Virginia Tech
ACM World Finals 2004 Reference Document

Section C++ STL 3
* File/Stream I/O <stdio.h> 3
* Printf 3
* String 3
* Character Class Tests <ctype.h> 3
* Algorithm functions <algorithm> 3
* STL map/multimap 3
* STL set/multiset 4
* STL hash_set or hash_map 4
* STL List 4
* STL Vector/bit_vector 5
* STL Notes 5
Section Java 5
* Java Start 5
* Java java.io.StreamTokenizer 5
* Java java.math.BigInteger 6
* Java java.util.LinkedList 6
* Java java.util.TreeSet 6
* Java java.util.TreeMap 6
* Java java.util.Map.Entry 6
* Java java.util.ArrayList 6
* Java java.util.Iterator 6
* Java java.text.DecimalFormat 6
Section Graph Theory 7
* Prim's Minimal Spanning Tree 7
* Floyd’s with path finding 7
* MaxFlow 7
* Min distance 8
* Greg: graph.cpp 8
//ispath - returns 1 if there is a path between v and w 8
//Hpath – tests for hamiltonian path between v and w 9
//degree - returns number of edges from vertex 9
//Epath - returns 1 if euler path between v and w 9
//Connected - returns a list of lists 9
//bipartite - returns 1 if the graph is bipartite 10
//cuts – returns vector of single articulation or cut vertices 10
//bridge – returns a vector of bridges 10
//prim's algorithm for finding minimum spanning tree 11
//shortest distance - all pairs 11
//shortest distance - single source 12
* Greg: next_combination.cpp 12
Section Math 13
* Basic Trig 13
* include <cmath> 13

* Min distance from point to line(2points) 13
* Min distance from point to line(point slope) 13
* Area of a polygon.......... 13
* Circle from three points on radius 13
* Convert two points in parametric form of line.......... 14
* Find line equidistant from two points 14
* Line intersection.......... 14
* polyareaConvex 14
* polyArea2: returns twice the signed area of the polygon 15
* Point In Poly 16
* LineIntersect.......... 16
* lineProj – Point onto line 16
* Kevin: geom-pretty.cpp 16
// intersection of lines 17
// intersection of segments.......... 17
// tangent line to circle.......... 17
// point in circle 18
// point of closest approach to line 18
// distance of closest approach to line 18
// given sides, return triangle.......... 18
// find the incenter of a triangle 18
// convex hull, not very efficient 18
// area of a triangle 19
// split a polygon into triangles.......... 19
// find the area of a polygon (clkwise) 19
// intersection of polygons; not efficient 19
Section Other Algorithms.......... 20
* Choose.......... 20
* Partitions 20
* Permutations.......... 20
* A-star search (or UCS or BFS) 20
* Rotate matrix.......... 21
* GCD and LCM.......... 21
* Day Of Week 21
* String search-replace.......... 21
* Prime Factor.......... 22
* Generate Combinations.......... 22
* Kevin: match.cpp 22
* Kevin: parse-pretty.c 23
* Kevin: sqrtmod.c 24

Section C++ STL

* File/Stream I/O <stdio.h>

```
int  fgetc(FILE *stream);
char *fgets(char *s,int n,FILE *stream); – get up to n chars until EOL/EOF
int  getc(FILE *stream);
char *gets(char *s);
int  ungetc(int c,FILE *stream); push char c back onto front of stream
int  feof(FILE *stream); - returns true if EOF, else 0
```

* Printf

```
printf can be used with cout, requires <cstdio>
printf("%o",255); displays the octal version of integer 255 - 377
%d=decimal, %x or %X=hex, %f=float, %u=unsigned int, %ld=long int, %c=char,
%e or %E is exponential format for floats
%g or %G is the same except, it wont display exp format where the exp is 0
%s is c-style char string
%+d or %+f will force + or minus signs to be printed for all nums (+0 is a problem)
%5d or %5f will force 5 minimum chars for the output, prepending w/ spaces if needed
%05d or %-5f will do the same as above except it will zero pad.
%-5d or %-5f will force 5 minimum chars for the output, appending w/ spaces if needed
%-8.1f works too - left justify.
%8.1f will max the number of digits before the decimal at 8 and after at 1.
%8.1e will do the same w/ exponential values
```

* String

```
int .length() - returns string length
string .substr(int start, int size) - returns substring from start size length
string& .insert(int n, string s) - Inserts a copy of s into string starting at position n. The rest of
   the original string is shifted right.
string& .erase(int from, int to) - erases from middle - two positions are the ints (not length)
int .find(string ss, [int start]) - starting position of first occurence of ss
returns string::npos if no occurence
getline(istream is, string s) - places next line from is into s (>> only gets until next whitespace)
```

* Character Class Tests <ctype.h>

```
isalnum(c)  - isalpha(c) || isdigit(c) (number or letter)
isalpha(c)  - isupper(c) || islower(c) (any letter)
iscntrl(c)  - control character
isdigit(c)  - decimal digit            (0-9)
isgraph(c)  - printing character except space
islower(c)  - lower-case character     (a-z)
isprint(c)  - printing character including space
ispunct(c)  - printing character except space or letter or digit
isspace(c)  - space, formfeed, NL, CR, tab, vert tab
isupper(c)  - upper-case character     (A-Z)
isxdigit(c) – hexadeciaml digit        (0-9,a-f,A-F)
int tolower(int c); - convert c to lower case
int toupper(int c); - convert c to upper case
```

* Algorithm functions <algorithm>

```
bool includes(a,a+an,b,b+bn);- will return true if all the elements in b are in a

bool next_permutation(first pointer, last pointer); -will repeatedly re-arrange the values between(and
   including) the pointers creating new permutations where they are progressively greater in order
   until the order is greatest and it returns false. ie 123,132,213...321, then false.
   prev_permutation() works too

void replace(first, last, &oldvalue, &newvalue) - will replace all items in the list of oldvalue with
   newvalue

void reverse(first, last) - reverses order of elements
void sort(first, last) - sorts an array
void unique(first,last) - removes consecutively equal elements
```

* STL map/multimap

```
#include <map>
map<key,value> M;

iterators:
begin() - end()
rbegin() - rend()

//get
```

```
int size()
int count(K) - returns number of entries with key K
V M[K] - can be used to get or set V for K warning, if K is not in map, it will be added with value=V()
- not available on mutlimap
bool empty()
pair<iterator,iterator> equal_range(K) - set of elements whose key is K
iterator lower_bound(K) - first >= K
iterator upper_bound(K) - first > K

//add
map[K]=V - not avalible with multimap
insert(pair<k,v>) - m.insert(pair<int,string>(9,"blah 9"));

//remove
void clear() - clear all entries
int erase(K) - erase all elements with key K – returns number removed
```

* STL set/multiset

```
#include <set>
iterators:
begin() - end()
rbegin() - rend()

//get
int size()
int count(K) - returns number using key K
bool empty()
pair<iterator,iterator> equal_range(K) - set of elements whos key is K
iterator lower_bound(K) - first k >= K
iterator upper_bound(K) - first k > K

//add
insert(k) - insert k

//remove
void clear() - clear all entries
int erase(K) - erase all elements with key K
void erase(iterator) - erase at iterator
```

* STL hash_set or hash_map

```
hash_map<Key, Data, HashFcn, EqualKey, Alloc>
hash_set<Key, HashFcn, EqualKey, Alloc>
it is not necessary to specify Alloc

struct hashfcn
{
    int operator()(const State &S) const
    {
        return MaxW*S.Loc.first + S.Loc.second;
    }
};
struct eqs
{
    bool operator()(const State &s1, const State &s2) const
    {
        return (s1.Loc == s2.Loc);
    }
};

hash_set<State,hashfcn,eqs> StateHash;

//Then pretty much like normal set or map
```

