University of Calgary
Team Notebook 2008 ACM ICPC

Darko Aleksic, Sean McIntyre, Tom Flanagan

`Our Magic Incantations List`

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 behavior 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 initialization***, ***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!!!***

Table of Contents

```java
/* Articulation Points / Bridges from U of A codebook
 * NOTE: Needs class Node below
 */
class ArticulationPointsBridges {
   private static final int MAXN = 256;
   private Node[] alist;
   private boolean[] art, seen;
   private int[] dfNum, low, parent;
   private int[][] bridge;
   private int count, bridges;

   public ArticulationPointsBridges() {
      alist = new Node[MAXN];
      for (int i = 0; i < MAXN; i++) {
         alist[i] = new Node(MAXN);
      }
      art = new boolean[MAXN];
      seen = new boolean[MAXN];
      dfNum = new int[MAXN];
      low = new int[MAXN];
      parent = new int[MAXN];
      bridge = new int[MAXN * MAXN][2];
   }

   private void search(int v, boolean root) {
      seen[v] = true;
      int child = 0;
      low[v] = dfNum[v] = count++;
      for (int i = 0; i < alist[v].deg; i++) {
         int w = alist[v].adj[i];
         if (dfNum[w] == -1) {
            parent[w] = v;
            child++;
            search(w, false);
            if (low[w] > dfNum[v])
               addBridge(v, w);
            if (low[w] >= dfNum[v] && !root)
               art[v] = true;
            low[v] = Math.min(low[v], low[w]);
         } else if (w != parent[v]) {
            low[v] = Math.min(low[v], dfNum[w]);
         }
      }
      if (root && child > 1) art[v] = true;
   }

   // see if you need undirected
   private void addEdge(int u, int v) {
      alist[u].adj[alist[u].deg++] = v;
      alist[v].adj[alist[v].deg++] = u;
   }

   private void addBridge(int u, int v) {
      bridge[bridges][0] = u;
      bridge[bridges][1] = v;
      bridges++;
   }

   private void articulate(int n) {
      for (int i = 0; i < n; i++) {
         art[i] = false;
         dfNum[i] = -1;
         parent[i] = -1;
      }
      count = bridges = 0;
      for (int i = 0; i < n; i++) {
         if (!seen[i])
            search(i, true);
      }
   }

   /* Articulation Points / Bridges example (UVa 796) */
   private void apbExample() {
      // clear graph, build it and then run articulation()
      // articulation points are marked in art[], bridges are in bridge[]
      int n = 8;
      for (int i = 0; i < n; i++) {
         alist[i].deg = 0;
         seen[i] = false;
      }
      int[][] input = {{0,1},{1,2},{1,3},{2,3},{3,4},{6,7}};
      for (int i = 0; i < input.length; i++) {
         addEdge(input[i][0], input[i][1]);
      }
      articulate(n);
      System.out.print("Articulation points:");
      for (int i = 0; i < n; i++) {
         if (art[i]) System.out.print(" " + i);
      }
      System.out.print("\nBridges:");
      for (int i = 0; i < bridges; i++) {
         System.out.print("  (" + bridge[i][0] + "-" +
            bridge[i][1] + ")");
      }
      System.out.println();
   }
}

class Node {
   int deg, adj[];
   public Node(int maxn) {
      deg = 0; adj = new int[maxn];
   }
}
```

```java
/**
* Bipartite matching - O(mn)? - takes almost no time for m=n=10,000
 * bottleneck is building the graph (think about adjacency list)
 */
class BipartiteMatching {
   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;
```

```java
            }
        }
    }
    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];
    Arrays.fill(matchL,-1);
    matchR = new int[n];
    Arrays.fill(matchR,-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.");
}
}
```

```java
/**
 * 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)
 */
class KMPSkipSearch {
    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 = 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();
        Arrays.fill(z,-1);
        Arrays.fill(list,-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;
```

```java
                } 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 = "aabbababaaaaababababbbaaababababaababab".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++;
    }
}
}
```

```java
/**
 * Thanks goes to Igor Navernouk.
 * MAX FLOW - both FF (nm^2) and Dinic (n^2m) (? check the complexity)
 */
class MaxFlow {
    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;
    // System.out.println("Flow: " + fordFulkerson(n, s, t));
    System.out.println("Flow: " + dinic(n, s, t));
  }
}
```

```
// Crime Wave --- an example of max flow [preflow-push-label].
// borrowed from somewhere - no idea whose code this is
int N,M; // number of verts and edges
struct edge { int x,y, f,c, rev; };
edge eg[500000];
int aj[5010][200]; int pc[5010];
int phi[5010]; int ex[5010];
int mac[6000]; int ac[5010];
int dead, born;
void push(int a){
    int x=eg[a].x; int y=eg[a].y; int gg=ex[x];
    if(gg>eg[a].c) gg=eg[a].c;
    eg[a].f+=gg; eg[a].c-=gg;
    int k=eg[a].rev;
    eg[k].f-=gg; eg[k].c+=gg;
    ex[x]-=gg; ex[y]+=gg;
    if(ex[x]==0) {dead=(dead+1)%6000; ac[x]=0;}
    if(y && y<N-1 && ac[y]==0) {mac[born]=y; ac[y]=1; born=(born+1)%6000;}
}
int maxflow(){
    int i,j,k,t1,t2,t3;
    for(i=1;i<N;i++) { ex[i]=0; ac[i]=0; }
    ex[0]=1000000000;
    dead=born=0;
    for(i=0, j=pc[0];i<j;i++)
        push(aj[0][i]);
    phi[0]=N;
    for(i=1;i<N;i++) phi[i]=0;
    while(dead!=born)
    {
        i=mac[dead];
        t2=100000000;
        for(t1=pc[i], j=0; j<t1; j++)
        {
            k=aj[i][j];
            if(eg[k].c==0) continue;
            t3=phi[eg[k].y]+1;
            if(t3<t2) t2=t3;
            if(phi[i]==phi[eg[k].y]+1)
            {
                push(k);
                j=t1+10;
            }
        }
        if(j<t1+5) phi[i]=t2;
    }
    int ans=0;
    for(i=0, j=pc[0];i<j;i++)
    {
```

```
        k=aj[0][i];
        ans+=eg[k].f;
    }
    //cout<<ans<<endl;
    return(ans);
}
void init(int a){
    int i; N=a;
    for(i=0;i<N;i++) pc[i]=0;
    M=0;
}
void addEdge(int x, int y, int c){
    eg[M].x=x; eg[M].y=y; eg[M].c=c; eg[M].f=0;
    eg[M].rev=M+1; eg[M+1].rev=M;
    eg[M+1].x=y; eg[M+1].y=x; eg[M+1].c=0;
    eg[M+1].f=0;
    aj[x][pc[x]]=M; pc[x]++;
    aj[y][pc[y]]=M+1; pc[y]++;
    M+=2;
}
int n,m, B;
int oPt(int a, int b){ return(2*(a*m+b)+1); }
int iPt(int a, int b){ return(2*(a*m+b)+2); }
int main(){
    int i,j,k;
    int q; cin>>q;
    while(q)
    {
        q--;
        cin>>n>>m;
        init(2*m*n+2);
        for(i=0;i<n;i++)
            for(j=0;j<m;j++)
            {
                k=oPt(i,j);
                addEdge(iPt(i,j),k,1);
                if(i==0) addEdge(k,N-1,1);
                else addEdge(k,iPt(i-1,j),1);
                if(i==n-1) addEdge(k,N-1,1);
                else addEdge(k,iPt(i+1,j),1);
                if(j==0) addEdge(k,N-1,1);
                else addEdge(k,iPt(i,j-1),1);
                if(j==m-1) addEdge(k,N-1,1);
                else addEdge(k,iPt(i,j+1),1);
            }
            cin>>B;
            for(k=0;k<B;k++)
            {
                cin>>i>>j;
                i--;j--;
                if(B<=200) addEdge(0,iPt(i,j),1);
            }
            if(B>200) cout<<"not possible";
            else if(maxflow()==B) cout<<"possible";
            else cout<<"not possible";
            cout<<endl;
    }
    return(0);
}
```

```
class MinCostMaxFlow {
    /*
     * Thanks goes to Frank Chu and Igor Naverniouk.
     */
```

```
// 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 - need to add!)
