Team Contest Reference
Team: Marcel Ultras

Lennart Meyling
Maximilian Saal
Devin Zielke

Contents

1 dp

1.1 LCS

```python
def LCS(S1, S2, m, n):
    L = [[0 for x in range(n + 1)] for x in range(m +
        1)]
    for i in range(m + 1):
        for j in range(n + 1):
            if i == 0 or j == 0:
                L[i][j] = 0
            elif S1[i - 1] == S2[j - 1]:
                L[i][j] = L[i - 1][j - 1] + 1
            else:
                L[i][j] = max(L[i - 1][j], L[i][j -
                    1])
    index = L[m][n]
    lcs_algo = [""] * (index + 1)
    lcs_algo[index] = ""
    i = m
    j = n
    while i > 0 and j > 0:
        if S1[i - 1] == S2[j - 1]:
            lcs_algo[index - 1] = S1[i - 1]
            i -= 1
            j -= 1
            index -= 1
        elif L[i - 1][j] > L[i][j - 1]:
            i -= 1
        else:
            j -= 1
    return lcs_algo
```

MD5: d4c2c050089e656220b23f7d1fd6963f | $\mathcal{O}(n^2)$

1.2 LIS

```python
def LIS(A, strict=True):
    from bisect import bisect_left
    T = []
    position = []

    for a in A:
        if len(T) == 0 or (strict and T[-1] < a) or (
            not strict and T[-1] <= a):
            position.append(len(T))
            T.append(a)
        else:
            if strict:
                k = bisect_left(T, a)
            else:
                k = bisect_left(T, a + 1)
            position.append(k)
            T[k] = a
    res = []
    t = len(T) - 1
    for i, p in enumerate(reversed(position)):
        if t == p:
            res.append(len(A) - 1 - i)
            t -= 1
    res.reverse()
    return res
```

MD5: 2ac7f6b4312cecab73e02153e1a764e8 | $\mathcal{O}(nlogn)$

1.3 TSP

```python
M={};Z={}
N=frozenset(range(1,len(dist_m)))
def dist(ni,N):
    if not N:
        Z[(ni,N)]=dist_m[ni][0]
        return
    for nj in N:
        if (nj,N.difference({nj})) not in Z:
            dist(nj,N.difference({nj}))
    c=[(nj,dist_m[ni][nj]+Z[(nj,N.difference({nj}))])
        for nj in N]
    nmin,min_cost=min(c,key=lambda x:x[1])
    M[(ni,N)] = nmin
    Z[(ni,N)] = min_cost
min_dist = dist(0,N)
ni = 0
solution = [0]

while N:
    ni = M[(ni, N)]
    solution.append(ni)
    N = N.difference({ni})
print(solution)
```

MD5: f546f5dabd450187bd027ac8a9b393c2 | $\mathcal{O}(2n * 2^n)$

1.4 hungarian

```python
def hungarian(A):
    inf = 1 << 40
    n = len(A) + 1
    m = len(A[0]) + 1
    P = [0] * m
    way = [0] * m
    U = [0] * n
    V = [0] * n
    for i in range(1, n):
        P[0] = i
        minV = [inf] * m
        used = [False] * m
        j0 = 0
        while P[j0] != 0:
            i0 = P[j0]
            j1 = 0
```

```python
            used[j0] = True
            delta = inf
            for j in range(1, m):
                if used[j]:
                    continue
                if i0 == 0 or j == 0:
                    cur =  -U[i0] - V[j]
                else:
                    cur = A[i0 - 1][j - 1] - U[i0] - V
                        [j]
                if cur < minV[j]:
                    minV[j] = cur
                    way[j] = j0
                if minV[j] < delta:
                    delta = minV[j]
                    j1 = j
            for j in range(m):
                if used[j]:
                    U[P[j]] += delta
                    V[j] -= delta
                else:
                    minV[j] -= delta
            j0 = j1
        P[j0] = P[way[j0]]
        j0 = way[j0]
        while j0 != 0:
            P[j0] = P[way[j0]]
            j0 = way[j0]

    ret = [-1] * (n - 1)
    for i in range(1, m):
        if P[i] != 0:
            ret[P[i] - 1] = i - 1
    return -V[0], ret
```

MD5: `482834ccbe5fe1dab437f8562dadd046` | $\mathcal{O}(n^3)$

2 ds

2.1 DSU

```python
class DisjointSetUnion():
    def __init__(self, n):
        self.n = n
        self.par_size = [-1] * n

    def merge(self, a, b):
        x = self.leader(a)
        y = self.leader(b)
        if x == y: return x
        if -self.par_size[x] < -self.par_size[y]: x, y
             = y, x
        self.par_size[x] += self.par_size[y]
        self.par_size[y] = x
        return x

    def same(self, a, b):
        return self.leader(a) == self.leader(b)

    def leader(self, a):
        x = a
        while self.par_size[x] >= 0:
            x = self.par_size[x]
        while self.par_size[a] >= 0:
            self.par_size[a] = x
            a = self.par_size[a]
        return x

    def size(self, a):
        return -self.par_size[self.leader(a)]

    def groups(self):
        leader_buf = [0] * self.n
        group_size = [0] * self.n
        res = [[] for _ in range(self.n)]
        for i in range(self.n):
            leader_buf[i] = self.leader(i)
            group_size[leader_buf[i]] += 1
        for i in range(self.n):
            res[leader_buf[i]].append(i)
        res = [res[i] for i in range(self.n) if res[i
            ]]
        return res
```

MD5: `6c03d887222f3ef9bbcdcef95855ae0a` | $\mathcal{O}(?)$

2.2 Double Prio Q

```python
import sys
from heapq import heappush, heappop

class DoubleEndedPriorityQueue:
    def __init__(self, arr: List[int] = None) -> None:
        self.hq1 = []
        self.hq2 = []
        if arr:
            self.used = bytearray(len(S))
            self.idx = len(S)
            for i, x in enumerate(S):
                tmp = x << 20 | i
                heappush(self.hq1, tmp)
                heappush(self.hq2, ~tmp)
        else:
            self.used = bytearray()
            self.idx = 0

    def pop_min(self) -> int:
        while 1:
            tmp = heappop(self.hq1)
            x, i = tmp >> 20, tmp & 0xfffff
            if self.used[i]: continue
            self.used[i] = 1
            return x

    def pop_max(self) -> int:
        while 1:
            tmp = heappop(self.hq2)
            tmp = ~tmp
            x, i = tmp >> 20, tmp & 0xfffff
            if self.used[i]: continue
            self.used[i] = 1
            return x

    def push(self, x: int) -> None:
        tmp = x << 20 | self.idx
        heappush(self.hq1, tmp)
        heappush(self.hq2, ~tmp)
        self.used.append(0)
        self.idx += 1
```

MD5: `135325808cc444038576cb368b0a7ce8` | $\mathcal{O}(?)$

2.3 SegTree

```python
# Python3 Code Addition

# limit for array size
N = 100000;

# Max size of tree
tree = [0] * (2 * N);

# function to build the tree
def build(arr) :

  # insert leaf nodes in tree
  for i in range(n) :
    tree[n + i] = arr[i];

  # build the tree by calculating parents
  for i in range(n - 1, 0, -1) :
    tree[i] = tree[i << 1] + tree[i << 1 | 1];

# function to update a tree node
def updateTreeNode(p, value) :

  # set value at position p
  tree[p + n] = value;
  p = p + n;

  # move upward and update parents
  i = p;

  while i > 1 :

    tree[i >> 1] = tree[i] + tree[i ^ 1];
    i >>= 1;

# function to get sum on interval [l, r)
def query(l, r) :

  res = 0;

  # loop to find the sum in the range
  l += n;
  r += n;

  while l < r :

    if (l & 1) :
      res += tree[l];
      l += 1

    if (r & 1) :
      r -= 1;
      res += tree[r];

    l >>= 1;
    r >>= 1

  return res;

# Driver Code
if __name__ == "__main__" :

  a = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12];

  # n is global
  n = len(a);

  # build tree
  build(a);

  # print the sum in range(1,2) index-based
  print(query(1, 3));

  # modify element at 2nd index
  updateTreeNode(2, 1);

  # print the sum in range(1,2) index-based
  print(query(1, 3));

# This code is contributed by AnkitRai01
```

MD5: `d8a873e7f8400180a2122e712ac3b6ec` | $\mathcal{O}(?)$

2.4 Trie

```python
class Trie:
    def __init__(self, words):
        self.root = {}
        for word in words:
            self.add(word)

    def add(self, word):
        cur = self.root
        for letter in word:
            if letter in cur:
                cur = cur[letter]
            else:
                cur[letter] = {}
                cur = cur[letter]

    def query(self, word):
        max_xor = 0
        cur = self.root
        l = len(word)
        cnt = 0
        for letter in word:
            cnt += 1
            if (1 ^ letter) in cur:
                max_xor ^= 2 ** (l - cnt)
                cur = cur[letter ^ 1]
            else:
                if letter not in cur:
                    return -1
                cur = cur[letter]
        return max_xor
```

MD5: `6993443b98053e208f55edeb37ad3cd5` | $\mathcal{O}(?)$

2.5 Fenwick-Tree

```python
class FenwickTree():
    def __init__(self, n):
        self.n = n
        self.data = [0] * n
    def build(self, arr):
        for i, a in enumerate(arr):
            self.data[i] = a
        for i in range(1, self.n + 1):
            if i + (i & -i) <= self.n:
                self.data[i + (i & -i) - 1] += self.
                    data[i - 1]
    def add(self, p, x):
```

```python
            p += 1
        while p <= self.n:
            self.data[p - 1] += x
            p += p & -p
    def sum(self, r):
        s = 0
        while r:
            s += self.data[r - 1]
            r -= r & -r
        return s
    def range_sum(self, l, r):
        #assert 0 <= l <= r <= self.n
        return self.sum(r) - self.sum(l)
```

MD5: d291e9d6c6b1f3c72f795e0889401184 | $\mathcal{O}(logn)$

2.6 Range count query

```python
from bisect import bisect_left
from collections import defaultdict
class RangeCountQuery:
    def __init__(self, arr):
        self.depth = defaultdict(list)
        for i, e in enumerate(arr):
            self.depth[e].append(i)

    def count(self, l, r, x):
        """l <= k < r """
        a = self.depth[x]
        s = bisect_left(a, l)
        t = bisect_left(a, r, s)
        return t - s
```

