

Contents

1 Geometry

- 1.1 Convex Hull Algorithm
- 1.2 Delaunay Triangulation
- 1.3 Various Geometry Functions

2 Graphs

- 2.1 Dijkstra's Algorithm
- 2.2 Max Flow (Dinic's Algorithm)
- 2.3 Max Flow (Edmonds-Karp Algorithm)
- 2.4 Eulerian Path
- 2.5 Hopcroft-Karp Algorithm
- 2.6 Lowest Common Ancestor
- 2.7 Strongly Connected Components
- 2.8 Union-Find Set

3 Tree

- 3.1 Cartesian Tree
- 3.2 Segment Tree

4 Math

- 4.1 Extended Euclid's Algorithm
- 4.2 Fast Prime Number Sieve
- 4.3 Fast Fourier Transform
- 4.4 Gauss-Jordan Elimination
- 4.5 GCD and LCM
- 4.6 Modular Linear Equation Solver

5 Strings

- 5.1 Aho-Corasick Algorithm
- 5.2 Knuth-Morris-Pratt Algorithm
- 5.3 Suffix Array

6 Techniques

- 6.1 Various algorithm techniques

1 Geometry

1.1 Convex Hull Algorithm

```
bool compare(PT a,PT b){ return a.y<b.y || (a.y==b.y && a.x<b.x); }
double cross(PT o,PT a, PT b)
{
    return (a.x-o.x)*(b.y-o.y)-(a.y-o.y)*(b.x-o.x);
}

vector<PT> ConvexHull(vector<PT> p) { int n=p.size(); int k=0;
vector<PT> h(2*n);
sort(p.begin(),p.end(),compare);
//build lower hull
for(int i=0;i<n;++i)
{
    while(k>=2 && cross(h[k-2],h[k-1],p[i])<=0) k--;
    h[k++]=p[i];
}
//build top hull
for(int i=n-2,t=k+1;i>=0;--i)
{
    while(k>=t && cross(h[k-2],h[k-1],p[i])<=0) k--;
    h[k++]=p[i];
}
h.resize(k);
return h;
}
```

1.2 Delaunay Triangulation

```
/*
Stanford notebook
-----
Delaunay Algorithm Does not handle degenerate cases
Running time: O(n^4)
INPUT: x[] = x-coordinates
        y[] = y-coordinates
OUTPUT: triples = a vector containing m triples
        (indices corresponding to triangle vertices)
-----
*/

typedef double T;

struct triple {
    int i, j, k;
    triple() {}
    triple(int i, int j, int k) : i(i), j(j), k(k) {}
};

vector<triple> delaunayTriangulation(vector<T>& x, vector<T>& y)
{
    int n = x.size();
    vector<T> z(n);
    vector<triple> ret;

    for (int i = 0; i < n; i++)
        z[i] = x[i] * x[i] + y[i] * y[i];
    for (int i = 0; i < n-2; i++)
    {
        for (int j = i+1; j < n; j++)
        {
            for (int k = i+1; k < n; k++)
            {
                if (j == k) continue;
                double xn = (y[j]-y[i])*(z[k]-z[i]) - (y[k]-y[i])*(z[j]-z[i]);
                double yn = (x[k]-x[i])*(z[j]-z[i]) - (x[j]-x[i])*(z[k]-z[i]);
                double zn = (x[j]-x[i])*(y[k]-y[i]) - (x[k]-x[i])*(y[j]-y[i]);
                bool flag = zn < 0;
                for (int m = 0; flag && m < n; m++)
                    flag = flag && ((x[m]-x[i])*xn + (y[m]-y[i])*yn + (z[m]-z[i])*zn <= 0);
                if (flag) ret.push_back(triple(i, j, k));
            }
        }
    }
    return ret;
}

int main() {
    T xs[]={0, 0, 1, 0.9};
    T ys[]={0, 1, 0, 0.9};
    vector<T> x(&xs[0], &xs[4]), y(&ys[0], &ys[4]);
    vector<triple> tri = delaunayTriangulation(x, y);
    //expected: 0 1 3
    //           0 3 2
    int i;
    for(i = 0; i < tri.size(); i++)
        printf("%d %d %d\n", tri[i].i, tri[i].j, tri[i].k);
    return 0;
}
```

1.3 Various Geometry Functions

```
// Stanford Notebook
double INF = 1e100;
double EPS = 1e-12;

struct PT {
    double x, y;
    PT() {}
    PT(double x, double y) : x(x), y(y) {}
    PT(const PT &p) : x(p.x), y(p.y) {}
    PT operator + (const PT &p) const { return PT(x+p.x, y+p.y); }
    PT operator - (const PT &p) const { return PT(x-p.x, y-p.y); }
    PT operator * (double c) const { return PT(x*c, y*c); }
    PT operator / (double c) const { return PT(x/c, y/c); }
};

double dot(PT p, PT q) { return p.x*q.x+p.y*q.y; }
double dist2(PT p, PT q) { return dot(p-q,p-q); }
double cross(PT p, PT q) { return p.x*q.y-p.y*q.x; }
```

```

// rotate a point CCW or CW around the origin
PT RotateCCW90(PT p) { return PT(-p.y,p.x); }
PT RotateCW90(PT p) { return PT(p.y,-p.x); }
PT RotateCCW(PT p, double t) { return PT(p.x*cos(t)-p.y*sin(t), p.x*sin(t)+p.y*cos(t)); }

// project point c onto line through a and b // assuming a != b
PT ProjectPointLine(PT a, PT b, PT c) { return a + (b-a)*dot(c-a, b-a)/dot(b-a, b-a); }
// project point c onto line segment through a and b
PT ProjectPointSegment(PT a, PT b, PT c)
{
    double r = dot(b-a,b-a);
    if (fabs(r) < EPS) return a;
    r = dot(c-a, b-a)/r;
    if (r < 0) return a;
    if (r > 1) return b;
    return a + (b-a)*r;
}

// compute distance from c to segment between a and b
double DistancePointSegment(PT a, PT b, PT c)
{
    return sqrt(dist2(c, ProjectPointSegment(a, b, c)));
}

// compute distance between point (x,y,z) and plane ax+by+cz=d
double DistancePointPlane(double x, double y, double z, double a, double b, double c, double d)
{
    return fabs(a*x+b*y+c*z-d)/sqrt(a*a+b*b+c*c);
}

// determine if lines from a to b and c to d are parallel or collinear
bool LinesParallel(PT a, PT b, PT c, PT d) { return fabs(cross(b-a, c-d)) < EPS; }

bool LinesCollinear(PT a, PT b, PT c, PT d)
{
    return LinesParallel(a, b, c, d)
    && fabs(cross(a-b, a-c)) < EPS
    && fabs(cross(c-d, c-a)) < EPS;
}

// determine if line segment from a to b intersects with
// line segment from c to d
bool SegmentsIntersect(PT a, PT b, PT c, PT d)
{
    if (LinesCollinear(a, b, c, d))
    {
        if (dist2(a, c) < EPS || dist2(a, d) < EPS ||
            dist2(b, c) < EPS || dist2(b, d) < EPS) return true;
        if (dot(c-a, c-b) > 0 && dot(d-a, d-b) > 0 && dot(c-b, d-b) > 0)
            return false;
        return true;
    }
    if (cross(d-a, b-a) * cross(c-a, b-a) > 0)
        return false;
    if (cross(a-c, d-c) * cross(b-c, d-c) > 0)
        return false;
    return true;
}

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

// compute center of circle given three points
PT ComputeCircleCenter(PT a, PT b, PT c)
{
    b=(a+b)/2;
    c=(a+c)/2;
    return ComputeLineIntersection(b, b+RotateCW90(a-b), c, c+RotateCW90(a-c));
}

// determine if point is in a possibly non-convex polygon (by William
// Randolph Franklin); returns 1 for strictly interior points, 0 for
// strictly exterior points, and 0 or 1 for the remaining points.
// Note that it is possible to convert this into an "exact" test using
// integer arithmetic by taking care of the division appropriately
// (making sure to deal with signs properly) and then by writing exact
// tests for checking point on polygon boundary
bool PointInPolygon(const vector<PT> &p, PT q)
{
    bool c = 0;
    for (int i = 0; i < p.size(); i++)

```

```

{
    int j = (i+1)%p.size();
    if ((p[i].y <= q.y && q.y < p[j].y ||
        p[j].y <= q.y && q.y < p[i].y) &&
        q.x < p[i].x + (p[j].x - p[i].x) * (q.y - p[i].y) / (p[j].y - p[i].y))
        c = !c;
    return c;
}

// determine if point is on the boundary of a polygon
bool PointOnPolygon(const vector<PT> &p, PT q)
{
    for (int i = 0; i < p.size(); i++)
        if (dist2(ProjectPointSegment(p[i], p[(i+1)%p.size()], q), q) < EPS)
            return true;
    return false;
}

// compute intersection of line through points a and b with
// circle centered at c with radius r > 0
vector<PT> CircleLineIntersection(PT a, PT b, PT c, double r)
{
    vector<PT> ret;
    b = b-a; a = a-c;
    double A = dot(b, b);
    double B = dot(a, b);
    double C = dot(a, a) - r*r;
    double D = B*B - A*C;
    if (D < -EPS) return ret;
    ret.push_back(c+a+b*(-B+sqrt(D+EPS))/A);
    if (D > EPS)
        ret.push_back(c+a+b*(-B-sqrt(D))/A);
    return ret;
}

// compute intersection of circle centered at a with radius r
// with circle centered at b with radius R
vector<PT> CircleCircleIntersection(PT a, PT b, double r, double R)
{
    vector<PT> ret;
    double d = sqrt(dist2(a, b));
    if (d > r+R || d+min(r, R) < max(r, R)) return ret;
    double x = (d+d-R+r)/(2*d);
    double y = sqrt(r*r-x*x);
    PT v = (b-a)/d;
    ret.push_back(a+v*x + RotateCCW90(v)*y);
    if (y > 0) ret.push_back(a+v*x - RotateCCW90(v)*y);
    return ret;
}

// This code computes the area or centroid of a (possibly nonconvex)
// polygon, assuming that the coordinates are listed in a clockwise or
// counterclockwise fashion. Note that the centroid is often known as
// the "center of gravity" or "center of mass".
double ComputeSignedArea(const vector<PT> &p)
{
    double area = 0;
    for(int i = 0; i < p.size(); i++)
    {
        int j = (i+1) % p.size();
        area += p[i].x*p[j].y - p[j].x*p[i].y;
    }
    return area / 2.0;
}

double ComputeArea(const vector<PT> &p) { return fabs(ComputeSignedArea(p)); }

PT ComputeCentroid(const vector<PT> &p)
{
    PT c(0,0);
    double scale = 6.0 * ComputeSignedArea(p);
    for (int i = 0; i < p.size(); i++)
    {
        int j = (i+1) % p.size();
        c = c + (p[i]+p[j])*(p[i].x*p[j].y - p[j].x*p[i].y);
    }
    return c / scale;
}

// tests whether or not a given polygon (in CW or CCW order) is simple
bool IsSimple(const vector<PT> &p)
{
    for (int i = 0; i < p.size(); i++)
    {
        for (int k = i+1; k < p.size(); k++)
        {
            int j = (i+1) % p.size();
            int l = (k+1) % p.size();
            if (i == 1 || j == k) continue;

```

```

        if (SegmentsIntersect(p[i], p[j], p[k], p[l]))
            return false;
    }
    return true;
}

