Mathematics

Monotonicity
P.53

Monotonically : if $m \leq n$
- increasing : $f(m) \leq f(n)$
- decreasing : $f(m) \geq f(n)$

Strictly : if $m < n$
- increasing : $f(m) < f(n)$
- decreasing : $f(m) > f(n)$

Floor and Ceiling
P.54

$\lceil \frac{a}{b} \rceil \leq \frac{a+(b-1)}{b}$ — $\lceil \frac{a}{b} \rceil$ round up

$\lfloor \frac{a}{b} \rfloor \geq \frac{a-(b-1)}{b}$ — $\lfloor \frac{a}{b} \rfloor$ round down

$\lceil \frac{n}{2} \rceil + \lfloor \frac{n}{2} \rfloor = n$

Modular arithmetic
P.54

$a \bmod b = a - b\lfloor a/b \rfloor$

$0 \leq a \bmod n < n$

If $a \bmod n = b \bmod n$

$a \equiv b \bmod n$

Polynomial of degree of d
P.55

$$P(n, d) = \sum_{i=0}^{d} a_i n^i$$

A polynomial is Asymptotically

Positive : $a_d > 0$

negative : $a_d < 0$

$f(n)$ is polynomially bounded if $f(n) = O(n^k)$ for $k \geq 0$

Exponentials
P.55

$a^0 = 1$, $a^1 = a$, $a^{-1} = 1/a$

$a^{mn} = (a^m)^n$, $a^m a^n = a^{m+n}$

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} \ldots$$

Logarithm Properties :
P.56

$y = b^x$ $\log_b(y) = x$ $b^{\log_b(y)} = y$

$\log(XY) = \log(X) + \log(Y)$, $\log(X/Y) = \log(X) - \log(Y)$

$\log(X^n) = n\log(X)$, $\log_b(n)\log_b(a) = \log_a(n)$

$\log_b 1 = 0$, $\log_b b^x = x$ $\log_a b = 1/\log_b a$

$\log_b(\frac{1}{a}) = -\log_b(a)$

$f(n)$ is polylogarithmically bounded if $f(n) = O(\lg^k n)$

Factorials
P.57

$n! = 1 \times 2 \times 3 \cdots n$

$0! = 1$

Stirling's approximation

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

$\lg(n!) = \Theta(n \lg n)$ P.58

Fibonacci Numbers
P.59

$F_0 = 0$, $F_1 = 1$ $F_i = F_{i-1} + F_{i-2}$

Φ_i and $\hat{\Phi}_i = \pm\sqrt{i+1}$

$$F_i = \frac{\Phi_i - \hat{\Phi}_i}{\sqrt{5}}$$

Summation

Properties:

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

$$\sum_{k=1}^{n} \Theta(f(k)) = \Theta\left(\sum_{k=1}^{n} f(k)\right)$$ P.1146

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

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

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

$$\sum_{0}^{n} x^k = \frac{x^{n+1}-1}{x-1}$$ P.1147

$$\sum_{0}^{\infty} x^k = 1/(1-x)$$

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

Data Structure

Heaps

- Array object seen as a binary tree

Parent(i) = $\lfloor \frac{i}{2} \rfloor$
Left(i) = $2i$
Right(i) = $2i+1$

- Essentially complete (complete except last depth)
- Verticles consequtive filled L→R

Methods:

Max-heapify(A, i) P.154

```
if A[l] > A[i]
  max = l
else if A[r] > A[i]
  max = r
else
  max = i

if max ≠ i
    swap(A, i, max)
    max-heapify(A, max)
```

Complexity: $\Theta(\lg n)$ P.156

Build-Max-Heap(A) P.157

for $i = \lfloor \frac{|A|}{2} \rfloor : 1$ max-Heapify(A, i)

Complexity: $O(n)$

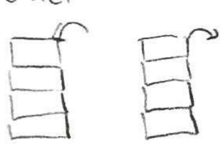
Priority Queue using a heap

Find max : $O(1)$ P.163
Extract max : $O \lg n)$ P.163
Increase key : $O(\lg n)$ P.164
Insert : $O(\lg n)$ P.164

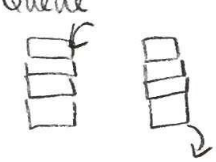
Stacks: LIFO push / pop

Queue: FIFO enqueue/dequeue

Stack



Queue



Linked List

- Object with reference to next/previous
- Head / tail.