MD5: 8f057ba70089cb686d1ba3e3cf770068 | $\mathcal{O}(?)$

2.7 Range min query

```python
import sys
class RMQ:
    def __init__(self, n):
        self.sz = 1
        self.inf = (1 << 31) - 1
        while self.sz <= n: self.sz = self.sz << 1
        self.dat = [self.inf] * (2 * self.sz - 1)

    def update(self, idx, x):
        idx += self.sz - 1
        self.dat[idx] = x
        while idx > 0:
            idx = (idx - 1) >> 1
            self.dat[idx] = min(self.dat[idx * 2 + 1],
                 self.dat[idx * 2 + 2])

    def query(self, a, b):
        return self.query_help(a, b, 0, 0, self.sz)

    def query_help(self, a, b, k, l, r):
        if r <= a or b <= l:
            return sys.maxsize
        elif a <= l and r <= b:
            return self.dat[k]
        else:
            return min(self.query_help(a, b, 2 * k +
                1, l, (l + r) >> 1),
                       self.query_help(a, b, 2 * k +
                 2, (l + r) >> 1, r))
```

MD5: 8d227bc7192fa2aa70ef41bbbf1f4cbd | $\mathcal{O}(?)$

2.8 Sorted Set

```python
from typing import Union
class SortedSet:
    def __init__(self, u: int):

        self.log = (u - 1).bit_length()
        self.size = 1 << self.log
        self.u = u
        self.data = bytearray(self.size << 1)

    def add(self, k: int) -> bool:
        k += self.size
        if self.data[k]: return False
        self.data[k] = 1
        while k > 1:
            k >>= 1
            if self.data[k]: break
            self.data[k] = 1
        return True
    def discard(self, k: int) -> bool:
        k += self.size
        if self.data[k] == 0: return False
        self.data[k] = 0
        while k > 1:
            if k & 1:
                if self.data[k - 1]: break
            else:
                if self.data[k + 1]: break
            k >>= 1
            self.data[k] = 0
        return True
    def __contains__(self, k: int):
        return self.data[k + self.size] == 1
    def get_min(self) -> Union[int, None]:
        if self.data[1] == 0: return None
        k = 1
        while k < self.size:
            k <<= 1
            if self.data[k] == 0: k |= 1
        return k - self.size
    def get_max(self) -> Union[int, None]:
        if self.data[1] == 0: return None
        k = 1
        while k < self.size:
            k <<= 1
            if self.data[k | 1]: k |= 1
        return k - self.size

    def lt(self, k: int) -> Union[int, None]:

        if self.data[1] == 0: return -1
        x = k
        k += self.size
        while k > 1:
            if k & 1 and self.data[k - 1]:
                k >>= 1
                break
            k >>= 1
        k <<= 1
        if self.data[k] == 0: return -1
        while k < self.size:
```

```python
            k <<= 1
            if self.data[k | 1]: k |= 1
        k -= self.size
        return k if k < x else -1

    def le(self, k: int) -> Union[int, None]:
        if self.data[k + self.size]: return k
        return self.lt(k)

    def gt(self, k: int) -> Union[int, None]:
        if self.data[1] == 0: return -1
        x = k
        k += self.size
        while k > 1:
            if k & 1 == 0 and self.data[k + 1]:
                k >>= 1
                break
            k >>= 1
        k = k << 1 | 1
        while k < self.size:
            k <<= 1
            if self.data[k] == 0: k |= 1
        k -= self.size
        return k if k > x and k < self.u else -1

    def ge(self, k: int) -> Union[int, None]:
        if self.data[k + self.size]: return k
        return self.gt(k)
```

MD5: `cc50d52e7105d808eee80adf1f66bd0c` | $\mathcal{O}(?)$

3 graph

3.1 MST

```python
class Graph:

    def __init__(self, vertices):
        self.V = vertices
        self.graph = []

    def addEdge(self, u, v, w):
        self.graph.append([u, v, w])

    def find(self, parent, i):
        if parent[i] != i:
            parent[i] = self.find(parent, parent[i])
        return parent[i]

    def union(self, parent, rank, x, y):
        if rank[x] < rank[y]:
            parent[x] = y
        elif rank[x] > rank[y]:
            parent[y] = x
        else:
            parent[y] = x
            rank[x] += 1

    def KruskalMST(self):
        result = []
        i = 0
        e = 0
        self.graph = sorted(self.graph,
                            key=lambda item: item[2])
        parent = []
        rank = []
        for node in range(self.V):
            parent.append(node)
            rank.append(0)
        while e < self.V - 1:
            u, v, w = self.graph[i]
            i = i + 1
            x = self.find(parent, u)
            y = self.find(parent, v)
            if x != y:
                e = e + 1
                result.append([u, v, w])
                self.union(parent, rank, x, y)

        minimumCost = 0
        for u, v, weight in result:
            minimumCost += weight
        return minimumCost

class Solution:
    def minCostConnectPoints(self, points: List[List[int]]) -> int:
        graph = Graph(len(points))
        for i in range(0, len(points)):
            for x in range(i + 1, len(points)):
                graph.addEdge(i, x, abs(points[i][0]-points[x][0])+abs(points[i][1]-points[x][1]))
        return graph.KruskalMST()
```

MD5: `d6b2891b163bd1b00cc65b79c8bf7271` | $\mathcal{O}(fast)$

3.2 LCA

input = parent of n-1 verticest (0 is root)

```python
from sys import stdin
from collections import deque

class UnionFind():
    def __init__(self, p):
        N = len(p)
        timer = 0
        cnt = [0] * N
        que = deque()
        self.parent_or_size = [-1] * N
        self.parent = [0] * N
        self.edge = [0] * N
        self.order = [0] * N

        for i in range(N):
            cnt[p[i]] += 1

        for i in range(N):
            if cnt[i] == 0:
                que.append(i)

        for _ in range(N - 1):
            v = que.popleft()
            par = p[v]
            x, y = self.leader(v), self.leader(par)
            if self.parent_or_size[x] > self.parent_or_size[y]: x, y = y, x
            self.parent_or_size[x] += self.parent_or_size[y]
            self.parent_or_size[y] = x
            self.parent[y] = x
            self.edge[y] = par
```

```python
            self.order[y] = timer
            timer += 1
            cnt[par] -= 1
            if cnt[par] == 0: que.append(par)

        self.order[self.leader(0)] = timer

    def leader(self, v):
        if self.parent_or_size[v] < 0: return v
        self.parent_or_size[v] = self.leader(self.
            parent_or_size[v])
        return self.parent_or_size[v]

    def lca(self, u, v):
        lcav = v
        while u != v:
            if self.order[u] < self.order[v]: u, v = v
                , u
            lcav = self.edge[v]
            v = self.parent[v]
        return lcav

N, Q = map(int, stdin.readline().split())
p = [0] + list(map(int, stdin.readline().split()))

uf = UnionFind(p)

for _ in range(Q):
    query = list(map(int, stdin.readline().split()))
    print(uf.lca(query[0], query[1]))
```

MD5: 1b0324312b616d266c73fe9ce7efd1af | $\mathcal{O}(fast)$

3.3 Strongest CC

```python
#Takes numbers as nodes, scc stores components as
    array, store node in connections array

V = len(nodes)
g, gt = [[] for _ in range(V)], [[] for _ in range(V)]
for a in connections:
    for b in connections[a]:
        g[a].append(b)
        gt[b].append(a)

top, vis, scc = [], set(), []

def DFS(s, add):
    vis.add(s)
    a = gt if add else g
    for v in a[s]:
        if v not in vis: DFS(v, add)
    if add:
        top.append(s)
    else:
        scc[-1].append(stores[s])

for i in range(V):
    if i not in vis: DFS(i, True)
vis.clear()
for i in top[::-1]:
    if i not in vis: scc.append([]), DFS(i, False)
```

MD5: 1e80c29b7df554b4c4b61721a22548ac | $\mathcal{O}(fast)$

3.4 Maximum Bipartite Matching

```python
class BipartiteMatching:
    def __init__(self, n, m):
        self._n = n
        self._m = m
        self._to = [[] for _ in range(n)]

    def add_edge(self, a, b):
        self._to[a].append(b)

    def solve(self):
        n, m, to = self._n, self._m, self._to
        prev = [-1] * n
        root = [-1] * n
        p = [-1] * n
        q = [-1] * m
        updated = True
        while updated:
            updated = False
            s = []
            s_front = 0
            for i in range(n):
                if p[i] == -1:
                    root[i] = i
                    s.append(i)
            while s_front < len(s):
                v = s[s_front]
                s_front += 1
                if p[root[v]] != -1:
                    continue
                for u in to[v]:
                    if q[u] == -1:
                        while u != -1:
                            q[u] = v
                            p[v], u = u, p[v]
                            v = prev[v]
                        updated = True
                        break
                    u = q[u]
                    if prev[u] != -1:
                        continue
                    prev[u] = v
                    root[u] = root[v]
                    s.append(u)
            if updated:
                for i in range(n):
                    prev[i] = -1
                    root[i] = -1
        return [(v, p[v]) for v in range(n) if p[v] !=
             -1]
```

MD5: 6c08e3f2668368058df74bc9b41fc041 | $\mathcal{O}(Fast)$

3.5 maxflow

Finds the greatest flow in a graph. Capacities must be positive.

```python
from collections import deque

class MaxFlow():
    def __init__(self, n):
        self.n = n
        self.graph = [[] for _ in range(n)]
        self.pos = []

    def add_edge(self, fr, to, cap):
```

```python
        m = len(self.pos)
        self.pos.append((fr, len(self.graph[fr])))
        fr_id = len(self.graph[fr])
        to_id = len(self.graph[to])
        if fr == to: to_id += 1
        self.graph[fr].append([to, to_id, cap])
        self.graph[to].append([fr, fr_id, 0])
        return m

    def get_edge(self, idx):
        to, rev, cap = self.graph[self.pos[idx][0]][self.pos[idx][1]]
        rev_to, rev_rev, rev_cap = self.graph[to][rev]
        return rev_to, to, cap + rev_cap, rev_cap

    def edges(self):
        m = len(self.pos)
        for i in range(m):
            yield self.get_edge(i)

    def dfs(self, s, t, up):
        stack = [t]
        while stack:
            v = stack.pop()
            if v == s:
                flow = up
                for v in stack:
                    to, rev, cap = self.graph[v][self.iter[v]]
                    flow = min(flow, self.graph[to][rev][2])
                for v in stack:
                    self.graph[v][self.iter[v]][2] += flow
                    to, rev, cap = self.graph[v][self.iter[v]]
                    self.graph[to][rev][2] -= flow
                return flow
            lv = self.level[v]
            for i in range(self.iter[v], len(self.graph[v])):
                to, rev, cap = self.graph[v][i]
                if lv > self.level[to] and self.graph[to][rev][2]:
                    self.iter[v] = i
                    stack.append(v)
                    stack.append(to)
                    break
            else:
                self.iter[v] = len(self.graph[v])
                self.level[v] = self.n
        return 0

    def max_flow(self, s, t):
        return self.max_flow_with_limit(s, t, 2**63 - 1)

    def max_flow_with_limit(self, s, t, limit):
        flow = 0
        while flow < limit:
            self.level = [-1] * self.n
            self.level[s] = 0
            queue = deque()
            queue.append(s)
            while queue:
                v = queue.popleft()
                for to, rev, cap in self.graph[v]:
                    if cap == 0 or self.level[to] >= 0: continue
                    self.level[to] = self.level[v] + 1
                    if to == t: break
                    queue.append(to)
            if self.level[t] == -1: break
            self.iter = [0] * self.n
            while flow < limit:
                f = self.dfs(s, t, limit - flow)
                if not f: break
                flow += f
        return flow
```

