'Index of programming contest code library Arithmetic:



bigini:::

(signed) integer arithmetic bignumber. j ava:

Java template for using large integer arithmetic (Biginteger) era:

Chinese remainder theorem diophantine\_sy s:

Linear system of diophantine equations (works for single equation too t ) euclid:

Euclidean algorithm eulerphi:

Computes the Euler phi (totient) function: given a positive n, return the number of integers between 1 and n relatively prime to n.

exteuclid:



Extended Euclidean algorithm exp:

Fast exponentiation expmod:

Fast exponentiation mod m factor:

Integer prime factorization factor\_large:

Integer prime factorization for larger integers ( >= 2 A 4 0)

fflinsolve:

Fraction-free solution of linear systems of equations (for systems with integer coefficients)

frac2dec:

Obtain the decimal representation of a fraction. fraction:

A rational number class.

infix:

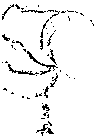
Parses and evaluates infix arithmetic expressions. int mult:

Multiply integer factors on the numerator, divide by the integer factors in.the denominator without overflow.

linsolv.e:

Solves linear systems of equations with LU decomposition. mult:

Multiply factors on the numerator, divide by the factors in the

denominator without overflow. ratlinsolve:

Rational solution of linear systems of e<;IUations (can be solved by fflinsolve as well).

roman numerals

Converts between Arabic and Roman numerals.

Geometric (mostly 2-D):

areapoly:

Computes the signed area of a simple (no self-intersection) polygon. ccw:

Determines the orientation of 3 points {counterclockwise, clockwise, undefined) .

circle\_3pts:

Computes the center and radius of a circle given 3 points. convex hull:

Computes the convex hull of a list of points.

dist3D:

Computes the distance between two points, a point and a line segment,

1



·

' . \. '

""·=·;

two line segments, or a point and a triangle in 3D. There are also corresponding versions for infinite lines and infinite planes.

dist line:

Computes the distance of a point to a line .. greatcircle:

Computes the distance between two points on a sphere along the surface. Also has routines to convert between Cartesian coordinates and spherical coordinates.

heron:

Computes the area of a triangle given the lengths of 3 sides. intersect circle circle:

Computes the intersection of two circles. intersectTF:

Given two line segments, return whether they intersect or not (but doesn't return the point of intersection)

intersect line:

Given two 2-D line segments, return whether they intersect or not, and return the point of intersection if there is a unique one.

intersect iline:

Given two 2-D lines (infinite), return whether they intersect or not, and return the point of intersection if there is a unique one.

intersect iline circle:

Given an infinite 2-D line and a circle, return whether they intersect and also the point(s) of intersection.

pointpoly:

Given a polygon and a point, determines whether the point is in the 'polygon. The behaviour when the point is on the boundary is left to the user.

polygon\_inter:

Given two convex polygons, compute their intersection as another polygon.

**Graph:**

bellmanford:

Computes the shortest distance from one vertex to all other vertices. Also computes the paths. It is slow (O(nA3)) but handles negative weights. Can also be used to detect negatiye cycles.

bfs\_path:

Computes the shortest distance from one vertex to all other vertices. Also computes the paths. The edges in the graph must have equal cost.

bicomp:

Finds the biconnected components and articulation points of a graph. dijkstra:

Computes the shortest distance from one vertex to all other vertices. Also computes the paths.

dijkstra\_sparse:

Same as dijkstra but for sparse graphs. Complexity O{{n+m) log(n+m)). eulertour:

Determines if there is an Eulerian tour in the graph. If so, find one.

floyd:

Computes the shortest distance between any two vertices. floyd path:

Like floyd, but also stores the paths. hungarian:

Maximum/minimum weight bipartite matching. O{NA3) matching:

Compute unweighted matching of bipartite graphs. (Matthew) mst:

2

V



r1

\.#'

V--\

*(* )

,.,

**::..;.it.<{**

Compute the. minimum spanning tree. mincostmaxflowdense:

Compute the minimum cost maximum flow in a network. Good for

dense graphs when maximum flow is small. Complexity is O(nA2 \* flow). mincostmaxflowsparse:

Compute the minimum cost maximum flow in a network. Good for sparse graphs when maximum flow is small. Complexity is

O(m log(m) \* flow) .

networkflow:

Compute the maximum flow in a network. Uses Ford-Fulkerson

with complexity O(fm) where f

and m is the number of edges. the maximum flow is small.

networkflow2:

is the value of the maximum flow Good for sparse graphs where

Compute the maximum flow in a network. Uses relabel-to-front with complexity O(nA3). Good for dense (but small) graphs where the maximum flow is large.

sec:

Compute the strongly connected components (and possibly the compressed graph) of a directed graph.

top\_sort:

Topological sort on directed acyclic graph (or detect if a cycle exists). O(n+m)

**Data Structures:**

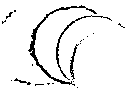
fenwicktree:

A data structure that supports the maintainence of cumulative sums in an array dynamically. Most operations can be done in O(log N) time where N is the number of elements.

suffixarray:

An O(n) algorithm to construct a suffix array (and longest common prefix information) from a string.

I



. (7/

*-*..\_*-*,.,

**Miscellaneous:**

asc subseq:

Longest (strictly) ascending/decreasing subsequence. binsearch:

Binary search that also returns the position to insert an element if it is not found.

common\_subseq:

Find the longest common subsequence of the two sequences. date:

A class for dealing with dates in the Gregorian calendar. dow;

Computing the day of the week. josephus:

Finding the last survivor and killing order of the Josephus problem. kmp:

Linear time st:r:ing searching r.outines. int prog:

Integer programming. simplex:

Linear programming by simplex algorithm. str\_rotation period:

Computes the lexicographically least rotation of a string, as well as its period.

unionfind:

Union-find implementation to compute equivalence classes. vecsum:

Find the contiguous subvector that gives the largest sum.

3

*D*

( '

zero one:

Zero-one programming.

*'\(*. *(* \I

\

"-,. ..

### <it /

r*,f* \

4

. *./:J*

Aug 19, 14 21:35 **2.sat.cc** Page 1/2

*II* 2SAT solver: rieturns *TIF* whether it is satisfiable -- O(n+m)

*II* - use NO'i;' () t:o negate a variable (works on negated ones too!)

*II* - ALWAYS use VAR() to talk about the non-negated version of the var i

*II* - use add clause to add a clause

*II* - one possible satisfying assignment is returned in val[), if

*II* it exists

*II* - To FORCE i to be true: add clause(G,VAR(i) ,VAR(i));

*II* - To implement XOR. -- say (i XOR j) :

*II* add clause(G,VAR(i),VAR(j)); add clause(G,NOT(VAR(i)),NOT( VAR(j)));

*II* NOTE-:- val[; is indexed by i for var i, not by VAR(i) !! !

#include <iostrearn>

#include <algorithm>

**#include <Stack>**

#include <Cassert>

#include <Vector> using namespace std;

canst int MAX VARS = 100; *II* maximum number of variables canst int MAX=NODES = 2\*MAX\_VARS;

struct Graph{

**int numNodes;**

vector<int> adj [MAX NODES] ; void clear() { -

**numNodes = O;**

for(int i=O;i<MAX NODES;i++) adj [i] .clear() ;-

Printed by Darcy Best

Aug 19, 14 21:35 **2sat.cc** Page 2/2

fill(comp,comp+G.numNodes,-1);

**for(int i;O;i<G.numNodes;i++)**

if (po [i] == -1)

DFS(i,G,C,P,S);

**return num\_scc;**

int VAR(int i) { return 2\*i; ) int NOT(int i) { rett1rn i 'l; }

void add\_clause(Graph &G, int v, int w) ( *II* adds (v J I w)

if (v == NOT(w)) return;

G.add edge(NOT(v), w);

)G.add--edge(NOT(w), v);

bool twoSAT{const Graph &G, bool val[]) { *II* assumes graph is built SCC(G);

for (int i = O ; i < G.numNodes; i += 2) ( if {comp[i] == comp[i+l)) return false; val[il2) = (comp[i] < comp[i+l) );

}

**return true;**

)

void add\_edge(int u,int v) {

if(find(adj [u) .begin(),adj [u) .end(),v) adj [u] .push\_back (v);

) ; )

int po[MA.X\_NODES) ,comp[MAX\_NODES) ;

**int num\_scc;** ·

adj [u) .end() )

*II* Declare this as a global variable if MAX NODES is large to

*II* avoid Runtime Error. - Graph G;

int main( ){

**int m,n;**

while{cin >> n >> m && (n I I m)) {

G.clear() ;

**G.numNodes = 2\*n;**

void DFS(int v, canst Graph& G, int& C, stack<int>& P,stack<int>& S)( po[v) = C++;

S.push(v); P.ptlsh(v);

for(unsigned int i=O;i<G.adj [v] .size();i++){ int w = G.adj [v) [i);

if (po [w) == -l.) (

for (int i= O; i < m; i++) (

**cotit << 11 Enter two variables for clause (1** - 11 << **n**

<< 11 ) , **negative means negated;** 11 •

int x, y;

**cin >> x >> y;**

. DFS(w,G,C,P,S);

else if(comp[w) == -1){

while(!P.empty() && (po[P.top()] > po[w])) P.pop();

int varl = VAR(abs{x)-1), var2

if (x < 0) varl = NOT(varl); if (y < 0) var2 = NOT(var2); add\_clause(G, varl, var2);

VAR(abs(y)-1);

) )

if ( !P.empty() &!, P.top () while (!S.empty()) {

int t = S.top();

S.pop();

comp[t) = num sec; if(t == v) -

break;

)

P.pop();

**num\_scc++;**

v)(

bool val[MAX VARS) ;

if· (twoSAT(G-;- val))

**for (int i = O; i < n; i++)**

cout << val[i) <<

}

cout << endl; else {

**cout << 11 Impossible 11 << endl;**

)

**return O;**

int SCC(const Graph& G){

**num sec = O;** int-C = l; **stack<int> P,S;**

fill(po,po+G.numNodes,-1);

•• Jay August 19, 2014

\\_ \_./

'-.....\_/.

2sat.cc

.

.-...,,,/ .'

-('·"-- *17*

*Jh*

,, ***J ,***



*rr* .

(> ;

Printed by Darcy Best

Aug 19, 14 21:35 **asc subseq.cc** Page 1/3

*I\**

* Longest Ascending Subsequence
* Author: Howard Cheng
* **Reference:**

Gries, D. The Science of Programming

* Given an array of size n, asc\_seq returns the length of the longest
* ascending subsequence, as well as one of the subsequences in S.
* sasc seq returns the length of the longest strictly ascending
* subsequence. It runs in O(n log n) time.
* **Also included are simplified versions when only the length is needed.**
* Note, If we want to find do the same things with descending
* subsequences, just reverse the array before calling the routines.

\*/

#include <iostream>

#include <algorithm>

**#include <vector>**

**#include <Cassert> using namespace std;**

int asc seq(int A[], int n, int

Aug 19, 14 21:35

. areapoly.cc

Page 1/1

*I\**

* Area of a polygon
* uthor: Howard Cheng
* **Reference:**

[http,//www.exaflop.org/docs/cgafaq/cga2.htm1](http://www.exaflop.org/docs/cgafaq/cga2.htm1)

* This routine returns the SIGNED area of a polygon represented as an
* array of n points (n >= 1). The result is positive if the orientation is
* **counterclockwise, and negative otherwise.**

\*/

**#include <iostream>**

**#include <iomanip>**

#include <cmath>

**#include <cassert> using namespace std;**

struct Point { double x, y;

) ;

double area\_polygon( Point polygon( ], int n)

{

double sum = 0.0;

for (int i = O; i < n-1; i++) {

sum += polygon[i) .x \* polygon[ i+l] .y - polygon[ i] .y \* polygon[i+l] .x;

sum += polygon[n-1) .x \* polygon[O] .y - polygon[n-1) .y \* polygon[O] .x; return sum/2.0;

int main (void)

{

**Point \*polygon; int n;**

while (cin >> n && n > .O) ( polygon = new Point(n]; assert(polygon);

for (int i •O; i < n; i++) (

cin >> polygon[i) .x ?> polygon[ i] *.y ;*

)

**cout**

**<< "Area = << fixed << setprecision(2)**

<< area\_polygon(polygon, n) << endl;

**11**

delete[) polygon;

)

**return 0 ;**

-

{

S [] )

vector<int> last(n+l), pos(n+l), pred(n);

if (n == 0) { return O;

int len = 1;

last[l] = A[pos[l) = OJ ;

for (int i = l;\_ i < n; i++) {

int j = upper bound(last.begin()+l, last.begin()+len+l, A(i]) - last .begin (f ;

pred[i] = (j-1 > 0) ? pos [j-1) : -1; last[j) = A(pos[j] = i);

len = max(len, j);

int start = pos[len);

for (int i = len-1; i >= O; i--) S[i) = A(start) ;

assert(i == 0 I I pred[start) < start);

start = pred[start) ;

return len;

int asc seq(int A(), int n)

-

{

vector<int> last(n+l);

if (n == O) {

**return O ;**

**int len = l;**

last (l] = A [OJ ;

Tuesc;lay August 19, 2014

for (int i = l; i < n; i++) {

int j = upper bound(last.begin()+l, last.begin()+len+l, A[il ) - last .begin();

last [j) = A[i);

areapoly.cc, asc\_subseq.cc 2/77



!Printed by Darcy Best

Aug 19, 14 21:35

**asc \_ subseq.cc**

Page 2/3

len = max(len, j);

return len;

i(nt sasc-seq(int A(], int n, int S(])

vector<int> last(n+l), pos(n+l), pred(n);

if (n == O ) (

**return O;**

int len = l;

last[l] = A(pos(l] = OJ ;

for (int i = 1; i< n; i++) (

int j = lowerbound(last.begin()+l, last.begin()+len+l, A(i] ) -

last.begin() ;

pred[i] = (jl > 0) ? pos [j-1] : -1; last[j] = A[pos[ j] = i];

len = max(len, j);

int start = pos[len] ;

for (int i = len-1; i >= O; i--) ( S(i] = A(start];

start = pred(start];

**return len;**

int sasc seq(int A(], int n)

{

-

vector<int> last(n+l);

if (n == O ) (

**return O;**

int len = 1; last(l] = A[O];

for (int i = l; i< n; i++) (

int j = lower\_bound(last.begin()+l, last .begin(),

last [j J = A [i];

len = max(len, j);

last.begin()+len+l,

A(i].) -

return len;

int main (void)

(

int \*A, \*S, n, i, k;

while (cin >> n I & n > 0) A = new int (n] ;

s = new int[n];

**for (i = O; i < n; i++)**

cin >> A(i] ;

}

k = asc seq(A, n, S);

cout <<-"length = " << k << endl; for (i = O; i c k; i++)

cout << S( i] << " "·

}

G ay August 19, 2014

cout << endl;

Aug 19, 14 21:35

**asc subseq.cc**

Page 3/3

k = sasc\_seq(A, n, S);

**cout cc ''length = " cc k cc endl;**

for (i = O; i < k; i++)

**cout << S [i] cc**

11 11 ***i***

}

**cout << endl;** delete[] A; delete(] S;

)

**return O;**

- asc\_subseq.cc

-

*--:J ,*

*l3];*



# cLj *)*

Aug 19, 14 21:35 bellmanford.cc Page 1/3

*I\**

\* Bellman-Ford Shortest Path Algorithm

* Author: Howard Cheng
* Given a weight matrix representing a graph and a source vertex, this
* algorithm computes the shortest distance, as well as path, to each
* of the other vertices. The paths are represented by an inverted list,
* such that if v preceeds immediately before w in a path from the
* **source to vertex w, then the path P[w] is v. The distances from**
* the source to v is given in D[v] (DISCONNECT if not connected) .
* Call get\_path to recover the path.

\*

\* Note: the Bellman-Ford algorithm has complexity O(nA3), but it works even

\* when edges have negative weights.. As long as there are no negative

cycles the computed results are correct.

We can make this O(n\*m) if we use an adjacency list representation. This works for directed graphs too.

* Printed by Dart s

·.You can use this to detect negative cycles too. See code.

Aug 19, 14 21:35

**bellmanford.cc**

Page 2/3

*I\** on the path from src to

*\*I*

) }

int get\_path(int v, int P[],·int path(])

{

int A(MAX NODES];

**.int i, k;-**

k = O ;

A(k++) = V;

while (P[v) != -1) {

V = P(v];

**A [k++} = v;**

)

for (i = k-1; >= O ; i--) {

path [k-1-i] A(i];

)

return k;

int main{void)

{

**int m , w , num;**

**int i, j ;**

int graph(MAX NODES] [MAX NODES] ;

int P(MAX NODES] [MAX NODES], D[MAX NODES] [MAX NODES];

int path [MAX\_NODES) ;- - -

/• clear graph \*/

for (i = O; i < MAX\_NODES; i·++) { for {j = O; j C MAX NODES; j++)

graph[i] [j] = DISCONNECT;

} }

/• read graph \*/

**cin >> i >> j >> w;**

while (!(i == -1 && j == -1) ) {

assert(O <= i && i c MAX NODES && 0 <= && < MAX\_NODES); graph[i] [j] = graph[j] ( i] = w ;

**cin** >> i>> i >> **w;**

for (i = O; i < MAX\_NODES; i++) {

bellmanford(graph, MAX\_NODES, i, D[i], P[i]);

/\* do queries \*/

**cin >> i >> *j ;***

"!hile (!(i == -1 && j == -1)) (

assert(o <= i && i c MAX NODES && 0 c= j && j c MAX NODES);

**cout << i** << << j <<-11 : << **D ( i] ( j ] << endl ;** -

11 11

11

for (m = j; m != -1; m = P[i] [ml ) (

)

**cout <c c< m;**

11

11

cout << endl;

num get\_path(j, P[i], path); for (m = O; m < num; m++)

}

**cout** << **cc path [m]** ;

'1 11

cout << endl;

**cin >> i >> j;**

\*

\*/

**#include <iostream>**

**#include <Climits>**

**#include <Cassert>**

using namespace std; canst int MAX NODES = 20;

canst int DISCONNECT = INT\_MAX;

/\* assume that D and P have been allocated \*/

void bellmanford( int graph[MAX\_NODES] [MAX\_NODES], int n, int src, int D[J, int P[])

**int v, w, k;**

for (v D[v]

P(v]

O ; v < n; v++) ( INT MAX;

-1;-

}

D[src] O;

for (k = Q; k c n-1; k++) (

**for (v = O; v < n; v++)** {

for (w·= O; w < n; w++) {

if (graph[v] [w] != DISCONNECT && D[v] != INT\_MAX) { if (D [w) == INT\_MAX 1 1 D [w) ,D [v] + graph[v] (w])

D [w] = D [v] + graph [v] (w] ; P[W] = V;

else if (D (w] == D (v] + graph [v] [w] )

*I\** do some tie-breaking here \*/

) }

/\* the following loop is used only to detect negative cycles, not \*/ *(\** needed if you don't care about this \*/ for (v = O; v < n; v++) {

for (w = O; w < n; w++) (

if (graph(v] (w] != DISCONNECT && D(v) != INT MAX) {

if (D (w] == INT\_MAX 1 1 D [w] > D (v] + graph v] [w)) (

*I\** **if we get here then there is a negative cycle somewhere \*/**

Tuesday August 19, 2014

bellmanford.cc 4/77

Printed by Darcy Best

Aug 19, 14 21:35

bellmanford.cc

Page 3/3

return o ; Aug 19, 14 21:35 **bfs path.cc** Page 1/3

/\*

* Shortest Path with BFS
* Author: Howard Cheng
* Given a graph represented by an adjacency list, this algorithm uses
* BFS to find the shortest path from a source vertex to each of the
* other vertices. The distances from the source to v is given in D[v), and
* D[v] is set to -1 if the source vertex is not connected to w. Also,

\* **the shortest path tree is stored in the array P.**

\* Call get\_path to recover the path.

