NYU ICPC Team Codebook

Compiled for the 2012 GNYR

October 24, 2012

This collaborative document is the central place for the algorithms you will need for the ACM ICPC programming contest.

Contents

The Ritual

When Choosing a Problem

* Find out which balloons are the popular ones! * Pick one with a nice, clean solution that you are totally convinced will work to do first.

Before Designing Your Solution

* Highlight the important information on the problem statement - input bounds, special rules, formatting, etc. * Look for code in this notebook that you can use! * Convince yourself that your algorithm will run with time to spare on the biggest input. * Create several test cases that you will use, especially for special or boundary cases.

Prior to Submitting

* Check maximum input, zero input, and other degenerate test cases. * Cross check with team mates' supplementary test cases. * Read the problem output specification one more time - your program's output behaviour is fresh in your mind. * Does your program work with negative numbers? * Make sure that your program is reading from an appropriate input file. * Check all variable initialisation, array bounds, and loop variables (i vs j, m vs n, etc.). * Finally, run a diff on the provided sample output and your program's output. * And don't forget to submit your solution under the correct problem number!

After Submitting

* Immediately print a copy of your source. * Staple the solution to the problem statement and keep them safe. Do not lose them!

If It Doesn't Work...

* Remember that a run-time error can be division by zero. * If the solution is not complex, allow a team mate to start the problem afresh. * Don't waste a lot of time - it's not shameful to simply give up!!!

Skeleton File

```
//COMPILING & RUNNING: javac Main.java && java Main < input.txt

import java.util.*;

public class Main {

    public static Scanner in;

    public static void main(String[] args) {
        in = new Scanner(System.in);

        doStuff();
    }

    public static void doStuff() {
        int N = in.nextInt();

        for(int i = 0;i < N; i++) {
            solve();
        }
    }

    public static void solve() {

    }
}
```