MD5: 6f622fc0f20b6d5813a979b581580661 | $\mathcal{O}(fast)$

4 math

4.1 DET

```python
mod = 998244353
def determinant(A, replace=False):
    if not replace:
        A = [a.copy() for a in A]
    n = len(A)
    res = 1
    for i, a_i in enumerate(A):
        if a_i[i] == 0:
            for j in range(i+1, n):
                if A[j][i]:
                    break
            else:
                return 0
            A[i], A[j] = A[j], A[i]
            a_i = A[i]
            res = -res
        inv = pow(a_i[i], mod-2, mod)
        for j in range(i+1, n):
            a_j = A[j]
            t = a_j[i] * inv % mod
            for k in range(i+1, n):
                a_j[k] -= t * a_i[k]
                a_j[k] %= mod
    for i in range(n):
        res *= A[i][i]
        res %= mod
    return res
```

MD5: ce49b2302df27710ecf92ad34d7a615a | $\mathcal{O}(N)$

4.2 FFT

```python

import math

class FFT():
    def primitive_root_constexpr(self, m):
        if m == 2: return 1
        if m == 167772161: return 3
        if m == 469762049: return 3
        if m == 754974721: return 11
        if m == 998244353: return 3
        divs = [0] * 20
        divs[0] = 2
        cnt = 1
        x = (m - 1) // 2
```

```python
        while (x % 2 == 0): x //= 2
        i = 3
        while (i * i <= x):
            if (x % i == 0):
                divs[cnt] = i
                cnt += 1
                while (x % i == 0):
                    x //= i
            i += 2
        if x > 1:
            divs[cnt] = x
            cnt += 1
        g = 2
        while (1):
            ok = True
            for i in range(cnt):
                if pow(g, (m - 1) // divs[i], m) == 1:
                    ok = False
                    break
            if ok:
                return g
            g += 1

    def bsf(self, x):
        res = 0
        while (x % 2 == 0):
            res += 1
            x //= 2
        return res

    rank2 = 0
    root = []
    iroot = []
    rate2 = []
    irate2 = []
    rate3 = []
    irate3 = []

    def __init__(self, MOD):
        self.mod = MOD
        self.g = self.primitive_root_constexpr(self.mod)
        self.rank2 = self.bsf(self.mod - 1)
        self.root = [0 for i in range(self.rank2 + 1)]
        self.iroot = [0 for i in range(self.rank2 + 1)]
        self.rate2 = [0 for i in range(self.rank2)]
        self.irate2 = [0 for i in range(self.rank2)]
        self.rate3 = [0 for i in range(self.rank2 - 1)]
        self.irate3 = [0 for i in range(self.rank2 - 1)]
        self.root[self.rank2] = pow(self.g, (self.mod - 1) >> self.rank2, self.mod)
        self.iroot[self.rank2] = pow(self.root[self.rank2], self.mod - 2, self.mod)
        for i in range(self.rank2 - 1, -1, -1):
            self.root[i] = (self.root[i + 1] ** 2) % self.mod
            self.iroot[i] = (self.iroot[i + 1] ** 2) % self.mod
        prod = 1;
        iprod = 1
        for i in range(self.rank2 - 1):
            self.rate2[i] = (self.root[i + 2] * prod) % self.mod
            self.irate2[i] = (self.iroot[i + 2] * iprod) % self.mod
            prod = (prod * self.iroot[i + 2]) % self.mod
            iprod = (iprod * self.root[i + 2]) % self.mod
        prod = 1;
        iprod = 1
        for i in range(self.rank2 - 2):
            self.rate3[i] = (self.root[i + 3] * prod) % self.mod
            self.irate3[i] = (self.iroot[i + 3] * iprod) % self.mod
            prod = (prod * self.iroot[i + 3]) % self.mod
            iprod = (iprod * self.root[i + 3]) % self.mod

    def butterfly(self, a):
        n = len(a)
        h = (n - 1).bit_length()

        LEN = 0
        while (LEN < h):
            if (h - LEN == 1):
                p = 1 << (h - LEN - 1)
                rot = 1
                for s in range(1 << LEN):
                    offset = s << (h - LEN)
                    for i in range(p):
                        l = a[i + offset]
                        r = a[i + offset + p] * rot
                        a[i + offset] = (l + r) % self.mod
                        a[i + offset + p] = (l - r) % self.mod
                    rot *= self.rate2[(~s & -~s).bit_length() - 1]
                    rot %= self.mod
                LEN += 1
            else:
                p = 1 << (h - LEN - 2)
                rot = 1
                imag = self.root[2]
                for s in range(1 << LEN):
                    rot2 = (rot * rot) % self.mod
                    rot3 = (rot2 * rot) % self.mod
                    offset = s << (h - LEN)
                    for i in range(p):
                        a0 = a[i + offset]
                        a1 = a[i + offset + p] * rot
                        a2 = a[i + offset + 2 * p] * rot2
                        a3 = a[i + offset + 3 * p] * rot3
                        a1na3imag = (a1 - a3) % self.mod * imag
                        a[i + offset] = (a0 + a2 + a1 + a3) % self.mod
                        a[i + offset + p] = (a0 + a2 - a1 - a3) % self.mod
                        a[i + offset + 2 * p] = (a0 - a2 + a1na3imag) % self.mod
                        a[i + offset + 3 * p] = (a0 - a2 - a1na3imag) % self.mod
                    rot *= self.rate3[(~s & -~s).bit_length() - 1]
                    rot %= self.mod
                LEN += 2
```

```python
    def butterfly_inv(self, a):
        n = len(a)
        h = (n - 1).bit_length()
        LEN = h
        while (LEN):
            if (LEN == 1):
                p = 1 << (h - LEN)
                irot = 1
                for s in range(1 << (LEN - 1)):
                    offset = s << (h - LEN + 1)
                    for i in range(p):
                        l = a[i + offset]
                        r = a[i + offset + p]
                        a[i + offset] = (l + r) % self.mod
                        a[i + offset + p] = (l - r) * irot % self.mod
                    irot *= self.irate2[(~s & -~s).bit_length() - 1]
                    irot %= self.mod
                LEN -= 1
            else:
                p = 1 << (h - LEN)
                irot = 1
                iimag = self.iroot[2]
                for s in range(1 << (LEN - 2)):
                    irot2 = (irot * irot) % self.mod
                    irot3 = (irot * irot2) % self.mod
                    offset = s << (h - LEN + 2)
                    for i in range(p):
                        a0 = a[i + offset]
                        a1 = a[i + offset + p]
                        a2 = a[i + offset + 2 * p]
                        a3 = a[i + offset + 3 * p]
                        a2na3iimag = (a2 - a3) * iimag % self.mod
                        a[i + offset] = (a0 + a1 + a2 + a3) % self.mod
                        a[i + offset + p] = (a0 - a1 + a2na3iimag) * irot % self.mod
                        a[i + offset + 2 * p] = (a0 + a1 - a2 - a3) * irot2 % self.mod
                        a[i + offset + 3 * p] = (a0 - a1 - a2na3iimag) * irot3 % self.mod
                    irot *= self.irate3[(~s & -~s).bit_length() - 1]
                    irot %= self.mod
                LEN -= 2

    def convolution(self, a, b):
        n = len(a);
        m = len(b)
        if not (a) or not (b):
            return []
        if min(n, m) <= 40:
            res = [0] * (n + m - 1)
            for i in range(n):
                for j in range(m):
                    res[i + j] += a[i] * b[j]
                    res[i + j] %= self.mod
            return res
        z = 1 << ((n + m - 2).bit_length())
        a = a + [0] * (z - n)
        b = b + [0] * (z - m)
        self.butterfly(a)
        self.butterfly(b)
        c = [(a[i] * b[i]) % self.mod for i in range(z)]
        self.butterfly_inv(c)
        iz = pow(z, self.mod - 2, self.mod)
        for i in range(n + m - 1):
            c[i] = (c[i] * iz) % self.mod
        return c[:n + m - 1]
```

```python
def inv_gcd(a, b):
    a = a % b
    if a == 0:
        return (b, 0)
    s = b;
    t = a
    m0 = 0;
    m1 = 1
    while (t):
        u = s // t
        s -= t * u
        m0 -= m1 * u
        s, t = t, s
        m0, m1 = m1, m0
    if m0 < 0:
        m0 += b // s
    return (s, m0)

def crt(r, m):
    assert len(r) == len(m)
    n = len(r)
    r0 = 0;
    m0 = 1
    for i in range(n):
        assert 1 <= m[i]
        r1 = r[i] % m[i]
        m1 = m[i]
        if m0 < m1:
            r0, r1 = r1, r0
            m0, m1 = m1, m0
        if (m0 % m1 == 0):
            if (r0 % m1 != r1):
                return (0, 0)
            continue
        g, im = inv_gcd(m0, m1)
        u1 = m1 // g
        if ((r1 - r0) % g):
            return (0, 0)
        x = (r1 - r0) // g % u1 * im % u1
        r0 += x * m0
        m0 *= u1
        if r0 < 0:
            r0 += m0
    return (r0, m0)

mod0 = 1012924417
mod1 = 167772161
mod2 = 469762049
mod3 = 1224736769
mod4 = 998244353
ntt0 = FFT(mod0)
ntt1 = FFT(mod1)
ntt2 = FFT(mod2)
ntt3 = FFT(mod3)
ntt4 = FFT(mod4)

def convolution_2pow64(a, b):
    mod = 1 << 64
    n = len(a)
```

```
    m = len(b)
    for i in range(n): a[i]
    for i in range(m): b[i]
    x0 = ntt0.convolution(a, b)
    x1 = ntt1.convolution(a, b)
    x2 = ntt2.convolution(a, b)
    x3 = ntt3.convolution(a, b)
    x4 = ntt4.convolution(a, b)
    ret = [0 for i in range(n + m - 1)]
    for i in range(n + m - 1):
        tmp = crt((x0[i], x1[i], x2[i], x3[i], x4[i]),
             (mod0, mod1, mod2, mod3, mod4))
        ret[i] = tmp[0] % mod
    return ret
```