* + Note: All edges must have the same cost for this to work. This algorithm has complexity O(n+m).

*\* I*

#include <iostream>

#include <cassert>

#include <algorithm>

#include <queue>

**using namespace std;**

const int MAX\_NODES = 100;

struct Node {

int deg; /\* number of outgoing edges \**I* int adj [MAX\_NODES);

/\* the following is not necessary, but useful in many situations \*/

) ; int cost[MAX-NODES];

v{oid BFS-shortest-path(Node graph[), int n, int src, int D(], int P[)) char used[MAX NODES);

queue<int> q,-

**int i, V, W;**

fill(used, used+MAX NODES, 0);

q.push(src); - used [src) = 1;

for (i = O; i < MAX NODES; i++) { D[i) = -1; -

P[i) = -1;

}

D[src) = O ;

while (!q.empty())

v = q.front();

q.pop();

for (i = O ; i < graph[v) .deg; i++) { w = graph[v) .adj [i);

if (!used[w)) {

D [w) = D [v) + 1;

P[w] = V;

q.push.(w); used(w] = l;

else if (D[v) + 1 == D[w]) {

*I\** put tie-breaker here \*/

/\* eg. find largest path in lexicographic

*I\** is considered in REVERSE! P(w] = max(P[w) , v) ;

order, when the- path \*/

\*/

*G,/*

jay August 19, 2014

L.-----' -----1/

vF

bellmanford.cc, bfs\_path.cc -,

/

/

*( - -*

\ ,.'

*··.'-. )*

*J i -..*

*,i I*





*a* -/

-

.*,*-*n*

*i*

.--.....

/

Printed by Da t

Aug 19, 14 21 35

bfs **path.cc**

Page 2/3

void clear(Node graph[], int n)

[

int i;

**for (i = O; i c n; i++)**

graph[i] .deg = O;

void add edge(Node graph[], int v, int w, int cost)

[

-

**int i;**

*I\** make sure that we have no duplicate edges \*/ for (i = O; i, < graph[v) .deg; i++) {

if (graph[v] .adj[i] == w) ( assert(O);

graph[v] .cost[graph[v] .deg] = cost;

graph[v) .adj [graph[v) .deg] = w; graph[v) .deg++;

int get\_path(int v, int P[), int path[])

(

int A[MAX NODES];

int i, k;-

k = O;

A[k++] = V;

while (P [v] != -1) {

V = P[v];

A[k++] = v;

}

for ( i = k-1; i >= O; i--) (

path[k-1-i) = A[i];

}

**return k;**

int main (void)

{

**int v, w, num;**

**int i ;**

Node graph[MAX\_NODES] ;

int P[MAX NODES] [MAX NODES), D[MAX NODES) [MAX NODES);

int path[MAX\_NODES] ;- - -

clear(graph, MAX\_NODES);

**while (cin >> v >> w && v >= O && w >= O)** {

} -

add edge(graph, v, w, 1);

for (i = O; i < MAX\_NODES; i++) {

BFS\_shortest\_path( graph, MAX\_NODES, i, D[i], P[i]);

while (cin >> *v* >> w && v >= O && w >= O) (

**cout << v** << << **w** <<

11 11

11 :

11

<< **D [v) (w] << endl ;**

num = get\_path(w, P[v), pah);

assert (D [v) [w) == -1 I / num == D [v] [w] +l); for (i = O; i < num; i++) {

)

**cout** << << **path [i]** *;*

11 11

**cout << endl;**

Aug 19, 14 21:35

**bfs\_path.cc**

Page 3/3

**return O;**

Tuesday August 19, 2014

bfs\_path.cc *6177*

Aug 19, 14 21:35 bi **comp.cc** Page 1/2 Aug 19, 14 21:35. **bicomp.cc** Page 2/2

*I\**

* Biconnected Components
* Author: Howard Cheng

\* Date: Oct 15, 2004

* The routine bicomp() uses DFS to find the biconnected components in
* a graph. The graph is stored as an adjacency list. Use clear\_graph()
* and add\_edge(l to build the graph.
* Note: This works only on connected graphs. See comment below in code.

\*

* The code simply prints the biconnected components and the articulation
* points. Replace the printing code to do whatever is appropriate.
* NOTE: some articulation points may be printed multiple times.

*\*I*

**#include ciostream>**

#include <stack,

#include <algorithm>

**#include ccassert**

**/\* back up from recursion \*/**

if (G[w] .back >= G[v] .dfs) {

*I\** new bicomponent *\*I*

**cout << 11 edges in new biconnected component: 11 << endl;**

while (v\_stack.top() != v I f w\_stack.top() != w) {

**cout << v\_stack . top {)** << 11 11 << **w\_stack . top () << endl ;**

v\_stack.pop (); w\_stack.pop ();

}

cout < v stack.top() << " **u << w\_stack.top() << endl;**

v\_stack.pop (); w\_stack.pop ();

if (pred != -1) {

**cout << ''articulation point:** 11 << **v << endl;**

}

} else {

G[v) .back = min(G[v) .back, G(w] .back);

}

else (

*I\** w has been examined already *\*I*

G[v) .back = min(G[v) .back, G[w] .dfs);

}

**using namespace std;**

if (pred == -1 && child > 1) {

**cout** << 11 **articulation point :** '1 << **v << endl ;**

/\* **maximum number of nodes, maximum degree, and maximum number of edges \*/**

**canst int MAX N = 1000;**

canst int MAX=DEG 4;

struct Node { int deg;

int nbrs[MAX DEG]; int dfs, back;

} ;

int dfn;

void clear graph(Node G(], int n)

-

(

**int i;**

for (i = O ; i < n; i++) ( G(i] .deg = 0 ;

} }

void bicomp(Node G[), int n)

(

int i;

**stack<int> v\_stack, w\_stack;**

dfn = O;

for (i = O ; i < n; i++) { G[i) .dfs = O ;

}

do\_dfs(G, 0, -1, v\_stack, w\_stack),

*II* NOTE: if you·wish to process all connected components, you can simply

*J I* run the following code instead of the line above:

*II*

*II tor* .(int i = O ; i < n; i++) {

*I I* if (G [il .dfs == o ) {

void add edge (Node G (), int u, int v)

*II* do dfs(G, i, -1, *v*

stack, w stack);

{ -

G[u).nbrs[G[u].deg++) = v;

G[v] .nbrs[G[v) .deg++] = u·

*II* - - -

*I I*

) int main(void)

{

void do dfs(Node G[], int v, int pred, stack<int> &v stack,

- **.stack..:int> &w\_s tack)** -

int i, w, child = O;

G[v] .dfs = G[v] .back = ++dfn;

Node G [MAX N) ;

**int n, m, I, u, v;**

**cin >> n;**

clear**>>**graph(G, n);

**cin m;**

for (i = O ; i < G[v) .deg; i++)

w = G[v) .nbrs[i) ;

for (i = O ; i < m;

**cin >> u >> v;**

i++l {

if (G[w) .dfs < G[v] .dfs && w

*I\** back edge or unexamined v stack.push(v);

w=stack.push(w);

!= pred) { forward edge *\*I*

} add-edge.(G, u-1, v-1); bicomp (G, n) ;

**return O ;**

)

if (!G(w].dfs) { .

do dfs(G, w, v, v stack, w\_stack); child<·+; -

*cs~~;T~~*

/ay August 19, 2014

''""' 'le:'.!/ '

/

*·:;*

bicomp.cc

*0S1·..* (::Z)

--- */*

- *- I*

Aug 19, 14 21:35 bigint.cc Page 1/11

*I\**

\* Big integer implementation

* Author: Howard Cheng

\* .

* Each digit in our representation represents LOG\_BASE decimal digits

\*

*\*I*

**#include <Vector>**

#include <String>

#include <CStdio>

#include <CCtype>

**#include <iostream>**

#include <algorithm>

#include <Utility>

#include <Cassert>



•> *((* /

-

>

/

Printed by Da:St

Aug 19, 14 21:35 **bigint.cc** Page 2/11

Biginteger operator+(const Biginteger &a, canst Biginteger &b) Biginteger operator-(const Biginteger &a, canst Biginteger &b) Biginteger operator\*(const Biginteger &a, canst Biginteger &b) Biginteger operator\*(const Biginteger &a, Digit b);

Biginteger operator<<(const Biginteger &a, Digit b);

Biginteger operatorl(const Biginteger &a, canst Biginteger &b); Biginteger operatorl(const Biginteger &a, Digit b);

Biglnteger operator%(const Biginteger &a, canst Biginteger &b);

Digit operator%( onst Biginteger &a, Digit b);

Biglnteger power(Biginteger x, Digit y);

istream &operator>>(istream &is, Biginteger &a); ostream &operator<<(ostream &as, corrst Biginteger &a);

void Biginteger::normalize()

{

using namespace std;

using namespace std::rel\_ops;

typedef long long Digit;

#define BASE 1000000000

#define LOG BASE 9

#define FMT-STR "%lld"

#define EMT=STRO "%09lld"

class Biginteger

**private:**

if (mag.size() == 0) {

**return;**

}

vector<Digit>::iterator p = mag.end(); do {

if (\*(--p) != O ) break;

) while (p != mag.begin( ));

if (p == mag.begin() && \*p == O ) (

clear(); **sign = O;** else {

mag.erase(++p, mag.end());

int sign; vector<Digit> mag;

**void normalize();**

*II* +l = positive, O

*II* magnitude

zero, -1 negative

Biginteger::Biginteger( Digit n)

(

public:

Biginteger(Digit n = O);

Biginteger( const string &s);

long long toLongLong() canst; string toString() canst;

void clear(); *II* set to zero

*II* comparison

*II* no error checking

*II* convert to long long (assumes no overflow)

*II* convert to string

if (n == O ) sign = O; **return;**

)

if (n < O ) { sign =e -1; n = **-n;** else { sign = 1;

bool operator<(const Biginteger &a) canst;

bool operator==(const Biginteger &a) canst; bool isZero() canst;

*II* arithmetic

Biginteger &operator+=(const Biginteger .&a);

Biginteger &operator-=( const Biginteger &a); Biginteger &operator\*=(const Biginteger &a); Biginteger &operator\*=(Digit a);

Biginteger &operator<<=(Digit a);

Biginteger &operatorl={const Biginteger &a); Biginteger &operatorl=(Digit a);

Biginteger &operator%=( const Biginteger &a);

friend Digit operator%(const Biginteger &a, Digit b);

*II* we have \*this = b \* q + r

*II* r is such that O <= r < l bl

void divide(const Biginteger &b, Biginteger &q, Biginteger &r) canst;

while (n > O J {

mag.push back(n % BASE); n *I=* BASE;

Biginteger: :Biginteger(const string &s)

{

int 1 = O;

bool zero = true; bool neg = false;

clear();

**sign = l ;**

if (s[l] == '-') neg = true; l++;

*II* root = floor(sqrt(a)).

Returns 1 if a is a perfect square, 0 otherwise. I I for (; 1 < s.length(); l++) {

void divide(Digit b, Biginteger &q, Digit &r) canst;

*II* assume >= O

\*this \*= 10;

int sqrt(Biglnteger &root) canst; \*this += s[l] - 'O';

} ; zero &= s[l] == '0';

Tuesday August 19, 2014 bigint.cc 8/77

Aug 19, 14 21:35 **bigint.cc**

Page 4/11

bool Biginteger: :operator==(const Biginteger &a) canst

{

return sign == a.sign £& mag == a.mag;

bool Biginteger: :isZero() canst

{

**return sign == O;**

Biginteger &Biginteger: :operator+=(const Biginteger &a)

{

if (a.sign == O) ( return \*this;

else if (sign == O) {

**sign = a.sign;** mag = a.mag; **return \*this;**

else if (sign < O && a.sign > 0) {

Biginteger b(a); sign = l;

b -= \*this; return \*this = b;

else if (sign > O && a.sign < O) ( Biginteger b(a);

b.sign = l;

return (\*this) -= b; else {

Digit carry = O;

unsigned int limit = max(mag.size[), a.mag.size()); for (unsigned int i = O; i < limit; i++) (

Digit sl = .(i < mag.size()) ? mag[i] : O;

Digit s2 = (i < a.mag.size()) ? a.mag[i] : O; Digit sum = sl + s2 + carry;

Digit result = (sum < BASE) ? sum : sum - BASE; carry = (sum >= BASE);

if (i < mag.size()) {

mag[i] = result;

} else {

} mag.push-back(result);

}

if (carry) {

* mag.push back(carry) ;

-

}

**return \*this;**

} }

Biginteger &Biginteger: :operator-=(const Biginteger &a)

{



Aug 19, 14 21:35

big int.cc

Page 3/11

if (zero) {

clear() ;

)

if (neg) { sign = -1;

long long Biginteger:,toLongLong() canst

{

long long a = O;

for (int i = mag.size()-1; i >= O; i--) a \*= BASE;

a += mag[i];

)

**return sign \* a;**

.

string Biginteger: :toString() canst

{

char buffer[LOG BJ\SE+l]; string s; -

if ( isZero()) {

**return**

**11 0 11 ;**

else {

if (sign < O)

$ += **11 \_ 11** *i*

)

for (int i = mag.size0-1; i >= O; i--)

if [i == (int) [mag.size[)-1)) { sprintf[buffer, FMT\_STR, mag(i]); else {

sprintf(buffer, FMT\_STRO, mag[i) );

}

**s += buffer;**

**return s;**

void Biginteger: :clear()

{

sign = O; mag. clear() ;

bool Biginteger,:operator<(const Biginteger

{

&a) canst

if (sign != a.sign) { **return sign < a.sign;** else if (sign == 0) { **return false;**

else if (mag.size[) < a.mag.size()) return sign > o;

else if (mag.size[) > a.mag.size())

**return sign < O;**

else {

for (int i = mag.size0-1; i >= O; i--)

if (mag[i) < a.mag[i) ) (

**return sign > O;**

else if (mag[i] > a.mag[i])

**return sign < O;**

)

return false;

{

Tue '-v August 19, 2014

cS>c, .

'

···"'-',:,,-,../ .

if (a.sign == o) { return \*this;

else if (sign == 0) {

**sign == -a.sign; mag = a.mag;** return \*this;

else if (sign != a.sign) Biginteger b(aJ ;

b.sign \*= -1; return \*this += b; else if (sign < O) Biginteger b(a); b.sign \*·= -1;

**sign \*= -1;**

b -= \*this;

**return \*this = b;**

big int.cc *( 8(* '

(,d,,,;i

,,.. '.  ~~,~~

Cb,, *G\*

r

'

Printed by Darcy Best

Tuesday August 19, 2014

/

'-- -'

Aug 19, 14 21:35

**bigint.cc**

Page 6/11

Digit prod = a \* mag[i];

mag[i] = (carry + prod) % BASE; carry = (carry + prod) *I* BASE;

)

if (carry) {

}

mag.push back(carry);

-

return •this;

Biginteger &Biginteger: :operatorcc=(Digit a)

{

**assert(a >= O);**

if (sign) {

while (a-- > OJ { mag.insert(mag.begin(J , O);

)

**return \*this;**

Biginteger &Biginteger: :operatorl=(const Biginteger &a)

{

Biginteger temp(\*this), r; **temp.divide(a, \*this, r); return \*this;**

Biginteger &Biginteger: :operatorl=(Digit a)

(

Biginteger temp(\*this); Digit r;

temp.divide(a, •this, r);

**return \*this;**

Biginteger &Biginteger: :operator%=(const Biginteger &a)

{

Biginteger ·temp(\*this), q; temp.divide( a, q, \*this); **return \*this;**

void Biglnteger: :divide(const Biginteger &b, Biginteger &g,

Biginteger &r) canst *II* reference Knuth v.2 Algorithm D assert(!b.isZero());

if (b.mag.size() == 1)

Digit r2;

divide(b.sign\*b.mag[ O) , q, r2); r = r2;

**return;**

**r = \*this;**

if Cr.sign c O)

r.sign = l;

)

*q.*clear();

int n = b.mag.size(J ;

**int m = mag siz'e** () - **n;**

if (m >= 0) (

Biginteger v(b); g.mag.resize·(m+l); g.sign *=* l;

|  |
| --- |
| Aug 19, 14 21:35 bigint.cc Page 5/11 |
| ) else {  if (\*this == a) clear(); **return \*this;**  else if (\*this c a) { Biglnteger b(a);  b -= \*this; b.sign \*= -1; **return \*this = b;** else {  *II* we know that \*this > a  unsigned int limit = mag.size();  **Digit borrow = O;**  for (unsigned int i = O; i c limit; i++) { Digit sl = mag[i];  Digit s2 = (i c a.mag.size()) ? a.mag[i] 0 ;  Digit diff = sl - s2 - borrow;  mag(i] = (diff >= 0) ? diff : diff + BASE; borrow = (diff c 0);  )  normalize() ;  **return \*this;**  ) }  Biginteger &Biginteger: :operator\*=(const Biginteger &a)  {  Biginteger temp(\*this) ;  Biginteger c·  if (this == &a) {  C = **a;** *II* make a copy\_ to prevent clobbering it  canst Biginteger &b (this &a) ?· C a; clear();  if (b.sign) (  for (unsigned int i = O; i c b.mag.size(); i++ ) {  if (b.mag[i] != 0) (  \*this += (temp \* b.mag[i));  )  temp cc= l;  **sign \*= b.sign;**  }  **return \*this;**  Biglnteger &Biginteger: :operator\*=(Digit a)  { .  if (a <= -BASE I I *a >=* BASE) {  Biginteger b(a); return (\*this \*= b);  if (isZero(J ) {  **return \*this;**  else if (a *==* O ) {  clear();  **return \*this;** else if (a c O) **sign \*= -1;**  **a = -a;**  Digit carry = O; |
| for (unsigned int i O; i c mag.size(); i++) { |