```

2 Graphs

2.1 Dijkstra's Algorithm

```

//Dijkstra Algorithm
int t,n,m,s,e;
vector<ii> edges[N]; //pair<NodeEnd,dist>
int distances[N]; // =INF=0x3f3f3f3f
int parent[N]; // ==-1

int Dijkstra()
{
    vector<ii> :: iterator it;
    priority_queue< ii, vector<ii>, greater<ii> > pq;
    distances[s]=0;
    pq.push(ii(distances[s],s));
    while(!pq.empty())
    {
        ii p = pq.top();
        pq.pop();
        int d=p.first;
        int a=p.second;
        for(it=edges[a].begin(); it!=edges[a].end(); ++it)
        {
            if(distances[it->first]>distances[a]+it->second)
            {
                distances[it->first]=distances[a]+it->second;
                parent[it->first]=a;
                pq.push(ii(distances[it->first],it->first));
            }
        }
    }
    return distances[e];
}

```

2.2 Max Flow (Dinic's Algorithm)

```

// Stanford Notebook
typedef long long LL;

struct Edge
{
    int from, to, cap, flow, index;
    Edge(int from, int to, int cap, int flow, int index) :
        from(from), to(to), cap(cap), flow(flow), index(index) {}
    LL rcap() { return cap - flow; }
};

struct Dinic
{
    int N;
    vector<vector<Edge> > G;
    vector<vector<Edge> > Lf;
    vector<int> layer;
    vector<int> Q;
    Dinic(int N) : N(N), G(N), Q(N) {}
    void AddEdge(int from, int to, int cap)
    {
        if (from == to) return;
        G[from].push_back(Edge(from, to, cap, 0, G[to].size()));
        G[to].push_back(Edge(to, from, 0, 0, G[from].size() - 1));
    }

    LL BlockingFlow(int s, int t)
    {
        layer.clear();
        layer.resize(N, -1);
        layer[s] = 0;
        Lf.clear(); Lf.resize(N);

        int head = 0, tail = 0;
        Q[tail++] = s;
        while (head < tail)

```

```

    {
        int x = Q[head++];
        for (int i = 0; i < G[x].size(); i++)
        {
            Edge &e = G[x][i]; if (e.rcap() <= 0) continue;
            if (layer[e.to] == -1)
            {
                layer[e.to] = layer[e.from] + 1;
                Q[tail++] = e.to;
            }
            if (layer[e.to] > layer[e.from])
            {
                Lf[e.from].push_back(&e);
            }
        }
    }

    if (layer[t] == -1) return 0;
    LL totflow = 0;
    vector<Edge> > P;
    while (!Lf[s].empty())
    {
        int curr = P.empty() ? s : P.back()->to;
        if (curr == t)
        {
            // Augment
            LL amt = P.front()->rcap();
            for (int i = 0; i < P.size(); ++i)
            {
                amt = min(amt, P[i]->rcap());
            }
            totflow += amt;
            for (int i = P.size() - 1; i >= 0; --i)
            {
                P[i]->flow += amt;
                G[P[i]->to][P[i]->index].flow -= amt;
                if (P[i]->rcap() <= 0)
                {
                    Lf[P[i]->from].pop_back();
                    P.resize(i);
                }
            }
        }
        else if (Lf[curr].empty())
        {
            // Retreat
            P.pop_back();
            for (int i = 0; i < N; ++i)
            {
                for (int j = 0; j < Lf[i].size(); ++j)
                {
                    if (Lf[i][j]->to == curr)
                        Lf[i].erase(Lf[i].begin() + j);
                }
            }
        }
        else
        {
            // Advance
            P.push_back(Lf[curr].back());
        }
    }
    return totflow;
}

LL GetMaxFlow(int s, int t)
{
    LL totflow = 0;
    while (LL flow = BlockingFlow(s, t))
        totflow += flow;
    return totflow;
}
};

```

2.3 Max Flow (Edmonds-Karp Algorithm)

```

/*
Stanford Notebook
MinCostMaxFlow (adjacency matrix, Edmonds and Karp 1972)
This implementation keeps track of forward and reverse edges separately
(so you can set cap[i][j] != cap[j][i]). For a regular max flow, set all edge costs to 0.
Running time,  $O(|V|^2)$  cost per augmentation
max flow:  $O(|V|^3)$  augmentations
min cost max flow:  $O(|V|^4 * MAX\_EDGE\_COST)$  augmentations
INPUT: - graph, constructed using AddEdge()
        - sink
OUTPUT: - (maximum flow value, minimum cost value)
        - To obtain the actual flow, look at positive values only.
*/
typedef vector<int> VI;

```

```

typedef vector<VI> VVI;
typedef long long L;
typedef vector<L> VL;
typedef vector<VL> VVL;
typedef pair<int, int> PII;
typedef vector<PII> VPII;

const L INF = numeric_limits<L>::max() / 4;

struct MinCostMaxFlow {
    int N;
    VVL cap, flow, cost;
    VI found;
    VL dist, pi, width;
    VPII dad;

    MinCostMaxFlow(int N) :
        N(N), cap(N, VL(N)), flow(N, VL(N)), cost(N, VL(N)),
        found(N), dist(N), pi(N), width(N), dad(N) {}

    void AddEdge(int from, int to, L cap, L cost)
    {
        this->cap[from][to] = cap;
        this->cost[from][to] = cost;
    }

    void Relax(int s, int k, L cap, L cost, int dir)
    {
        L val = dist[s] + pi[s] - pi[k] + cost;
        if (cap && val < dist[k])
        {
            dist[k] = val;
            dad[k] = make_pair(s, dir);
            width[k] = min(cap, width[s]);
        }
    }

    L Dijkstra(int s, int t)
    {
        fill(found.begin(), found.end(), false);
        fill(dist.begin(), dist.end(), INF);
        fill(width.begin(), width.end(), 0);
        dist[s] = 0;
        width[s] = INF;

        while (s != -1)
        {
            int best = -1;
            found[s] = true;
            for (int k = 0; k < N; k++)
            {
                if (found[k]) continue;
                Relax(s, k, cap[s][k] - flow[s][k], cost[s][k], 1);
                Relax(s, k, flow[k][s], -cost[k][s], -1);
                if (best == -1 || dist[k] < dist[best]) best = k;
            }
            s = best;
        }

        for (int k = 0; k < N; k++)
            pi[k] = min(pi[k] + dist[k], INF);
        return width[t];
    }

    pair<L, L> GetMaxFlow(int s, int t)
    {
        L totflow = 0, totcost = 0;
        while (L amt = Dijkstra(s, t))
        {
            totflow += amt;
            for (int x = t; x != s; x = dad[x].first)
            {
                if (dad[x].second == 1)
                {
                    flow[dad[x].first][x] += amt;
                    totcost += amt * cost[dad[x].first][x];
                }
                else
                {
                    flow[x][dad[x].first] -= amt;
                    totcost -= amt * cost[x][dad[x].first];
                }
            }
        }
        return make_pair(totflow, totcost);
    }
};

```

2.4 Eulerian Path

```

struct Edge;
typedef list<Edge>::iterator iter;
struct Edge
{
    int next_vertex;
    iter reverse_edge;
    Edge(int next_vertex) :next_vertex(next_vertex) {}
};

const int max_vertices = ;
int num_vertices;
list<Edge> adj[max_vertices]; // adjacency list
vector<int> path;

void find_path(int v)
{
    while(adj[v].size() > 0)
    {
        int vn = adj[v].front().next_vertex;
        adj[vn].erase(adj[v].front().reverse_edge);
        adj[v].pop_front();
        find_path(vn);
    }
    path.push_back(v);
}

void add_edge(int a, int b)
{
    adj[a].push_front(Edge(b));
    iter ita = adj[a].begin();
    adj[b].push_front(Edge(a));
    iter itb = adj[b].begin();
    ita->reverse_edge = itb;
    itb->reverse_edge = ita;
}

```

2.5 Hopcroft-Karp Algorithm

```

#include <vector>

vector<int> g[N];
int r[N], l[N], n, m, e, a, b;

// generate g;

bool dfs(int v)
{
    if(vis[v]) return false;
    vis[v] = true;
    for(int u=0; u<g[v].size(); ++u)
    {
        if(!r[g[v][u]])
        {
            l[v]=g[v][u];
            r[g[v][u]]=v;
            return true;
        }
    }
    for(int u=0; u<g[v].size(); ++u)
    {
        if(dfs(r[g[v][u]]))
        {
            l[v]=g[v][u];
            r[g[v][u]]=v;
            return true;
        }
    }
    return false;
}

void hopcroft_karp()
{
    bool change = true;
    while(change)
    {
        change = false;
        fill(vis, vis+n+1, false);
        for(int i=1; i<=n; ++i)
            if(!l[i])
                change |= dfs(i);
    }
}

```

2.6 Lowest Common Ancestor

```

const int max_nodes, log_max_nodes;

int num_nodes, log_num_nodes, root;
vector<int> children[max_nodes];
// children[i] contains the children of node i
int A[max_nodes][log_max_nodes+1]; // A[i][j] is the 2^j-th ancestor of node i, or -1 if that ancestor does not exist
int L[max_nodes]; // L[i] is the distance between node i and the root
// floor of the binary logarithm of n
int lb(unsigned int n)
{
    if(n==0) return -1;
    int p = 0;
    if (n >= 1<<16) { n >= 16; p += 16; }
    if (n >= 1<<8) { n >= 8; p += 8; }
    if (n >= 1<<4) { n >= 4; p += 4; }
    if (n >= 1<<2) { n >= 2; p += 2; }
    if (n >= 1<<1) { p += 1; }
    return p;
}

void DFS(int i, int l)
{
    L[i] = l;
    for(int j = 0; j < children[i].size(); j++)
        DFS(children[i][j], l+1);
}

int LCA(int p, int q) {
    // ensure node p is at least as deep as node q
    if(L[p] < L[q]) swap(p, q);

    // "binary search" for the ancestor of node p situated on the same level as q
    for(int i = log_num_nodes; i >= 0; i--)
        if(L[p] - (1<<i) >= L[q])
            p = A[p][i];
    if(p == q) return p;

    // "binary search" for the LCA
    for(int i = log_num_nodes; i >= 0; i--)
    {
        if(A[p][i] != -1 && A[p][i] != A[q][i])
        {
            p = A[p][i];
            q = A[q][i];
        }
    }
    return A[p][0];
}

int main(int argc, char* argv[])
{
    // read num_nodes, the total number of nodes
    log_num_nodes = lb(num_nodes);
    for(int i = 0; i < num_nodes; i++)
    {
        int p;
        // read p, the parent of node i or -1 if node i is the root
        A[i][0] = p;
        if(p != -1) children[p].push_back(i);
        else root = i;
    }
    // precompute A using dynamic programming
    for(int j = 1; j <= log_num_nodes; j++)
        for(int i = 0; i < num_nodes; i++)
            if(A[i][j-1] != -1) A[i][j] = A[A[i][j-1]][j-1];
            else A[i][j] = -1;
    // precompute L
    DFS(root, 0);
    return 0;
}

```

2.7 Strongly Connected Components

```

#include <memory.h>

struct edge
{
    int e, nxt;
};

int V, E;
edge e[MAXE], er[MAXE];

```

```

int sp[MAXV], spr[MAXV];
int group_cnt, group_num[MAXV];
bool v[MAXV];
int stk[MAXV];

void fill_forward(int x)
{
    int i;
    v[x] = true;
    for(i = sp[x]; i != e[i].nxt)
        if(!v[e[i].e]) fill_forward(e[i].e);
    stk[++stk[0]] = x;
}

void fill_backward(int x)
{
    int i;
    v[x] = false;
    group_num[x] = group_cnt;
    for(i = spr[x]; i != er[i].nxt)
        if(v[er[i].e]) fill_backward(er[i].e);
}

void add_edge(int v1, int v2) //add edge v1->v2
{
    e[++E].e = v2;
    e[E].nxt = sp[v1];
    sp[v1] = E;
    er[E].e = v1;
    er[E].nxt = spr[v2];
    spr[v2] = E;
}

void SCC()
{
    int i;
    stk[0] = 0;
    memset(v, false, sizeof(v));
    for(i = 1; i <= V; i++)
        if(!v[i]) fill_forward(i);

    group_cnt = 0;
    for(i = stk[0]; i >= 1; i--)
        if(v[stk[i]]) { group_cnt++; fill_backward(stk[i]); }
}

