Final Cribsheet

Analysis of Algorithms

Illya Starikov

December 10, 2017

Recursion

- A program is **recursive** if it calls or refers to itself.
- The structure of recursion is as follows:

```
    if CONDITION:
        DIRECT CASE
    else:
        RECURSIVE CASE
```

- The 3–5 problem is as follows:

```
def f35(n):
    if n < 8:
        return "Error"
    elif n == 8:
        return (1, 1)
    elif n == 9:
        return (3, 0)
    elif n == 10:
        return (0,2)
    else:
        m3, m5 = f35(n - 3)
        return (1 + m3, m5)
```

- Recursive Insertion Sort is as follows:

```
def InsSortR(L):
    if len(L) < 2:
        return L
    else:
        return Insert(L[-1], InsSortR(L[:-1]))
```

```

def Insert(e,sL):
    if 0 == len(sL):
        return [e]
    elif (e >= sL[-1]):
        return sL+[e]
    else:
        return Insert(e, sL[:-1]) + [sL[-1]]
```

- Euclid's Elements is the most important book ever published.
  - It laid the foundations for modern mathematics and science.
- `GCD` can be defined as follows

```
def GCD(a, b):
    if (a % b == 0):
        return b

    return GCD(b, a % b)
```

Recursion Part II

- The Tower of Hanoi has runtime $2^n - 1$.

```
def H(n,a,b):
    if n == 0:
        return []
    if x == y:
        return []

    m = otherNeedle(x, y)
    return Hanoi(n - 1, x, m) + [(n, x, y)] + Hanoi(n - 1, m, y)

    def otherNeedle(n1,n2):
        if n1 == n2:
            return "ERROR -- Two needles are the same!"
            J = ['A','B','C']

        J.remove(n1)
        J.remove(n2)
        return J[0]
```

- To generate anagrams:

```python
def anagram(st):
    if st == '':
        return ['']

    lout =[]
    for i in range(len(st)):
        st2 = st[:i] + st[i+1:]
        lout2 = anagram(st2)

        for w in lout2:
            lout.append(st[i]+w)
            return lout
```

Proof by Recursion

- A **property** P **on a domain** D is a function D that accepts inputs from the domain D and return a Boolean value. If P is a property on D, and d is in D, then we say that P **holds for** d if $P(d)$ is true. Similarly, P **does not hold for** d if $P(d)$ is false.

- The proof by recursion has the following steps:

  Define the Problem Clearly state the objects you are dealing with, provide names and definitions for all functions you are talking about, clearly state what you are trying to prove.

  Check the Stopping Values + Two More Checking that whatever you are tying to prove is required for the stopping value, but optional for two other values.

  Check that the Recursion Stays in D Prove that if recursion in P is triggered by values in D, then the values in the call are also in D.

  Check That Recursion Halts We use the **counting strategy**: assign a **non-negative** integer as a counter to every value in the domain D. If whenever recursion is triggered, the counter associated with called values are $<$ the value associate with the calling value, recursion will halt!

  Check Inheritance You may assume that P is true for all values called recursively; however, you must then prove that P holds for the calling value.

  Conclude the Proof Invoke the Principle of Recursion.

Induction

- Some notes on The Modulus (%)
  - $P\%Q$ is always between 0 and $Q-1$.
  - $(P+R)\%Q = (P\%Q + R\%Q)\%Q$

  - $P = S \times Q + P\%Q$.
  - $(P \times R)\%Q = (P\%Q) \times (R\%Q)$

- Mathematical induction is like recursion except that:
  - Uses predicates vs. recursive program
  - The domain is always a set of **integers**.

- Proof by simple induction has the following steps:

  Define the problem Clearly define the domain D, which must be of the form $D = s \ldots \infty$.

  Check Base Case & Two Other Values

  Prove for all $n > s$, that if $P(n-1)$ is true, then $P(n)$ is true

  Conclude the proof

- The principle of well-ordering is as follows

  > Every non-empty set of natural numbers has at least one element.

- To Euclid, **axioms** are statements that are true in all domains. Postulates are statements that are true in some domain.

