# Contents


**1 StringProcessing** 1
- 1.1 Z array.py . . . 1
- 1.2 suffix array.py . . . 1
- 1.3 KMP.py . . . 2

**2 Math** 3
- 2.1 Simplex.py . . . 3
- 2.2 ModularExponentiation.py . . . 4
- 2.3 BrentCycleDetection.py . . . 4
- 2.4 Factors.py . . . 4
- 2.5 NthPermutationRepetitions.py . . . 4
- 2.6 GaussJordan.py . . . 4
- 2.7 EulerFunction.py . . . 5
- 2.8 NumberTheory.py . . . 5
- 2.9 Sieve.py . . . 6
- 2.10 NthPermutation.py . . . 6

**3 Misc** 7
- 3.1 LongestIncreasingSubsequence.py . . . 7

**4 Graphs** 7
- 4.1 LowestCommonAncestor.py . . . 7
- 4.2 Kruskal.py . . . 8
- 4.3 Dijkstra.py . . . 8
- 4.4 MaxFlow.py . . . 8
- 4.5 MinCostMaxFlow.py . . . 9
- 4.6 ArticulationPoints.py . . . 9
- 4.7 MinCostBipartiteMatching.py . . . 10
- 4.8 SPFA.py . . . 11
- 4.9 MaxBipartiteMatching.py . . . 11
- 4.10 FloydWarshall.py . . . 12
- 4.11 StronglyConnectedComponents.py . . . 12

**5 DataStructures** 12
- 5.1 BinaryIndexedTree.py . . . 12
- 5.2 LinkedList.py . . . 13
- 5.3 UnionFindDisjointSet.py . . . 13
- 5.4 SparseTableRMQ.py . . . 14
- 5.5 SegmentTree.py . . . 14

**6 Geometry** 15
- 6.1 KDTree.py . . . 15
- 6.2 Convex Hull.py . . . 15
- 6.3 GeometryMisc.py . . . 16

**7 Tricks With Bits** 18

**8 Policy Based Data Structures (C++)** 18

**9 Quick runtime complexity reference** 19


# 1 StringProcessing

## 1.1 Z array.py

```python
# Fills Z array for given string str[]
def getZarr(string, z):
    n = len(string)

    # [L,R] make a window which matches
    # with prefix of s
    l, r, k = 0, 0, 0
    for i in range(1, n):

        # if i>R nothing matches so we will calculate.
        # Z[i] using naive way.
        if i > r:
            l, r = i, i

            # R-L = 0 in starting, so it will start
            # checking from 0'th index. For example,
            # for "ababab" and i = 1, the value of R
            # remains 0 and Z[i] becomes 0. For string
            # "aaaaaa" and i = 1, Z[i] and R become 5
            while r < n and string[r - l] == string[r]:
                r += 1
            z[i] = r - l
            r -= 1
        else:

            # k = i-L so k corresponds to number which
            # matches in [L,R] interval.
            k = i - l

            # if Z[k] is less than remaining interval
            # then Z[i] will be equal to Z[k].
            # For example, str = "ababab", i = 3, R = 5
            # and L = 2
            if z[k] < r - i + 1:
                z[i] = z[k]

            # For example str = "aaaaaa" and i = 2,
            # R is 5, L is 0
            else:

                # else start from R and check manually
                l = i
                while r < n and string[r - l] == string[r]:
                    r += 1
                z[i] = r - l
                r -= 1
```

```python

# prints all occurrences of pattern
# in text using Z algo
def search(text, pattern):

    # Create concatenated string "P$T"
    concat = pattern + "$" + text
    l = len(concat)

    # Construct Z array
    z = [0] * l
    getZarr(concat, z)

    # now looping through Z array for matching condition
    for i in range(l):

        # if Z[i] (matched region) is equal to pattern
        # length we got the pattern
        if z[i] == len(pattern):
            print("Pattern found at index",
                      i - len(pattern) - 1)
```

## 1.2 suffix array.py

```python
class suffix:

    def __init__(self):

        self.index = 0
        self.rank = [0, 0]

# This is the main function that takes a
# string 'txt' of size n as an argument,
# builds and return the suffix array for
# the given string
def buildSuffixArray(txt, n):

    # A structure to store suffixes
    # and their indexes
    suffixes = [suffix() for _ in range(n)]

    # Store suffixes and their indexes in
    # an array of structures. The structure
    # is needed to sort the suffixes alphabetically
    # and maintain their old indexes while sorting
    for i in range(n):
        suffixes[i].index = i
        suffixes[i].rank[0] = (ord(txt[i]) -
                               ord("a"))
        suffixes[i].rank[1] = (ord(txt[i + 1]) -
                        ord("a")) if ((i + 1) < n) else -1

    # Sort the suffixes according to the rank
    # and next rank
    suffixes = sorted(
        suffixes, key = lambda x: (
            x.rank[0], x.rank[1]))

    # At this point, all suffixes are sorted
    # according to first 2 characters.  Let
    # us sort suffixes according to first 4
    # characters, then first 8 and so on
    ind = [0] * n  # This array is needed to get the
                   # index in suffixes[] from original
                   # index.This mapping is needed to get
                   # next suffix.
    k = 4
    while (k < 2 * n):

        # Assigning rank and index
        # values to first suffix
        rank = 0
        prev_rank = suffixes[0].rank[0]
        suffixes[0].rank[0] = rank
        ind[suffixes[0].index] = 0

        # Assigning rank to suffixes
        for i in range(1, n):

            # If first rank and next ranks are
            # same as that of previous suffix in
            # array, assign the same new rank to
            # this suffix
            if (suffixes[i].rank[0] == prev_rank and
                suffixes[i].rank[1] == suffixes[i - 1].rank[1]):
                prev_rank = suffixes[i].rank[0]
                suffixes[i].rank[0] = rank

            # Otherwise increment rank and assign
            else:
                prev_rank = suffixes[i].rank[0]
                rank += 1
                suffixes[i].rank[0] = rank
            ind[suffixes[i].index] = i

        # Assign next rank to every suffix
        for i in range(n):
            nextindex = suffixes[i].index + k // 2
            suffixes[i].rank[1] = suffixes[ind[nextindex]].rank[0] \
                if (nextindex < n) else -1

        # Sort the suffixes according to
        # first k characters
        suffixes = sorted(
            suffixes, key = lambda x: (
                x.rank[0], x.rank[1]))

        k *= 2

    # Store indexes of all sorted
    # suffixes in the suffix array
    suffixArr = [0] * n

```

```
    for i in range(n):
        suffixArr[i] = suffixes[i].index

    # Return the suffix array
    return suffixArr

# A utility function to print an array
# of given size
def printArr(arr, n):

    for i in range(n):
        print(arr[i], end = " ")

    print()

# Driver code
if __name__ == "__main__":

    txt = "banana"
    n = len(txt)

    suffixArr = buildSuffixArray(txt, n)

    print("Following is suffix array for", txt)

    printArr(suffixArr, n)
```

## 1.3 KMP.py

```
#Buscar indice de un patron en un texto
def KMPSearch(pat, txt):
    M = len(pat)
    N = len(txt)
    lps = [0]*M
    j = 0
    computeLPSArray(pat, M, lps)
    i = 0
    while i < N:
        if pat[j] == txt[i]:
            i += 1
            j += 1
        if j == M:
            print ("Found pattern at index " + str(i-j))
            j = lps[j-1]
        elif i < N and pat[j] != txt[i]:
            if j != 0:
                j = lps[j-1]
            else:
                i += 1

def computeLPSArray(pat, M, lps):
    len = 0
    lps[0]
    i = 1
    while i < M:
        if pat[i]== pat[len]:
            len += 1
            lps[i] = len
            i += 1
        else:
            if len != 0:
                len = lps[len-1]
            else:
                lps[i] = 0
                i += 1
```

# 2 Math

## 2.1 Simplex.py

```
# Two-phase simplex algorithm for solving linear programs of the form
#     maximize     c^T x
#     subject to   Ax <= b
#                  x >= 0
# INPUT: A -- an m x n matrix
#        b -- an m-dimensional vector
#        c -- an n-dimensional vector
# OUTPUT: value of the optimal solution (infinity if unbounded
#         above, nan if infeasible), and a vector where the optimal
#         solution will be stored
# To use this code, create an LPSolver object with A, b, and c as
# arguments.  Then, call Solve().

EPS = 1e-9

class LPSolver:
    def __init__(self, a, b, c):
        self.m = len(b)
        self.n = len(c)
        self.N = [0 for _ in range(self.n + 1)]
        self.B = [0 for _ in range(self.m)]
        self.D = [[.0 for x in range(self.n + 2)] for y in range(self.m + 2)]
        for i in range(self.m):
            for j in range(self.n):
                self.D[i][j] = a[i][j]

        for i in range(self.m):
            self.B[i] = self.n + 1
            self.D[i][self.n] = -1
            self.D[i][self.n + 1] = b[i]

        for j in range(self.n):
            self.N[j] = j
            self.D[self.m][j] = -c[j]

        self.N[self.n], self.D[self.m + 1][self.n] = -1, 1

    def pivot(self, r, s):
        for i in range(self.m + 2):
            if i != r:
                for j in range(self.n + 2):
                    if j != s:
```

```python
                        self.D[i][j] -= self.D[r][j] * self.D[i][s] / self.D[r][s]

        for j in range(self.n + 2):
            if j != s:
                self.D[r][j] /= self.D[r][s]

        for i in range(self.m + 2):
            if i != r:
                self.D[i][s] /= -self.D[r][s]

        self.D[r][s] = 1.0 / self.D[r][s]
        self.B[r], self.N[s] = self.N[s], self.B[r]

    def simplex(self, phase):
        x = self.m + 1 if phase == 1 else self.m
        while True:
            s = -1
            for j in range(self.n + 1):
                if phase == 2 and self.N[j] == -1:
                    continue
                if s == -1 or self.D[x][j] < self.D[x][s] or self.D[x][j] == self.D[x][s] and self.N[j] < self.N[s]:
                    s = j

            if self.D[x][s] >= -EPS:
                return True

            r = -1
            for i in range(self.m):
                if self.D[i][s] <= 0:
                    continue
                if r == -1 or (self.D[i][self.n + 1] / self.D[i][s]) < (self.D[r][self.n + 1] / self.D[r][s]) or (self.D[i][self.n + 1] / self.D[i][s] == self.D[r][self.n + 1] / self.D[r][s]) and self.B[i] < self.B[r]:
                    r = i

            if r == -1:
                return False

            self.pivot(r, s)

    def solve(self):
        r = 0
        for i in range(1, self.m):
            if self.D[i][self.n + 1] < self.D[r][self.n + 1]:
                r = i
        if self.D[r][self.n + 1] <= -EPS:
            self.pivot(r, self.n)
            if not self.simplex(1) or self.D[self.m + 1][self.n + 1] < -EPS:
                return -float("inf")

            for i in range(self.m):
                if self.B[i] == -1:
                    s = -1
                    for j in range(self.n + 1):
                        if s == -1 or self.D[i][j] < self.D[i][s] or self.D[i][j] == self.D[i][s] and self.N[j] < self.N[s]:
                            s = j
                    self.pivot(i, s)

        if not self.simplex(2):
            return float("inf")

        x = [0. for _ in range(self.n)]
        for i in range(self.m):
            if self.B[i] < self.n:
                x[self.B[i]] = self.D[i][self.n + 1]

        return self.D[self.m][self.n + 1], x
```

## 2.2 ModularExponentiation.py

```python
# Complexity: O(log b)
# Returns (a**b)%c
def modular_pow(a, b, c):
    ans = 1
    while b > 0:
        if b % 2 == 1:
            ans = (ans * a) % c
        b = b >> 1
        a = (a * a) % c
    return ans
```

## 2.3 BrentCycleDetection.py

```python
#Find a cycle in a function f starting from a value x0
def findCycle(f, x0):
    # main phase: search successive powers of two
    power = lam = 1
    tortoise = x0
    hare = f(x0)  # f(x0) is the element/node next to x0.
    while tortoise != hare:
        if power == lam:  # time to start a new power of two?
            tortoise = hare
            power *= 2
            lam = 0
        hare = f(hare)
        lam += 1

    # Find the position of the first repetition of length
    tortoise = hare = x0
    for i in range(lam):
    # range(lam) produces a list with the values 0, 1, ... , lam-1
        hare = f(hare)
    # The distance between the hare and tortoise is now lambda.

    # Next, the hare and tortoise move at same speed until they agree
    mu = 0
    while tortoise != hare:
        tortoise = f(tortoise)
```

```
        hare = f(hare)
        mu += 1

    #lam = cycle length
    #mu = starting index of the detected cycle
    return lam, mu
```

## 2.4 Factors.py

```
# O(sqrt(n))
# Returns a list of all the factors of n
# Example: n = 12 -> result = [2, 2, 3]
# n > 1
def factors(n):
    z = 2
    results = []
    while(z*z<=n):
        if(n%z==0):
            results.append(int(z))
            n /= z
        else:
            z += 1
    if(n>1):
        results.append(int(n))
    return results
```

## 2.5 NthPermutationRepetitions.py

```
from math import factorial
from collections import defaultdict

def nthPermutationRepetitions(inp, n):
    mp = defaultdict(int)
    for i in range(len(inp)):
        mp[inp[i]] += 1

    buffer = ["" for _ in range(len(inp))]
    total = 0
    for i in range(len(inp)):
        for key in mp.keys():
            if mp[key] > 0:
                mp[key] -= 1
                perm = factorial(len(inp) - i - 1)

                for value in mp.values():
                    perm = perm // factorial(value)

                if n < total + perm:
                    buffer[i] = key
                    break

                total += perm
                mp[key] += 1

    return "".join(buffer)
```