*1wn*

Aug 19, 14 21:35 bigint.cc Page 7/11

*II* Dl: normalize

Digit d = BASE *I* (v.mag[n-1] + 1); *II* Book is wrong. See errata on web

**r \*= d;**

V \*c: **d;**

while (·(int)r.mag.size() < m+n+l) {

} r.mag.push-back(O);

*II* loop

for (int j = m; j >= O; j--) {

*II* D3: calculate q2

Digit t = r.mag(j+n) \* BASE + r.mag[j+n-1); Digit q2 = t *I* v.mag[n-1) ;

Digit r2 = t - 2 \* v.mag[n-1) ;

if (q2 == BASE I I q2 \* v.mag(n-2] > BASE \* r2 + r.mag(j+n-2]) { q2--; .

r2 += v.mag[n-1); if (r2 < BASE &&

(q2 == BASE I I q2 \* v.mag[n-2] > BASE \* r2 + r.mag[j+n-2])) { q2--;

r2 += v.mag(n-1);

} }

*II* D4: multiply and subtract Digit carry, borrow, diff; carry = borrow = O;

for (int i = O; i *<=* n; i++) {

t = q2 \* ( (i < n) ? v.mag(il : O) + carry; carry = t *I* BASE;

t %= BASE;

diff = r.mag(j+i) - t - borrow;

r.mag[j+i] = (diff >= o I I i == n) ? diff diff + BASE; borrow = (diff < 0);

*II* DS: test remainder q.mag(j)- = q2;

if (r·.mag(n+j] < O) {

*II D6:* add back q.mag(j)--;

carry = o ;

for (int i = O; i < n; i++) {

t = r.mag[j+i] + v.mag[i) + carry; r.mag[j+i) = (t < BASE) ? t : t - BASE; carry = (t >= BASE);

)

r.mag[j+n] += cari-y;

}

q.normalize ();

**r . normali ze** () ;

*II* DB: unnormalize

r *I=* d;

*II* normalize

if (sign < o && b.sign > O) { q.sign \*= -1;

**r \*= -1;**

if ( !r.isZero ()) {

**r += b;**

q -= l;

J

} else if (sign > O && b.sign < 0) {

*(1\_!*

Aug 19, 14 21:35

**bigint.cc**

Page 8/11

q.sign \*= -1;

else if (sign < O && b.sign < O && !r.isZero()) r += b;

**r** \*= -1;

q += l;

void Biginteger::divide(Digit b, Biginteger &q, Digit &r) canst

{

if (b <= -BASE 1 1 b >"= BASE)

.Biginteger bb(b), rr; divide(bb, q, rr);

r = rr.toLongLong();

**return;**

int bsign = 1;

if (b < 0) {

b \*= -1;

bsign = -1;

}

q.clear();

**r = O ;**

for (int i = mag.size()-1; i >= O; i--) ( Digit t = r \* BASE + mag[i];

if (t *I* b > o ) {

q.sign =·l;

-}

q.mag.insert(q.mag.begin(), t *I* b); r = t - q.mag[O) \* b;

}

*II* normalize q.normalize() ;

if (sign < 0 && bsign > O) ( q.sign \*= -1;

r \*= **-1;**

if (r) {

}

**r += b;**

q -= 1;

else if (sign > o && bsign < O) ( q.sign \*= -1;

else if (sign < o && bsign < 0 && r) (

**r = b - r;**

**q += l;**

} }

int Biginteger,:sqrt(Biginteger &root) canst

{

assert(sign >= O); root.clear();

if (sign == O)

**retuxn 1;**

*II* figure out how many digits there are

**Biginteger x, r, t2;**

r.sign *=* 1-;

int d = mag.size();

int root\_d = (d % 2) ? (d+l)l2

if (d % 2) {

r.mag.resize(l);

d *I* 2;

*....\_,,,,,,.: ·*

'ay August 19, 2014

ra

&:

bigint.cc

*:* t:zj





*r* '.

-

*17* 1

*/*

/

\

(L, ,'

/

---..,

'')

*i*

**Printed by**

*/*

**Darcy est**

**Aug** 19, 14 **21:35**

r.mag[O] = mag[--d]; else { r.mag.resize(i), r.mag[l) = mag[--d);

r.mag[O) = mag[--d];

**bigint.cc**

**Page 9/11**

root.sign = l;

*II* figure out one digit at a time

for· (int k = root\_d - 1; k >= 0; k--) {

*II* invariant: result is the sqrt (integer part) of the digits processed

*II* so far

*II* look for next digit in result by binary search

**x = root \* 2;**

**X** <<= **l;**

Digit t;

Digit lo = 0, hi = BASE;

while (hi - lo > 1) {

Digit mid = (lo + hi) *I* 2;

x.mag[O] = t = mid;

t2 = X \* t;

if (t2 < r I I t2 == r)

lo = mid; else { hi = mid;

{

}

**root** <<=:: **1;**

root.mag[O] = lo;

x.mag[O] = t = lo;

*II* form the next r

**t2 = X \* t;**

r -= t2;

**r** <<= **1;**

r += (d > 0) ? mag[--d) O;

**r** <<= **1;**

r += (d > 0) mag[--d] 0 ;

**return r . isZero** () ;

Biginteger operator+(const Biginteger &a, const Biginteger &b)

{

Biginteger r(a);

r += b;

return r;

Biginteger operator-( const Biginteger &a, canst Biglnteger &b)

{

Biginteger r(a);

r -= b;

**return r;**

Biginteger operator\*(const Biginteger &a, canst Biginteger &b)

{

Biginteger r(a);

**r \*= b;**

**return r;**

Biglnteger operator\*(const Biginteger &a, Digit b)

*t*

**Aug 19, 14 21:35**

Biginteger r(a);

**r \*= b;**

**return r;**

**bigint.cc**

**Page 10/11**

Biginteger operator<<( const Biginteger &a, Digit b)

{

Biginteger r(a);

**r** <<= **b;**

**return r;**

Biginteger operatorl( const Biginteger &a, canst Biginteger &b)

{

Biginteger r(a);

r *I=* b;

return r;

Biginteger operatorl(const Biginteger &a, Digit b)

{

Biginteger r(a);

r *I=* b;

**return r;**

Biginteger operator%( const Biginteger &a, canst Biglnteger &b)

{

Biginteger r (a);

**r %= b;**

**return r;**

Digit operator%( const Biginteger &a, Digit b)

{

Digit r;

if (b > 0 && b < BASE) {

r = 0;

**for** (int i = a.mag.size()-1; i >= O;

r = ((r \* BASE) + a.mag[i)) % b;

i--)

{

}

if (a.sign < O) {

r = (b - r) % b;

J

**return r;**

Biginteger q;

a.divide(b, *q,* r);

**retu-z:n r;**

B{iginteger power(Biginteger x, Digit y)

.

Biginteger result(l), sx(x);

assert (y >= 0) ; while (y > 0) { if (y & OxOl)

y--;

**result \*= sx;**

else {

**SX \*= SX;**

**y** >>= **l;**

). )

return result;

Tuesday August 19, 2014

bigintcc **12/77**

*I•*

* **Binary Search**

\*

* Author: Howard Cheng

\* Note: you may wish to use the STL functions lower\_bound and upper\_bound

* **instead .**
* Given a sorted array A of size n, it tries to find an item x in the
* **the array using binary search. The function returns non-zero if**

\* x is found, and zero otherwise. Furthermore, if it is found, then

* A[index] = x. If it is not found, then inde,c is the place x should
* be inserted into A.
* ie. A[i] <= X
* X < A[i]

**for O <= i < index**

**for index <= i < n**

* This routine is written for integer arrays, but can be adapted to
* other types by changing the comparison operator.

\*

* There is also an insert routine here that will insert the element into
* the right place after the array has been reallocated (if necessary) to
* **store n+l elemnts.**

*\*I*

**#include <iostream>**

#include <cassert> using namespace std;

b{ool bin-search(const int A[], int n, int x, int &index)

**int l, u, m;**

if (n <= 0 I I x < A[O]) { *II* check the first element, but only if it exists

**index = O;**

return false;

}

if (A[n-1] < x) { **index = n; return false;**

}

if (x == A[n-1]) **index = n-1; return true;**

}

l = O;

**u = n-1;**

while (1+1 < u) {

assert(A[l] <= x && x < A[u]); m = (l+u)l2,

if (A[m] <= x)

1 = m;

else {

**u = m;**

}

if (A[l] == x)

index = l;

**rett1rn true;**

else {

**index = u; return false;**

v{oid insert(int A[], int n, int.

x, int index)



Aug 19, 14 21:35

**binsearch.cc**

Page 1/2

|  |  |
| --- | --- |
| Aug 19, 14 21:35 bigint.cc · Page 11/11 | |
| **istream &operator> > {isCream &is, Biginteger &a)**  {  string s;  **char c** =  is.get(c);  .while (!is.eof () && isspace (c)) { is .get(c) ;  }  if ( is .eof ()) {  if (isdigit(c)·1 (  a = (int)(c ·- ' O ' ) ;  is.clear() ;  )  **return is;**  if (c == '-') {  **s** = !' -ll .  } else { '  is.unget ();  if (!isdigit(c)) (  **return** is;  } }  is.get(c);  while (!is.eof() && isdigit(c)) {  **S += C;**  is.get(c);  }  if (!is.eof()) { is.unget ();  }  **a .:::. s;**  is.clear() ;  **return is;**  o(stream &operator<<(ostream &as, canst Biginteger &a)  .  return (as << a.toString());  }  int main(void)  {  **Biginteger a, b;**  while (cin » a » b && (!(a == 0) 1 1 ! (b == 0) ) ) {  **cout << 11 a** = 11 << **a << endl ;**  **caut << 11 b = u << b << endl;**  if (!(a < O ) ) {  if (a.sgrt(b)) {  **11**  } **cout << perfect square 11 << endl;**  .  **cout < < 11 sqrt (a)** = 11 << **b < < endl ;**  } }  **return O ;** |  |

-/1y August 19, 2014

&

7.::,\_...,,

bigint.cc, binsearch.cc

*0(::3,*

(;J.l



*r* "'\

*/ /*

int i

for ( = n-1; i >= index+l; i--) A[ ) = A[i-1];

}

A[index] = x;

int main(void)

{

int A[lOOOO];

**int n, i, x, index;**

*II* implements binary insertion sort, ·but only keeps the unique elements

**n = O;**

while (cin >> x && n < 10000) {

if {!bin\_search(A, n, x, index))

**n++;**

insert(A, n, x, index);

}

**cout << ntist:";**

for (i = O; i < n; i++) {

**cout** << " << **A[i] ;**

11

if (i == index) {

**cout << 11 \*11 ;**

*II* show which one is just inserted

**cout << endl;**

**return O ;**

Aug 19, 14 21:35

**binsearch.cc**

Page 2/2

*((* '\

­

/ ' **Printed by Dar st** )

Aug 19, 14 21:36 **ccw.cc Page 1/2**

*I\**

\* Orientation analysis

\* Author: Howard Cheng

* Reference:

http:llwilma.cs.brown.edu/courseslcs016/packet/node18.html

* Given thee points a, b, c, it returns whether the path from a to b to c
* is counterclockwise, clockwise, or undefined.
* Undefined is returned if the 3 points are colinear, and c is between
* a and b.

\*/

**#include <.iostream>**

#include <Cmath>

using namespace std;

*!•*how close to call equal \*/ const double EPSILON = lE-8;

struct Point { double x, y;

};

*I\** counterclockwise, clockwise, or undefined *\* I*

enum Orientation {ccw, cw, CNEITHER);

O{rientation ccw(Point . a, Point b, Point c) double dxl = b.x - a.x;

double dx2 = c.x - b.x;

double dyl = b.y - a.y; double dy2 = c.y - b.y; double tl = dy2 \* dxl; double t2 = dyl \* dx2;

if (fabs(tl - t2) < EPSILON) {

if (ax1 • ax2 < o 1 1 dy1 \* dy2 < o) {

if (dxl\*dxl + dyl\*dyl >= dx2\*dx2 + dy2\*dy2 - EPSILON) ( return CNEITHER;

} else { return CW;

l

else ·{ return CCW;

l

else if (tl > t2) { return CCW;

else (

**return CW;**

int main (void)

{

Tuesday August 19, 2014

**Point a, b, c; Orientation res;**

while (cin >> a.x >> a.y >> b.x >> b.y >> c.x >> c.y) res = ccw(a,b,cl ;

if (res == CW) {

**cout << 11 CW 11 < < endl;** else if (res == CCW) { **cout << 11 CCW"** <<" **endl ;**

else if (res == CNEITHER) { **cottt << 11 CNEITHER 11 << endl ;** else {

binsearch.cc, ccw.cc 14/77

Aug 19, 14 21:36 **circle\_3pts.cc**

*I\**

\* Parameters of circle from 3 points

* Author: Howard Cheng
* **Reference:**

<http://www.exaflop.org/docs/cgafaq/>

Page 1/1

\* This routine computes the parameters of a circle (center and radius)

* from 3 points. ·Returns non-zero if successful, zero if the three
* **points are colinear.** ·

*\* I*

#include <iostream>

#include <iomanip>

**#include <cmath>**

**#include <Cassert> using namespace std;**

*I\** how close to call equal \*/

canst double EPSILON = lE-8;

struct Point { double x, y;

) ;

int circle(Point pl, Point p2, Point p3, Point &center, double &r)

{

'

double a,b,c,d,e,f,g;

a = p2.x - pl.x;

b = p2.y - pl.y;

c = p3.x - pl.x;

d = p3.y - pl.y;

e = a\*(pl.x + p2.x) + b\*(pl.y + p2.y); f = c\*(pl.x + p3.x) + d\*(pl.y + p3.y);

g = 2.0\*(a\*(p3.y - p2.y) - b\*(p3.x - p2.x)); if (fabs(g) < EPSILON) {

**return O;**

}

center.x = (d\*e b\*f) / g; center.y = (a\*f - c\*e) *I* g;

r = sqrt( (pl.x-center.x)\*(pl.x-center.x) + (pl.y-center.y)\*(pl.y-center.y));

**return l;** '

int main (void)

{

**Point a, b, c, center;**

double r;

**while (cin >> a.x >> a.y >> b.x >> b.y >> c.x >> c.y)** {

if (circle(a, b, c, center, r)) { cout << fixed << setprecision(3);

**cout << "center** = { 11 << **center .x** << 11 11 < < **center . y** << 11

) "

**cout << '1 radius** = 11 << **r << endl;**

} else (

**cout << 11 colinear 11 << endl;**

<< **endl i**

} }

**return O ;**

*v* );1ay August 19, 2014

"'v.1.>J&.'?· *&/*

Aug 19, 14 21:36

**ccw.cc**

Page 2/2

printf("Help, I am in trouble!\n"); exit (1);

} }

**return O ;**

}

r :,

ccw.cc, circle\_3pts.cc *(*

.

'---./ '

-\ .,.,,,,

c\\ *(( (*

/ t

'>-

Printed by Dast

|  |  |
| --- | --- |
|  | Aug 19, 14 21:36 common\_subseq. cc Page 1/2 |
| /\*  \* Longest common subsequence  \*   * Author: Howard Cheng * **Reference:**   <http://www.ics.uci.edu/-eppstein/161/960229.html>  \* Given two arrays A and B with sizes n and m respectively, compute the   * length of the longest common subsequence. It also returns in s a longest * common subsequence (it may not be unique). One can specify which one * to choose when multiple longest common subsequences exist. * Running time and space requirement is O{mn).   *\* I*  **#include <iostream>**  #include <algorithm>  #include <Cassert>  **using namespace std;**  canst int MAX\_LEN = 20;  int LCS(int A(], int n, int B[J , int m, int s[])  {  int L [MAX LEN+l) [MAX LEN+l) ;  int i, j,-k; -  for (i= n; i *>=* o ;i-- ) {  for ( j = m; j *>=* 0 ; j"--) (  if (i == n 1 1 j == m) {  L [i] [j ] = 0 ;  else if (A[i] == B [j] ) {  L[i] [j ] = 1 + L[i+l] [j+l];  else {  L[i) [j) = max(L[i+l) [j), L(i] [j+l) );  *I\** the following is not needed if you -are not interested in the sequence \*/  k = O ;  i= j = O ;  while (i < n && j < m) if (A[i) == B [j ] ) { s [k++J = A (i] ;  **i++;**  **j ++ ;**  else if (L[i+ll (j] *>* L [i) [j+l])  **i++;**  else if (L[i+l][j] < L [i) [j+l)) j++;  else {  *I\** put tie-breaking conditions here \*/  *I\** eg. pick the one that starts at the first one the earliest \*/ j++;  }  return L[O) [OJ ;  int main(void)  {  int A[MAX LEN) , B[MAX LEN), s[MAX LEN) ;  **int m, n,-i, l;** - -  while (cin *>>* n *>>* m && 1 <= n && 1 <= m && |

Aug 19, 14 21:36

**common subseq.cc**

Page 2/2

for (i = O ;· i-< n; i++) · -

**cin >> A[i] ;**

n <= MAX LEN && m <= MAX LEN)

}

for (i = O; i < m; i++)

cin » B[i);

}

l = LCS(A, n, B, m, s);

for (i = O; i < i; i++)

**cout << s (i] << 11 11 ;**

**cout << end! << uLen** = " << **1 << endl;**

}

return O ;

Tuesday August 19, 2014

common\_subseq.cc 16/77

}

return CCW;

else {

(

( -

return dxl > dx2 I I (dxl == dx2 && dyl > dy2); return ccw(start\_p, a, bl·== CCW;