- Via Peano's postulates, there is a set called N consisting of objects we call "numbers". There is also a operation of *successor* denoted by $*$ from N into N.

  1. There is an objected called 0.
  2. For all p, $p* \neq 0$
  3. If $p* = q*$, then $p = q$.
  4. If Q is a subset of N, such that
     - 0 is in Q
     - If q in Q, then $q*$ in Q
     - Then $Q = N$

Big Oh Algorithms

Big-O $O(g(n)) = \{f(n) :$ there exists positive constant c and n_0 such that $0 \leq f(n) \leq c\, g(n)$ for all $n \geq n_0\}$

Big-Ω $\Omega(g(n)) = \{f(n) :$ there exists positive constants c and n_0 such that $0 \leq c\, g(n) \leq f(n)$ for all $n \geq n_0\}$

Big-Θ $\Theta(g(n)) = \{f(n) :$ there exists positive constants c_1, c_2 and n_0 such that $0 \leq c_1\, g(n) \leq f(n) \leq c_2\, g(n)$ for all $n \geq n_0\}$

- A loop invariant needs to satisfy 3 conditions:

  Initilization Must be correct before the loop begins.

  Maintenance Each iteration maintains loop property.

  Termination Property holds when loop terminates.

Incremental Improvement Improve things one step at a time.

Divide and Conquer Divide the problem into a number of subproblems that are instances of the same problem and put your solution together from the sub-solutions.

Fibonacci Numbers, Sums, and Series

- The Fibonacci numbers are defined as follows:

$$F_n = \begin{cases} 0 & \Leftrightarrow n = 0 \\ 1 & \Leftrightarrow n = 1 \\ F_{n-1} + F_{n-2} & \Leftrightarrow n > 1 \end{cases}$$

- A *sequence* is a function $f : \mathbb{N} \to \mathbb{R}$ where we denote $f(n)$ by a_n.
- A *series* is a sequence S_n associated with another sequence a_n such that $S_n = \sum_{k=0}^{n} a_k$.
- Given a sequence we will refer to the partial sums as the *associated series.*
- A *arithmetic progression* is a series where each term in the associated sequence differs from the preceding term by a constant.
- A *geometric progression* is a series were each term in the associated sequence is a constant multiple of the preceding term.
- The harmonic series is defined as

$$H_n = \sum_{k=1}^{n} \frac{1}{k}$$

- If f is monotonically increasing, then

$$\int_{m-1}^{n} f(x)\,dx \leq \sum_{k=m}^{n} f(k) \leq \int_{m}^{n+1} f(x)\,dx$$

- If f is monotonically decreasing, then

$$\int_{m-1}^{n} f(x)\,dx \geq \sum_{k=m}^{n} f(k) \geq \int_{m}^{n+1} f(x)\,dx$$

- for $|r| < 1$,

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

Disjoint Sets

- For all equivalence relations, $a\,R\,b$ and $b\,R\,c \implies a\,R\,c$, so once there is an intersection between two blocks in a partition, those blocks need to be joined into a single block.

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

- We can approximate $n!$ by

$$n! \approx \sqrt{2\pi n}\left(\frac{n}{e}\right)^n$$

- Sterling's Formula is as follows:

$$\int_{1}^{n} \ln x\,dx \leq \sum_{k=2}^{n} \ln k \leq \int_{2}^{n+1} \ln x\,dx \tag{1}$$

- A set is a collection of objects.
- Some commons sets are:
  - $\varnothing$ The Empty Set
  - $\mathbb{N}$ The natural numbers
  - $\mathbb{Z}$ The integers
  - $\mathbb{Q}$ The rational numbers
  - $\mathbb{R}$ The real numbers
  - The set of all finite graphs
  - The set of all finite trees
  - The set of all strings