```

2.8 Union-Find Set

```

struct Edge // MST
{
    int a, b, d;
    bool operator < (const Edge &E) const {
        return this->d < E.d;
    }
};

int ranks[M]; int c[N];

int Find(int x)
{
    int y = x;
    while(y != c[y])
        y = c[y];
    while(x != c[x])
    {
        int aux = c[x];
        c[x] = y;
        x = aux;
    }
    return y;
}

void Union(int x, int y)
{
    if(ranks[x] > ranks[y])
        c[x] = y;
    else
        c[y] = x;
    if(ranks[x] == ranks[y])
        ranks[y]++;
}

```

3 Tree

3.1 Cartesian Tree

```

struct Tr
{
    Tr *l,*r;
    int key,pr,cnt,val,rev;
    long long sum;
    Tr(int new_key,int new_pr,int new_val)
    {
        rev=0;
        key=new_key;
        cnt=1;
        l=r=NULL;
        pr=new_pr;
        val=new_val;
        sum=new_val;
    }
};

#define T Tr*
T R=NULL;

int cnt(T t)
{
    if(!t) return 0;
    return t->cnt;
}

void upd_cnt(T &t)
{
    if(t) t->cnt=cnt(t->l)+cnt(t->r)+1;
}

long long sum(T t)
{
    if(!t) return 0;
    return t->sum;
}

void upd_sum(T &t)
{
    if(t) t->sum=sum(t->l)+sum(t->r)+t->val;
}

void push(T &t)
{
    if(t && t->rev)
    {
        t->rev=0;
        swap(t->l,t->r);
        upd_sum(t);
        if(t->l) t->l->rev^=1;
        if(t->r) t->r->rev^=1;
    }
}

void split(T t,T &l,T &r,int key,int add)
{
    if(!t) return void(l=r=NULL);
    push(t);
    upd_cnt(t);
    int current_key=add+cnt(t->l)+1;
    if(key<=current_key)
        split(t->l,l,t->l,key,add),r=t;
    else
        split(t->r,t->r,r,key,current_key),l=t;
    upd_cnt(t);
    upd_sum(t);
}

void merge(T &t,T l,T r)
{
    push(l);
    push(r);
    if(!l || !r)
        t=l?l:r;
    else if(l->pr>r->pr)
        merge(l->r,l->r,r), t=l;
    else
        merge(r->l,l,r->l), t=r;
    upd_cnt(t);
    upd_sum(t);
}

```

```

}

void insert(T &t,T it,int add)
{
    push(t);
    if(!t)
    {
        t=it;
        upd_cnt(t);
        return;
    }
    upd_sum(t);
    if(it->pr > t->pr)
        split(t,it->l,it->r,it->key,add),t=it;
    else if(it->key>add+cnt(t->l)+1)
        insert(t->r,it,add+cnt(t->l)+1);
    else
        insert(t->l,it,add);
    upd_sum(t);
    upd_cnt(t);
}

void print(T t)
{
    if(!t) return;
    print(t->l);
    cout<<t->val<<" ";
    print(t->r);
}

void reverse(int left,int right)
{
    Tr *t1,*t2,*t3;
    t1=t2=t3=NULL;
    split(R,t1,t2,left,0);
    split(t2,t2,t3,right-left+2,0);
    t2->rev^=1;
    merge(R,t1,t2);
    merge(R,R,t3);
}

void get_sum(int left,int right)
{
    Tr *t1,*t2,*t3;
    t1=t2=t3=NULL;
    split(R,t1,t2,left,0);
    split(t2,t2,t3,right-left+2,0);
    cout<<t2->sum<<"\n";
    merge(R,t1,t2);
    merge(R,R,t3);
}

int n,m, q,a,b;
void example()
{
    ios_base::sync_with_stdio(0);
    cin.tie(0);
    freopen("reverse.in","r",stdin);
    freopen("reverse.out","w",stdout);
    srand(time(0));
    cin>>n>>m;
    for(int i=1;i<=n;++i)
    {
        cin>>a;
        T it=new Tr(i,rand()+1,a);
        insert(R,it,0);
    }
    for(int i=0;i<m;++i)
    {
        cin>>q>>a>>b;
        if(q) reverse(a,b);
        else get_sum(a,b);
    }
    return 0;
}

```

3.2 Segment Tree

```

//Segment Tree
#include <iostream>
#define N (1<<18)

using namespace std;
//int find(vector <int>& C, int x){return (C[x]==x) ? x : C[x]=find(C, C[x]);} C++
//int find(int x){return (C[x]==x)?x:C[x]=find(C[x]);}

```

```

typedef pair<int,int> ii;
ii arb[N]={ {0,0} };
int n,m,a,b,v;
char type;

void update(int node,int l,int r,int a,int b,int val,int p)
{
    if(a<=l && r<=b)
    {
        arb[node].first=val;
        arb[node].second=p;
    }
    else
    {
        int mid=(l+r)/2;
        if(a<=mid)
            update(node*2,l,mid,a,b,val,p);
        if(b>mid)
            update(2*node+1,mid+1,r,a,b,val,p);
    }
}

pair<int,int> search(int node,int l,int r,int a)
{
    if(a==l && a==r)
    {
        return arb[node];
    }
    else
    {
        int mid=(l+r)/2;
        ii cur;
        if(a<=mid)
            cur=search(2*node,l,mid,a);
        else
            cur=search(2*node+1,mid+1,r,a);
        if(cur.second<arb[node].second)
            return arb[node];
        else
            return cur;
    }
}

```

4 Math

4.1 Extended Euclid's Algorithm

```

#include "GcdLcm.h"

// returns d = gcd(a,b); find x, y such that d = ax + by
int extended_euclid(int a, int b, int &x, int &y){
    int current_x = y = 0;
    int current_y = x = 1;
    while(b)
    {
        int q = a/b;
        int t = b;
        b = a%b;
        a = t;
        t = current_x; current_x = x-q*current_x; x = t;
        t = current_y; current_y = y-q*current_y; y = t;
    }
    return a;
}

```

4.2 Fast Prime Number Sieve

```

// compute prime numbers to N really fast (Sieve)
#define N 10000100
vector<int> primes;
char p[N/8];

void GeneratePrimes()
{
    primes.push_back(2);
    int i,j;
    for(i=1; ((i+1)<<1)+(i<<1)<N; ++i)
    {
        if((p[i>>3] & (1<<(i&7)))==0)

```

```

        primes.push_back((i<<1)+1);
        for(j=((i+1)<<1)+(i<<1),k=(i<<1)+1; (j<<1)+1<N; j+=k)
        {
            p[j>>3] |= (1<<(j&7));
        }
    }
    for(; (i<<1)+1<N; ++i)
    {
        if((p[i>>3] & (1<<(i&7)))==0)
        {
            primes.push_back((i<<1)+1);
        }
    }
}

```

4.3 Fast Fourier Transform

```

struct cpx
{
    cpx() {}
    cpx(double aa): a(aa) {}
    cpx(double aa, double bb): a(aa), b(bb) {}
    double a;
    double b;
    double modsq(void) const{ return a*a+b*b; }
    cpx bar(void) const {return cpx(a,-b);}
};

cpx b[N+100],c[N+100],B[N+100],C[N+100];
int a[N+100], int x[N+100];
double coss[N+100], sins[N+100];
int n,m,p;

cpx operator +(cpx a,cpx b) { return cpx(a.a+b.a,a.b+b.b); }
cpx operator *(cpx a,cpx b) { return cpx(a.a*b.a-a.b*b.b,a.a*b.b+a.b*b.a); }
cpx operator /(cpx a,cpx b) { cpx r = a*b.bar(); return cpx(r.a/b.modsq(),r.b/b.modsq()); }
cpx EXP(int i,int dir) { return cpx(coss[i],sins[i]*dir); }

const double two_pi = 4 * acos(0);

void FFT(cpx *in,cpx *out,int step,int size,int dir)
{
    if(size<1) return;
    if(size==1)
    {
        out[0]=in[0];
        return;
    }
    FFT(in,out,step*2,size/2,dir);
    FFT(in+step,out+size/2,step*2,size/2,dir);
    for(int i=0;i<size/2; ++i)
    {
        cpx even=out[i];
        cpx odd=out[i+size/2];
        out[i] = even+EXP(i*step,dir)*odd;
        out[i+size/2]=even+EXP((i+size/2)*step,dir)*odd;
    }
}

void exemple
{
    for(int i=0;i<N; ++i)
    {
        coss[i]=cos(two_pi*i/N);
        sins[i]=sin(two_pi*i/N);
    }
    while(scanf("%d",&n)==1)
    {
        fill(x,x+N+100,0);
        fill(a,a+N+100,0);
        for(int i=0;i<n; ++i)
        {
            scanf("%d",&p);
            x[p]=1;
        }
        for(int i=0;i<N+100; ++i)
        {
            b[i]=cpx(x[i],0);
        }
        scanf("%d",&m);
        for(int i=0;i<m; ++i)
        {
            scanf("%d",&a[i]);
        }
    }
}

```

```

        FFT(b,B,1,N,1);
        for(int i=0;i<N;++i)
            C[i]=B[i]*B[i];
        FFT(C,c,1,N,-1);
        for(int i=0;i<N;++i)
            c[i]=c[i]/N;
        int cnt=0;
        for(int i=0;i<16;++i)
            cout<<c[i].a<<" ";
        for(int i=0;i<m;++i)
            if(c[a[i]].a>0.5 || x[a[i]])
                cnt++;
        printf("%d\n",cnt);
    }
}

```

4.4 Gauss-Jordan Elimination

```

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

const double EPS = 1e-10;
typedef vector<int> VI;
typedef double T;
typedef vector<T> VT;
typedef vector<VT> VVT;

T GaussJordan(VVT &a, VVT &b)
{
    const int n = a.size();
    const int m = b[0].size();
    VI irow(n), icol(n), ipiv(n);
    T det = 1;

    for (int i = 0; i < n; i++)
    {
        int pj = -1, pk = -1;
        for (int j = 0; j < n; j++)
            if (!ipiv[j])
                for (int k = 0; k < n; k++)
                    if (!ipiv[k])
                        if (pj == -1 || fabs(a[j][k]) > fabs(a[pj][pk])) { pj = j; pk = k; }

        if (fabs(a[pj][pk]) < EPS)
        {
            cerr << "Matrix is singular." << endl;
            exit(0);
        }

        ipiv[pj]++;
        swap(a[pj], a[pk]);
        swap(b[pj], b[pk]);
        if (pj != pk) det *= -1;
        irow[i] = pj;
        icol[i] = pk;

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

        for (int p = 0; p < n; p++)
            a[pk][p] *= c;
        for (int p = 0; p < m; p++)
            b[pk][p] *= c;
        for (int p = 0; p < n; p++)
        {
            if (p != pk)
            {
                c = a[p][pk];
                a[p][pk] = 0;
                for (int q = 0; q < n; q++)
                    a[p][q] -= a[pk][q] * c;
                for (int q = 0; q < m; q++)
                    b[p][q] -= b[pk][q] * c;
            }
        }
    }
}