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!!!(?)
 * @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++;
```

```java
            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;
}
```

```java
/**
 * Dijkstra's shortest path with PQ - use for sparse graphs (use the
 * one below for dense ones)
 * @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)
 * @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];
```

```java
                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
 */
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;
}
```

```java
    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);
    }
}
```

```java
/**
 * Thanks goes to Igor Naverniouk.
 * Stoer-Wagner's O(n^3) mincut (graph undirected, weighted)
 */
class MincutWeighted {
    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;
```

```java
            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));
}
}
```

```java
/**
 * Thanks goes to Gilbert Lee.
 * Minimum Spanning Tree - Kruskal's O(mlogm) (sorting edges)
 * NOTE: Needs class Edge below
 */
class MinimumSpanningTree {
    Edge[] edges, tree;
    int n, m, sets[];

    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);
    }
}

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;
    }
}
```

```java
class MinWBPM {
    /*
     * Minimum weighted bipartite matching. Hungarian algorithm O(n^3). Use
     * maxValue-weight and adjust result in solve() for maximum matching. Thanks
     * goes to rgrig.
     */
    private int M, N, MAXM, MAXN, urCount, r, c, rowsLeft;
    private int[] unmatchedRows, matchR, matchC, par, rDec, cInc, slack;
    private int[][] w;
    private final static int oo = Integer.MAX_VALUE / 16; // adjust if needed

    /* example minimum weighted bpm SWERC 2007 G */
    private void work() {
        MAXM = MAXN = 16; // or whatever
        // make sure M<=N !!!
        M = 4; N = 5;
        init(); clear(oo); // use more than max val
        // read the graph (if maximum, use max - weight)
        w[0][2] = 5; w[0][3] = 3; w[1][1] = 20; w[1][4] = 10; w[2][1] = 25;
        w[2][4] = 30; w[3][0] = 2; w[3][2] = 10; w[3][3] = 12;
        System.out.println(solve()); // if max change result
    }

    private void init() {
        unmatchedRows = new int[MAXM];
        matchR = new int[MAXM];
        matchC = new int[MAXN];
        par = new int[MAXN];
```

```java
        rDec = new int[MAXM];
        cInc = new int[MAXN];
        slack = new int[MAXN];
        w = new int[MAXM][MAXN];
    }

    private int solve() {
        while (rowsLeft > 0) {
            Arrays.fill(par, -1);
            Arrays.fill(slack, oo);
            urCount = 0;
            for (r = 0; r < M; ++r) {
                if (matchR[r] == -1) {
                    unmatchedRows[urCount++] = r;
                }
            }
            boolean done = false;
            for (int i = 0; (i < urCount) && !done; i++) {
                r = unmatchedRows[i];
                for (c = 0; c < N; ++c) {
                    if (w[r][c] - rDec[r] + cInc[c] < slack[c]) {
                        slack[c] = w[r][c] - rDec[r] + cInc[c];
                        if (slack[c] == 0) {
                            par[c] = r;
                            if (matchC[c] == -1) {
                                done = true;
                                break;
                            }
                            unmatchedRows[urCount++] = matchC[c];
                        }
                    }
                }
            }
            if (c == N) { // no augmenting path
                int del = oo;
                for (c = 0; c < N; ++c) {
                    if (par[c] == -1 && slack[c] < del) {
                        del = slack[c];
                    }
                }
                for (c = 0; c < N; ++c) {
                    if (par[c] != -1) {
                        cInc[c] += del;
                    }
                }
                for (int i = 0; i < urCount; ++i) {
                    rDec[unmatchedRows[i]] += del;
                }
            } else {
                r = par[c];
                while (matchR[r] != -1) {
                    int nc = matchR[r];
                    matchC[c] = r;
                    matchR[r] = c;
                    c = nc;
                    r = par[c];
                }
                matchC[c] = r;
                matchR[r] = c;
                --rowsLeft;
            }
        }
        int result = 0;
        for (r = 0; r < M; ++r) {
            result += w[r][matchR[r]]; // if max, use maxValue-w[][]
        }
        return result;
    }

    private void clear(int fill) { // fill with some sensible maxValue
        for (int i = 0; i < M; ++i) {
            Arrays.fill(w[i],fill);
        }
        Arrays.fill(cInc, 0); Arrays.fill(matchC, -1);
        Arrays.fill(rDec, 0); Arrays.fill(matchR, -1);
        rowsLeft = M;
    }

    public static void main(String args[]) {
        MinWBPM myWork = new MinWBPM();
        myWork.work();
    }
}
```

```java
class SuffixArray {
    // Suffix Array O(nlogn)
    // Author: Howard Cheng
    // Converted to Java: Darko Aleksic
    // Notes:
    // - if p == sarray[i], then str[p..n-1] is ith suffix
    // - empty suffix not included! (sarray[0] != n)
    // - lcp[i]: length of common prefix of ith and (i-1)th suffixes
    // (lcp[0]=0)
    // - to find a pattern P in str, do binary search over suffixes
    // (look for a suffix that has P as its prefix) - O(|P|logn)
    // this is kinda slow - it is either not O(nlogn) or it has a huge
        // constant
    private static final int MAXN = 100010;
    private int[] bucket = new int[256];
    private int[] prm = new int[MAXN];
    private int[] count = new int[MAXN];
    byte[] bh = new byte[MAXN + 1];
    char[] str, pstr;
    int[] sarray, lcp;
    int n;

    // needs arrays of length of the str.length
    private void build_sarray() {
        int n = str.length;
        for (int i = 0; i < 256; i++) {
            bucket[i] = -1;
        }
        for (int i = 0; i < n; i++) {
            prm[i] = bucket[str[i]];
            bucket[str[i]] = i;
        }
        for (int a = 0, c = 0; a < 256; a++) {
            int i = bucket[a];
            while (i != -1) {
                int j = prm[i];
                prm[i] = c;
                bh[c++] = (byte) (i == bucket[a] ? 1 : 0);
                i = j;
            }
        }
        bh[n] = (byte) 1;
        for (int i = 0; i < n; i++) {
            sarray[prm[i]] = i;
```

```
    }
    int x = 0;
    for (int h = 1; h < n; h *= 2) {
        for (int i = 0; i < n; i++) {
            if ((bh[i] & 1) != 0) {
                x = i;
                count[x] = 0;
            }
            prm[sarray[i]] = x;
        }
        int d = n - h;
        int e = prm[d];
        prm[d] = e + count[e];
        count[e]++;
        bh[prm[d]] = (byte) (bh[prm[d]] | 2);
        int i = 0;
        while (i < n) {
            int j = i;
            while (j == i || ((bh[j] & 1) == 0 && j < n)) {
                d = sarray[j] - h;
                if (d >= 0) {
                    e = prm[d];
                    prm[d] = e + count[e];
                    count[e]++;
                    bh[prm[d]] = (byte)(bh[prm[d]] | 2);
                }
                j++;
            }
            j = i;
            while (j == i || ((bh[j]&1) == 0 && j < n)){
                d = sarray[j] - h;
                if (d >= 0 && (bh[prm[d]] & 2) != 0){
                    for (e = prm[d] + 1; bh[e] == (byte) 2; e++);