} }

CTS• , , , ay August 19, 2014



|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Aug 19, 14 21:36 | convex | hull.cc |  | Page 1/3 |  | Aug 19, 14 21:36 |  | **convex\_hull.cc** | Page 2/3 |
| *!•*   * Convex hull * Author: Howard Cheng * **Refererice:**   <http://wilma.cs.brown.edu/courses/cs0l6/packet/node25.html>   * **Given a list of n (n >= l} points in an array, it returns the vertices of** * the convex hull in counterclockwise order. Also returns the number of * **vertices in the convex hull . Assumes that the hull array has b'een** * allocated to store the right number of elements (n elements is safe). * The points in the original polygon will be re-ordered. * **Note: The hull contains a maximum number of points. ie. all colinear**   points and non-distinct points are included in the hull.  \*/  **#include <iostream>**  **#include <iomanip>**  #include <cmath> llinclude <algorithm>  **#include <Cassert>**  using namesace std;  *I\** how close to call equal •/ canst double EPSILON = lE-8;  struct Point { double x, y;  bool operator<(const Point &p) canst { return y < p.y 1 1 (y == p.y && x < p.x);  };  *I\** **counterclockwise, clockwise or undefined \*/**  enum orientation {ccw, cw, CNEITHER};  /\* Global point for computing convex hull \*/ Point start\_p, max\_p;  bool colinear(Point a, Point b, Point c)  {  double dxl = b.x - a.x; double dx2 = c.x - b.x  double dyl = b.y - a.y double dy2 = c.y - b.y double tl = dy2 \* dxl; double t2 = dyl \* dx2;  return fabs(tl - t2) < EPSILON;  Orientation ccw(Point a, Point b, Point c)  {  double dxl b.x - **a.x;** double dx2 c.x - b.x; double dyl = b.y - a.y; double dy2 c.y - b.y;  double tl = dy2 \* dxl;  double t2 = dyl \* dx2;  if (fabs(tl - t2) < EPSILON) (  if (dxl \* dx2 < O I I dyl \* dy2 < 0) {  if (dxl\*dxl + dyl\*dyl dx2\*dx2 + dy2\*dy2 - EPSILON) ( return CNEITHER;  } else ( | | | | |  | return CW;  ) else (  }  ) else if (tl > t2) { return CCW;  return ·cw;  bool **ccw\_cmp(const Point &a, canst P0int &b)**  return ccw(start\_p, a, b) == CCW;  }  bool sort cmp(const Point &a, const Point &b)  if (colinear(start\_p, a, max\_p) && colinear(start\_p, b, max double dxl = abs(start\_p.x - a.x);  double dx2 = abs(start\_p.x - b..x); double dyl = abs(start p.y - a.y); double dy2 = abs(start-p.y - b.y):  else (  irrt convex\_hull(Point polygon[], int n, Point hull[]) ( int count, best\_i, i;  sort(polygon, polygon+n);  for (int i = n-1; i >= l; i--) (  if (fabs(polygon[i] .x - polygon[ i-1] .x) < EPSILON && fabs(polygon( i] .y - polygon[i-1) .y) < EPSILON) (  for iint j = i; j < n-1; j++) · polygon( j] = polygon[ j+l];  **n--;**  assert (n > O);  if (n == 1) {  hull[O] = polygon[ O]; return 1;  /\* find the first point: min y, and then min x \*/ best\_i = min\_element( po'lygon, polygon+n) - polygon; swap(polygon[O] , polygon[best\_i) );  start\_p = polygon[ O) ;  /\* find the maximum angle wrt start\_j, and positive x-axis best i = l;  for Ti = 2; i < n; i++) {  if (ccw cmp(polygon[best i], polygon[i])) { best\_I = i; -  max\_p polygon[best\_i];  *I\** get simple closed polygon \*/ sort(polygon+l, polygon+n, sort\_cmp);  /\*.do convex hull \*/ | | | \_p) ) {  \*/ |

,,,,,;;'-·

' •'¥:':';"",·

&:

cqnvex\_hull.cc

*(*

*··t;:3*

· t -./:

l

'

,\

Cb.

- 1

,--

**count == O;**

hull (count] = polygon [count]-; count++; hull[count] = polygon( count] ; count++; for (i = 2; i < n; i++) {

while (count > 1 &&

ccw(hull[count-2], hull[count-1], polygon[ i]) CW) {

*I\** pop point \*/

**count--;**

)

hull[count++] polygon( i];

)

**return count;**

int main (void)

{

Point \*polygon, \*hull; int n; hull size;

**int i ;** -

while (cin >> n && n > 0) { polygon = new Point(n]; hull = new Point(n]; assert(polygon && hull); for (i = O; i < n; i++) {

cin >> polygon[i] .x >> polygon[i].y;

)

hullsize = convex\_hull(polygon, n, hull) ;

**caut << 11 Sorted: 11 << endl;**

for ( i = O ; i < n; i++) {

cout << fixed << setprecision(2);

**cout** << << **polygon [i] .x** <<

11 ( "

11 11

<< **polygon { i] .y** << << **enl ;**

11 11

)

}

**cout << endl;**

**cout << 11 Hull size =** << **hull size << endl;**

11

for (i = O; i < hull\_size; i++ {

)

**cout** << << **hull [i] . x** <<

" { 11

11 , 11

<< **hull (i] .y** << << **endl;**

" ) 11

cout << endl; delete[] polygon; delete[] hull;

**return O;**

*/*

*(( ,1*

*/* Printed by Dar sl

Aug 19, 14 21:36

convex **hull.cc** Page 3/3

Aug 19, 14 21:36

/\*

* Chinese Remainder Theorem

\*

* Author: Howard Cheng

\* Reference:

**era.cc**

**Page 1/2**

\* Geddes, K.O., Czapor, S.R., and Labahn, G.

Algorithms for Computer

Algebra, Kluwer Academic Publishers, 1992, p. 180

\*

\* Given n relatively prime modular in m(O], ..., m[n-1], and right-hand

* sides a[O], ..., a[n-1), the routine solves for the unique solution
* in the range o <= x < m[O]\*m[l]\* ...\*m[n-1] such that x = a(i] mod m(i]
* for all O <= i < n. The algorithm used is Garner' s algorithm, which is not the same as the one usually used in number theory textbooks.

\*

\* **It is assumed that m[i] are positive and pairwise relatively prime.**

\* a[i] can be any integer.

\*/

**#include <iostrearn>**

**#include <Cassert> using namespace std;**

int gcd(int a, int b, int &s,· int &t)

{

int r, rl, r2, al, a2, bl, b2 , q;

**int A = a;**

int B = b;

al = b2 = l; a2 = bl = O ;

while (b) {

assert(al\*A •a2\*B

q = a *I* b;

**r = a % b;**

**rl = al - q\*bl;**

r2 = a2 - a\*b2;

a = b; -

al = bl; a2 = b2; b = r; bl = rl;

***D2* = r2;**

a ) ;

**s = al ;**

**t = a2;**

assert(a >= O);

**return a;**

int cra(int n, int m[], int a[])

{

**int x, i, k, prod, temp; int \*gamma, \*v;**

gamma = new int(n]; v = new int[n]; assert(gamma && v);

*I\** compute inverses \*/ for (k = 1; k < n; k++) prod = m[O] % m(k];

for (i = l; i< k; i++) { prod = (prod \* m(i]) % m(k];

Tuesday August 19, 2014

)

gcd(pr.od, m[k], gamma(k], temp);

convex\_hull.cc, era.cc 18/77

Printed by Darcy Best



Aug 19, 14 21:36

. era.cc

Page 2/2

gamma [kl %= m [kl ; if (gamma[kl *<* O) {

gamma[kl += m[kl ;

*I\** compute coefficients *\* I*

v[Ol = a[Ol ;

for (k = 1; k < n; k++) temp = v[k-11 ;

for (i = k-2; i >= O; i--) {

temp = (temp \* m[il + v[il ) % m[kl ;

if (temp < O) {

temp += m [kl ;

}

v[k) = ((a[kl - temp) \* gamma [kl ) % m [kl ;

if (v[k) *<* 0) {

v[kl += m[kl ;

} }

*I\** **convert from mixed-radix representation \*/**

x = v[n-ll ;

for (k- = n-2; k >= O; k--)

X = X \* m [kl + V [kl ;

delete(] gamma; delete [l v;

**return x;**

int main (void)

(

**int n, \*m, \*a, i, x;**

while (cin >> n && n > 0) ( m = new int[nl ;

a = new int[n); assert (m && a);

**cout << 11 Enter modul i :** 11 << **endl ;**

for (i = O; i < n; i++) {

**cin >> m [i] ;**

}

**cout << 11 Enter right-hand side: 11 << endl;**

for (i = o ; i *<* n; i++) { cin >> a [i) ;

}

**x = cra(n, m, a} ;**

**cout** << = << x << **endl;**

11 x

11

for (i = O; i *<* n; i++) {

assert ((x-a [ill % m [i) == 0);

}

delete[) m;

delete [l a;

**return O;**

Aug 19, 14 21:36 **date.cc** Page 1/3

*II*

*II* Date class

*II*

*II* This is an implementation of some common functionalities for dates. *II* It can represent dates from Jan 1, 1753 to after (dates before that *II* time are complicated ...)

*II* ·

#include <iostream,

#include <string>

#include <utility,

#include <iomanip,

#include <cctype,

using namespace std;

**using namespace std: :rel\_ops;**

struct Date·

int yyyy; **int mm;** int dd;

*II* no dates before 1753

static int canst BASE\_YEAR = 1753;

*II* Enumerated type for names of the days of the week enum dayName {SUN,MON,TUE,WED,THU,FRI,SAT};

*ll* Is a date valid

s{tatic b. ool validDate(int yr, int man, int day)

return yr >= BASE YEAR && man >= 1 && man <= 12 && day > 0 && day = daysin(mon, yr);

bool isValid() canst

{

return validDate(yyyy, mm, dd);

)

*II* Constructor to create a specific date. If the date is invalid,

*II* the behaviour is undefined

Date(int yr = 1970, int man = 1, int day = 1)

{

YYYY = yr;

**mm = man;**

dd = day;

*II* Returns the day of the week for.this dayName dayOfWeek() canst

{

int a = (14 - mm) *I* 12; int y = yyyy - a;

**int m = mm + 12 \* a - 2;**

return (dayName)((dd + y + y/4 - y/100 + yl400 + 31 \* m / 12) % 7);

1. *I* comparison operators-

bool operator==( const Date &d) canst

{

return dd == d.dd && mm == d.mm && yyyy d.yyyy;

b{ool operator<(const Da.te &d) const

return yyyy < d.yyyy I I (yyyy == d.yyyy && mm < d.mm) I I

(yyyy == d.yyyy && mm == d.mm && dd < d.dd);

Tue. · ·,August 19, 2014

Cs1,

I

10.1,

;,1,'.\_c::::.\_....-·

', ..,.,;...,,.- .·

,

era.cc, date.cc

*C';:3:*

*';(::}i*



/..:,:::\

- , *f7 r*

. \ *(7"* ')

l

Printed by Darcy

|  |  |
| --- | --- |
| Aug 19, 14 21:36 date.cc | Page 2/3 |
| *II* Returns true if yr is a leap year static bool leapYear(int y)  {  return (y % 400 ==0 I I (y % 4·== O && y % 100 != O));  *JI* number of days in this month static int daysln(int m, int y)  {  switch (m)  **case 4**  **case 6**  **case 9**  case ll return 30;  **case 2** *:*  if (leapYear(y)) return 29;  }  else { return 28;  }  default return 31;  *JI* increment by day, month, or year  *II*  *J I* Use negative argument to decrement  *II*  *JI* If adding a month/year results in a date before BASE YEAR, the result  *II* is undefined. -  *II*  // If adding a month/year results in an invalid date (Feb 29 *on* a non-leap *II* **year, Feb 31, Jun 31, etc.), the results are automatically "rotmded down"** *II* to the last valid date  *II* add n days to the date: complexity is about n/30 iterations void addDay(int n = 1)  {  dd += n;  while (dd > daysin (mm.,yyyy)) dd -= daysin(mm,yyyy);  if (++mm > 12) (  **mm = l;**  **yyyy++;**  } }  while (dd < 1) {  if (--mm < 1) {  **mm = 12;**  yyyy--;  }  dd += daysin(mm,yyyy);  // add *n* months to the date: complexity is about n/12 iterations void addMonth(int n = 1)  {  **mm += n;**  while (mm > 12)  mm -= 12;  **yyyy++;** | |

Aug 19, 14 21:36

**date.cc**

Page 3/3

while (mm < 1)

**mm += 12;**

yyyy--;

if (dd > daysin(mm,yyyy)) dd = daysin(mm,yyyy);

*JI* add n years to the date void addYear(int n - 1)

{

**yyyy += n;**

if (!leapYear(yyyy) && mm

dd = 28;

2 && dd

29)

} }

*II* number of days since 1753/01/0l, including the current date int daysFromStart() canst

{

**int C = 0;**

Date d(BASE"YEAR, 1, 1);

Date d2(d);-

d2.addYear(l);

while (d2 < \*this) {

c += '.!:.2-PXe.a,.r( <!:..YY.YY) ? 366 365;

d = d2;

d2.addYear(l);

d2 = d;

.

d2.addMonth( l);

while (d2 < \*this) {

c += daysin(d.mm, d.yyyy); d = d2;

d2.addMonth(l);

}

while (d <= \*this) d.addDay *();*

**c++;**

**return c;**

)

} ;

*JI* Reads a date in yyyy/mm/dd format, assumes date is valid and in the

*JI* right format

istream& operator>>(istream &is, Date &d)

{

**char c;**

return is >> d.yyyy >> c >> d.mm >> c >> d.dd;

*II* print date in yyyy/mm/dd format

ostream& operator<< (ostream &as, canst Date &d) {

char t = os.fill('O');

os << d.yyyy << '/' << setw(2) << d.mm << '/' << setw(2) << d.dd; os.fill(t);

**return as;**

Tuesday August 19, 2014

date.cc *20177*

Printed by Darcy Best

Aug 19, 14 21:36 dijkstra. cc Page 1/3 Aug 19, 14 21:36 **dijkstra.cc** Page 2/3

/\*

* Dijkstra's Algorithm

\*

* Author: Howard Cheng
* **Reference:**

**Ian Parberry' s 11 Problems on Algorithms 11 , page 102.**

* **Given a weight matrix representing a graph and a source vertex, this**
* algorithm computes the shortest distance, as well as path, to each
* of the other vertices. The paths are represented by an inverted list,
* such that if v preceeds immediately before w in a path from the
* source to vertex w, then the path P(w] is v. The distances from

\* the source to v is given in D[v) (DISCONNECT if not connected).

\*

* Call get\_path to recover the path.
* Note: Dijkstra's algorithm only works if all weight edges are

**non-negative.**

\* /

if (best\_init) {

assert(D[w) != DISCONNECT);

assert(fringe[wj] == w);

/\* get rid of w from fringe \*/

**f size--;**

for (j - wj; j < f size; j++) fringe[j) = fringe[j+l);

*I\** update distances and add new vertices to fringe \*/ for (v = O; v < n; v++) {

if (v != src && graph (w] [v) != DISCONNECT) {

if (D [v) == DISCONNECT 1 1 D [w) + graph [w] (v] < D (v]) {

D (v] = D [w] + graph (w) [v) ; P(v] = w;

else if (D [w] + graph (w] [v] == D [v)) {

*I\** put tie-breaker here \*/

#include ciostream>

#include calgorithm>

**#include <cassert using namespace std;**

canst int MAX NODES = 10; canst int DISCONNECT = -1;

) }

)

}

if ( !used[v]) { used[v) = l; fringe[f size++) v;

-

}

/ \* assume that D and P have been allocated t/

void dijkstra(int graph(MAX\_NODES] [MAX\_NODES], int n, int src, int D[], int P[])

char used[MAX NODES); int fringe[MAX\_NODES); **int f\_size;**

**int v, w, j , wj;**

int best, best\_init;

**f size = O;**

for (V = 0; V < n; v++) {

if (graph[src] [v] != DISCONNECT && src != v) { D [v] = graph [src) [v] ;

D[src] = O;

int get\_path(int v, int P[], int path[])

{

int A[MAX NODES];

int i, k,-

k = O;

A[k++] = v;

while (P [v] != -1) {

V • P(v);

A[k++] = V;

}

P[v] = src;

fringe[f\_size++] = v; used[v] = l;

else {

D [v] = DISCONt-"'BCT; P[v] = -1;

used(v] = O;

)

for (i = k-1; path [k-1-i]

}

return k;

int main (void)

{

**int m, w, num;**

>= 9; i--) {

A[i];

D [src] = O;

P (src) = -1; used[src] = l;

best init = l; while (best\_init) {

/\* find unused vertex with smallest D \*/

best init = O;

for j = O; j < f\_size; j++) { v = fringe[j];

assert(D[v) != DISCONNECT);

if (!best\_init \ \ D[v] < best) best = D [v);

int i, j;·

int graph[MAX NODES] [MAX NODES] ;

int P[MAX NODES][MAX NODES], D[MAX NODES] [MAX NODES];

int path[MAX\_NODES] ;- -

*I\** clear graph \*/

for (i = O; i < MAX\_NODES; i++) { for (j = O; j < MAX NODES; j++)

graph[il [j] = DISCONNECT;

*I\** read graph \*/

**w ? v;**

wj = j;

**cin >> i > > j**

while (!( i ==

>> **w;**

-1 && j == -1) ) {

best\_init = 1;

Tur . '' August 19, 20·14

.,..,.,,.,./'

:

·".::::.-

'-

.dijkstra.cc ·

assert(O <= i && i < MAX\_NODES && 0 <=

-

&& j < MAX\_NODES);

*21.!*

*.* .

' ,

..,,•''



ct:. *C'.*

graph[i] [j] = graph[j][i] W;

**cin >> i >> j >> w;**

for (i = O; i < MAX\_NODES; i++) (

dijkstra(graph, MAX\_NODES, i, D[i]' P[i]);

/\* do queries \*/

**cin >> i** >> j;

while ·(!(i == -1 && j == -1)) (

assert(O <= i && i < MAX NODES && 0 <= j && j < MAX NODES);

**cout << i** << " " << **j** <<- 11 : << **D [i] [j] << endl ;** -

11

for (m = j ; m != -1; m = P[i] [ml) (

)

**cout** << " << **m;**

11

**cout << endl;**

num = get\_path(j, P[il , path);

**for (m = O; m < num; m++}**

cout << " " << path [ml ;

}

**cout << endl; cin** >> i >> **j;**

**return O ;**

Aug 19, 14 21:36

dijkstra. cc

Page 3/3

.