```

```

    }
    for (int p = n-1; p >= 0; p--)
    {
        if (irow[p] != icol[p])
        {
            for (int k = 0; k < n; k++)
                swap(a[k][irow[p]], a[k][icol[p]]);
        }
        return det;
    }
}

int main()
{
    const int n = 4;
    const int m = 2;
    double A[n][n] = { {1,2,3,4},{1,0,1,0},{5,3,2,4},{6,1,4,6} };
    double B[n][m] = { {1,2},{4,3},{5,6},{8,7} };
    VVT a(n), b(n);
    for (int i = 0; i < n; i++)
    {
        a[i] = VT(A[i], A[i] + n);
        b[i] = VT(B[i], B[i] + m);
    }
    double det = GaussJordan(a, b);
    // expected: 60
    cout << "Determinant: " << det << endl;
    // expected: -0.233333 0.166667 0.133333 0.0666667
    //           0.166667 0.166667 0.333333 -0.333333
    //           0.233333 0.833333 -0.133333 -0.0666667
    //           0.05 -0.75 -0.1 0.2
    cout << "Inverse: " << endl;
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
            cout << a[i][j] << ' ';
        cout << endl;
    }
    // expected: 1.63333 1.3
    //           -0.166667 0.5
    //           2.36667 1.7
    //           -1.85 -1.35
    cout << "Solution: " << endl;
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < m; j++)
            cout << b[i][j] << ' ';
        cout << endl;
    }
}

```

4.5 GCD and LCM

```

// return a*b
int mod(int a, int b)
{
    return ((a%b)+b)%b;
}

// computes gcd(a,b)
int gcd(int a, int b)
{
    int tmp;
    while(b)
    {
        a %= b;
        tmp = a;
        a = b;
        b = tmp;
    }
    return a;
}

// computes lcm(a,b)
int lcm(int a, int b)
{
    return a/gcd(a,b)*b;
}

```

4.6 Modular Linear Equation Solver


```
// Stanford Notebook
#include <vector>
#include "ExtendedEuclid.h"

// find all solutions to ax = b (mod n)
vector<int> modular_linear_equation_solver(int a, int b, int n)
{
    int x, y;
    vector<int> sol;
    int d = extended_euclid(a, n, x, y);
    if(!(b%d)) {
        x = mod(x*(b/d), n);
        for(int i=0; i < d; ++i)
            sol.push_back(mod(x + i*(n/d), n));
    }
    return sol;
}

// computes b such tath ab = 1(mod n), returs -1 on failure
int mod_inverse(int a, int n)
{
    int x, y;
    int d = extended_euclid(a, n, x, y);
    if(d>1) return -1;
    return mod(x, n);
}

// computes x and y such that ax + by =c; on failure, x = y =-1
void linear_diophantine(int a, int b, int c, int &x, int &y)
{
    int d = gcd(a, b);
    if(c%d) x = y = -1;
    else {
        x = c/d + mod_inverse(a/d, b/d);
        y = (c-a*x)/b;
    }
}

```

5 Strings

5.1 Aho-Corasick Algorithm

```
/*
Implementation - Benoit Chabod
=====
Aho Corasick algorithm
=====
*/

struct node
{
    int f;
    map<char, int> g;
    vector<short> out;
    node(int fail = -1): f(fail) {}
};

vector<node> nodes;

void add_str(const string &s, int num)
{
    int cur = 0;
    int n = s.size();
    for(int i = 0; i < n; ++i)
    {
        auto it = nodes[cur].g.find(s[i]);
        if(it == nodes[cur].g.end())
        {
            nodes[cur].g[s[i]] = nodes.size();
            cur = nodes.size();
            nodes.push_back(node());
        }
        else {
            cur = it->second;
        }
    }
    nodes[cur].out.push_back(num);
}

void init_fail()
{
    int cur = 0;

```

```
queue<int> q;
q.push(cur);

while( !q.empty() )
{
    cur = q.front();
    map<char, int>::iterator it;
    for(it = nodes[cur].g.begin(); it != nodes[cur].g.end(); it++)
    {
        int child = it->second;
        int pfail = nodes[cur].f;
        char ch = it->first;
        map<char, int>::iterator f;
        while( pfail != -1 && ((f = nodes[pfail].g.find(ch)) == nodes[pfail].g.end()) )
        {
            pfail = nodes[pfail].f;
        }
        nodes[child].f = (pfail == -1)? 0 : f->second;
        pfail = nodes[child].f;
        nodes[child].out.insert(nodes[child].out.end(), nodes[pfail].out.begin(), nodes[pfail].out.end());
        q.push(child);
    }
    q.pop();
}

// Usage
void usage()
{
    nodes.push_back(node());
    for [each word] add_str(word, i)
        init_fail();
    for [each letter]
    {
        map<char, int>::iterator f;
        while( cur != -1 && ((f = nodes[cur].g.find(letter)) == nodes[cur].g.end()) )
        {
            cur = nodes[cur].f;
            if( cur == -1 )
            {
                cur = 0;
                continue;
            }
        }
        cur = f->second;
        for(auto v : nodes[cur].out)
        {
            // Word v was found
        }
    }
}

```

5.2 Knuth-Morris-Pratt Algorithm

```
/*
-----
KMP/Pi function
Note : cin>>(s+1) (the operations in the pi-function start at 1)
-----
*/

void preKmp()
{
    int k;
    k=kmpNext[1]=0;
    for(int i=2; i<=n; ++i)
    {
        while(k && p[k+1]!=p[i]) k=kmpNext[k];
        if(p[k+1]==p[i])
            k++;
        kmpNext[i]=k;
    }
}

void KMP()
{
    preKmp();
    int k=0;

    for(int i=1; i<=m; ++i)
    {
        while(k && p[k+1]!=s[i])
            k=kmpNext[k];
        if(p[k+1]==s[i])
            k++;
        if(k==n)
        {
            // here we have a match
            k=kmpNext[k];
        }
    }
}

```

5.3 Suffix Array

```
//Suffix Array

struct SuffixArray
{
    const int L;
    string s;
    vector<vector<int>> > P;
    vector<pair<pair<int,int>,int>> > M;
    SuffixArray(const string &s) : L(s.length()), s(s), P(1, vector<int>(L, 0)), M(L)
    {
        for (int i = 0; i < L; i++)
            P[0][i] = int(s[i]);
        for (int skip = 1, level = 1; skip < L; skip *= 2, level++)
        {
            P.push_back(vector<int>(L, 0));
            for (int i = 0; i < L; i++)
                M[i] = make_pair(make_pair(P[level-1][i], i + skip < L ? P[level-1][i]
                    + skip : -1000), i);
            sort(M.begin(), M.end());
            for (int i = 0; i < L; i++)
                P[level][M[i].second] = (i > 0 && M[i].first == M[i-1].first) ? P[
                    level][M[i-1].second] : i;
        }
    }

    vector<int> GetSuffixArray()
    {
        return P.back();
    }

    // returns the length of the longest common prefix of s[i...L-1] and s[j...L-1]
    int LongestCommonPrefix(int i, int j)
    {
        int len = 0;
        if (i == j) return L - i;
        for (int k = P.size() - 1; k >= 0 && i < L && j < L; k--)
        {
            if (P[k][i] == P[k][j])
            {
                i += 1 << k;
                j += 1 << k;
                len += 1 << k;
            }
        }
        return len;
    }
};

int main()
{
    // bobocel is the 0'th suffix
    // obocel is the 5'th suffix
    // bocel is the 1'st suffix
    // ocel is the 6'th suffix
    // cel is the 2'nd suffix
    // el is the 3'rd suffix
    // l is the 4'th suffix
    SuffixArray suffix("bobocel");
    vector<int> v = suffix.GetSuffixArray();
    // Expected output: 0 5 1 6 2 3 4
    //                2
    for (int i = 0; i < v.size(); i++)
        cout << v[i] << " ";
    cout << endl;
    cout << suffix.LongestCommonPrefix(0, 2) << endl;
}
```

6 Techniques

6.1 Various algorithm techniques

Recursion
Divide and conquer

Finding interesting points in $N \log N$

Greedy algorithm

- Scheduling
- Max contiguous subvector sum
- Invariants
- Huffman encoding

Graph theory

- Dynamic graphs (extra book-keeping)
- Breadth first search
- Depth first search
- * Normal trees / DFS trees
- Dijkstra's algorithm
- MST: Prim's algorithm
- Bellman-Ford
- Konig's theorem and vertex cover
- Min-cost max flow
- Lovasz toggle
- Matrix tree theorem
- Maximal matching, general graphs
- Hopcroft-Karp
- Hall's marriage theorem
- Graphical sequences
- Floyd-Warshall
- Eulercykler
- Flow networks
- * Augmenting paths
- * Edmonds-Karp
- Bipartite matching
- Min. path cover
- Topological sorting
- Strongly connected components
- 2-SAT
- Cutvertices, cutedges och biconnected components
- Edge coloring
- * Trees
- Vertex coloring
- * Bipartite graphs (\Rightarrow trees)
- * 3^n (special case of set cover)
- Diameter and centroid
- K'th shortest path
- Shortest cycle

Dynamic programming

- Knapsack
- Coin change
- Longest common subsequence
- Longest increasing subsequence
- Number of paths in a dag
- Shortest path in a dag
- Dynprog over intervals
- Dynprog over subsets
- Dynprog over probabilities
- Dynprog over trees
- 3^n set cover
- Divide and conquer
- Knuth optimization
- Convex hull optimizations
- RMQ (sparse table a.k.a 2^k -jumps)
- Bitonic cycle
- Log partitioning (loop over most restricted)

Combinatorics

- Computation of binomial coefficients
- Pigeon-hole principle
- Inclusion/exclusion
- Catalan number
- Pick's theorem

Number theory

- Integer parts
- Divisibility
- Euklidean algorithm
- Modular arithmetic
- * Modular multiplication
- * Modular inverses
- * Modular exponentiation by squaring
- Chinese remainder theorem
- Fermat's small theorem
- Euler's theorem
- Phi function
- Frobenius number
- Quadratic reciprocity
- Pollard-Rho
- Miller-Rabin
- Hensel lifting
- Vieta root jumping

Game theory

- Combinatorial games
- Game trees
- Mini-max
- Nim
- Games on graphs
- Games on graphs with loops
- Grundy numbers
- Bipartite games without repetition

- General games without repetition
- Alpha-beta pruning
- Probability theory
- Optimization
 - Binary search
 - Ternary search
 - Unimodality and convex functions
 - Binary search on derivative
- Numerical methods
 - Numeric integration
 - Newton's method
 - Root-finding with binary/ternary search
 - Golden section search
- Matrices
 - Gaussian elimination
 - Exponentiation by squaring
- Sorting
 - Radix sort
- Geometry
 - Coordinates and vectors
 - * Cross product
 - * Scalar product
 - Convex hull
 - Polygon cut
 - Closest pair
 - Coordinate-compression
 - Quadrees
 - KD-trees
 - All segment-segment intersection
- Sweeping
 - Discretization (convert to events and sweep)
 - Angle sweeping
 - Line sweeping