                    for (int f = prm[d] + 1; f < e; f++) {
                        bh[f] = (byte)(bh[f] & 1);
                    }
                }
                j++;
            }
            i = j;
        }
        for (i = 0; i < n; i++) {
            sarray[prm[i]] = i;
            if (bh[i] == (byte) 2) {
                bh[i] = (byte) 3;
            }
        }
    }
    int h = 0;
    for (int i = 0; i < n; i++) {
        int e = prm[i];
        if (e > 0) {
            int j = sarray[e - 1];
            while (i + h < n && j + h < n && str[i + h] == str[j + h])
                h++;
            lcp[e] = h;
            if (h > 0)
                h--;
        }
    }
    lcp[0] = 0;
}

private boolean find() {
        int lo = 0;
        int hi = n;
        int plen = pstr.length;
        while (hi > lo) {
            int mid = lo + (hi - lo) / 2;
            int comp = 0;
            for (int i = 0; i < plen && mid + i < n && comp == 0; i++) {
                if (str[sarray[mid] + i] < pstr[i])
                    comp = -1;
                if (str[sarray[mid] + i] > pstr[i])
                    comp = 1;
            }
            if (comp == 0)
                return true;
            if (comp < 0)
                lo = mid + 1;
            else
                hi = mid;
        }
        return false;
    }

    private void saExample() {
        String s = "asdfadfsdaf";
        str = s.toCharArray();
        n = s.length();
        sarray = new int[n];
        lcp = new int[n];
        build_sarray();
        for (int i = 0; i < n; i++) {
            System.out.println(s.substring(sarray[i]));
        }
        System.out.println();
        pstr = "fa".toCharArray();
        System.out.println(find());
    }

    public static void main(String args[]) {
        SuffixArray sa = new SuffixArray();
        sa.saExample();
    }
}
```

```
//----------------------------------------------------------------------------
// POWERS OF TWO AND COMBINATIONS
// This file includes nifty bit manipulation algorithms to calculate:
//    - power of 2 floor and ceiling for integer
//    - determining whether or not an integer is a power of 2
//    - next integer with same amout of 1 bits (snoob)
// These algorithms are courtesy of "Hacker's Delight" by Henry S. Warren Jr.
// (Addison-Wesley 2003 ISBN 0-201-91465-4)
// Author:  Sonny Chan
// Date:    November 12, 2003
//----------------------------------------------------------------------------

// integer power of 2 floor function
unsigned int p2floor(unsigned int x)
{
   x = x | (x >> 1);
   x = x | (x >> 2);
   x = x | (x >> 4);
   x = x | (x >> 8);
   x = x | (x >> 16);
```

```
    return x - (x >> 1);
}

// integer power of 2 ceiling function
unsigned int p2ceiling(unsigned int x)
{
    x -= 1;
    x = x | (x >> 1);
    x = x | (x >> 2);
    x = x | (x >> 4);
    x = x | (x >> 8);
    x = x | (x >> 16);
    return x + 1;
}

// determines whether an integer is a power of 2
bool ispower2(unsigned int x)
{
    return (!(x & (x-1)) && x);
}

//------------------------------------------------------------------------
// calculates the next integer with the same number of 1-bits
unsigned int snoob(unsigned int x)
{
    unsigned int smallest, ripple, ones;
                                    // x = xxx0 1111 0000
    smallest = x & -x;              //     0000 0001 0000
    ripple = x + smallest;          //     xxx1 0000 0000
    ones = x ^ ripple;              //     0001 1111 0000
    ones = (ones >> 2) / smallest;  //     0000 0000 0111
    return ripple | ones;           //     xxx1 0000 0111
}
//------------------------------------------------------------------------
```

```
/*************************************************************************
BINARY SEARCH
Submitted March 21, 2004 by Kelly Poon
Original source courtesy of The University of Alberta
*************************************************************************/

/* Returns non-zero if x is found, and zero otherwise.  If x is found, then
A[index] = x.  If not, then index is the place x should be inserted into A. */
int bin_search(int *A, int n, int x, int *index){
    int low, up, mid;

    if (n <= 0 || x < A[0]) { *index = 0; return 0; }
    if (A[n-1] < x) { *index = n; return 0; }
    if (x == A[n-1]) { *index = n-1; return 1; }
    for(low = 0, up = n-1; low + 1 < up;){
      mid = (low+up)/2;
      if (A[mid] <= x) low = mid;
      else up = mid;
    }
    if (A[low] == x) { *index = low; return 1; }
    else { *index = up; return 0; }
}
```

```
/*************************************************************************
GEOMETRY ROUTINES
*************************************************************************/
```

```
/* closest point to c on line ab */
Point closest_pt_iline(Point a, Point b, Point c) {
  Point p;
  double dp;
  b.x -= a.x;
  b.y -= a.y;
  dp = (b.x*(c.x-a.x) + b.y*(c.y-a.y)) / (SQR(b.x)+SQR(b.y));
  p.x = b.x*dp + a.x;
  p.y = b.y*dp + a.y;
  return p;
}

/* reflection of c across ab */
Point reflect(Point a, Point b, Point c) {
  Point d, p;
  d = closest_pt_iline(a,b,c);
  p.x = 2.0*d.x - c.x;
  p.y = 2.0*d.y - c.y;
  return p;
}

/* rotation of p around o */
Point rotate_2d(Point p, Point o, double theta){
  double m[2][2];
  Point r;
  m[0][0] = m[1][1] = cos(theta);
  m[0][1] = -sin(theta);
  m[1][0] = -m[0][1];
  p.x -= o.x;
  p.y -= o.y;
  r.x = m[0][0] * p.x + m[0][1] * p.y + o.x;
  r.y = m[1][0] * p.x + m[1][1] * p.y + o.y;
  if(fabs(r.x) < EPS) r.x = 0;
  if(fabs(r.y) < EPS) r.y = 0;
  return r;
}

/* point in polygon */
#define BOUNDARY 1 // what to return for boundary points

int pt_in_poly(Point *p, int n, Point a) {
  int i, j, c = 0;
  for (i = 0, j = n-1; i < n; j = i++) {
    if (dist_2d(p[i],a)+dist_2d(p[j],a)-dist_2d(p[i],p[j]) < EPS)
      return BOUNDARY;
    if ((((p[i].y<=a.y) && (a.y<p[j].y)) ||
         ((p[j].y<=a.y) && (a.y<p[i].y))) &&
        (a.x < (p[j].x-p[i].x) * (a.y - p[i].y)
               / (p[j].y-p[i].y) + p[i].x)) c = !c;
  }
  return c;
}

/* Pick's theorem */
void lat_poly_pick(Point *p, int n, long long *I, long long *B){
  int i, j, dx, dy;
  double A = fabs(area_poly(p, n));
  *B = 0;
  for(i = n-1, j = 0; j < n; i = j++){
    dx = abs(p[i].x - p[j].x);
    dy = abs(p[i].y - p[j].y);
    *B += gcd(dx,dy);
  }
```

```
  *I = A+1-*B/2.0;
}

/* tangents from point p to circle c, r */
void circ_tangents(Point c, double r, Point p, Point *a, Point *b) {
  double perp, para, tmp = dist2(p,c);
  para = r*r/tmp;
  perp = r*sqrt(tmp-r*r)/tmp;
  a->x = c.x + (p.x-c.x)*para - (p.y-c.y)*perp;
  a->y = c.y + (p.y-c.y)*para + (p.x-c.x)*perp;
  b->x = c.x + (p.x-c.x)*para + (p.y-c.y)*perp;
  b->y = c.y + (p.y-c.y)*para - (p.x-c.x)*perp;
}
```

```
/*****************************************************************************
JAVA GEOMETRY ROUTINES
*****************************************************************************/

/*
 * PtD - Point class (convex hull, 3 point circle, intersection of circles,
centroid/area of a polygon)
 * Seg - segment/ray/line class (distances/intersections)
 */
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;
   }

   /* 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
```

```java
        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)
     * NOTE: if you need original p[], save it somewhere!!!
     */
    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) {
        if (end - begin <= 1) return;
        int 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++];
```

```java