Direct-address Table

- use key as index directly. $O(1)$ for all

Hash table P.258

- uses hash function h to diret key to slot
- resolve conflict by chaining

Insert : $O(1)$
search : worst case $O(n)$
delete : same as search (singly)
$O(1)$ (doubly)

expected search: $\Theta(1+\alpha)$ P.259

Hash Function

- intepret key as $\mathbb{N}$ P.263

Division Method : $h(K) = K \bmod m$ m = slots

Multiplication Method : - multiply K by A : $0 < A < 1$
- take fraction of KA
- $h(K) = m(KA \bmod 1)$

Universal Hashing P.265

Given a finite set $\mathcal{H}$ of hash functions that map a given universe U of keys into $\{1 : m\}$

such set $\mathcal{H}$ is said to be universal if $\forall k, l \in U$, $\forall h \in \mathcal{H}$ $h(k) = h(l)$ is at most $|\mathcal{H}|/m$. No more than $\frac{1}{m}$ chance

Designing a universal hash : P.267

Open Addressing P.269

Analysis P.274

Perfect Hashing

- use a second hash table to handle collision Pg 278
- Static set of keys only

Ensures $O(1)$ worst performance

Asymptotics
- study of the growth of a function

Notations: O, Θ, Ω

Big O. upper bound P.47

$f(n) = O(g(n)) \Rightarrow \exists c: \exists n_0: \forall n. n \geq n_0 \Rightarrow 0 \leq f(n) \leq cg(n)$

Big Ω. lower bound P.48

$f(n) = \Omega(g(n)) \Rightarrow \exists c: \exists n_0 \forall n\ n \geq n_0 \Rightarrow 0 \leq cg(n) \leq f(n)$

Big Θ, tight bound P.44

$f(n) = \Theta(g(n)) \Rightarrow \exists c_1 \exists c_2 \exists n_0 \forall n.\ n \geq n_0 \Rightarrow c_1 g(n) \leq f(n) \leq c_2 g(n)$

Limit test

$f(n) = O(g(n))$:

$$\lim_{n \to \infty} f(n)/g(n) = \begin{cases} 0 \\ c \in \mathbb{R}^+ \end{cases}$$

$f(n) = \Omega(g(n))$:

$$\lim_{n \to \infty} f(n)/g(n) = \begin{cases} c \in \mathbb{R}^+ \\ +\infty \end{cases}$$

$f(n) = \Theta(g(n))$:

$$\lim_{n \to \infty} f(n)/g(n) = c \in \mathbb{R}^+$$

$\alpha = \{O, \Omega, \Theta\}$ any

Properties: P.51

- Transitivity:

  $f(n) = \alpha(g(n))$ and $g(n) = \alpha(h(n))$
  then $f(n) = \alpha(h(n))$
- Reflexive:

  $f(n) = \alpha(f(n))$

Recurrence: writing runtime of a function in terms of itself

How to Solve:

- Recurrence tree P.88
- Manipulate formula
- Substitution and induction Pg.83
- Master's method Pg.94
- Master's theorem Pg.94

  Proof P.98

Master's Method P.94

Recurrences given in the form:

$$T(n) = aT\left(\frac{n}{b}\right) + cn^k \text{ if } n \geq n_0$$
$$T(n) = \Theta(1) \text{ or } c \text{ if } n < n_0$$

$$T(n) = \begin{cases} \Theta(n^k) & a < b^k \\ \Theta(n^k \log_2 n) & a = b^k \\ \Theta(n^{\log_b a}) & a > b^k \end{cases}$$