## 2.6 GaussJordan.py

```
# Gauss-Jordan elimination with full pivoting.
# Uses:
#   (1) solving systems of linear equations (AX=B)
#   (2) inverting matrices (AX=I)
#   (3) computing determinants of square matrices
# Running time: O(n^3)
# INPUT:    a[][] = an nxn matrix
#           b[][] = an nxm matrix
# OUTPUT:   X      = an nxm matrix (stored in b[][])
#           A^{-1} = an nxn matrix (stored in a[][])
#           returns determinant of a[][]

import sys
EPS = 1e-10

def GaussJordan(a, b):
    n, m = len(a), len(b[0])
    irow, icol, ipiv = [0 for _ in range(n)], [0 for _ in range(n)], [0
        for _ in range(n)]
    det = 1.

    for i in range(n):
        pj, pk = -1, -1
        for j in range(n):
            if ipiv[j] == 0:
                for k in range(n):
                    if ipiv[k] == 0 and (pj == -1 or abs(a[j][k]) > abs(a
                        [pj][pk])):
                        pj, pk = j, k

        if abs(a[pj][pk]) < EPS:
            print("Matrix is singular.", file=sys.stderr)
            sys.exit(1)

        ipiv[pk] += 1
        a[pj], a[pk] = a[pk], a[pj]
        b[pj], b[pk] = b[pk], b[pj]

        if pj != pk:
            det *= -1

        irow[i], icol[i] = pj, pk

        c = 1.0 / a[pk][pk]
        det *= a[pk][pk]
        a[pk][pk] = 1.0

        for p in range(n):
            a[pk][p] *= c

        for p in range(m):
            b[pk][p] *= c

        for p in range(n):
            if p != pk:
                c = a[p][pk]
```

```python
                a[p][pk] = 0
                for q in range(n):
                    a[p][q] -= a[pk][q] * c

                for q in range(m):
                    b[p][q] -= b[pk][q] * c

    for p in reversed(range(n)):
        if irow[p] != icol[p]:
            for k in range(n):
                a[k][irow[p]], a[k][icol[p]] = a[k][icol[p]], a[k][irow[p]]

    return det
```

## 2.7 EulerFunction.py

```python
# Euler's Totient Function
# Amount of numbers 1..n that are relatively prime to n
# a^phi(N) = 1 (mod N) if gcd(a, N) = 1
# O(n^(1/2) log n)
def phi(n):
    # Initialize result as n
    result = n
    # Consider all prime factors
    # of n and subtract their
    # multiples from result
    p = 2
    while(p * p <= n):

        # Check if p is a
        # prime factor.
        if (n % p == 0):
            # If yes, then
            # update n and result
            while (n % p == 0):
                n = int(n / p);
            result -= int(result / p)
        p += 1;

    # If n has a prime factor
    # greater than sqrt(n)
    # (There can be at-most
    # one such prime factor)
    if (n > 1):
        result -= int(result / n)
    return result
```

## 2.8 NumberTheory.py

```python
# This is a collection of useful code for solving problems that
# involve modular linear equations.  Note that all of the
# algorithms described here work on nonnegative integers.

# return a % b (positive value)
def mod(a, b):
    return ((a % b) + b) % b

# computes gcd(a, b)
def gcd(a, b):
    while b != 0:
        a %= b
        a, b = b, a
    return a

# computes lcm(a,b)
def lcm(a, b):
    return a // gcd(a, b) * b

# returns (d, x, y) where d = gcd(a,b) && d = ax + by
def extended_euclid(a, b):
    xx = y = 0
    yy = x = 1
    while b != 0:
        q = a // b
        a, b = b, a % b
        x, xx = xx, x - q * xx
        y, yy = yy, y - q * yy

    return a, x, y

# finds all solutions to ax = b (mod n)
def modular_linear_equation_solver(a, b, n):
    solutions = []
    d, x, y = extended_euclid(a, n)
    if b % d == 0:
        x = mod(x * (b // d), n)
        for i in range(d):
            solutions.append(mod(x + i * (n // d), n))

    return solutions

# computes b such that ab = 1 (mod n), returns -1 on failure
def mod_inverse(a, n):
    d, x, y = extended_euclid(a, n)
    return -1 if d > 1 else mod(x, n)

# Chinese remainder theorem (special case): find z such that
# z % x = a, z % y = b.  Here, z is unique modulo M = lcm(x,y).
# Return (z,M).  On failure, M = -1.
def chinese_remainder_theorem(x, a, y, b):
    d, s, t = extended_euclid(x, y)
    if a % d != b % d:
        return 0, -1

    return (
        mod(s * b * x + t * a * y, x * y) // d,
        x * y // d
    )

# Chinese remainder theorem: find z such that
# z % x[i] = a[i] for all i.  Note that the solution is
# unique modulo M = lcm_i (x[i]).  Return (z,M).  On
# failure, M = -1.  Note that we do not require the a[i]'s
```

```
# to be relatively prime.
def chinese_remainder_theorem_system(x, a):
    ans = a[0], x[0]
    for i in range(1, len(x)):
        ans = chinese_remainder_theorem(ans[1], ans[0], x[i], a[i])
        if ans[1] == -1:
            break
    return ans

# computes x and y such that ax + by = c; on failure, x = y = -1
def linear_diophantine(a, b, c):
    d = gcd(a, b)
    if c % d != 0:
        return -1, -1
    else:
        x = c // d * mod_inverse(a // d, b // d)
        y = (c - a * x) // b
        return x, y
```

### 2.9 Sieve.py

```
from itertools import count

def postponed_sieve():
    yield 2; yield 3; yield 5; yield 7;
    sieve = {}
    ps = postponed_sieve()
    p = next(ps) and next(ps)
    q = p*p
    for c in count(9,2):
        if c in sieve:
            s = sieve.pop(c)
        elif c < q:
             yield c
             continue
        else:
            s=count(q+2*p,2*p)
            p=next(ps)
            q=p*p
        for m in s:
            if m not in sieve:
                break
        sieve[m] = s
```

### 2.10 NthPermutation.py

```
# e The number of entries
# n The index of the permutation
def nthPermutation(e, n):
    j, k = 0, 1
    fact = [0 for _ in range(e)]
    perm = [0 for _ in range(e)]
    fact[0] = 1

    # compute factorial numbers
    while k < e:
        fact[k] = fact[k - 1] * k
        k += 1

    # compute factorial code
    for k in range(e):
        perm[k] = n // fact[e - 1 - k]
        n = n % fact[e - 1 - k]

    # readjust values to obtain the permutation
    # start from the end and check if preceding values are lower
    for k in reversed(range(e)):
        for j in reversed(range(k)):
            if perm[j] <= perm[k]:
                perm[k] += 1

    # perm[0..e] contains the nth permutation
    return perm[:e]
```

## 3 Misc

### 3.1 LongestIncreasingSubsequence.py

```
# LIS
# in O(n Log n) time.
# Binary search
def GetCeilIndex(arr, T, l, r, key):
    while (r - l > 1):
        m = l + (r - l)//2
        if (arr[T[m]] >= key):
            r = m
        else:
            l = m
    return r

def LongestIncreasingSubsequence(arr, n):
    # Add boundary case,
    # when array n is zero
    # Depend on smart pointers
    # Initialized with 0
    tailIndices =[0 for i in range(n + 1)]
    # Initialized with -1
    prevIndices =[-1 for i in range(n + 1)]
    # it will always point
    # to empty location
    len = 1
    for i in range(1, n):
        if (arr[i] < arr[tailIndices[0]]):
            # new smallest value
            tailIndices[0] = i
        elif (arr[i] > arr[tailIndices[len-1]]):
            # arr[i] wants to extend
            # largest subsequence
            prevIndices[i] = tailIndices[len-1]
            tailIndices[len] = i
            len += 1
        else:
            # arr[i] wants to be a
            # potential condidate of
            # future subsequence
```

```python
            # It will replace ceil
            # value in tailIndices
            pos = GetCeilIndex(arr, tailIndices, -1,len-1, arr[i])
            prevIndices[i] = tailIndices[pos-1]
            tailIndices[pos] = i
    i = tailIndices[len-1]
    res = []
    while(i >= 0):
        res.append(arr[i])
        i = prevIndices[i]
    res.reverse()
    return len, res
```

# 4 Graphs

## 4.1 LowestCommonAncestor.py

```python
#USAGE
#Create LCA object for the given adjacency list
#Use find(u,v) to find LCA of nodes u and v
#REQUIRES RMQ IMPLEMENTATION
#Par[i] = parent of node i in the DFS, root is its own parent
#E[i] = i-th node visited in the DFS (Euler tour)
#L[i] = levels of the i-th node visited in the DFS (Euler tour)
#H[i] = index of the first occurrence of node i in E
class LCA:

    def __init__(self, V, AL):
        self.idx = 0
        self.AL = AL
        self.V = V
        self.Par = [-1 for _ in range(V)]
        self.E = [None for _ in range(2*V-1)]
        self.L = [None for _ in range(2*V-1)]
        self.H = [-1 for _ in range(V)]
        self.dfs(0,0,0)
        self.rmq = RMQ(self.L)

    def dfs(self,cur, depth, parent):
        self.Par[cur] = parent
        self.H[cur] = self.idx
        self.E[self.idx] = cur
        self.L[self.idx] = depth
        self.idx += 1
        for i in range(len(self.AL[cur])):
            if (self.Par[self.AL[cur][i]] == -1):
                self.dfs(self.AL[cur][i], depth + 1, cur)
                self.E[self.idx] = cur
                self.L[self.idx] = depth
                self.idx += 1

    def depth(self,u):
        return self.L[self.H[u]]

    def parent(self,u):
        return self.Par[u]

    def find(self,u,v):
        if (self.H[u] > self.H[v]):
            u,v = v,u
        return self.E[self.rmq.query(self.H[u], self.H[v])]

#INCREASE RECURSION
#sys.setrecursionlimit(100000)
```

## 4.2 Kruskal.py

```python
from heapq import heappush, heappop

#EL -> Edge list
# COMPLEXITY: O(E log E)
def kruskal(EL,V,E):
    EL.sort()                                  # sort by w, O(E log E)

    mst_cost = 0
    num_taken = 0
    UF = UnionFindDisjointSet(V)               # all V are disjoint sets

    for i in range(E):                         # for each edge, O(E)
        if num_taken == V-1:
            break
        w, u, v = EL[i]
        if (not UF.is_same_set(u, v)):          # check
            num_taken += 1                     # 1 more edge is taken
            mst_cost += w                      # add w of this edge
            UF.union_set(u, v)                  # link them
            # note: the runtime cost of UFDS is very light

    # note: the number of disjoint sets must eventually be 1 for a valid MST
    print("MST cost = {} (Kruskal's)".format(mst_cost))
```

## 4.3 Dijkstra.py

```python
from heapq import heappush, heappop

# AL -> Adjacency list
# COMPLEXITY: O((V+E)log V) (V,E < 300K)
def dijkstra(AL,V,s):
    INF = float("inf")

    # (Modified) Dijkstra's routine
    dist = [INF for u in range(V)]
    dist[s] = 0
    pq = []
    heappush(pq, (0, s))

    # sort the pairs by non-decreasing distance from s
    while (len(pq) > 0):                    # main loop
        d, u = heappop(pq)                  # shortest unvisited u
        if (d > dist[u]): continue          # a very important check
        for v, w in AL[u]:                  # all edges from u
```

```python
            if (dist[u]+w >= dist[v]): continue # not improving, skip
            dist[v] = dist[u]+w              # relax operation
            heappush(pq, (dist[v], v))

    for u in range(V):
        print("SSSP({}, {}) = {}".format(s, u, dist[u]))
```

## 4.4 MaxFlow.py

```python
from numbers import Number
from copy import deepcopy

INF = float('inf')

#USAGE
#Create MaxFlow(V) and add edges to graph using add_edge(u,v,w).
#Use dinic(s,t) to find max flow from s to t.
class MaxFlow:
    def __init__(self, V: int):
        self.V = V
        self.EL = []
        self.AL = [list() for _ in range(self.V)]
        self.d = []
        self.last = []
        self.p = []

    def BFS(self, s: int, t: int) -> bool:
        self.d = [-1] * self.V
        self.d[s] = 0
        self.p = [[-1, -1] for _ in range(self.V)]
        q = [s]
        while len(q) != 0:
            u = q[0]
            q.pop(0)
            if u == t:
                break
            for idx in self.AL[u]:
                v, cap, flow = self.EL[idx]
                if cap - flow > 0 and self.d[v] == -1:
                    self.d[v] = self.d[u]+1
                    q.append(v)
                    self.p[v] = [u, idx]
        return self.d[t] != -1

    def DFS(self, u: int, t: int, f: Number = INF) -> Number:
        if u == t or f == 0:
            return f
        for i in range(self.last[u], len(self.AL[u])):
            self.last[u] = i
            v, cap, flow = self.EL[self.AL[u][i]]
            if self.d[v] != self.d[u]+1:
                continue
            pushed = self.DFS(v, t, min(f, cap - flow))
            if pushed != 0:
                flow += pushed
                self.EL[self.AL[u][i]][2] = flow
                self.EL[self.AL[u][i] ^ 1][2] -= pushed
                return pushed
        return 0

    #Default directed edge
    def add_edge(self, u: int, v: int, capacity: Number,
                 directed: bool = True) -> None:
        if u == v:
            return
        self.EL.append([v, capacity, 0])
        self.AL[u].append(len(self.EL)-1)
        self.EL.append([u, 0 if directed else capacity, 0])
        self.AL[v].append(len(self.EL)-1)

    #Max flow
    def dinic(self, s: int, t: int) -> Number:

        mf = 0
        while self.BFS(s, t):
            self.last = [0] * self.V
            f = self.DFS(s, t)
            while f != 0:
                mf += f
                f = self.DFS(s, t)
        return mf

    #Copy the object (avoid destroying instance)
    def copy(self) -> 'MaxFlow':
        return deepcopy(self)

#INCREASE RECURSION
#sys.setrecursionlimit(100000)
```