* STL List

```
Good for queue or a stack
#include <list>

iterators:
begin() - end()
rbegin() - rend()

//get
size() - O(n)
bool empty()
T front()
T back()
```

```
//add
void push_front(T) - insert at beginning
void push_back(T) - insert at end

//ordering
void sort() - O(n log(n))
void reverse() - reverse order - O(1)

//removing
void pop_front() - remove front
void pop_back() - remove back
void clear() - erase all
```

* STL Vector/bit_vector

```
#include <vector>

iterators:
begin() - end()
rbegin() - rend()

//get
size() - O(n)
bool empty()
T front()
T back()
T vector[]

//add
void push_back(T) - insert at end

//removing
void pop_back() - remove back
void clear() - erase all

//sizing
void reserve(int Size) - makes size avalible in memory, but not in array
void resize(int Size, T initalValue) - extends array
```

* STL Notes

```
//operator needed for list.sort() or algorithm sort()
//or set or map sorting.  Where Object is the name of
//the type
bool operator< (Object const &c2) const
```

Section Java

* Java Start

```
class d
{
        public static void main(String[] args) throws Exception
        { new d(); }
        public d()
        {
        }
}
```

* Java java.io.StreamTokenizer

```
BufferedReader In=new BufferedReader(new InputStreamReader(System.in));
BufferedReader In=new BufferedReader(new FileReader("bulbs.in"));
StreamTokenizer STOK=new StreamTokenizer(In);
STOK.resetSyntax();
STOK.wordChars(0,65535);
STOK.whitespaceChars(' ',' ');
STOK.whitespaceChars('\n','\n');
STOK.whitespaceChars('\r','\r');
STOK.whitespaceChars('\t','\t');
//STOK.eolIsSignificant(true);
//STOK.quoteChar('"');

STOK.nextToken();

while (STOK.ttype!=STOK.TT_EOF)
{
    String S=STOK.sval;
    STOK.nextToken();
}
```

* Java java.math.BigInteger

```
BigInteger(String val, int radix)
BigInteger(String val)
toString(int radix)

Math:
BigInteger add(BigInteger val)
BigInteger subtract(BigInteger val)
BigInteger divide(BigInteger val)
BigInteger multiply(BigInteger val)
BigInteger mod(BigInteger val)
BigInteger pow(int exponent)
BigInteger abs() - abs

Bits:
BigInteger and(BigInteger val) - bitwise and
BigInteger or(BigInteger val) - bitwise or
BigInteger xor(BigInteger val) - bitwise xor
BigInteger not() - bitwise not
int bitLength() - Returns the number of bits excluding a sign bit.
BigInteger clearBit(int n)
boolean testBit(int n)
BigInteger setBit(int n)
BigInteger shiftLeft(int n)
BigInteger shiftRight(int n)
```

* Java java.util.LinkedList

```
void add(Object O) or addLast(Object O)
void addFirst(Object O)
Object get(int index)
Object getFirst()
Object getLast()
remove(int index)
remove(Object O)
Object removeFirst()
Object removeLast()
int size()
```

* Java java.util.TreeSet

```
add(Object O)
Object first()
Object last()
remove(Object O)
boolean contains(Object O)
```

* Java java.util.TreeMap

```
put(Object Key, Object Val)
remove(Object Key)
Object firstKey()
Set<Map.Entry> entrySet()
boolean containsKey(Object Key)
```

* Java java.util.Map.Entry

```
Object getKey()
Object getValue()
```

* Java java.util.ArrayList

```
add(Object O)
Object get(int idx)
remove(int idx)
```

* Java java.util.Iterator

```
TS=new TreeSet();
for(Iterator I=TS.iterator(); I.hasNext; )
{
    State S=(State)I.next();
}
TM=new TreeMap();
for(Iterator I=TS.entrySet().iterator(); I.hasNext; )
{
    Map.Entry E=(Map.Entry)I.next();
    Object key=E.getKey();
    Object val=E.getValue();
}
```

* Java java.text.DecimalFormat

```
DecimalFormat(String pattern)
//0 - Digit, show 0 on absent
//# - Digit
//. - decimal place
//, - inserts separator
```

```
String format(double d) or String format(int d)
//DecimalFormat DF=new DecimalFormat(",##0.000");
//produces: "900.000" "0.500" and "1,204.333" and "90,000,000.801"
//Does rounding automatically
```

Section Graph Theory

* Prim's Minimal Spanning Tree

```
Q = sorted queue
Start with single node
Add all possible links to Q sorted by least length first
Connected=1

While (Connected < Total)
  Add shortest link
  Add possible lines from new connected node to Q
```

* Floyd’s with path finding

```
//Conn[][] is a 2d array of distances/costs
//Next[][] is a 2d array of next hops - must be initialized such
//that if there is a direct path from A->B Next[A][B]=B

for (int k=0; k<Size; k++)
{
    for (int i=0; i<Size; i++)
    {
        for (int j=0; j<Size; j++)
        {
            if (Conn[i][k]+Conn[k][j] < Conn[i][j])
            {
                Conn[i][j]=Conn[i][k]+Conn[k][j];
                Next[i][j]=Next[i][k];
            }
        }
    }
}

//Best path from i to j is
//Loc=Current Location
Loc=i
while (Loc != j)
{Loc=Next[Loc][j];}
```

* MaxFlow

```
struct Node
{
    char ChgType; //A=add, D=delete, S=src
    int ChgAmt;
    unsigned int ChgSrc;
    map<int,int> OutFlows; //Target -> CurrFlow
};
vector<Node> Nodes;
map<int,map<int,int> > Paths; //Src->Target->MaxFlow
int Src;
int Snk;
int SrcMax=2147483647;
void clearChg()
{
    for(unsigned int i=0; i<Nodes.size(); i++)
    {Nodes[i].ChgAmt=0;}
}
int MaxFlow()
{
    clearChg();
    Nodes[Src].ChgType='S';
    Nodes[Src].ChgAmt=SrcMax;
    int ToSink=0;
    set<int> UpNodes;
    UpNodes.insert(Src);
    while(!UpNodes.empty())
    {
        //cout << "za" << endl;
        int N=*(UpNodes.begin()); UpNodes.erase(N);
        if (N==Snk)
        {
            int Amt=Nodes[Snk].ChgAmt;
            ToSink+=Amt;
            int Loc=Snk;
            while(Loc!=Src)
            {
                if (Nodes[Loc].ChgType=='A')
                {
                    Nodes[Nodes[Loc].ChgSrc].OutFlows[Loc]+=Amt;
                }
                if (Nodes[Loc].ChgType=='D')
                {
                    Nodes[Loc].OutFlows[Nodes[Loc].ChgSrc]-=Amt;
                }
                Loc=Nodes[Loc].ChgSrc;
            }
            clearChg(); Nodes[Src].ChgAmt=SrcMax;
            UpNodes.clear(); UpNodes.insert(Src);
        }
        else
        {
            for(map<int,int>::iterator I=Paths[N].begin(); I!=Paths[N].end(); I++)
            {
                int Tar=I->first; int Max=I->second; int Flow=Nodes[N].OutFlows[Tar];
                if ((Nodes[Tar].ChgAmt==0) && (Flow<Max))
                {
                    Nodes[Tar].ChgType='A';
                    Nodes[Tar].ChgAmt=(Max-Nodes[N].OutFlows[Tar]) <? Nodes[N].ChgAmt;
                    Nodes[Tar].ChgSrc=N;
                    UpNodes.insert(Tar);
                }
            }
            for(unsigned int Tar=0; Tar<Nodes.size(); Tar++)
            {
                int Flow=Nodes[Tar].OutFlows[N];
                if ((Flow>0) && (Nodes[N].ChgSrc!=Tar) && (Nodes[Tar].ChgAmt==0))
                {
                    Nodes[Tar].ChgType='D';
                    Nodes[Tar].ChgAmt=Flow <? Nodes[N].ChgAmt;
                    Nodes[Tar].ChgSrc=N;
                    UpNodes.insert(Tar);
                }
            }
        }
    }
    return ToSink;
}
Nodes.clear();
Paths.clear();
for(int i=0; i<PathCount; i++)
 {Paths[from][to]=amt;}
```

