ clauses; that is, in some cases the conversion to CNF can lead to an exponential explosion of the formula. The same is true for DNF.

- *Prenex normal form*: A formula is in prenex normal form if it is written as a string of quantifiers followed by a quantifier-free part (referred to as the matrix).
- *Skolem normal form*: A formula is in Skolem normal form if it is in conjunctive prenex normal form with only universal first-order quantifiers.

9 Discrete probability

Let U be a *finite sample* space and $P : U \to \mathbb{R}$ such that $P(t) \geq 0, \forall t \in U$ and $\sum_{t \in U} P(t) = 1$, then P is called a *probability function* on U and the pair (U, P) is called a *probability space*. We extend this definition to subsets of U called *events*: $E \subseteq U : P(E) = \sum_{t \in E} P(t)$.

9.1 Probability of disjoint events

If $X \subset U$, $Y \subset U$ and $X \cap Y = \emptyset$ then

- $P(X \cup Y) = P(X) + P(Y)$ and
- $P(X \cap Y) = 0$.

When events are *NOT* disjoint we can use Venn diagrams to calculate probability of events. The following equivalences may me handy:

- $P(\bar{A}) = 1 - P(A)$
- $P(A \cap B) = P(A) - P(A - B)$

9.2 Discrete probability distributions

9.2.1 Bernoulli

$B(1, p)$ represents an event whose probability is p.

e.g. The probability of obtaining head (a success) when flipping a coin is $B(1, 0.5)$.

9.2.2 Binomial

$B(n, p)$ represents the probability of obtaining exactly k successes in a series of n independent events each one with a probability p.

The probability mass function (p.m.f.) is:

$$P(K = k) = \binom{n}{k} p^k (1-p)^{n-k}$$

e.g. The probability of obtaining k heads when flipping a coin is $B(k, 0.5)$

9.2.3 Geometric

Represents the number of Bernoulli trials before a success.

The probability mass function is:

$$P(K = k) = p(1-p)^{k-1}$$

e.g. The probability that you need exactly k trials, each one with probability p, before a success is the p.m.f.

9.2.4 Poisson

$Pois(\lambda)$ represents the probability of a series of events occurring in a fixed amount of time knowing they occur with a certain rate.

The probability mass function is:

$$P(K = k) = \frac{\lambda^k}{k!} \cdot e^{-\lambda}$$

where λ is the expected number of occurrences and k is the actual number of occurrences of an event - the probability of which is given by the p.m.f.

e.g. The probability that there will be exactly k calls to a call-center in an hour, where the average calling rate is λ calls/hour, is the p.m.f.

9.2.5 Hypergeometric

Represents the number of successes on n draws from a population without replacement

The probability mass function is:

$$P(K = k) = \frac{\binom{m}{k}\binom{N-m}{n-k}}{\binom{N}{m}}$$

where m is the possible number of successes, n is the number of draws, N is the cardinality of the population and k is the number of successes - the probability of which is given by the p.m.f.

e.g. The probability of extracting exactly k white marbles (i.e. having k successes) drawing n without replacement from an urn containing N marbles in total, m of which are white is the p.m.f.

10 Graphs

10.1 Definitions

The following definitions may refer to directed graphs (D), undirected graphs (U) or both (B).

10.1.1 Basic definitions

- (B) Usually $n = |V|$ and $m = |E|$. In any graph, $m \leq n^2$.
- (D) A **directed graph** G is a pair (V, E) where V is a finite set (called the vertex set) and E is a binary relation on V and is called the edge set of G.
- (U) In an **undirected graph** $G = (V, E)$ the edge set E consists of unordered pairs of *distinct* vertices.
- (D) **Self-loops** are edges from a vertex to itself and are only possible in directed graphs.
- (D) A directed graph with no self-loops is **simple**.
- (D) If (u, v) is an edge in a graph $G = (V, E)$, we say that the vertex v is **adjacent** to vertex u.
- (D) In a directed graph G, a **neighbor** of a vertex u is any vertex that is adjacent to u in the undirected version of G.
- (U) In undirected graphs **adjacency** relation is symmetric.
- (U) In an undirected graph, u and v are **neighbors** if they are adjacent.

10.1.2 Degree

- (D) The **degree** of a vertex in a directed graph is its **in-degree** plus its **out-degree**.
- (U) The **degree** of a vertex in an undirected graph is the number of edge incident on it.
- (B) The *degree* of a vertex v is denoted $\deg(v)$. The maximum degree of a graph G, denoted by $\Delta(G)$, and the minimum degree of a graph, denoted by $\delta(G)$, are the maximum and minimum degree of its vertices.
- (B) A vertex whose degree is 0 is **isolated**.
- (B) In a graph $G = (V, E)$, $\sum_{v \in V} d_v = 2 \cdot |E|$ where d_v is the degree of vertex $v \in V$.