- Discrete second derivatives
- Strings
 - Longest common substring
 - Palindrome subsequences
 - Knuth-Morris-Pratt
 - Tries
 - Rolling polynomial hashes
 - Suffix array
 - Suffix tree
 - Aho-Corasick
 - Manacher's algorithm
 - Letter position lists
- Combinatorial search
 - Meet in the middle
 - Brute-force with pruning
 - Best-first (A*)
 - Bidirectional search
 - Iterative deepening DFS / A*
- Data structures
 - LCA (2*k-jumps in trees in general)
 - Pull/push-technique on trees
 - Heavy-light decomposition
 - Centroid decomposition
 - Lazy propagation
 - Self-balancing trees
 - Convex hull trick (wcipeg.com/wiki/Convex_hull_trick)
 - Monotone queues / monotone stacks / sliding queues
 - Sliding queue using 2 stacks
 - Persistent segment tree

| | | |
|--|--|---|
| $f(n) = O(g(n))$ | iff \exists positive c, n_0 such that $0 \leq f(n) \leq cg(n) \forall n \geq n_0$. | $\sum_{i=1}^n i = \frac{n(n+1)}{2}, \quad \sum_{i=1}^n i^2 = \frac{n(n+1)(2n+1)}{6}, \quad \sum_{i=1}^n i^3 = \frac{n^2(n+1)^2}{4}.$ |
| $f(n) = \Omega(g(n))$ | iff \exists positive c, n_0 such that $f(n) \geq cg(n) \geq 0 \forall n \geq n_0$. | In general: |
| $f(n) = \Theta(g(n))$ | iff $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$. | $\sum_{i=1}^n i^m = \frac{1}{m+1} \left[(n+1)^{m+1} - 1 - \sum_{i=1}^n ((i+1)^{m+1} - i^{m+1} - (m+1)i^m) \right]$ |
| $f(n) = o(g(n))$ | iff $\lim_{n \rightarrow \infty} f(n)/g(n) = 0$. | $\sum_{i=1}^{n-1} i^m = \frac{1}{m+1} \sum_{k=0}^m \binom{m+1}{k} B_k n^{m+1-k}.$ |
| $\lim_{n \rightarrow \infty} a_n = a$ | iff $\forall \epsilon > 0, \exists n_0$ such that $ a_n - a < \epsilon, \forall n \geq n_0$. | Geometric series: |
| $\sup S$ | least $b \in \mathbb{R}$ such that $b \geq s, \forall s \in S$. | $\sum_{i=0}^n c^i = \frac{c^{n+1} - 1}{c - 1}, \quad c \neq 1, \quad \sum_{i=0}^{\infty} c^i = \frac{1}{1 - c}, \quad \sum_{i=1}^{\infty} c^i = \frac{c}{1 - c}, \quad c < 1,$ |
| $\inf S$ | greatest $b \in \mathbb{R}$ such that $b \leq s, \forall s \in S$. | $\sum_{i=0}^n i c^i = \frac{nc^{n+2} - (n+1)c^{n+1} + c}{(c-1)^2}, \quad c \neq 1, \quad \sum_{i=0}^{\infty} i c^i = \frac{c}{(1-c)^2}, \quad c < 1.$ |
| $\liminf_{n \rightarrow \infty} a_n$ | $\lim_{n \rightarrow \infty} \inf \{a_i \mid i \geq n, i \in \mathbb{N}\}.$ | Harmonic series: |
| $\limsup_{n \rightarrow \infty} a_n$ | $\lim_{n \rightarrow \infty} \sup \{a_i \mid i \geq n, i \in \mathbb{N}\}.$ | $H_n = \sum_{i=1}^n \frac{1}{i}, \quad \sum_{i=1}^n i H_i = \frac{n(n+1)}{2} H_n - \frac{n(n-1)}{4}.$ |
| $\binom{n}{k}$ | Combinations: Size k subsets of a size n set. | $\sum_{i=1}^n H_i = (n+1)H_n - n, \quad \sum_{i=1}^n \binom{i}{m} H_i = \binom{n+1}{m+1} \left(H_{n+1} - \frac{1}{m+1} \right).$ |
| $[n]$ | Stirling numbers (1st kind): Arrangements of an n element set into k cycles. | 1. $\binom{n}{k} = \frac{n!}{(n-k)!k!}, \quad 2. \sum_{k=0}^n \binom{n}{k} = 2^n, \quad 3. \binom{n}{k} = \binom{n}{n-k},$ |
| $\left\{ \begin{matrix} n \\ k \end{matrix} \right\}$ | Stirling numbers (2nd kind): Partitions of an n element set into k non-empty sets. | 4. $\binom{n}{k} = \frac{n}{k} \binom{n-1}{k-1}, \quad 5. \binom{n}{k} = \binom{n-1}{k} + \binom{n-1}{k-1},$ |
| $\langle \begin{matrix} n \\ k \end{matrix} \rangle$ | 1st order Eulerian numbers: Permutations $\pi_1 \pi_2 \dots \pi_n$ on $\{1, 2, \dots, n\}$ with k ascents. | 6. $\binom{n}{m} \binom{m}{k} = \binom{n}{k} \binom{n-k}{m-k}, \quad 7. \sum_{k=0}^n \binom{r+k}{k} = \binom{r+n+1}{n},$ |
| $\langle \langle \begin{matrix} n \\ k \end{matrix} \rangle \rangle$ | 2nd order Eulerian numbers. | 8. $\sum_{k=0}^n \binom{k}{m} = \binom{n+1}{m+1}, \quad 9. \sum_{k=0}^n \binom{r}{k} \binom{s}{n-k} = \binom{r+s}{n},$ |
| C_n | Catalan Numbers: Binary trees with $n+1$ vertices. | 10. $\binom{n}{k} = (-1)^k \binom{k-n-1}{k}, \quad 11. \left\{ \begin{matrix} n \\ 1 \end{matrix} \right\} = \left\{ \begin{matrix} n \\ n \end{matrix} \right\} = 1,$ |
| 14. $\begin{bmatrix} n \\ 1 \end{bmatrix} = (n-1)!,$ | 15. $\begin{bmatrix} n \\ 2 \end{bmatrix} = (n-1)!H_{n-1},$ | 16. $\begin{bmatrix} n \\ n \end{bmatrix} = 1, \quad 17. \begin{bmatrix} n \\ k \end{bmatrix} \geq \left\{ \begin{matrix} n \\ k \end{matrix} \right\},$ |
| 18. $\begin{bmatrix} n \\ k \end{bmatrix} = (n-1) \begin{bmatrix} n-1 \\ k \end{bmatrix} + \begin{bmatrix} n-1 \\ k-1 \end{bmatrix},$ | 19. $\left\{ \begin{matrix} n \\ n-1 \end{matrix} \right\} = \begin{bmatrix} n \\ n-1 \end{bmatrix} = \binom{n}{2},$ | 20. $\sum_{k=0}^n \begin{bmatrix} n \\ k \end{bmatrix} = n!, \quad 21. C_n = \frac{1}{n+1} \binom{2n}{n},$ |
| 22. $\langle \begin{matrix} n \\ 0 \end{matrix} \rangle = \langle \begin{matrix} n \\ n-1 \end{matrix} \rangle = 1,$ | 23. $\langle \begin{matrix} n \\ k \end{matrix} \rangle = \langle \begin{matrix} n \\ n-1-k \end{matrix} \rangle,$ | 24. $\langle \begin{matrix} n \\ k \end{matrix} \rangle = (k+1) \langle \begin{matrix} n-1 \\ k \end{matrix} \rangle + (n-k) \langle \begin{matrix} n-1 \\ k-1 \end{matrix} \rangle,$ |
| 25. $\langle \begin{matrix} 0 \\ k \end{matrix} \rangle = \begin{cases} 1 & \text{if } k=0, \\ 0 & \text{otherwise} \end{cases}$ | 26. $\langle \begin{matrix} n \\ 1 \end{matrix} \rangle = 2^n - n - 1,$ | 27. $\langle \begin{matrix} n \\ 2 \end{matrix} \rangle = 3^n - (n+1)2^n + \binom{n+1}{2},$ |
| 28. $x^n = \sum_{k=0}^n \langle \begin{matrix} n \\ k \end{matrix} \rangle \binom{x+k}{n},$ | 29. $\langle \begin{matrix} n \\ m \end{matrix} \rangle = \sum_{k=0}^m \binom{n+1}{k} (m+1-k)^n (-1)^k,$ | 30. $m! \left\{ \begin{matrix} n \\ m \end{matrix} \right\} = \sum_{k=0}^n \langle \begin{matrix} n \\ k \end{matrix} \rangle \binom{k}{n-m},$ |
| 31. $\langle \begin{matrix} n \\ m \end{matrix} \rangle = \sum_{k=0}^n \left\{ \begin{matrix} n \\ k \end{matrix} \right\} \binom{n-k}{m} (-1)^{n-k-m} k!,$ | 32. $\langle \langle \begin{matrix} n \\ 0 \end{matrix} \rangle \rangle = 1,$ | 33. $\langle \langle \begin{matrix} n \\ n \end{matrix} \rangle \rangle = 0 \quad \text{for } n \neq 0,$ |
| 34. $\langle \langle \begin{matrix} n \\ k \end{matrix} \rangle \rangle = (k+1) \langle \langle \begin{matrix} n-1 \\ k \end{matrix} \rangle \rangle + (2n-1-k) \langle \langle \begin{matrix} n-1 \\ k-1 \end{matrix} \rangle \rangle,$ | 35. $\sum_{k=0}^n \langle \langle \begin{matrix} n \\ k \end{matrix} \rangle \rangle = \frac{(2n)n}{2^n},$ | 36. $\left\{ \begin{matrix} x \\ x-n \end{matrix} \right\} = \sum_{k=0}^n \langle \langle \begin{matrix} n \\ k \end{matrix} \rangle \rangle \binom{x+n-1-k}{2n},$ |
| 37. $\left\{ \begin{matrix} n+1 \\ m+1 \end{matrix} \right\} = \sum_k \binom{n}{k} \left\{ \begin{matrix} k \\ m \end{matrix} \right\} = \sum_{k=0}^n \left\{ \begin{matrix} k \\ m \end{matrix} \right\} (m+1)^{n-k},$ | | |

| | | |
|--|--|---|
| 38. $\begin{bmatrix} n+1 \\ m+1 \end{bmatrix} = \sum_k \begin{bmatrix} n \\ k \end{bmatrix} \begin{bmatrix} k \\ m \end{bmatrix} = \sum_{k=0}^n \begin{bmatrix} k \\ m \end{bmatrix} n^{n-k} = n! \sum_{k=0}^n \frac{1}{k!} \begin{bmatrix} k \\ m \end{bmatrix},$ | 39. $\begin{bmatrix} x \\ x-n \end{bmatrix} = \sum_{k=0}^n \left\langle \begin{bmatrix} n \\ k \end{bmatrix} \right\rangle \begin{bmatrix} x+k \\ 2n \end{bmatrix},$ | Every tree with n vertices has $n-1$ edges. Kraft inequality: If the depths of the leaves of a binary tree are d_1, \dots, d_n : $\sum_{i=1}^n 2^{-d_i} \leq 1,$ and equality holds only if every internal node has 2 sons. |
| 40. $\left\{ \begin{matrix} n \\ m \end{matrix} \right\} = \sum_k \binom{n}{k} \left\{ \begin{matrix} k+1 \\ m+1 \end{matrix} \right\} (-1)^{n-k},$ | 41. $\begin{bmatrix} n \\ m \end{bmatrix} = \sum_k \begin{bmatrix} n+1 \\ k+1 \end{bmatrix} \begin{bmatrix} k \\ m \end{bmatrix} (-1)^{m-k},$ | |
| 42. $\left\{ \begin{matrix} m+n+1 \\ m \end{matrix} \right\} = \sum_{k=0}^m k \left\{ \begin{matrix} n+k \\ k \end{matrix} \right\},$ | 43. $\begin{bmatrix} m+n+1 \\ m \end{bmatrix} = \sum_{k=0}^m k(n+k) \begin{bmatrix} n+k \\ k \end{bmatrix},$ | |
| 44. $\binom{n}{m} = \sum_k \left\{ \begin{matrix} n+1 \\ k+1 \end{matrix} \right\} \begin{bmatrix} k \\ m \end{bmatrix} (-1)^{m-k},$ | 45. $(n-m)! \binom{n}{m} = \sum_k \begin{bmatrix} n+1 \\ k+1 \end{bmatrix} \left\{ \begin{matrix} k \\ m \end{matrix} \right\} (-1)^{m-k}, \text{ for } n \geq m,$ | |
| 46. $\left\{ \begin{matrix} n \\ n-m \end{matrix} \right\} = \sum_k \binom{m-n}{m+k} \binom{m+n}{n+k} \begin{bmatrix} m+k \\ k \end{bmatrix},$ | 47. $\begin{bmatrix} n \\ n-m \end{bmatrix} = \sum_k \binom{m-n}{m+k} \binom{m+n}{n+k} \left\{ \begin{matrix} m+k \\ k \end{matrix} \right\},$ | |
| 48. $\left\{ \begin{matrix} n \\ \ell+m \end{matrix} \right\} \binom{\ell+m}{\ell} = \sum_k \left\{ \begin{matrix} k \\ \ell \end{matrix} \right\} \left\{ \begin{matrix} n-k \\ m \end{matrix} \right\} \binom{n}{k},$ | 49. $\begin{bmatrix} n \\ \ell+m \end{bmatrix} \binom{\ell+m}{\ell} = \sum_k \begin{bmatrix} k \\ \ell \end{bmatrix} \begin{bmatrix} n-k \\ m \end{bmatrix} \binom{n}{k}.$ | |