        } 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];
    }

    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;
    }
}

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) {
```

```java
        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)
     * @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;
    }
}
```

```c
*******************************************************************************
3D GEOMETRY ROUTINES
Submitted March 21, 2004 by Kelly Poon
Original source courtesy of The University of Alberta
*******************************************************************************/

#include <math.h>

#define EPS 1E-8
#define pt(a) &(a.x), &(a.y), &(a.z)

struct Point{
  double x, y, z;
  Point(){};
  Point(double xi, double yi, double zi){x = xi; y = yi; z = zi;}
};

Point operator + (const Point& a, const Point& b) {
  return Point(a.x + b.x, a.y + b.y, a.z + b.z);
}

Point operator * (double k, const Point& a) {
  return Point(k*a.x, k*a.y, k*a.z);
}

Point operator - (const Point& a, const Point& b) {
  return Point(a.x - b.x, a.y - b.y, a.z - b.z);
}

Point operator * (Point a, double k) {
  return (k*a);
}

Point operator / (Point a, double k) {
  return (1.0/k)*a;
}

double dot(const Point& a, const Point& b) {
  return a.x*b.x + a.y*b.y + a.z*b.z;
}

Point cross(const Point& a, const Point& b) {
  return Point(a.y*b.z-b.y*a.z, b.x*a.z-a.x*b.z, a.x*b.y-b.x*a.y);
}

double length2(const Point& a) {
  return dot(a,a);
}

double length(const Point& a) {
  return sqrt(dot(a,a));
}

Point closest_pt_iline(const Point& a, const Point& b, const Point& p) {
  double along = dot(b-a,p-a)/length2(b-a);
  return (b-a)*along + a;
}

Point closest_pt_seg(const Point& a, const Point& b, const Point& p) {
  double along;
  if (length2(b-a) < EPS) return a;
  along = dot(b-a,p-a)/length2(b-a);
  if (along < 0) along = 0;
  if (along > 1) along = 1;
  return (b-a)*along + a;
}

/* plane represented by a normal and a point on plane */
Point closest_pt_plane(const Point& norm, const Point& a, const Point& p) {
  Point res = cross(cross(norm,p-a),norm);
  if (length2(res) < EPS) return a;
  return res*dot(res,p-a)/length2(res);
```

```
}

/* plane represented by three points */
Point closest_pt_plane(const Point& a, const Point& b, const Point& c, const
Point& p) {
  Point norm;

  norm = cross(b-a,c-a);
  /*assert(length2(norm) > EPS);*/  // collinearity
  return closest_pt_plane(norm,a,p);
}

/* returns number of intersections and the intersections*/
int sphere_iline_isect(const Point& c, double r, const Point& a, const Point& b,
                      Point *p, Point *q) {
  Point vec, mid = closest_pt_iline(a,b,c);

  if (length2(c-mid) > r*r) return 0;
  vec = (a-b)*sqrt((r*r - length2(c-mid))/length2(a-b));
  *p = mid + vec;
  *q = mid - vec;
  return ((length2(vec) > EPS) ? 2 : 1);
}

/* project point p to the plane defined by a, b and c */
Point to_plane(const Point& a, const Point& b, const Point& c, const Point& p) {
  Point norm, ydir, xdir, res;

  norm = cross(b-a,c-a);
  /*assert(length2(norm) > EPS);*/  // collinearity
  xdir = (b-a)/length(b-a); // create orthonormal vectors
  ydir = cross(norm,xdir);
  ydir = ydir/length(ydir);
  res.x = dot(p-a,xdir);
  res.y = dot(p-a,ydir);
  res.z = 0;
  return res;
}

/* given two lines in 3D space, find distance of closest approach */
double line_line_dist(const Point& a, const Point& b, const Point& c, const
Point& d) {
  Point perp = cross(b-a,d-c);

  if (length2(perp) < EPS)  /* parallel */
    perp = cross(b-a,cross(b-a,c-a));
  if (length2(perp) < EPS) return 0; /* coincident */

  return fabs(dot(a-c,perp))/length(perp);
}

/* same as line_line_dist, but returns the points of closest approach */
double closest_approach(const Point& a, const Point& b, const Point& c, const
Point& d,
                     Point *p, Point *q) {
  double s = dot(d-c,b-a), t = dot(a-c,d-c);
  double num, den, tmp;

  den = length2(b-a)*length2(d-c) - s*s;
  num = t*s - dot(a-c,b-a)*length2(d-c);
  if (fabs(den) < EPS) { /* parallel */
    *p = a;
    *q = (d-c)*t/length2(d-c) + c;
    if (fabs(s) < EPS) *q = a;  /* coincident */
  } else {  /* skew */
    tmp = num/den;
    *p = a + (b-a)*tmp;
    *q = c + (d-c)*(t + s*tmp)/length2(d-c);
  }
  return length(*p-*q);
}

/* is the point p on the infinite line ab? */
int on_iline(const Point& a, const Point& b, const Point& p) {
  return (length2(p-closest_pt_iline(a,b,p)) < EPS);
}

/* is the point p on the segment ab? */
int on_seg(const Point& a, const Point& b, const Point& p) {
  return (length(a-p) + length(p-b) - length(a-b) < EPS);
}

/* Given a plane and a line ab, determine if the two intersect,
   and if so, find the single point of intersection */
int plane_iline_isect(const Point& norm, const Point& ori, const Point& a, const
Point& b, Point *p) {
  double along, den = dot(norm,b-a);

  if (fabs(den) < EPS) { /* parallel */
    if (length2(cross(ori-a,b-a)) < EPS) return -1; /* coincident */
    return 0;  /* non-intersecting */
  }
  along = dot(norm,ori-a)/den;

  /* if you want to intersect a plane with a finite segment,
     check that (along <= 1 && along => 0) */
  *p = a + along*(b-a);
  return 1;
}
```

```
/* triangulate.h  - triangulates a polygon in O(n^2) time */
/*                  (note: fails on degenerate case of 3 collinear points) */
#include <list>
#include <vector>

using namespace std;

#define EPS 1e-8
#define ORDER 1 /* 1: cw, -1: ccw */

struct Point {
  double x, y;
};
struct Triangle {
  Point p[3];
};

/* classifies p as either being -1 left of, 1 right of or 0 on the line ab. */
int leftRight(Point &a, Point &b, Point &p){
  double res = ((b.x - a.x)*(p.y - a.y) - (p.x - a.x)*(b.y - a.y));
  if (res > EPS) return -1;
  else if (res < -EPS) return 1;
  return 0;
}

/* returns non-0 if b in the sequence a->b->c is concave, 0 for convex. */
int isConcave(Point &a, Point &b, Point &c){
```

```
  return (ORDER*leftRight(a, b, c) <= 0);
}

/* returns non-zero if point p is located on or inside the triangle <a b c>. */
int isInsideTriangle(Point &a, Point &b, Point &c, Point &p){
  int r1 = leftRight(a, b, p);
  int r2 = leftRight(b, c, p);
  int r3 = leftRight(c, a, p);
  return ((ORDER*r1 >= 0) && (ORDER*r2 >= 0) && (ORDER*r3 >= 0));
}

/* P - n cw-ordered points of a polygon (n>=3, P modified during function
   T - n-2 triangles, returns the triangulation of P */
void triangulate(list<Point> &P, vector<Triangle> &T){
  list<Point>::iterator a, b, c, q;
  Triangle t;

  T.clear();
  if (P.size() < 3) return;

  for (a=b=P.begin(), c=++b, ++c; c != P.end(); a=b, c=++b, ++c) {
    if (!isConcave(*a, *b, *c)) {
      for (q = P.begin(); q != P.end(); q++) {
        if (q == a) { ++q; ++q; continue; }
        if (isInsideTriangle(*a, *b, *c, *q)) break;
      }
      if (q == P.end()) {
        t.p[0] = *a; t.p[1] = *b; t.p[2] = *c;
        T.push_back(t);
        P.erase(b);
        b = a;
        if (b != P.begin()) b--;
      }
    }
  }
}
```

```
/* matrix.h - contains matrix and vector maths
   NOTE: be careful using homogenous coords */
#include <vector>
#include <math.h>
using namespace std;
#ifndef MATRIX_H
#define MATRIX_H
#define Matrix vector < vector<double> >
#define Vector vector<double>

bool ludcmp(Matrix& a, vector<int>& indx, double& d);
void lubksb(const Matrix& a, const vector<int>& indx, Vector& b);

Vector operator+(const Vector& v1, const Vector& v2){
    Vector ret(v1.size());
    for(int i = 0; i < v1.size(); i++) ret[i] = v1[i] + v2[i];
    return ret;