)

Aug 19, 14 21:36 **dijkstra\_spar e.c'?**

*I\**

\* Dijkstra' s Algorithm for sparse graphs

\*

\* Author .: Howard Cheng

\*

*(i* '1

Printed by Darcy ,

Page 1/3

* **Given a weight matrix representing a graph and a source vertex, this**
* algorithm computes the shortest distance, as well as path, to each
* of the other vertices. The paths are represented by an inverted list,
* such that if v preceeds immediately before w in a path from the

\* source to vertex w, then the path P[w] is v. The distances from

* the source to v is given in D(v] (-1 if not connected).

.

* Call get\_path to r.ecover the path.

•Note: Dijkstra's algorithm only works if all weight edges are

**non-negative.**

* This version works well if the graph is not dense. The complexity
* is O( (n + m) log (n + m)) where n is the number of vertices and

\* m is the number of edges.

\*

*\*I*

#include <iostream>

#include <algorithm>

#include <vector>

**#include <cassert>**

**#include <queue> using namespace std;**

struct Edge (

**int to;**

int weight; *II* can be double or other numeric type Edge(int t, int w)

: to(t), weight(w) ( )

);

typedef vector<Edge>: :iterator Edgeiter; struct Graph {

**vector<Edge> \*nbr;** int num nodes; Graph(int n)

num\_nodes (n)

nbr = new vector<Edge>[num\_nodes];

.-Graph()

(

delete[] **nbr;**

)

1. note: There is no check on duplicate edge, so it is possible to

*II* add multiple edges between two vertices

*II*

*II* If this is an undirected graph, be sure to add an edge both

*II* ways

void add edge(int u, int v, int weight)

. ( -

)nbr(u].push-back(Edge(v, weight));

Tuesday August 19, 2014

);

*I\** assume that D and P have been allocated *\*I*

void dijkstra( const Graph &G, int src, vector<int> &D, vector<int> &P)

(

dijkstra.cc, dijkstra\_ sparse.cc 22/77

Aug 19, 14 21:36 **dijkstra\_sparse.cc** Page 2/3

typedef pair<int,int> pii;

int n = G.num nodes; vector<bool> used{n, false);

priority\_queue<pii, vector<pii>, greater<pii> > fringe;

D.resize(n);

P.resize(n);

fill{D.begin(), D.end{), -1);

fill(P.begin(), P.end(), -1);

D[src] = O;

fringe .push(make\_\_pair (D [src], src));

while (!fringe.empty()) { pii next = fringe.top(); fringe.pop() ;

int u = next.second; if (used[u) ) continue;

,1sed[u) = true;

for (Edgeiter it = G.nbr[u) .begin(); it != G.nbr[u) .end(); ++it)

**int v = it->to;**

int weight = it->weight + next.first; if (used[v)) continue; ·

if (D [v) == -1 1 1 weight < D [v)) { D[v] = weight;

P[v] = u;

fringe .push(make pair (D [v), v));

)

-

Printed by Darcy Best

)

int get\_path(int v, canst vector<int> &P, vector<int> &path)

{

path.clear() ; path.push back(v); while (P[v] != -1)

v = P(v];

) path.push-back(v);

reverse (path.begin (), path. end·()); return path.size();

int main (void)

{

**int n;**

while (cin >> n && n > O ) (

Graph G(n);

**int u 1 v, w;**

**while ( cin >> '..l >> v >> w** && ! **(u**

G.add\_edge(u, v, w);

-1 && V -1 && w -1) ) {

**while (cin >> u >> v** && ! **(u**

**veCtor<int> D, P, path;** dijkstra (G, u, D, P); get\_path(v, P, path);

-1 && V -1) ) {

cout << "distance • " << D[v) << endl;

**cout << ''path = 11 ;**

for (unsigned int i •O; i < path.size(); i++) {

cout cc path(i] << • ·

)

cout cc endl;

T•· . 'ay August 19, 2014

G

c, "

/ I

*··· :.:::)*

*.r ;;/*

\_.

.....

dijkstra\_sparse.cc .6

*·*;*·*I*(.'-:./)*

Aug 19, 14 21:36

**dijkstra\_sparse.cc**

* Page 3/3

}

**return o ;**

.I - \

...*!*

'···

,.,

,,... ,



*I'* . ,

(L.

·:-"\

-

---- ""l .l

. /

-.

6/

Printed by Dare t

Aug 19, 14 21:36 dist30.cc Page 1/5

*II*

// 3-D distances between point to point, point to line segment ,

*II* line segment to line segment, and point to triangle.

*II*

*II* There are corresponding versions of the same code for distances *II* between point to infinite lines, infinite line to infinite line, *II* and point to infinite plane.

*II*

*II* It is assumed that segments/lines/triangles/plane are defined by

*II* distinct points (so the objects are not degenerate).

*II*

*II* They can be used for 2-D objects as well by setting the z coordinates

*II* to o. ·

*II*

*II* Author: Howard Cheng

*II*

#include <iostream>

**#include <iomaniP>**

#include <Cmath>

#include <algorithm>

#include <Cassert>

using namespace std;

const double PI = acos(-1.0); struct Vector {

double x, *y ,* z;

Vector(double xx = 0, double *yy* = 0, double zz = O)

z

: x(xx), y(yy), (zz) { }

Vector(const Vector &pl, const Vector &p2)

: x(p2.x - pl.x), y(p2.y - pl.y) , z{p2.z - pl.z) { }

Vector(const Vector &pl, const Vector &p2, double t)

: x(pl.x + t\*p2.x), y(pl.y + t\*p2.y), z\_ (pl.z + t\*p2.z) { }

double norm() const {

return sqrt(x\*x + *y\* y* + z•z);

} ;

**istream &operator>>(istream &is, Vector &p)**

{

**return is >> p.x >> p.y > > p.z;**

Aug 19, 14 21:36 · **dist30.cc** Page 2/5

double dist point to noint(const Vector'&pl, const Vector &p2)

{ - - \_;c

Vector p(pl, p2); return p.norm{);

//·angle between two vectors (in radians) double angle(const Vector &pl, const Vector &p2)

{

return acos(dot(pl, p2)/pl.norm(J /p2.norm());

*II* distance from p to the line segment from a to b

double dist\_point\_to\_segment( const Vector &p, const Vector &a,

const Vector &b)

Vector u(a, p), v(a, b).;

double s = dot(u,v) *I* dot(v,v);

if Is < o 1 1 s > 1> {

} return min(dist-point-to\_point(p, a), dist\_point-to\_point(p, b));

Vector proj (a, v, s);

return dist\_point\_to\_point( proj, p);

*II* distance from p to the infinite line defined by a and b double dist\_point\_to\_line{ const Vector &p, const Vector &a,

const Vector &b)

Vector u(a, p), v{a, b);

double s = dot(u,v) *I* dot(v,v); Vector proj (a, v, s);

return dist\_point\_to\_point( proj, p);

*II* distance from p to the triangle defined by a, b, c

double dist\_point\_to\_triangle{ const Vector &p, const Vector &a,

const Vector &b, const Vector &c)

**Vector u{a, p), vl(a, b), v2(a, c) *i***

Vector normal = cross(vl, v2);

double s = dot(u, normal) *I* (normal.norm() \* normal.norm()); Vector proj (p, normal, -s);

*II* check projection: inside if sum of angles is 2\*pi Vector wa(proj, a), wb(proj, bl , wc(proj, c); double al = angle(wa, wb);

double a2 = angle(wa, we);

**ostream**

{

**&operator<<(ostream &os, canst Vector &p)**

double a3 = angle(wb, we);

if (fabs(al + a2 + a3 - 2\*PI) < le-8) (

**return as** << 11 ( 11 << **p . x** << It << **p** • y << 11 H << **p . Z** << 11 ) II ;

)

double dot(const Vector &pl, const Vector &p2)

{

return pl.x \* p2.x + pl.y \* p2.y + pl.z \* p2.Z;

Vector cross(const Vector &pl, · const Vector &p2)

{

Vector v{pl.y\*p2.z - p2.y\*pl.z, p2.x\*pl. z - pl.x\*p2.z, pl.X\*p2.y - p2.X\*pl.y);

**return v;**

*II* distance between two points

Tuesday August 19, 2014

return dist\_point\_to\_point(proj, p); else {

return min(dist\_point\_to\_segment( p, a, b), min(dist\_point\_to\_segment( p, a, c),

dist point\_to\_segment( p, b, c)));

} )

*II* distance from p to the infinite plane defined by a, b, c double dist\_point\_to\_plane( const Vector &p, canst Vector &a,

const Vector &b, const Vector &c)

Vector u(a, p) , vl(a, b), v2(a, c);

**Vector normal = cross(vl, v2);**

double s = dot(u, normal) *I* (normal.norm() • normal.norm()); Vector proj (p, normal, -s);

return distpoint\_to\_point( proj, p);

dist3D.cc 24/77

Aug 19, 14 21:36 dist3D.cc Page 3/5

*II* distance from segment pl->ql to p2->q2

double dist segment to segment(const Vector &pl, canst Vector &ql,

- - - canst Vector &p2, canst Vector &q2)

*II*

*II* the. points on the 1st line are pl + t \* vl *II* the points on the 2nd line are p2 + s \* v2 *II*

*II* o <= s, t <= 1

*I I*

*II* squared distanc, is

*II*

*II* S = (pl.x - p2.x + t • vl.x - s • v2.x)A2 + *II* (pl.y -·p2.y + t \* vl.y - s \* v2.y)A2 + *II* (pl.z - p2.z + t \* vl.z - s \* v2.z)A2 *II*

*II* deriviative wrt t and s are:

*II*

*JI* ll2 dSldt = norm(vl)A2 \* t - dot(vl, v2) \* s + dot(vl, pl) - dot(vl, p2) *JI* 1/2 dS/ds = -dot(vl, v2) \* t + norm(v2)A2 \* s - dot(v2, pl) + dot(v2, p2) *II*

*J I* solving for s and t with both derivatives = 0:

*II*

Vector vl(pl, ql), v2(p2, q2);

Vector rhs(dot(vl, p2) - dot(vl, pl), dot(v2, pl) - dot(v2, p2)); double det = vl.norm ()\*vl .norm ()\*v2.norm ()•v2 .norm () -

dot(vl, v2)\*dot(vl, v2);

if (det < le-8) {

*JI* parallel lines (if vl and v2 != O) goto degenerate;

else (

double t = (rhs·.x•v2.norm()\*v2.norm() + rhs.y \* dot(vl, v2)) *I* det; double s = (vl.norm()\*vl.norm()\*rhs.y + dot(vl, v2) \* rhs.x) *I* det;

if (0 <= s && s <= 1 && 0 <= t && t <= 1) Vector ppl (pl, vl, t) , pp2 (p2, v2, s) ; return dist\_point\_to\_point (ppl, pp2);

degenerate:

return min(min(dist point\_to\_segment( pl, p2, q2),

dist poi t\_to\_segment(ql, p2, q2)), min(dist point\_to\_segment( p2, pl, ql),

dist point\_to\_segment(q2, pl, \_ ql)));

*II* distance from infinite lines defined by pl->ql and p2->q2 double dist line\_to\_line(const Vector &pl, canst Vector &ql,

canst Vector &p2, canst Vector &q2)

*II*

*II* the points on the 1st line are pl + t \* vl *JI* the points on the 2nd line are p2 + s \* v2 *II* .

*II* O <= S, t <= 1

*II*

*II* squared distance is

*II*

*II* S = (pl.x - p2.x + t \* vl.x - s •v2.x)A2 + *II* (pl.y - p2.y + t \* vl.y - s \* v2.y) 2 + *II* (pl.z - p2.z + t \* vl.z - s \* v2.z) 2 *II*

*II* deriviative wrt t and s are:

*II*

*II* ll2 dSldt = norm(vl)A2 \* t - dot(vl, v2) \* s + dot(vl, pl) - dot(vl, p2)

Printed by Darcy Best

Aug 19, 14 21:36 **dist3D.cc** Page 4/5

*II ll2* dSlds = -cfot(vl, v2) \* t + norm(v2)A2 \* s - dot(v2, pi) + dot(v2, p2)

*II*

*II* solving for s and t with both derivatives = 0:

*II*

Vector vl {pi,\_ ql), v2 (p2, q2);

Vector rhs(dot(vl, p2) - dot(vl, pl), dot(v2, pl) - dot(v2, p2)); double det vl.norm ()\*vl.norm() \*v2.norm()\*v2.norm() -

=

dot(vl, v2)\*dot(vl, v2);

if (det < le-8) {

*JI* parallel lines (if vl and v2 != O.)

return dist point to line(pl, p2, q2); else ( - -

double t = (rhs.x\*v2.norm()\*v2.norm() + rhs.y \* dot(vl, v2)) *I* det; double s = (vl.norm()•vl.norm() •rhs.y + dot(vl, v2) \* rhs.x) *I* det;

Vector ppl(pl, vl, t), pp2(p2, v2, s); return dist\_point\_to point(ppl, pp2);

*IJIIIIIIIIIIIJ llllllllllllllllll/lllllllllllllll/llllllllllilllllllllllll II*

*II* This is the solution' to 11836 (Star War)

*II*

*IIIIIIIIIIIIIIIIIIIIJ IIIIIIIIIIIIIIIIIJ IJIIIIIJIIJIIIJIII/IIIIIJIIII/J! II*

void do case()

-

(

Vector tl [4), t2 [4);

for (int i = O; i < 4; i++) cin >> t1 [i] ;

}

for (int i = O ; i < 4; i++) ( cin >> t2 [i);

double best ? dist point\_to\_point(tl[ O], t2[0));

*II* vertex-face distance

for (int il = 0 ; i1 < 4; il++) {

for (int jl = O; jl < 4; jl++) {

best = min (best, dist point to triangle (tl [ill, t2 [jll,. t2 [(j1+1)%4),

- - - t2( (j1+2)%4] ));

best = min(best, dist\_point\_to\_triangle(t2[il), tl(jl.), tl[ (jl+l)%4),

tl [(jl+2)%4)));

} }

*II* edge-edge distance

for (int il = O; il < 4 ; il++) {

for (int i2 = il+l; i2 < 4; i2++l ( for (int jl = O ; jl < 4 ; jl++) {

for (int j2 = jl+l; j2 < 4; j2++) {

best = min(best, dist segment to segment(tl[il), tl[i2),

- - - t2 [j 1), t2 [j 2) ) ) ;

} }

}

**cout << setprecision(2) < < fixed << best << endl;**

int main(void)

(

int T;

**cin >> T;**

while (T-- > O) {

Tut . ' August 19, 2014

*r( , ,,*

-·

dist3D.cc

'. :

'

*25,S."'/ )*

*Q)*



c[;'i

*i L./'*

,,-- I

\

*-'-'-':\*

Printed by Darcy .I

do\_case ();

* **return O ;**

)

.--

Aug 19, 14 21-:36

*/*

dist30.cc

Page 5/5

Tuesday August 19, 2014 dist30.cc, dist\_line.cc 26/77

/\*

* Distance from a point to a line.
* Author: Howard Cheng

\* Reference:

<http://www.exaflop.org/docs/cgafaq/cgal.html>

\*

* This routine computes the shortest distance from a point to a line.
* ie. distance from point to its orthogonal projection onto the line.
* Works even if the projection is not on the line.

\*/

#include <iostream>

**#include <iomanip>**

#include <cmath>

#include <cassert>

**using namespace std;**

struct Point { double x, y;

}; .

/\* **computes the distance from to the line defined by and 11 b 11 •** \*/

d{ouble dist-line(Point a, Point b, Point c) double L2, s;

**11 c 11**

**11 a 11**

L2 = (b.x-a.x)\*(b.x-a.x)+(b.y-a.y)\*(b.y-a.y); assert(L2 > 0);

s = ((a.y-c.y)\*(b.x-a.x)-(a.x-c.x)\*( b.y-a.y)) *I* L2;

retnrn fabs(s\*sqrt(L2));

int main(void)

{

**Point a, b, c;**

while (cin >> a.x >> a.y >> b.x >> b.y >> c.x >> c.y) {

cout << "distance = " << fixed << setprecision(2) << dist\_line(a, b, c)

<< **endl ;**

**return O ;**

Aug 19, 14 21:36

**dist line.cc**

Page 1/1

Printed by Darcy Best

Aug 19, 14 21:36

**eulerphi.cc**

Page 1/2

/•

\* **Euler' s Phi function:**

\*

* Author: Ethan Kim
* Complexity: O(sqrt(n))

\* Computes Euler's Phi(Totient) function; Given a positive n, computes

\* the number of positive integers that are <= n and relatively prime to n.

\*

\* For prime n, it is easy to see that phi(n)=n-1.

\* For powers of prime, phi(pAk)=pA( k-1) \* {p-1).

\* Also, phi is multiplicative,

* relatively prime.

\*/

#include <iostream>

**#include <Cassert>**

so phi(pq)=phi{p)\*phi{q),

if p and q are

using namespace std;

i{nt fast-exp(int b, int n) int res = l;

int X = b;

while {n > 0) (

if {n & OxOl)

**n--;**

**res \*= x;**

else { **n** >= **1; X \*= X;**

**return res;**

int phi{int n) **int k, res;** long long p;

assert (n > 0) ;

**res=l;**

for (k = O ; n % 2 == o ; k++) {

n *I=* 2,

)

if (kl

res \*= fast\_exp(2, k-1);

for (p = 3; p•p <= n; p += 2) {

for {k = O ; n % p == O ; k++) {

n /= p;

}

if (kl {

res \*= fast\_exp(p, k-1) • (p-1);

}

if {n > l) {

**res \*= n-1;**

}

**return res;**

int main {void) int p;

while{cin >> p && pl

Aug 19, 14 21:36

euclid.cc

Page 1/1

/•

* Euclidean Algorithm

\*

* Author: Howard Cheng

\*

* **Given two integers,**

\*

*\*I*

**#include <iostream**

**#include <Cassert>**

return their gcd.