Geometry

```
/*
 * PtD - Point class
 * Required for most of the geometric algorithms below.
 * convex hull, 3 point circle,
 * intersection of circles, centroid/area of a polygon
 * @author Darko Aleksic
 */
class PtD {
  public static final double EPS = 1e-9;
  double x, y;

  /* add hashCode() and equals() if needed */
  PtD(double x, double y) {
    this.x = x;
    this.y = y;
  }

  public double dist(PtD p) {
    return Math.sqrt(distSquared(p));
  }

  public double distSquared(PtD p) {
    double dx = x - p.x;
    double dy = y - p.y;
    return dx * dx + dy * dy;
  }

  // For debugging.
  public String toString(){
    return "(" + x + "," + y + ")";
  }

  public int compareTo(PtD p2) {
    if (Math.abs(y - p2.y) < EPS) {
      if (Math.abs(x - p2.x) < EPS)
        return 0;
      if (x < p2.x)
        return -1;
      return 1;
    }
    if (y < p2.y)
      return -1;
    return 1;
  }

  public int compareTo(PtD p2, PtD pivot) {
    if (Math.abs(y - pivot.y) < EPS && Math.abs(y - p2.y) < EPS) {
      if (Math.abs(x - p2.x) < EPS)
        return 0;
      if (x > p2.x) // !!
```

```
      return -1;
    return 1;
  }
  double k = sub(pivot).cross(p2.sub(pivot));
  if (Math.abs(k) < EPS) {
    double d = distSquared(pivot) - p2.distSquared(pivot);
    if (Math.abs(d) < EPS)
      return 0;
    if (d < 0)
      return -1;
    return 1;
  }
  if (k < 0)
    return -1;
  return 1;
}

public PtD sub(PtD p2) {
  return new PtD(x - p2.x, y - p2.y);
}

public double dot(PtD p2) {
  return x * p2.x + y * p2.y;
}

public double cross(PtD p2) {
  return x * p2.y - p2.x * y;
}



/* centroid - they must be in order
 * (CW or CCW, does not matter)
 */
public static PtD centroid(PtD[] p, int n) {
  PtD c = new PtD(0, 0);
  int i, j;
  double sum = 0;
  double area = 0;
  for (i = n - 1, j = 0; j < n; i = j++) {
    area = p[i].x * p[j].y - p[i].y * p[j].x;
    sum += area;
    c.x += (p[i].x + p[j].x) * area;
    c.y += (p[i].y + p[j].y) * area;
  }
  sum *= 3.0;
  c.x /= sum;
  c.y /= sum;
  return c;
}
```

```
/* signed area of a polygon */
public static double signedArea(PtD[] p, int n) {
  double sum = 0;
  for (int i = n - 1, j = 0; j < n; i = j++) {
    sum += p[i].x * p[j].y - p[i].y * p[j].x;
  }
  return 0.5 * sum;
}
```

```
/**
 * Circle through three points
 *
 * @return a[]={x,y,r}, null if colinear
 */
public static double[] circleThroughThreePoints(PtD A, PtD B, PtD C) {
  double[] a = new double[3];
  double area = 0.5 * ((B.x - A.x) * (C.y - A.y) - (C.x - A.x)
      * (B.y - A.y));
  if (Math.abs(area) < EPS)
    return null;
  double lbaSqr = (B.x - A.x) * (B.x - A.x) + (B.y - A.y) * (B.y - A.y);
  double lcaSqr = (C.x - A.x) * (C.x - A.x) + (C.y - A.y) * (C.y - A.y);
  a[0] = A.x + ((C.y - A.y) * lbaSqr - ((B.y - A.y) * lcaSqr))
      / (4 * area);
  a[1] = A.y + ((B.x - A.x) * lcaSqr - ((C.x - A.x) * lbaSqr))
      / (4 * area);
  a[2] = Math.sqrt((a[0] - A.x) * (a[0] - A.x) + (a[1] - A.y)
      * (a[1] - A.y));
  return a;
}
```

```
/**
 * Intersection of circles. PtD needs sub[traction](), dot() and EPS defined
 *
 * @param p0
 *      Center of 1st circle
 * @param p1
 *      Center of 2nd circle
 * @param r0
 *      radius of 1st circle
 * @param r1
 *      radius of 2nd circle
 * @param a
 *      array that will hold the points (if any)
 * @return number of intersection points (-1 means infinity)
 */
public static int circleIntersection(PtD p0, PtD p1, double r0, double r1,
```

```
      PtD[] a) {
    PtD U = p1.sub(p0);
    PtD V = new PtD(U.y, -U.x);
    double duSqr = U.dot(U);
    double du = Math.sqrt(duSqr);
    if (Math.abs(U.x) < EPS && Math.abs(U.y) < EPS
        && Math.abs(r0 - r1) < EPS) {
      return -1; // same circles
    }
    if (Math.abs(du - (r0 + r1)) < EPS) {
      // one point from outside
      double cc = r0 / (r0 + r1);
      a[0] = new PtD(p0.x + cc * U.x, p0.y + cc * U.y);
      return 1;
    }
    if (Math.abs(du - Math.abs(r0 - r1)) < EPS) {
      // one point from inside
      double cc = r0 / (r0 - r1);
      a[0] = new PtD(p0.x + cc * U.x, p0.y + cc * U.y);
      return 1;
    }
    if (du - Math.abs(r0 - r1) >= EPS && (r0 + r1) - du >= EPS) {
      // two points
      double s = 0.5 * ((r0 * r0 - r1 * r1) / duSqr + 1);
      double t = Math.sqrt(r0 * r0 / duSqr - s * s);
      a[0] = new PtD(p0.x + s * U.x + t * V.x, p0.y + s * U.y + t * V.y);
      a[1] = new PtD(p0.x + s * U.x - t * V.x, p0.y + s * U.y - t * V.y);
      return 2;
    }
    // no intersection
    return 0;
  }



    /**
     * @param ps
     *            array containing the set of distinct points
     * @param n
     *            number of points
     * @return array of points on the convex hull (may be empty, if n<=0)
     */
    public static PtD[] grahamScan(PtD[] ps, int n, boolean keepColinear) {
        // maybe check for these outside?
        if (n <= 0) {
            PtD[] ret = new PtD[0];
            return ret; // or null?
        }
        if (n == 1) {
            PtD[] ret = new PtD[1];
            ret[0] = ps[0];
            return ret;
```

```
        }
        // find pivot and sort
        int p = 0;
        for (int i = 1; i < n; i++) {
            if (ps[i].compareTo(ps[p]) < 0)
                p = i;
        }
        PtD tmp = ps[0];
        ps[0] = ps[p];
        ps[p] = tmp;
        angularSort(ps, 1, n);
        // check if they are all on the same line
        if (Math.abs((ps[n - 1].sub(ps[0])).cross(ps[1].sub(ps[0]))) < EPS) {
            if (keepColinear) {
                PtD[] ret = new PtD[n];
                ret[0] = ps[0];
                if (ps[0].distSquared(ps[1]) >= ps[0].distSquared(ps[n - 1])
                        + EPS)
                    for (int i = 1; i < n; i++)
                        ret[n - i] = ps[i];
                else
                    for (int i = 1; i < n; i++)
                        ret[i] = ps[i];
                return ret;
            } else {
                PtD[] ret = new PtD[2];
                ret[0] = ps[0];
                if (ps[0].distSquared(ps[1]) >= ps[0].distSquared(ps[n - 1])
                        + EPS)
                    ret[1] = ps[1];
                else
                    ret[1] = ps[n - 1];
                return ret;
            }
        }
        // remove closer ones on the same line
        PtD[] tps = new PtD[n];
        tps[0] = ps[0];
        tps[1] = ps[1];
        int tt = 0;
        int start = 2;
        int end = n;
        if (keepColinear) {
            PtD a = ps[0].sub(ps[1]);
            while (Math.abs(a.cross(ps[0].sub(ps[start]))) < EPS) {
                tps[start] = ps[start];
                start++;
            }
            a = ps[0].sub(ps[n - 1]);
            while (Math.abs(a.cross(ps[0].sub(ps[end - 1]))) < EPS) {
                end--;
            }
```

```
            end++;
        }
        for (int i = start; i < end; i++) {
            PtD a = tps[i - tt - 1].sub(tps[i - tt - 2]);
            PtD b = ps[i].sub(tps[i - tt - 2]);
            if (!keepColinear && Math.abs(a.cross(b)) < EPS) {
                tps[i - tt - 1] = ps[i];
                tt++;
            } else {
                tps[i - tt] = ps[i];
            }
        }
        for (int i = end; i < n; i++) {
            tps[i - tt] = ps[i];
        }
        // remove last point if colinear
        if (!keepColinear && n - tt > 2) {
            PtD a = tps[0].sub(tps[n - tt - 2]);
            PtD b = tps[0].sub(tps[n - tt - 1]);
            if (Math.abs(a.cross(b)) < EPS)
                tt++;
        }
        n -= tt;
        PtD[] stack = new PtD[n];
        int stackSize = 0;
        stack[stackSize++] = tps[0];
        stack[stackSize++] = tps[1];
        for (int i = 2; i < n; i++) {
            while (true) {
                PtD a = stack[stackSize - 1].sub(stack[stackSize - 2]);
                PtD b = tps[i].sub(stack[stackSize - 2]);
                double cross = a.cross(b);
                if (cross <= -EPS || (cross < EPS && keepColinear))
                    break;
                stackSize--;
            }
            stack[stackSize++] = tps[i];
        }
        PtD[] ret = new PtD[stackSize];
        System.arraycopy(stack, 0, ret, 0, stackSize);
        return ret;
    }



    private static void angularSort(PtD ps[], int begin, int end) {
        int mid;
        if (end - begin <= 1) {
            return;
        }
        mid = (begin + end) / 2;
        angularSort(ps, begin, mid);
```

```
        angularSort(ps, mid, end);
        merge(ps, begin, mid, end);
    }

    private static void merge(PtD[] ps, int start, int mid, int end) {
        int i = start;
        int j = mid;
        int k = 0;
        PtD[] temp = new PtD[end - start];
        while ((i < mid) && (j < end))
            if (ps[i].compareTo(ps[j], ps[0]) <= 0) {
                temp[k++] = ps[i++];
            } else {
                temp[k++] = ps[j++];
            }
        while (i < mid) {
            temp[k++] = ps[i++];
        }
        while (j < end) {
            temp[k++] = ps[j++];
        }
        for (i = start; i < end; i++)
            ps[i] = temp[i - start];
    }

}
```

```
/*
 * Seg - segment/ray/line class (distances/intersections)
 * @author Darko Aleksic
 */
class Seg { // needs PtD (not all of it, add as needed)
    double a, b, c; // line ax + by = c
    PtD P0, P1;
    PtD N; // normal, line is nX=c, X=(x,y)
    PtD D; // dir vector, line is P0+tD

    // if it's a ray, pass endpoint as P0
    public Seg(PtD P0, PtD P1) {
        this.P0 = P0;
        this.P1 = P1;
        a = P1.y - P0.y;
        b = P0.x - P1.x;
        c = a * P0.x + b * P0.y;
        // careful with zero-length segments!
        // normalize it?
        double d = P0.dist(P1);
        if (d > PtD.EPS) {
            a /= d;
            b /= d;
```

```
        c /= d;
    }
    N = new PtD(a, b);
    D = new PtD(b, -a);
}




// generic point-to-segment, can be adjusted to p-to-line or p-to-ray
public static double squaredDistance(PtD Y, Seg S) {
    PtD DD = S.P1.sub(S.P0);
    PtD YmP0 = Y.sub(S.P0);
    double t = DD.dot(YmP0);
    if (t <= PtD.EPS) // remove if line!
        return YmP0.dot(YmP0);
    double ddd = DD.dot(DD);
    if (t >= ddd - PtD.EPS) { // remove if line OR ray!
        PtD YmP1 = Y.sub(S.P1);
        return YmP1.dot(YmP1);
    }
    return YmP0.dot(YmP0) - t * t / ddd; // maybe abs() if 0.0?
}




public static double lineToLineDistance(Seg line1, Seg line2) {
    double cross = line1.N.dot(line2.D);
    if (Math.abs(cross) >= PtD.EPS)
        return 0;// they intersect
    double dot = line1.N.dot(line2.N);
    if (dot < 0) // fishy? but if close to 0, does not matter?
        return Math.abs(line2.c + line1.c);
    else
        return Math.abs(line2.c - line1.c);
}




public static double lineToSegmentDistance(Seg line, Seg segment) {
    double q0 = line.N.dot(segment.P0) - line.c;
    double q1 = line.N.dot(segment.P1) - line.c;
    if (q0 * q1 <= -PtD.EPS)
        return 0;
    return Math.min(Math.abs(q0), Math.abs(q1));
}




public static double segmentToSegmentDistance(Seg seg1, Seg seg2) {
    if (overlap(seg1, seg2) != null)
```

```
        return 0;
    if (isect(seg1, seg2) != null)
        return 0;
    double d = squaredDistance(seg1.P0, seg2);
    d = Math.min(d, squaredDistance(seg1.P1, seg2));
    d = Math.min(d, squaredDistance(seg2.P0, seg1));
    return Math.sqrt(Math.min(d, squaredDistance(seg2.P1, seg1)));
}

public boolean contains(PtD p) {
    return Math.abs(a * p.x + b * p.y - c) < PtD.EPS
            && Math.min(P0.x, P1.x) - PtD.EPS <= p.x
            && p.x <= Math.max(P0.x, P1.x) + PtD.EPS
            && Math.min(P0.y, P1.y) - PtD.EPS <= p.y
            && p.y <= Math.max(P0.y, P1.y) + PtD.EPS;
}



public static PtD isect(Seg s, Seg t) {
    double d = s.a * t.b - s.b * t.a;
    if (Math.abs(d) < PtD.EPS)
        return null; // parallel lines, deal with them somewhere else
    PtD p = new PtD((s.c * t.b - s.b * t.c) / d, (s.a * t.c - s.c * t.a)
            / d);
    if (!s.contains(p) || !t.contains(p))
        return null;
    return p;
}



// if segments overlap, return their union,
// otherwise return null
public static Seg overlap(Seg s, Seg t) {
    if (Math.abs(s.a * t.b - s.b * t.a) >= PtD.EPS)
        return null;
    if (s.contains(t.P0) && s.contains(t.P1))
        return s;
    if (t.contains(s.P0) && t.contains(s.P1))
        return t;
    if (t.contains(s.P1))
        s.swapEnds();
    if (!t.contains(s.P0))
        return null;
    if (s.contains(t.P1))
        t.swapEnds();
    if (!s.contains(t.P0))
        return null;
    return new Seg(s.P1, t.P1);
}
```

```
    private void swapEnds() {
        PtD t = P0;
        P0 = P1;
        P1 = t;
    }




    /**
     * Line - Circle intersection (add contains() check if segment)
     *
     * @param line
     * @param C
     * @param r
     * @param ips
     * @return number of intersection points (held in ips)
     */
    public static int lineCircleIntersection(Seg line, PtD C, double r,
            PtD[] ips) {
        PtD delta = line.P0.sub(C);
        double dd = line.D.dot(delta);
        double discr = dd * dd + r * r - delta.dot(delta);
        if (discr <= -PtD.EPS)
            return 0; // no intersection
        if (Math.abs(discr) < PtD.EPS) { // single point (line tangent)
            ips[0] = new PtD(line.P0.x - dd * line.D.x, line.P0.y - dd
                    * line.D.y);
            return 1;
        }
        discr = Math.sqrt(discr);
        double t = -dd + discr;
        ips[0] = new PtD(line.P0.x + t * line.D.x, line.P0.y + t * line.D.y);
        t = -dd - discr;
        ips[1] = new PtD(line.P0.x + t * line.D.x, line.P0.y + t * line.D.y);
        return 2;
    }
}




/*
 * HalfCircle - good for one thing: area of intersecting circles
 * @author Darko Aleksic
 */
class HalfCircle {
    double x, y, r; // center and radius of the circle
    int type; // 1 top half, -1 bottom half

    /* area between two halfcircle segments on [a,b] */
    public static double getArea(HalfCircle hc1, HalfCircle hc2, double a,
```

```
            double b) {
        double area = (hc1.y - hc2.y) * (b - a);
        area += (hc1.type) * hc1.integral(a, b);
        area -= (hc2.type) * hc2.integral(a, b);
        return area;
    }

    private double integral(double a, double b) {
        return f2(b) - f2(a);
    }

    private double f2(double xx) {
        xx -= x;
        double tt = xx / r;
        if (tt >= 1.0 - 1e-9)
            return 0.5 * xx * f(xx) + 0.25 * r * r * Math.PI;
        if (tt <= -1.0 + 1e-9)
            return 0.5 * xx * f(xx) - 0.25 * r * r * Math.PI;
        return 0.5 * xx * f(xx) + 0.5 * r * r * Math.asin(tt);
    }

    private double f(double xx) {
        double tt = r * r - xx * xx;
        if (tt <= 1e-15)
            return 0;
        return Math.sqrt(tt);
    }
}
```