By master's method

Master's Theorem P.94

Recurrence given in the form:

$$T(n) = aT\left(\frac{n}{b}\right) + f(n)$$

Case 1:

$f(n) = O(n^{\log_b a - \epsilon})$ for some $\epsilon > 0$

$T(n) = \Theta(n^{\log_b a})$

Case 2:

$f(n) = \Theta(n^{\log_b a})$

$T(n) = \Theta(n^{\log_b a} \log_2 n)$

Case 3:

$f(n) = \Omega(n^{\log_b a + \epsilon})$ for some $\epsilon > 0$

and if $af(n/b) \leq cf(n)$ for some $c < 1$

$T(n) = \Theta(f(n))$

Sorting Algorithms

Insertion-Sort P.16 (Stable)

- maintain A[1: J-1] sorted
- insert A[J] into A[1:J-1]
- initialize sorted to be A[1]
- insert A[J] for J = 2:n

complexity: $\Theta(n^2)$ P.28

Merge-Sort P.34 (Stable) recursive

- Divide and conquer algorithm
- Divide pre-sorted array into two equal halves
- Conquer by sorting sub-Array with merge-sort
- combine the solution by Merging (P.37)
- Merge compares both sub-array from $i=1:\lfloor \frac{n}{2} \rfloor$ and put it on a new array

complexity: $\Theta(n \log_2 n)$ P.37

Quick-sort P.171 (unstable) recursive

- Choose a pivot p and partion the array such that A[1: P-1] is $\leq$ to p and A[P+1 :n] is $\geq$ to p
- Divide and conquer
- Divide the array by the pivot
- Conquer by recursively sorting A[1:P-1] & A[P+1:n]

Complexity: $\Theta(n^2)$ worst case P.175

$\Theta(n \log_2 n)$ average P.175

Heap-sort P.160 (unstable)

- Uses a datastructure heap P.151
- extract the root of the heap and place it into a new sorted array
- maintain heap's properties throughout sort

Complexity: $\Theta(n \log_2 n)$ P.160

Counting-sort P.195 (Stable) Linear

- input A[1:n] of integers from 0:K for some K
- requires two other array B[1:n] to hold the output and C[0:K] to provide working storage
- count the number of times an integer from 0:K appears by going through J=1:n, C[A[J]]+1
- The algorithm then now do a running sum of C for 0:K, C[i] ± C[i-1] such that C[i] stand for how many elements are before i
- Now place A[j] from n:1 depending on C[A[j]] and decrement C[A[j]] after placement.

complexity: $\Theta(n+k)$ P.196

if $k = O(n)$ then $\Theta(n)$

Radix-sort Pg. 198 (stable) linear

- input A[1:n] of integers with d-digits where each digit can take on K possible values
- go from lowest to highest significance digit and sorting using a stable sort

Proof of correctness: Pg 198

Complexity: $\Theta(d(n+K))$ if each stable sort takes $\Theta(n+K)$

$\Theta(n)$ if d is constant and $K = O(n)$ Pg 198

Bucket-Sort Pg 201 (Stable) linear

- Assume input A[1:n] contain random elements (double) from 0:1

Lower bounds for Sorting Pg 193

Proof: lower bound of any comparison algorithm is $\Omega(n \log n)$

I. Given input A[1:n] and the algorithm always makes $\leq$ K comparisons. Assume A[i] are unique and randomly arranged with n! permutations

II. Across all n! Permutation inputs, algorithm exhibits $\leq 2^K$ distinct executions

III. If $2^K < n!$, then it execute identically on two distinct input (does not work)

IV. By III $2^K \geq n!$, $2^K \geq (\frac{n}{2})^{\frac{n}{2}}$

V. $K \geq \frac{n}{2} \log_2(\frac{n}{2}) = \Omega(n \lg n)$ QED

Selection Pg. 220

- finding the i'th smallest element
- max: i=n, min: i=1, upper median $i = \lceil \frac{n+1}{2} \rceil$ lowermedian $i = \lfloor \frac{n+1}{2} \rfloor$ (median = lower)