## 4.5 MinCostMaxFlow.py

```python
INF = float("inf")

#USAGE
#Create MinCostMaxFlow(V) and add edges to graph using add_edge(u,v,w,c). (w capacity, c cost)
#Use mcmf(s,t) to find min cost max flow from s to t.
class MinCostMaxFlow:
  def __init__(self, V):
    self.V = V
    self.EL = []
    self.AL = [list() for _ in range(V)]
    self.vis = [False] * V
    self.total_cost = 0
    self.d = None
    self.last = None

  def SPFA(self, s, t):
    self.d = [INF] * self.V
    self.d[s] = 0
    self.vis[s] = True
    q = [s]
    while len(q) != 0:
      u = q[0]
```

```
      q.pop(0)
      self.vis[u] = False
      for idx in self.AL[u]:
        v, cap, flow, cost = self.EL[idx]
        if cap-flow > 0 and self.d[v] > self.d[u]+cost:
          self.d[v] = self.d[u]+cost
          if not self.vis[v]:
            q.append(v)
            self.vis[v] = True
    return self.d[t] != INF

  def DFS(self, u, t, f=INF):
    if u == t or f == 0:
      return f
    self.vis[u] = True
    for i in range(self.last[u], len(self.AL[u])):
      v, cap, flow, cost = self.EL[self.AL[u][i]]
      if not self.vis[v] and self.d[v] == self.d[u]+cost:
        pushed = self.DFS(v, t, min(f, cap-flow))
        if pushed != 0:
          self.total_cost += pushed * cost
          flow += pushed
          self.EL[self.AL[u][i]][2] = flow
          rv, rcap, rflow, rcost = self.EL[self.AL[u][i]^1]
          rflow -= pushed
          self.EL[self.AL[u][i]^1][2] = rflow
          self.vis[u] = False
          self.last[u] = i
          return pushed
    self.vis[u] = False
    return 0

  def add_edge(self, u, v, w, c, directed=True):
    if u == v:
      return
    self.EL.append([v, w, 0, c])
    self.AL[u].append(len(self.EL)-1)
    self.EL.append([u, 0 if directed else w, 0, -c])
    self.AL[v].append(len(self.EL)-1)

  def mcmf(self, s, t):
    mf = 0
    while self.SPFA(s, t):
      self.last = [0] * self.V
      f = self.DFS(s, t)
      while f != 0:
        mf += f
        f = self.DFS(s, t)
    return mf, self.total_cost

#INCREASE RECURSION
#sys.setrecursionlimit(100000)
```

## 4.6 ArticulationPoints.py

```
dfs_num = []
dfs_low = []
dfs_parent = []
articulation_vertex = []
dfsNumberCounter = 0
dfsRoot = 0
rootChildren = 0

# dfs_num[i] = orden en el que se visita por primera vez el nodo i
# dfs_low[i] = mínimo num alcanzable desde el nodo i y desde sus hijos en la búsqueda
# Establecer 'dfsRoot' al nodo raíz de la búsqueda, y 'rootChildren' y 'dfsNumberCounter' a 0 antes de llamar a
# articulationPointAndBridge(AL,root)
# Al -> Adjacency list
def articulationPointAndBridge(AL,u):
  global dfs_num, dfs_parent, dfs_low, articulation_vertex
  global dfsNumberCounter, dfsRoot, rootChildren

  dfs_low[u] = dfs_num[u] = dfsNumberCounter
  dfsNumberCounter += 1
  for (v, w) in AL[u]:
    if dfs_num[v] == -1:
      dfs_parent[v] = u
      if u == dfsRoot:
        rootChildren += 1

      articulationPointAndBridge(AL,v)

      if dfs_low[v] >= dfs_num[u]:
        articulation_vertex[u] = True
      if dfs_low[v] > dfs_num[u]:
        print(' Edge (%d, %d) is a bridge' % (u, v))
      dfs_low[u] = min(dfs_low[u], dfs_low[v])
    elif v != dfs_parent[u]:
      dfs_low[u] = min(dfs_low[u], dfs_num[v])

# Articulation -> Nodo que tras ser eliminado divide el componente conexo
# Bridge -> Arista entre u y v que tras ser eliminada hace que no haya camino entre u y v
def findArtBrid(AL, V):
  global dfs_num, dfs_parent, dfs_low, articulation_vertex
  global dfsNumberCounter, dfsRoot, rootChildren

  print('Articulation Points & Bridges (the input graph must be UNDIRECTED)')
  dfs_num = [-1] * V
  dfs_low = [0] * V
  dfs_parent = [-1] * V
  articulation_vertex = [False] * V
  dfsNumberCounter = 0
  print('Bridges:')
  for u in range(V):
    if dfs_num[u] == -1:
      dfsRoot = u
      rootChildren = 0
      articulationPointAndBridge(AL,u)
      articulation_vertex[dfsRoot] = (rootChildren > 1)
```

```python
  print('Articulation Points:')
  for u in range(V):
    if articulation_vertex[u]:
      print(' Vertex %d' % u)

#INCREASE RECURSION LIMIT!
#sys.setrecursionlimit(100000)
```

## 4.7 MinCostBipartiteMatching.py

```python
# Min cost bipartite matching via shortest augmenting paths
#
# This is an O(n^3) implementation of a shortest augmenting path
# algorithm for finding min cost perfect matchings in dense
# graphs. Note that both partitions must be of equal size!!
# (IF NOT EQUAL SIZE, ADD FAKE VERTICES AND EDGES THAT DON'T
# MODIFY THE ANSWER (0.0 costs))
#
#   cost[i][j] = cost for pairing left node i with right node j
#   Lmate[i] = index of right node that left node i pairs with
#   Rmate[j] = index of left node that right node j pairs with
#
# The values in cost[i][j] may be positive or negative.  To perform
# maximization, simply negate the cost[][] matrix.

def MinCostMatching(cost):
    n = len(cost);
    # construct dual feasible solution
    u = [None] * n
    v = [None] * n
    for i in range(n):
        u[i] = cost[i][0]
        for j in range(1,n):
            u[i] = min(u[i], cost[i][j])
    for j in range(n):
        v[j] = cost[0][j] - u[0]
        for i in range(1,n):
            v[j] = min(v[j], cost[i][j] - u[i])
    # construct primal solution satisfying complementary slackness
    Lmate = [-1]*n
    Rmate = [-1]*n
    mated = 0
    for i in range(n):
        for j in range(n):
            if (Rmate[j] != -1):
                continue
            if (abs(cost[i][j] - u[i] - v[j]) < 10**(-10)):
                Lmate[i] = j
                Rmate[j] = i
                mated += 1
                break

    dist = [None] * n
    # repeat until primal solution is feasible
    while (mated < n):
        # find an unmatched left node
        s = 0
        while (Lmate[s] != -1):
            s += 1
        # initialize Dijkstra

        dad = [-1] * n
        seen = [0] * n
        for k in range(n):
            dist[k] = cost[s][k] - u[s] - v[k]

        j = 0;
        while (True):
            # find closest
            j = -1;
            for k in range(n):
                if (seen[k]):
                    continue
                if (j == -1 or dist[k] < dist[j]):
                    j = k
            seen[j] = 1
            # termination condition
            if (Rmate[j] == -1):
                break
            # relax neighbors
            i = Rmate[j]
            for k in range(n):
                if (seen[k]):
                    continue
                new_dist = dist[j] + cost[i][k] - u[i] - v[k]
                if (dist[k] > new_dist):
                    dist[k] = new_dist
                    dad[k] = j
        # update dual variables
        for k in range(n):
            if (k == j or not seen[k]):
                continue
            i = Rmate[k]
            v[k] += dist[k] - dist[j]
            u[i] -= dist[k] - dist[j]
        u[s] += dist[j]
        # augment along path
        while (dad[j] >= 0):
            d = dad[j]
            Rmate[j] = Rmate[d]
            Lmate[Rmate[j]] = j
            j = d
        Rmate[j] = s
        Lmate[s] = j
        mated += 1
    value = 0
    for i in range(n):
        value += cost[i][Lmate[i]]
    return value
```

## 4.8 SPFA.py

```python
from collections import deque
```

```python

# Shortest Path Faster Algorithm
# SSSP adjacency-list implementation that handles negative weight cycles.
# The function returns true if such a cycle is detected (i.e., it can be reached from s).
# If not, dist[i] = distance from source node s to node i.
# Worst-case complexity: O(VE), in practice better than Bellman-Ford, but not than Dijkstra.
def SPFA(AL,V,s):
    INF = float("inf")

    # SPFA from source S
    # initially, only source vertex s has dist[s] = 0 and in the queue
    dist = [INF for u in range(V)]
    dist[s] = 0
    q = deque()
    q.append(s)
    in_queue = [0 for u in range(V)]
    veces = [0 for u in range(V)]
    in_queue[s] = 1
    veces[s] = 1
    while (len(q) > 0):
        u = q.popleft()                          # pop from queue
        in_queue[u] = 0
        for v, w in AL[u]:
            if (dist[u]+w >= dist[v]): continue  # not improving, skip
            dist[v] = dist[u]+w                  # relax operation
            if not in_queue[v]:                  # add to the queue
                q.append(v)                      # only if v is not
                in_queue[v] = 1                  # already in the queue
                veces[v] += 1
                #Negative cycle
                if(veces[v]==V):
                    return True

    for u in range(V):
        print("SSSP({}, {}) = {}".format(s, u, dist[u]))

    return False
```

## 4.9 MaxBipartiteMatching.py

```python
import random

match = []
vis = []

# Maximum Cardinality Bipartite Matching
# También es útil para:
#   Maximum Independent Set = |V| - MCBM
#   Minimum Vertex Cover = MCBM
def Aug(AL,L):
    global match, vis

    if vis[L]:
        return 0
    vis[L] = 1
    for R in AL[L]:
        if match[R] == -1 or Aug(AL,match[R]):
            match[R] = L
            return 1
    return 0

def matching(AL,V,Vleft):
    global match, vis

    freeV = set()
    for L in range(Vleft):
        freeV.add(L)
    match = [-1] * V
    MCBM = 0

    for L in range(Vleft):
        candidates = []
        for R in AL[L]:
            if match[R] == -1:
                candidates.append(R)
        if len(candidates) > 0:
            MCBM += 1
            freeV.remove(L)
            a = random.randrange(len(candidates))
            match[candidates[a]] = L

    for f in freeV:
        vis = [0] * Vleft
        MCBM += Aug(AL,f)

    print('Found %d matchings' % MCBM)
```

## 4.10 FloydWarshall.py

```python
# COMPLEXITY: O(V^3) (V < 400)
# adj_mat = matriz de adyacencia del grafo
# adj_mat[i][j] = INF si no hay arista
# adj_mat[i][i] = 0
# V = cantidad de nodos
# Si despues de todo la diagonal tiene un valor menor que cero, tiene ciclos negativos
def floyd_warshall(AM,V):
    for k in range(V):  # loop order is k->u->v
        for u in range(V):
            for v in range(V):
                AM[u][v] = min(AM[u][v], AM[u][k] + AM[k][v])

    for u in range(V):
        for v in range(V):
            print("APSP({}, {}) = {}".format(u, v, AM[u][v]))
```

## 4.11 StronglyConnectedComponents.py

```python
dfsNumberCounter = 0
numSCC = 0
dfs_num = []
dfs_low = []
S = []
```

```python
visited = []
st = []
nodesSCC = []

# dfs_num[i] = orden en el que se visita por primera vez el nodo i */
# dfs_low[i] = minimo num alcanzable desde el nodo i y desde sus hijos en la busqueda */
# st = Pila que guarda los nodos según el orden en que se exploran */
# Inicializar 'dfsNumberCounter' y 'numSCC' a 0 antes de llamar a la función */
# 'nodesSCC' guarda los componentes fuertemente conexos (SON DISJUNTOS)
def tarjanSCC(AL,u):
  global dfs_low, dfs_num, dfsNumberCounter, visited
  global numSCC, st, nodesSCC

  dfs_low[u] = dfs_num[u] = dfsNumberCounter
  dfsNumberCounter += 1
  st.append(u)
  visited[u] = True
  for v, w in AL[u]:
    if dfs_num[v] == -1:
      tarjanSCC(AL,v)
    if visited[v]:
      dfs_low[u] = min(dfs_low[u], dfs_low[v])

  if dfs_low[u] == dfs_num[u]:
    numSCC += 1
    while True:
      v = st[-1]
      st.pop()
      visited[v] = False
      nodesSCC[numSCC-1].append(v)
      if u == v:
        break

def SCC(AL,V):
  global dfs_low, dfs_num, dfsNumberCounter, visited
  global numSCC, st, nodesSCC
  dfs_num = [-1] * V
  dfs_low = [0] * V
  visited = [False] * V
  nodesSCC = [[] for _ in range(V)]
  st = []
  numSCC = 0
  dfsNumberCounter = 0
  for u in range(V):
      if dfs_num[u] == -1:
        tarjanSCC(AL,u)

#INCREASE RECURSION LIMIT!
#sys.setrecursionlimit(100000)
```