MD5: 81010ba542ca59077527a18c77f90d2f | $\mathcal{O}(NlogN)$

4.3 GEO

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

    def cross(self,P):
        # pos = left, 0 = straight, neg = right
        return self.x*P.y - P.x * self.y

    def subtract(self, P):

        return Point(self.x-P.x, self.y - P.y)

    def same(self, P):
        return self.x == P.x and self.y == P.y

    def abst(self):
        return (self.x**2+self.y**2)**0.5

    def scal(self, P):
        return self.x*P.x + self.y*P.y

class Line:
    def __init__(self, P1, P2):
        self.P1 = P1
        self.P2 = P2

    def location(self, P):
        # 0 = on line, > 0 = left, < 0 = right
        return P.subtract(self.P1).cross(P.subtract(
            self.P2))

    def closes_point(self, P):

        u = self.P2.subtract(self.P1).abst()

        return abs(self.P1.subtract(P).cross(self.P2.
            subtract(P)) / u)

class Segment:
    def __init__(self, P1, P2):
        self.P1 = P1
        self.P2 = P2

    def intersect(self, S):
        V = self.P2.subtract(self.P1)
        C1 = V.cross(S.P1)
        C2 = V.cross(S.P2)
        if self.P1.same(S.P1) or self.P2.same(S.P1) or
             self.P1.same(S.P2) or self.P2.same(S.P2):
            return True

        if C1 == C2 == 0:
            LIST = [(self.P1.x,self.P1.y,0),(self.P2.x
                ,self.P2.y,1),(S.P1.x,S.P1.y,2),(S.P2.
                x,S.P2.y,3)]
            LIST.sort()
            if (LIST[0][2] + LIST[1][2] == 1) or  (
                LIST[2][2] + LIST[3][2] == 1):
                return False
            return True
        V1 = S.P2.subtract(S.P1)
        C3 = V1.cross(self.P1)
        C4 = V1.cross(self.P2)

        if C1 * C2 <= 0 and C3 * C4 <= 0:
            return True

        return False

import itertools
class ConvexHull():
    def __init__(self):
        self.points = []
    def add_point(self,x,y):
        self.points.append([x,y])

    def ccw(self, A, B, C):
        return (B[0]-A[0])*(C[1]-A[1]) - (B[1]-A[1])*(
            C[0]-A[0])

    def get_hull_points(self):

        if len(self.points) <= 1:
            return self.points

        hull = []
        self.points.sort()
        points = self.points
        for i in itertools.chain(range(len(points)),
            reversed(range(len(points)-1))):
            while len(hull) >= 2 and self.ccw(hull
                [-2], hull[-1], points[i]) < 0:
                hull.pop()
            hull.append(points[i])
        hull.pop()

        for i in range(1, (len(hull)+1)//2):
            if hull[i] != hull[-1]:
                break
            hull.pop()
        return hull
```

MD5: 2232d33ad491ecf4f90b005b059db934 | $\mathcal{O}(N)$

4.4 Linear Systems

Ax = B or something like that

```
MOD = 998244353

def linear_equations(mat, vec):
    n = len(mat)
    m = len(mat[0])
```

```python
    assert n == len(vec)
    aug = [mat[i] + [vec[i]] for i in range(n)]
    rank = 0
    p = []
    q = []
    for j in range(m + 1):
        for i in range(rank, n):
            if aug[i][j] != 0:
                break
        else:
            q.append(j)
            continue
        if j == m: return -1, [], []
        p.append(j)
        aug[rank], aug[i] = aug[i], aug[rank]
        inv = pow(aug[rank][j], MOD - 2, MOD)
        for k in range(m + 1):
            aug[rank][k] *= inv
            aug[rank][k] %= MOD
        for i in range(n):
            if i == rank: continue
            c = -aug[i][j]
            for k in range(m + 1):
                aug[i][k] += c * aug[rank][k]
                aug[i][k] %= MOD
        rank += 1
    dim = m - rank
    sol = [0] * m
    for i in range(rank):
        sol[p[i]] = aug[i][-1]
    vecs = [[0] * m for _ in range(dim)]
    for i in range(dim):
        vecs[i][q[i]] = 1
    for i in range(dim):
        for j in range(rank):
            vecs[i][p[j]] = -aug[j][q[i]] % MOD
    return dim, sol, vecs
```

MD5: 8e00b6593527aa48cc0748f1dd885e52 | $\mathcal{O}(N)$

4.5 MATMUL

```python
def mat_pro(A,B):
    N,M,K = len(A),len(A[0]),len(B[0])
    C = [[0]*K for _ in range(N)]
    for i in range(N):
        row_A = A[i]
        row_C = C[i]
        for j in range(M):
            a = row_A[j]
            row_B = B[j]
            for k in range(K):
                row_C[k] = (row_C[k]+a*row_B[k])%mod
    return C
mod = 998244353
```

MD5: 3dc72b294acdc69f80e1d515fe59aea2 | $\mathcal{O}(N^3)$

4.6 MATPOW

```python
from typing import Callable, Optional

class MatrixMod:
    _mod = 998244353
    def __init__(self, n: int, m: int, from_array: Optional[list[list[int]]] = None) -> None:
        self._n = n
        self._m = m
        if from_array is None:
            self._matrix = [[0] * m for _ in range(n)]
        else:
            self._matrix = [row[:] for row in from_array]

    @classmethod
    def set_mod(cls, mod: int) -> None:
        cls._mod = mod

    @classmethod
    def ie(cls, n: int) -> "MatrixMod":
        ret = cls(n, n)
        for i in range(n):
            ret[i, i] = 1
        return ret

    def is_square(self) -> bool:
        return self._n == self._m

    def __str__(self) -> str:
        return "\n".join(" ".join(map(str, row)) for row in self._matrix)

    def __getitem__(self, idxs: tuple[int, int]) -> int:
        return self._matrix[idxs[0]][idxs[1]]

    def __setitem__(self, idxs: tuple[int, int], value: int) -> None:
        self._matrix[idxs[0]][idxs[1]] = value

    def __add__(self, other: MatrixMod) -> MatrixMod:
        assert self._n == other._n and self._m == other._m
        ret = MatrixMod(self._n, self._m)
        for i in range(self._n):
            res_i = ret._matrix[i]
            self_i = self._matrix[i]
            other_i = other._matrix[i]
            for j in range(self._m):
                res_i[j] = (self_i[j] + other_i[j]) % self._mod
        return ret

    def __pos__(self) -> MatrixMod:
        return self

    def __neg__(self) -> MatrixMod:
        ret = MatrixMod(self._n, self._m)
        for i in range(self._n):
            res_i = ret._matrix[i]
            self_i = self._matrix[i]
            for j in range(self._m):
                res_i[j] = -self_i[j] % self._mod
        return ret

    def __sub__(self, other: MatrixMod) -> MatrixMod:
        assert self._n == other._n and self._m == other._m
        ret = MatrixMod(self._n, self._m)
        for i in range(self._n):
            res_i = ret._matrix[i]
            self_i = self._matrix[i]
```

```python
            other_i = other._matrix[i]
            for j in range(self._m):
                res_i[j] = (self_i[j] - other_i[j]) %
                    self._mod
        return ret

    def __mul__(self, other: MatrixMod) -> MatrixMod:
        assert self._m == other._n
        ret = MatrixMod(self._n, other._m)
        for i in range(self._n):
            res_i = ret._matrix[i]
            self_i = self._matrix[i]
            for k in range(self._m):
                self_ik = self_i[k]
                other_k = other._matrix[k]
                for j in range(other._m):
                    res_i[j] += self_ik * other_k[j]
                    res_i[j] %= self._mod
        return ret

    def times(self, k: int) -> MarixMod:
        ret = MatrixMod(self._n, self._m)
        for i in range(self._n):
            res_i = ret._matrix[i]
            self_i = self._matrix[i]
            for j in range(self._m):
                res_i[j] = self_i[j] * k % self._mod
        return ret

    def __pow__(self, k: int) -> MatrixMod:
        assert self._n == self._m
        ret = MatrixMod.ie(self._n)
        tmp = self
        while k:
            if k & 1:
                ret = ret * tmp
            tmp = tmp * tmp
            k >>= 1
        return ret

N, K = map(int, input().split())
A = MatrixMod(N, N, from_array = [list(map(int, input
    ().split())) for _ in range(N)])

B = A ** K

print(B)
```

MD5: d973ea209e2e1b4a87e15d5733f4cc7d | $\mathcal{O}(N^3)$

4.7 MOD

```python
MOD = 998244353

fac_arr = [1]
finv_arr = [1]

def enlarge_fac():
    old_size = len(fac_arr)
    new_size = old_size * 2
    for i in range(old_size, new_size + 1):
        fac_arr.append((fac_arr[-1] * i) % MOD)
        finv_arr.append(pow(fac_arr[-1], -1, MOD))

def fac(n):
    while n >= len(fac_arr): enlarge_fac()
    return fac_arr[n]

def finv(n):
    while n >= len(finv_arr): enlarge_fac()
    return finv_arr[n]

def binom(n, k):
    if k < 0 or k > n: return 0
    return ((fac(n) * finv(k)) % MOD * finv(n - k)) %
        MOD
```

MD5: 563f35f15f93d1fa344f70ccb432d791 | $\mathcal{O}(N)$

4.8 Fast prime check

```python
def is_prime(n):
    if n == 2: return 1
    if n == 1 or not n&1: return 0
    #miller_rabin
    if n < 1<<30: test_numbers = [2, 7, 61]
    else: test_numbers = [2, 325, 9375, 28178, 450775,
         9780504, 1795265022]
    d = n - 1
    while ~d&1: d>>=1
    for a in test_numbers:
        if n <= a: break
        t = d
        y = pow(a, t, n)
        while t != n-1 and y != 1 and y != n-1:
            y = y * y % n
            t <<= 1
        if y != n-1 and not t&1: return 0
    return 1
```

MD5: dd31122281a49a705d1930e030221355 | $\mathcal{O}(logN)$

5 misc

5.1 KMP

no idea what this does

```python
import sys
from sys import stdin
def KMPSearch(pat, txt):
    sol = []
    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:
            sol.append(i-j)
            j = lps[j - 1]

        elif i < N and pat[j] != txt[i]:
            if j != 0:
                j = lps[j - 1]
            else:
                i += 1
```

```
    print(*sol)

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
```

MD5: `d5c461938f0b8209b7f03e7256838fa1` | $\mathcal{O}(faster)$

5.2 Bootstrap

Use when desperate