* Min distance

```
//find minimum distance between points on a graph
//function assumes array is a representation of a graph (directed or not) including weights
//function changes graph to store the minimum distances between any two points.  0's
//are assummed to mean not connected in both input and output.
//takes a pointer an square array of integers (ie: NxN size) and v, the 'width' of the array
//example:  mindist(&graph[0][0], 5);
void mindist(int *g, int v) {
    int a,b,c,d;
    for(a=0;a<v;a++)
        for(b=0+(a==0);b<v;b=b+1+(a==(b+1)))    {
            d=g[v*a+b];
            for(c=0+(a==0);c<v && d;c=c+1+(a==(c+1)))   {
                if(g[v*b+c])
                    if((d+g[v*b+c])<g[v*a+c] || g[v*a+c]==0)
                        g[v*a+c]=d+g[v*b+c];
}}}
```

* Greg: graph.cpp

```
//square graph variable -1 = no path >=0 = weighted path
vector<vector<int> > g;
//temporary variable used by some recursive functions for processing
//assummed empty

//ispath - returns 1 if there is a path between v and w
//MUST clear visited before re-use
vector<int> visited;
int isPath(int v,int w)
{
    int i;
    if(visited.empty())
        for(i=0;i<g.size();i++)
```

```
            visited.push_back(0);

    if(v==w) return 1;
    visited[v]=1;
    for(i=0;i<g.size();i++)
        if(g[v][i]!=-1)
            if(visited[i]==0)
                if(isPath(i,w)) return 1;

    return 0;
}

//Hpath – tests for hamiltonian path between v and w
//hamiltonian path is a path that goes through all vertices
//MUST clear *visited* before re-use
//vector<int> visited;  <-uncomment line
int HPath(int v,int w, int d=(g.size()-1))
{
    //cout<<"*"<<d<<"*";
    int i;
    if(visited.empty())
        for(i=0;i<g.size();i++)
            visited.push_back(0);

        if(v==w) {return (d==0); }//if(d==0) return 1; else return 0; }
    visited[v]=1;
    for(i=0;i<g.size();i++)
        if(g[v][i]!=-1)
            if(visited[i]==0)
                if(HPath(i,w,d-1)) return 1;

    visited[v]=0;
    return 0;
}

//degree - returns number of edges from vertex
//does not count self-loops
int degree(int v)
{
    int t=0;
    for(int i=0;i<g.size();i++)
        if(i!=v)
            t+=(g[v][i]!=-1);

    return t;
}

//Epath - returns 1 if euler path between v and w
//euler path uses all vertices exactly once
//assumes a fully connected graph
int EPath(int v,int w)
{
    int i;
    i=degree(v)+degree(w);
    if(i%2) return 0;
    for(i=0;i<g.size();i++)
        if((i!=v) && (i!=w))
            if(degree(i)%2) return 0;
    return 1;
}

//Connected - returns a list of lists
//each list represents a set of connected vertices
//parameter should be an empty vector<vector<int> >
//uses isPath()
//if(connected().size()==1) ~= if(graph is completely connected)
vector<vector<int> > connected()
{
    vector<vector<int> > out;
    for(int i=0;i<g.size();i++)
    {
        int flag=1;
        for(int j=0;j<out.size();j++)
        {
            visited.clear();
            if(isPath(i,out[j][0]))
            {
                //cout<<i;
                out[j].push_back(i);
                flag=0;
                break;
            }
        }
        if(flag)
        {
            vector<int> t;
            t.push_back(i);
            out.push_back(t);
        }
    }
    return out;
}

//bipartite - returns 1 if the graph is bipartite
//ie: it can be colored with only two colors, where no 2 nodes
//that are adjacent have the same color
//two functions, first is helper
vector<int> color;
int bphelp(int v, int c)
{
    color[v]=1-c;
    for(int i=0;i<g.size();i++)
    {
        if(g[v][i]==-1 || v==i)
            continue;
        if(color[i]==-1)
        {   if(!bphelp(i,1-c)) return 0;    }
        else if(color[i]!=c) return 0;
    }
    return 1;
}
int bipartite()
{
    int v, id=0;
    for(v=0;v<g.size();v++)
        color.push_back(-1);
    for(v=0;v<g.size();v++)
        if(color[v]==-1)
            if(!bphelp(v,0)) return 0;
    return 1;
}

//cuts – returns vector of single articulation or cut vertices
//returns all single articulation or cut vertices
//cut vertices, if removed, would make graph non-connected
//assumption: graph is connected and not digraph
//requires connected() and degree()
vector<int> cuts()
{
    vector<int> out;
    int j;
    for(int i=0;i<g.size();i++)
    {
        if(degree(i)<=1)    continue;
        vector<int> t=g[i];
        for(j=0;j<g.size();j++)
            g[i][j]=g[j][i]=-1;
        if(connected().size()>2)
            out.push_back(i);
        for(j=0;j<g.size();j++)
            g[i][j]=g[j][i]=t[j];
    }
    return out;
}

//bridge – returns a vector of bridges
//returns a vector of edges which are bridges in the graph
//a bridge is an edge which if removed would make the graph non-connected
//requires edge struct (w/o operator) and cuts()
struct edge {
    int v,w;
    operator<(edge &tc) { return (g[v][w]<g[tc.v][tc.w]); };
};
vector<edge> bridges()
{
    vector<int> in=cuts();
    vector<edge> out;
    int i;
    for(i=0;i<g.size();i++)
        if(degree(i)==1)
```

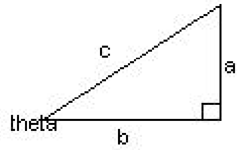
```
            in.push_back(i);
    for(i=0;i<in.size();i++)
        for(int j=i+1;j<in.size();j++)
        {
            edge t;
            t.v=in[i];
            t.w=in[j];
            if(g[in[i]][in[j]]!=-1)
                out.push_back(t);
        }
    return out;
}

//prim's algorithm for finding minimum spanning tree
//preferred over kruskal's algorithm for dense graphs
//requires edge struct w/ comparison function
//requires degree2 function
//assumes graph is fully connected
/*struct edge {
    int v,w;
    operator<(edge &tc) { return (graph[v][w]<graph[tc.v][tc.w]);
};*/
vector<vector<int> > prim()
{
    vector<vector<int> > out;
    vector<edge> e;
    vector<int> z;
    int i;
    for(i=0;i<g.size();i++)
        z.push_back(-1);
    for(i=0;i<g.size();i++)
    {
        out.push_back(z);
        out[i][i]=0;
    }
    z.clear();
    z.push_back(0);
    while(z.size()!=g.size())
    {
        for(i=0;i<g.size();i++)
            if(g[z[z.size()-1]][i]>0)
            {
                edge t;
                t.v=z[z.size()-1];
                t.w=i;
                e.push_back(t);
            }
        sort(e.begin(),e.end());
        for(;;)
        {
            if(degree2(e[0].w,out)<1)
            {
                z.push_back(e[0].w);
                int d=g[e[0].v][e[0].w];
                out[e[0].v][e[0].w]=d;
                out[e[0].w][e[0].v]=d;
                break;
            }
            e.erase(e.begin());
        }
    }
    return out;
}

//shortest distance - all pairs
//makes changes to global variable g
//g[i][j] on output will store the distance from i to j
//g[i][j] == INT_MIN if no path
//returned variable r[i][j] is the last vertex on path from i to j
//r[i][j] = junk if no path exists
// lines with /// after them can be removed if you dont need to know the path
vector<vector<int> > floyds()
{
    vector<vector<int> > p; ///
    int i,j,k;
    for(k=0;k<g.size();k++)     ///
    {                           ///
        vector<int> t;          ///
        for(i=0;i<g.size();i++) ///
            t.push_back(k);     ///
        p.push_back(t);     ///
    }                           ///

    for(k=0;k<g.size();k++)
        for(i=0;i<g.size();i++)
            if(g[i][k]!=-1)
                for(j=0;j<g.size();j++)
                    if(g[k][j]!=-1 && (g[i][j]==-1 || g[i][k] + g[k][j] < g[i][j]))
                    {
                        g[i][j]=g[i][k]+g[k][j];
                        p[i][j]=p[k][j];    ///
                    }
    return p;   ///
}

//shortest distance - single source
//returned variable is distance between v and w
//path[i] is a variable for the previous vertex in path from v to i
vector<int> path;
vector<int> dijkstra(int v)
{
    vector<int> d,s;
    int i,j,w;
    for(i=0; i<g.size();i++)
    {
        d.push_back(g[v][i]);
        s.push_back(0);
    }
    for(i=0;i<g.size()-1;i++)
    {
        w=-1;
        for(j=0;j<g.size();j++)
            if(!s[j] && d[j]!=-1&& (w==-1||d[j]<d[w]))
                w=j;
        s[w]=1;
        for(j=0;j<g.size();j++)
            if(!s[j] && d[w]+g[w][j]<d[j] ||  d[j]==-1)
                d[j]=d[w]+g[w][j];
    }
    for(j=0;j<d.size();j++)
    {
        cout<<d[j]<<"\t";
    }
    return d;
}
```