Graphs

```
/*
 * Maxflow
 * @author Darko Aleksic
 */
class MaxFlow {
  /**
   * Thanks goes to Igor Naverniouk.
   *
   * MAX FLOW - both FF (nm^2) and Dinic (n^2m) (? check the complexity)
   */
  private final static int NN = 256; // max number of nodes
  private int[][] cap = new int[NN][NN]; // both
  private int[][] fnet = new int[NN][NN]; // ff
  private int[][] adj = new int[NN][NN]; // dinic
  private int[] deg = new int[NN]; // dinic

  // BFS (both)
  private int[] q = new int[NN];
  private int[] prev = new int[NN];
  private int qf, qb;

  private int fordFulkerson(int n, int s, int t) {
    for (int i = 0; i < n; i++) {
      for (int j = 0; j < n; j++) {
        fnet[i][j] = 0;
      }
    }
    int flow = 0;
    while (true) {
      // find an augmenting path
      for (int i = 0; i < n; i++) {
        prev[i] = -1;
      }
      qf = qb = 0;
      prev[s] = -2;
      q[qb++] = s;

      while (qb > qf && prev[t] == -1) {
        int u = q[qf++];
        int v = 0;
        while (v < n) {
          if (prev[v] == -1 && fnet[u][v] - fnet[v][u] < cap[u][v]) {
            prev[v] = u;
            q[qb++] = v;
          }
          v++;
        }
      }
      // see if we are done
      if (prev[t] == -1)
```

```
        break;
      // get the bottleneck capacity
      int bot = Integer.MAX_VALUE;
      int v = t;
      int u = prev[v];
      while (u >= 0) {
        bot = Math.min(bot, cap[u][v] - fnet[u][v] + fnet[v][u]);
        v = u;
        u = prev[v];
      }
      // update the flow network
      v = t;
      u = prev[v];
      while (u >= 0) {
        fnet[u][v] += bot;
        v = u;
        u = prev[v];
      }
      flow += bot;
    }
    return flow;
  }

  private int dinic(int n, int s, int t) {
    int flow = 0;
    while (true) {
      // find an augmenting path
      for (int i = 0; i < n; i++) {
        prev[i] = -1;
      }
      qf = qb = 0;
      prev[s] = -2;
      q[qb++] = s;
      while (qb > qf && prev[t] == -1) {
        int u = q[qf++];
        for (int i = 0; i < deg[u]; i++) {
          int v = adj[u][i];
          if (prev[v] == -1 && cap[u][v] != 0) {
            prev[v] = u;
            q[qb++] = v;
          }
        }
      }
      // see if we're done
      if (prev[t] == -1)
        break;
      // try finding more paths
      for (int z = 0; z < n; z++)
        if (cap[z][t] > 0 && prev[z] != -1) {
          int bot = cap[z][t];
          int v = z;
          int u = prev[z];
```

```
        while (u >= 0) {
          bot = Math.min(bot, cap[u][v]);
          v = u;
          u = prev[v];
        }
        if (bot == 0)
          continue;

        cap[z][t] -= bot;
        cap[t][z] += bot;

        v = z;
        u = prev[z];
        while (u >= 0) {
          cap[u][v] -= bot;
          cap[v][u] += bot;
          v = u;
          u = prev[v];
        }
        flow += bot;
      }
  }
  return flow;
}

private void addEdge(int u, int v, int cp) {
  cap[u][v] += cp;
}

private void addEdgeUndirected(int u, int v, int cp) {
  addEdge(u, v, cp);
  addEdge(v, u, cp);
}

/* Max Flow example usage (UVa 820 sample network) */
private void maxFlowExample() {
  int n = 4;
  /* CLEAR - add source/sink if needed */
  for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++) {
      cap[i][j] = 0;
    }
    deg[i] = 0; // for dinic only
  }
  addEdgeUndirected(0, 1, 20);
  addEdgeUndirected(0, 2, 10);
  addEdgeUndirected(1, 2, 5);
  addEdgeUndirected(1, 3, 10);
  addEdgeUndirected(2, 3, 20);
  /* start dinic specific */
  for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++) {
```

```
        if (cap[i][j] > 0) {
          adj[i][deg[i]++] = j;
        }
      }
    }
    /* end dinic specific */
    int s = 0, t = 3;
    // int flow = fordFulkerson(n, s, t);
    int flow = dinic(n, s, t);
    System.out.println("Flow: " + flow);
  }

  public static void main(String args[]) {
    MaxFlow mf = new MaxFlow();
    mf.maxFlowExample();
  }
}
```

```
/*
 * Min-cost max flow
 * @author Darko Aleksic
 */
class MinCostMaxFlow {

    /*
     * Thanks goes to Frank Chu and Igor Naverniouk.
     *
     * NOTE: anything with longs here should be changed to ints or doubles if
     * needed
     */
    // max number of vertices (make sure there is enough with sink/source)
    private final static int NN = 128; // needed by all
    // infinity is kinda fishy, change it as needed
    // (careful - you need to add it!)
    private final static long oo = Long.MAX_VALUE / 4; // needed by all
    // capacity of edges, 0 if none
    private long[][] cap = new long[NN][NN]; // needed by flows
    // network flow (hm, figure this one out)
    private long[][] fnet = new long[NN][NN]; // needed by flows
    // cost of traversing edges
    private long[][] cost = new long[NN][NN]; // needed by mfmc()
    // potentials on nodes?
    private long[] pi = new long[NN]; // needed by mfmc()
    // cost of network flow
    public long fcost; // needed by mfmc()
    // graph ?
    private long[][] graph = new long[NN][NN]; // used by dijkstra
    // graph itself (it is a list! not a matrix!)
    private int[][] adj = new int[NN][NN]; // needed by all
    // with adj[][] is our graph
```

```
private int[] deg = new int[NN]; // needed by all
// parent array
private int[] par = new int[NN]; // needed by all
// distances
private long[] d = new long[NN]; // needed by all
// queue
private int[] q = new int[NN]; // only if we are using dijkstraPQ()
// is it in queue? -1 no, 0 yes?
private int[] inq = new int[NN]; // only if we are using dijkstraPQ()
// queue size
private int qs;// only if we are using dijkstraPQ()

// only if we are using dijkstraPQ()
private void bubl(int i, int j) {
    int t = q[i];
    q[i] = q[j];
    q[j] = t;
    t = inq[q[i]];
    inq[q[i]] = inq[q[j]];
    inq[q[j]] = t;
}

// calculate vertex potential - mfmc() only
private long pot(int u, int v) {
    return d[u] + pi[u] - pi[v];
}



/**
 * dijkstra using PQ (change longs to ints if needed) UNTESTED!!!
 *
 * @param n
 *            number of vertices
 * @param s
 *            source
 * @param t
 *            target
 * @return path length (-1 if none)
 */
public long dijkstra(int n, int s, int t) {
    for (int i = 0; i < n; i++) {
        d[i] = oo;
        inq[i] = -1;
        par[i] = -1;
    }
    d[s] = qs = 0;
    inq[q[qs++] = s] = 0;
    par[s] = -2;
    while (qs > 0) {
        // get the minimum from the q
        int u = q[0];
```

```
            inq[u] = -1;
            // bubble down
            q[0] = q[--qs];
            if (qs > 0)
                inq[q[0]] = 0;
            for (int i = 0, j = 2 * i + 1; j < qs; i = j, j = 2 * i + 1) {
                if (j + 1 < qs && d[q[j + 1]] < d[q[j]])
                    j++;
                if (d[q[j]] >= d[q[i]])
                    break;
                bubl(i, j);
            }
            // relax neighbours
            for (int k = 0, v = adj[u][k]; k < deg[u]; v = adj[u][++k]) {
                long newd = d[u] + graph[u][v];
                if (newd < d[v]) {
                    d[v] = newd;
                    par[v] = u;
                    if (inq[v] < 0) {
                        inq[q[qs] = v] = qs;
                        qs++;
                    }
                    // bubble up
                    int i = inq[v];
                    int j = (i - 1) / 2;
                    while (j >= 0 && d[q[i]] < d[q[j]]) {
                        bubl(i, j);
                        i = j;
                        j = (i - 1) / 2;
                    }
                }
            }
        }
        return (d[t] < oo) ? d[t] : -1;
    }



  /**
   * Dijkstra's shortest path with PQ - use for sparse graphs (use the one
   * below for dense ones)
   *
   * @param n
   *      number of vertices
   * @param s
   *      source
   * @param t
   *      sink
   *
   * @return true if s-t path exists, can be retrieved using par[]
   */
  public boolean dijkstraMCMFPQ(int n, int s, int t) {
```

```
for (int i = 0; i < n; i++) {
  d[i] = oo;
  par[i] = -1;
  inq[i] = -1;
}
d[s] = 0;
qs = 0;
inq[s] = 0;
q[qs++] = s;
par[s] = n;
while (qs > 0) {
  // get the minimum from q and bubble down
  int u = q[0];
  if (d[u] == oo)
    break;
  inq[u] = -1;
  q[0] = q[--qs];
  if (qs > 0)
    inq[q[0]] = 0;
  int i = 0;
  int j = 1;
  while (j < qs) {
    if (j + 1 < qs && d[q[j + 1]] < d[q[j]])
      j++;
    if (d[q[j]] >= d[q[i]])
      break;
    bubl(i, j);
    i = j;
    j = 2 * i + 1;
  }
  // relax edge (u,i) or (i,u) for all i
  for (int k = 0; k < deg[u]; k++) {
    int v = adj[u][k];
    // try undoing edge v->u
    if (fnet[v][u] != 0 && d[v] > pot(u, v) - cost[v][u]) {
      d[v] = pot(u, v) - cost[v][u];
      par[v] = u;
    }
    // try using edge u->v
    if (fnet[u][v] < cap[u][v] && d[v] > pot(u, v) + cost[u][v]) {
      d[v] = pot(u, v) + cost[u][v];
      par[v] = u;
    }
    if (par[v] == u) {
      // bubble up or decrease key
      if (inq[v] < 0) {
        inq[v] = qs;
        q[qs++] = v;
      }
      i = inq[v];
      j = (i - 1) / 2;
      while (j >= 0 && d[q[i]] < d[q[j]]) {
```

```
        bubl(i, j);
        i = j;
        j = (i - 1) / 2;
      }
    }
  }
}
for (int i = 0; i < n; i++) {
  if (pi[i] < oo) {
    if (d[i] == oo)
      pi[i] = oo;
    else
      pi[i] += d[i];
  }
}
return par[t] >= 0;
}



/**
 * Dijkstra's shortest path - use for dense graphs (use the one above for
 * sparse ones)
 *
 * @param n
 *      number of vertices
 * @param s
 *      source
 * @param t
 *      sink
 *
 * @return true if s-t path exists, can be retrieved using par[]
 */
public boolean dijkstraMCMF(int n, int s, int t) {
  for (int i = 0; i < n; i++) {
    d[i] = oo;
    par[i] = -1;
  }
  d[s] = 0;
  par[s] = -n - 1;
  while (true) {
    // get the minimum from q and bubble down
    int u = -1;
    long bestD = oo;
    for (int i = 0; i < n; i++) {
      if (par[i] < 0 && d[i] < bestD) {