```
def bootstrap(f, stack=[]):
    from types import GeneratorType
    def wrappedfunc(*args, **kwargs):
        if stack:
            return f(*args, **kwargs)
        else:
            to = f(*args, **kwargs)
            while True:
                if type(to) is GeneratorType:
                    stack.append(to)
                    to = next(to)
                else:
                    stack.pop()
                    if not stack:
                        break
                    to = stack[-1].send(to)
            return to

    return wrappedfunc
```

MD5: `026c45e94790fbc1d108dfccc34abb77` | $\mathcal{O}(faster)$

6 more math

6.1 Tree

Diameter: BFS from any node, then BFS from last visited node. Max dist is then the diameter. Center: Middle vertex in second step from above.

6.2 Divisability Explanation

$D \mid M \Leftrightarrow D \mid$ `digit_sum`(M, k, alt), refer to table for values of D, k, alt.

6.3 Combinatorics

- Variations (ordered): k out of n objects (permutations for $k = n$)
  - without repetition: $M = \{(x_1, \ldots, x_k) : 1 \leq x_i \leq n,\ x_i \neq x_j \text{ if } i \neq j\}$, $|M| = \frac{n!}{(n-k)!}$
  - with repetition: $M = \{(x_1, \ldots, x_k) : 1 \leq x_i \leq n\}, |M| = n^k$
- Combinations (unordered): k out of n objects
  - without repetition: $M = \{(x_1, \ldots, x_n) : x_i \in \{0, 1\},\ x_1 + \ldots + x_n = k\}, |M| = \binom{n}{k}$
  - with repetition: $M = \{(x_1, \ldots, x_n) : x_i \in \{0, 1, \ldots, k\},\ x_1 + \ldots + x_n = k\}, |M| = \binom{n+k-1}{k}$
- Ordered partition of numbers: $x_1 + \ldots + x_k = n$ (i.e. 1+3 = 3+1 = 4 are counted as 2 solutions)
  - #Solutions for $x_i \in \mathbb{N}_0$: $\binom{n+k-1}{k-1}$
  - #Solutions for $x_i \in \mathbb{N}$: $\binom{n-1}{k-1}$
- Unordered partition of numbers: $x_1 + \ldots + x_k = n$ (i.e. 1+3 = 3+1 = 4 are counted as 1 solution)
  - #Solutions for $x_i \in \mathbb{N}$: $P_{n,k} = P_{n-k,k} + P_{n-1,k-1}$ where $P_{n,1} = P_{n,n} = 1$
- Derangements (permutations without fixed points): $!n = n! \sum_{k=0}^{n} \frac{(-1)^k}{k!} = \lfloor \frac{n!}{e} + \frac{1}{2} \rfloor$

6.4 Polynomial Interpolation

6.4.1 Theory

Problem: for $\{(x_0, y_0), \ldots, (x_n, y_n)\}$ find $p \in \Pi_n$ with $p(x_i) = y_i$ for all $i = 0, \ldots, n$.
Solution: $p(x) = \sum_{i=0}^{n} \gamma_{0,i} \prod_{j=0}^{i-1} (x - x_i)$ where $\gamma_{j,k} = y_j$ for $k = 0$ and $\gamma_{j,k} = \frac{\gamma_{j+1,k-1} - \gamma_{j,k-1}}{x_{j+k} - x_j}$ otherwise.
Efficient evaluation of $p(x)$: $b_n = \gamma_{0,n}$, $b_i = b_{i+1}(x - x_i) + \gamma_{0,i}$ for $i = n-1, \ldots, 0$ with $b_0 = p(x)$.

6.5 Fibonacci Sequence

6.5.1 Binet's formula

$\begin{pmatrix} f_n \\ f_{n+1} \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}^n \begin{pmatrix} 0 \\ 1 \end{pmatrix} \Rightarrow f_n = \frac{1}{\sqrt{5}}(\phi^n - \tilde{\phi}^n)$ where $\phi = \frac{1+\sqrt{5}}{2}$ and $\tilde{\phi} = \frac{1-\sqrt{5}}{2}$.

6.5.2 Generalization

$g_n = \frac{1}{\sqrt{5}}(g_0(\phi^{n-1} - \tilde{\phi}^{n-1}) + g_1(\phi^n - \tilde{\phi}^n)) = g_0 f_{n-1} + g_1 f_n$ for all $g_0, g_1 \in \mathbb{N}_0$

6.5.3 Pisano Period

Both $(f_n \bmod k)_{n \in \mathbb{N}_0}$ and $(g_n \bmod k)_{n \in \mathbb{N}_0}$ are periodic.

6.6 Series

$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}, \sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}, \sum_{i=1}^{n} i^3 = \frac{n^2(n+1)^2}{4}$
$\sum_{i=0}^{n} c^i = \frac{c^{n+1}-1}{c-1}, c \neq 1, \sum_{i=0}^{\infty} c^i = \frac{1}{1-c}, \sum_{i=1}^{n} c^i = \frac{c}{1-c}, |c| < 1$
$\sum_{i=0}^{n} ic^i = \frac{nc^{n+2}-(n+1)c^{n+1}+c}{(c-1)^2}, c \neq 1, \sum_{i=0}^{\infty} ic^i = \frac{c}{(1-c)^2}, |c| < 1$

6.7 Binomial coefficients

$\binom{n}{k} = \binom{n-1}{k} + \binom{n-1}{k-1}$, $\binom{n}{m}\binom{m}{k} = \binom{n}{k}\binom{n-k}{m-k}$, $\binom{m+n}{r} = \sum_{k=0}^{r} \binom{m}{k}\binom{n}{r-k}$ and in general, $n_1 + \cdots + n_p = \sum_{k_1+\cdots+k_p=m} \binom{n_1}{k_1} \cdots \binom{n_p}{k_p}$

6.8 Catalan numbers

$C_n = \frac{1}{n+1}\binom{2n}{n} = \frac{(2n)!}{(n+1)!n!}$
$C_0 = 1, C_{n+1} = \sum_{k=0}^{n} C_k C_{n-k}, C_{n+1} = \frac{4n+2}{n+2} C_n$

6.9 Geometry

Area of a polygon: $A = \frac{1}{2}(x_1y_2 - x_2y_1 + x_2y_3 - x_3y_2 + \cdots + x_{n-1}y_n - x_ny_{n-1} + x_ny_1 - x_1y_n)$

6.10 Number Theory

Chinese Remainder Theorem: There exists a number C, such that:
$C \equiv a_1 \mod n_1, \cdots, C \equiv a_k \mod n_k, \mathrm{ggt}(n_i, n_j) = 1, i \neq j$
Case $k = 2$: $m_1n_1 + m_2n_2 = 1$ with EEA.
Solution is $x = a_1m_2n_2 + a_2m_1n_1$.
General case: iterative application of $k = 2$
Euler's φ-Funktion: $\varphi(n) = n \prod_{p|n}(1 - \frac{1}{p}), p$ prime
$\varphi(p) = p - 1, \varphi(pq) = \varphi(p)\varphi(q), p, q$ prime
$\varphi(p^k) = p^k - p^{k-1}, p, q$ prime, $k \geq 1$
Eulers Theorem: $a^{\varphi(n)} \equiv 1 \mod n$
Fermats Theorem: $a^p \equiv a \mod p, p$ prime

6.11 Convolution

$(f * g)(n) = \sum_{m=-\infty}^{\infty} f(m)g(n-m) = \sum_{m=-\infty}^{\infty} f(n-m)g(m)$

6.12 DP Optimization

- Convex Hull Optimization:

$$T[i] = \min_{j<i}(T[j] + b[j] \cdot a[i])$$

with the constraints $b[j] \geq b[j+1]$ and $a[j] \leq a[j+1]$. Solution is convex and thus the optimal j for i will always be smaller than the one for $i+1$. So we can use a pointer which we increment as long as the solution gets better. Running time is $\mathcal{O}(n)$ as the pointer visits each element no more than once.

- Divide and Conquer Optimization:

$$T[i][j] = \min_{k<j}(T[i-1][k] + C[k][j])$$

with the constraint $A[i][j] \leq A[i][j+1]$ with $A[i][j]$ giving the smallest optimal k. Is dealt with (including code) in misc chapter above.

- Knuth Optimization:

$$T[i][j] = \min_{i<k<j}(T[i][k] + T[k][j]) + C[i][j]$$

with the constraint $A[i][j-1] \leq A[i][j] \leq A[i+1][j]$ which is apparently equal to the following two constraints:

$$C[a][c] + C[b][d] \leq C[a][d] + C[b][c],\ a \leq b \leq c \leq d$$
$$C[b][c] \leq C[a][d],\ a \leq b \leq c \leq d$$

With above constraint we get good bounds on k by going calculating T with increasing $j - i$. Also see the code in misc.

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 $b \in$ such that $b \ge s$, $\forall s \in S$.
$\inf S$	greatest $b \in$ such that $b \le s$, $\forall s \in S$.
$\liminf\limits_{n\to\infty} a_n$	$\lim\limits_{n\to\infty} \inf\{a_i \mid i \ge n, i \in \}$.
$\limsup\limits_{n\to\infty} a_n$	$\lim\limits_{n\to\infty} \sup\{a_i \mid i \ge n, i \in \}$.
$\binom{n}{k}$	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 $\pi_1\pi_2\ldots\pi_n$ on $\{1, 2, \ldots, n\}$ 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:

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

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

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

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

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

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

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, ...

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