* Greg: next_combination.cpp

```
template <typename src, typename trg> bool next_combination(src sb,src se,trg tb,trg te)
{
  if(tb==te) return false;
  trg tq=tb+1;
  while(*sb++!=*tb);
  if(next_combination(sb,se,tq,te)) return true;
  for(;;)
  {
    if (sb==se) return false; // overflow
    *tb++=*sb++;
    if (tb==te) return true;
  }
}
int main()
{
  int arr[5]={1, 2, 3, 4,5};
  vector<int> v(arr,arr+5);
  vector<int> w(v.begin(),v.begin()+3);
  copy(w.begin(),w.end(),ostream_iterator<int>(cout," "));
  cout<<"\n";
  while(next_combination(v.begin(),v.end(),w.begin(),w.end()))
  {
     copy(w.begin(),w.end(),ostream_iterator<int>(cout," "));
     cout<<"\n";
  }
}
/*
Output is
{1 2 3}{1 2 4}{1 2 5}{1 3 4}{1 3 5}{1 4 5}{2 3 4}{2 3 5}{2 4 5}{3 4 5}
program calculates all combinations of 3 numbers within a set of 5
*/
```

Section Math

* Basic Trig



$c^2 = a^2 + b^2$

sin(theta)=a/c

cos(theta)=b/c

tan(theta)=a/b

Law of cosines: $a^2 = b^2 + c^2 - 2bc \cos A$

* include <cmath>

double hypot(double x, double y); returns (x^2+y^2)^(1/2)

double rint(double x); rounds to nearest int

double exp(double x); returns the value of e^x

double log(double x); returns natural log of x

double log10(double x); returns $\log_{10}$ of x

double pow(double x, double y); returns x^y

double sqrt(double x); returns square root of x

double cbrt(double x); returns cube root of x

double atan2(double y, double x); returns angle on unit circle in the range of pi to –pi in the direction of line with rise y and run x

* Min distance from point to line(2points)

/*Equation for distance from pt to line*/

$$\mathbf{d}(\mathbf{P},\mathbf{L}) = \frac{(y_0 - y_1)x + (x_1 - x_0)y + (x_0 y_1 - x_1 y_0)}{\sqrt{(x_1 - x_0)^2 + (y_1 - y_0)^2}}$$

* Min distance from point to line(point slope)

```
//returns minimum distance between point and line
double distpointline(double x, double y, double a, double b, double c) {
   return fabs(a*x + b*y - c)/sqrt(a*a + b*b);
}
```

* Area of a polygon

```
//find the area of a polygon outlined by the points stored in x and y
//requires that the points be in clockwise order around the figure
//if points are in counterclockwise order, the output will be the area
//negated (possibly useful - or just abs() the result)
//If the points are not in any order, the output is junk, but probably will be zero
//Note: triangles are always in one or the other order!
double polyarea(vector<double> x,vector<double> y)
{
    double a=0;
    int s=x.size();
    for(int i=0;i<s;i++)
        a+=( y[i]*x[(i+1)%s] - x[i]*y[(i+1)%s]);
    return a/2;
}
```

* Circle from three points on radius

```
//finds the center and radius of a circle defined by three points on
//its radius, returns true if result found, false otherwise
//requires the point structure
int circle(point p1, point p2, point p3, point *center, double *r)
{
  double a,b,c,d,e,f,g;

  a = p2.x - p1.x;
  b = p2.y - p1.y;
  c = p3.x - p1.x;
  d = p3.y - p1.y;
  e = a*(p1.x + p2.x) + b*(p1.y + p2.y);
  f = c*(p1.x + p3.x) + d*(p1.y + p3.y);
  g = 2.0*(a*(p3.y - p2.y) - b*(p3.x - p2.x));
  if (fabs(g) < 1e-8) {
    return 0;
  }
  center->x = (d*e - b*f) / g;
  center->y = (a*f - c*e) / g;
  *r = sqrt((p1.x-center->x)*(p1.x-center->x) +
       (p1.y-center->y)*(p1.y-center->y));
  return 1;
}
```

* Convert two points in parametric form of line

```
//converts two points to a parametric form of a line
void linepoints(double x1, double y1, double x2, double y2,
                double *a, double *b, double *c)
{
   *a = y2 - y1;
   *b = x1 - x2;
   *c = *a * x1 + *b * y1;
}
```

* Find line equidistant from two points

```
//given two points (x1,y1); (x2,y2) returns a,b,c for the line equidistant from both points
void bi(double x1, double y1, double x2, double y2, double *a,
        double *b, double *c)
{
   *a = 2*(x2-x1);
   *b = 2*(y2-y1);
   *c = x2*x2 + y2*y2 - x1*x1 - y1*y1;
}
```

* Line intersection

```
//given two lines in abc parametric form, returns the intersection point (x,y)
//if lines are parallel, function returns 0, else function returns 1
int isct(double a, double b, double c, double aa, double bb,
         double cc, double *x, double *y)
{
   double det = a*bb - b*aa;
   if (fabs(det) < 1e-10) return 0;
   *x = (-b*cc + c*bb)/det;
   *y = (a*cc - c*aa)/det;
   return 1;
}
```

* polyareaConvex

```
/* returns area of CONVEX polygon where points are already in sorted angular order */
//not a very flexible function, but short for special case problems
double polyareaConvex(vector<double> x,vector<double> y)
{
       double a=0;
       int s=x.size();
       for(int i=0;i<s;i++)
              a+=( y[i]*x[(i+1)%s] - x[i]*y[(i+1)%s]);
       return a/2;
}
/* choose: number of combinations of n items, taken m at a time (no overflow) */
/* Uses gcd() from above */
unsigned choose(int _n,int _m){
 unsigned ret;
 int      i;
 ret=1;
 for(i=1;i<=_m;_n--;i++){
  if(_n%i==0)ret*=_n/i;
  else if(ret%i==0)ret=(ret/i)*_n;
  else{
   int d;
   d=gcd(_n,i);
   ret=(ret/(i/d))*(_n/d);} }
 return ret;}
```

/* detAng: determine angle from ray p2->p1 to p2->p3

```
/* Uses struct Point and type coord from above */
# define _Sqr(_x) ((_x)*(_x))
```

```
double detAng(Point *_p1,Point *_p2,Point *_p3){
 double a2,b2,c2,a,b,cosc;
 a2=_Sqr(_p2->x-_p1->x)+_Sqr(_p2->y-_p1->y);
 b2=_Sqr(_p3->x-_p2->x)+_Sqr(_p3->y-_p2->y);
 c2=_Sqr(_p1->x-_p3->x)+_Sqr(_p1->y-_p3->y);
 a=sqrt(a2);
 b=sqrt(b2);
 cosc=(a2+b2-c2)/(2*a*b);
 if(cosc<-1)cosc=-1;
 else if(cosc>1)cosc=1;
 return acos(cosc);}
```

