Modular Arithmetic

- Addition: O(n)
- Multiplication: O(n^2) *(naive)*
- Multiplication: O(nlogn) *(FFT)*
- Euclid's Rule: gcd(x, y) = gcd(x mod y, y)
- # of bits in x^y = y$\log_2$x $\leq$ n$\cdot 2^n$
- $\frac{n}{2}^{\frac{n}{2}} \leq n! \leq n^n$
- f: S $\rightarrow$ T is 1-to-1 (injective) & onto (surjective) $\Rightarrow$ $|S| = |T|$
- f: S $\rightarrow$ T is 1-to-1 (injective) $\Rightarrow$ $|T| \geq |S|$
- $\sum_{i=0}^{\infty} r^i = \frac{1}{1-r}$, if r < 1
- $\sum_{i=0}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}$
- $\sum_{i=0}^{n} \frac{1}{i}$ = O($\log_2$n)

Extended Euclid's GCD(x,y)

O(n^3); gcd(x,y) = d = xi + yb; x $\geq$ y; # mod x

```
ext-gcd(x,y):
  if y == 0:  return (x, 1, 0)
  else:
    (d, a, b) = ext-gcd(y, x mod y)
    return (d, b, a-x/y·b)
```

#	X	Y	X/Y	X%Y	#	d	a	b
1.	26	15	1	11	6.	1	1	0
2.	15	11	1	4	5.	1	0	1-(3*0)
3.	11	4	2	3	4.	1	1	0-(1*1)
4.	4	3	1	1	3.	1	-1	1-(2*-1)
5.	3	1	3	0	2.	1	3	-1-(1*3)
6.	1	0			1.	1	-4	3-(1*-4)

Fermat's Little Theorem

if p is prime, then $\forall$ 1 $\leq$ a < p
a^{p-1} = 1 mod p

Proof: *Start by listing first p-1 positive multiples of a:*
$S = \{a, 2a, 3a, \cdots (p-1)a\}$
Suppose that ra and sa are the same mod p, $\Rightarrow$ r = s mod p
$\therefore$ set S of p-1 multiples of a are distinct and nonzero, that is, they must be congruent to 1, 2, 3, $\cdots$ p-1 after being sorted.
Multiply all congruences together and we find
$a \cdot 2a \cdot 3a \cdots (p-1) \cdot a = 1 \cdot 2 \cdot 3 \cdots (p-1)$ (mod p) or better,
$a^{(p-1)}(p-1)! = (p-1)!$ mod p. Divide both side by (p-1)! ■

Primality Testing

$$any\ a \rightarrow a^{N-1} = 1\ mod\ N? \begin{cases} yes \Rightarrow \text{"prime"} \\ no \Rightarrow composite \end{cases}$$

if N is not prime a^{N-1} = 1 mod N $\leq$ half values of a < N

Lagrange's Prime Theorem

Let π(x) be the # of primes *leq* x, then
$\pi(x) \approx \frac{x}{ln(x)}$, or more precisely $\lim_{x\rightarrow\infty} \frac{\pi(x)}{(\frac{x}{ln(x)})} = 1$

Modular Exponentiation

x^y mod N $\rightarrow$ start with repeated squaring mod N
x mod N $\rightarrow$ x^2 mod N $\rightarrow$ $(x^2)^2 \cdots x^{log_2 y}$ mod N
each step takes O($\log^2 N$) times to compute and there are $\log_2$y steps, $\therefore \in$ O(n^3),
where n is the # of bits in N

Formal Limit Proof

$$lim_{n\rightarrow\infty} \frac{f(n)}{g(n)} \begin{cases} \geq\ 0\ (\infty) \Rightarrow\ f(n)\ \in\ \Omega(g(n)) \\ <\ \infty\ (0) \Rightarrow\ f(n)\ \in\ O(g(n)) \\ =c_{|0<c<\infty}\ \Rightarrow\ f(n)\ \in\ \Theta(g(n)) \end{cases}$$

Logarithm Tricks

$\log_b x^p = p log_b x$
$\frac{ln(x)}{ln(m)} = \log_m$x
$x^{log_b y} = y^{log_b x}$