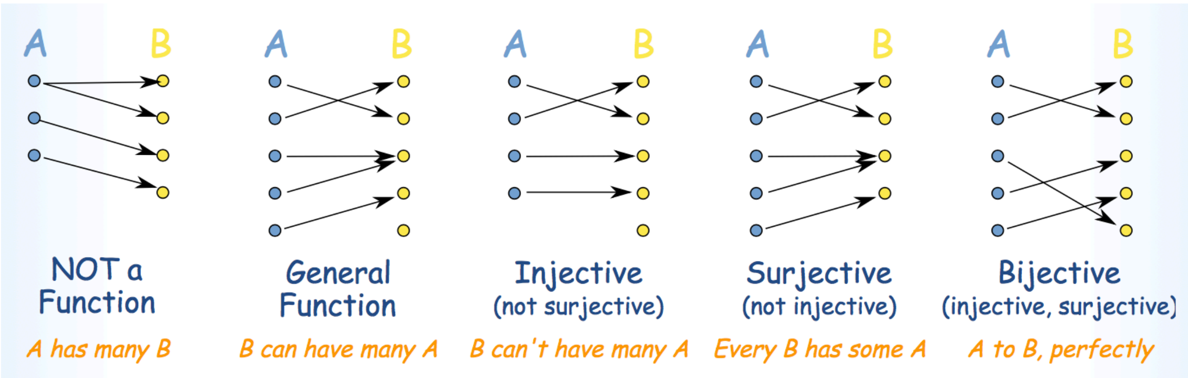
- DeMorgan's Laws

$$(A \cap B)' = A' \cup B' \qquad (A \cup B)' = A' \cup B'$$

- Absorption Laws

$$A \cap (A \cup B) = A \qquad A \cup (A \cap B) = A$$

- R is **reflexive** iff $\forall a \in A$, $a\,R\,a$, which means $(a, a) \in R$.
- R is **symmetric** iff $\forall a, b \in A$, $a\,R\,b \equiv b\,R\,a$.
- R is **antisymmetric** iff $\forall a, b \in A$, $a\,R\,b$ and $b\,R\,a \to a = b$.
- R is **transitive** iff $\forall a, b, c \in A$ and $a\,R\,b$, $a\,R\,b \to a\,R\,b$.
- R is an **equivalence relation** iff it is reflexive, symmetric and transitive.
- R is a **partial order** iff it is reflexive, antisymmetric and transitive.
- R is a **total order** iff it is a partial order such that $\forall a, b \in A$ either $a\,R\,b$ or $b\,R\,a$.



Graph Theory I

- Let P be a path of length $K > 0$ and let $i, j \in range(k+1) \mid P(i) = P(j)$. We define $Trim(p, i, j)$ to be a function $g : k + i - j + 1 \to \mathbb{V}$.
- Let p and q be paths in a graph G. q is said to be **reduction** of p iff q can be derived from p by a finite number of trimming operations.
- The set of all paths that can be obtained by applying finite sequences of trimming operations to p is called the set of **reduced path of** p or **reductions of** p
- In a finite graph with even a single edge, the number of paths is infinite.

- We call the equivalence classes of R the **connected components** of G.
- A **cycle** is a path of length ≥ 3 that begins and ends at the same vertex — *self-loops are not allowed in graphs.*
- A **simple cycle** is a cycle that has no repetition of vertices except for the first and last vertex.
- A simple cycle is not a simple path.
- If there are two different simple paths between a and b in a graph, G, then there is a simple cycle in G.
- A **simple path** is a path such that no vertex appears more than once. (if $f : \underline{n} \rightarrow \mathbb{V}$ it's an *injection*)
- If $\exists P = (a, \ldots, b)$, then b is **Reachable** from a
  - let R be the relation on $\mathbb{V} \iff b$ is reachable from a.
  - R is *reflexive*: aRa given by $f(0) = a$
  - R is *symmetric*: $aRb \rightarrow bRa$ given by *Reverse Paths*
  - R is *transitive*: $aRb, bRc \rightarrow aRc$ given by *Path Concatenation*
  - Equivalence Classes of R are all the connected Components of a graph G.
- A graph is **acyclic** iff it does not contain simple cycles.
- Two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are said to be **isomorphic** iff there is a bijection $f : V_1 \rightarrow V_2$ such that $\{a, b\} \in E_1$ iff $\{f(a), f(b)\} \in E_2$.
  - Basically, if the graphs are the same.
- No Euler path is possible if there are more than 2 odd degree vertices.
  - The converse is also true, there exists an Euler path if a graph has 2 or 0 vertices of odd degree.