* convHull: generate convex hull

```
/* around pts, return num points in hull
/* hull requires npts+1 Points of storage */
/* pts and hull can be the same */
/* Uses struct Point and type coord from above */
/*Ken Clarkson wrote this.  Copyright (c) 1996 by AT&T. Permission to use, copy, modify, and distribute
this software for any purpose without fee is hereby granted, provided that this entire notice
  is included in all copies of any software which is or includes a copy or modification of this software
and in all copies of the supporting documentation for such software.
  THIS SOFTWARE IS BEING PROVIDED "AS IS", WITHOUT ANY EXPRESS OR IMPLIED WARRANTY.  IN PARTICULAR,
NEITHER THE AUTHORS NOR AT&T MAKE ANY REPRESENTATION OR WARRANTY OF ANY KIND CONCERNING THE
MERCHANTABILITY
  OF THIS SOFTWARE OR ITS FITNESS FOR ANY PARTICULAR PURPOSE.*/
int ccw(Point *_pts,int _i,int _j,int _k){
 return (_pts[_i].x-_pts[_j].x)*(_pts[_k].y-_pts[_j].y)-
        (_pts[_i].y-_pts[_j].y)*(_pts[_k].x-_pts[_j].x)<=0;}

int cmpl(const void *_a,const void *_b){
 Point *a=(Point *)_a;
 Point *b=(Point *)_b;
 coord  v;
 v=a->x-b->x;
 if(v<0)return -1;
 if(v>0)return 1;
 v=a->y-b->y;
 return v<0?-1:v>0;}

int cmph(const void *_a,const void *_b){
 return cmpl(_b,_a);}

int makeChain(Point *_pts,int _npts,int (*_cmp)(const void *,const void *)){
 int i;
 int j;
 int s;
 qsort(_pts,_npts,sizeof(Point),_cmp);
 s=1;
 for(i=2;i<_npts;i++){
  Point t;
  for(j=s;j>=1&&ccw(_pts,i,j,j-1);j--);
  s=j+1;
  t=_pts[s];
  _pts[s]=_pts[i];
  _pts[i]=t;}
 return s;}

int convHull(Point *_pts,int _npts,Point *_hull){
 int u;
 if(_hull!=_pts)memcpy(_hull,_pts,_npts*sizeof(Point));
 u=makeChain(_hull,_npts,cmpl);
 if(!_npts)return 0;
 _hull[_npts]=_hull[0];
 return u+makeChain(_hull+u,_npts-u+1,cmph);}
```

* polyArea2: returns twice the signed area of the polygon

```
/* Orientation does not matter. The polygon may be self-intersecting. */
/* Uses struct Point and type coord from above */

/* Returns twice the signed area of the given triangle */
/* The area is positive iff a,b,c are oriented ccw, and negative iff cw (0 => colinear) */
coord triArea2(Point *_a,Point *_b,Point *_c){
 return (_b->x-_a->x)*(_c->y-_a->y)-
        (_c->x-_a->x)*(_b->y-_a->y);}

coord polyArea2(Point *_poly,int _npts){
 int   i;
 coord ret;
 ret=0;
 for(i=1;i<_npts-1;i++)ret+=triArea2(_poly,_poly+i,_poly+i+1);
 return ret;}
```

* Point In Poly

```
/* returns 1 if pt is inside, -1 if pt is on the boundary, 0 if pt is outside
/* Orientation does not matter. The polygon may be self-intersecting (even-odd rule). */
/* Uses struct Point and type coord from above */
int pointInPoly(Point *_poly,int _npts,Point *_pt){
 int i,j;
 int hits=0;
 for(i=0,j=_npts-1;i<_npts;j=i++){
  coord dy=_poly[j].y-_poly[i].y;
  if(_poly[i].y==_pt->y){
   if(!dy&&((_poly[i].x<=_pt->x&&_pt->x<=_poly[j].x)||
          (_poly[j].x<=_pt->x&&_pt->x<=_poly[i].x)))return -1;}
  else if(dy){
   coord dx=_poly[j].x-_poly[i].x;
   coord rx=_pt->x-_poly[i].x;
   coord ry=_pt->y-_poly[i].y;
   if(dy<0){
    ry=-ry;
    dy=-dy;}
   if(ry<=0&&ry>=dy){
    coord xl=ry*dx;
    coord xr=rx*dy;
    if(xl==xr)return -1;
    if(xl>xr)hits++;} } }
 return hits&1;}
```

* LineIntersect

```
/* lineIntersect: computes the intersection of two parametric lines. The intersection
    point is defined as
   xi=x1+(s/d)*(x2-x1); or xi=x3+(t/d)*(x4-x3);
   yi=y1+(s/d)*(y2-y1); or yi=y3+(t/d)*(y4-y3);
   d is always non-negative. If s and t are between 0 and d, the intersection falls on
   the actual line segments */
/* Uses struct Point and type coord from above */
int lineIntersect(Point *_p1,Point *_p2,Point *_p3,Point *_p4,
                  coord *_s,coord *_t,coord *_d){
 coord a=_p2->x-_p1->x;
 coord b=_p3->x-_p4->x;
 coord c=_p2->y-_p1->y;
 coord d=_p3->y-_p4->y;
 coord e=_p3->x-_p1->x;
 coord f=_p3->y-_p1->y;
 *_d=a*d-b*c;
 *_s=d*e-b*f;
 *_t=a*f-c*e;
 if(!*_d){
  if(*_s)return DISJOINT;
  return COINCIDENT;}
 if(*_d<0){
  *_d=-*_d;
  *_s=-*_s;
  *_t=-*_t;}
 return INTERSECTS;}
```

* lineProj – Point onto line

```
/* lineProj: computes the projection of a point to a line segment.
   The perpendicular projection of (x3,y3) onto (x1,y1),(x2,y2) is
   xi=x1+(r/d)*(x2-x1);
   yi=y1+(r/d)*(y1-y1);
   The distance from (x1,y1) to (xi,yi) is |r|/sqrt(d)
   The distance from (x3,y3) to (xi,yi) (i.e. to the line) is |s|/sqrt(d)
   If r<0, (xi,yi) is on the backward extension of (x1,y1),(x2,y2)
   If r>d, (xi,yi) is on the forward extension of (x1,y1),(x2,y2)
   If 0<=r<=d, (xi,yi) is on the segment (x1,y1),(x2,y2)
   If s<0, (x3,y3) is to the left of the line (x1,y1),(x2,y2)
   If s>0, (x3,y3) is to the right of the line (x1,y1),(x2,y2)
   If s==0, (x3,y3) is on the line (x1,y1),(x2,y2) */
void lineProj(Point *_p1,Point *_p2,Point *_p3,
              coord *_r,coord *_s,coord *_d){
 *_d=(_p2->x-_p1->x)*(_p2->x-_p1->x)+(_p2->y-_p1->y)*(_p2->y-_p1->y);
 *_r=(_p3->x-_p1->x)*(_p2->x-_p1->x)+(_p3->y-_p1->y)*(_p2->y-_p1->y);
 *_s=(_p3->x-_p1->x)*(_p2->y-_p1->y)-(_p3->y-_p1->y)*(_p2->x-_p1->x);}
```