**using namespace std;**

int gcd (int a,

{

**int r;**

int bl

/\* **unnecessary i f a, b >= O** \*/

if (a < O ) (

**a = -a;**

)

if (b < 0) {

b = -b;

)

while .(b) { r = a % b; a = b;

b = r;

)

assert (a >= a) ;

**return a;**

}

int main (void)

{

int a, b;

while **( cin > > a >> b)** {

cout << gcd(a, b)

}

<< endl;

return O ;

)

l)' -lay August 19, 2014

*Q \*

*+-,.,>'IJ!:!!,-*

-: -:::>·

.

euclid.cc, eulerphi.cc

*.&:3*

*1 --1*

*'----! /*

' l



*r'·.*

· ;

-\

*6)* - '1

Printed by Dar;'

Aug 19, 14 21:36

eulerphi.cc

Page 2/2

cout << phi(p) << endl;

return O;

Aug 19, 14 21:36 **eulertour.cc** Page 1/4

/\*

* Finding an Eulerian Tour
* Author: Howard Cheng
* The routine eulerian () d.etermines if a graph has an Eulerian tour.
* That is, it checks that it' is connected and all vertices have even
* degree. We assume that the graph is represented as an adjacency matrix
* **and the an auxillary array called 11 deg 11 gives the degree of the vertex.**
* The routine eulerian tour() returns one (arbitrary) Eulerian tour.
* The tour is stored in an array of the vertices visited in the tour,
* and the first and last vertex is the same.
* WARNING: eulerian tour() destroys the graph as it uses edges. If
* you need-the graph back then you should save a copy.

\*

\* NOTE: converting this code for directed graphs should not be that much

work. You should also be able to convert this code for Eulerian paths.

\*

\*/

**#include <iostream>**

#include <algorithm>

#include <Cassert> using namespace std;

**const int NUM VERTICES = SO;**

const int NUM=EDGES = 1000; /\* maximum number of edges in graph \*/

int graph [NUM VERTICES+l] [NUM VERTICES+l] ;

int deg[NUM\_VERTICES+l]; -

void clear graph(void)

-

{

fill(deg, deg+NUM VERTICES+l, O);

for (int i = 1; i-<= NUM VERTICES; i++) {

fill(graph(i], graph[i +NUM VERTICES+l, 0);

-

}

void visit(int.v, char visited(])

{

**int w;**

visited[v] = l;

for (w = l; w <= NUM VERTICES; w++) {

if (!visited[w] &&-graph[v] [w] > 0) visit(w, visited);

} )

int connected(void)

{

char visited[NUM VERTICES+l) ;

int i; -

fill(visited, visited,-NUM VERTICES+l, 0); for (i = l; i <= NUM\_VERTICES; i++) (

if (deg [i) > 0) {

visit(i, visited);

**break;**

)

for (i = 1; i <= NUM VERTICES; i++)

if (deg(i] > O && Tvisited(i]) {

Tuesday August 19, 2014 eulerphi.cc, eulertour.cc 28/77

Aug 19, 14 21:36 eulertour.cc Page 2/4

return o;

return l;

int eulerian(void)

{

Printed by Darcy Best

**int i;**

for (i = l; i <= NUM VERTICES; i++) (

if (deg[i] % 2 == 1) (

return O ;

}

return connected();

i{nt find-tour(int start, int temp(]) int len = O;

**int next;**

temp[len++] = start; while (deg[start) > 0) (

for (next = l; next <= NUM VERTICES; next++) { if (graph[start] (next] >-0) {

**break;**

} }

temp[len++J = next;

graph [start] [next] --; deg [start]-- ;

graph(next] [start]--; deg(next]--;

**start = next;**

}

**return len;**

i{ nt graft-tour(int old[], int old-len, int tour[], int tour-len) int pos[NUM\_VERTICES+l);

int i, j , pl, p2;

fill(pos, pos+NUM\_VERTICES+l, -1);

for (i = O ; i < old\_len; i++) { pos[old(i]] = i;

}

for (i = O ; 1. < tour\_len; 1.++) {

if (pos(tour[ i]J >= O ) (

break;

} )

assert(i < tour len); pl = pos[tour[i J ;

p2 = i;

f'or (i = old len-1; i > pl; i--) {

old[i+tour=len-1) = old[i];

)

for (i = p2+1, j = O; i < tour\_len-1; old(pl+j+l) = tour[i];

**i++,** j++) {

}

for ( i = o ; i <= p2; i++) {

old (pl+j+l]

)

= tour (i);

} return old-len+tour-len-1;

i{nt eulerian-tour(int tour(])

Tue, · ' August 19, 2014

..::);

eulertour.cc

.

*29P*

Aug 19, 14 21:36

int temp[NUM\_EDGES+l] ;

**int tour\_len, temp\_len, first\_time;**

int i, found;

**eulertour.cc**

Page 3/4

tour len = temp len = O;

first\_time = 1;-

while (1) {

found = O;

if (first time)

for (i 1; i <= NUM VERTICES; i++) {

if (deg[i] > 0) { -

found = l;

**break;**

} }

} else {

/\* **this ensures that we can graft next tour on to existing**

for (i = O; i < tour\_len; i++) {

if (deg (tour(il J > O ) {

found = l;

break;

)

i = tour(i);

**one**

**\*/**

}

if ( !found) break;

if (first\_time) (

tour len = find tour(i, tour);

else-{

-

}

temp len = find tour(i, temp);

tour=len = graft\_tour(tour, tour\_len, temp, temp\_len);

) **first**-**time** O;

**return tour\_len;**

int main(void)

{

int T, N, i, j , k;

**int u, v;**

int tour(NUM\_EDGES+l), tour\_len;

**cin >> T;**

for (i = l; i <= T; i++) ( clear\_graph ();

if ( i > 1) (

cout << endl;

}

**cout << "Case #" << i << endl; cin > > N;**

for (j = 0 ; j < N; j ++) {

**cin >> u >> v;** graph(u] [vl ++; graph[v] [ul ++; deg[u]++;

deg(v)++;

if (eulerian ()) (

tour len = eulerian tour(tour); for k = O; k < tour\_len-1; k++) {

}

cout << tour(k] << " " << tour[k+l] << endl;

else {

*,··'<,*

*-/),*

··'.'•1--J...i;,-';(.·'

·-,,·:..·

! *(J)*



,,..- .........

*I* '

... ·

,,,,--.,

-' )

*/ ,'*

*,{* ---)

*/*

*r7* --..,)

.. ··

Printed by Darcy Bst

Aug 19, 14 21:36

eulertour.cc

Page 4/4

**cout << 11 some beads may .be .1ost 11 << endl ;**

}

**return O;**

Aug 19, 14 21:36

**exp.cc**

Page 1/1

*I\**

Fast Exponentiation

* Author: Howard Cheng
* Given b and n, computes bAn quickly.

\*/

#include <iostream> using namespace std;

int fast exp(int b, int n)

{

-

int X = b;

while (n > O) {

if (n & OxOl)

**n--;**

**res \*= x;** else ( **n** >>= **1;**

**X \*= X;**

**int res = l;**

**return res;**

int main(void)

{

**int b, n;**

while (cin >> b >> n)

}

**cout << b** << << **n** <<

11 "' 11

11

" << fast\_exp(b, nl << endl;

return O;

Tuesday August 19, 2014 eulertour.cc, exp.cc 30/77

,'.·\_ \_ ----...

' - *­*, ;

£"/

*)/*

*.,:*6v *)*

*i*

-' ---

|  |  |  |
| --- | --- | --- |
|  | Aug 19, 14 21:36 **factor.cc** | Page 1/2 |
| *I\**   * Prime Factorization   \*   * Author: Ethan Kim * Complexity: O(sqrt(n)) * Takes an integer and writes out the prime factorization in * ascending order. Prints -1 first, when given a negative integer. * Note: you can change this code to store the factors in an array or process * the factors in other ways.   \*   * Also, this code works for all integers even on INT\_MIN (note that negating * INT\_MIN does nothing, but it still works because INT\_MIN is a power of 2).   *\* I*  #include <iostream> using namespace std;  void factor(int n) int printed = O; long long p;  .if (n == o 1 1 n == 1)  **cout << n << endl;**  **return;**  lf (n < o) ! )  Qn \*- -1·  **1 11**  **co:1 -111 << endl;**  **<<**  printed = 1; .  while (n % 2 == 0) (  n/=2;  **cout** << 11 211 << **endl;**  printed = l·  ' | |
|  | for (p = 3; p\*p <= n; p += 2) { while (n % p == O) {  n /= p;  cout << p << endl; printed = 1;  } )  if (n>l 1 1 !printed)  **cout << n << endl;**  int main(void) int p;  while(cin >> p && p != 0) { factor(p) ;  }  return O ; | |
|  | |

/ *'* , V \_/

*I* Printed by D, ,Best

Aug 19, 14 21:36

factor.cc

Page 2/2

Tuesday August 19., 2014

factor.cc

32177

Aug 19, 14 21:36 exteuclid.cc

Aug 19, 14 21:36

**expmod.cc**

Page 1/1

*I\**

\* Fast Exponentiation mod m

* Author: Howard Cheng
* **Given b, n, and m, computes bAn mod m quickly.**

\*/

**#include <iostream**

**#include ccassert>**

using namespace std;

i{nt fast-exp(int b, int n, int m)

**int res = 1;**

long long x = b;

while (n > O) { if (n & OxOl)

**n--;**

**res = (res \* x) % m;**

else {

**n** >>= **1;**

**x = {x \* x) % m;**

**return res;**

int main(void)

{

**int b, n, m;**

while (cin >> b >> n >> m)

**cout c< b** << << **n** <<

'A 11

**mod " << m << n**

<< endl;

" << fast\_exp(b, n, m)

)

**return O ;**

Printed by Darcy Best

Page 1/1

*I\**

\* Extended Euclidean Algorithm

* Author: Howard Cheng
* **Given two integers, return their gcd and the cofactors to form the**
* **gcd as a linear combination.**

\* a\*s + b\*t = gcd(a,b)

\*/

#include <iostream>

#include <Cassert>

**using namespace std;**

int gcd(int a, int b, int &s, int &t)

{

.

int r, rl, r2, al, a2, bl, b2, q;

**int A = a;**

int B = b;

*I\** unnecessary if a, b >= O

if (a < O ) {

r = gcd(-a, b, s, t);

**s \*= -1; return r;**

\*/

}

if (b < 0) {

r = gcd(a, -b, s, t);

t \*= -1;

**return r;**

al = b2 = l;

a2 = bl = o ;

while (b) { assert(al\*A + a2\*B q = a *I* b:

r = a % b;

rl = al - q\*bl; r2 = a2 - q\*b2;

a ) ;

al = bl; a2 = b2; b = -r;

bl = rl; b2 = r2;

a = b;

s = al;

t = a2; assert(a >= O) ; **return a;**

int main(void)

{

**int a, b, s, t, res;**

while (cin >> a >> b) { res = gcd(a, b, s, t);

**cout cc res cc** " = " **cc a**

<< **b** << u \* II **<C t << endl;**

**cc**

" \*

" cc **s cc** " + •

1

)

**return O ;**

,1,dayAugust 19, 2014

,,

-4),

"-!'!:

expmod.cc, exteucl"id.cc */1):*

*/* **'** *t***7***j*





Aug 19, 14 21:36 factor **large.cc** Page 1/4

*IIII* Gives the prime factorization of natural numbers (Uses probability)

*II* Author: Darcy Best

*II* Date January 7, 2010

*II*

*JI* This should be used for factoring large integers. If you're *II* dealing with are small integers (N < 2A31), this is going *JI* overboard. -- The normal sieve of Sieve of Eratosthenes is *JI* usually good even for values up to 2A40.

*II*

*II* This impiementation should only be used if you have numbers

*II* larger than 2A40 (10A12) to factor.

*II*

*JI* Notes:

*II* - You need to handle N < 2 separately.

*JI* - Uses Miller-Rabin Primality Test

// - This is a probabilistic test, there is a (1/4)AK

*II* probability that a composite will return prime.

/ / (K = l.O or 15 should be reasonably reliable). *JI* - Uses Pollard's Rho algorithm to factor composites. *II* - Ihave also added Brent's improvement

*JI* - This program writes a number as a product of its primes,

***I I* wi th normal exponents ( ie . 11 6 0 = 2""2 \* 3 \* 5 11 )**

**#include <iostream>**

#include <aigorithm>

**#include <Set>**

#include <map>

#include <cmath>

**#include <ctime>**

#include <cstdlib>

#include <vector> using namespace std;

typedef long long int 11; canst 11 MAX NUM = lel6;

canst 11 CB RT = ll(pow(l.O\*MAX NUM,1.0/3)) + 2;

**vector< ll > Primes;** -

**11 numPrimes;**

11 C = 2;

const 11 K = 10;

**set<ll> lgPrimes;**

**map<ll,11> semiPrimes i**

11 gcd(ll a,11 b) {

11 r;

while (b) {

Pirinted by Darcy Best

r a % b;

a b;

b = r;

**return a;**

11 mult mod(ll x,11 y,11 n) { 11 res = 0;

while(y){

if (y %- 2) (

y--;

**res += x;**

if (res >= n)

**res -= n;**

else { **X** <<= **l;** *y* >>= 1;

if (x >= n)

**X -= n;**

,;·August 19, 2014

'--..\_../ *j!*

--c.·..-..;,J;,

,

Aug 19, 14 21:36

**factor large.cc**

Page 2/4

} )

**return res;**

11 fast\_exp\_mod(ll b, 11 n,11 m) { 11 res = l;

11 X = b;

while (n) {

if (n % 2)

**n--;**

res = mult mod(res,x,m); else { -

**n** >>= **l;**

x = mult\_mod(x,x,m);

**return res;**

void genSmallPrimes() { bool isPrime(CB RT+?); for(int i=3;i<CB RT;i+=2l

isPrime(i] = true;

primes.clear(); primes.push\_back(2)' ;

**int i ;**

for(i=3;i\*i<CB RT;i+=2) if (isPrime(i])(

primes.push back(i);

for(int j=i\*i;j<CB RT;j+=(2\*i)l isPrime(j] = false;

for(;i<CB RT;i+=2)

if(isPrime[i)) primes.push back(i);

numPrimes = primes.size();

11 F(ll x,11 n) {

x = mult\_mod(x,x,n);

**X -= C;**

**return (x < O ? x + n** x) ;

11 pollardRho(ll n) { 11 b,g,x,y,z;

newC:

**c++;**

q = b = X = l;

;;,hile(g == 1) {

**z = l;**

*y* = X;

x = F(x,n);

z = mult\_mod(z,abs(x y),n);

g = gcd(z,n);

**b** <<= **l;**

for(ll i=O;i<b;i++) {

)

if (g == n 1 1 g

goto newC;

C = 2;

0) .

"•.:;.J

factor\_large.cc

:0:3, 33/)'\ :-c.

( *!*

....\_.,.,,.....

cc,r--y--.

/,--.\_,\_

,\



*(( \*

c\

Printed by Darcy ·;

);

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Aug 19, 14 21:36 | factor | large.cc | Page 3/4 |  | Aug 19, 14 21:36 | **factor large.cc** | Page 4/4 |
| **return g;**  bool miller(ll n) { 11 d = n-1;  11 s = O,a,x; while(d % 2 == O) {  **d** >>== l;  s++;  )  for(int i=O;i<K;i++) {  a = (rand() % (n-2)) + 2; *II* (2,n-1] ;  x = fast\_ex\_mod(a,d,n); if(x == 1 I I x == n-1)  **continue;**  for(ll r=l;r<s;r++) { x = mult mod(x,x,n);  if(x == 1)  **return false;**  if (x == n-1)  **goto nextK;**  )  **return false;**  nextK: ;  )  return true;  void printEntry(bool& printed,11 prime,int ex){  if ( !printed) printed = true;  else  **cout** << 11 \* 11 ;  **cout << prime;**  if(ex > 1)  **cout** << 11 ;,.. 11 << **ex;**  void factor(ll x) { **cout << x << 11** = •• ; bool printed = false;  for(int i=O;i<numPrimes;i++) if(x % primes(i] == 0) {  int ex = O; // Exponent do{  ex++;  )while(x % piimes( i] == 0); printEntry(printed,primes( i] ,ex);  )  if (x == 1) { **cout << endl; return;**  *II* lgPrimes and semiPrimes are useful if there  *II* in the test data if(lgPrimes.find(x) != lgPrimes.end()) {  printEntry(printed,x,l);  **cout << endl; return;**  if(semiPrimes.find(x) != semiPrimes.end()) (  11 lgFac = semiPrimes(x); printEntry(printed,x/lgFac,l); printEntry(printed,lgFac,l); | | |  |  | **cout << endl;**  **return;**  ) .  if(miller(x)){ *II* if x is prime printEntry(printed,x,l  **cout << endl;** lgPrimes.insert( x); **return;**  *II* Pollard's Rho does not work well with squares,  *II* so we'll check for  11 sqrtX = ll{sqrt(x) + 0.1);  printEntry(printed,sqrtX,2);  **cout << endl;**  11 smFac = pollardRho( x); if{lgFac < smFac)  swap(smFac,lgFac); printEntry{printed,smFac,l); printEntry{printed,lgFac,l); cout << endl;  semiPrimes[x) = lgFac;  int main() { genSmallPrimes{ );  srand ((unsigned int) time (NULL) ) ;  11 T,N;  **cin >> T;**  while (T--) { **cin >> N;** factor(N);  **return O ;** | |  |

if(sqrtX\*sqrtX == x){

it manually.

return;

11 lgFac = x/smFac;

x *I=* primes(i];

*I I* is a lot repetition of large primes/semi-primes

Tuesday August 19, 2014

factor\_large. cc 34/77

Aug 19, 14 21:36 fenwicktree.cc Page 1/3

/\*

* Fenwick Tree
* Author: Howard Cheng
* **Reference:**

**Fenwick, P.M. 11 A. New Data Structure for Cumulative Frequency Tables. 11**

Software---Practice and Experience, 24(3), 327-336 (March 1994).

* This code has been tested on UVa 11525 and 11610.