Complexity

- $f \in$ O(g) if $f \leq$ c$\cdot g$
- $f \in \Omega(g)$ if $f \geq$ c$\cdot g$
- $f \in \Theta(g)$ if $f \in$ O(g) & $\Omega(g)$

Hierarchy:
- Exponential
- Polynomial
- Logarithmic
- Constant

Master's Theorem

T(n) = aT($\frac{n}{b}$) + O(n^d), if a >0, b>1, $d \geq 0$

$$T(n) = \begin{cases} O(n^d) \text{ if d} > \log_b a \\ O(n^d log_b n) \text{ if d} = \log_b a \\ O(n^{log_b n}) \text{ if d} < \log_b a \end{cases}$$

Volker Strassen

faster matrix multiplication...

$$X = \begin{bmatrix} A & B \\ C & D \end{bmatrix} \times Y = \begin{bmatrix} E & F \\ G & H \end{bmatrix} = \begin{bmatrix} AE+BG & AF+BH \\ CE+DG & CF+DH \end{bmatrix}$$

$\in$ O(n^3) with recurrence T(n)=8T($\frac{n}{2}$)+O(n^2)
but thanks to Stassen...

$$XY = \begin{bmatrix} P_5+P_4-P_2+P_6 & P_1+P_2 \\ P_3+P_4 & P_1+P_5-P_3+P_7 \end{bmatrix}$$

P_1 = A(F-H) P_2 = (A+B)H P_3 = (C+D)E P_4 = D(G-E)
P_5 = (A+D)(E+H) P_6 = (B-D)(G+H) P_7 = (A-C)(E+F)
$\in$ O($n^{log_2 7}$) $\approx$ O($n^{2.81}$) with recurrence T(n)=7T($\frac{n}{2}$)+O(n^2)

Polynomial Multiplication

$A(x) = a_0 + a_1 x + \cdots + a_d x^d \quad B(x) = b_0 + b_1 x + \cdots + b_d x^d$
$C(x)$=$A(x)\times B(x) = a_0 b_k + a_1 b_{k-1} + \cdots + a_k b_0 = \sum_{i=0}^{k} a_i b_{k-i}$

Fast Fourier Transform

complex n^{th} roots of unity are given by $\omega = e^{\frac{2\pi i}{n}}, \omega^2, \omega^3, \cdots$
< values > = FFT(< coefficients >, ω)
< coefficients > = $\frac{1}{n}$ FFT(< values >, ω^{-1}) $\in$ O(nlogn)

Vandermonde Matrix, $M_n(\omega)$=

$$\begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega & \omega^2 & \cdots & \omega^{n-1} \\ 1 & \omega^2 & \omega^4 & \cdots & \omega^{2(n-1)} \\ & & & & \\ 1 & \cdots & \cdots & \ddots & \omega^{j(n-1)} \\ 1 & \omega^{n-1} & \omega^{2(n-1)} & \cdots & \omega^{(n-1)(n-1)} \end{bmatrix} where (j,k)^{th} entry\ is\ \omega^{jk}$$

Graphs

- ***graph*** – set of nodes & edges between select nodes
- ***tree*** – a connected graph with no cycles
- ***tree edge*** – part of DFS forest
- ***forward edge*** – edge leading from node $\rightarrow$ non-child descendant
- ***back edge*** – edge leading back to previously visited node
- ***cross edge*** – edge leading to neither descendant nor ancestor

Given an Edge (u,v):
- ***tree/forward edge****: pre(u) < pre(v) < post(v) < post(u)*
- ***back edge****: pre(v) < pre(u) < post(u) < post(v)*
- ***cross edge****: pre(v) < post(v) < pre(u) < post(u)*

Properties:
- a tree on n nodes has n-1 edges
- any connected undirected graph with |E| = |V|-1 edges is a tree
- a directed graph has a cycle iff its DFS reveals a back edge
- every DAG has at least 1 source & 1 sink
- in a DAG, every edge leads to a vertex with lower post #
- every directed graph is a DAG of its SCCs
- acyclic, linearizability, & absence of back edges are all the same property
- any path of DAG, vertices appear in increasing linearized order (linearize, topological sort DAG by DFS, then visit vertices in sorted order, updating edges out of each)
- if explore starts at u, it will terminate when all nodes reachable from u have been visited
- node receives highest post order in DFS must lie in source SCC
- if C & C' are SCCs & $\exists$ an edge from a node in C $\rightarrow$ C' $\Rightarrow$ highest post order number in C > than C''s highest post #
- min edges to make graph strongly connected with n-sinks & m-sources$\rightarrow$*max(n,m)*