# 5 DataStructures

## 5.1 BinaryIndexedTree.py

```python
def LSOne(s):
    return s & -s


# Queries for dynamic RSQ in O(log n), elements numbered from 1 to n
class FenwickTree:
    def __init__(self, n):
        self.ft = [0 for _ in range(n + 1)]

    def get_sum(self, a):
        sum = 0
        while a > 0:
            sum += self.ft[a]
            a -= LSOne(a)
        return sum

    def get_range_sum(self, a, b):
        return self.get_sum(b) - (0 if a == 1 else self.get_sum(a - 1))

    def adjust(self, k, v):
        while k <= len(self.ft):
            self.ft[k] += v
            k += LSOne(k)
```

## 5.2 LinkedList.py

```python
class Node:
    def __init__(self, data=None):
        self.data = data
        self.prev = None
        self.next = None

    def append(self, x):
        node = x if type(x) is Node else Node(x)
        node.prev = self
        node.next = self.next
        if self.next is not None:
            self.next.prev = node
        self.next = node

    def prepend(self, x):
        node = x if type(x) is Node else Node(x)
        node.prev = self.prev
        node.next = self
        if self.prev is not None:
            self.prev.next = node
        self.prev = node

class LinkedList:
    def __init__(self, data=tuple()):
        self.head = None
        self.tail = None
        for d in data:
            self.append(d)

    def append(self, x):
        node = x if type(x) == Node else Node(x)
        if self.tail is None:
```

```python
            self.head = node
            self.tail = node
        else:
            self.tail.next = node
            node.prev = self.tail
            self.tail = node

    def prepend(self, x):
        node = x if type(x) == Node else Node(x)
        if self.head is None:
            self.head = node
            self.tail = node
        else:
            self.head.prev = node
            node.next = self.head
            self.head = node

    def insert_after(self, target, x):
        node = x if type(x) == Node else Node(x)
        target.append(x)
        if target == self.tail:
            self.tail = node

    def insert_before(self, target, x):
        node = x if type(x) == Node else Node(x)
        target.prepend(x)
        if target == self.head:
            self.head = node

    def search(self, x):
        pointer = self.head
        while pointer is not None:
            if pointer.data == x:
                break
            pointer = pointer.next
        return pointer

    def remove(self, x):
        node = x if type(x) == Node else self.search(x)
        if node is None:
            return
        if node.prev is not None:
            node.prev.next = node.next
        if node.next is not None:
            node.next.prev = node.prev
        if node == self.head:
            self.head = node.next
        if node == self.tail:
            self.tail = node.prev
```

## 5.3 UnionFindDisjointSet.py

```python
class UnionFindDisjointSet:
    def __init__(self, n: int):
        self.p = [-1 for _ in range(n)]
        self.set_size = [1 for _ in range(n)]
        self.n = n

    def find_set(self, i: int):
        if self.p[i] < 0:
            return i

        self.p[i] = self.find_set(self.p[i])
        return self.p[i]

    def is_same_set(self, i: int, j: int):
        return self.find_set(i) == self.find_set(j)

    def set_amount(self):
        return self.n

    def set_size(self, i: int):
        return self.set_size[self.find_set(i)]

    def union_set(self, i: int, j: int):
        if self.is_same_set(i, j):
            return

        self.n -= 1
        x, y = self.find_set(i), self.find_set(j)

        if self.p[x] < self.p[y]:  # rank[x] > rank[y]
            self.p[y] = x
            self.set_size[x] += self.set_size[y]
        else:
            self.p[x] = y
            self.set_size[y] += self.set_size[x]
            if self.p[x] == self.p[y]:
                self.p[y] -= 1
```

## 5.4 SparseTableRMQ.py

```python
class RMQ:
    def __init__(self, data):
        self.a = data
        self.log_table = [0 for _ in range(len(data) + 1)]

        for i in range(2,len(data)+1):
            self.log_table[i] = self.log_table[i >> 1] + 1

        self.rmq = [[0 for j in range(len(data))] for i in range(self.
            log_table[len(data)]+1)]
        for i in range(len(data)):
            self.rmq[0][i] = i

        k = 1
        while (1<<k) < len(data):
            i = 0
            while i+(1<<k) <= len(data):
                x = self.rmq[k - 1][i]
                y = self.rmq[k - 1][i+(1<<k-1)]
                self.rmq[k][i] = x if self.a[x] <= self.a[y] else y
                i += 1
            k += 1
```

```python

    def query(self, l, r):
        k = self.log_table[r - l + 1]
        x = self.rmq[k][l]
        y = self.rmq[k][r - (1 << k) + 1]
        return x if self.a[x] <= self.a[y] else y
```

### 5.5 SegmentTree.py

```python
import sys
import math
sys.setrecursionlimit(10**7)

class SegmentTree():
    DEFAULT = {
        'min': 1 << 60,
        'max': -(1 << 60),
        'sum': 0,
        'prd': 1,
        'gcd': 0,
        'lmc': 1,
        '^': 0,
        '&': (1 << 60) - 1,
        '|': 0,
    }

    FUNC = {
        'min': min,
        'max': max,
        'sum': (lambda x, y: x + y),
        'prd': (lambda x, y: x * y),
        'gcd': math.gcd,
        'lmc': (lambda x, y: (x * y) // math.gcd(x, y)),
        '^': (lambda x, y: x ^ y),
        '&': (lambda x, y: x & y),
        '|': (lambda x, y: x | y),
    }

    def __init__(self, N, ls, mode='min'):
        """
Number of leaves N,Element ls,Function mode(min,max,sum,prd(product),gcd,lmc,^,&,|)
        """
        self.default = self.DEFAULT[mode]
        self.func = self.FUNC[mode]
        self.N = N
        self.K = (N - 1).bit_length()
        self.N2 = 1 << self.K
        self.dat = [self.default] * (2**(self.K + 1))
        for i in range(self.N):  #Leaf construction
            self.dat[self.N2 + i] = ls[i]
        self.build()

    def build(self):
        for j in range(self.N2 - 1, -1, -1):
            self.dat[j] = self.func(self.dat[j << 1], self.dat[j << 1 | 1])  #Conditions that parents have

    def leafvalue(self, x):  #The xth value in the list
        return self.dat[x + self.N2]

    def update(self, x, y):  # index(x)To y
        i = x + self.N2
        self.dat[i] = y
        while i > 0:  #Change parent value
            i >>= 1
            self.dat[i] = self.func(self.dat[i << 1], self.dat[i << 1 | 1])
        return

    def query(self, L, R):  # [L,R)Section acquisition
        L += self.N2
        R += self.N2
        vL = self.default
        vR = self.default
        while L < R:
            if L & 1:
                vL = self.func(vL, self.dat[L])
                L += 1
            if R & 1:
                R -= 1
                vR = self.func(self.dat[R], vR)
            L >>= 1
            R >>= 1
        return self.func(vL, vR)
```

## 6 Geometry

### 6.1 KDTree.py

```python
class KDTree(object):
    """
    Usage:
    1. Make the KD-Tree:
        `kd_tree = KDTree(points, dim)`
    2. You can then use `get_knn` for k nearest neighbors or
       `get_nearest` for the nearest neighbor
    points are be a list of points: [[0, 1, 2], [12.3, 4.5, 2.3], ...]
    """
    def __init__(self, points, dim, dist_sq_func=None):
        if dist_sq_func is None:
            dist_sq_func = lambda a, b: sum((x - b[i]) ** 2
                for i, x in enumerate(a))

        def make(points, i=0):
            if len(points) > 1:
                points.sort(key=lambda x: x[i])
                i = (i + 1) % dim
                m = len(points) >> 1
                return [make(points[:m], i), make(points[m + 1:], i),
                    points[m]]
            if len(points) == 1:
                return [None, None, points[0]]

```

```
        def add_point(node, point, i=0):
            if node is not None:
                dx = node[2][i] - point[i]
                for j, c in ((0, dx >= 0), (1, dx < 0)):
                    if c and node[j] is None:
                        node[j] = [None, None, point]
                    elif c:
                        add_point(node[j], point, (i + 1) % dim)

        import heapq
        def get_knn(node, point, k, return_dist_sq, heap, i=0, tiebreaker=1):
            if node is not None:
                dist_sq = dist_sq_func(point, node[2])
                dx = node[2][i] - point[i]
                if len(heap) < k:
                    heapq.heappush(heap, (-dist_sq, tiebreaker, node[2]))
                elif dist_sq < -heap[0][0]:
                    heapq.heappushpop(heap, (-dist_sq, tiebreaker, node[2]))
                i = (i + 1) % dim
                # Goes into the left branch, then the right branch if needed
                for b in (dx < 0, dx >= 0)[:1 + (dx * dx < -heap[0][0])]:
                    get_knn(node[b], point, k, return_dist_sq,
                        heap, i, (tiebreaker << 1) | b)
            if tiebreaker == 1:
                return [(-h[0], h[2]) if return_dist_sq else h[2]
                    for h in sorted(heap)][::-1]

        def walk(node):
            if node is not None:
                for j in 0, 1:
                    for x in walk(node[j]):
                        yield x
                yield node[2]

        self._add_point = add_point
        self._get_knn = get_knn
        self._root = make(points)
        self._walk = walk

    def __iter__(self):
        return self._walk(self._root)

    def add_point(self, point):
        if self._root is None:
            self._root = [None, None, point]
        else:
            self._add_point(self._root, point)

    def get_knn(self, point, k, return_dist_sq=True):
        return self._get_knn(self._root, point, k, return_dist_sq, [])

    def get_nearest(self, point, return_dist_sq=True):
        l = self._get_knn(self._root, point, 1, return_dist_sq, [])
            return l[0] if len(l) else None
```

## 6.2 Convex Hull.py

```
class Point:
    def __init__(self, x, y):
        self.x = x
        self.y = y

def Left_index(points):

    '''
    Finding the left most point
    '''
    minn = 0
    for i in range(1,len(points)):
        if points[i].x < points[minn].x:
            minn = i
        elif points[i].x == points[minn].x:
            if points[i].y > points[minn].y:
                minn = i
    return minn

def orientation(p, q, r):
    '''
    To find orientation of ordered triplet (p, q, r).
    The function returns following values
    0 --> p, q and r are collinear
    1 --> Clockwise
    2 --> Counterclockwise
    '''
    val = (q.y - p.y) * (r.x - q.x) - \
          (q.x - p.x) * (r.y - q.y)

    if val == 0:
        return 0
    elif val > 0:
        return 1
    else:
        return 2

def convexHull(points, n):

    # There must be at least 3 points
    if n < 3:
        return

    # Find the leftmost point
    l = Left_index(points)

    hull = []

    '''
    Start from leftmost point, keep moving counterclockwise
    until reach the start point again. This loop runs O(h)
    times where h is number of points in result or output.
    '''
```

```
    p = l
    q = 0
    while(True):

        # Add current point to result
        hull.append(p)

        '''
        Search for a point 'q' such that orientation(p, q,
        x) is counterclockwise for all points 'x'. The idea
        is to keep track of last visited most counterclock-
        wise point in q. If any point 'i' is more counterclock-
        wise than q, then update q.
        '''
        q = (p + 1) % n

        for i in range(n):

            # If i is more counterclockwise
            # than current q, then update q
            if(orientation(points[p],
                           points[i], points[q]) == 2):
                q = i

        '''
        Now q is the most counterclockwise with respect to p
        Set p as q for next iteration, so that q is added to
        result 'hull'
        '''
        p = q

        # While we don't come to first point
        if(p == l):
            break

    # Print Result
    for each in hull:
        print(points[each].x, points[each].y)
```

### 6.3 GeometryMisc.py

```
import math

EPS = 1e-12

class Point:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def __add__(self, obj):
        return Point(self.x + obj.x, self.y + obj.y)

    def __sub__(self, obj):
        return Point(self.x - obj.x, self.y - obj.y)

    def __mul__(self, c):
        return Point(self.x * c, self.y * c)

    def __div__(self, c):
        return Point(self.x / c, self.y / c)

    def __str__(self):
        return f"({self.x},{self.y})"

def dot(p, q):
    return p.x * q.x + p.y * q.y

def dist2(p, q):
    return dot(p-q, p-q)

def cross(p, q):
    return p.x * q.y - p.y * q.x

# Rotate a point CCW or CC around the origin
def rotateCCW90(p):
    return Point(-p.y, p.x)

def rotateCW90(p):
    return Point(p.y, -p.x)

def rotateCCW(p, t):
    return Point(p.x * math.cos(t) - p.y * math.sin(t), p.x * math.sin(t)
         + p.y * math.cos(t))

# project point c onto line through a and b
# assuming a != b
def project_point_line(a, b, c):
    return a + (b - a) * dot(c - a, b - a) / dot(b - a, b - a)

# project point c onto line segment through a and b
def project_point_segment(a, b, c):
    r = dot(b - a, b - a)
    if abs(r) < EPS:
        return a

    r = dot(c - a, b - a) / r
    if r < 0:
        return a
    if r > 1:
        return b

    return a + (b - a) * r

# compute distance from c to segment between a and b
def distance_point_segment(a, b, c):
    return math.sqrt(dist2(c, project_point_segment(a, b, c)))

# compute distance between point (x,y,z) and plane ax+by+cz=d
def distance_point_plane(x, y, z, a, b, c, d):
    return abs(a * x + b * y + c * z - d) / math.sqrt(a * a + b * b + c *
         c)