Euler's Formula (Digraphs)

- For a planar connected graph, we have

$$V - E + F = 2$$

- True for closed shapes because $E = V$ and $F = 2$.
- For any planar graph with $V \geq 3$, we must have $E \leq 3V - 6$.
- As a result of Euler's Formula,

  - K_3 is not planar.
  - $K_{3,3}$ is not planar.
  - There are only 5 regular solids.
  - A soccer ball must have exactly 12 pentagon

- Any planar graph has a vertex of degree ≤ 5.

Graph Theory II

- If $G = (V, E)$ is an undirected graph, then $\sum_{v \in V} \deg(v) = 2|E|$.
- A planar graph is a graph that can be drawn in the plane without having any lines cross.

Master Theorem & Fast Multiplication

Theorem 1. *Let $a \geq 1$ and $b > 1$ be constants, let $f(n)$ be a function, and let $T(n)$ be defined on the non-negative numbers by the recurrence*

$$T(n) = a\,T(^n/_b) + f(n) \tag{2}$$

where we interpret $^n/_b$ to mean either $\lfloor ^n/_b \rfloor$ or $\lceil ^n/_b \rceil$. Then $T(n)$ has the following asymptotic bounds:

1. *If $f(n) = \mathcal{O}\left(n^{\log_b a - \epsilon}\right)$ for some constant $\epsilon > 0$, then $T(n) = \Theta\left(n^{\log_b a}\right)$.*
2. *If $f(n) = \Theta\left(n^{\log_b a}\right)$, then $T(n) = \Theta\left(n^{\log_b a} \lg n\right)$.*
3. *If $f(n) = \Omega\left(n^{\log_b a + \epsilon}\right)$ for some constant $\epsilon > 0$, and if*

$$a\,f(^n/_b) < c\,f(n)$$

   for some constant $c < 1$ and sufficiently large n, then $T(n) = \Theta\left(f(n)\right)$.

- To use the substitution method,
  1. Guess the form of the solution
  2. Use Mathematical Induction to find the constants and show that the solutions works
- The sum of two homogeneous solutions is a homogeneous solution
- Multiplying a homogeneous solution by a constant gives a homogeneous solution
- The linear combination of homogeneous solutions is a homogeneous solution

- For the Fibonacci recursion

$$F_n + F_{n-1} - F_{n-2} = 0$$

- The associated equations is $x^2 - x - 1 = 0$, which has the roots $\frac{1 \pm \sqrt{5}}{2}$.
  - Note that $\frac{1+\sqrt{5}}{2}$ the golden ration ϕ.

Graphs & Trees

- A digraph (V, A) is **bipartite digraph** iff we can write $V = B \cup C$, where $B \cap C = \varnothing$ and all arcs have their tails in B and their heads in C.
- A **forest** is an acyclic graph
- A **free tree** is an acyclic, connected graph
- A **DAG** is a directed, acyclic graph
- A **rooted tree** is a triple (V, E, r) where (V, E) is a free tree and $r \in V$ is a distinguished vertex called the **root**.
- We say that a is **proper ancestor** of b iff a is an ancestor of b, but $a \neq b$.
- We say that a is **proper descendant** of a iff b is an ancestor of a, but $a \neq b$.
- A **positional tree** is an ordered tree where the children are labeled by distinct positive integers
- A k**-ary** tree is a positional tree where the labels are $\leq k$.
- A **complete** k**-ary** tree is a k-ary tree where each non-leaf node has all k children and all leaves have the same depth.
- $G = (V, E)$. The following are equivalent.
  - G is a free tree (connected and acyclic).
  - Any two vertices in G are connected by a unique simple path.
  - G is connected, but removing any edge disconnects G
  - G is connected and $|E| = |V| - 1$.
  - G is a acyclic and $|E| = |V| - 1$
  - G is acyclic but adding any edge creates a cycle.
- If G is connected then $|E| \geq |V| - 1$.