Linearize (topologically sort from earliest $\rightarrow$ latest)
- perform tasks in decreasing order of their post numbers (DFS)
- or find a source, output it, delete it, repeat until empty

Algorithm to Decompose G into SSCs

```
Run DFS on G^R, then run DFS on G, every node it
reaches is in that SCC, pick next vertex to run DFS
from in order of decreasing post #s discovered from
DFS ordering on G^R
```

Shortest/Longest path in a DAG

```
Linearize DAG by DFS, visit vertices in sorted order,
updating edges out of each.  Note for longest paths,
just negate all edge lengths.
```

Depth First Search

discovers what nodes are reachable from a vertex $\in O(|V|+|E|)$

```
explore(G, v):
  v.visit = true
  previsit(v)
  for each edge (v, u) in E:
    if u.visit = false:  explore(G, u)
  postvisit(v)
dfs(G):
  for all v ∈ V: v.visit = false
  for all v ∈ V: if v.visit = false:
explore(G,v)
```

Breadth First Search

$\in O(|V|+|E|)$

```
bfs(G, s):
  for all u ∈ V: dist(u) = ∞
    dist(s) = 0
  Q = [s] (queue containing just s)
  while Q is not empty:
    u = eject(Q)
    for all edges (u,v) ∈ E:
      if dist(v) = ∞:
        inject(Q,v)
        dist(v) = dist(u) + 1
```

Dijkstra's Algorithm

shortest path algorithm (+ edge weights) $\in O((|V|+|E|)log|V|)$

```
dijkstra(G, l, s):
  for all u ∈ V:
    dist(u) = ∞
    prev(u) = nil
  dist(s) = 0
  H = makequeue(V) (using dist-values as keys)
  while H is not empty:
    u = deletemin(H)
    for all edges (u,v) ∈ E:
      if dist(v) > dist(u) + l(u,v):
        dist(v) = dist(u) + l(u,v)
        prev(v) = u
        decreasekey(H,v)
```

Implementation	deletemin	insert/ decreasekey	$\lvert V\rvert \times$ deletemin + $(\lvert V\rvert+\lvert E\rvert)\times$ insert
Array	$O(\lvert V\rvert)$	$O(1)$	$O(\lvert V\rvert^2)$
Binary heap	$O(\log \lvert V\rvert)$	$O(\log \lvert V\rvert)$	$O((\lvert V\rvert+\lvert E\rvert)\log \lvert V\rvert)$
d-ary heap	$O(\frac{d\log \lvert V\rvert}{\log d})$	$O(\frac{\log \lvert V\rvert}{\log d})$	$O((\lvert V\rvert\cdot d+\lvert E\rvert)\frac{\log \lvert V\rvert}{\log d})$
Fibonacci heap	$O(\log \lvert V\rvert)$	$O(1)$ (amortized)	$O(\lvert V\rvert\log \lvert V\rvert+\lvert E\rvert)$

Bellman-Ford Algorithm

shortest path algorithm (+/- edge weights) $\in O((|V|\cdot|E|)$

```
bellman_ford(G, l, s):
  for all u ∈ V:
    dist(u) = ∞
    prev(u) = nil
  dist(s) = 0
  repeat |V| - 1 times:
    for all e ∈ E:
      update(e)
update(u,v):
  min{dist(v), dist(u) + l(u,v)}
```

note: negative cycle exists if any edge distance value is reduced on $|V|^{th}$ *iteration*

Kruskal's Minimum Spanning Tree Algorithm

starts with an empty graph & selects edges from E repeatedly with lightest weight that does not produce a cycle $\in O(|E|log|V|)$*. Uses disjoint sets to determine whether a cycle exists in amortized constant time (see disjoint set data structure).*

Cut Property

Suppose edges X are part of a minimum spanning tree