```

```python
# determine if lines from a to b and c to d are parallel or collinear
def lines_parallel(a, b, c, d):
    return abs(cross(b - a, c - d)) < EPS

def lines_collinear(a, b, c, d):
    return lines_parallel(a, b, c, d) and abs(cross(a - b, a - c)) < EPS
        and abs(cross(c - d, c - a)) < EPS

# determine if line segment from a to b intersects with line segment from
    c to d
def segment_intersect(a, b, c, d):
    if lines_collinear(a, b, c, d):
        if dist2(a, c) < EPS or dist2(a, d) < EPS or dist2(b, c) < EPS or
             dist2(b, d) < EPS:
            return True
        if dot(c - a, c - b) > 0 and dot(d - a, d - b) > 0 and dot(c - b,
             d - b) > 0:
            return False
        return True

    if cross(d - a, b - a) * cross(c - a, b - a) > 0:
        return False
    if cross(a - c, d - c) * cross(b - c, d - c) > 0:
        return False
    return True

# compute intersection of line passing through a and b
# with line passing through c and d, assuming that unique
# intersection exists; for segment intersection, check if
# segments intersect first
def compute_line_intersection(a, b, c, d):
    b, d, c = b - a, c - d, c - a
    assert dot(b, b) > EPS and dot(d, d) > EPS
    return a + b * cross(c, d) / cross(b, d)

# compute center of circle given three points
def compute_circle_center(a, b, c):
    b, c = (a + b) / 2, (a + c) / 2
    return compute_line_intersection(b, b + rotateCW90(a - b), c, c +
        rotateCW90(a - c))

# determine if point q is in a possibly non-convex polygon p (by William
# Randolph Franklin); returns 1 for strictly interior points, 0 for
# strictly exterior points, and 0 or 1 for the remaining points.
# Note that it is possible to convert this into an *exact* test using
# integer arithmetic by taking care of the division appropriately
# (making sure to deal with signs properly) and then by writing exact
# tests for checking point on polygon boundary
def point_in_polygon(p, q):
    c = False
    for i in range(len(p)):
        j = (i + 1) % len(p)
        if (p[i].y <= q.y and q.y < p[j].y or p[j].y <= q.y and q.y < p[i
            ].y) and q.x < p[i].x + (p[j].x - p[i].x) * (q.y - p[i].y) /
            (p[j].y - p[i].y):
            c = not c
    return c

# determine if point is on the boundary of a polygon
def point_on_polygon(p, q):
    for i in range(len(p)):
        if dist2(project_point_segment(p[i], p[(i + 1) % len(p)], q), q)
            < EPS:
            return True
    return False

# compute intersection of line through points a and b with
# circle centered at c with radius r > 0
def circle_line_intersection(a, b, c, r):
    ans = []
    a, b = a - c, b - a
    A, B = dot(b, b), dot(a, b)
    C, D = dot(a, a) - r * r, B * B - A * C
    if D < -EPS:
        return ans
    ans.append(c + a + b * (-B + math.sqrt(D + EPS)) / A)
    if D > EPS:
        ans.append(c + a + b * (-B - math.sqrt(D)) / A)
    return ans

# compute intersection of circle centered at a with radius r
# with circle centered at b with radius R
def circle_circle_intersection(a, b, r, R):
    ans = []
    d = math.sqrt(dist2(a, b))
    if d > r + R or d + min(r, R) < max (r, R):
        return ans

    x = (d * d - R * R + r * r) / (2 * d)
    y = math.sqrt(r * r - x * x)
    v = (b - a) / d
    ans.append(a + v * x + rotateCCW90(v) * y)
    if y > 0:
        ans.append(a + v * x - rotateCCW90(v) * y)

    return ans

# This code computes the area or centroid of a (possibly nonconvex)
# polygon, assuming that the coordinates are listed in a clockwise or
# counterclockwise fashion.  Note that the centroid is often known as
# the "center of gravity" or "center of mass".
def compute_signed_area(p):
    area = 0
    for i in range(len(p)):
        j = (i + 1) % len(p)
        area += p[i].x * p[j].y - p[j].x * p[i].y

    return area / 2.

def compute_area(p):
    return abs(compute_signed_area(p))
```

```python
def compute_centroid(p):
    c = Point(0, 0)
    scale = 6.0 * compute_signed_area(p)
    for i in range(len(p)):
        j = (i + 1) % len(p)
        c = c + (p[i] + p[j]) * (p[i].x * p[j].y - p[j].x * p[i].y)

    return c / scale

# tests whether or not a given polygon (in CW or CCW order) is simple
def is_simple(p):
    for i in range(len(p)):
        for k in range(i + 1, len(p)):
            j = (i + 1) % len(p)
            l = (k + 1) % len(p)
            if i == l or j == k:
                continue
            if segment_intersect(p[i], p[j], p[k], p[l]):
                return False

    return True
```

# 7 Tricks With Bits

In python3, ~x (flip all bits in other languages) is achieved with
(~x & 0xFFFFFFFF) (use repit1 lenght of HEXA as you wish)
x & (x-1)
clear the lowest set bit of x
x & ~(x-1)
extracts the lowest set bit of x (all others are clear).
Pretty patterns when applied to a linear sequence.
x & (x + (1 << n))
the run of set bits (possibly length 0) starting at bit n cleared.
x & ~(x + (1 << n))
the run of set bits (possibly length 0) in x, starting at bit n.
x | (x + 1)
x with the lowest cleared bit set.
x | ~(x + 1)
Extracts the lowest cleared bit of x (all others are set),
if ~ wrapping the expression, you have that cleared value.
x | (x - (1 << n))
x With the run of cleared bits (possibly length 0) starting at bit n set.
x | ~(x - (1 << n))
The lowest run of cleared bits (possibly length 0) in x,
starting at bit n are the only clear bits.
By 'run' is intended the number formed by all consecutive
1's at the left of n-th bit, starting at n-th bit.

# 8 Policy Based Data Structures (C++)

```cpp
#include <bits/stdc++.h>
using namespace std;

#include <bits/extc++.h>                              // pbds
using namespace __gnu_pbds;
typedef tree<int, null_type, less<int>, rb_tree_tag,
             tree_order_statistics_node_update> ost;

// Custom comparator function
template <class T>
struct comp_fx
{
    bool operator()(const T &a, const T &b)
    {

        return a < b;
    }
};

int main() {
  int n = 9;
  int A[] = { 2, 4, 7,10,15,23,50,65,71};            // as in Chapter 2
  ost tree;
  for (int i = 0; i < n; ++i)                          // O(n log n)
    tree.insert(A[i]);
  // O(log n) select
  cout << *tree.find_by_order(0) << "\n";              // 1-smallest = 2
  cout << *tree.find_by_order(n-1) << "\n";            // 9-smallest/largest =
      71
  cout << *tree.find_by_order(4) << "\n";              // 5-smallest = 15
  // O(log n) rank
  cout << tree.order_of_key(2) << "\n";                // index 0 (rank 1)
  cout << tree.order_of_key(71) << "\n";               // index 8 (rank 9)
  cout << tree.order_of_key(15) << "\n";               // index 4 (rank 5)
  return 0;
}
```

# 9 Quick runtime complexity reference

| $n$ | Worst AC |
|---|---|
| $\le [10..11]$ | $O(n!), O(n^6)$ |
| $\le [15..18]$ | $O(2^n \times n^2)$ |
| $\le [18..22]$ | $O(2^n \times n)$ |
| $\le [24..26]$ | $O(2^n)$ |
| $\le 100$ | $O(n^4)$ |
| $\le 450$ | $O(n^3)$ |
| $\le 1.5K$ | $O(n^{2.5})$ |

| $n$ | Worst AC |
|---|---|
| $\le 2.5K$ | $O(n^2 \log n)$ |
| $\le 10K$ | $O(n^2)$ |
| $\le 200K$ | $O(n^{1.5})$ |
| $\le 4.5M$ | $O(n \log n)$ |
| $\le 10M$ | $O(n \log \log n)$ |
| $\le 100M$ | $O(n), O(\log n), O(1)$ |

## Theoretical Computer Science Cheat Sheet

### Definitions

| | |
|---|---|
| $f(n) = O(g(n))$ | iff $\exists$ positive $c, n_0$ such that $0 \le f(n) \le cg(n)$ $\forall n \ge n_0$. |
| $f(n) = \Omega(g(n))$ | iff $\exists$ positive $c, n_0$ such that $f(n) \ge cg(n) \ge 0$ $\forall n \ge n_0$. |
| $f(n) = \Theta(g(n))$ | iff $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$. |
| $f(n) = o(g(n))$ | iff $\lim_{n\to\infty} f(n)/g(n) = 0$. |
| $\lim\limits_{n\to\infty} a_n = a$ | iff $\forall \epsilon > 0$, $\exists n_0$ such that $\lvert a_n - a\rvert < \epsilon$, $\forall n \ge n_0$. |
| $\sup S$ | least $b \in \mathbb{R}$ such that $b \ge s$, $\forall s \in S$. |
| $\inf S$ | greatest $b \in \mathbb{R}$ such that $b \le s$, $\forall s \in S$. |
| $\liminf\limits_{n\to\infty} a_n$ | $\lim\limits_{n\to\infty} \inf\{a_i \mid i \ge n, i \in \mathbb{N}\}$. |
| $\limsup\limits_{n\to\infty} a_n$ | $\lim\limits_{n\to\infty} \sup\{a_i \mid i \ge n, i \in \mathbb{N}\}$. |
| $\binom{n}{k}$ | Combinations: Size $k$ subsets of a size $n$ set. |
| $\left[{n \atop k}\right]$ | Stirling numbers (1st kind): Arrangements of an $n$ element set into $k$ cycles. |
| $\left\{{n \atop k}\right\}$ | Stirling numbers (2nd kind): Partitions of an $n$ element set into $k$ non-empty sets. |
| $\left\langle{n \atop k}\right\rangle$ | 1st order Eulerian numbers: Permutations $\pi_1\pi_2\dots\pi_n$ on $\{1, 2, \dots, n\}$ with $k$ ascents. |
| $\left\langle\!\left\langle{n \atop k}\right\rangle\!\right\rangle$ | 2nd order Eulerian numbers. |
| $C_n$ | Catalan Numbers: Binary trees with $n+1$ vertices. |

### Series

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}, \qquad \sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}, \qquad \sum_{i=1}^{n} i^3 = \frac{n^2(n+1)^2}{4}.$$

In general:

$$\sum_{i=1}^{n} i^m = \frac{1}{m+1}\left[(n+1)^{m+1} - 1 - \sum_{i=1}^{n}\left((i+1)^{m+1} - i^{m+1} - (m+1)i^m\right)\right]$$

$$\sum_{i=1}^{n-1} i^m = \frac{1}{m+1}\sum_{k=0}^{m}\binom{m+1}{k} B_k n^{m+1-k}.$$

Geometric series:

$$\sum_{i=0}^{n} c^i = \frac{c^{n+1}-1}{c-1}, \quad c \ne 1, \qquad \sum_{i=0}^{\infty} c^i = \frac{1}{1-c}, \qquad \sum_{i=1}^{\infty} c^i = \frac{c}{1-c}, \quad \lvert c\rvert < 1,$$

$$\sum_{i=0}^{n} ic^i = \frac{nc^{n+2} - (n+1)c^{n+1} + c}{(c-1)^2}, \quad c \ne 1, \qquad \sum_{i=0}^{\infty} ic^i = \frac{c}{(1-c)^2}, \quad \lvert c\rvert < 1.$$

Harmonic series:

$$H_n = \sum_{i=1}^{n} \frac{1}{i}, \qquad \sum_{i=1}^{n} iH_i = \frac{n(n+1)}{2}H_n - \frac{n(n-1)}{4}.$$

$$\sum_{i=1}^{n} H_i = (n+1)H_n - n, \qquad \sum_{i=1}^{n}\binom{i}{m}H_i = \binom{n+1}{m+1}\left(H_{n+1} - \frac{1}{m+1}\right).$$

**1.** $\displaystyle\binom{n}{k} = \frac{n!}{(n-k)!k!}$, **2.** $\displaystyle\sum_{k=0}^{n}\binom{n}{k} = 2^n$, **3.** $\displaystyle\binom{n}{k} = \binom{n}{n-k}$,

**4.** $\displaystyle\binom{n}{k} = \frac{n}{k}\binom{n-1}{k-1}$, **5.** $\displaystyle\binom{n}{k} = \binom{n-1}{k} + \binom{n-1}{k-1}$,

**6.** $\displaystyle\binom{n}{m}\binom{m}{k} = \binom{n}{k}\binom{n-k}{m-k}$, **7.** $\displaystyle\sum_{k=0}^{n}\binom{r+k}{k} = \binom{r+n+1}{n}$,

**8.** $\displaystyle\sum_{k=0}^{n}\binom{k}{m} = \binom{n+1}{m+1}$, **9.** $\displaystyle\sum_{k=0}^{n}\binom{r}{k}\binom{s}{n-k} = \binom{r+s}{n}$,

**10.** $\displaystyle\binom{n}{k} = (-1)^k\binom{k-n-1}{k}$, **11.** $\displaystyle\left\{{n \atop 1}\right\} = \left\{{n \atop n}\right\} = 1$,

**12.** $\displaystyle\left\{{n \atop 2}\right\} = 2^{n-1} - 1$, **13.** $\displaystyle\left\{{n \atop k}\right\} = k\left\{{n-1 \atop k}\right\} + \left\{{n-1 \atop k-1}\right\}$,

**14.** $\displaystyle\left[{n \atop 1}\right] = (n-1)!$, **15.** $\displaystyle\left[{n \atop 2}\right] = (n-1)!H_{n-1}$, **16.** $\displaystyle\left[{n \atop n}\right] = 1$, **17.** $\displaystyle\left[{n \atop k}\right] \ge \left\{{n \atop k}\right\}$,

**18.** $\displaystyle\left[{n \atop k}\right] = (n-1)\left[{n-1 \atop k}\right] + \left[{n-1 \atop k-1}\right]$, **19.** $\displaystyle\left\{{n \atop n-1}\right\} = \left[{n \atop n-1}\right] = \binom{n}{2}$, **20.** $\displaystyle\sum_{k=0}^{n}\left[{n \atop k}\right] = n!$, **21.** $\displaystyle C_n = \frac{1}{n+1}\binom{2n}{n}$,