}
Vector operator-(const Vector& v1, const Vector& v2){
    Vector ret(v1.size());
    for(int i = 0; i < v1.size(); i++) ret[i] = v1[i] - v2[i];
    return ret;
}
Vector operator*(const double d, const Vector& v){
    Vector ret(v.size());
    for(int i = 0; i < v.size(); i++) ret[i] = d*v[i];
    return ret;
}
double dot(const Vector& v1, const Vector& v2){
    double ret = 0.0;
    for(int i = 0; i < v1.size(); i++) ret += v1[i]*v2[i];
    return ret;
}
double length(const Vector& v){
    return sqrt(dot(v,v));
}
Matrix operator+(const Matrix& m1, const Matrix& m2){
    Matrix ret(m1.size(), Vector(m1[0].size()));
    for(int i = 0; i < m1.size(); i++)
        for(int j = 0; j < m1[0].size(); j++)
            ret[i][j] = m1[i][j] + m2[i][j];
    return ret;
}
Matrix operator-(const Matrix& m1, const Matrix& m2){
    Matrix ret(m1.size(), Vector(m1[0].size()));
    for(int i = 0; i < m1.size(); i++)
        for(int j = 0; j < m1[0].size(); j++)
            ret[i][j] = m1[i][j] - m2[i][j];
    return ret;
}
Matrix operator*(const double s, const Matrix& m){
    Matrix ret(m.size(), Vector(m[0].size()));
    for(int i = 0; i < m.size(); i++)
        for(int j = 0; j < m[0].size(); j++)
            ret[i][j] = s*m[i][j];
    return ret;
}
Matrix operator*(const Matrix& m1, const Matrix& m2){
    Matrix ret(m1.size(), Vector(m2[0].size(), 0.0));
    for(int r = 0; r < m1.size(); r++)
        for(int c = 0; c < m2[0].size(); c++)
            for(int i = 0; i < m2.size(); i++)
                ret[r][c] += m1[r][i]*m2[i][c];
    return ret;
}
Vector operator*(const Matrix& m, const Vector& v){
    Vector ret(m.size(), 0.0);
    for(int r = 0; r < m.size(); r++)
        for(int c = 0; c < m[0].size(); c++)
            ret[r] += m[r][c]*v[c];
    return ret;
}
Matrix id(int N){
    Matrix ret(N, Vector(N, 0.0));
    for(int i = 0; i < N; i++)
        ret[i][i] = 1.0;
    return ret;
}

/* inverts m (assumes m is square) */
Matrix inverse(const Matrix& m){
    int N = m.size();
    Matrix mT = m, inv(N, Vector(N, 0.0));
    double d;
    vector<int> indx(N);

    ludcmp(mT, indx, d);
    for(int j = 0; j < N; j++){
        Vector col(N, 0.0); col[j] = 1.0;
        lubksb(mT, indx, col);
        for(int i = 0; i < N; i++) inv[i][j] = col[i];
```

```
    }
    return inv;
}

/* returns the determinant of m, an NxN matrix in O(N^3) */
double determinant(const Matrix& m){
    int N = m.size();
    Matrix mT = m;
    double d;
    vector<int> indx(N);
    ludcmp(mT, indx, d);
    for(int j = 0; j < N; j++) d*= mT[j][j];
    return d;
}

/* return the solution, x, to Ax = b (assumes A is NxN and b is N) */
Vector solve(const Matrix& A, const Vector& b){
    int N = A.size();
    Matrix aT = A;
    Vector x = b;
    double d;
    vector<int> indx(N);
    ludcmp(aT, indx, d);
    lubksb(aT, indx, x);
    return x;
}

#endif
/* lu.h - LU Decomposition */
#include "matrix.h"
#define TINY 1.0e-20

/* replaces A with the LU decomposition of A rowwise permutation of A
   indx records the row permutation effected by pivoting
   d is 1.0 if n interchanges is even, else -1 */
bool ludcmp(Matrix& A, vector<int>& indx, double& d)
{
    int i, j, k, imax = 0, n = A.size();
    double big, dum, sum, temp;
    vector<double> vv(n+1);
    d = 1.0;
    for(i = 0; i < n; i++){
        big = 0.0;
        for(j = 0;j < n; j++)
            if((temp = fabs(A[i][j])) > big) big = temp;
        if(big == 0.0) return false; /* singular matrix */
        vv[i] = 1.0/big;
    }
    for(j = 0; j < n; j++){
        for(i = 0; i < j; i++){
            sum = A[i][j];
            for(k = 0; k < i; k++) sum -= A[i][k]*A[k][j];
            A[i][j] = sum;
        }
        big = 0.0;
        for(i = j; i < n; i++){
            sum = A[i][j];
            for(k = 0; k < j; k++) sum -=A[i][k]*A[k][j];
            A[i][j] = sum;
            if((dum = vv[i]*fabs(sum)) >= big){
                big = dum;
                imax = i;
            }
        }
        if(j != imax){
            for(k = 0; k < n; k++){
                dum = A[imax][k];
                A[imax][k] = A[j][k];
                A[j][k] = dum;
            }
            d = -d;
            vv[imax] = vv[j];
        }
        indx[j] = imax;
        if(A[j][j] == 0.0) A[j][j]=TINY;
        if(j != n-1){
            dum = 1.0/(A[j][j]);
            for(i = j+1; i < n; i++) A[i][j] *= dum;
        }
    }
    return true;
}

/* solves Ax=b (returns x in b) */
void lubksb(const Matrix& A, const vector<int>& indx, Vector& b)
{
    int i, ip, j, ii=0, n = A.size();
    double sum;
    for(i = 0; i < n; i++){
        ip = indx[i];
        sum = b[ip];
        b[ip] = b[i];
        if(ii) for(j = ii-1; j < i; j++) sum -= A[i][j]*b[j];
        else if(sum) ii = i+1;
        b[i] = sum;
    }
    for(i = n-1; i >= 0; i--){
        sum = b[i];
        for(j = i+1; j < n; j++) sum -= A[i][j]*b[j];
        b[i] = sum/A[i][i];
    }
}
```

```
/* rotations.h - makes rotation matrices */
#include "matrix.h"
#include <math.h>

/* rotations about main axes */
Matrix rotX(double angle){
    double cosa = cos(angle), sina = sin(angle);
    Matrix ret = id(4);
    ret[1][1] = cosa; ret[1][2] = -sina;
    ret[2][1] = sina; ret[2][2] = cosa;
    return ret;
}
Matrix rotY(double angle){
    double cosa = cos(angle), sina = sin(angle);
    Matrix ret = id(4);
    ret[0][0] = cosa; ret[0][2] = sina;
    ret[2][0] = -sina; ret[2][2] = cosa;
    return ret;
}
Matrix rotZ(double angle){
    double cosa = cos(angle), sina = sin(angle);
    Matrix ret = id(4);
    ret[0][0] = cosa; ret[0][1] = -sina;
    ret[1][0] = sina; ret[1][1] = cosa;
```

```java
    return ret;
}

/* rotation about arbitrary axis (flattens to z) */
Matrix rot(double angle, Vector axis){
    double u = axis[0], v = axis[1], w = axis[2];
    double u2 = u*u, v2 = v*v, w2 = w*w, len2 = u2 + v2 + w2, len = sqrt(len2);
    double cosa = cos(angle), sina = sin(angle);

    Matrix ret = id(4);
    ret[0][0] = (u2+(v2+w2)*cosa)/len2;
    ret[0][1] = (u*v*(1-cosa)-w*len*sina)/len2;
    ret[0][3] = (u*w*(1-cosa)+v*len*sina)/len2;
    ret[1][0] = (u*v*(1-cosa)+w*len*sina)/len2;
    ret[1][1] = (v2+(u2+w2)*cosa)/len2;
    ret[1][2] = (v*w*(1-cosa)-u*len*sina)/len2;
    ret[2][0] = (u*w*(1-cosa) - v*len*sina)/len2;
    ret[2][1] = (v*w*(1-cosa)+ u*len*sina)/len2;
    ret[2][2] = (w2+(u2+v2)*cosa)/len2;

    return ret;
}

/* rotation about the axis parallel to axis that goes through point */
Matrix rot(double angle, Vector axis, Vector point){
    double u = axis[0], v = axis[1], w = axis[2];
    double a = point[0], b = point[1], c = point[2];
    double u2 = u*u, v2 = v*v, w2 = w*w, len2 = u2 + v2 + w2, len = sqrt(len2);
    double cosa = cos(angle), sina = sin(angle);

    Matrix ret = rot(angle, axis);
    ret[0][3] = (a*(v2+w2)-u*(b*v-c*w)+(u*(b*v+c*w)-a*(v2+w2))*cosa+(b*w-
c*v)*len*sina)/len2;
    ret[1][3] = (b*(u2+w2)-v*(a*u+c*w)+(v*(a*u+c*w)-b*(u2+w2))*cosa+(c*u-
a*w)*len*sina)/len2;
    ret[2][3] = (c*(u2+v2)-w*(a*u+b*v)+(w*(a*u+b*v)-c*(u2+v2))*cosa+(a*v-
b*u)*len*sina)/len2;

    return ret;
}
```

```java
class Primes {
   /**
    * Primes: generate, find number of divisors and totient function
    * nextPerm() is in here (had no idea where to put it
    */
   private final static int SIEVE_SIZE = 46341; // up to 10 mil is OK
   private final static int PRIMES_SIZE = 4792;
   private boolean[] nonPrimes = new boolean[SIEVE_SIZE];
   private int[] primes = new int[PRIMES_SIZE];

   void primesExample() {
      sieve(); // run sieve
      int m = 0;