Printed by Darcy Best

Aug 19, 14 21:36 **fenwicktree.cc** Page 2/3

*II* decrement) . All affected cumulative sums are updated. void incEntry(int idx, int val)

{

assert(O <= ·idx && idx < N);

if (idx == 0) { tree[idx] += val; else {

do {

tree[idx] += val; idx += idx & (-idx);

* Fenwick trees are data structures that allows the maintainence of
* cumulative sum tables dynamically. The following operations
* are supported: ·

while (idx <

) )

(int)tree. size());

* + Initialize the tree from a list of N integers:

\*

* + Read the ·cumulative sum at index k < N:
  + Read the entry -3.t index o <= k < N,
  + Increment/decrement an entry at index O <= k < N in the list:
  + Given a value, find an index such that the cumulative sum at that position is the value:

0 (N log N) O (log k) 0 (log N)

0 (log N)

0 (log N)

*II* return the cumulative sum val[O] + val[l) + ... + val[idx) int cumulativeSum(int idx) canst

{

assert(O <= idx && idx < (int)tree.size());

int sum = tree[O] ; while (idx > 0) {

sum += tree(idx); idx &= idx-1;

)

**return sum;**

\*

* **The space usage is at most** 2\*N **for N input entries.**
* NOTE, it is assumed that all entries are non-negative (even after a

**decrement operation).**

*\* I*

**#include <vector>**

**#inciude <cassert>**

**using namespace std; class FenwickTree**

{

public:

FenwickTree(int n 0)

: N (n), tree (n)

iBM = l;

while (iBM < N)

**iBM \*= 2;**

)

fi11(tree.begin (), tree.end() , 0) ;

*II* initialize the tree with the given array of values FenwickTree(int val[], int n)

: N(n)

iBM = l;

while (iBM < N) iBM \*= 2;

tree.resize(iBM+l); fill(tree.begin(), tree.end(), O); for (int i = O; i< n i++) {

assert(val·(i] >= 0) incEntry(i, val[i] )

*II* increment the entry at position idx by val (use negative val for

Tu9' ,August 19, 2014

*II* return the entry indexed by idx int getEntry(int idx) canst

{

assert(O <= idx && idx < N);

**int val, parent;** val = tree(idx); if (idx > O) {

parent = idx & (idx-1); idx--;

while (parent != idx) val -= tree(idx]; idx &= idx-1;

}

return val;

// return the largest index such that the cumulative freqciency is

// what is given, or -1 if it is not found

*II*

*II* NOTE: the index returned can be greater than the length of the original

*II* array. If that is not wanted, take the minimum of the original

// N-1 and the result (if it's not -1). int getindex(int sum) canst

(

if (sum < tree[O]) return -1; sum -= tree[O);

**int idx = O;**

**int bitmask = iBM;**

while (bitmask != 0 && idx < (int)tree.size O -l) { int tidx = idx + bitmask;

if (sum >= tree [tidx]) { idx = tidx;

sum -= tree(tidx);

)

bitmask >>= l;

if (sum != O) **return -1;** else {

fenwicktree.cc *35l'---...-'..*

*r·\ .* ·.. ,

*),i*

'>,<.i:."-"'.;)

l;

'\•,,.. .

*0::},:*

1 · . *)*

*\_.,.i*



*r*

CI ?

Aug 19, 14 21:36

fenwicktree.cc

Page 3/3

,,---,

*11* ,

*-*

*/ \ (7* )

/ }

**return idx;**

\..\_ / :

Aug 19, .14 21:36

/\*

**fflinsolve.cc**

Printed by Darcy ·

Page 1/2

private:

int N, iBM;

**vector<int> tree;**

} ;

\* Solution of systems of linear equations over the integers

* Author: Howard Cheng
* Reference:

\* K.O. Geddes, S.R. Czapor, G. Labahn. "Algorithms for Computer Algebra." Kluwer Academic Publishers, 1992, pages 393-399. ISBN 0-7923-9259-0

\*

* The routine fflinsolve solves the system Ax = b where A is an n x n matrix
* of integers and b is an n-dimensional vector of integers.

\*

* The inputs to fflinsolve are the matrix A, the dimension n, and an
* output array to store the solution x star = det(A)\*x. The function
* also returns the det(A). In the case that det(A) = O, the solution
  + vector is undefined.
* Note that the matrix A and b may be modified.

*\* I*

#include <iostream> using namespace std; canst int MAX\_N = 10;

i(nt fflinsolve(int A[MAX-NJ [MAX-N], int b[], int x-star[], int n) int sign, d, i, j, k, k\_c, k\_r, pivot, t;

sign = d = l;

for (k c = k r = O; k c < n; k c++)

/\* eliminate column-k\_c *\*I* -

*I\** find nonzero pivot \*/

for (pivot = k\_r; pivot < n && !A[pivot] [k\_r]; pivot++)

if (pivot < n) (

*I\** swap rows pivot and k r \*/ if (pivot != k\_r) ( -

for (j = k\_c; j < n; j++J ( t = A[pivot] (j];

A[pivot] [j] = A[k r] [j];

A [k r] [ j ] = t ; -

-

}

t = b[pivot]; b[pivot] = b[k r];

b[k\_r] = t, -

**sgn \*= -1;**

*!•* do elimination *\*I*

for (i = k\_r+l; i < n; i++) ( for (j = k\_c+l; j < n; j++) (

} A[i] [j ] = (A[k-r] [k-c]\*A[i] [j]-A[i] [k-c]\*A[k-r] [j])/d;

b [i] = (A[k r] [k c]\*b [i]-A [il [k cl \*b [k r])/d; A(i] [k\_c] =-0; - - -

}

if (d) (

d = A [k\_r] [k\_c] ;

}

**k r++;**

else (

*I\** **entire** column is 0, det(A) o •/

d = O;

Tuesday August 19, 2014 fenwicktree.cc, fflinsolve.cc 36/77

Aug 19, 14 21:36 **floyd.cc**

*I\**

\* Floyd's Algorithm

\*

\* Author: Howard Cheng

\*

Printed by Darcy Best

Page 1/2

* The following code takes a graph stori,d in an adjacency matrix "graph",
* and returns the shortest distance from node i to node j in dist (i] [jJ
* We assume that the weights of the edges is not DISCONNECT, and the

\* DISCONNECT constant is used to indicate the absence of an edge.

*\*I*

#include <iostream>

**#include <cassert> using namespace std;**

canst int MAX NODES = 26; canst int DISCONNECT = -1;

int graph(MAX NODES] (MAX NODES];

int dist[MAX\_NODESl [MAX\_NODESl ;

void floyd(void)

{

**int i , j , k;**

for (i = O; < MAX\_NODES; i++) {

for (j = 0 j < MAX\_NODES; j++) dist[i] [ ] = graph[i] [j ] ;

for (k = O; k < MAX\_NODES; k++) { for (i = O; i < MAX NODES; i++) {

for (j = O; j < MAX\_NODES; j++) {

if (dist[i](kl != DISCONNECT && dist [kl ·[jJ != DISCONNECT)

·int temp = dist [i] [kl + dist [kl [j ] ; .

if (dist[i](j] == DISCONNECT I I dist[i][j) > temp) { dist (i] [j ] = temp;

) )

for (i = O; i < MAX\_NODES; i++). {

dist [il [il = O;

int main(void)

{

**int W;**

**int i, j ;**

*I\** clear graph \*/

for (i = O; i < MAX\_NODES; i++) { for (j *=* O; j < MAX NODES; j++)

graph[i] (j] = DISCONNECT;

) }

A'' ay August 19, 2014

1. *t*

/\* read graph \*/

**cin >> i >> j >> w;**

while (!(i *==* -1 && j *==* -1)) {

assert(O *<=* i && i < MAX NODES·&& 0 <= graph[i] [j] = graph[j] [i] = w;

**cin >> i >> i >> w;**

fflinsolve.cc, floyd.cc

&& j < MAX\_NODES);

d- :::.:..: *-*

Aug 19, 14 21 :36

**fflinsolve.cc**

Page 2/2

if (!d) {

for (k = k r; k < n; k++) {

if (b[k)) {

/\* inconsistent system \*/

**cout << 11 Inconsistent system. 11 << endl;**

**return O ;**

} )

*I\** multiple solutions \*/

**cout << )\ More than one solution. 11 << endl; return O;**

*I\** now backsolve \*/

for (k = n-1; k = O; k--) { x\_star[k] = sign\*d\*b(k]; for ( j = k+l; j < n; j++) {

)

x-star [kl -= A [kl [j ] \*x-star [ j ] ;

x\_star[k] *I=* A[k] [kl ;

**return sign\*d;**

int main(void)

{

**int n, i,-j;** - -

**int det;**

int A(MAX N] [MAX N]' X star[MAX N]' b[MAX N];

-

-

**while (cin >> n && 0 < n && n**

MAX\_N)

**cout << 11 Enter < < endl;**

for ( i = O; i < n; i++) (

**A: 11**

for (j = o; j < n; j ++ ) {

cin » A[i] [j ] ;

}

**cout << 11 Enter << endl;**

for (i = O; i < n; i++) { cin » b[i];

**b: 11**

)

if (det = fflinsolve(A, b, x star,

n) ) {

**cout << 0 det** = << **det < <-endl;**

11

**cout << star =**

for (i = O; i < n; *i++)* {

**11 x**

11 ;

**cout << x\_star [i]** <<

11

**11** •

}

**cout** << endl; else {

**cout < < is singular. 11**

**11 A**

**<< endl;**

}

return O;

- / '--./).

( , J

' .

-.. ··'/ .

CL/ .

*" ;* c\

,-,.--.:- *r--*



Aug 19, 14 21:36 floyd.cc Page 2/2

floyd ();

*I\** do queries \*/

**cin** >> i >> **j;**

while (! (i == -1 && j == -1)) {

assert(O <= i && i < MAX NODES && 0 <= j && j < MAX NODES);

**cout << i** << 11 11 << **j** <<-11 : 11 << **dist [i] [j] << endl7**

**cin** >> i>> j;

**return \_O ;**

Printed by Dar t,

.---- ----------,

Aug 19, 14 21:36 **floydpath.cc** Page 1/2

*I\**

\* Floyd's Algorithm

* Author: Howard Cheng
* The following code takes a graph stored ·in an adjacency matrix "graph",
  + and returns the shortest distance from node i to node j in dist[i) [j) .
* We assume that the weights of the edges is not DISCONNECT, and the
* DISCONNECT constant is used to indicate the absence of an edge.

\* Call extract\_path to return the path, as well as its length (in terms

* + of vertices). The length is -1 if no such path exists.

\*/

**#include <iostream>**

#include <cassert>

**using namespace std;**

canst int MAX NODES = 26; canst int DISCONNECT = -1;

int graph[MAX NODES) [MAX NODES) ; int dist[MAX NODES) [MAX NODES) ; int succ(MAX=NODES) (MAX=NODES);

void floyd(void)

{

int i, j, k;

for (i = O; i < MAX\_NODES; i++) { for (j = O; j *<* MAX NODES; j++) (

dist(i] [j] = graph(i] (j];

if (i == j 1 1 graph[i) [j) == DISCONNECT)

.succ[i) [j) = -1; else {

succ[il [j) = j ;

for (k = O; *k* < MAX\_NODES; k++) { for (i = 0; i < MAX NODES; i++) (

for (j = O; j < MAX\_NODES; j++) {

if ( i != *k* && dist [i) [k) != DISCONNECT && dist (k] [j ] != DISCONNECT) int temp = dist[il (kl + dist[k) (j];

if (dist[i) [j] == DISCONNECT· 1 1 dist [i] [j) > temp)

dist [i)( j J = temp; succ[il (jl = succ[i] [kl ;

.} else if (dist(il [j) == temp && succ[i) (kl < succ(i] ( j ] )

*/ +* put tie-breaking on paths here \*/

/\* e.g. the test above chooses lexicographically smallest \*/ *I\** paths, but ignores the number of vertices in the \*/ *I\** path. To really do lexicographically sorting \*/ *I\** properly, you also need to have len[i] (j] which .\*/ *I\** . can be computed easily as well. \*/ succ [i] (jl = succ [i] [kl ;

) }

) )

for (i = O; i *<* MAX NODES; i++) {

dist!il [i] = o; -

}

Tuesday August 19, 2014

floyd.cc, floyd\_path. cc 38/77

Aug 19, 14 21:36 **floyd\_path.cc** Page 2/2

int extract path(int u, int v, int path[])

( -

Printed by Darcy Best

Aug 19, 14 21:36 **frac2dec.cc** Page 1/2

*II* Converts a fraction (with integral numerator and denominator)

*II* to its decimal expansion.

*II*

int len = O ;

if (dist[u) [v}

**return -1;**

DISCONNECT) {

*II* Author: Darcy Best

*II* Date : August 22, 2010

*II* .

***J I* Since we are dealing with rational numbers, one of two cases**

path[len++] *=* u; while (u != v) {

*II* **occur:**

*II* 1. The number will terminate

*II* 2. The number will repeat

*II*

u *=* succ [u} [v] ;

path[len++] = u;

**return len;**

int main(void)

{

**int m, w , i, j ;**

*II* The algorithm is O(D) where D is the absolute value of the

*II* denominator.

**#include <iostream>**

#include <String>

#include <algorithm>

#include <Cstdlib>

**#include <Cassert> using namespace std;**

int path[MAX\_NODES], len;

*!•* clear graph \*/

for (i *=* O; i < MAX\_NODES; i++) {

for (j = O; j < MAX NODES; j++)

canst int MAX\_DENOM

sting itoa(int x) { string ans; while(x) {

1001;

graph[i} [j} = DISCONNECT;

ans += (x % 10) + ' D ' ;

*X I=* 10;

}

*I\** read graph \*/

**cin >> i >> j >> w;**

while (! ( i == -1 *&Ec* j *==* -1) ) (

assert(O <= i && i < MAX NODES "" 0

graph[i) [j) = l\*graph[j] i] *=\* I* w;

*&Ec* j < MAX\_NODES);

reverse(ans.begin() ,ans.end());

return (ans.length () ? ans : " o " ) ;

int firstSeen[MAX\_DENOM];

**cin > > i** >>

floyd ();

*I\** do queries *\* I*

**cin >> i > > j ;**

>> **W;**

void frac2dec(int numer,int denom,string& decimal,int& numRepDigs) {

**assert(denom != 0);**

*II* Determine if it is a plus or a minus

**decimal** = ,," ;

if(numer < O && denom *>=* 0 J J numer >= O *&Ec* denom < O) {

**decimal** += 11 - 11 ;

while (! (i *==* -1 "" j == -1)) {

°'" "°'

assert(O <= i i < MAX NODES *&Ec* O <= j j < MAX NODES);

**cout << i** << ". 11 << **j** <<-" : **n < < dist ( i] [j] << endli**

} else {

**decimal** += 11 + 11 ;

len =·extract\_path( i, j , path); for (m = *D ;* m < len; m++) {

if {m) {

**numer**

denom

abs(numer); ab,; (denom);

**cout** << 11 11 ;

J

cout << path[m];

}

**cout << endl; cin >> i >> j;**

*II* Left of the decimal point decimal += itoa(numer *I* denom);

**numer %= denom;**

if ( !numer) { numRepDigs = O; **return;**

return O;

*Twr r* August 19, 2014

*(l ..*

*::./*

*II* Add the decimal point

**decimal +=** '.';

*II* Right of the decimal point fill(firstSeen,firstSeen+denom,-1); **int rem = numer;**

while(rem != 0 "" firstSeen[rem] == -1) { firstSeen[rem] decimal.length();

**rem \*= 10;**

decimal += itoa(rem *I* denom);

**rem %= denom;**

floyd\_path.cc, frac2dec.cc *(sJ.*

*. .*

30/

,,···

...\_,*!*

*·<::.*I!

(''''-c. . *!*·*)*



;--,

CL-

Aug 19, 14 21:36

frac2dec.cc

Page 2/2

numRepDigs = (rem ? decimal,length{ ) - firstSeen[rem] 0) ;

int main() {

int numerator,denominator,repDigs; string decimal;

while{cin >> numerator >> slash >> denominator) {

frac2dec(numerator,denominator,decimal,repDigs);

**cout << numerator <<** *I* **<< denominator <<** =

11 11

11 11

**<< decimal << endl;**

if{repDigs == O)

**cout << "This expansion terminates. 11 << endl; else**

**cout << 11 The last** << **repDigs** <<

11

" **digits repeat foz-ever .** '' << **endl ;**

cout << endl;

}

**return O;**

*,I'-*

*/*

,,

*/ ·*

*(( '*

*1t- '(* /

Aug 19, 14 21:36 **fraction.cc**

*II*

*II* Fraction implementation

*II*

*II* Author: Darcy Best

*II*

*II* Does NOT ever check for division by O.

Printed by Darcy

Page 1/4

*II* Division by o will only cause a runtime error if you us the

*II* toDouble() function.

*II*

#include <iostream>

#include <CStdlib>

**using namespace std;**

*II* Change this to whatever integer data type will prevent overflow

*II* - Biginteger works with this class typedef long long int dataType;

class Fraction{ public: ·

Fraction(dataType num=O,dataType denom=ll ;

double toDouble( ) canst; void reduce{);

*II* Changes the fraction itself. void selfReciprocal();

*II* Returns a new fraction, leaving the original. Fraction reciprocal{) canst;

Fraction& operator+=(const Fraction& x); Fraction& operator-={const Fraction& x); Fraction& operator\*=( const Fraction& x); Fraction& operator/=(const Fraction& x);

bool operator<{const Fraction& x) canst; bool operator=={const Fraction& x) canst;

dataType num.,denom;

) ;

Fraction operator+{const Fraction& x,const Fraction& y); **Fraction operator-(const Fraction& x, const Fraction& y);** Fraction operator\*{const Fraction& x,const Fraction& y); Fraction operator/(const Fraction& x,const Fraction& y);

istream& operator>>{istream& is,Fraction& x); ostream& operator<<(ostream& os,const Fraction& x);

Fraction: :Fraction{dataType n,dataType d){

if {d < O ) {

**num =. -n;** denom = -d; else {

**num = n;**

**denom = d;**

)

reduce();

double Fraction: :toDouble() canst( return l.O\*numldenom;

*/!* Howard' s GCD function with no checks dataType gcd(d.ataType a, dataType b)

-

{

Tuesday August 19, 2014 frac2dec.cc, fraction.cc

*40177*



Aug 19, 14 21:36 **fraction.cc**

dataType r; while {b) (

r = a % b;

a = b;

b = r;

}

**return a;**

void Fraction: :reduce(){

dataType g = gcd (abs(num) , denom) ; num /= g;

denom *I=* g;

void Fraction: :selfReciprocal() { swap(num,denom);

if (denom < 0) {

**num = -num;**

denom = -denom;

) )

Fraction Fraction: :reciprocal() const{

**return Fraction(deom,num);**

*II* Overflow potential in the denominator.

Page 2/4

Printed by Darcy Best

Aug 19, 14 21:36 **fraction.cc** Page 3/4

denom \*= x.num;

if(denom < 0){ **num = -num;** denom = -denom;

)

reduce(); return (\*this);

*II* Careful with overflow. If it is an issue, you can compare the

*II* double values, but you SHOULD check for equality ·BEFORE converting

bool Fraction: :operator<(const Fraction& x) canst{

**return (num\*x.denom) < (x.num\*denom);**

bool Fraction: :operator==(const Fraction& x) canst{ return (num == x.num) && (denom == x.denom);

Fraction operator+( const Fraction& x,const Fraction& y){ Fraction a (x) ;

**a += y;**

**return a;**

.Fraction operator-(const Fraction& x,const Fraction& y)( Fraction a(xl ;

*II* I've tried to fac1 or out as much as possible before,

*II* But be careful.