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

$$u_{i+1} = \frac{1}{2} + u_i, \quad u_1 = \frac{1}{2},$$

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”

$$1(T(n) - 3T(n/2)) = n$$

$$3(T(n/2) - 3T(n/4)) = n/2$$

$$\vdots \quad \vdots \quad \vdots$$

$$3^{\log_2 n - 1}(T(2) - 3T(1)) = 2$$

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

$$\sum_{i=0}^{m-1} \frac{n}{2^i} 3^i = n \sum_{i=0}^{m-1} \left(\frac{3}{2}\right)^i.$$

Let $c = \frac{3}{2}$. Then we have

$$n \sum_{i=0}^{m-1} c^i = n \left(\frac{c^m - 1}{c - 1} \right)$$

$$= 2n(c^{\log_2 n} - 1)$$

$$= 2n(c^{(k-1)\log_2 n} - 1)$$

$$= 2n^k - 2n,$$

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 $G(x)$:

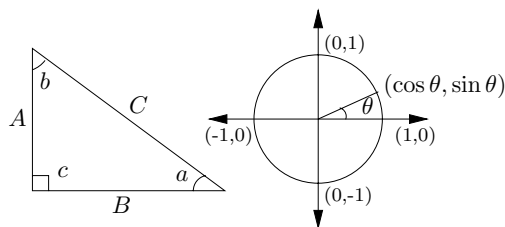
$$G(x) = \frac{x}{(1-x)(1-2x)}.$$

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 $g_i = 2^i - 1$.

| $n \sim 0.11100,$ | | | $\psi = 2 \sim 1.01000,$ | | |
|-------------------------------------|---------------|-------|--|--|---|
| $\psi = 2 \sim 0.01000$ | | | | | |
| i | 2^i | p_i | General | | Probability |
| 1 | 2 | 2 | Bernoulli Numbers ($B_i = 0$, odd $i \neq 1$): | | Continuous distributions: If |
| 2 | 4 | 3 | $B_0 = 1, B_1 = -\frac{1}{2}, B_2 = \frac{1}{6}, B_4 = -\frac{1}{30},$ | | $\Pr[a < X < b] = \int_a^b p(x) dx,$ |
| 3 | 8 | 5 | $B_6 = \frac{1}{42}, B_8 = -\frac{1}{30}, B_{10} = \frac{5}{66}.$ | | then p is the probability density function of X . If |
| 4 | 16 | 7 | Change of base, quadratic formula: | | $\Pr[X < a] = P(a),$ |
| 5 | 32 | 11 | $\log_b x = \frac{\log_a x}{\log_a b}, \quad \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}.$ | | then P is the distribution function of X . If P and p both exist then |
| 6 | 64 | 13 | Euler's number e : | | $P(a) = \int_{-\infty}^a p(x) dx.$ |
| 7 | 128 | 17 | $e = 1 + \frac{1}{2} + \frac{1}{6} + \frac{1}{24} + \frac{1}{120} + \dots$ | | Expectation: If X is discrete |
| 8 | 256 | 19 | $\lim_{n \rightarrow \infty} \left(1 + \frac{x}{n}\right)^n = e^x.$ | | $E[g(X)] = \sum_x g(x) \Pr[X = x].$ |
| 9 | 512 | 23 | $\left(1 + \frac{1}{n}\right)^n < e < \left(1 + \frac{1}{n}\right)^{n+1}.$ | | If X continuous then |
| 10 | 1,024 | 29 | $\left(1 + \frac{1}{n}\right)^n = e - \frac{e}{2n} + \frac{11e}{24n^2} - O\left(\frac{1}{n^3}\right).$ | | $E[g(X)] = \int_{-\infty}^{\infty} g(x)p(x) dx = \int_{-\infty}^{\infty} g(x) dP(x).$ |
| 11 | 2,048 | 31 | Harmonic numbers: | | Variance, standard deviation: |
| 12 | 4,096 | 37 | $1, \frac{3}{2}, \frac{11}{6}, \frac{25}{12}, \frac{137}{60}, \frac{49}{20}, \frac{363}{140}, \frac{761}{280}, \frac{7129}{2520}, \dots$ | | $\text{VAR}[X] = E[X^2] - E[X]^2,$ |
| 13 | 8,192 | 41 | $\ln n < H_n < \ln n + 1,$ | | $\sigma = \sqrt{\text{VAR}[X]}.$ |
| 14 | 16,384 | 43 | $H_n = \ln n + \gamma + O\left(\frac{1}{n}\right).$ | | For events A and B : |
| 15 | 32,768 | 47 | Factorial, Stirling's approximation: | | $\Pr[A \vee B] = \Pr[A] + \Pr[B] - \Pr[A \wedge B]$ |
| 16 | 65,536 | 53 | $1, 2, 6, 24, 120, 720, 5040, 40320, 362880, \dots$ | | $\Pr[A \wedge B] = \Pr[A] \cdot \Pr[B],$ |
| 17 | 131,072 | 59 | $n! = \sqrt{2\pi n} \left(\frac{n}{e}\right)^n \left(1 + \Theta\left(\frac{1}{n}\right)\right).$ | | iff A and B are independent. |
| 18 | 262,144 | 61 | Ackermann's function and inverse: | | $\Pr[A B] = \frac{\Pr[A \wedge B]}{\Pr[B]}$ |
| 19 | 524,288 | 67 | $a(i, j) = \begin{cases} 2^j & i = 1 \\ a(i-1, 2) & j = 1 \\ a(i-1, a(i, j-1)) & i, j \geq 2 \end{cases}$ | | For random variables X and Y : |
| 20 | 1,048,576 | 71 | $\alpha(i) = \min\{j \mid a(j, j) \geq i\}.$ | | $E[X \cdot Y] = E[X] \cdot E[Y],$ |
| 21 | 2,097,152 | 73 | Binomial distribution: | | if X and Y are independent. |
| 22 | 4,194,304 | 79 | $\Pr[X = k] = \binom{n}{k} p^k q^{n-k}, \quad q = 1 - p,$ | | $E[X + Y] = E[X] + E[Y],$ |
| 23 | 8,388,608 | 83 | $E[X] = \sum_{k=1}^n k \binom{n}{k} p^k q^{n-k} = np.$ | | $E[cX] = cE[X].$ |
| 24 | 16,777,216 | 89 | Poisson distribution: | | Bayes' theorem: |
| 25 | 33,554,432 | 97 | $\Pr[X = k] = \frac{e^{-\lambda} \lambda^k}{k!}, \quad E[X] = \lambda.$ | | $\Pr[A_i B] = \frac{\Pr[B A_i] \Pr[A_i]}{\sum_{j=1}^n \Pr[A_j] \Pr[B A_j]}.$ |
| 26 | 67,108,864 | 101 | Normal (Gaussian) distribution: | | Inclusion-exclusion: |
| 27 | 134,217,728 | 103 | $p(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-(x-\mu)^2/2\sigma^2}, \quad E[X] = \mu.$ | | $\Pr\left[\bigvee_{i=1}^n X_i\right] = \sum_{i=1}^n \Pr[X_i] +$ |
| 28 | 268,435,456 | 107 | 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 | | $\sum_{k=2}^n (-1)^{k+1} \sum_{i_1 < \dots < i_k} \Pr\left[\bigwedge_{j=1}^k X_{i_j}\right].$ |
| 29 | 536,870,912 | 109 | $nH_n.$ | | Moment inequalities: |
| 30 | 1,073,741,824 | 113 | | | $\Pr[X \geq \lambda E[X]] \leq \frac{1}{\lambda},$ |
| 31 | 2,147,483,648 | 127 | | | $\Pr[X - E[X] \geq \lambda \cdot \sigma] \leq \frac{1}{\lambda^2}.$ |
| 32 | 4,294,967,296 | 131 | | | Geometric distribution: |
| Pascal's Triangle | | | | | $\Pr[X = k] = pq^{k-1}, \quad q = 1 - p,$ |
| 1 | | | | | $E[X] = \sum_{k=1}^{\infty} kpq^{k-1} = \frac{1}{p}.$ |
| 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 | | | | | |



Pythagorean theorem:

$$C^2 = A^2 + B^2.$$

Definitions:

$$\begin{aligned} \sin a &= A/C, & \cos a &= B/C, \\ \csc a &= C/A, & \sec a &= C/B, \\ \tan a &= \frac{\sin a}{\cos a} = \frac{A}{B}, & \cot a &= \frac{\cos a}{\sin a} = \frac{B}{A}. \end{aligned}$$

Area, radius of inscribed circle:

$$\frac{1}{2}AB, \quad \frac{AB}{A+B+C}.$$

Identities:

$$\begin{aligned} \sin x &= \frac{1}{\csc x}, & \cos x &= \frac{1}{\sec x}, \\ \tan x &= \frac{1}{\cot x}, & \sin^2 x + \cos^2 x &= 1, \\ 1 + \tan^2 x &= \sec^2 x, & 1 + \cot^2 x &= \csc^2 x, \\ \sin x &= \cos\left(\frac{\pi}{2} - x\right), & \sin x &= \sin(\pi - x), \\ \cos x &= -\cos(\pi - x), & \tan x &= \cot\left(\frac{\pi}{2} - x\right), \\ \cot x &= -\cot(\pi - x), & \csc x &= \cot\frac{x}{2} - \cot x, \\ \sin(x \pm y) &= \sin x \cos y \pm \cos x \sin y, \\ \cos(x \pm y) &= \cos x \cos y \mp \sin x \sin y, \\ \tan(x \pm y) &= \frac{\tan x \pm \tan y}{1 \mp \tan x \tan y}, \\ \cot(x \pm y) &= \frac{\cot x \cot y \mp 1}{\cot x \pm \cot y}, \\ \sin 2x &= 2 \sin x \cos x, & \sin 2x &= \frac{2 \tan x}{1 + \tan^2 x}, \\ \cos 2x &= \cos^2 x - \sin^2 x, & \cos 2x &= 2 \cos^2 x - 1, \\ \cos 2x &= 1 - 2 \sin^2 x, & \cos 2x &= \frac{1 - \tan^2 x}{1 + \tan^2 x}, \\ \tan 2x &= \frac{2 \tan x}{1 - \tan^2 x}, & \cot 2x &= \frac{\cot^2 x - 1}{2 \cot x}, \\ \sin(x + y) \sin(x - y) &= \sin^2 x - \sin^2 y, \\ \cos(x + y) \cos(x - y) &= \cos^2 x - \sin^2 y. \end{aligned}$$

Euler's equation:

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

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

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.