      for (int i = 0; i < SIEVE_SIZE; i++) {
         if (!nonPrimes[i])
            primes[m++] = i;
      }
      // System.out.println(m); // find out the PRIME_SIZE
      int n = 256; // or whatever we want it to be, up to SIEVE_SIZE ^ 2
      long div = 1; // number of divisors
      long phi = 1; // totient function
      int tmp = n;
      for (int i = 0; i < PRIMES_SIZE; i++) {
         if (tmp % primes[i] == 0) {
            int cnt = 1;
            tmp /= primes[i];
            while (tmp % primes[i] == 0) {
               cnt++;
               tmp /= primes[i];
            }
            div *= cnt + 1;
            phi *= (primes[i] - 1);
            for (int j = 1; j < cnt; j++) {
               phi *= primes[i];
            }
         }
         if (tmp == 1)
            break;
      }
      if (tmp != 1) { // it's prime
         div <<= 1;
         phi *= (tmp - 1);
      }
      System.out.println("phi(" + n + ")=" + phi + " div(" + n + ")=" + div);
   }

   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;
            }
         }
      }
   }

   /* extended gcd */
   private long[] egcd(long a, long b) {
      if (b == 0) {
         long[] ret = new long[3];
         ret[0] = a; ret[1] = 1; ret[2] = 0;
         return ret;
      }
      long[] q = egcd(b, a % b);
      long[] ret = new long[3];
      ret[0] = q[0]; ret[1] = q[2];
      ret[2] = q[1] - (a / b) * q[2];
      return ret;
   }

   /* mod inverse */
   private long inverse(long a, long n) {
      long[] t = egcd(a, n);
      if (t[0] > 1)
         return 0;
      long r = t[1] % n;
      return (r < 0 ? r + n : r);
   }
```

```java
    /* solves ax=b(mod n) for x - modify to return only one value if needed */
    private ArrayList<Long> msolve(long a, long b, long n) {
        if (n < 0)
            n = -n;
        long[] t = egcd(a, n);
        ArrayList<Long> r = new ArrayList<Long>();
        if (b % t[0] != 0)
            return r;
        long x = (b / t[0] * t[1]) % n;
        if (x < 0)
            x += n;
        for (long i = 0; i < t[0]; i++)
            r.add((x + i * n / t[0]) % n);
        return r;
    }

    /*
     * Linear Diophantine Equation Solver - Solves integer equations of the form
     * ax + by = c for integers x and y. Returns a long[3] containing the answer
     * (in [1] and [2]) and a flag (in [0]). If the returned flag is zero, then
     * there are no solutions. Otherwise, there is an infinite number of
     * solutions of the form x = [1] + k * b / [0], y = [2] - k * a / [0]; for
     * all k
     */
    private long[] ldioph(long a, long b, long c) {
        long[] t = egcd(a, b);
        if (c % t[0] != 0)
            return new long[] { 0, 0, 0 };
        t[1] *= c / t[0];
        t[2] *= c / t[0];
        return t;
    }

    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]) {
            if (--i == 0) return false;
        }
        int j = a.length;
        while (a[j - 1] <= a[i - 1]) {
            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;
    }
}
```

```
Pollard's Rho (from Wikipedia):
Inputs: n,the integer to be factored and f(x), a pseudo-random function modulo n
(f(x)=x^2+c, c!=0, c!=-2 works fine)
Output: a non-trivial factor of n, or failure.
1. x ← 2, y ← 2; d ← 1
2. While d = 1:
  1. x ← f(x)
  2. y ← f(f(y))
  3. d ← GCD(|x − y|, n)
3. If d = n, return failure.
4. Else, return d.
Note that this algorithm will return failure for all prime n, but it can also
fail for composite n. In that case, use a different f(x) and try again.
```

```c
/* Chinese remainder theorem for x % m[i] = a[i] */
int cra(int n, int *m, int *a){
  int x, i, k, prod, temp;
  int *gamma, *v;

  gamma = (int *)malloc(n*sizeof(int));
  v     = (int *)malloc(n*sizeof(int));

  /* compute inverses */
  for (k = 1; k < n; k++) {
    prod = m[0] % m[k];
    for (i = 1; i < k; i++) {
      prod = (prod * m[i]) % m[k];
    }
    extended_euclid(prod, m[k], gamma+k, &temp);
    gamma[k] %= m[k];
    if (gamma[k] < 0) {
      gamma[k] += m[k];
    }
  }

  /* compute coefficients */
  v[0] = a[0];
  for (k = 1; k < n; k++) {
    temp = v[k-1];
    for (i = k-2; i >= 0; i--) {
      temp = (temp * m[i] + v[i]) % m[k];
      if (temp < 0) {
        temp += m[k];
      }
    }
    v[k] = ((a[k] - temp) * gamma[k]) % m[k];
    if (v[k] < 0) {
      v[k] += m[k];
    }
  }

  /* convert from mixed-radix representation */
  x = v[n-1];
  for (k = n-2; k >= 0; k--) {
    x = x * m[k] + v[k];
  }
  free(gamma);
  free(v);
  return x;
}
```

```c
/* solve a cubic equation */
typedef struct{
  int n;       /* Number of solutions */
  double x[3]; /* Solutions */
} Result;

double PI; // PI = acos(-1);
```

```
Result solve_cubic(double a, double b, double c, double d){
  Result s;
  long double a1 = b/a, a2 = c/a, a3 = d/a;
  long double q = (a1*a1 - 3*a2)/9.0, sq = -2*sqrt(q);
  long double r = (2*a1*a1*a1 - 9*a1*a2 + 27*a3)/54.0;
  double z = r*r-q*q*q;
  double theta;

  if(z <= 0){
    s.n = 3;
    theta = acos(r/sqrt(q*q*q));
    s.x[0] = sq*cos(theta/3.0) - a1/3.0;
    s.x[1] = sq*cos((theta+2.0*PI)/3.0) - a1/3.0;
    s.x[2] = sq*cos((theta+4.0*PI)/3.0) - a1/3.0;
  } else {
    s.n = 1;
    s.x[0] = pow(sqrt(z)+fabs(r),1/3.0);
    s.x[0] += q/s.x[0];
    s.x[0] *= (r < 0) ? 1 : -1;
    s.x[0] -= a1/3.0;
  }
  return s;
}
```

```
/** Longest Increasing Subsequence O(n log(log n))
Author: Darko Aleksic
Adapted by: Sean McIntyre
Notes:
- see comment below for how to convert to
Longest Nondecreasing Subsequence*/
public class LIS {
    Item[] list;
    public static void main(String[] args) {
        new LIS().lisExample();
    }
    public void lisExample() {
        int[] nums = new int[] { 5, 9, 1, 3, 4, 10, 10, 11 };
        Item end = lis(nums);
        int len = end.len;
        System.out.println(len); // length of LIS
        System.out.write('-');
        StringBuilder sb = new StringBuilder();
        while (end != null) { // loop through LIS backwards
            sb.insert(0, end.toString());
            sb.insert(0, '\n'); // insert at front
            end = end.prev;
        }
        System.out.println(sb);
    }
    public Item lis(int[] nums) {
        int index, n, len = 1;
        list = new Item[nums.length];
        list[0] = new Item(nums[0], null, len);
        for (int i = 1; i < nums.length; i++) {
            n = nums[i];
            index = getIndex(n, len);
            if (index > 0)
                list[index] = new Item(n, list[index - 1], len);
            else
                list[index] = new Item(n, null, len);
            if (index == len)
                len++;
        }
        return list[len - 1];
    }

    public int getIndex(int n, int len) {
        Item tmp = new Item(n, null, 0);
        int lo = 0;
        int hi = len;
        int mid = 0;
        while (hi > lo) {
            mid = (hi + lo) >> 1;
            if (list[mid].compareTo(tmp) == 0)
                break;
            if (list[mid].compareTo(tmp) < 0) {
                lo = mid + 1;
            } else {
                hi = mid;
            }
        }
        // set latter condition <= for
        // longest nondecreasing subsequence
        while (mid < len && list[mid].compareTo(tmp) < 0)
            mid++;
        return mid;
    }
}

class Item {
    int val, len;
    Item prev;
    public Item(int val, Item prev, int len) {
        this.val = val;this.prev = prev;this.len = len;
    }
    public int compareTo(Item li2) {
        return val - li2.val;
    }
    public String toString() {
        return "" + val;
    }
}
```