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:

$$\Pr\left[|X| \geq \lambda\,\mathrm{E}[X]\right] \leq \frac{1}{\lambda},$$

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

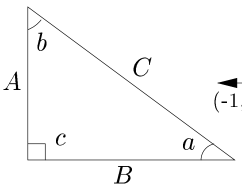
Geometric distribution:

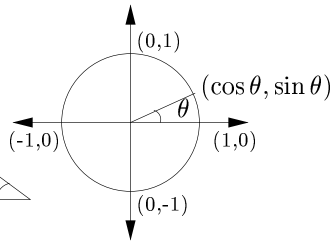
$$\Pr[X = k] = pq^{k-1}, \qquad q = 1 - p,$$

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

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:
$$\tfrac{1}{2}AB, \qquad \frac{AB}{A+B+C}.$$

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

v2.01 ©1994 by Steve Seiden
`sseiden@acm.org`
`http://www.csc.lsu.edu/~seiden`

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.
$$\det A \cdot B = \det A \cdot \det B,$$
$$\det A = \sum_{\pi} \prod_{i=1}^{n} \operatorname{sign}(\pi) a_{i,\pi(i)}.$$

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

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

Hyperbolic Functions

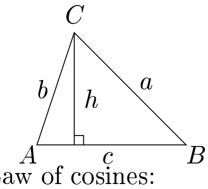
Definitions:
$$\sinh x = \frac{e^x - e^{-x}}{2}, \qquad \cosh x = \frac{e^x + e^{-x}}{2},$$
$$\tanh x = \frac{e^x - e^{-x}}{e^x + e^{-x}}, \qquad \operatorname{csch} x = \frac{1}{\sinh x},$$
$$\operatorname{sech} x = \frac{1}{\cosh x}, \qquad \coth x = \frac{1}{\tanh x}.$$

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

θ	$\sin\theta$	$\cos\theta$	$\tan\theta$
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	∞

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

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 $k \cdot c(G - S) \leq \|S\|$.
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 embedded 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 ≤ 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)	$(x, y, 1)$
$y = mx + b$	$(m, -1, b)$
$x = c$	$(1, 0, -c)$

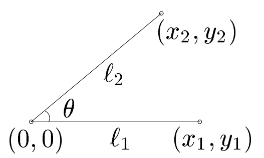
Distance formula, L_p and L_∞ metric:

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

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

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

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

π

Wallis' identity:
$$\pi = 2 \cdot \frac{2 \cdot 2 \cdot 4 \cdot 4 \cdot 6 \cdot 6 \cdots}{1 \cdot 3 \cdot 3 \cdot 5 \cdot 5 \cdot 7 \cdots}$$

Brouncker's continued fraction expansion:
$$\tfrac{\pi}{4} = 1 + \cfrac{1^2}{2 + \cfrac{3^2}{2 + \cfrac{5^2}{2 + \cfrac{7^2}{2 + \cdots}}}}$$

Gregrory's series:
$$\tfrac{\pi}{4} = 1 - \tfrac{1}{3} + \tfrac{1}{5} - \tfrac{1}{7} + \tfrac{1}{9} - \cdots$$

Newton's series:
$$\tfrac{\pi}{6} = \frac{1}{2} + \frac{1}{2 \cdot 3 \cdot 2^3} + \frac{1 \cdot 3}{2 \cdot 4 \cdot 5 \cdot 2^5} + \cdots$$

Sharp's series:
$$\tfrac{\pi}{6} = \frac{1}{\sqrt{3}}\left(1 - \frac{1}{3^1 \cdot 3} + \frac{1}{3^2 \cdot 5} - \frac{1}{3^3 \cdot 7} + \cdots\right)$$

Euler's series:
$$\tfrac{\pi^2}{6} = \tfrac{1}{1^2} + \tfrac{1}{2^2} + \tfrac{1}{3^2} + \tfrac{1}{4^2} + \tfrac{1}{5^2} + \cdots$$
$$\tfrac{\pi^2}{8} = \tfrac{1}{1^2} + \tfrac{1}{3^2} + \tfrac{1}{5^2} + \tfrac{1}{7^2} + \tfrac{1}{9^2} + \cdots$$
$$\tfrac{\pi^2}{12} = \tfrac{1}{1^2} - \tfrac{1}{2^2} + \tfrac{1}{3^2} - \tfrac{1}{4^2} + \tfrac{1}{5^2} - \cdots$$

Partial Fractions

Let $N(x)$ and $D(x)$ be polynomial functions of x. We can break down $N(x)/D(x)$ using partial fraction expansion. First, if the degree of N is greater than or equal to the degree of D, divide N by D, obtaining
$$\frac{N(x)}{D(x)} = Q(x) + \frac{N'(x)}{D(x)},$$
where the degree of N' is less than that of D. Second, factor $D(x)$. Use the following rules: For a non-repeated factor:
$$\frac{N(x)}{(x-a)D(x)} = \frac{A}{x-a} + \frac{N'(x)}{D(x)},$$
where
$$A = \left[\frac{N(x)}{D(x)}\right]_{x=a}.$$
For a repeated factor:
$$\frac{N(x)}{(x-a)^m D(x)} = \sum_{k=0}^{m-1} \frac{A_k}{(x-a)^{m-k}} + \frac{N'(x)}{D(x)},$$
where
$$A_k = \frac{1}{k!}\left[\frac{d^k}{dx^k}\left(\frac{N(x)}{D(x)}\right)\right]_{x=a}.$$

The reasonable man adapts himself to the world; the unreasonable persists in trying to adapt the world to himself. Therefore all progress depends on the unreasonable.
– George Bernard Shaw

Calculus

Derivatives:

1. $\dfrac{d(cu)}{dx} = c\dfrac{du}{dx}$, **2.** $\dfrac{d(u+v)}{dx} = \dfrac{du}{dx} + \dfrac{dv}{dx}$, **3.** $\dfrac{d(uv)}{dx} = u\dfrac{dv}{dx} + v\dfrac{du}{dx}$,

4. $\dfrac{d(u^n)}{dx} = nu^{n-1}\dfrac{du}{dx}$, **5.** $\dfrac{d(u/v)}{dx} = \dfrac{v\left(\frac{du}{dx}\right) - u\left(\frac{dv}{dx}\right)}{v^2}$, **6.** $\dfrac{d(e^{cu})}{dx} = ce^{cu}\dfrac{du}{dx}$,

7. $\dfrac{d(c^u)}{dx} = (\ln c)c^u\dfrac{du}{dx}$, **8.** $\dfrac{d(\ln u)}{dx} = \dfrac{1}{u}\dfrac{du}{dx}$,

9. $\dfrac{d(\sin u)}{dx} = \cos u\dfrac{du}{dx}$, **10.** $\dfrac{d(\cos u)}{dx} = -\sin u\dfrac{du}{dx}$,

11. $\dfrac{d(\tan u)}{dx} = \sec^2 u\dfrac{du}{dx}$, **12.** $\dfrac{d(\cot u)}{dx} = \csc^2 u\dfrac{du}{dx}$,

13. $\dfrac{d(\sec u)}{dx} = \tan u \, \sec u\dfrac{du}{dx}$, **14.** $\dfrac{d(\csc u)}{dx} = -\cot u \, \csc u\dfrac{du}{dx}$,

15. $\dfrac{d(\arcsin u)}{dx} = \dfrac{1}{\sqrt{1-u^2}}\dfrac{du}{dx}$, **16.** $\dfrac{d(\arccos u)}{dx} = \dfrac{-1}{\sqrt{1-u^2}}\dfrac{du}{dx}$,

17. $\dfrac{d(\arctan u)}{dx} = \dfrac{1}{1-u^2}\dfrac{du}{dx}$, **18.** $\dfrac{d(\operatorname{arccot} u)}{dx} = \dfrac{-1}{1-u^2}\dfrac{du}{dx}$,

19. $\dfrac{d(\operatorname{arcsec} u)}{dx} = \dfrac{1}{u\sqrt{1-u^2}}\dfrac{du}{dx}$, **20.** $\dfrac{d(\operatorname{arccsc} u)}{dx} = \dfrac{-1}{u\sqrt{1-u^2}}\dfrac{du}{dx}$,

21. $\dfrac{d(\sinh u)}{dx} = \cosh u\dfrac{du}{dx}$, **22.** $\dfrac{d(\cosh u)}{dx} = \sinh u\dfrac{du}{dx}$,

23. $\dfrac{d(\tanh u)}{dx} = \operatorname{sech}^2 u\dfrac{du}{dx}$, **24.** $\dfrac{d(\coth u)}{dx} = -\operatorname{csch}^2 u\dfrac{du}{dx}$,

25. $\dfrac{d(\operatorname{sech} u)}{dx} = -\operatorname{sech} u \, \tanh u\dfrac{du}{dx}$, **26.** $\dfrac{d(\operatorname{csch} u)}{dx} = -\operatorname{csch} u \, \coth u\dfrac{du}{dx}$,

27. $\dfrac{d(\operatorname{arcsinh} u)}{dx} = \dfrac{1}{\sqrt{1+u^2}}\dfrac{du}{dx}$, **28.** $\dfrac{d(\operatorname{arccosh} u)}{dx} = \dfrac{1}{\sqrt{u^2-1}}\dfrac{du}{dx}$,

29. $\dfrac{d(\operatorname{arctanh} u)}{dx} = \dfrac{1}{1-u^2}\dfrac{du}{dx}$, **30.** $\dfrac{d(\operatorname{arccoth} u)}{dx} = \dfrac{1}{u^2-1}\dfrac{du}{dx}$,

31. $\dfrac{d(\operatorname{arcsech} u)}{dx} = \dfrac{-1}{u\sqrt{1-u^2}}\dfrac{du}{dx}$, **32.** $\dfrac{d(\operatorname{arccsch} u)}{dx} = \dfrac{-1}{|u|\sqrt{1+u^2}}\dfrac{du}{dx}$.

Integrals:

1. $\displaystyle\int cu\,dx = c\int u\,dx$, **2.** $\displaystyle\int (u+v)\,dx = \int u\,dx + \int v\,dx$,

3. $\displaystyle\int x^n\,dx = \frac{1}{n+1}x^{n+1}, \quad n \neq -1$, **4.** $\displaystyle\int \frac{1}{x}dx = \ln x$, **5.** $\displaystyle\int e^x\,dx = e^x$,

6. $\displaystyle\int \frac{dx}{1+x^2} = \arctan x$, **7.** $\displaystyle\int u\frac{dv}{dx}dx = uv - \int v\frac{du}{dx}dx$,

8. $\displaystyle\int \sin x\,dx = -\cos x$, **9.** $\displaystyle\int \cos x\,dx = \sin x$,

10. $\displaystyle\int \tan x\,dx = -\ln|\cos x|$, **11.** $\displaystyle\int \cot x\,dx = \ln|\cos x|$,