10.1.3 Paths and cycles

- (B) In a graph $G = (V, E)$, a **path** of length k from a vertex u to a vertex u' it's a sequence $\langle v_0, v_1, v_2, \ldots, v_k \rangle$ of vertices such that $u = v_0$, $u' = v_k$ and $(v_{i-1}, v_i) \in E$ for $i = 1, 2, \ldots, k$.
- (B) The **length** of the **path** is the number of edges in the path: therefore the path contains k edges and $k + 1$ vertices.
- (B) If there is a path p from u to u' we say that u' is **reachable** from u via p.
- (B) A **trail** is a path in which all edges are distinct.
- (B) A **simple path** is a path where all vertices (and thus all edges) are distinct.
- (B) Two **paths** are **independent** (alternatively, **internally vertex-disjoint**) if they do not have any internal vertex in common.
- (B) A **subpath** of a path is a contiguous subsequence of its vertices.
- (D) A **cycle** is a path $\langle v_0, v_1, v_2, \ldots, v_k \rangle$ where $v_0 = v_k$ and the path contains at least one edge. (a self-loop is a cycle of length 1)
- (U) A **cycle** is a path $\langle v_0, v_1, v_2, \ldots, v_k \rangle$ where $v_0 = v_k$ and the path contains at least 3 edges.
- (B) A **simple cycle** is a cycle where all vertices are distinct.
- (B) A graph with no cycles is **acyclic**.

10.1.4 Connectivity

- (B) The **connected components** of a graph are the equivalence classes of vertices under the "it's reachable from" relation.
- (U) An undirected graph is **connected** if it has *exactly one* connected component. In any connected, undirected graph $|E| \geq |V| - 1$.
- (D) A directed graph is **strongly connected** if every two vertices are reachable from each other. The *strongly connected components* of a directed graph are the equivalence classes of vertices under the "are mutually reachable" relation. A directed graph is strongly connected if it has *exactly one* strongly connected component
- (U) An **Eulerian circuit** is a cycle in an undirected graph which visits *each edge exactly once* and also returns to the starting vertex.
- (U) A **Hamiltonian cycle** is a cycle in an undirected graph which visits *each vertex exactly once* and also returns to the starting vertex.

10.1.5 Graph transformations

- (B) We say that a graph $G' = (V', E')$ is a **subgraph** of $G = (V, E)$ if $V' \subseteq V$ and $E' \subseteq E$. Given a set $V' \subseteq V$ the subgraph of G **induced** by V' is the graph $G' = (V', E')$, where $E' = \{(u, v) \in E : u, v \in V'\}$.
- (B) A **cut** is a partition of the vertices of a graph into two disjoint subsets. The **cut-set** of the cut is the set of edges whose end points are in different subsets of the partition. Edges are said to be **crossing** the cut if they are in its cut-set.
- (U) The **contraction** of an undirected graph $G = (V, E)$ by an edge $e = (u, v)$ is a graph $G' = (V', E')$, where $V' = V - \{u, v\} \cup \{x\}$ where x is a new vertex. The set of edges E' is formed from E by first deleting the edge (u, v) and, for each vertex w incident on u or v, deleting whichever of (u, w) or (v, w) is in E and then adding the new edge (x, w). This will produce a graph G' where u and v are "contracted" into a single vertex x.
- (U) The **directed version** of an undirected graph G is the directed graph G' obtained from G replacing each undirected edge (u, v) with the two directed edges (u, v) and (v, u).
- (D) The **undirected version** of a directed graph G is the undirected graph G' obtained from G by removing self-loops and "directions from edges".
- (D) The **transpose** of a directed graph $G = (V, E)$ is the graph $G = (V, E^T)$, where $E^T = \{(v, u) \in V \times V : (u, v) \in E\}$. Thus G^T is G with all its edges reversed.

10.1.6 Graph shapes

- (B) Two graphs $G = (V, E)$ and $G' = (V', E')$ are **isomorphic** if there exists a *bijection* $f : V \to V'$ such that $(u, v) \in E \iff (f(u), f(v)) \in E'$. In other words, we can "relabel" the vertices of G to be vertices of G' maintaining the corresponding edges in G and G'.
- (U) A **complete graph** is an undirected graph $G = (V, E)$ in which *every pair* of vertices is adjacent. Also, $|E| = \frac{n(n-1)}{2}$.
- (U) A **clique** in an undirected graph $G = (V, E)$ is a subset of the vertex set$C \subseteq V$, such that for every two vertices in C, there exists an edge connecting the two. This is equivalent to saying that the subgraph induced by C is complete (in some cases, the term clique may also refer to the subgraph itself).
- (U) A **bipartite graph** is an undirected graph $G = (V, E)$ in which V can be partitioned into two sets V_1 and V_2 such that $(u, v) \in E \implies (u \in V_1 \wedge v \in V_2) \vee (u \in V_2 \wedge v \in V_1)$. Also, G is bipartite if and only if every cycle has *even* length.

10.2 Trees

10.2.1 Definitions

- An acyclic, undirected graph is a **forest**.
- A connected, acyclic, undirected graph is a **(free) tree**.
- A **spanning tree** T of a connected, undirected graph $G = (V, E)$ is a subgraph of G such that $T = (V, E')$; that is a tree "covering" all vertices of G.