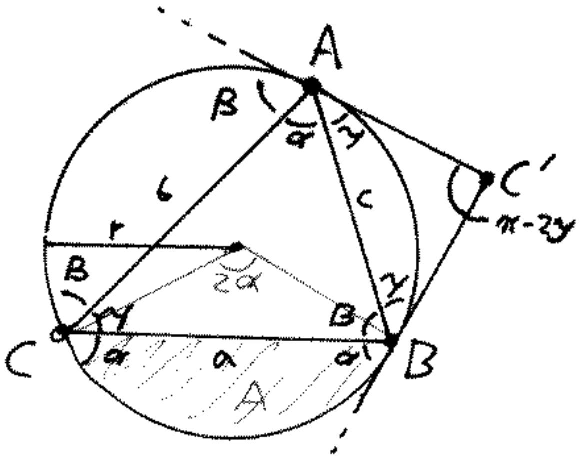
```
/* Great Circle distance (lat[-90,90], long[-180,180])  */
double greatcircle(double lt1, double lo1, double lt2, double lo2, double r) {
    double a = PI*(lt1/180.0), b = PI*(lt2/180.0);
    double c = PI*((lo2-lo1)/180.0);
    return r*acos(sin(a)*sin(b) + cos(a)*cos(b)*cos(c));
}
```

```
// Simplex Method for Linear Programming
// m - number of (less than) inequalities
// n - number of variables
// C - (m+1) by (n+1) array of coefficients:
// row 0 - objective function coefficients
// row 1:m - less-than inequalities
// column 0:n-1 - inequality coefficients
// column n - inequality constants (0 for objective function)
// X[n] - result variables
// return value - maximum value of objective function
// (-inf for infeasible, inf for unbounded)
//
#define MAXM 400 // leave one extra
#define MAXN 400 // leave one extra
```

```
#define EPS 1e-9
#define INF 1.0/0.0
double A[MAXM][MAXN];
int basis[MAXM], out[MAXN];
void pivot(int m, int n, int a, int b) {
   int i,j;
   for (i=0;i<=m;i++) if (i!=a) for (j=0;j<=n;j++) if (j!=b) {
      A[i][j] -= A[a][j] * A[i][b] / A[a][b];
   }
   for (j=0;j<=n;j++) if (j!=b) A[a][j] /= A[a][b];
   for (i=0;i<=m;i++) if (i!=a) A[i][b] = -A[i][b]/A[a][b];
   A[a][b] = 1/A[a][b];
   17
      i = basis[a];
   basis[a] = out[b];
   out[b] = i;
}
double simplex(int m, int n, double C[][MAXN], double X[]) {
   int i,j,ii,jj; // i,ii are row indexes; j,jj are column indexes
   for (i=1;i<=m;i++) for (j=0;j<=n;j++) A[i][j] = C[i][j];
   for (j=0;j<=n;j++) A[0][j] = -C[0][j];
   for (i=0;i<=m;i++) basis[i] = -i;
   for (j=0;j<=n;j++) out[j] = j;
   for(;;) {
      for (i=ii=1;i<=m;i++) {
         if (A[i][n]<A[ii][n]
         || (A[i][n]==A[ii][n] && basis[i]<basis[ii]))
            ii=i;
      }
      if (A[ii][n] >= -EPS) break;
      for (j=jj=0;j<n;j++)
         if (A[ii][j]<A[ii][jj]-EPS
            || (A[ii][j]<A[ii][jj]-EPS && out[i]<out[j]))
            jj=j;
      if (A[ii][jj] >= -EPS) return -INF;
      pivot(m,n,ii,jj);
   }
   for(;;) {
      for (j=jj=0;j<n;j++)
         if (A[0][j]<A[0][jj]
         || (A[0][j]==A[0][jj] && out[j]<out[jj]))
            jj=j;
         if (A[0][jj] > -EPS) break;
         for (i=1,ii=0;i<=m;i++)
            if (A[i][jj]>EPS &&
               (!ii || A[i][n]/A[i][jj]<A[ii][n]/A[ii][jj]-EPS ||
               (A[i][n]/A[i][jj]<A[ii][n]/A[ii][jj]+EPS
               && basis[i]<basis[ii])))
               ii=i;
         if (A[ii][jj] <= EPS) return INF;
         pivot(m,n,ii,jj);
   }
   for (j=0;j<n;j++) X[j] = 0;
   for (i=1;i<=m;i++) if (basis[i] >= 0) X[basis[i]] = A[i][n];
   return A[0][n];
}
void print(int m, int n, char *msg) { // not used -- debug only
   int i,j;
   printf(“%s\n”,msg);
   for(i=0;i<=m;i++) {
      for (j=0;j<=m;j++) printf(“ %10d”,i==j);
      for (j=0;j<=n;j++) printf(“ %10g”,A[i][j]);
      printf(“\n”);
   }
   for (i=0;i<=m;i++) printf(“ %10d”,basis[i]);
   for (j=0;j<n;j++) printf(“ %10d”,out[j]);
   printf(“\n”);
}
```

```
/*** STRONGLY CONNECTED COMPONENTS - Gilbert Lee*/
#define VI vector<int>
#define MAXN 1000
VI g[MAXN], curr;
vector< VI > scc;
int dfsnum[MAXN], low[MAXN], id;
char done[MAXN];
void visit(int x){
  curr.push_back(x);
  dfsnum[x] = low[x] = id++;
  for(size_t i = 0; i < g[x].size(); i++)
    if(dfsnum[g[x][i]] == -1){
      visit(g[x][i]);
      low[x] <?= low[g[x][i]];
    } else if(!done[g[x][i]])
      low[x] <?= dfsnum[g[x][i]];

  if(low[x] == dfsnum[x]){
    VI c; int y;
    do{
      done[y = curr[curr.size()-1]] = 1;
      c.push_back(y);
      curr.pop_back();
    } while(y != x);
    scc.push_back(c);
  }
}
void strong_conn(int n){
  memset(dfsnum, -1, n*sizeof(int));
  memset(done, 0, sizeof(done));
  scc.clear(); curr.clear();
  for(int i = id = 0; i < n; i++)
    if(dfsnum[i] == -1) visit(i);
}
int main(){
  int n, m, i, x, y;
  while(scanf("%d %d", &n, &m) == 2){
    for(i = 0; i < n; i++) g[i].clear();
    for(i = 0; i < m; i++){
      scanf("%d %d", &x, &y);
      g[x].push_back(y);
    }
    strong_conn(n);
    for(size_t i = 0; i < scc.size(); i++){
      printf("Component %d:", i+1);
      for(size_t j = 0; j < scc[i].size(); j++)
   printf(" %d", scc[i][j]);
      printf("\n");
    }
  }
  return 0;
}
```