12. $\displaystyle\int \sec x\,dx = \ln|\sec x + \tan x|$, **13.** $\displaystyle\int \csc x\,dx = \ln|\csc x + \cot x|$,

14. $\displaystyle\int \arcsin\tfrac{x}{a}dx = \arcsin\tfrac{x}{a} + \sqrt{a^2-x^2}, \quad a > 0$,

Theoretical Computer Science Cheat Sheet

Calculus Cont.

15. $\int \arccos \frac{x}{a} dx = \arccos \frac{x}{a} - \sqrt{a^2 - x^2}, \quad a > 0,$

16. $\int \arctan \frac{x}{a} dx = x \arctan \frac{x}{a} - \frac{a}{2} \ln(a^2 + x^2), \quad a > 0,$

17. $\int \sin^2(ax) dx = \frac{1}{2a}\big(ax - \sin(ax)\cos(ax)\big),$

18. $\int \cos^2(ax) dx = \frac{1}{2a}\big(ax + \sin(ax)\cos(ax)\big),$

19. $\int \sec^2 x\, dx = \tan x,$

20. $\int \csc^2 x\, dx = -\cot x,$

21. $\int \sin^n x\, dx = -\frac{\sin^{n-1} x \cos x}{n} + \frac{n-1}{n} \int \sin^{n-2} x\, dx,$

22. $\int \cos^n x\, dx = \frac{\cos^{n-1} x \sin x}{n} + \frac{n-1}{n} \int \cos^{n-2} x\, dx,$

23. $\int \tan^n x\, dx = \frac{\tan^{n-1} x}{n-1} - \int \tan^{n-2} x\, dx, \quad n \neq 1,$

24. $\int \cot^n x\, dx = -\frac{\cot^{n-1} x}{n-1} - \int \cot^{n-2} x\, dx, \quad n \neq 1,$

25. $\int \sec^n x\, dx = \frac{\tan x \sec^{n-1} x}{n-1} + \frac{n-2}{n-1} \int \sec^{n-2} x\, dx, \quad n \neq 1,$

26. $\int \csc^n x\, dx = -\frac{\cot x \csc^{n-1} x}{n-1} + \frac{n-2}{n-1} \int \csc^{n-2} x\, dx, \quad n \neq 1,$

27. $\int \sinh x\, dx = \cosh x,$

28. $\int \cosh x\, dx = \sinh x,$

29. $\int \tanh x\, dx = \ln |\cosh x|,$

30. $\int \coth x\, dx = \ln |\sinh x|,$

31. $\int \operatorname{sech} x\, dx = \arctan \sinh x,$

32. $\int \operatorname{csch} x\, dx = \ln \left|\tanh \frac{x}{2}\right|,$

33. $\int \sinh^2 x\, dx = \frac{1}{4}\sinh(2x) - \frac{1}{2}x,$

34. $\int \cosh^2 x\, dx = \frac{1}{4}\sinh(2x) + \frac{1}{2}x,$

35. $\int \operatorname{sech}^2 x\, dx = \tanh x,$

36. $\int \operatorname{arcsinh} \frac{x}{a} dx = x \operatorname{arcsinh} \frac{x}{a} - \sqrt{x^2 + a^2}, \quad a > 0,$

37. $\int \operatorname{arctanh} \frac{x}{a} dx = x \operatorname{arctanh} \frac{x}{a} + \frac{a}{2} \ln |a^2 - x^2|,$

38. $$\int \operatorname{arccosh} \frac{x}{a} dx = \begin{cases} x \operatorname{arccosh} \dfrac{x}{a} - \sqrt{x^2 + a^2}, & \text{if } \operatorname{arccosh} \frac{x}{a} > 0 \text{ and } a > 0, \\ x \operatorname{arccosh} \dfrac{x}{a} + \sqrt{x^2 + a^2}, & \text{if } \operatorname{arccosh} \frac{x}{a} < 0 \text{ and } a > 0, \end{cases}$$

39. $\int \frac{dx}{\sqrt{a^2 + x^2}} = \ln\left(x + \sqrt{a^2 + x^2}\right), \quad a > 0,$

40. $\int \frac{dx}{a^2 + x^2} = \frac{1}{a} \arctan \frac{x}{a}, \quad a > 0,$

41. $\int \sqrt{a^2 - x^2}\, dx = \frac{x}{2}\sqrt{a^2 - x^2} + \frac{a^2}{2} \arcsin \frac{x}{a}, \quad a > 0,$

42. $\int (a^2 - x^2)^{3/2} dx = \frac{x}{8}(5a^2 - 2x^2)\sqrt{a^2 - x^2} + \frac{3a^4}{8} \arcsin \frac{x}{a}, \quad a > 0,$

43. $\int \frac{dx}{\sqrt{a^2 - x^2}} = \arcsin \frac{x}{a}, \quad a > 0,$

44. $\int \frac{dx}{a^2 - x^2} = \frac{1}{2a} \ln \left|\frac{a + x}{a - x}\right|,$

45. $\int \frac{dx}{(a^2 - x^2)^{3/2}} = \frac{x}{a^2\sqrt{a^2 - x^2}},$

46. $\int \sqrt{a^2 \pm x^2}\, dx = \frac{x}{2}\sqrt{a^2 \pm x^2} \pm \frac{a^2}{2} \ln \left|x + \sqrt{a^2 \pm x^2}\right|,$

47. $\int \frac{dx}{\sqrt{x^2 - a^2}} = \ln \left|x + \sqrt{x^2 - a^2}\right|, \quad a > 0,$

48. $\int \frac{dx}{ax^2 + bx} = \frac{1}{a} \ln \left|\frac{x}{a + bx}\right|,$

49. $\int x\sqrt{a + bx}\, dx = \frac{2(3bx - 2a)(a + bx)^{3/2}}{15b^2},$

50. $\int \frac{\sqrt{a + bx}}{x} dx = 2\sqrt{a + bx} + a \int \frac{1}{x\sqrt{a + bx}} dx,$

51. $\int \frac{x}{\sqrt{a + bx}} dx = \frac{1}{\sqrt{2}} \ln \left|\frac{\sqrt{a + bx} - \sqrt{a}}{\sqrt{a + bx} + \sqrt{a}}\right|, \quad a > 0,$

52. $\int \frac{\sqrt{a^2 - x^2}}{x} dx = \sqrt{a^2 - x^2} - a \ln \left|\frac{a + \sqrt{a^2 - x^2}}{x}\right|,$

53. $\int x\sqrt{a^2 - x^2}\, dx = -\frac{1}{3}(a^2 - x^2)^{3/2},$

54. $\int x^2\sqrt{a^2 - x^2}\, dx = \frac{x}{8}(2x^2 - a^2)\sqrt{a^2 - x^2} + \frac{a^4}{8} \arcsin \frac{x}{a}, \quad a > 0,$

55. $\int \frac{dx}{\sqrt{a^2 - x^2}} = -\frac{1}{a} \ln \left|\frac{a + \sqrt{a^2 - x^2}}{x}\right|,$

56. $\int \frac{x\, dx}{\sqrt{a^2 - x^2}} = -\sqrt{a^2 - x^2},$

57. $\int \frac{x^2\, dx}{\sqrt{a^2 - x^2}} = -\frac{x}{2}\sqrt{a^2 - x^2} + \frac{a^2}{2} \arcsin \frac{x}{a}, \quad a > 0,$

58. $\int \frac{\sqrt{a^2 + x^2}}{x} dx = \sqrt{a^2 + x^2} - a \ln \left|\frac{a + \sqrt{a^2 + x^2}}{x}\right|,$

59. $\int \frac{\sqrt{x^2 - a^2}}{x} dx = \sqrt{x^2 - a^2} - a \arccos \frac{a}{|x|}, \quad a > 0,$

60. $\int x\sqrt{x^2 \pm a^2}\, dx = \frac{1}{3}(x^2 \pm a^2)^{3/2},$

61. $\int \frac{dx}{x\sqrt{x^2 + a^2}} = \frac{1}{a} \ln \left|\frac{x}{a + \sqrt{a^2 + x^2}}\right|,$

Theoretical Computer Science Cheat Sheet

Calculus Cont.

62. $\displaystyle\int \frac{dx}{x\sqrt{x^2-a^2}} = \frac{1}{a}\arccos\frac{a}{|x|}, \quad a>0,$ **63.** $\displaystyle\int \frac{dx}{x^2\sqrt{x^2\pm a^2}} = \mp\frac{\sqrt{x^2\pm a^2}}{a^2x},$

64. $\displaystyle\int \frac{x\,dx}{\sqrt{x^2\pm a^2}} = \sqrt{x^2\pm a^2},$ **65.** $\displaystyle\int \frac{\sqrt{x^2\pm a^2}}{x^4}\,dx = \mp\frac{(x^2+a^2)^{3/2}}{3a^2x^3},$

66. $$\int \frac{dx}{ax^2+bx+c} = \begin{cases} \dfrac{1}{\sqrt{b^2-4ac}}\ln\left|\dfrac{2ax+b-\sqrt{b^2-4ac}}{2ax+b+\sqrt{b^2-4ac}}\right|, & \text{if } b^2>4ac,\\[2ex] \dfrac{2}{\sqrt{4ac-b^2}}\arctan\dfrac{2ax+b}{\sqrt{4ac-b^2}}, & \text{if } b^2<4ac,\end{cases}$$

67. $$\int \frac{dx}{\sqrt{ax^2+bx+c}} = \begin{cases} \dfrac{1}{\sqrt{a}}\ln\left|2ax+b+2\sqrt{a}\sqrt{ax^2+bx+c}\right|, & \text{if } a>0,\\[2ex] \dfrac{1}{\sqrt{-a}}\arcsin\dfrac{-2ax-b}{\sqrt{b^2-4ac}}, & \text{if } a<0,\end{cases}$$

68. $$\int \sqrt{ax^2+bx+c}\,dx = \frac{2ax+b}{4a}\sqrt{ax^2+bx+c} + \frac{4ax-b^2}{8a}\int\frac{dx}{\sqrt{ax^2+bx+c}},$$

69. $$\int \frac{x\,dx}{\sqrt{ax^2+bx+c}} = \frac{\sqrt{ax^2+bx+c}}{a} - \frac{b}{2a}\int\frac{dx}{\sqrt{ax^2+bx+c}},$$

70. $$\int \frac{dx}{x\sqrt{ax^2+bx+c}} = \begin{cases} \dfrac{-1}{\sqrt{c}}\ln\left|\dfrac{2\sqrt{c}\sqrt{ax^2+bx+c}+bx+2c}{x}\right|, & \text{if } c>0,\\[2ex] \dfrac{1}{\sqrt{-c}}\arcsin\dfrac{bx+2c}{|x|\sqrt{b^2-4ac}}, & \text{if } c<0,\end{cases}$$

71. $$\int x^3\sqrt{x^2+a^2}\,dx = (\tfrac{1}{3}x^2-\tfrac{2}{15}a^2)(x^2+a^2)^{3/2},$$

72. $$\int x^n\sin(ax)\,dx = -\tfrac{1}{a}x^n\cos(ax) + \tfrac{n}{a}\int x^{n-1}\cos(ax)\,dx,$$

73. $$\int x^n\cos(ax)\,dx = \tfrac{1}{a}x^n\sin(ax) - \tfrac{n}{a}\int x^{n-1}\sin(ax)\,dx,$$

74. $$\int x^n e^{ax}\,dx = \frac{x^ne^{ax}}{a} - \tfrac{n}{a}\int x^{n-1}e^{ax}\,dx,$$

75. $$\int x^n\ln(ax)\,dx = x^{n+1}\left(\frac{\ln(ax)}{n+1} - \frac{1}{(n+1)^2}\right),$$

76. $$\int x^n(\ln ax)^m\,dx = \frac{x^{n+1}}{n+1}(\ln ax)^m - \frac{m}{n+1}\int x^n(\ln ax)^{m-1}\,dx.$$

$$\begin{array}{rcl} x^1 = & x^{\underline{1}} & = \quad x^{\overline{1}}\\ x^2 = & x^{\underline{2}}+x^{\underline{1}} & = \quad x^{\overline{2}}-x^{\overline{1}}\\ x^3 = & x^{\underline{3}}+3x^{\underline{2}}+x^{\underline{1}} & = \quad x^{\overline{3}}-3x^{\overline{2}}+x^{\overline{1}}\\ x^4 = & x^{\underline{4}}+6x^{\underline{3}}+7x^{\underline{2}}+x^{\underline{1}} & = \quad x^{\overline{4}}-6x^{\overline{3}}+7x^{\overline{2}}-x^{\overline{1}}\\ x^5 = & x^{\underline{5}}+15x^{\underline{4}}+25x^{\underline{3}}+10x^{\underline{2}}+x^{\underline{1}} & = \quad x^{\overline{5}}-15x^{\overline{4}}+25x^{\overline{3}}-10x^{\overline{2}}+x^{\overline{1}} \end{array}$$