        bestD = d[i];
        u = i;
      }
    }
    if (bestD == oo)
      break;
```

```
    // relax edge (u,i) or (i,u) for all i
    par[u] = -par[u] - 1;
    for (int i = 0; i < deg[u]; i++) {
      // try undoing edge v->u
      int v = adj[u][i];
      if (par[v] >= 0)
        continue;
      if (fnet[v][u] != 0 && d[v] > pot(u, v) - cost[v][u]) {
        d[v] = pot(u, v) - cost[v][u];
        par[v] = -u - 1;
      }
      // try edge u->v
      if (fnet[u][v] < cap[u][v] && d[v] > pot(u, v) + cost[u][v]) {
        d[v] = pot(u, v) + cost[u][v];
        par[v] = -u - 1;
      }
    }
  }
  for (int i = 0; i < n; i++) {
    if (pi[i] < oo) {
      if (d[i] == oo)
        pi[i] = oo;
      else
        pi[i] += d[i];
    }
  }

  return par[t] >= 0;
}



  /**
   * Min cost max flow
   *
   * @param n
   *            number of vertices
   * @param s
   *            source vertex
   * @param t
   *            sink vertex
   * @return s-t flow (and cost is in fcost)
   */
  public int mcmf(int n, int s, int t) {
      // build the adjacency list
      for (int i = 0; i < n; i++) {
          deg[i] = 0;
          pi[i] = 0;
          for (int j = 0; j < n; j++) {
              fnet[i][j] = 0;
              if (cap[i][j] != 0 || cap[j][i] != 0)
                  adj[i][deg[i]++] = j;
```

```
            }
        }
        int flow = 0;
        fcost = 0;
        // repeatedly find the cheapest path from s to t
        /** * CHANGE THE DIJKSTRA'S IF NEEDED ** */
        while (dijkstraMCMF(n, s, t)) {
            // get the bottleneck capacity
            long bot = oo;
            int v = t;
            int u = par[v];
            while (v != s) {
                bot = Math.min(bot, (fnet[v][u] != 0) ? fnet[v][u]
                        : (cap[u][v] - fnet[u][v]));
                v = u;
                u = par[u];
            }
            // update the flow network
            v = t;
            u = par[v];
            while (v != s) {
                if (fnet[v][u] != 0) {
                    fnet[v][u] -= bot;
                    fcost -= bot * cost[v][u];
                } else {
                    fnet[u][v] += bot;
                    fcost += bot * cost[u][v];
                }
                v = u;
                u = par[u];
            }
            flow += bot;
        }
        return flow;
    }

    private void addEdge(int u, int v, int co, int cp) {
        cap[u][v] = cp;
        cost[u][v] = co;
    }

    private void addEdgeUndirected(int u, int v, int co, int cp) {
        addEdge(u, v, co, cp);
        addEdge(v, u, co, cp);
    }

    /* Mincut Maxflow example usage (UVa 10594 (undirected, unique edges)) */
    private void mcmfExample() {
        int n = 4; // n number of nodes, not counting sink and source if
        // external ones needed (use n+2 nodes)
        // CLEAR FIRST ! (set all cap[][] to 0, cost[][] to oo)
        for (int i = 0; i < n + 2; i++)
```

```
            for (int j = 0; j < n + 2; j++) {
                cap[i][j] = 0;
                cost[i][j] = oo;
            }
        // NOTE: Beware of parallel edges!!! (add nodes if needed)
        addEdgeUndirected(1, 4, 1, 10);
        addEdgeUndirected(1, 3, 3, 10);
        addEdgeUndirected(3, 4, 4, 10);
        addEdgeUndirected(1, 2, 2, 10);
        addEdgeUndirected(2, 4, 5, 10);
        // 0 - source, n+1 - sink, change accordingly
        addEdge(0, 1, 0, 20);
        addEdge(n, n + 1, 0, 20);
        // this one looks for mcmf from 1 to n
        int flow = mcmf(n + 2, 0, n + 1);
        System.out.println("Flow: " + flow + " Cost: " + fcost);
    }

    public static void main(String[] args) {
        MinCostMaxFlow gu = new MinCostMaxFlow();
        gu.mcmfExample();
    }
}
```

```
/*
 * Bipartite matching
 * @author Darko Aleksic
 */
class BipartiteMatching {
  /**
   * Thanks goes to Igor Naverniouk.
   *
   * Bipartite matching - O(mn)? - takes almost no time for m=n=10,000
   * bottleneck is building the graph (think about adjacency list)
   */
  boolean[][] graph; // [m][n]
  boolean[] seen; // n
  int[] matchL; // m
  int[] matchR; // n
  int n, m; // CAREFUL! DON'T REDECLARE THEM!

  private boolean bpm(int u) {
    for (int v = 0; v < n; v++) {
      if (graph[u][v]) {
        if (seen[v])
          continue;
        seen[v] = true;
        if (matchR[v] < 0 || bpm(matchR[v])) {
          matchL[u] = v;
```

```
          matchR[v] = u;
          return true;
        }
      }
    }
    return false;
  }

  /* Bipartite Matching example (UVa 11138 simple sample (heh) graph) */
  void bpmExample() {
    m = 3;
    n = 4;
    graph = new boolean[m][n];
    int[][] input = { { 0, 0, 1, 0 }, { 1, 1, 0, 1 }, { 0, 0, 1, 0 } };
    for (int i = 0; i < m; i++) {
      for (int j = 0; j < n; j++) {
        graph[i][j] = (1 == input[i][j]);
      }
    }
    matchL = new int[m];
    for (int i = 0; i < m; i++) {
      matchL[i] = -1;
    }
    matchR = new int[n];
    for (int i = 0; i < n; i++) {
      matchR[i] = -1;
    }
    int count = 0;
    for (int i = 0; i < m; i++) {
      seen = new boolean[n];
      if (bpm(i))
        count++;
    }
    System.out.println("We can match " + count + " pair(s) in that graph.");
  }

  public static void main(String args[]) {
    BipartiteMatching bpm = new BipartiteMatching();
    bpm.bpmExample();
  }
}
```

```
/*
 * Minimum spanning tree
 * @author Darko Aleksic
 */
 class MinimumSpanningTree {

  /**
   * Thanks goes to Gilbert Lee.
```

```
 *
 * Minimum Spanning Tree - Kruskal's O(mlogm) (sorting edges)
 *
 * NOTE: Needs class Edge below
 */

Edge[] edges, tree;
int[] sets;
int n, m;

private int MST() {
  int w = 0;
  int cnt = 0;
  for (int i = 0; i < m; i++) {
    int s1 = find(edges[i].u);
    int s2 = find(edges[i].v);
    if (s1 != s2) {
      union(s1, s2);
      w += edges[i].w;
      tree[cnt] = edges[i];
      cnt++;
    }
    if (cnt == n - 1)
      break;
  }
  if (cnt < n - 1)
    return 0; // or something meaningful (no tree)
  return w;
}

private void union(int s1, int s2) {
  // not sure if this max/min thingy is needed, I needed it somewhere
  sets[Math.min(s1, s2)] = Math.max(s1, s2);
}

private int find(int index) {
  if (sets[index] == index)
    return index;
  return sets[index] = find(sets[index]);
}

/* Minimum Spanning Tree example - UVa LA 2515 */
void mstExample() {
  n = 3; // number of nodes
  m = 7; // number of edges
  int[][] input = { { 1, 2, 19 }, { 2, 3, 11 }, { 3, 1, 7 }, { 1, 3, 5 },
      { 2, 3, 89 }, { 3, 1, 91 }, { 1, 2, 32 } };
  sets = new int[n];
  for (int i = 0; i < n; i++) {
    sets[i] = i;
  }
  edges = new Edge[m];
```

```
    for (int i = 0; i < m; i++) {
      int u = input[i][0] - 1; // 0-based!
      int v = input[i][1] - 1; // 0-based!
      int w = input[i][2];
      edges[i] = new Edge(u, v, w);
    }
    Arrays.sort(edges, 0, m);
    tree = new Edge[n - 1];
    System.out.println("MST length: " + MST());
    for (int i = 0; i < n - 1; i++)
      System.out.println((tree[i].u + 1) + "-" + (tree[i].v + 1) + " "
          + tree[i].w);
  }

  public static void main(String args[]) {
    MinimumSpanningTree mst = new MinimumSpanningTree();
    mst.mstExample();
  }
}

class Edge implements Comparable<Edge> {
  public int u, v, w;

  public Edge(int u, int v, int w) {
    this.u = u;
    this.v = v;
    this.w = w;
  }

  public int compareTo(Edge e2) {
    return w - e2.w;
  }
}



/*
 * Minimum cut
 * @author Darko Aleksic
 */
class MincutWeighted {
    /**
     * Thanks goes to Igor Naverniouk.
     *
     * Stoer-Wagner's O(n^3) mincut (graph undirected, weighted)
     */
    private static final int NN = 256; // max num of nodes
    // Maximum edge weight (MAXW * NN * NN must fit into an int)
    private static final int MAXW = 1024;
    int[][] g = new int[NN][NN];
    int[] v = new int[NN];
    int[] w = new int[NN];
```

```
int[] na = new int[NN];
boolean[] a = new boolean[NN];

private int minCut(int n) {
    for (int i = 0; i < n; i++)
        v[i] = i;
    int best = MAXW * n * n;
    while (n > 1) {
        a[v[0]] = true;
        for (int i = 1; i < n; i++) {
            a[v[i]] = false;
            na[i - 1] = i;
            w[i] = g[v[0]][v[i]];
        }
        int prev = v[0];
        for (int i = 1; i < n; i++) {
            int zj = -1;
            for (int j = 1; j < n; j++)
                if (!a[v[j]] && (zj < 0 || w[j] > w[zj]))
                    zj = j;
            a[v[zj]] = true;
            if (i == n - 1) {
                best = Math.min(best, w[zj]);
                for (int j = 0; j < n; j++)
                    g[v[j]][prev] = g[prev][v[j]] += g[v[zj]][v[j]];
                v[zj] = v[--n];
                break;
            }
            prev = v[zj];
            for (int j = 1; j < n; j++)
                if (!a[v[j]])
                    w[j] += g[v[zj]][v[j]];
        }
    }
    return best;
}

/* Weighted Mincut example - sample graph from UVa 10989 */
private void mcwExample() {
    int n = 4;
    g[0][1] = g[1][0] = 10;
    g[1][2] = g[2][1] = 100;
    g[2][3] = g[3][2] = 10;
    g[3][0] = g[0][3] = 100;
    g[0][2] = g[2][0] = 10;
    System.out.println("Min cut: " + minCut(n));
}

public static void main(String args[]) {
    MincutWeighted mcw = new MincutWeighted();
    mcw.mcwExample();
}
```

```
}
```

```
procedure FloydWarshallWithPathReconstruction ()
   for k := 1 to n
      for i := 1 to n
         for j := 1 to n
            if path[i][k] + path[k][j] < path[i][j] then {
               path[i][j] := path[i][k]+path[k][j];
               next[i][j] := k; }

function Path (i,j)
   if path[i][j] equals infinity then
     return "no path";
   int intermediate := next[i][j];
   if intermediate equals 'null' then
     return " ";   /* there is an edge from i to j, with no vertices between */
   else
     return Path(i,intermediate) + intermediate + GetPath(intermediate,j);
```