Divide: Pick a pivot x using median of median

Partition: Partition A using x, A[P] = x where x is the p'th smallest element if p = i, return x

Conquer: if p > i then select(L, i) if p < i then select(R, i-P)

Complexity: $O(n)$ Pg. 222

Number Theory

$\mathbb{Z} = \{\dots -2, -1, 0, 1, 2 \dots\}$ Integers

$\mathbb{N} = \{0, 1, 2 \dots\}$ Naturals

Elementary notions

Divisibility: $d \mid a \rightarrow a = kd \quad k \in \mathbb{Z}$
(d divides a)

trivial divisors: 1 and a

Prime and composite numbers

P 928

a is prime if it ONLY has trivial divisors

composite numbers are products of prime with an unique prime factorization

1 is a unit #, 0 is neither comp or prime.

Division Theorem

P 929

$0 \leq r \leq n \quad a = qn + r$

$n \mid a \iff r = a \bmod n = 0$

Greatest common divisor (gcd)

properties:

$gcd(a,b) = gcd(b,a)$
$= gcd(-a,b)$
$= gcd(|a|,|b|)$

$gcd(a,0) = |a|$

$gcd(a,ka) = |a| \quad k \in \mathbb{Z}$

useful GCD proofs P. 930

Relatively Prime

a & b are relative prime if $gcd(a,b) = 1$

Unique factorization

Given a $\mathbb{N}$ d not prime, with prime fact. of $\{P_0\, P_1\, P_2 \dots P_n\}$ Assume factorization not unique, $\{q_0\, q_1\, q_2 \dots q_m\}$. P_0 Ild $P_0 \mid \{q_0 \dots q_m\}$ since $q_0 \dots q_m$ are prime, $\exists i: q_i = P_0$ since q_i's divisors are only 1 or itself.

$\forall j: \prod_{i=0}^{j-1} P_i \mid \{q_0 \dots q_m\} \Rightarrow$ There exist $q_i = P_i$

Inductive proven.

QED

Euclid's Algorithm

$gcd(a,b) = Euclid(a,b)$

```
Euclid(a, b)
  if b == 0
     return a
  else return Euclid(b, a mod b)
```

Complexity: $O(\lg b)$ P 936

Extended Euclid

$d = gcd(a,b) = ax + by$

P 937

```
ext-Euclid(a, b)
  if b == 0
     return (a, 1, 0)
  else (d', x', y') = Ext-Euclid(b, a mod b)
     (d, x, y) = (d', y', x' - ⌊a/b⌋y')
     return (d, x, y)
```

Complexity $O(\lg b)$

Modular Arithmetic

P 940

Finite group $(S, \oplus)$ is a set S w/ binary operation $\oplus$ defined on S

- $a, b \in S, \quad a \oplus b \in S$
- $\exists I: I \in S \quad a \oplus I = I \oplus a = a$
- $a \oplus b \oplus c = (a \oplus b) \oplus c = a \oplus (b \oplus c)$
- $\forall a: \exists b: (a \oplus b) = (b \oplus a) = I$

If $a \equiv a' \pmod n$, $b \equiv b' \pmod n$

- $(a+b) \equiv (a' + b') \pmod n$
- $(ab) \equiv (a'b') \pmod n$

Modular addition/multiplication/exponentiation

$(A+B) \bmod C = (A \bmod C + B \bmod C) \bmod C$

$(A \cdot B) \bmod C = (A \bmod C \cdot B \bmod C) \bmod C$

$(A \wedge B) \bmod C = (A \bmod C)^B \bmod C$ (slow)