10.2.2 Properties

The following are all equivalent.

- G is a tree.
- Any two vertices in a tree are connected by a unique simple path.
- G is connected but if you remove a single edge from E the resulting graph becomes disconnected.
- G is acyclic but if you add a single edge to E then the resulting graph contains a cycle.
- $|E| = |V| - 1$.

10.2.3 Rooted trees

- We call any node y on the unique single path from the root r to x and **ancestor** of x.
- If y is an ancestor of x, then x is a **descendant** of y.
- The **subtree rooted at x** is the tree induced by descendant of x, rooted at x.
- The **degree** of a node x in a rooted tree$T$ equals the number of children of a node.
- The **depth** of a node x is the length of the simple path from the root r to x.
- A level of a tree consists of all nodes at the same depth.
- The **height** of a node x in a rooted tree is the number of edges on the longest simple *downward* (don't go back to the root) path from x to a leaf and the height of a tree is the height of its root.
- An **ordered tree** is a rooted tree in which the children of each node are ordered.

10.2.4 Binary and positional trees

- A binary tree T is a structure defined on a finite set of nodes that either
  - contains no nodes, or
  - is composed of three disjoint sets of nodes, a root node, a binary tree called its *left subtree*, and a binary tree called its *right subtree*.[1]
- A **full binary tree** is a binary tree where each node is either a leaf or has degree exactly 2.
  - The **number of leaves** in a full binary tree is one more the number of internal nodes.
- A **complete binary** tree is a binary tree in which every level, *except possibly the last*, is completely filled, and all nodes are as far *left* as possible.
  - The **height** in a complete binary tree with n leaves is $\lceil \log n \rceil$.

[1] This implies that a binary tree is an ordered tree (at least, but there is more).

  - If **all the h levels** are completely filled then:
    * The number of leaves is 2^h.
    * The number of internal nodes is $2^h - 1$.

- Full VS complete binary trees:
  - All nodes that are not leaves have two children in a full binary tree, this is not true for complete binary trees because there can be a node with just the left children (the last one that has been filled)
  - A complete binary tree becomes a full binary tree every time all nodes have exactly two children (including the last one that has been filled)
  - A full binary tree has no shape restriction (can have "holes" in the middle) while a complete binary tree does.
  - In a complete binary tree all the leaves must be on level h or $h-1$ while this is not true for full binary trees (can have deep left/right side and very shallow right/left side)

- The minimum height of a binary tree with n nodes is $\lfloor \log n \rfloor$.

10.3 Weights in a graph

Let $G = (V, E)$ be a simple graph and let λ be a function from E to the *positive*[2] real numbers. We call $\lambda(e)$ the **weight** of the edge e. If $H = (V', E')$ is a subgraph of G, then $\lambda(H)$, the weight of H, is the sum of $\lambda(e')$ over all $e' \in E'$.

- A **minimum weight spanning tree** for a connected graph G is a spanning tree such that $\lambda(T) \leq \lambda(T')$ whenever T' is another spanning tree.

10.4 Graph coloring

Given an undirected graph $G = (V, E)$ a **graph coloring** is a way of coloring the vertices of a graph such that no two adjacent vertices share the same color; this is called a vertex coloring. A coloring using at most k colors is called a (proper) k-coloring. The smallest number of colors needed to color a graph G is called its chromatic number, $\chi(G)$. Some facts about graph coloring follow:

- $1 \leq \chi(G) \leq n$, since assigning distinct colors to distinct vertices always yields a proper coloring.
- $\chi(G) \leq \Delta(G) + 1$, since it is possible to devise a greedy algorithm that produces such a coloring.
- Since complete graphs have $\Delta(G) = n-1$ and $\chi(G) = n$, and odd cycles have $\Delta(G) = 2$ and $\chi(G) = 3$, for these graphs this bound is the best possible.
- A graph can be colored with **1 color** if and only if $E = \emptyset$.
- Determining if a graph can be colored with **2 colors** is equivalent to determining whether or not the graph is bipartite, and thus computable in linear time using breadth-first search.
- Generally deciding if a given graph admits a k-coloring is NP-complete.

[2] We must be careful because sometime this is not always true and there are cycles of negative length for which some algorithms don't work.

10.5 Matchings

Given an undirected graph $G = (V,E)$, a **matching** M in G is a set of pairwise non-adjacent edges; that is, no two edges share a common vertex.

- A vertex is **matched** (or **saturated**) if it is incident to an edge in the matching. Otherwise the vertex is **unmatched**.
- A **perfect matching** (or **complete matching**) is a matching which matches all vertices of the graph. That is, every vertex of the graph is incident to exactly one edge of the matching.
- A **stable matching** is a matching that is perfect and stable. Stable means there is no element A of the first matched set that prefers an element B (that it's not matched to) of the second matched set, and at the same time B also prefers A over the one element it is currently matched with.