Other

```
/*
 * Next permutation
 */
    private void swap(int[] a, int i, int j) {
        int t = a[i];
        a[i] = a[j];
        a[j] = t;
    }

    private boolean nextPerm(int[] a) {
        if (a.length <= 1)
            return false;
        int i = a.length - 1;
        while (a[i - 1] >= a[i]) {
            i--;
            if (i == 0)
                return false;
        }
        int j = a.length;
        while (a[j - 1] <= a[i - 1]) {
            j--;
            if (j == 0)
                return false;
        }
        swap(a, i - 1, j - 1);
        i++;
        j = a.length;
        while (i < j) {
            swap(a, i - 1, j - 1);
            i++;
            j--;
        }
        return true;
    }
```

```
/*
 * Sieve of Eratosthenes
 */

     private void sieve() {
        int lim = (int) (Math.round(Math.sqrt(SIEVE_SIZE))) + 1;
        nonPrimes[0] = true;
        nonPrimes[1] = true;
        for (int i = 4; i < SIEVE_SIZE; i += 2) {
            nonPrimes[i] = true;
        }
        for (int i = 3; i <= lim; i += 2) {
            if (!nonPrimes[i]) {
```

```
                int tmp = i * i;
                while (tmp < SIEVE_SIZE) {
                    nonPrimes[tmp] = true;
                    tmp += i << 1;
                }
            }
        }
    }




/*
 * Euclidean Algorithm (GCD)
  */
public int getGCD(int a, int b)
        {
                while (b!= 0)
                {
                        int m = a%b;
                        a = b;
                        b = m;
                }
                return a;
        }



/*
 * Newton's method (Zero finding with the derivative)
*/
public class Newton{
  interface ContinuousFunction{
    public double function(double x);
    public double derivative(double x);
  }
  final static int MAX_IT = 100000;
  final static double PRECISION = 1*Math.pow(10,-8);
  public static double newton(ContinuousFunction f, double guess,
                                  double precision, int maxIt){
    double curX = guess;
    double curVal = f.function(curX);
    int it = 0;
    //Xn+1 = Xn - f(xn)/f'(xn)
    while(Math.abs(curVal) > precision && it < maxIt){
      curX = curX - curVal/f.derivative(curX);
      curVal = f.function(curX);
      it++;
    }
    if(it >= maxIt)
      System.err.println("Newton's method: Too many iterations.");
    return curX;
  }
```