* Kevin: geom-pretty.cpp

```
#define TOL 1e-12
```

```
struct pt { double x, y; pt(double _x=0.0,double _y=0.0):x(_x),y(_y){} };
struct seg { pt p1, p2;
 seg( double x1, double y1, double x2, double y2 ):p1(x1,y1),p2(x2,y2){}
 seg( pt const &p, pt const &q ):p1(p),p2(q) {}
};

double det( double a, double b, double c, double d ) {
 return a*d-b*c;
}

double dist( pt const &a, pt const &b ) {
 return sqrt((a.x-b.x)*(a.x-b.x)+(a.y-b.y)*(a.y-b.y));
}

// intersection of lines
pt isect( seg s1, seg s2 ) {
 double m = det(
  s1.p2.x-s1.p1.x, s2.p1.x-s2.p2.x,
  s1.p2.y-s1.p1.y, s2.p1.y-s2.p2.y );
 // m==0 => parallel
 double m1 = det(
  s2.p1.x-s1.p1.x, s2.p1.x-s2.p2.x,
  s2.p1.y-s1.p1.y, s2.p1.y-s2.p2.y );
 double m2 = det(
  s1.p2.x-s1.p1.x, s2.p1.x-s1.p1.x,
  s1.p2.y-s1.p1.y, s2.p1.y-s1.p1.y );
 double t = m1/m, s = m2/m;
 // 0 < t < 1 => intersection lies on s1
 // 0 < s < 1 => intersection lies on s2
 return pt(
  s1.p1.x + t*(s1.p2.x-s1.p1.x),
  s1.p1.y + t*(s1.p2.y-s1.p1.y) );
}

// intersection of segments
bool isect2( seg s1, seg s2, bool endp = false, bool parallel = true ) {
 double zero = 0 + (endp?(-TOL):TOL);
 double one = 1 + (endp?(TOL):(-TOL));
 double m = det(
  s1.p2.x-s1.p1.x, s2.p1.x-s2.p2.x,
  s1.p2.y-s1.p1.y, s2.p1.y-s2.p2.y );
 // m==0 => parallel
 if( fabs(m) < TOL ) {
  if( parallel ) {
   double t0 = (s2.p1.x - s1.p1.x)/(s1.p2.x - s1.p1.x);
   double t1 = (s2.p2.x - s1.p1.x)/(s1.p2.x - s1.p1.x);
   if((zero < t0 && t0 < one) || (zero < t1 && t1 < one))
    return true;
  }
  return false;
 }
 double m1 = det(
  s2.p1.x-s1.p1.x, s2.p1.x-s2.p2.x,
  s2.p1.y-s1.p1.y, s2.p1.y-s2.p2.y );
 double m2 = det(
  s1.p2.x-s1.p1.x, s2.p1.x-s1.p1.x,
  s1.p2.y-s1.p1.y, s2.p1.y-s1.p1.y );
 double t = m1/m, s = m2/m;
 // 0 < t < 1 => intersection lies on s1
 // 0 < s < 1 => intersection lies on s2
 return ((zero < t && t < one) && (zero < s && s < one));
}

// tangent line to circle
// returns a seg whose endpoints are tangent points on the circle
seg circtan( pt p, pt c, double r ) {
 double leg = dist(p,c);
 double ll = sqrt(leg*leg-r*r);
 double theta = atan2(r,ll);
 double phi = atan2(p.y-c.y,p.x-c.y);
 pt p1(c.x + r*cos(phi+(M_PI/2-theta)), c.y + r*sin(phi+(M_PI/2-theta)));
 pt p2(c.x + r*cos(phi-(M_PI/2-theta)), c.y + r*sin(phi-(M_PI/2-theta)));

 return seg( p1, p2 );
}

// point in polygon, poly must be convex; not efficient
bool pt_in_poly( pt p, vector<pt> poly ) {
 vector<seg> sg1, sg2;
 int i,j,k,m,n=poly.size();
 for( i=0; i < n; ++i ) {
  sg1.push_back( seg(p,poly[i]) );
  sg2.push_back( seg(poly[i],poly[(i+1)%n]) );
 }
 for(i=0;i<sg1.size();++i)
  for(j=0;j<sg2.size();++j)
   if( isect2( sg1[i], sg2[j], false, true ) )
    return false;
 return true;
}

// point in circle
bool pt_in_circ( pt p, pt c, double r ) {
 return dist(p,c)<r;
}

// point of closest approach to line
pt closest_pt( pt p, seg s ) {
 double d = dist(s.p2,s.p1);
 double le = ((p.x-s.p1.x)*(s.p2.x-s.p1.x)+(p.y-s.p1.y)*(s.p2.y-s.p1.y))/d;
 return pt(s.p1.x+(le/d)*(s.p2.x-s.p1.x),s.p1.y+(le/d)*(s.p2.y-s.p1.y));
}

// distance of closest approach to line
double closest_dist( pt p, seg s ) {
 double d = dist(s.p2,s.p1);
 double le = ((p.x-s.p1.x)*(s.p2.x-s.p1.x)+(p.y-s.p1.y)*(s.p2.y-s.p1.y))/d;
 double h = dist(s.p1,p);
 return sqrt(h*h-le*le);
}

// given sides, return triangle
vector<pt> triangle( double a, double b, double c ) {
 vector<pt> r;
 r.push_back( pt(0,0) );
 r.push_back( pt(a,0) );
 double B = acos((a*a+c*c-b*b)/(2*a*c));
 r.push_back( pt(c*cos(B),c*sin(B)) );
 return r;
}

// find the incenter of a triangle
pt incenter( vector<pt> p ) {
 double a = dist(p[0],p[1]),
 b = dist(p[1],p[2]),
 c = dist(p[2],p[0]);
 double lo = a; lo >?=b; lo >?= c;
 double B = acos((a*a+c*c-b*b)/(2*a*c));
 double phi = atan2(p[1].y-p[0].y,p[1].x-p[0].x);
 if( phi < 0 ) phi = 2*M_PI+phi;
 seg s1( p[0], pt(p[0].x + lo*cos(phi+B/2),p[0].y + lo*sin(phi+B/2)) );
 double C = acos((a*a+b*b-c*c)/(2*a*b));
 phi = atan2(p[2].y-p[1].y,p[2].x-p[1].x);
 if( phi < 0 ) phi = 2*M_PI+phi;
 seg s2( p[1], pt(p[1].x + lo*cos(phi+C/2),p[1].y + lo*sin(phi+C/2)) );
 return isect(s1,s2);
}

// convex hull, not very efficient
bool hull_lt( pt const &p1, pt const &p2 ) {
 if( p1.y < p2.y ) return true;
 if( p2.y < p1.y ) return false;
 if( p1.x < p2.x ) return true;
 if( p2.x < p1.x ) return false;
 return false;
}

vector<pt> hull( vector<pt> p, bool far = true ) {
 int i,j,k,m,n;
 vector<pt> r;
 vector<bool> skip( p.size(), false );
 sort( p.begin(), p.end(), hull_lt );
 r.push_back( p[0] );
 double maxt = -1;
 double maxd = -1;
 int cur = 0;
 for( i=1; i < p.size(); ++i ) {
  double th = atan2( p[i].y-p[0].y, p[i].x-p[0].x );
  double cd = dist(p[i],p[0]);
  if( th > maxt + TOL ) {
   maxt = th;
   maxd = cd;
   cur = i;
  } else if( fabs(maxt-th)<TOL ) {
   if( far && cd > maxd ) {
    maxt = th; maxd = cd; cur = i;
   } else if( !far && cd < maxd ) {
    maxt = th; maxd = cd; cur = i;
   }
  }
 }
 while( cur != 0 ) {
  skip[cur] = true;
  pt prev = r[r.size()-1];
  double maxt = -1;
  double maxd = -1;
  int next = 0;
  for( i=0; i<p.size(); ++i ) {
   if( skip[i] ) continue;
   double d = dist(p[i],p[cur]);
   double x =
    ((p[i].x-p[cur].x)*(prev.x-p[cur].x)+
    (p[i].y-p[cur].y)*(prev.y-p[cur].y))/dist(prev,p[cur]);
   double th = atan2( sqrt(d*d-x*x), x );
   if( th > maxt + TOL ) {
    maxt = th;
    maxd = d;
    next = i;
   } else if( fabs(maxt-th)<TOL ) {
    if( far && d > maxd ) {
     maxt = th; maxd = d; next = i;
    } else if( !far && d < maxd ) {
     maxt = th; maxd = d; next = i;
    }
   }
  }
  r.push_back( p[cur] );
  cur = next;
 }
 return r;
}

// area of a triangle
double tri_area( vector<pt> p ) {
 double a = dist(p[0],p[1]);
 double b = dist(p[1],p[2]);
 double c = dist(p[2],p[0]);
 double s = (a+b+c)/2;
 return sqrt(s*(s-a)*(s-b)*(s-c));
}

// split a polygon into triangles
vector< vector<pt> > tri_split( vector<pt> p ) {
 vector< vector<pt> > r;
 int i,j,k,m,n=p.size();
 if( n<=3 ) { r.push_back(p); return r; }
 vector<pt> cur(3);
 for( i=1; i<=n-2; ++i ) {
  cur[0] = p[0]; cur[1] = p[i]; cur[2] = p[i+1];
  r.push_back(cur);
 }
 return r;
}

// find the area of a polygon (clkwise)
double poly_area( vector<pt> p ) {
 double r = 0;
 vector< vector<pt> > t = tri_split( p );
 for(int i=0; i<t.size(); ++i )
  r += tri_area(t[i]);
 return r;
}

// intersection of polygons; not efficient
vector<pt> isect_poly( vector<pt> p1, vector<pt> p2 ) {
 vector<pt> p;
 int i,j;
 for( i=0; i < p1.size(); ++i )
  if( pt_in_poly(p1[i],p2) )
```

```
   p.push_back(p1[i]);
 for( i=0; i < p2.size(); ++i )
  if( pt_in_poly(p2[i],p1) )
   p.push_back(p2[i]);
 for( i=0; i < p1.size(); ++i )
  for( j=0; j < p2.size(); ++j ) {
   seg s1( p1[i], p1[(i+1)%p1.size()] );
   seg s2( p2[j], p2[(j+1)%p2.size()] );
   if( isect2( s1, s2, false, false ) )
    p.push_back( isect(s1,s2) );
  }
 return hull(p);
}
```