**22.** $\displaystyle\left\langle{n \atop 0}\right\rangle = \left\langle{n \atop n-1}\right\rangle = 1$, **23.** $\displaystyle\left\langle{n \atop k}\right\rangle = \left\langle{n \atop n-1-k}\right\rangle$, **24.** $\displaystyle\left\langle{n \atop k}\right\rangle = (k+1)\left\langle{n-1 \atop k}\right\rangle + (n-k)\left\langle{n-1 \atop k-1}\right\rangle$,

**25.** $\displaystyle\left\langle{0 \atop k}\right\rangle = \begin{cases} 1 & \text{if } k = 0, \\ 0 & \text{otherwise} \end{cases}$ **26.** $\displaystyle\left\langle{n \atop 1}\right\rangle = 2^n - n - 1$, **27.** $\displaystyle\left\langle{n \atop 2}\right\rangle = 3^n - (n+1)2^n + \binom{n+1}{2}$,

**28.** $\displaystyle x^n = \sum_{k=0}^{n}\left\langle{n \atop k}\right\rangle\binom{x+k}{n}$, **29.** $\displaystyle\left\langle{n \atop m}\right\rangle = \sum_{k=0}^{m}\binom{n+1}{k}(m+1-k)^n(-1)^k$, **30.** $\displaystyle m!\left\{{n \atop m}\right\} = \sum_{k=0}^{n}\left\langle{n \atop k}\right\rangle\binom{k}{n-m}$,

**31.** $\displaystyle\left\langle{n \atop m}\right\rangle = \sum_{k=0}^{n}\left\{{n \atop k}\right\}\binom{n-k}{m}(-1)^{n-k-m}k!$, **32.** $\displaystyle\left\langle\!\left\langle{n \atop 0}\right\rangle\!\right\rangle = 1$, **33.** $\displaystyle\left\langle\!\left\langle{n \atop n}\right\rangle\!\right\rangle = 0 \quad \text{for } n \ne 0$,

**34.** $\displaystyle\left\langle\!\left\langle{n \atop k}\right\rangle\!\right\rangle = (k+1)\left\langle\!\left\langle{n-1 \atop k}\right\rangle\!\right\rangle + (2n-1-k)\left\langle\!\left\langle{n-1 \atop k-1}\right\rangle\!\right\rangle$, **35.** $\displaystyle\sum_{k=0}^{n}\left\langle\!\left\langle{n \atop k}\right\rangle\!\right\rangle = \frac{(2n)^{\underline{n}}}{2^n}$,

**36.** $\displaystyle\left\{{x \atop x-n}\right\} = \sum_{k=0}^{n}\left\langle\!\left\langle{n \atop k}\right\rangle\!\right\rangle\binom{x+n-1-k}{2n}$, **37.** $\displaystyle\left\{{n+1 \atop m+1}\right\} = \sum_{k}\binom{n}{k}\left\{{k \atop m}\right\} = \sum_{k=0}^{n}\left\{{k \atop m}\right\}(m+1)^{n-k}$,

**Theoretical Computer Science Cheat Sheet**

Identities Cont.

**38.** $\left[{n+1 \atop m+1}\right] = \sum_k \left[{n \atop k}\right]\binom{k}{m} = \sum_{k=0}^{n} \left[{k \atop m}\right] n^{\underline{n-k}} = n! \sum_{k=0}^{n} \frac{1}{k!}\left[{k \atop m}\right],$

**39.** $\left[{x \atop x-n}\right] = \sum_{k=0}^{n} \left\langle\!\!\left\langle {n \atop k} \right\rangle\!\!\right\rangle \binom{x+k}{2n},$

**40.** $\left\{{n \atop m}\right\} = \sum_k \binom{n}{k}\left\{{k+1 \atop m+1}\right\}(-1)^{n-k},$

**41.** $\left[{n \atop m}\right] = \sum_k \left[{n+1 \atop k+1}\right]\binom{k}{m}(-1)^{m-k},$

**42.** $\left\{{m+n+1 \atop m}\right\} = \sum_{k=0}^{m} k\left\{{n+k \atop k}\right\},$

**43.** $\left[{m+n+1 \atop m}\right] = \sum_{k=0}^{m} k(n+k)\left[{n+k \atop k}\right],$

**44.** $\binom{n}{m} = \sum_k \left\{{n+1 \atop k+1}\right\}\left[{k \atop m}\right](-1)^{m-k},$

**45.** $(n-m)!\binom{n}{m} = \sum_k \left[{n+1 \atop k+1}\right]\left\{{k \atop m}\right\}(-1)^{m-k}, \quad \text{for } n \geq m,$

**46.** $\left\{{n \atop n-m}\right\} = \sum_k \binom{m-n}{m+k}\binom{m+n}{n+k}\left[{m+k \atop k}\right],$

**47.** $\left[{n \atop n-m}\right] = \sum_k \binom{m-n}{m+k}\binom{m+n}{n+k}\left\{{m+k \atop k}\right\},$

**48.** $\left\{{n \atop \ell+m}\right\}\binom{\ell+m}{\ell} = \sum_k \left\{{k \atop \ell}\right\}\left\{{n-k \atop m}\right\}\binom{n}{k},$

**49.** $\left[{n \atop \ell+m}\right]\binom{\ell+m}{\ell} = \sum_k \left[{k \atop \ell}\right]\left[{n-k \atop m}\right]\binom{n}{k}.$

Trees

Every tree with $n$ vertices has $n-1$ edges.

Kraft inequality: If the depths of the leaves of a binary tree are $d_1, \ldots, d_n$:
$$\sum_{i=1}^{n} 2^{-d_i} \leq 1,$$
and equality holds only if every internal node has 2 sons.

Recurrences

Master method:
$$T(n) = aT(n/b) + f(n), \quad a \geq 1, b > 1$$

If $\exists \epsilon > 0$ such that $f(n) = O(n^{\log_b a - \epsilon})$ then
$$T(n) = \Theta(n^{\log_b a}).$$

If $f(n) = \Theta(n^{\log_b a})$ then
$$T(n) = \Theta(n^{\log_b a} \log_2 n).$$

If $\exists \epsilon > 0$ such that $f(n) = \Omega(n^{\log_b a + \epsilon})$, and $\exists c < 1$ such that $af(n/b) \leq cf(n)$ for large $n$, then
$$T(n) = \Theta(f(n)).$$

Substitution (example): Consider the following recurrence
$$T_{i+1} = 2^{2^i} \cdot T_i^2, \quad T_1 = 2.$$

Note that $T_i$ is always a power of two. Let $t_i = \log_2 T_i$. Then we have
$$t_{i+1} = 2^i + 2t_i, \quad t_1 = 1.$$

Let $u_i = t_i/2^i$. Dividing both sides of the previous equation by $2^{i+1}$ we get
$$\frac{t_{i+1}}{2^{i+1}} = \frac{2^i}{2^{i+1}} + \frac{t_i}{2^i}.$$

Substituting we find
$$u_{i+1} = \tfrac{1}{2} + u_i, \qquad u_1 = \tfrac{1}{2},$$

which is simply $u_i = i/2$. So we find that $T_i$ has the closed form $T_i = 2^{i2^{i-1}}$.
Summing factors (example): Consider the following recurrence
$$T(n) = 3T(n/2) + n, \quad T(1) = 1.$$

Rewrite so that all terms involving $T$ are on the left side
$$T(n) - 3T(n/2) = n.$$

Now expand the recurrence, and choose a factor which makes the left side "telescope"

$$\begin{gathered} 1\big(T(n) - 3T(n/2) = n\big) \\ 3\big(T(n/2) - 3T(n/4) = n/2\big) \\ \vdots \quad \vdots \quad \vdots \\ 3^{\log_2 n - 1}\big(T(2) - 3T(1) = 2\big) \end{gathered}$$

Let $m = \log_2 n$. Summing the left side we get $T(n) - 3^m T(1) = T(n) - 3^m = T(n) - n^k$ where $k = \log_2 3 \approx 1.58496$. Summing the right side we get
$$\sum_{i=0}^{m-1} \frac{n}{2^i} 3^i = n \sum_{i=0}^{m-1} \left(\tfrac{3}{2}\right)^i.$$

Let $c = \frac{3}{2}$. Then we have
$$\begin{aligned} n \sum_{i=0}^{m-1} c^i &= n\left(\frac{c^m - 1}{c - 1}\right) \\ &= 2n(c^{\log_2 n} - 1) \\ &= 2n(c^{(k-1)\log_c n} - 1) \\ &= 2n^k - 2n, \end{aligned}$$

and so $T(n) = 3n^k - 2n$. Full history recurrences can often be changed to limited history ones (example): Consider
$$T_i = 1 + \sum_{j=0}^{i-1} T_j, \quad T_0 = 1.$$

Note that
$$T_{i+1} = 1 + \sum_{j=0}^{i} T_j.$$

Subtracting we find
$$\begin{aligned} T_{i+1} - T_i &= 1 + \sum_{j=0}^{i} T_j - 1 - \sum_{j=0}^{i-1} T_j \\ &= T_i. \end{aligned}$$

And so $T_{i+1} = 2T_i = 2^{i+1}$.

Generating functions:

1. Multiply both sides of the equation by $x^i$.
2. Sum both sides over all $i$ for which the equation is valid.
3. Choose a generating function $G(x)$. Usually $G(x) = \sum_{i=0}^{\infty} x^i g_i$.
3. Rewrite the equation in terms of the generating function $G(x)$.
4. Solve for $G(x)$.
5. The coefficient of $x^i$ in $G(x)$ is $g_i$.

Example:
$$g_{i+1} = 2g_i + 1, \quad g_0 = 0.$$

Multiply and sum:
$$\sum_{i \geq 0} g_{i+1} x^i = \sum_{i \geq 0} 2g_i x^i + \sum_{i \geq 0} x^i.$$

We choose $G(x) = \sum_{i \geq 0} x^i g_i$. Rewrite in terms of $G(x)$:
$$\frac{G(x) - g_0}{x} = 2G(x) + \sum_{i \geq 0} x^i.$$

Simplify:
$$\frac{G(x)}{x} = 2G(x) + \frac{1}{1-x}.$$

Solve for $G(x)$:
$$G(x) = \frac{x}{(1-x)(1-2x)}.$$

Expand this using partial fractions:
$$\begin{aligned} G(x) &= x\left(\frac{2}{1-2x} - \frac{1}{1-x}\right) \\ &= x\left(2\sum_{i \geq 0} 2^i x^i - \sum_{i \geq 0} x^i\right) \\ &= \sum_{i \geq 0} (2^{i+1} - 1)x^{i+1}. \end{aligned}$$

So $g_i = 2^i - 1$.

## Theoretical Computer Science Cheat Sheet

$\pi \approx 3.14159$, $e \approx 2.71828$, $\gamma \approx 0.57721$, $\phi = \frac{1+\sqrt{5}}{2} \approx 1.61803$, $\hat{\phi} = \frac{1-\sqrt{5}}{2} \approx -.61803$

| $i$ | $2^i$ | $p_i$ |
|---|---|---|
| 1 | 2 | 2 |
| 2 | 4 | 3 |
| 3 | 8 | 5 |
| 4 | 16 | 7 |
| 5 | 32 | 11 |
| 6 | 64 | 13 |
| 7 | 128 | 17 |
| 8 | 256 | 19 |
| 9 | 512 | 23 |
| 10 | 1,024 | 29 |
| 11 | 2,048 | 31 |
| 12 | 4,096 | 37 |
| 13 | 8,192 | 41 |
| 14 | 16,384 | 43 |
| 15 | 32,768 | 47 |
| 16 | 65,536 | 53 |
| 17 | 131,072 | 59 |
| 18 | 262,144 | 61 |
| 19 | 524,288 | 67 |
| 20 | 1,048,576 | 71 |
| 21 | 2,097,152 | 73 |
| 22 | 4,194,304 | 79 |
| 23 | 8,388,608 | 83 |
| 24 | 16,777,216 | 89 |
| 25 | 33,554,432 | 97 |
| 26 | 67,108,864 | 101 |
| 27 | 134,217,728 | 103 |
| 28 | 268,435,456 | 107 |
| 29 | 536,870,912 | 109 |
| 30 | 1,073,741,824 | 113 |
| 31 | 2,147,483,648 | 127 |
| 32 | 4,294,967,296 | 131 |

### Pascal's Triangle

1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
1 5 10 10 5 1
1 6 15 20 15 6 1
1 7 21 35 35 21 7 1
1 8 28 56 70 56 28 8 1
1 9 36 84 126 126 84 36 9 1
1 10 45 120 210 252 210 120 45 10 1

### General

Bernoulli Numbers ($B_i = 0$, odd $i \neq 1$):

$$B_0 = 1,\ B_1 = -\tfrac{1}{2},\ B_2 = \tfrac{1}{6},\ B_4 = -\tfrac{1}{30},$$
$$B_6 = \tfrac{1}{42},\ B_8 = -\tfrac{1}{30},\ B_{10} = \tfrac{5}{66}.$$

Change of base, quadratic formula:

$$\log_b x = \frac{\log_a x}{\log_a b}, \qquad \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}.$$

Euler's number $e$:

$$e = 1 + \tfrac{1}{2} + \tfrac{1}{6} + \tfrac{1}{24} + \tfrac{1}{120} + \cdots$$
$$\lim_{n \to \infty} \left(1 + \frac{x}{n}\right)^n = e^x.$$
$$\left(1 + \tfrac{1}{n}\right)^n < e < \left(1 + \tfrac{1}{n}\right)^{n+1}.$$
$$\left(1 + \tfrac{1}{n}\right)^n = e - \frac{e}{2n} + \frac{11e}{24n^2} - O\left(\frac{1}{n^3}\right).$$