Disjoint Set Graph Algorithms

```
DFS(G)
for each vertex u ∈ G.V
    u.color = WHITE
    u.π = NIL

time = 0
for each vertex u ∈ G.V
    if u.color == WHITE
        DFS-VISIT(g, u)

DFS-VISIT(G, u)
time = time + 1
u.d = time
u.color = GRAY

for each v ∈ G.Adj[u]
    if v.color == WHITE
        v.π = u
        DFS-VISIT(g, v)

u.color = BLACK
time = time + 1
u.f = time
```

```
BFS(G,s)
for each vertex u ∈ G.V − {s}
    u.colors = WHITE
    u.d = ∞
    u.π = NIL

s.color = gray
s.d = 0
s.π = NIL
Q = ∅

ENQUEUE(Q, s)
while Q ≠ ∅
    u = DEQUEUE(Q)

    for each v ∈ G.Adj[u]
        if v.color == WHITE
            v.color = GRAY
            v.d = u.d + 1
            v.π = u
            ENQUEUE(Q, v)
    u.color = BLACK
```

Disjoint Sets Graph Algorithms II

- To topologically sorting a *DAG*

1. Apply `DFS` to the digraph D
2. As each node finishes, put it at the beginning of the list
3. Return the list

Probability

- Using set notation, a *Sample Space* is a set.
- An event is just a subset of a sample space S
- An **elementary** is just a point in the sample space S.
- A **certain event** is just S
- A **null event** is just $\varnothing$
- Two events are **mutually exclusive** iff their intersection is $\varnothing$
- Nothing is random about sample spaces and events.
- A **discrete probability distribution** on a sample space S is just a function $f : S \to \mathbb{R}$
  1. $f(x) \geq 0, \forall x \in S$
  2. $\sum_{x \in S} f(x) = 1$
- A **probability space** is a pair (S, f), where is a S is a sample space and f is a probability distribution on S.
- Given a probability space (S, f) and $E \subset S$ an event, the **probability** of E is denoted $\Pr(E) = \sum_{x \in E} f(x)$
- The **uniform distribution** is defined for *finite* sample spaces as follows:
  - $f : S \to \mathbb{R}$ is given by $f(x) = \frac{1}{|S|} \forall x \in S$
  - Usually assume uniform distribution unless have other evidence
- The normal distribution is a distribution over the reals, $\mathbb{R}$, so we will not use it much.
- The **conditional probability of** A **given** B denoted $\Pr(A|B)$ is **defined** as $\frac{\Pr(A \cap B)}{\Pr(B)}$
- Bayes's Theorem is as follows:

$$\Pr(A|B) = \frac{\Pr(A)\Pr(B|A)}{\Pr(B)}$$

- A **random variable** is a function from a sample space into the real numbers.
- The **density probability function** of X, written $f_x(r) = \Pr(X^{-1}(r)) \forall r \in \mathbb{R}$.
- The **expected value of random variable** V, $E(V)$, is defined by

$$E(V) = \sum_{x \in S} V(x) \Pr(\{x\})$$

  This is just the average value of V.

- For all probability spaces and random variables, $\min \{V(x) | x \in S\} \leq E(V) \leq \max \{V(x) | x \in S\}$