Section Other Algorithms

* Choose

```
//Returns number of combinations of M elements
//being selected in space N
//when order does of the elements does not matter
map<pair<int,int>,long long> ChooseCache;
long long choose(int N, int M)
{
    if (M==0) return 1;
    if (M==N) return 1;
    if (ChooseCache.count(pair<int,int>(N,M))>0)
    { return ChooseCache[pair<int,int>(N,M)];}
    long long V=choose(N-1,M)+choose(N-1,M-1);
    ChooseCache[pair<int,int>(N,M)]=V;
    return V;
}
//ways of putting N balls into K bins
    choose(n+k-1,n);
```

* Partitions

```
//ways to split n into k groups
map<pair<int,int>,long long> PartCache;
long long part(int n, int k)
{
    if (n==k) return 1;
    if (k==1) return 1;
    if (PartCache.count(pair<int,int>(n,k))>0)
    { return PartCache[pair<int,int>(n,k)];}
    long long V=part(k-n,n)+part(n-1,k-1);
    PartCache[pair<int,int>(n,k)]=V;
    return V;
}
```

* Permutations

```
//ways to create words (ordered lists) of length r from n elements
long long perm(int n, int r)
{
    long long S=1;
    for (int i=n; i>(n-r); i--)
    { S=S*i; }
    return S;
}
//permutations of N with k1, k2 and k3 identical components
perm(N,N)/perm(k1,k1)/perm(k2,k2)/perm(k3,k3)
//example mississippi
//N=11
//k1=4 (i)
//k2=4 (s)
//k3=2 (p)
```

* A-star search (or UCS or BFS)

```
struct State
{
    double f; //f = cost + heur
    double heur;
    double cost;
    //Create next states and add them to N
    //Entries in N must have valid f
    void setNext(multimap<double,State> &Q){………}
    bool isGoal(){………}
    //Used to avoid duplicates
    bool operator< (State const &c2) const

};

bool Astar(State First, State &Sol)
{
    multimap<double,State> Q;
    set<State> ExpandedStates; //avoid dups
    Q.insert(pair<double,State>(First.f,First));
    while (!Q.empty())
    {
        State S=Q.begin()->second;
        Q.erase(Q.begin());

        if (ExpandedStates.count(S)==0) //avoid dups
        {
            ExpandedStates.insert(S);

            if (S.isGoal())
            {
                Sol=S;
                return true;
            }
            S.addNext(N);
        }
    }
    return false;
}
```

* Rotate matrix

```
// Rotate a rectangular matrix right 90
vector<vector<int> > rot(vector<vector<int> > &V)
{
    vector<vector<int> > R;
    R.resize(V[0].size());
    for(int i=0; i<V[0].size(); i++)
    {
        R[i].resize(V.size(),-1);
    }
    for(int i=0; i<V.size(); i++)
    for(int j=0; j<V[i].size(); j++)
    {
        int J=V.size()-1-i;
        int I=j;
        R[I][J]=V[i][j];
    }
    return R;
}
```

* GCD and LCM

```
//gets gcd of 2 numbers
//works for any order
//lcm is a*b/gcd(a,b)
int gcd(int a, int b)
{
    return (b==0)?a:gcd(b, a % b);
}
```

* Day Of Week

```
/* dayOfWeek: return day of week given month, day, and year (>1752) */
int dayOfWeek(int _year,int _month,int _day){                       /*0=Sunday*/
 int offset[]={0,3,2,5,0,3,5,1,4,6,2,4};
 _year-=_month<3;
 return (_year+_year/4-_year/100+_year/400+offset[_month-1]+_day)%7;}

/* These typedefs are used by most of the functions below */
typedef double coord;    /*This can be any signed numeric type (including int)*/
typedef struct Point Point;
struct Point{
 coord x;
 coord y;};
```

* String search-replace

```
//replaces all instances of search with replace in toSearch
string str_replace(string toSearch, string search, string replace)
{
    int g=toSearch.find(search);
    while(g!=string::npos)
    {
        toSearch.erase(g, g+search.length()-1);
        toSearch.insert(g, replace);
        g=toSearch.find(search, g+replace.length());
    }
    return toSearch;
}
```

* Prime Factor

```
//returns the prime factors of n in ascending order
vector<int> primefactor(int n)
{
    vector<int> out;
    for(int i=2;i<=n; i++)
        while(n%i==0)
        {
            n/=i;
            out.push_back(i);
        }
    return out;
}
```

* Generate Combinations

```
//generates all possible combinations/subsets of any length
//of n elements in the array elem
//output is currently done on a number by number basis
//where each line is a new combination
//can only handle numbers, includes null set
// [2,1] is assummed to be equivalent to [1,2] and only one is included
void combinations()
{
    int n=3,*elem,j;
    unsigned int i;
    elem = new int[n];
    //fills elem
    for(i=0;i<n;i++)
        elem[i]=i+1;
    for(i=0; i<(1<<n); i++) //change to i=1 to skip null set
    {
        for (j=0;j<n; j++) if (i & (1<<j)) cout<<elem[j]<<" ";
        cout<<"\n";
    }
}
```