Harmonic numbers:

$$1,\ \tfrac{3}{2},\ \tfrac{11}{6},\ \tfrac{25}{12},\ \tfrac{137}{60},\ \tfrac{49}{20},\ \tfrac{363}{140},\ \tfrac{761}{280},\ \tfrac{7129}{2520},\ \ldots$$

$$\ln n < H_n < \ln n + 1,$$
$$H_n = \ln n + \gamma + O\left(\frac{1}{n}\right).$$

Factorial, Stirling's approximation:

$$1,\ 2,\ 6,\ 24,\ 120,\ 720,\ 5040,\ 40320,\ 362880,\ \ldots$$

$$n! = \sqrt{2\pi n}\left(\frac{n}{e}\right)^n \left(1 + \Theta\left(\frac{1}{n}\right)\right).$$

Ackermann's function and inverse:

$$a(i,j) = \begin{cases} 2^j & i = 1 \\ a(i-1, 2) & j = 1 \\ a(i-1, a(i, j-1)) & i, j \geq 2 \end{cases}$$
$$\alpha(i) = \min\{j \mid a(j,j) \geq i\}.$$

Binomial distribution:

$$\Pr[X = k] = \binom{n}{k} p^k q^{n-k}, \qquad q = 1 - p,$$
$$\mathrm{E}[X] = \sum_{k=1}^{n} k \binom{n}{k} p^k q^{n-k} = np.$$

Poisson distribution:

$$\Pr[X = k] = \frac{e^{-\lambda}\lambda^k}{k!}, \quad \mathrm{E}[X] = \lambda.$$

Normal (Gaussian) distribution:

$$p(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-(x-\mu)^2/2\sigma^2}, \quad \mathrm{E}[X] = \mu.$$

The "coupon collector": We are given a random coupon each day, and there are $n$ different types of coupons. The distribution of coupons is uniform. The expected number of days to pass before we to collect all $n$ types is

$$nH_n.$$

### Probability

Continuous distributions: If

$$\Pr[a < X < b] = \int_a^b p(x)\,dx,$$

then $p$ is the probability density function of $X$. If

$$\Pr[X < a] = P(a),$$

then $P$ is the distribution function of $X$. If $P$ and $p$ both exist then

$$P(a) = \int_{-\infty}^{a} p(x)\,dx.$$

Expectation: If $X$ is discrete

$$\mathrm{E}[g(X)] = \sum_x g(x) \Pr[X = x].$$

If $X$ continuous then

$$\mathrm{E}[g(X)] = \int_{-\infty}^{\infty} g(x)p(x)\,dx = \int_{-\infty}^{\infty} g(x)\,dP(x).$$

Variance, standard deviation:

$$\mathrm{VAR}[X] = \mathrm{E}[X^2] - \mathrm{E}[X]^2,$$
$$\sigma = \sqrt{\mathrm{VAR}[X]}.$$

For events $A$ and $B$:

$$\Pr[A \vee B] = \Pr[A] + \Pr[B] - \Pr[A \wedge B]$$
$$\Pr[A \wedge B] = \Pr[A] \cdot \Pr[B],$$

iff $A$ and $B$ are independent.

$$\Pr[A|B] = \frac{\Pr[A \wedge B]}{\Pr[B]}$$

For random variables $X$ and $Y$:

$$\mathrm{E}[X \cdot Y] = \mathrm{E}[X] \cdot \mathrm{E}[Y],$$

if $X$ and $Y$ are independent.

$$\mathrm{E}[X + Y] = \mathrm{E}[X] + \mathrm{E}[Y],$$
$$\mathrm{E}[cX] = c\,\mathrm{E}[X].$$

Bayes' theorem:

$$\Pr[A_i|B] = \frac{\Pr[B|A_i]\Pr[A_i]}{\sum_{j=1}^{n} \Pr[A_j]\Pr[B|A_j]}.$$

Inclusion-exclusion:

$$\Pr\left[\bigvee_{i=1}^{n} X_i\right] = \sum_{i=1}^{n} \Pr[X_i] +$$
$$\sum_{k=2}^{n} (-1)^{k+1} \sum_{i_i < \cdots < i_k} \Pr\left[\bigwedge_{j=1}^{k} X_{i_j}\right].$$

Moment inequalities:

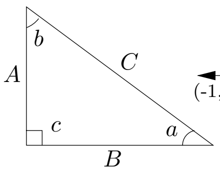
$$\Pr\left[|X| \geq \lambda\,\mathrm{E}[X]\right] \leq \frac{1}{\lambda},$$
$$\Pr\left[\left|X - \mathrm{E}[X]\right| \geq \lambda \cdot \sigma\right] \leq \frac{1}{\lambda^2}.$$

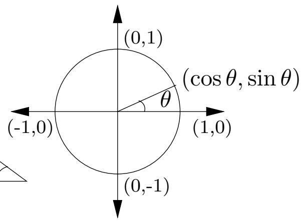
Geometric distribution:

$$\Pr[X = k] = pq^{k-1}, \qquad q = 1 - p,$$
$$\mathrm{E}[X] = \sum_{k=1}^{\infty} kpq^{k-1} = \frac{1}{p}.$$

# Theoretical Computer Science Cheat Sheet

## Trigonometry




Pythagorean theorem:
$$C^2 = A^2 + B^2.$$

Definitions:
$$\sin a = A/C, \quad \cos a = B/C,$$
$$\csc a = C/A, \quad \sec a = C/B,$$
$$\tan a = \frac{\sin a}{\cos a} = \frac{A}{B}, \quad \cot a = \frac{\cos a}{\sin a} = \frac{B}{A}.$$

Area, radius of inscribed circle:
$$\tfrac{1}{2}AB, \quad \frac{AB}{A+B+C}.$$

Identities:
$$\sin x = \frac{1}{\csc x}, \qquad \cos x = \frac{1}{\sec x},$$
$$\tan x = \frac{1}{\cot x}, \qquad \sin^2 x + \cos^2 x = 1,$$
$$1 + \tan^2 x = \sec^2 x, \qquad 1 + \cot^2 x = \csc^2 x,$$
$$\sin x = \cos\left(\tfrac{\pi}{2} - x\right), \qquad \sin x = \sin(\pi - x),$$
$$\cos x = -\cos(\pi - x), \qquad \tan x = \cot\left(\tfrac{\pi}{2} - x\right),$$
$$\cot x = -\cot(\pi - x), \qquad \csc x = \cot\tfrac{x}{2} - \cot x,$$
$$\sin(x \pm y) = \sin x \cos y \pm \cos x \sin y,$$
$$\cos(x \pm y) = \cos x \cos y \mp \sin x \sin y,$$
$$\tan(x \pm y) = \frac{\tan x \pm \tan y}{1 \mp \tan x \tan y},$$
$$\cot(x \pm y) = \frac{\cot x \cot y \mp 1}{\cot x \pm \cot y},$$
$$\sin 2x = 2 \sin x \cos x, \qquad \sin 2x = \frac{2\tan x}{1 + \tan^2 x},$$
$$\cos 2x = \cos^2 x - \sin^2 x, \qquad \cos 2x = 2\cos^2 x - 1,$$
$$\cos 2x = 1 - 2\sin^2 x, \qquad \cos 2x = \frac{1 - \tan^2 x}{1 + \tan^2 x},$$
$$\tan 2x = \frac{2\tan x}{1 - \tan^2 x}, \qquad \cot 2x = \frac{\cot^2 x - 1}{2\cot x},$$
$$\sin(x+y)\sin(x-y) = \sin^2 x - \sin^2 y,$$
$$\cos(x+y)\cos(x-y) = \cos^2 x - \sin^2 y.$$

Euler's equation:
$$e^{ix} = \cos x + i \sin x, \qquad e^{i\pi} = -1.$$

## Matrices

Multiplication:
$$C = A \cdot B, \quad c_{i,j} = \sum_{k=1}^{n} a_{i,k} b_{k,j}.$$

Determinants: $\det A \neq 0$ iff $A$ is non-singular.
$$\det A \cdot B = \det A \cdot \det B,$$
$$\det A = \sum_{\pi} \prod_{i=1}^{n} \operatorname{sign}(\pi) a_{i,\pi(i)}.$$

$2 \times 2$ and $3 \times 3$ determinant:
$$\begin{vmatrix} a & b \\ c & d \end{vmatrix} = ad - bc,$$
$$\begin{vmatrix} a & b & c \\ d & e & f \\ g & h & i \end{vmatrix} = g \begin{vmatrix} b & c \\ e & f \end{vmatrix} - h \begin{vmatrix} a & c \\ d & f \end{vmatrix} + i \begin{vmatrix} a & b \\ d & e \end{vmatrix}$$
$$= \begin{matrix} aei + bfg + cdh \\ - ceg - fha - ibd. \end{matrix}$$

Permanents:
$$\operatorname{perm} A = \sum_{\pi} \prod_{i=1}^{n} a_{i,\pi(i)}.$$

## Hyperbolic Functions

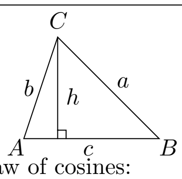
Definitions:
$$\sinh x = \frac{e^x - e^{-x}}{2}, \qquad \cosh x = \frac{e^x + e^{-x}}{2},$$
$$\tanh x = \frac{e^x - e^{-x}}{e^x + e^{-x}}, \qquad \operatorname{csch} x = \frac{1}{\sinh x},$$
$$\operatorname{sech} x = \frac{1}{\cosh x}, \qquad \coth x = \frac{1}{\tanh x}.$$

Identities:
$$\cosh^2 x - \sinh^2 x = 1, \qquad \tanh^2 x + \operatorname{sech}^2 x = 1,$$
$$\coth^2 x - \operatorname{csch}^2 x = 1, \qquad \sinh(-x) = -\sinh x,$$
$$\cosh(-x) = \cosh x, \qquad \tanh(-x) = -\tanh x,$$
$$\sinh(x+y) = \sinh x \cosh y + \cosh x \sinh y,$$
$$\cosh(x+y) = \cosh x \cosh y + \sinh x \sinh y,$$
$$\sinh 2x = 2 \sinh x \cosh x,$$
$$\cosh 2x = \cosh^2 x + \sinh^2 x,$$
$$\cosh x + \sinh x = e^x, \qquad \cosh x - \sinh x = e^{-x},$$
$$(\cosh x + \sinh x)^n = \cosh nx + \sinh nx, \quad n \in \mathbb{Z},$$
$$2\sinh^2 \tfrac{x}{2} = \cosh x - 1, \qquad 2\cosh^2 \tfrac{x}{2} = \cosh x + 1.$$

| $\theta$ | $\sin\theta$ | $\cos\theta$ | $\tan\theta$ |
|---|---|---|---|
| 0 | 0 | 1 | 0 |
| $\frac{\pi}{6}$ | $\frac{1}{2}$ | $\frac{\sqrt{3}}{2}$ | $\frac{\sqrt{3}}{3}$ |
| $\frac{\pi}{4}$ | $\frac{\sqrt{2}}{2}$ | $\frac{\sqrt{2}}{2}$ | 1 |
| $\frac{\pi}{3}$ | $\frac{\sqrt{3}}{2}$ | $\frac{1}{2}$ | $\sqrt{3}$ |
| $\frac{\pi}{2}$ | 1 | 0 | $\infty$ |

... in mathematics you don't understand things, you just get used to them.
– J. von Neumann

## More Trig.




Law of cosines:
$$c^2 = a^2 + b^2 - 2ab\cos C.$$

Area:
$$A = \tfrac{1}{2}hc,$$
$$= \tfrac{1}{2}ab\sin C,$$
$$= \frac{c^2 \sin A \sin B}{2 \sin C}.$$

Heron's formula:
$$A = \sqrt{s \cdot s_a \cdot s_b \cdot s_c},$$
$$s = \tfrac{1}{2}(a+b+c),$$
$$s_a = s - a,$$
$$s_b = s - b,$$
$$s_c = s - c.$$

More identities:
$$\sin\tfrac{x}{2} = \sqrt{\frac{1 - \cos x}{2}},$$
$$\cos\tfrac{x}{2} = \sqrt{\frac{1 + \cos x}{2}},$$
$$\tan\tfrac{x}{2} = \sqrt{\frac{1 - \cos x}{1 + \cos x}},$$
$$= \frac{1 - \cos x}{\sin x},$$
$$= \frac{\sin x}{1 + \cos x},$$
$$\cot\tfrac{x}{2} = \sqrt{\frac{1 + \cos x}{1 - \cos x}},$$
$$= \frac{1 + \cos x}{\sin x},$$
$$= \frac{\sin x}{1 - \cos x},$$
$$\sin x = \frac{e^{ix} - e^{-ix}}{2i},$$
$$\cos x = \frac{e^{ix} + e^{-ix}}{2},$$
$$\tan x = -i\frac{e^{ix} - e^{-ix}}{e^{ix} + e^{-ix}},$$
$$= -i\frac{e^{2ix} - 1}{e^{2ix} + 1},$$
$$\sin x = \frac{\sinh ix}{i},$$
$$\cos x = \cosh ix,$$
$$\tan x = \frac{\tanh ix}{i}.$$

v2.02 ©1994 by Steve Seiden
sseiden@acm.org
http://www.csc.lsu.edu/~seiden

**Theoretical Computer Science Cheat Sheet**

## Number Theory

The Chinese remainder theorem: There exists a number $C$ such that:

$$C \equiv r_1 \bmod m_1$$
$$\vdots \quad \vdots \quad \vdots$$
$$C \equiv r_n \bmod m_n$$

if $m_i$ and $m_j$ are relatively prime for $i \neq j$.