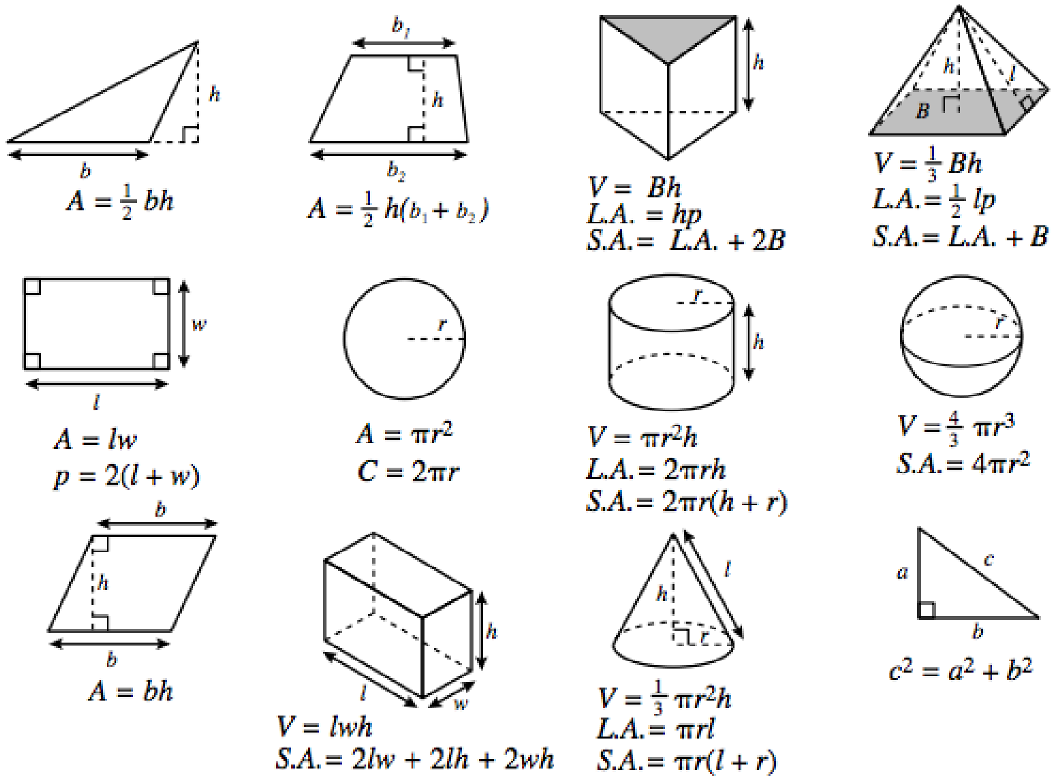
```
  public static double newton(ContinuousFunction f, double guess){
    return newton(f, guess, PRECISION, MAX_IT);
  }
  public static double newton(ContinuousFunction f, double guess, double precision){
    return newton(f, guess, precision, MAX_IT);
  }
  public static double newton(ContinuousFunction f, double guess, int maxIt){
    return newton(f, guess, PRECISION, maxIt);
  }
}


class KMPSkipSearch {
    /**
     * KMP Skip Search - 3x faster than regular KMP - from
     * http://www-igm.univ-mlv.fr/~lecroq/string/
     *
     * Find occurrences of P in T - implement readP(), readT() - complexity:
     * preprocessing O(plen), search O(tlen)
     */
    private char[] T, P;
    private int tlen, plen, matchNum;
    private int[] mpNext, kmpNext, list, z, matches;

    private void preMp() {
        int i, j;
        i = 0;
        j = mpNext[0] = -1;
        while (i < plen) {
            while (j > -1 && P[i] != P[j])
                j = mpNext[j];
            mpNext[++i] = ++j;
        }
    }

    private void preKmp() {
        int i, j;
        i = 0;
        j = kmpNext[0] = -1;
        while (i < plen) {
            while (j > -1 && P[i] != P[j])
                j = kmpNext[j];
            i++;
            j++;
            if (i == plen)
                break; // I guess not needed in C?
            if (P[i] == P[j])
                kmpNext[i] = kmpNext[j];
            else
                kmpNext[i] = j;
        }
    }
```

```
private int attempt(int start, int wall) {
    int k;
    k = wall - start;
    while (k < plen && P[k] == T[k + start])
        ++k;
    return (k);
}

private boolean KMPSKIP() {
    int i, j, k, kmpStart, start, wall;
    /* Preprocessing */
    preMp();
    preKmp();
    for (int ii = 0; ii < 256; ii++) {
        z[ii] = -1;
    }
    for (int ii = 0; ii < 1024; ii++) {
        list[ii] = -1;
    }
    z[P[0]] = 0;
    for (i = 1; i < plen; ++i) {
        list[i] = z[P[i]];
        z[P[i]] = i;
    }
    /* Searching */
    wall = 0;
    int per = plen - kmpNext[plen];
    i = j = -1;
    do {
        j += plen;
    } while (j < tlen && z[T[j]] < 0);
    if (j >= tlen)
        return false;
    i = z[T[j]];
    start = j - i;
    while (start <= tlen - plen) {
        if (start > wall)
            wall = start;
        k = attempt(start, wall);
        wall = start + k;
        if (k == plen) {
            // return true; // if only presence needed
            matches[matchNum++] = start;
            i -= per;
        } else
            i = list[i];
        if (i < 0) {
            do {
                j += plen;
            } while (j < tlen && z[T[j]] < 0);
            if (j >= tlen)
```

```
                return false;
            i = z[T[j]];
        }
        kmpStart = start + k - kmpNext[k];
        k = kmpNext[k];
        start = j - i;
        while (start < kmpStart || (kmpStart < start && start < wall)) {
            if (start < kmpStart) {
                i = list[i];
                if (i < 0) {
                    do {
                        j += plen;
                    } while (j < tlen && z[T[j]] < 0);
                    if (j >= tlen)
                        return false;
                    i = z[T[j]];
                }
                start = j - i;
            } else {
                kmpStart += (k - mpNext[k]);
                k = mpNext[k];
            }
        }
    }
    return false;
}

void kmpssExample() {
    T = new char[100010];
    P = new char[1024];
    mpNext = new int[1024];
    kmpNext = new int[1024];
    list = new int[1024];
    z = new int[256];
    matches = new int[100010];
    matchNum = 0;
    // readT(); // set tlen in there
    // readP(); // set plen in there
    T = "aabbababaaaaababababbbaaabababaababab".toCharArray();
    P = "aba".toCharArray();
    tlen = T.length;
    plen = P.length;
    KMPSKIP();
    System.out.println(new String(T));
    int i = 0;
    int j = 0;
    while (i < tlen) {
        if (matches[j] == i) {
            System.out.print('^');
            j++;
        } else {
            System.out.print(' ');
```

```
            }
            i++;
        }
    }

    public static void main(String args[]) {
        KMPSkipSearch kmpss = new KMPSkipSearch();
        kmpss.kmpssExample();
    }
}



// lcm requires the allSame method below.
    public static int lcm(int[] list){
      int[] a = Arrays.copyOf(list, list.length);
      while(!allSame(a)){
        int minIndex = minIndex(a);
        a[minIndex] += list[minIndex];

      }
      return a[0];
    }
// required for lcm
    public static boolean allSame(int[] list){
      for(int i : list){
        if(!i.equals(list[0])){
          return false;
        }
      }
      return true;

    }
```