Modular Exponentiation algorithm P 957

Chinese Remainder Theorem

P. 950

$(a+b) \bmod n = ((a_1+b_1) \bmod n_1 + (a_2+b_2) \bmod n_2 \dots)$

$(a-b) \bmod n = (\text{"} \quad \text{"} - \text{"} \quad \text{"})$

$a \cdot b \bmod n = (\text{"} \quad \text{"} \times \text{"} \quad \text{"})$

m_i = Product of all n_j except n_i

$c_i = m_i (m_i^{-1} \bmod n_i)$

$a = (a_1 c_1 + a_2 c_2 + a_3 c_3 \dots) \bmod n \qquad n = n_1 \cdot n_2 \cdots n_i$

$m_i \cdot x_i \equiv 1 \bmod (n_i)$

$a = (a_1 m_1 x_1 + a_2 m_2 x_2 \dots) \pm n$

Greedy Algorithm

Steps:

1. Define optimal substructure of the problem
2. Develop a recursive solution
3. Show greedy choice means one sub-problem remains
4. Proof that it is safe to make the greedy choice
5. Develop a recursive algorithm then convert to iterative

Activity-selection Algorithm P.416

Optimal substructure:

S_{ij} = Set of activites that start after a_i finishes and before a_j starts

$$\left.\boxed{a_i}\ \ S_{ij}\ \ \boxed{a_j}\right\}A_{ij}$$

Suppose there is a maximum set A_{ij} and a_k is in its optimal solution it breaks the problem up into two sub problems

$$\underbrace{\boxed{a_i}\ \ S_{ik}}_{A_{ik}}\ \ \boxed{a_k}\ \ \underbrace{S_{kj}\ \ \boxed{a_j}}_{A_{kj}} \qquad A_{ik} = A_{ij} \cap S_{ik},\ A_{kj} = A_{ij} \cap S_{kj}$$

$$A_{ij} = \{a_k\} \cup A_{ik} \cup A_{kj}$$

Greedy choice:

- Select the activity with the earliest finish time, all others are are compatible with it   Proof P.418

Recursive P.419
Iterative P.421

Fractional Knapsack problem

- calculate value per pound
- carry as much items with max value / pound

1-0 Knapsack problem

- requires dynamic programming

Graph Algorithms

Breadth-first search

P. 596

- Starting point s
- maintain a queue, add s to the queue
- dequeue and explore all edges and add them to the queue

Complexity: $O(V+E)$

Proofs: pg 598

Depth-first search

P. 605

- Starting at point s
- maintain a stack, highlight s and add s to the stack.
- explore one edge of s and put it on the stack and so one and so forth
- When an edge has no more outedges that are not explored, pop it off the stack

Complexity: $\Theta(V+E)$

Edge-classification in DFS

P. 609

- Tree edge: explore on the path of DFS
- Forward edge: shortcut edge not on path
- Backward edge: edge to ancestor not on path
- Cross edge. connect edge to neither ancester nor descendent

(P. 609 good detail)

Topological Sort

- only works on acyclic graphs
- Sort vertices such that u is before v if edge from u to v exist

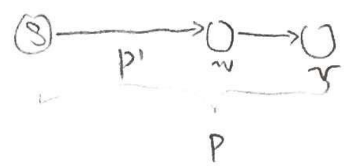
P. 613

complexity: $\Theta(V+E$

Shortest Path Algorithm

Bellman-ford Algorithm

- Artificially restrict # of edges in a path
- can be modify to return negative cycle
- No solution with negative cycle.



if P is shortest path from s to v
P' must be the shortest path to w

$$v.d_i = \min \begin{cases} v.d_{i-1} \\ \min\limits_{(u,v)\in E} \{u.d_{i-1} + w(u,v)\} \end{cases}$$

distance to v when i hops

Detecting negative cycles

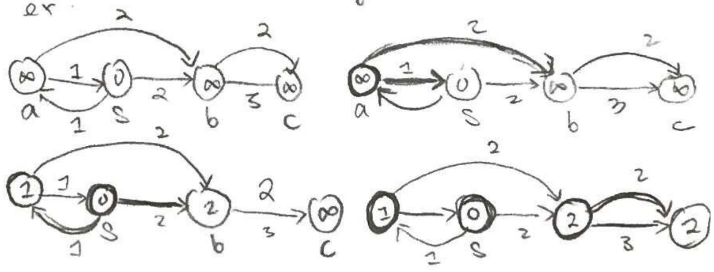
for each $(u,v) \in E$
 if $v.d < u.d + w(u,v)$
  return true //negative cycle

Complexity: $O(VE)$ P 651

DAG (directed acyclic graph) shortest path

- Topologically sort the graph
- Go in to topologically sorted order and upate the edges

ex.