Euler's function: $\phi(x)$ is the number of positive integers less than $x$ relatively prime to $x$. If $\prod_{i=1}^n p_i^{e_i}$ is the prime factorization of $x$ then

$$\phi(x) = \prod_{i=1}^n p_i^{e_i - 1}(p_i - 1).$$

Euler's theorem: If $a$ and $b$ are relatively prime then

$$1 \equiv a^{\phi(b)} \bmod b.$$

Fermat's theorem:

$$1 \equiv a^{p-1} \bmod p.$$

The Euclidean algorithm: if $a > b$ are integers then

$$\gcd(a, b) = \gcd(a \bmod b, b).$$

If $\prod_{i=1}^n p_i^{e_i}$ is the prime factorization of $x$ then

$$S(x) = \sum_{d|x} d = \prod_{i=1}^n \frac{p_i^{e_i+1} - 1}{p_i - 1}.$$

Perfect Numbers: $x$ is an even perfect number iff $x = 2^{n-1}(2^n - 1)$ and $2^n - 1$ is prime.

Wilson's theorem: $n$ is a prime iff

$$(n-1)! \equiv -1 \bmod n.$$

Möbius inversion:

$$\mu(i) = \begin{cases} 1 & \text{if } i = 1. \\ 0 & \text{if } i \text{ is not square-free.} \\ (-1)^r & \text{if } i \text{ is the product of} \\ & r \text{ distinct primes.} \end{cases}$$

If

$$G(a) = \sum_{d|a} F(d),$$

then

$$F(a) = \sum_{d|a} \mu(d) G\left(\frac{a}{d}\right).$$

Prime numbers:

$$p_n = n \ln n + n \ln \ln n - n + n \frac{\ln \ln n}{\ln n} + O\left(\frac{n}{\ln n}\right),$$

$$\pi(n) = \frac{n}{\ln n} + \frac{n}{(\ln n)^2} + \frac{2!n}{(\ln n)^3} + O\left(\frac{n}{(\ln n)^4}\right).$$

## Graph Theory

Definitions:

| | |
|---|---|
| *Loop* | An edge connecting a vertex to itself. |
| *Directed* | Each edge has a direction. |
| *Simple* | Graph with no loops or multi-edges. |
| *Walk* | A sequence $v_0 e_1 v_1 \ldots e_\ell v_\ell$. |
| *Trail* | A walk with distinct edges. |
| *Path* | A trail with distinct vertices. |
| *Connected* | A graph where there exists a path between any two vertices. |
| *Component* | A maximal connected subgraph. |
| *Tree* | A connected acyclic graph. |
| *Free tree* | A tree with no root. |
| *DAG* | Directed acyclic graph. |
| *Eulerian* | Graph with a trail visiting each edge exactly once. |
| *Hamiltonian* | Graph with a cycle visiting each vertex exactly once. |
| *Cut* | A set of edges whose removal increases the number of components. |
| *Cut-set* | A minimal cut. |
| *Cut edge* | A size 1 cut. |
| *k-Connected* | A graph connected with the removal of any $k-1$ vertices. |
| *k-Tough* | $\forall S \subseteq V, S \neq \emptyset$ we have $k \cdot c(G - S) \leq \lvert S \rvert$. |
| *k-Regular* | A graph where all vertices have degree $k$. |
| *k-Factor* | A $k$-regular spanning subgraph. |
| *Matching* | A set of edges, no two of which are adjacent. |
| *Clique* | A set of vertices, all of which are adjacent. |
| *Ind. set* | A set of vertices, none of which are adjacent. |
| *Vertex cover* | A set of vertices which cover all edges. |
| *Planar graph* | A graph which can be embeded in the plane. |
| *Plane graph* | An embedding of a planar graph. |

$$\sum_{v \in V} \deg(v) = 2m.$$

If $G$ is planar then $n - m + f = 2$, so

$$f \leq 2n - 4, \quad m \leq 3n - 6.$$

Any planar graph has a vertex with degree $\leq 5$.

Notation:

| | |
|---|---|
| $E(G)$ | Edge set |
| $V(G)$ | Vertex set |
| $c(G)$ | Number of components |
| $G[S]$ | Induced subgraph |
| $\deg(v)$ | Degree of $v$ |
| $\Delta(G)$ | Maximum degree |
| $\delta(G)$ | Minimum degree |
| $\chi(G)$ | Chromatic number |
| $\chi_E(G)$ | Edge chromatic number |
| $G^c$ | Complement graph |
| $K_n$ | Complete graph |
| $K_{n_1,n_2}$ | Complete bipartite graph |
| $\mathrm{r}(k, \ell)$ | Ramsey number |

## Geometry

Projective coordinates: triples $(x, y, z)$, not all $x$, $y$ and $z$ zero.

$$(x, y, z) = (cx, cy, cz) \quad \forall c \neq 0.$$

| Cartesian | Projective |
|---|---|
| $(x, y)$ | $(x, y, 1)$ |
| $y = mx + b$ | $(m, -1, b)$ |
| $x = c$ | $(1, 0, -c)$ |

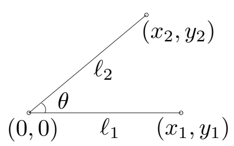
Distance formula, $L_p$ and $L_\infty$ metric:

$$\sqrt{(x_1 - x_0)^2 + (y_1 - y_0)^2},$$
$$\left[\lvert x_1 - x_0 \rvert^p + \lvert y_1 - y_0 \rvert^p\right]^{1/p},$$
$$\lim_{p \to \infty} \left[\lvert x_1 - x_0 \rvert^p + \lvert y_1 - y_0 \rvert^p\right]^{1/p}.$$

Area of triangle $(x_0, y_0)$, $(x_1, y_1)$ and $(x_2, y_2)$:

$$\tfrac{1}{2} \operatorname{abs} \begin{vmatrix} x_1 - x_0 & y_1 - y_0 \\ x_2 - x_0 & y_2 - y_0 \end{vmatrix}.$$

Angle formed by three points:




$$\cos\theta = \frac{(x_1, y_1) \cdot (x_2, y_2)}{\ell_1 \ell_2}.$$

Line through two points $(x_0, y_0)$ and $(x_1, y_1)$:

$$\begin{vmatrix} x & y & 1 \\ x_0 & y_0 & 1 \\ x_1 & y_1 & 1 \end{vmatrix} = 0.$$

Area of circle, volume of sphere:

$$A = \pi r^2, \qquad V = \tfrac{4}{3}\pi r^3.$$

If I have seen farther than others, it is because I have stood on the shoulders of giants.
– Issac Newton

**Theoretical Computer Science Cheat Sheet**

Series

Taylor's series:

$$f(x) = f(a) + (x-a)f'(a) + \frac{(x-a)^2}{2}f''(a) + \cdots = \sum_{i=0}^{\infty} \frac{(x-a)^i}{i!} f^{(i)}(a).$$

Expansions:

$$\frac{1}{1-x} = 1 + x + x^2 + x^3 + x^4 + \cdots = \sum_{i=0}^{\infty} x^i,$$

$$\frac{1}{1-cx} = 1 + cx + c^2x^2 + c^3x^3 + \cdots = \sum_{i=0}^{\infty} c^i x^i,$$

$$\frac{1}{1-x^n} = 1 + x^n + x^{2n} + x^{3n} + \cdots = \sum_{i=0}^{\infty} x^{ni},$$

$$\frac{x}{(1-x)^2} = x + 2x^2 + 3x^3 + 4x^4 + \cdots = \sum_{i=0}^{\infty} i x^i,$$

$$x^k \frac{d^n}{dx^n}\left(\frac{1}{1-x}\right) = x + 2^n x^2 + 3^n x^3 + 4^n x^4 + \cdots = \sum_{i=0}^{\infty} i^n x^i,$$

$$e^x = 1 + x + \tfrac{1}{2}x^2 + \tfrac{1}{6}x^3 + \cdots = \sum_{i=0}^{\infty} \frac{x^i}{i!},$$

$$\ln(1+x) = x - \tfrac{1}{2}x^2 + \tfrac{1}{3}x^3 - \tfrac{1}{4}x^4 - \cdots = \sum_{i=1}^{\infty} (-1)^{i+1}\frac{x^i}{i},$$

$$\ln\frac{1}{1-x} = x + \tfrac{1}{2}x^2 + \tfrac{1}{3}x^3 + \tfrac{1}{4}x^4 + \cdots = \sum_{i=1}^{\infty} \frac{x^i}{i},$$

$$\sin x = x - \tfrac{1}{3!}x^3 + \tfrac{1}{5!}x^5 - \tfrac{1}{7!}x^7 + \cdots = \sum_{i=0}^{\infty} (-1)^i \frac{x^{2i+1}}{(2i+1)!},$$

$$\cos x = 1 - \tfrac{1}{2!}x^2 + \tfrac{1}{4!}x^4 - \tfrac{1}{6!}x^6 + \cdots = \sum_{i=0}^{\infty} (-1)^i \frac{x^{2i}}{(2i)!},$$

$$\tan^{-1} x = x - \tfrac{1}{3}x^3 + \tfrac{1}{5}x^5 - \tfrac{1}{7}x^7 + \cdots = \sum_{i=0}^{\infty} (-1)^i \frac{x^{2i+1}}{(2i+1)},$$

$$(1+x)^n = 1 + nx + \tfrac{n(n-1)}{2}x^2 + \cdots = \sum_{i=0}^{\infty} \binom{n}{i} x^i,$$

$$\frac{1}{(1-x)^{n+1}} = 1 + (n+1)x + \tbinom{n+2}{2}x^2 + \cdots = \sum_{i=0}^{\infty} \binom{i+n}{i} x^i,$$

$$\frac{x}{e^x - 1} = 1 - \tfrac{1}{2}x + \tfrac{1}{12}x^2 - \tfrac{1}{720}x^4 + \cdots = \sum_{i=0}^{\infty} \frac{B_i x^i}{i!},$$

$$\frac{1}{2x}(1 - \sqrt{1-4x}) = 1 + x + 2x^2 + 5x^3 + \cdots = \sum_{i=0}^{\infty} \frac{1}{i+1}\binom{2i}{i} x^i,$$

$$\frac{1}{\sqrt{1-4x}} = 1 + x + 2x^2 + 6x^3 + \cdots = \sum_{i=0}^{\infty} \binom{2i}{i} x^i,$$

$$\frac{1}{\sqrt{1-4x}}\left(\frac{1-\sqrt{1-4x}}{2x}\right)^n = 1 + (2+n)x + \tbinom{4+n}{2}x^2 + \cdots = \sum_{i=0}^{\infty} \binom{2i+n}{i} x^i,$$

$$\frac{1}{1-x}\ln\frac{1}{1-x} = x + \tfrac{3}{2}x^2 + \tfrac{11}{6}x^3 + \tfrac{25}{12}x^4 + \cdots = \sum_{i=1}^{\infty} H_i x^i,$$

$$\frac{1}{2}\left(\ln\frac{1}{1-x}\right)^2 = \tfrac{1}{2}x^2 + \tfrac{3}{4}x^3 + \tfrac{11}{24}x^4 + \cdots = \sum_{i=2}^{\infty} \frac{H_{i-1}x^i}{i},$$

$$\frac{x}{1-x-x^2} = x + x^2 + 2x^3 + 3x^4 + \cdots = \sum_{i=0}^{\infty} F_i x^i,$$

$$\frac{F_n x}{1 - (F_{n-1} + F_{n+1})x - (-1)^n x^2} = F_n x + F_{2n}x^2 + F_{3n}x^3 + \cdots = \sum_{i=0}^{\infty} F_{ni} x^i.$$

Ordinary power series:

$$A(x) = \sum_{i=0}^{\infty} a_i x^i.$$

Exponential power series:

$$A(x) = \sum_{i=0}^{\infty} a_i \frac{x^i}{i!}.$$

Dirichlet power series:

$$A(x) = \sum_{i=1}^{\infty} \frac{a_i}{i^x}.$$

Binomial theorem:

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

Difference of like powers:

$$x^n - y^n = (x-y)\sum_{k=0}^{n-1} x^{n-1-k} y^k.$$

For ordinary power series:

$$\alpha A(x) + \beta B(x) = \sum_{i=0}^{\infty} (\alpha a_i + \beta b_i) x^i,$$

$$x^k A(x) = \sum_{i=k}^{\infty} a_{i-k} x^i,$$

$$\frac{A(x) - \sum_{i=0}^{k-1} a_i x^i}{x^k} = \sum_{i=0}^{\infty} a_{i+k} x^i,$$

$$A(cx) = \sum_{i=0}^{\infty} c^i a_i x^i,$$

$$A'(x) = \sum_{i=0}^{\infty} (i+1) a_{i+1} x^i,$$

$$xA'(x) = \sum_{i=1}^{\infty} i a_i x^i,$$

$$\int A(x)\,dx = \sum_{i=1}^{\infty} \frac{a_{i-1}}{i} x^i,$$

$$\frac{A(x) + A(-x)}{2} = \sum_{i=0}^{\infty} a_{2i} x^{2i},$$

$$\frac{A(x) - A(-x)}{2} = \sum_{i=0}^{\infty} a_{2i+1} x^{2i+1}.$$

Summation: If $b_i = \sum_{j=0}^{i} a_i$ then

$$B(x) = \frac{1}{1-x} A(x).$$

Convolution:

$$A(x)B(x) = \sum_{i=0}^{\infty}\left(\sum_{j=0}^{i} a_j b_{i-j}\right) x^i.$$

God made the natural numbers;
all the rest is the work of man.
– Leopold Kronecker