Common Geometric Formulas



Theoretical Computer Science Cheat Sheet

Definitions

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

Series

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

In general:

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

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

Geometric series:

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

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

Harmonic series:

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Theoretical Computer Science Cheat Sheet

Identities Cont.

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

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

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. $\left[\begin{matrix} n \\ m \end{matrix}\right] = \sum_k \left[\begin{matrix} n+1 \\ k+1 \end{matrix}\right] \binom{k}{m} (-1)^{m-k},$

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. $\left[\begin{matrix} m+n+1 \\ m \end{matrix}\right] = \sum_{k=0}^{m} k(n+k) \left[\begin{matrix} n+k \\ k \end{matrix}\right],$

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

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

46. $\left\{\begin{matrix} n \\ n-m \end{matrix}\right\} = \sum_k \binom{m-n}{m+k} \binom{m+n}{n+k} \left[\begin{matrix} m+k \\ k \end{matrix}\right],$

47. $\left[\begin{matrix} n \\ n-m \end{matrix}\right] = \sum_k \binom{m-n}{m+k} \binom{m+n}{n+k} \left\{\begin{matrix} m+k \\ k \end{matrix}\right\},$

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. $\left[\begin{matrix} n \\ \ell+m \end{matrix}\right] \binom{\ell+m}{\ell} = \sum_k \left[\begin{matrix} k \\ \ell \end{matrix}\right] \left[\begin{matrix} n-k \\ m \end{matrix}\right] \binom{n}{k}.$

Trees

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

Kraft inequality: If the depths of the leaves of a binary tree are $d_1, \ldots, d_n$:

$$\sum_{i=1}^{n} 2^{-d_i} \leq 1,$$

and equality holds only if every internal node has 2 sons.

Recurrences

Master method:

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

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

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

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

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

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

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

Substitution (example): Consider the following recurrence

$$T_{i+1} = 2^{2^i} \cdot T_i^2, \quad T_1 = 2.$$

Note that T_i is always a power of two. Let $t_i = \log_2 T_i$. Then we have

$$t_{i+1} = 2^i + 2t_i, \quad t_1 = 1.$$

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

$$\frac{t_{i+1}}{2^{i+1}} = \frac{2^i}{2^{i+1}} + \frac{t_i}{2^i}.$$

Substituting we find

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

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

$$T(n) = 3T(n/2) + n, \quad T(1) = 1.$$

Rewrite so that all terms involving T are on the left side

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

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

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

Let $m = \log_2 n$. Summing the left side we get $T(n) - 3^m T(1) = T(n) - 3^m = T(n) - n^k$ where $k = \log_2 3 \approx 1.58496$. Summing the right side we get

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

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

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

and so $T(n) = 3n^k - 2n$. Full history recurrences can often be changed to limited history ones (example): Consider

$$T_i = 1 + \sum_{j=0}^{i-1} T_j, \quad T_0 = 1.$$

Note that

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

Subtracting we find

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

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

Generating functions:

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

Example:

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

Multiply and sum:

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

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

$$\frac{G(x) - g_0}{x} = 2G(x) + \sum_{i \geq 0} x^i.$$

Simplify:

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

Solve for $G(x)$:

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

Expand this using partial fractions:

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

So $g_i = 2^i - 1$.