$a^2 = b^2 + c^2 - 2bc\cos(\alpha)$

$a = b\cos(\gamma) + c\cos(\beta)$

$a/\sin(\alpha) = b/\sin(\beta) = c/\sin(\gamma)$

$A = (r^2 / 2) * (2\alpha - \sin(2\alpha))$

Area = sqrt(s(s-a)(s-b)(s-c)), s=(a+b+c)/2

= (AB X AC(\) / 2 (signed!)

$A \bullet B = AxBx + AyBy + AzBz$

$= A^T B$

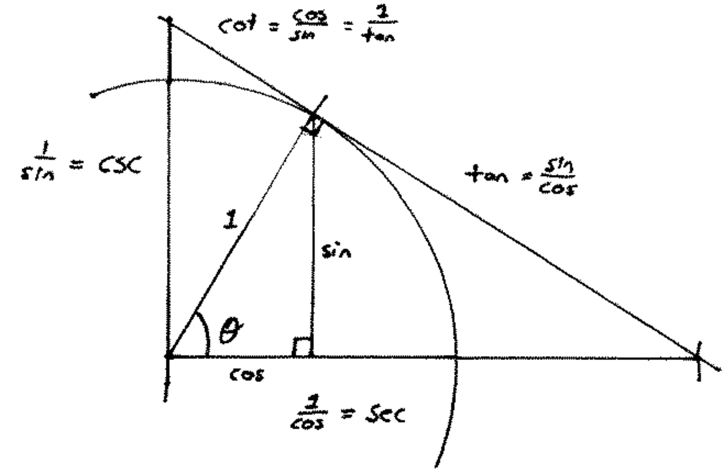
$= |A||B|\cos(\theta)$

$$A \times B = \begin{vmatrix} x & y & z \\ Ax & Ay & Az \\ Bx & By & Bz \end{vmatrix}$$

$|A \times B| = |A||B|\sin(\theta)$

$= |A||B|\mathrm{sqrt}(1-(A \bullet B)^2)$

= (in 2D) AxBy - AyBx




```
/*** Tree ID - Gilbert Lee (use for Isomorphism) ***/
typedef struct{  int n; list<int> adj[MAXN];} Tree;
string getTreeID(Tree t){
  multiset<string> s[MAXN]; multiset<string>::iterator it;
  set<int> leaf; set<int>::iterator j;
  char parent[MAXN]; string id[MAXN], res[2];
  int left = t.n, i, k, x;
  for(i = 0; i < t.n; i++) id[i] = "01";
  while(left > 2){
    memset(parent, 0, sizeof(parent));
    for(i = 0; i < t.n; i++) s[i].clear();
    leaf.clear();
    for(i = 0; i < t.n; i++)
      if(t.adj[i].size() == 1){
          leaf.insert(i);
          x = *t.adj[i].begin();
          s[x].insert(id[i]);
          parent[x] = 1; left--;
      }
    for(i = 0; i < t.n; i++)
      if(parent[i]){
          x = id[i].size();
          if(x > 2) s[i].insert(id[i].substr(1,x-2));
          for(id[i] = "0", it = s[i].begin(); it != s[i].end(); ++it)
          id[i] += *it; id[i] += '1';
      }
    for(i = 0; i < t.n; i++)
      if(leaf.count(i) == 1) t.adj[i].clear();
      else
          for(j = leaf.begin(); j != leaf.end(); ++j)
             for(k = 0; k < t.n; k++)
               t.adj[k].remove(*j);
  }
  for(i = x =  0; i < t.n; i++)
    if(parent[i]) res[x++] = id[i];

  if(left == 1) return res[0];
  return (res[0] < res[1]) ? res[0]+res[1] : res[1]+res[0];
}
```

Permutations	n distinct objects	$n!$
	a_i objects of type i, $\sum_{i=1}^{k} a_i = n$	$\frac{n!}{a_1!\cdot a_2!\cdots a_k!}$
Lists	n distinct objects, list of length k	$(n)_k = \frac{n!}{(n-k)!}$
	n distinct letters, words of length k	n^k
Subsets	k-element subsets of $[n]$	$\binom{n}{k}$
	k-element multisets with elements from $[n]$	$\binom{n+k-1}{k}$

Table 1: Enumeration formulas

Surjections	n distinct objects, k distinct boxes	$S(n,k)k!$
	n distinct objects, any number of distinct boxes	$\sum_{i=1}^{n} S(n,i)i!$
Compositions	n identical objects, k distinct boxes	$\binom{n-1}{k-1}$
	n identical objects, any number of distinct boxes	2^{n-1}
Set partitions	n distinct objects, k identical boxes	$S(n,k)$
	n distinct objects, any number of identical boxes	$B(n)$
Integer partitions	n identical objects, n identical boxes	$p_k(n)$
	n identical objects, any number of identical boxes	$p(n)$

Table 2: Enumeration formulas if no boxes are empty

Functions	n distinct objects, k distinct boxes	k^n
Weak compositions	n identical objects, k distinct boxes	$\binom{n+k-1}{k-1}$
Set partitions	n distinct objects, k identical boxes	$\sum_{i=1}^{k} S(n,i)$
Integer partitions	n identical objects, k identical boxes	$\sum_{i=1}^{k} p_i(n)$

Table 3: Enumeration formulas if empty boxes are allowed

Stirling numbers, second kind Number of ways to partition a set of n things into k nonempty subsets.

$$\left\{ {n \atop k} \right\} = k \left\{ {n-1 \atop k} \right\} + \left\{ {n-1 \atop k-1} \right\}$$

$$\left\{ {0 \atop 0} \right\} = 1 \qquad \left\{ {n \atop 0} \right\} = \left\{ {0 \atop k} \right\} = 0$$

0	1					
1	0	1				
2	0	1	1			
3	0	1	3	1		
4	0	1	7	6	1	
5	0	1	15	25	10	1

Stirling numbers, first kind Number of ways to partition n objects into k cycles.

$$\left[{n \atop k} \right] = (n-1) \left[{n-1 \atop k} \right] + \left[{n-1 \atop k-1} \right]$$

$$\left[{0 \atop 0} \right] = 1 \qquad \left[{n \atop 0} \right] = \left[{0 \atop k} \right] = 0$$

0	1					
1	0	1				
2	0	1	1			
3	0	2	3	1		
4	0	6	11	6	1	
5	0	14	50	35	10	1

Eulerian numbers Number of permutations of n elements having k ascents ($\pi_j < \pi_{j+1}$).

$$\left\langle {n \atop k} \right\rangle = (k+1) \left\langle {n-1 \atop k} \right\rangle + (n-k) \left\langle {n-1 \atop k-1} \right\rangle$$

$$\left\langle {n \atop 0} \right\rangle = 1$$

0	1					
1	1	0				
2	1	1	0			
3	1	4	1	0		
4	1	11	11	1	0	
5	1	26	66	26	1	0

Fibonacci numbers Those lovable devils.

n	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
$a[n]$	0	1	1	2	3	5	8	13	21	34	55	89	144	233	377	610	987	1597	2584

$$a[n] = a[n-1] + a[n-2] \qquad \varphi = \frac{1+\sqrt{5}}{2} \qquad a[n] = \lfloor \frac{\varphi^n}{\sqrt{5}} + \frac{1}{2} \rfloor$$

Catalin numbers

- Number of ways to match a pair of parenthesis.
- Number of different binary trees with $n+1$ leaves.
- Number of different monotonic paths in $n \times n$ grid (below or on diagonal).
- Number of convex triangulations.

n	0	1	2	3	4	5	6	7	8	9	10
$a[n]$	1	1	2	5	14	42	132	429	1430	4862	16796

$$C_n = \frac{1}{n+1}\binom{2n}{n} = \binom{2n}{n} - \binom{2}{n-1}$$

Bayes' theorem $P(A \cap B) = P(A)P(B|A) = P(B)P(A|B)$

Binomial distribution The probability of k successes in n independent trials where each trial has probability p. $P(X=k) = \binom{n}{k}p^k(1-p)^{n-k}$, $E(X) = np$.

Geometric distribution The probability that the kth independent trial is the first successful trial where each trial has probability p. $P(X=k) = (1-p)^{k-1}p$, $E(X) = 1/p$.

Negative binomial distribution The probability that the rth success occurs on the kth independent trial where each trial has probability p.
$P(X=k) = \binom{y-1}{r-1}p^r(1-p)^{k-r}, E(X) = r/p$.

Poisson distribution The probability of k independent events occurring in an interval that usually has λ events. $P(X=k) = \lambda^k e^{-\lambda}/k!$, $E(X) = \lambda$.

Hypergeometric distribution Suppose there is a bucket of N balls, r of which are red. Then this distribution models the probability of drawing k red balls after n random selections without replacement. $P(X=k) = \binom{r}{k}\binom{N-r}{n-k}/\binom{N}{n}$, $E(X) = nr/N$.