$$\begin{array}{rcrc} x^{\overline{1}} = & x^1 & x^{\underline{1}} = & x^1\\ x^{\overline{2}} = & x^2+x^1 & x^{\underline{2}} = & x^2-x^1\\ x^{\overline{3}} = & x^3+3x^2+2x^1 & x^{\underline{3}} = & x^3-3x^2+2x^1\\ x^{\overline{4}} = & x^4+6x^3+11x^2+6x^1 & x^{\underline{4}} = & x^4-6x^3+11x^2-6x^1\\ x^{\overline{5}} = & x^5+10x^4+35x^3+50x^2+24x^1 & x^{\underline{5}} = & x^5-10x^4+35x^3-50x^2+24x^1 \end{array}$$

Finite Calculus

Difference, shift operators:

$$\Delta f(x) = f(x+1) - f(x),$$
$$\mathrm{E}\, f(x) = f(x+1).$$

Fundamental Theorem:

$$f(x) = \Delta F(x) \Leftrightarrow \sum f(x)\delta x = F(x) + C.$$
$$\sum_a^b f(x)\delta x = \sum_{i=a}^{b-1} f(i).$$

Differences:

$\Delta(cu) = c\Delta u, \qquad \Delta(u+v) = \Delta u + \Delta v,$

$\Delta(uv) = u\Delta v + \mathrm{E}\, v\Delta u,$

$\Delta(x^{\underline{n}}) = nx^{\underline{n-1}},$

$\Delta(H_x) = x^{\underline{-1}}, \qquad \Delta(2^x) = 2^x,$

$\Delta(c^x) = (c-1)c^x, \qquad \Delta\binom{x}{m} = \binom{x}{m-1}.$

Sums:

$\sum cu\,\delta x = c\sum u\,\delta x,$

$\sum (u+v)\,\delta x = \sum u\,\delta x + \sum v\,\delta x,$

$\sum u\Delta v\,\delta x = uv - \sum \mathrm{E}\, v\Delta u\,\delta x,$

$\sum x^{\underline{n}}\,\delta x = \frac{x^{\underline{n+1}}}{m+1}, \qquad \sum x^{\underline{-1}}\,\delta x = H_x,$

$\sum c^x\,\delta x = \frac{c^x}{c-1}, \qquad \sum\binom{x}{m}\,\delta x = \binom{x}{m+1}.$

Falling Factorial Powers:

$x^{\underline{n}} = x(x-1)\cdots(x-n+1), \quad n>0,$

$x^{\underline{0}} = 1,$

$x^{\underline{n}} = \dfrac{1}{(x+1)\cdots(x+|n|)}, \quad n<0,$

$x^{\underline{n+m}} = x^{\underline{m}}(x-m)^{\underline{n}}.$

Rising Factorial Powers:

$x^{\overline{n}} = x(x+1)\cdots(x+n-1), \quad n>0,$

$x^{\overline{0}} = 1,$

$x^{\overline{n}} = \dfrac{1}{(x-1)\cdots(x-|n|)}, \quad n<0,$

$x^{\overline{n+m}} = x^{\overline{m}}(x+m)^{\overline{n}}.$

Conversion:

$x^{\underline{n}} = (-1)^n(-x)^{\overline{n}} = (x-n+1)^{\overline{n}}$

$\quad = 1/(x+1)^{\overline{-n}},$

$x^{\overline{n}} = (-1)^n(-x)^{\underline{n}} = (x+n-1)^{\underline{n}}$

$\quad = 1/(x-1)^{\underline{-n}},$

$$x^n = \sum_{k=1}^n \left\{ n \atop k \right\} x^{\underline{k}} = \sum_{k=1}^n \left\{ n \atop k \right\} (-1)^{n-k} x^{\overline{k}},$$

$$x^{\underline{n}} = \sum_{k=1}^n \begin{bmatrix} n \\ k \end{bmatrix} (-1)^{n-k} x^k,$$

$$x^{\overline{n}} = \sum_{k=1}^n \begin{bmatrix} n \\ k \end{bmatrix} x^k.$$

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:

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Theoretical Computer Science Cheat Sheet

Series

Expansions:

$$\frac{1}{(1-x)^{n+1}} \ln \frac{1}{1-x} = \sum_{i=0}^{\infty} (H_{n+i} - H_n) \binom{n+i}{i} x^i,$$

$$\left(\frac{1}{x}\right)^{\overline{-n}} = \sum_{i=0}^{\infty} \left\{ {i \atop n} \right\} x^i,$$

$$x^{\overline{n}} = \sum_{i=0}^{\infty} \left[{n \atop i} \right] x^i,$$

$$(e^x - 1)^n = \sum_{i=0}^{\infty} \left\{ {i \atop n} \right\} \frac{n! x^i}{i!},$$

$$\left(\ln \frac{1}{1-x}\right)^n = \sum_{i=0}^{\infty} \left[{i \atop n} \right] \frac{n! x^i}{i!},$$

$$x \cot x = \sum_{i=0}^{\infty} \frac{(-4)^i B_{2i} x^{2i}}{(2i)!},$$

$$\tan x = \sum_{i=1}^{\infty} (-1)^{i-1} \frac{2^{2i}(2^{2i}-1) B_{2i} x^{2i-1}}{(2i)!},$$

$$\zeta(x) = \sum_{i=1}^{\infty} \frac{1}{i^x},$$

$$\frac{1}{\zeta(x)} = \sum_{i=1}^{\infty} \frac{\mu(i)}{i^x},$$

$$\frac{\zeta(x-1)}{\zeta(x)} = \sum_{i=1}^{\infty} \frac{\phi(i)}{i^x},$$

$$\zeta(x) = \prod_p \frac{1}{1-p^{-x}},$$

$$\zeta^2(x) = \sum_{i=1}^{\infty} \frac{d(i)}{x^i} \quad \text{where } d(n) = \textstyle\sum_{d|n} 1,$$

$$\zeta(x)\zeta(x-1) = \sum_{i=1}^{\infty} \frac{S(i)}{x^i} \quad \text{where } S(n) = \textstyle\sum_{d|n} d,$$

$$\zeta(2n) = \frac{2^{2n-1}|B_{2n}|}{(2n)!} \pi^{2n}, \quad n \in \mathbb{N},$$

$$\frac{x}{\sin x} = \sum_{i=0}^{\infty} (-1)^{i-1} \frac{(4^i - 2) B_{2i} x^{2i}}{(2i)!},$$

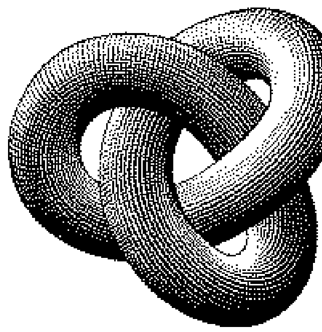
$$\left(\frac{1-\sqrt{1-4x}}{2x}\right)^n = \sum_{i=0}^{\infty} \frac{n(2i+n-1)!}{i!(n+i)!} x^i,$$

$$e^x \sin x = \sum_{i=1}^{\infty} \frac{2^{i/2} \sin \frac{i\pi}{4}}{i!} x^i,$$

$$\sqrt{\frac{1-\sqrt{1-x}}{x}} = \sum_{i=0}^{\infty} \frac{(4i)!}{16^i \sqrt{2}(2i)!(2i+1)!} x^i,$$

$$\left(\frac{\arcsin x}{x}\right)^2 = \sum_{i=0}^{\infty} \frac{4^i i!^2}{(i+1)(2i+1)!} x^{2i}.$$

Escher's Knot



Stieltjes Integration

If G is continuous in the interval $[a,b]$ and F is nondecreasing then

$$\int_a^b G(x)\, dF(x)$$

exists. If $a \le b \le c$ then

$$\int_a^c G(x)\, dF(x) = \int_a^b G(x)\, dF(x) + \int_b^c G(x)\, dF(x).$$

If the integrals involved exist

$$\int_a^b \big(G(x) + H(x)\big)\, dF(x) = \int_a^b G(x)\, dF(x) + \int_a^b H(x)\, dF(x),$$

$$\int_a^b G(x)\, d\big(F(x) + H(x)\big) = \int_a^b G(x)\, dF(x) + \int_a^b G(x)\, dH(x),$$

$$\int_a^b c \cdot G(x)\, dF(x) = \int_a^b G(x)\, d\big(c \cdot F(x)\big) = c \int_a^b G(x)\, dF(x),$$

$$\int_a^b G(x)\, dF(x) = G(b)F(b) - G(a)F(a) - \int_a^b F(x)\, dG(x).$$

If the integrals involved exist, and F possesses a derivative F' at every point in $[a,b]$ then

$$\int_a^b G(x)\, dF(x) = \int_a^b G(x) F'(x)\, dx.$$

Cramer's Rule

If we have equations:

$$\begin{aligned} a_{1,1}x_1 + a_{1,2}x_2 + \cdots + a_{1,n}x_n &= b_1 \\ a_{2,1}x_1 + a_{2,2}x_2 + \cdots + a_{2,n}x_n &= b_2 \\ \vdots \qquad \vdots \qquad \qquad \vdots \\ a_{n,1}x_1 + a_{n,2}x_2 + \cdots + a_{n,n}x_n &= b_n \end{aligned}$$

Let $A = (a_{i,j})$ and B be the column matrix (b_i). Then there is a unique solution iff $\det A \neq 0$. Let A_i be A with column i replaced by B. Then

$$x_i = \frac{\det A_i}{\det A}.$$

00	47	18	76	29	93	85	34	61	52
86	11	57	28	70	39	94	45	02	63
95	80	22	67	38	71	49	56	13	04
59	96	81	33	07	48	72	60	24	15
73	69	90	82	44	17	58	01	35	26
68	74	09	91	83	55	27	12	46	30
37	08	75	19	92	84	66	23	50	41
14	25	36	40	51	62	03	77	88	99
21	32	43	54	65	06	10	89	97	78
42	53	64	05	16	20	31	98	79	87

Fibonacci Numbers

$1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, \ldots$

Definitions:

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

$$F_{-i} = (-1)^{i-1} F_i,$$

$$F_i = \frac{1}{\sqrt{5}} \left(\phi^i - \hat{\phi}^i\right),$$

Cassini's identity: for $i > 0$:

$$F_{i+1}F_{i-1} - F_i^2 = (-1)^i.$$

Additive rule:

$$F_{n+k} = F_k F_{n+1} + F_{k-1} F_n,$$

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

Calculation by matrices:

$$\begin{pmatrix} F_{n-2} & F_{n-1} \\ F_{n-1} & F_n \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}^n.$$

The Fibonacci number system: Every integer n has a unique representation

$$n = F_{k_1} + F_{k_2} + \cdots + F_{k_m},$$

where $k_i \ge k_{i+1} + 2$ for all i, $1 \le i < m$ and $k_m \ge 2$.

Improvement makes strait roads, but the crooked roads without Improvement, are roads of Genius.
– William Blake (The Marriage of Heaven and Hell)