Theoretical Computer Science Cheat Sheet

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

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

Pascal's Triangle

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

General

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

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

Change of base, quadratic formula:

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

Euler's number e:

$$e = 1 + \tfrac{1}{2} + \tfrac{1}{6} + \tfrac{1}{24} + \tfrac{1}{120} + \cdots$$

$$\lim_{n\to\infty} \left(1 + \frac{x}{n}\right)^n = e^x.$$

$$\left(1 + \tfrac{1}{n}\right)^n < e < \left(1 + \tfrac{1}{n}\right)^{n+1}.$$

$$\left(1 + \tfrac{1}{n}\right)^n = e - \frac{e}{2n} + \frac{11e}{24n^2} - O\left(\frac{1}{n^3}\right).$$

Harmonic numbers:

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

$$\ln n < H_n < \ln n + 1,$$

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

Factorial, Stirling's approximation:

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

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

Ackermann's function and inverse:

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

$$\alpha(i) = \min\{j \mid a(j,j) \geq i\}.$$

Binomial distribution:

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

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

Poisson distribution:

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

Normal (Gaussian) distribution:

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

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

$$nH_n.$$

Probability

Continuous distributions: If

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

then p is the probability density function of X. If

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

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

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

Expectation: If X is discrete

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

If X continuous then

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

Variance, standard deviation:

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

$$\sigma = \sqrt{\mathrm{VAR}[X]}.$$

For events A and B:

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

$$\Pr[A \wedge B] = \Pr[A] \cdot \Pr[B],$$

iff A and B are independent.

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

For random variables X and Y:

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

if X and Y are independent.

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

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

Bayes' theorem:

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

Inclusion-exclusion:

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

Moment inequalities:

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

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

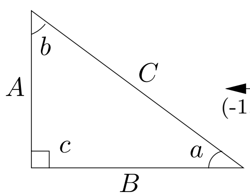
Geometric distribution:

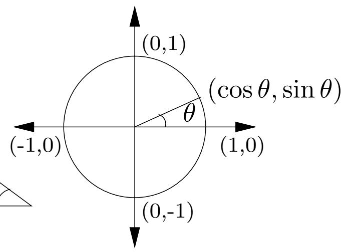
$$\Pr[X = k] = pq^{k-1}, \qquad q = 1 - p,$$

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

Theoretical Computer Science Cheat Sheet

Trigonometry





Pythagorean theorem:

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

Definitions:

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

Area, radius of inscribed circle:

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

Identities:

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

Euler's equation:

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

Matrices

Multiplication:

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

Determinants: $\det A \neq 0$ iff A is non-singular.

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

2×2 and 3×3 determinant:

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

Permanents:

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

Hyperbolic Functions

Definitions:

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

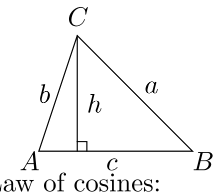
Identities:

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

θ	$\sin\theta$	$\cos\theta$	$\tan\theta$
0	0	1	0
$\frac{\pi}{6}$	$\frac{1}{2}$	$\frac{\sqrt{3}}{2}$	$\frac{\sqrt{3}}{3}$
$\frac{\pi}{4}$	$\frac{\sqrt{2}}{2}$	$\frac{\sqrt{2}}{2}$	1
$\frac{\pi}{3}$	$\frac{\sqrt{3}}{2}$	$\frac{1}{2}$	$\sqrt{3}$
$\frac{\pi}{2}$	1	0	∞

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

More Trig.



Law of cosines:

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

Area:

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

Heron's formula:

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

More identities:

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

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

Theoretical Computer Science Cheat Sheet

Number Theory

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

$$\begin{aligned} C &\equiv r_1 \bmod m_1 \\ \vdots\ &\ \vdots\ \ \vdots \\ C &\equiv r_n \bmod m_n \end{aligned}$$

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

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

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

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

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

Fermat's theorem:

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

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

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

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

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

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

Wilson's theorem: n is a prime iff

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

Möbius inversion:

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

If

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

then

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

Prime numbers:

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

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

Graph Theory

Definitions:

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

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

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

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

Any planar graph has a vertex with degree ≤ 5.

Notation:

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

Geometry

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

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

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

Distance formula, L_p and L_∞ metric:

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

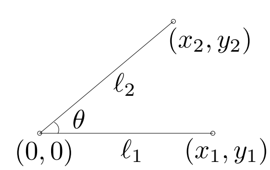
$$\left[|x_1 - x_0|^p + |y_1 - y_0|^p\right]^{1/p},$$

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

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

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

Angle formed by three points:



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

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

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

Area of circle, volume of sphere:

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

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

Theoretical Computer Science Cheat Sheet

π

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

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

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

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

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

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

Partial Fractions

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

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

Calculus

Derivatives:

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Integrals:

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

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

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

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

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

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

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

Calculus Cont.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Theoretical Computer Science Cheat Sheet

Calculus Cont.

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

63. $\displaystyle\int \frac{dx}{x^2\sqrt{x^2\pm a^2}} = \mp\frac{\sqrt{x^2\pm a^2}}{a^2x},$

64. $\displaystyle\int \frac{x\,dx}{\sqrt{x^2\pm a^2}} = \sqrt{x^2\pm a^2},$

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Finite Calculus

Difference, shift operators:

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

Fundamental Theorem:

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

Differences:

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

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

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

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

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

Sums:

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

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

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

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

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

Falling Factorial Powers:

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

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:

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

Theoretical Computer Science Cheat Sheet

Series

Taylor's series:

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

Expansions:

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Ordinary power series:

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

Exponential power series:

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

Dirichlet power series:

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

Binomial theorem:

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

Difference of like powers:

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

For ordinary power series:

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

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

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

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

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

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

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

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

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

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

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

Convolution:

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

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

Theoretical Computer Science Cheat Sheet

Series

Expansions:

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

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

$$\left(\ln\frac{1}{1-x}\right)^n = \sum_{i=0}^{\infty}\left[{i \atop n} \right]\frac{n!x^i}{i!}, \qquad x\cot x = \sum_{i=0}^{\infty}\frac{(-4)^iB_{2i}x^{2i}}{(2i)!},$$

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

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

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

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

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

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

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

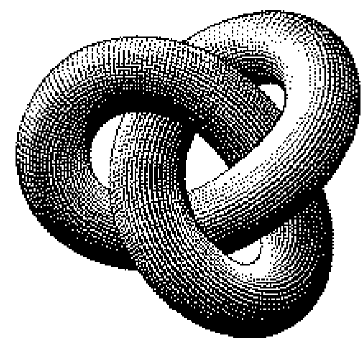
$$\left(\frac{1-\sqrt{1-4x}}{2x}\right)^n = \sum_{i=0}^{\infty}\frac{n(2i+n-1)!}{i!(n+i)!}x^i,$$

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

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

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

Escher's Knot



Stieltjes Integration

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

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

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

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

If the integrals involved exist

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

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

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

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

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

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

Cramer's Rule

If we have equations:

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

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

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

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

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

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

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

Fibonacci Numbers

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

Definitions:

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

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

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

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

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

Additive rule:

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

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

Calculation by matrices:

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

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