$$\det A \cdot B = \det A \cdot \det B,$$

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

2×2 and 3×3 determinant:

$$\begin{aligned} \begin{vmatrix} a & b \\ c & d \end{vmatrix} &= ad - bc, \\ \begin{vmatrix} a & b & c \\ d & e & f \\ g & h & i \end{vmatrix} &= g \begin{vmatrix} b & c \\ e & f \end{vmatrix} - h \begin{vmatrix} a & c \\ d & f \end{vmatrix} + i \begin{vmatrix} a & b \\ d & e \end{vmatrix} \\ &= aei + bfg + cdh \\ &\quad - ceg - fha - ibd. \end{aligned}$$

Permanents:

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

Hyperbolic Functions

Definitions:

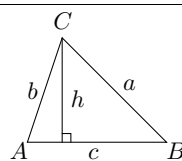
$$\begin{aligned} \sinh x &= \frac{e^x - e^{-x}}{2}, & \cosh x &= \frac{e^x + e^{-x}}{2}, \\ \tanh x &= \frac{e^x - e^{-x}}{e^x + e^{-x}}, & \text{csch } x &= \frac{1}{\sinh x}, \\ \text{sech } x &= \frac{1}{\cosh x}, & \coth x &= \frac{1}{\tanh x}. \end{aligned}$$

Identities:

$$\begin{aligned} \cosh^2 x - \sinh^2 x &= 1, & \tanh^2 x + \text{sech}^2 x &= 1, \\ \coth^2 x - \text{csch}^2 x &= 1, & \sinh(-x) &= -\sinh x, \\ \cosh(-x) &= \cosh x, & \tanh(-x) &= -\tanh x, \\ \sinh(x + y) &= \sinh x \cosh y + \cosh x \sinh y, \\ \cosh(x + y) &= \cosh x \cosh y + \sinh x \sinh y, \\ \sinh 2x &= 2 \sinh x \cosh x, \\ \cosh 2x &= \cosh^2 x + \sinh^2 x, \\ \cosh x + \sinh x &= e^x, & \cosh x - \sinh x &= e^{-x}, \\ (\cosh x + \sinh x)^n &= \cosh nx + \sinh nx, & n \in \mathbb{Z}, \\ 2 \sinh^2 \frac{x}{2} &= \cosh x - 1, & 2 \cosh^2 \frac{x}{2} &= \cosh x + 1. \end{aligned}$$

| 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 under-
stand things, you
just get used to
them.
- J. von Neumann



Law of cosines:

$$c^2 = a^2 + b^2 - 2ab \cos C.$$

Area:

$$\begin{aligned} A &= \frac{1}{2}hc, \\ &= \frac{1}{2}ab \sin C, \\ &= \frac{c^2 \sin A \sin B}{2 \sin C}. \end{aligned}$$

Heron's formula:

$$\begin{aligned} A &= \sqrt{s \cdot s_a \cdot s_b \cdot s_c}, \\ s &= \frac{1}{2}(a + b + c), \\ s_a &= s - a, \\ s_b &= s - b, \\ s_c &= s - c. \end{aligned}$$

More identities:

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

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

$$C \equiv r_1 \pmod{m_1}$$

$$\vdots \quad \vdots \quad \vdots$$

$$C \equiv r_n \pmod{m_n}$$

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

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

Fermat's theorem:

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

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

$$(n-1)! \equiv -1 \pmod{n}.$$

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:

$$p_n = n \ln n + n \ln \ln n - n + n \frac{\ln \ln n}{\ln n}$$

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

$$\pi(n) = \frac{n}{\ln n} + \frac{n}{(\ln n)^2} + \frac{2!n}{(\ln n)^3}$$

$$+ O\left(\frac{n}{(\ln n)^4}\right).$$

Definitions:

Loop An edge connecting a vertex to itself.

Directed Simple Each edge has a direction. Graph with no loops or multi-edges.

Walk A sequence $v_0 e_1 v_1 \dots e_\ell v_\ell$.

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

$r(k, \ell)$ 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

$$(x, y) \quad (x, y, 1)$$

$$y = mx + b \quad (m, -1, b)$$

$$x = c \quad (1, 0, -c)$$

Distance formula, L_p and L_∞ metric:

$$\sqrt{(x_1 - x_0)^2 + (y_1 - y_0)^2},$$

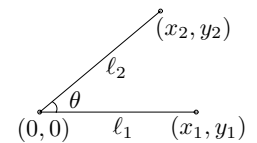
$$[|x_1 - x_0|^p + |y_1 - y_0|^p]^{1/p},$$

$$\lim_{p \rightarrow \infty} [|x_1 - x_0|^p + |y_1 - y_0|^p]^{1/p}.$$

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

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

Angle formed by three points:



$$\cos \theta = \frac{(x_1, y_1) \cdot (x_2, y_2)}{l_1 l_2}.$$

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:

$$A = \pi r^2, \quad V = \frac{4}{3} \pi r^3.$$

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

– Issac Newton

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:

$$\frac{\pi}{4} = 1 + \frac{1^2}{2 + \frac{3^2}{2 + \frac{5^2}{2 + \frac{7^2}{2 + \cdots}}}}$$

Gregory's series:

$$\frac{\pi}{4} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \cdots$$

Newton's series:

$$\frac{\pi}{6} = \frac{1}{2} + \frac{1}{2 \cdot 3 \cdot 2^3} + \frac{1 \cdot 3}{2 \cdot 4 \cdot 5 \cdot 2^5} + \cdots$$

Sharp's series:

$$\frac{\pi}{6} = \frac{1}{\sqrt{3}} \left(1 - \frac{1}{3^1 \cdot 3} + \frac{1}{3^2 \cdot 5} - \frac{1}{3^3 \cdot 7} + \cdots \right)$$

Euler's series:

$$\frac{\pi^2}{6} = \frac{1}{1^2} + \frac{1}{2^2} + \frac{1}{3^2} + \frac{1}{4^2} + \frac{1}{5^2} + \cdots$$

$$\frac{\pi^2}{8} = \frac{1}{1^2} + \frac{1}{3^2} + \frac{1}{5^2} + \frac{1}{7^2} + \frac{1}{9^2} + \cdots$$

$$\frac{\pi^2}{12} = \frac{1}{1^2} - \frac{1}{2^2} + \frac{1}{3^2} - \frac{1}{4^2} + \frac{1}{5^2} - \cdots$$

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

Derivatives:

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Integrals:

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

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

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

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

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

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

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

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. $\int \sin^2(ax) dx = \frac{1}{2a}(ax - \sin(ax) \cos(ax)),$
18. $\int \cos^2(ax) dx = \frac{1}{2a}(ax + \sin(ax) \cos(ax)),$
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. $\int \operatorname{arccosh} \frac{x}{a} dx = \begin{cases} x \operatorname{arccosh} \frac{x}{a} - \sqrt{x^2 + a^2}, & \text{if } \operatorname{arccosh} \frac{x}{a} > 0 \text{ and } a > 0, \\ x \operatorname{arccosh} \frac{x}{a} + \sqrt{x^2 + a^2}, & \text{if } \operatorname{arccosh} \frac{x}{a} < 0 \text{ and } a > 0, \end{cases}$
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. $\int \frac{dx}{x \sqrt{x^2 + a^2}} = \frac{1}{a} \ln \left| \frac{x}{a + \sqrt{a^2 + x^2}} \right|,$

$$\begin{aligned}
62. \int \frac{dx}{x\sqrt{x^2-a^2}} &= \frac{1}{a} \arccos \frac{a}{|x|}, \quad a > 0, & 63. \int \frac{dx}{x^2\sqrt{x^2 \pm a^2}} &= \mp \frac{\sqrt{x^2 \pm a^2}}{a^2 x}, \\
64. \int \frac{x dx}{\sqrt{x^2 \pm a^2}} &= \sqrt{x^2 \pm a^2}, & 65. \int \frac{\sqrt{x^2 \pm a^2}}{x^4} dx &= \mp \frac{(x^2 + a^2)^{3/2}}{3a^2 x^3}, \\
66. \int \frac{dx}{ax^2 + bx + c} &= \begin{cases} \frac{1}{\sqrt{b^2 - 4ac}} \ln \left| \frac{2ax + b - \sqrt{b^2 - 4ac}}{2ax + b + \sqrt{b^2 - 4ac}} \right|, & \text{if } b^2 > 4ac, \\ \frac{2}{\sqrt{4ac - b^2}} \arctan \frac{2ax + b}{\sqrt{4ac - b^2}}, & \text{if } b^2 < 4ac, \end{cases} \\
67. \int \frac{dx}{\sqrt{ax^2 + bx + c}} &= \begin{cases} \frac{1}{\sqrt{a}} \ln \left| 2ax + b + 2\sqrt{a}\sqrt{ax^2 + bx + c} \right|, & \text{if } a > 0, \\ \frac{1}{\sqrt{-a}} \arcsin \frac{-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{4ac - 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} \frac{-1}{\sqrt{c}} \ln \left| \frac{2\sqrt{c}\sqrt{ax^2 + bx + c} + bx + 2c}{x} \right|, & \text{if } c > 0, \\ \frac{1}{\sqrt{-c}} \arcsin \frac{bx + 2c}{|x|\sqrt{b^2 - 4ac}}, & \text{if } c < 0, \end{cases} \\
71. \int x^3 \sqrt{x^2 + a^2} dx &= \left(\frac{1}{3}x^2 - \frac{2}{15}a^2\right)(x^2 + a^2)^{3/2}, \\
72. \int x^n \sin(ax) dx &= -\frac{1}{a}x^n \cos(ax) + \frac{n}{a} \int x^{n-1} \cos(ax) dx, \\
73. \int x^n \cos(ax) dx &= \frac{1}{a}x^n \sin(ax) - \frac{n}{a} \int x^{n-1} \sin(ax) dx, \\
74. \int x^n e^{ax} dx &= \frac{x^n e^{ax}}{a} - \frac{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.
\end{aligned}$$

$$\begin{aligned}
x^1 &= x^1 & x^{\bar{1}} &= x^{\bar{1}} \\
x^2 &= x^2 + x^1 & x^{\bar{2}} &= x^{\bar{2}} - x^{\bar{1}} \\
x^3 &= x^3 + 3x^2 + x^1 & x^{\bar{3}} &= x^{\bar{3}} - 3x^{\bar{2}} + x^{\bar{1}} \\
x^4 &= x^4 + 6x^3 + 7x^2 + x^1 & x^{\bar{4}} &= x^{\bar{4}} - 6x^{\bar{3}} + 7x^{\bar{2}} - x^{\bar{1}} \\
x^5 &= x^5 + 15x^4 + 25x^3 + 10x^2 + x^1 & x^{\bar{5}} &= x^{\bar{5}} - 15x^{\bar{4}} + 25x^{\bar{3}} - 10x^{\bar{2}} + x^{\bar{1}} \\
x^{\bar{1}} &= x^1 & x^1 &= x^1 \\
x^{\bar{2}} &= x^2 + x^1 & x^2 &= x^2 - x^1 \\
x^{\bar{3}} &= x^3 + 3x^2 + 2x^1 & x^3 &= x^3 - 3x^2 + 2x^1 \\
x^{\bar{4}} &= x^4 + 6x^3 + 11x^2 + 6x^1 & x^4 &= x^4 - 6x^3 + 11x^2 - 6x^1 \\
x^{\bar{5}} &= x^5 + 10x^4 + 35x^3 + 50x^2 + 24x^1 & x^{\bar{5}} &= x^5 - 10x^4 + 35x^3 - 50x^2 + 24x^1
\end{aligned}$$

Difference, shift operators:

$$\Delta f(x) = f(x+1) - f(x),$$

$$\mathbb{E} f(x) = f(x+1).$$

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:

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

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

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

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

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

Sums:

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

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

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

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

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

Falling Factorial Powers:

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

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

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

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

Rising Factorial Powers:

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

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

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

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

Conversion:

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

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

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

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

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

$$x^{\underline{n}} = \sum_{k=1}^n \left[\begin{matrix} n \\ k \end{matrix} \right] (-1)^{n-k} x^k,$$

$$x^{\overline{n}} = \sum_{k=1}^n \left[\begin{matrix} n \\ k \end{matrix} \right] x^k.$$

Taylor's series:

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

Expansions:

$$\begin{aligned} \frac{1}{1-x} &= 1 + x + x^2 + x^3 + x^4 + \dots = \sum_{i=0}^{\infty} x^i, \\ \frac{1}{1-cx} &= 1 + cx + c^2x^2 + c^3x^3 + \dots = \sum_{i=0}^{\infty} c^i x^i, \\ \frac{1}{1-x^n} &= 1 + x^n + x^{2n} + x^{3n} + \dots = \sum_{i=0}^{\infty} x^{ni}, \\ \frac{x}{(1-x)^2} &= x + 2x^2 + 3x^3 + 4x^4 + \dots = \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 + \dots = \sum_{i=0}^{\infty} i^n x^i, \\ e^x &= 1 + x + \frac{1}{2}x^2 + \frac{1}{6}x^3 + \dots = \sum_{i=0}^{\infty} \frac{x^i}{i!}, \\ \ln(1+x) &= x - \frac{1}{2}x^2 + \frac{1}{3}x^3 - \frac{1}{4}x^4 + \dots = \sum_{i=1}^{\infty} (-1)^{i+1} \frac{x^i}{i}, \\ \ln \frac{1}{1-x} &= x + \frac{1}{2}x^2 + \frac{1}{3}x^3 + \frac{1}{4}x^4 + \dots = \sum_{i=1}^{\infty} \frac{x^i}{i}, \\ \sin x &= x - \frac{1}{3!}x^3 + \frac{1}{5!}x^5 - \frac{1}{7!}x^7 + \dots = \sum_{i=0}^{\infty} (-1)^i \frac{x^{2i+1}}{(2i+1)!}, \\ \cos x &= 1 - \frac{1}{2!}x^2 + \frac{1}{4!}x^4 - \frac{1}{6!}x^6 + \dots = \sum_{i=0}^{\infty} (-1)^i \frac{x^{2i}}{(2i)!}, \\ \tan^{-1} x &= x - \frac{1}{3}x^3 + \frac{1}{5}x^5 - \frac{1}{7}x^7 + \dots = \sum_{i=0}^{\infty} (-1)^i \frac{x^{2i+1}}{(2i+1)}, \\ (1+x)^n &= 1 + nx + \frac{n(n-1)}{2}x^2 + \dots = \sum_{i=0}^{\infty} \binom{n}{i} x^i, \\ \frac{1}{(1-x)^{n+1}} &= 1 + (n+1)x + \binom{n+2}{2}x^2 + \dots = \sum_{i=0}^{\infty} \binom{i+n}{i} x^i, \\ \frac{x}{e^x - 1} &= 1 - \frac{1}{2}x + \frac{1}{12}x^2 - \frac{1}{720}x^4 + \dots = \sum_{i=0}^{\infty} \frac{B_i x^i}{i!}, \\ \frac{1}{2x}(1 - \sqrt{1-4x}) &= 1 + x + 2x^2 + 5x^3 + \dots = \sum_{i=0}^{\infty} \frac{1}{i+1} \binom{2i}{i} x^i, \\ \frac{1}{\sqrt{1-4x}} &= 1 + x + 2x^2 + 6x^3 + \dots = \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 + \binom{4+n}{2}x^2 + \dots = \sum_{i=0}^{\infty} \binom{2i+n}{i} x^i, \\ \frac{1}{1-x} \ln \frac{1}{1-x} &= x + \frac{3}{2}x^2 + \frac{11}{6}x^3 + \frac{25}{12}x^4 + \dots = \sum_{i=1}^{\infty} H_i x^i, \\ \frac{1}{2} \left(\ln \frac{1}{1-x} \right)^2 &= \frac{1}{2}x^2 + \frac{3}{4}x^3 + \frac{11}{24}x^4 + \dots = \sum_{i=2}^{\infty} \frac{H_{i-1} x^i}{i}, \\ \frac{x}{1-x-x^2} &= x + x^2 + 2x^3 + 3x^4 + \dots = \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 + \dots = \sum_{i=0}^{\infty} F_{ni} x^i. \end{aligned}$$

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:

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

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

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

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

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

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

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

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

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

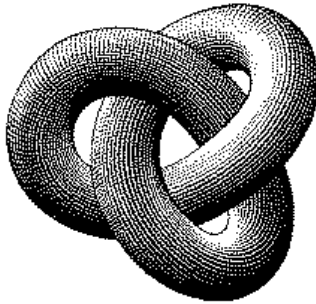
Summation: If $b_i = \sum_{j=0}^i a_j$ then

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

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

| | | | | |
|--|--|--|--|--|
| Expansions: | |  | | |
| $\frac{1}{(1-x)^{n+1}} \ln \frac{1}{1-x}$ | $= \sum_{i=0}^{\infty} (H_{n+i} - H_n) \binom{n+i}{i} x^i,$ | | $\left(\frac{1}{x}\right)^{-n}$ | $= \sum_{i=0}^{\infty} \left\{ \begin{matrix} i \\ n \end{matrix} \right\} x^i,$ |
| $x^{\overline{n}}$ | $= \sum_{i=0}^{\infty} \left[\begin{matrix} n \\ i \end{matrix} \right] x^i,$ | | $(e^x - 1)^n$ | $= \sum_{i=0}^{\infty} \left\{ \begin{matrix} i \\ n \end{matrix} \right\} \frac{n! x^i}{i!},$ |
| $\left(\ln \frac{1}{1-x}\right)^n$ | $= \sum_{i=0}^{\infty} \left[\begin{matrix} i \\ n \end{matrix} \right] \frac{n! x^i}{i!},$ | | $x \cot x$ | $= \sum_{i=0}^{\infty} \frac{(-4)^i B_{2i} x^{2i}}{(2i)!},$ |
| $\tan x$ | $= \sum_{i=1}^{\infty} (-1)^{i-1} \frac{2^{2i} (2^{2i} - 1) B_{2i} x^{2i-1}}{(2i)!},$ | $\zeta(x)$ | $= \sum_{i=1}^{\infty} \frac{1}{i^x},$ | |
| $\frac{1}{\zeta(x)}$ | $= \sum_{i=1}^{\infty} \frac{\mu(i)}{i^x},$ | $\frac{\zeta(x-1)}{\zeta(x)}$ | $= \sum_{i=1}^{\infty} \frac{\phi(i)}{i^x},$ | |
| $\zeta(x)$ | $= \prod_p \frac{1}{1 - p^{-x}},$ | | | |
| $\zeta^2(x)$ | $= \sum_{i=1}^{\infty} \frac{d(i)}{x^i} \quad \text{where } d(n) = \sum_{d n} 1,$ | Stieltjes Integration | | |
| $\zeta(x)\zeta(x-1)$ | $= \sum_{i=1}^{\infty} \frac{S(i)}{x^i} \quad \text{where } S(n) = \sum_{d n} d,$ | If G is continuous in the interval $[a, b]$ and F is nondecreasing then | | |
| $\zeta(2n)$ | $= \frac{2^{2n-1} B_{2n} }{(2n)!} \pi^{2n}, \quad n \in \mathbb{N},$ | $\int_a^b G(x) dF(x)$ | | |
| $\frac{x}{\sin x}$ | $= \sum_{i=0}^{\infty} (-1)^{i-1} \frac{(4^i - 2) B_{2i} x^{2i}}{(2i)!},$ | exists. If $a \leq b \leq c$ then | | |
| $\left(\frac{1 - \sqrt{1 - 4x}}{2x}\right)^n$ | $= \sum_{i=0}^{\infty} \frac{n(2i + n - 1)!}{i!(n + i)!} x^i,$ | $\int_a^c G(x) dF(x) = \int_a^b G(x) dF(x) + \int_b^c G(x) dF(x).$ | | |
| $e^x \sin x$ | $= \sum_{i=1}^{\infty} \frac{2^{i/2} \sin \frac{i\pi}{4}}{i!} x^i,$ | If the integrals involved exist | | |
| $\sqrt{\frac{1 - \sqrt{1 - x}}{x}}$ | $= \sum_{i=0}^{\infty} \frac{(4i)!}{16^i \sqrt{2} (2i)!(2i + 1)!} x^i,$ | $\int_a^b (G(x) + H(x)) dF(x) = \int_a^b G(x) dF(x) + \int_a^b H(x) dF(x),$ | | |
| $\left(\frac{\arcsin x}{x}\right)^2$ | $= \sum_{i=0}^{\infty} \frac{4^i i!^2}{(i + 1)(2i + 1)!} x^{2i}.$ | $\int_a^b G(x) d(F(x) + H(x)) = \int_a^b G(x) dF(x) + \int_a^b G(x) dH(x),$ | | |
| | | $\int_a^b c \cdot G(x) dF(x) = \int_a^b G(x) d(c \cdot F(x)) = c \int_a^b G(x) dF(x),$ | | |
| | | $\int_a^b G(x) dF(x) = G(b)F(b) - G(a)F(a) - \int_a^b F(x) dG(x).$ | | |
| | | 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 | | 00 47 18 76 29 93 85 34 61 52 | | Fibonacci Numbers |
| If we have equations: | | 86 11 57 28 70 39 94 45 02 63 | | 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, ... |
| $a_{1,1}x_1 + a_{1,2}x_2 + \cdots + a_{1,n}x_n = b_1$ | | 95 80 22 67 38 71 49 56 13 04 | | Definitions: |
| $a_{2,1}x_1 + a_{2,2}x_2 + \cdots + a_{2,n}x_n = b_2$ | | 59 96 81 33 07 48 72 60 24 15 | | $F_i = F_{i-1} + F_{i-2}, \quad F_0 = F_1 = 1,$ |
| \vdots | | 73 69 90 82 44 17 58 01 35 26 | | $F_{-i} = (-1)^{i-1} F_i,$ |
| \vdots | | 68 74 09 91 83 55 27 12 46 30 | | $F_i = \frac{1}{\sqrt{5}} \left(\phi^i - \hat{\phi}^i \right),$ |
| $a_{n,1}x_1 + a_{n,2}x_2 + \cdots + a_{n,n}x_n = b_n$ | | 37 08 75 19 92 84 66 23 50 41 | | Cassini's identity: for $i > 0$: |
| 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 | | 14 25 36 40 51 62 03 77 88 99 | | $F_{i+1}F_{i-1} - F_i^2 = (-1)^i.$ |
| $x_i = \frac{\det A_i}{\det A}.$ | | 21 32 43 54 65 06 10 89 97 78 | | Additive rule: |
| | | 42 53 64 05 16 20 31 98 79 87 | | $F_{n+k} = F_k F_{n+1} + F_{k-1} F_n,$ |
| | | | | $F_{2n} = F_n F_{n+1} + F_{n-1} F_n.$ |
| Improvement makes strait roads, but the crooked roads without Improvement, are roads of Genius. | | The Fibonacci number system: | | Calculation by matrices: |
| – William Blake (The Marriage of Heaven and Hell) | | Every integer n has a unique representation | | $\begin{pmatrix} F_{n-2} & F_{n-1} \\ F_{n-1} & F_n \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}^n.$ |
| | | $n = F_{k_1} + F_{k_2} + \cdots + F_{k_m},$ | | |
| | | where $k_i \geq k_{i+1} + 2$ for all $i,$ | | |
| | | $1 \leq i < m$ and $k_m \geq 2.$ | | |