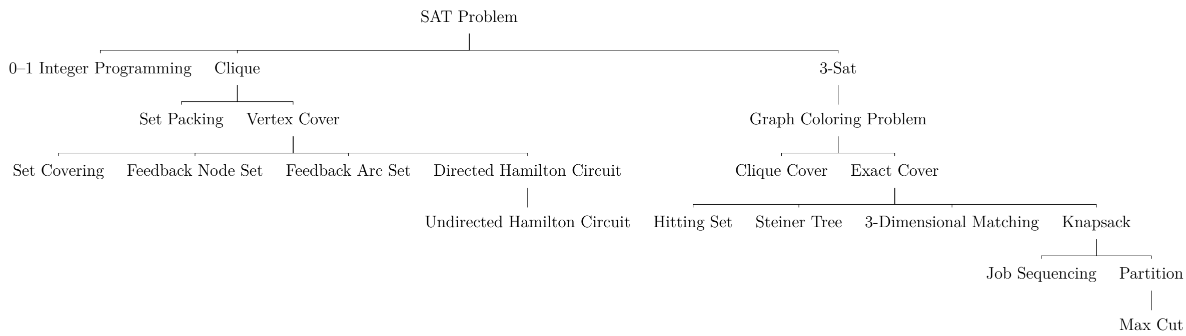
Probability II

- Chebychev's theorem is as follows

$$\Pr(|X - \mu| \geq t\sigma) \leq \frac{1}{t^2}$$

Heaps

- In-place sorts sort an array using the space occupied by the array and a constant amount of additional space
- Sort an array using a non-fixed amount of space in addition to the amount of space occupied by the array
- **Max Heaps** are binary trees with the following properties
  1. Level k in the tree must have 2^k nodes before level and $k + 1$ can have any nodes
  2. If level k does not have 2^k nodes, the nodes it has must be arranged from left to right
  3. The value stored in a node is $\geq$ the value of it's descendent.
- Min heaps are similar but Condition 3.
- The left child of node number i is $2i$
- The right child of node number i is $2i + 1$
- The parent of node i is $\lfloor \frac{i}{2} \rfloor$
- The height of a heap is $\lceil \lg n \rceil$



Quicksort Median

- For quicksort

  Worst Case $\Theta(n^2)$

  Best Case $\Theta(n \log n)$

  Expected Case $\theta(n \log n)$

- A sort is **stable** if whenever $A[i] = A[j]$ and $i < j$ in the original data, then $A[i]$ will precede $A[j]$ in the output.

- Counting sort and radix sort are both stable.

NP-Complete Problems

- NP-Complete problems have two properties.
  - There problems have a yes/no answer.
  - There is a polynomial time algorithm that can verify whether a purported solution is indeed a solution.

- To be an NP-Complete problem, a problem must be in the class NP and have the property, that if a polynomial time algorithm can be discovered to answer it, then all problems in the class NP have a polynomial time algorithm.

- Given that one problem is NP complete, we can show that another problem is NP-Complete in two steps:
  1. Prove that the second problem is in NP.
  2. Prove that if we could solve the second problem in Polynomial Time, we could solve the NP complete problem in polynomial time.
     - This generally means that we have to show a polynomial time transformation from the NP complete problem to the second problem.

Discrete Mathematics & Computer Science

- For approximation algorithms, we want to find a bounds that look like

$$\max\left(\frac{\text{solution}}{\text{optimal}}, \frac{\text{optimal}}{\text{solution}}\right) \leq r(n)$$

- We use $\frac{\text{solution}}{\text{optimal}}$ for min problems and $\frac{\text{optimal}}{\text{solution}}$ for max problems.
- The relative error δ is defined as

$$\delta = \frac{|\text{solution} - \text{optimal}|}{|\text{optimal}|}$$

- The random method of generating vertex covers can can produce ratio $\frac{|\text{cover}|}{|\text{optimal}|} \leq 2$.
- If $P \neq NP$ and k is an integer, there does not exist a poly-time algorithm for optimal TSP such that $\text{cost}\,(\text{Approximation}) \leq k \times \text{cost}\,(\text{Optimal})$.

Greedy Algorithms

- Greedy algorithms are algorithms that make locally optimal choices in the hope that they can produce a global optimal.
  - Sometimes they succeed, sometimes not.
  - They always produce something.
- For the scheduling problem, arranging activities so that finishing times are monotone increasing and then taking activities in order as long as they don't overlap with previously chosen activities yields a best answer.

Amortized Analysis

- This is a combination of worst-case and average-case analysis.
- The basic idea is that you have some very expensive operations but they happen rarely.
- You figure out a way to "average" these costs over the mean cheap operations in various ways.