* Kevin: match.cpp

```
#define SHARP -99
using namespace std;
void debug( vector< vector<int> > adj, int ROW, int COL, vector<int> row_lbl, vector<int> col_lbl ) {
    for( int j = 0; j < ROW; ++j ) {
        if( row_lbl[j] >= 0 ) cout << row_lbl[j];
        else if( row_lbl[j]==SHARP ) cout << "#";
        else cout << " ";
        cout << " ";
        for( int i = 0; i < COL; ++i )
        switch( adj[j][i] ) {
            case 0: cout << '0'; break;
            case 1: cout << '1'; break;
            case -1: cout << '*'; break;
        }
        cout << endl;
    }
    cout << "  ";
    for( int i=0; i < COL; ++i ) {
        if( col_lbl[i] >= 0 ) cout << col_lbl[i];
        else if( col_lbl[i]==SHARP ) cout << '#';
        else cout << " ";
    }
    cout << endl;
    cout << endl;
}
vector<string> match( vector<string> a ) {
    int ROW = a.size(), COL = a[0].length();
    vector< vector<int> > adj( ROW, vector<int>( COL, 0 ) );
    bool one_found = false;
    for( int j = 0; j < ROW; ++j )
        for( int i = 0; i < COL; ++i ) {
        if( a[j][i]=='1' && !one_found )
            adj[j][i] = -1, one_found = true;
        else adj[j][i] = a[j][i]-'0';
        }
    while( 1 ) {
        vector<int> row_lbl( ROW, -1 );
        vector<int> col_lbl( COL, -1 );
        vector<bool> row_scanned( ROW, false );
        vector<bool> col_scanned( COL, false );

        bool very_done = false;
        for( int i = 0; i < COL; ++i ) {
        int nstar = 0;
        for( int j = 0; j < ROW; ++j )
            if( adj[j][i]==-1 )
                ++nstar;
        if( nstar==0 )
            col_lbl[i] = SHARP;
        }
        bool done = false;
        int last = -1;
        while( !done ) {
        for( int i = 0; i < COL; ++i )
            if( col_lbl[i] != -1 && !col_scanned[i] ) {
                for( int j = 0; j < ROW; ++j )
                if( adj[j][i]==1 && row_lbl[j]==-1 )
                    row_lbl[j] = i;
                col_scanned[i] = true;
            }
        for( int j = 0; j < ROW && !done; ++j )
            if( row_lbl[j]!=-1 && !row_scanned[j] ) {
                bool found = false;
                for( int i = 0; i < COL && !found; ++i )
                if( adj[j][i]==-1 )
                    found = true, col_lbl[i]=j;
                if( !found ) {
                done = true;
                last = j;
                } else row_scanned[j] = true;
            }
        int lbl_noscan = 0;
        for( int i = 0; i < COL; ++i )
            if( col_lbl[i] != -1 && !col_scanned[i] ) ++lbl_noscan;
        for( int j = 0; j < ROW; ++j )
            if( row_lbl[j] != -1 && !row_scanned[j] ) ++lbl_noscan;
        if( lbl_noscan==0 ) {
            very_done = true;
            break;
        }
        }
        if( very_done ) break;

        while( true ) {
        adj[last][row_lbl[last]] *= -1;
        if( col_lbl[row_lbl[last]]==SHARP ) break;
        int c = row_lbl[last];
        adj[col_lbl[c]][c] *= -1;
        last = col_lbl[c];
        }
    }
    vector<string> r;
    for( int j = 0; j < ROW; ++j ) {
        string t;
        for( int i = 0; i < COL; ++i )
        t.push_back( (adj[j][i]==-1)?char('1'):char('0') );
        r.push_back( t );
    }
    return r;
}
```

* Kevin: parse-pretty.c

```
#include <stdlib.h>
#include <string.h>
#include <math.h>

#define STR 256
#define NOP 3
#define NB 2

char *op[NOP] = { "+-", "*/", "^" };
char *br[NB] = { "()", "[]" };

char isl( char c ) {
 int i;
 for(i=0;i<NB;++i) if(br[i][0]==c) return br[i][1];
 return 0;
}

char isr( char c ) {
 int i;
 for(i=0;i<NB;++i) if(br[i][1]==c) return br[i][0];
 return 0;
}

struct pn {
 char node[STR];
 char op;
 struct pn *left, *right;
};

struct pn *mknode( char op, struct pn *l, struct pn *r ) {
 struct pn *p = malloc(sizeof(struct pn));
 assert(p!=NULL);
 p->op = op;
 p->left = l;
 p->right = r;
 strcpy(p->node,"");
 return p;
}

struct pn *mkleaf( char *expr ) {
 struct pn *p = malloc(sizeof(struct pn));
 assert(p!=NULL);
 strncpy(p->node,expr,STR);
 p->left = p->right = 0;
 p->op = 0;
 return p;
}

struct pn *parse( char *s ) {
 int i,j,k,b,c;
 char q;
 for(i=0;i<NOP;++i)
  for(j=strlen(s);j>=0;--j)
   if(isr(s[j])){
    b=j;c=1;
    while(c) if(isr(s[--j]))++c; else if(isl(s[j]))--c;
    if(b==strlen(s)-1&&j==0){
     s[strlen(s)-1] = 0;
     return parse(s+1);
    }
   } else {
    for(k=0;k<strlen(op[i]);++k)
     if(s[j]==op[i][k]){
      q = s[j];
      s[j] = 0;
      return mknode( q, parse(s), parse(s+j+1) );
     }
   }
 return mkleaf(s);
}

void showtree( struct pn *p ) {
 if( p->op ) {
  if( p->left ) { showtree(p->left); printf(" "); }
  if( p->right ) { showtree(p->right); printf(" "); }
  printf( "%c ", p->op );
 } else {
  printf( "%s ", p->node );
 }
}

int evaltree( struct pn *p ) {
 int a,b;
 if( p->op ) {
  a = evaltree( p->left );
  b = evaltree( p->right );
  if( p->op=='+' ) return a+b;
  if( p->op=='-' ) return a-b;
  if( p->op=='*' ) return a*b;
  if( p->op=='/' ) return a/b;
  if( p->op=='^' ) return pow(a,b);
  return a;
 } else {
  return atoi( p->node );
 }
}
```

* Kevin: sqrtmod.c

```
#include <stdio.h>

long long mod( long long a, long long p ) {
  if( a>=0 ) return a%p;
  else return (p-((-a)%p))%p;
}

long long powmod( long a, long b, long p ) {
  long long ex = 1, r = a;

  if( b==0 ) return 1;
  while( ex < b ) {
    if( 2*ex <= b ) {
      r = mod( r*r, p );
      ex *= 2;
    } else return mod( r*powmod(a,b-ex,p), p );
  }
  return r;
}

long long sqrtmod( long long a, long long p ) {
  long long S = 0, Q = p - 1, ainv = powmod( a, p - 2, p );
  long long x, R;
  long long W, V;
  long long i, j, y, z;

  while( Q%2==0 ) { S++; Q /= 2; }

  printf( "  %Ld = 2^(%Ld)*%Ld\n", p, S, Q );

  R = powmod( a, (Q+1)/2, p );
  W = 2;
  while( powmod( W, (p-1)/2, p )==1 ) W++;
  V = powmod( W, Q, p );
  printf( "  W = %Ld\n", W );

  while( 1 ) {
    printf( "  R = %Ld\n", R );
    y = mod( mod( R*R, p )*ainv, p );
    for( i = 0; i < S; i++ ) {
      if( y==1 ) break;
      y = mod( y*y, p );
    }
    if( i > 0 ) printf( "    i = %d\n", i );

    if( i==0 ) return R;
    z = V;
    for( j = 0; j < (S-i-1); j++ )
      z = mod( z*z, p );

    R = mod( R*V, p );
  }

  return R;
}

long long divmod( long long a, long long b, long long p ) {
  return mod( a*powmod( b, p - 2, p ), p );
}

void print_roots( long long a, long long b, long long c, long long p ) {
  long long det = mod( b*b - 4*a*c, p );
  long long r, x1, x2;

  if( det==0 ) printf( "{ %Ld }\n\n", divmod( -b, 2*a, p ) );
  else if( powmod( det, (p-1)/2, p )!=1 ) printf( "{ has no roots }\n\n" );
  else {
    r = sqrtmod( det, p );
    x1 = divmod( -b + r, 2*a, p );
    x2 = divmod( -b + (p-r), 2*a, p );
    printf( "{ %Ld, %Ld }\n\n", x1, x2 );
  }
}
```