DONE

Dijkstra's Algorithm

P. 662

- No negative edges
- main a visited unvisited set
- initiate all states en unvisited
- start by removing s

- when ever an edge u is removed update all neighbour v of u if $u.d > v.d + w(u,v)$

- pick u from unvisited set by finding $\min\{u.d\}$ for all u.

Complexity $O(E \lg V)$ regular
$O(V \lg V + E)$

Floyd-Warshall Algorithm

- If non-negative edge, just run dijkstra's for all pairs
- order vertices arbitrarily $V = \{1 \cdots n\}$
  $V^k = \{1 \cdots n\}$ prefix of first k vertices

$A[i,j,k]$ shortest path from i to j using first k vertices

$$A[i,j,0] = \begin{cases} 0, & i=j \\ w(i,j), & (i,j) \in E \\ +\infty & \text{if } (i \neq j) \wedge (i,j) \notin E \end{cases}$$

for k = 1 to n
  for i = 1 to n
    for j = 1 to n

$$A[i,j,k] = \min \begin{cases} A[i,j,k-1] \\ A[i,k,k-1] + A[k,j,k-1] \end{cases}$$

Complexity: $O(n^3)$ or $O(|V|^3)$

p. 698 transitive-closure

Minimum Spanning Tree

P. 631

- Find a tree that contain all the V with minimum total edge weight

Kruskal's algorithm

- make V into independent disjoint sets such that
  $V = \{v_1\} \cup \{v_2\} \cup \{v_3\} \cdots$
- Create a MST $A = \emptyset$
- sort the edges to ascending order
- go from lowest edge to highest (u, v). check if edge is safe by checking if $u.set = v.set$
- $A = A \cup \{(u,v)\}$ if safe and $u.set \cup v.set$

complexity: $O(E \lg V)$ p. 633

Prim's algorithm (similar to Dijkstra)

- select a random root r
- add r to a min-piriority-queue
- for every element removed, add all of its edges to queue (unexplored)
- extract the minimum from the queue
- terminate when all has been explored

p. 634

complexity: $O(E + V \lg V)$

NP-Completeness

P - Polynomial time

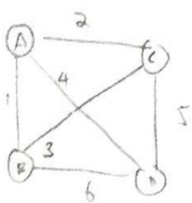
A problem that can be solved in Polynomial time $O(n^k)$ where k is some constant

NP-completeness

1. Decision Problem
2. Show it is NP
3. Reduce Π_1 to Π_2
   Showing $\Pi_1 \leq_p \Pi_2$
   Solving Π_2 solves Π_1

Traveling Salesmen Problem

- Find a cycle in a complete directed graph with no negative edges that minimize total edge cost



for example:
A B C D
$= 1 + 3 + 5 = 9$

How To proof NP-complete

- Show it is in NP with polynomial verification
- input, output YES & NO
- Reduce to a NP-complete

SAT

- Boolean expression written by $\wedge$ $\vee$ or $\neg$
- Return Yes if some boolean assignment cause it to be true

ex $X_1 \vee X_2 \wedge X_3 \vee X_4 \wedge \neg X_5$ ---

- 3SAT : SAT problem with 3 Literals
  $(x_1 \vee x_2 \vee x_3) \wedge (x_4 \vee x_5 \vee x_6)$ ---

CLIQUE

- input - G, k
- output YES or NO
- CLIQUE of size k is a set of k in G that are mutually connected
- Clique return yes if there is

NP-complete Problems: P. 1056