*II*

*II* (w)*I* (a\*g) + (z)/ (b\*g)

*II =* (w\*b)/(a\*g\*b) + (a\*z)/(a\*g\*b)

*II =* (w\*b + a\*z)l(a\*g\*b)

Fraction& Fraction::operator+=(const Fraction& x) { dataType g = gcd(denom,x.denom);

dataType a = denom *I* g; dataType b = x.denom *I* g;

**num = num \* b + x.num \* a; denom \*= b;**

reduce(); return (\*this);

Fraction& Fraction::operator-=(const Fraction& x) {· dataType g = gcd(denam,x.denom);

dataType a = denom *I* g; dataType b = x.denom *I* g;

**num = num \* b - x.num \* a;**

**denom \*== b;**

reduce(); return (\*this);

**Fraction& Fraction: : c,perator\*,,,, *(* canst Fract ion& x)** {

**num \*= x.num;** denom \*= x.denom; reduce();

return (\*this);

a -= y;

**return a;**

Fraction operator\*(const Fraction& x,const Fraction& y){ Fraction a(x);

**a \*= y;**

**return a;**

Fraction operatorl(const Fraction& x,const Fraction& y){ Fraction a(x);

a *I=* y;

**return a;**

*JI* Note that you can read in Fractions of two forms:

*JI* a/b (With any number of space between a,l,b) - The input "points" to

// the NEXT character after the denom (White space or not)

*II* c (Just an integer - The input "points" to the next NON-WHITE SPACE

*II* character. Careful when mixing this with getline.) istream& operator>>(istream& is,Fraction& x){

**is >> x.num;**

char ch;

**is >> ch;**

if(ch != ' I ' ) {

is.putback(ch);

**x.denom = l;**

else {

**is >> x.denom;**

if(x.denom < O ) (

**x.num = -x.num;**

**x.denom = -x.denom;**

)

x.reduce ();

Fraction&· Fraction: :operator/= (const Fraction& x){

**num \*= x.denom;**

Tuer . A

(,;: ugust 19, 2014

-. ·..,,

•·t ··' ·

**return is;**

fraction.cc

.·

41f ""0

*(' '----'1 .*'·

\, ·



c; l-

*II* Will output 5 for Sil and O for Oil. If you want always

*II* fractions, get rid of the if statement·

ostream& operator<<( ostream& os,const Fraction& x){

**os << x.num;**

if(x.num != 0 && x.denom != 1)

**os** << '/' << **x.denom;**

**return os;**

int main() (

**Fraction x,y;**

while(cin >> x >> y){

**"x:**

**cout** <;,<

**cout** << "y:

**cout** << **nx+y=** << x+y << endl

"

"

<< x << **endl;**

**<< . y << endl;**

**cout** << **ux-y=** << **x-y << endl**

"

"

**cout** << **ux\*y=** << **x\*y << endl**

cout <<

*"xl y,=*

**cout**

**<< endl;**

"

"

« *xly* « endl

}

**return O ;**

*l::/*

*/*

*((* \)

*/ ·*

*(( ]}*

Printed by Darcy */*

Aug 19, 14 21:36 fraction.cc

Page 4/4

Aug 19, 14 21:36 **greatcircle.cc** Page 112

*II* Great Circle distance between two points using Heaverside formula

*II*

*II* Author: Howard Cheng

*II* Reference: http:llmathforum.orgllibraryldrmathlviewl51879.html

*II* .

*II* Given two points specified by their latitdes and longitudes, as well *II* as the radius of the sphere, return the shortest distance between the *II* two points along the surface of the sphere.

*II*

*II* latitude should ·be between -90 to 90 degrees (inclusive), and *II* longitude should be between -180 to 180 degrees (inclusive) *II*

*II* There are also routines that will convert between cartesian coordinates

*II* (x,y,z) and spherical coordinates (latitude, longitude, radius).

*II*

#include <iostream>

#include <iomanip>

#include <cmath>

**using namepace std;**

const double PI = acos(-1.0);

double greatcircle(double latl, double longl, double lat2, double long2, double radius)

lat1 \*= Pil180.0; lat2 \*= Plll80.0; longl \*= Pil180.0; long2 \*= PillBO.O;

double dlong = long2 - longl; double dlat = lat2 - latl;

double a = sin(dlatl2)\*sin(dlatl2) +

cos(latl)\*cos(lat2)\*sin(dlongl2)\*sin( dlongl2); return radius \* *2* \* atan2(sqrt(a), sqrt(l-a));

void longlat2cart(double lat, double lon, double radius,

* double &x, double &y, double &z)

lat \*= Pil180.0; lon \*= PillBO.O;

x = radius \* cos(lat) \* cos(lon);

*y* = radius \* cos(lat) • sin(lon) ; z = radius \* sin(lat);

void cart2longlat(double x, double y, double z,

double &lat, double &lon, double &radius)

radius = sgrt(x\*x + Y\*Y + z\*z);

lat = (Pil2 - acos(z *I* radius)) \* 180.0 *I* PI; lon = atan2(y, x) \* 180.0 *I* PI;

int main(void)

{

int T;

**cin >> T;**

while (T-- > 0) {

const double radius = 6371009; double latl, longl, lat2, long2;

Tuesday August 19, 2014

cin >> latl >> longl >> lat2 >> long2; double xl, yl, zl, x2, y2, z2;

fraction.cc, greatcircle.cc 42/77



Printed by Darcy Best Aug 19, 14 21:36 greatcircle.cc Page 2/2 Aug 19, 14 21:36 **heron.cc** Page 1/1

longlat2cart(latl, longl, radhts, xl, yl,. zl); longlat2cart(lat2, long2, radius, x2, y2, z2);

*II* Heron's formula

*II*

double dl = sqrt( (xl-x2)\*(xl-x2) + (yl-y2)\*(yl-y2) + {zl-z2)\*(zl-z2)); double d2 = greatcircle( latl, longl, lat2, long2, radius);

cout << fixed << setprecision(O) << d2 - dl << endl; double radiusl;

cart2longlat(xl, yl, zl, latl, longl, radiusl);

**cout << latl** << ' ' << **longl** << ' ' << **radiusl << endl;**

*II* Computes the area of a triangle given the lengths of the three sides.

*II*

*II* Author: Howard Cheng

*II*

#include <iostream>

#include <iomanip>

#include <utility>

#include <Cmath>

return o; **using .namespace std;**

*II* the lengths of the three sides are a, b, and c. The routine returns *II* the area of the triangle, or -1 if the three lengths do not make a *II* triangle.

double area heron{double a, double b, double c)

-

{

if (a < b) swap(a, b);

if (a < c) swap (a, c) ;

if (b < c) swap(b, c);

if (c < a - b) return -1;

return sqrt{(a+b+c)\*{c-a+b)\*(c+a-b) \*(a+b-c))l4.0;

int main(void)

{

**double a, b, c;**

while (cin- >> a >> b >> c) {

cout << fixed << setprecision(4) << area\_heron(a, b, c) << endl;

**return O;**

,{ "' 1ay August 19, 2014

..>.......

1v.1 \_.

greatcircle.cc, heron.cc

:

*(,)*

I .

/





cC,.-

Aug 19, 14 21:36

/•

|  |
| --- |
| Aug 19, 14 21:36 **hungarian.cc** Page 2/4 |
| slackx[yJ = x;  } }  )  int max weight matching(int G[MAX\_N] [MAX\_NJ , int N, int matching[MAX\_N])  { - -  int revmatch[MAX NJ ; *II* match from right to left  **int max\_match = O;** *II* number of vertices in current matching  fill(matching, matching+N, -1);  fill(revmatch, revmatch+N, -1);  *II* find an initial feasible labelling int lx[MAX NJ ' ly[MAX NJ *i*  fill(ly, ly+N, 0); -  for (int x = O; x < N; x++) {  lx(x] = •max\_element(G[xJ , G[xJ +N);  *II* now repeatedly find alternating tree, augment, and relabel while (max\_match < N) (  **queue<int> g;** ·  bool S[MAX NJ , T[MAX NJ ;  int prev[MAX NJ ; - fill(S, S+N,-falsel ; fill(T, T+N, false); fill(prev, prev+N, -1);  // find root of alternating tree  int root = find(matching, matching+N, -1) - matching; q.push(root);  prev[root] = -2;  S[rootJ = true;  int slack[MAX NJ , slackx[MAX NJ ; for (int y = O; y < N; y++) T  slack[yJ = (G[root] [yJ == INT MIN) ? INT MAX lx [root] + ly [y) - G [root] [y) ; -  slackx[yJ = root;  bool path\_found = false; int x, y;  while (!path\_found) {  *II* build alternating tree with BFS while (!path found && !q.empty() )  x = q.front ();  q.pop ();  for (y = O; y < N; y++) (  *II* go through edges in equality graph  if (G[x] [y) == lx[x] + ly[y) && tT[y)) if (revmatch[y) == -1) {  path found = true; break;  )  T[y] = true; q.push(revmatch[ yJ );  add\_to\_tree(revmatch[yJ , x, G, S, prev, lx, ly, slack, slackx, N);  )  if (path\_found) break; |
| *II* no augmenting path, update the labels  updatelabels(lx, ly, S, T, slack, NJ ; while (!q.empty()) { |

'

hungarian.cc

Page 1/4

*(;>}*

,-

*(( )*

Printed by Darcyst

* Maximum/minimum weight bipartite matching
* Author: Howard Cheng
* Reference:
* [,http://www.topcoder.com/tc?module=Static&](http://www.topcoder.com/tc?module=Static) dl=tutorials&d2=hungarianAlgorithm
* This file contains routines for computing the maximum/minimum weight
* bipartite matching.
* It is assumed that each half of the graph has exactly N vertices, labelled
* o to N-1. The weight between vertex i on the left and vertex j on the
* right is stored in G(iJ [jJ . The cost of the optimal matching is returned,

**and matching(i] is the verte on the right that *is* matched to vertex i**

* on the left.
* If an edge is absent, the corresponding edge weight should be:

INT MIN INT=MAX

if maximum weight matching is desired if minimum weight matching is desired

* This is an implementation of the Hungarian algorithm.
* is O (NA3 ) •

\*/

The complexity

**#include <iostream>**

#include <algorithm>

**#include <queue>**

**#include <Cassert>**

#include <Climits>

using namespace std; canst int MAX\_N = 3;

void update labels(int lx[MAX NJ , int ly[MAX NJ , bool S[MAX NJ , bool T[MAX NJ ,

* + - int slack [MAX\_NJ , int NJ - - -

int delta;

bool delta\_init = false;

for (int y = O; y < N; y++) if (T[y]) continue;

delta = delta init delta\_init = true;

}

min(delta, slack[yJ ) slack [yJ ;

for (int x = O; x < N; x++)

if (S[xJ ) lx[xJ -= delta;

}

for (int y = O; y < N; y++)

if (T[yJ I {

ly[y) += delta; else {

slack(yJ -= delta,

void add to tree(int x, int prevx, int G[MAX NJ [MAX NJ , bool S[MAX NJ ,

- - int prev[MAX NJ , int lx[MAX-N] , int ly[MAX NJ , - int slack[MAX\_NJ , int slackx[MAX\_N], int N)

S[xJ = true; prev(x] = prevx;

for (int y = O; y < N; y++) {

int temp = (G[x1 (yJ == INT MIN) ? INT MAX

if (temp < slack[y]) { - - slack[yJ = temp;

Tuesday August 19, 2014

lx(x] + ly[yJ - G(x][yJ ;

q.pop ();

hungarian.cc 44/77

Aug 19, 14 21:36 hungarian.cc Page 3/4

}

for (y = O; y < N; y++) {

if {!T[yJ && slack(yJ == O) ( if (revmat.ch [y] == -1) {

x = slackx[y); path fond = true; break;

} else {

T(y] = t.rue;

if (!S [revmatch [yJ )) { q.push(revmatch(y]);

add to tree(revmatch[y),

- ·- slackx, N) ; slackx[y), G, S, prev, lx, ly; slack,

Aug 19, 14 21:36 **hungarian.cc**

J

**return O ;**

Printed by Darcy Best

Page 4/4

} }

}

assert(path found) ;

**max\_match++7**

*II* augment along the path

for (int ex = x, cy = y, ty; ex != -2; ex = prev[cx), cy = ty) { ty = matching[ cx]

revmatch[cyJ = ex matching[ cxJ = cy

} }

*II* return the final answer int weight = O;

for (int x = O; x c N; x++) { weight += G [xJ [matching [xJ );

}

return weight;

i{nt min-weight-matching(int G[MAX-NJ [MAX-NJ , int N, int matching[MAX -NJ ) int M = INT\_MIN;

for (int i = O; < N; i++) { for (int j = o j < N; j++) {

if (G[i) [jJ = INT MAX) {

M = max(M, G[iJ )));·

J }

int newG[MAX NJ [MAX NJ ;

--o ,

for (int i i<-N, i++) ( for (int j = O; j < N; j ++) {

} newG[iJ [jJ = {G(iJ [jJ == INT-MAX) ? INT-MIN : M - G[iJ ( j ] ;

}

int weight = max weight matching(newG, N, matching); return N\*M - weight; -

J

int main(void)

{

int G(3J (3) = { {INT\_MAX,4,5}, {5,7,6}. {5, 8, 8) );

int matching(3J ;

int w = min weight matching(G, 3, matching);

**cout << "weight =** tt << w << **endl;**

for (int i = O; i < 3; i++) {

**cout** << i<< 11 is **matched to** 11 << **matching (i] << endl;**

*Tr* 'Y August 19, 2014

G·--- ...:.-

hungarian.cc

:

*4g*

t' > j i

' *j*

..

\_/





; )

*<L*

/

,---

Aug 19, 14 21:36 infix.cc Page 1/4

*I\**

* Infix expressions evaluation
* Author: Howard Cheng
* The evaluate() routine takes a string containing an infix arithmetic
* expression, and return the numeric result after evaluation. The
* **parameter error indicates whether an error has occurred (syntax**
* error, illegal operation, etc.). If there is an error the result
* **returned is meaningless.**
* The routine assumes that the operands in the input are integers
* with no leading signs. It supports the standard +, -, \*, *I* and
* parentheses. If you need to support more operators, operand types,
* etc., you will need to modify the code. See comments below.

*\*I*

**#include <iostream>**

#include <string>

#include <stack>

#include <cctype>

#include <cstdlib>

**using namespace std;**

*II* What is a token? Modify if needed (e.g. to support variables, extra

*II* operators, etc.)

**struct Token**

{

enum Type {NUMBER, PLUS, MINUS, TIMES, DIVIDE, LEFT\_PAREN, RIGHT\_PAREN};

*II* priority of the operators: bigger number means higher priority

// e.g. *\*I* has priority 2, +- has priority 1, ( has priority O

int priority[?];

*II* is the operator left associative? It's assumed that all operators

// of the same priority level has the same left/right associative property bool left\_assoc[7];

Type type; long val;

Token()

{

*.*

*/ ) / )*

Printed by Darcy Best

Aug 19, 14 21:36 **infix.cc** Page 2/4

return false;

switch (expr[start])

**case** ' ( ' ;

type = LEFT\_PAREN; break;

case ')':

type = RIGHT PAREN;

break; -

**case** ,\*, :

typ·e = TIMES; break;

case '/':

type = DIVIDE; break;

**case** ' + ' :

type = PLUS; break;

**case** '-' :

type = MINUS; break;

**default:**

// check for number

canst char •s = expr.c\_str() + start; char \*p;

val = strtol(s, &p, 10);

if (s == p) { **error = true;** return false;

}

type = NUMBER; start += (p - s);

)

if (type != NUMBER)

**start++;**

**return true;**

) ; }

*II* Modify this if you need to support more operators or change their

*II* meanings.

*II*

// **returns true if operation is successful**

priority [l] priority[3] priority [SJ left\_assoc [1]

priority[ 2J = 1;

priority[ 4] = 2; O;

left\_assoc[2] left\_assoc[3] left\_assoc[4) = **true;**

b{ool apply-op(stack<long> &operands, Token token) long a, J,;

if (operands.size() < 2) {

**return false;**

in.t get\_priority () { return priority[type];

bool is\_left\_assoc() { return left\_assoc[type] ;

*JI* returns true if there is a next token

bool next token(string &expr, int &start, bool &error)

-

(

int len = expr.length( );

**error = false;**

while (start < len && isspace(expr[start]))

**start++;**

}

if (start >= len) {

Tuesday August 19, 2014

infix.cc

}

if (token.type == Token::PLUS) {

b = operands.top(); operands.pop(); a = operands.top(); operands.pop(); operands.push(a+b);

else if (token.type == Token: :MINUS) b = operands.top(); operands.pop(); a = operands.top(); operands.pop(); operands.push( a-b);

else if (token.type == Token: :TIMES) b = operands.top( ); operands.pop(); a = operands. top(); operands.pop(); operands.push(a\*b);

else if (token.type == Token: :DIVIDE) { b = operands.top(); operands.pop();

a = operands.top(); operands.pop(); if (b == 0) {

**retun false;**

46/77

*\_./ .......-----*

Aug 19, 14 21:36 infix.cc Page 3/4

}

operands .push(a/b) ;

} else { return false;

)

return true;

long evaluate(string expr, bool &error)

{

**stack<Token> s;**

stack<long> operands;

**int i ;**

**Token token;**

**error = false;**

i= O;

while (token.next\_token(expr, i, error) && !error) { switch (token.type) (

case Token: :NUMBER: operands.push(token.val); break;

case Token: :LEFT PAREN: s.push(token) ;­ **break;**

case Token: :RIGHT PAREN:

while (!(error s.empty()) && s.top() .type != Token: :LEFT PAREN) if ((error = J apply\_op(operands, s.top()) )) { -

**break;**

Aug 19, 14 21:36 **infix.cc** Page 4/4

getline(cin, expr); while (!cin.eof ()) {

reult = evaluate(expr, error); if (error) { . ·

cout << "Invalid expression" << endl;

) else (

**cout** << 11 = u **<< result << endl;**

}

getline(cin, expr);

}

**return O ;**

)

s.pop();

)

if (!error) s.pop();

)

break;

default: // arithmetic operators while (!error && !s.empty() &&

(token.get\_priority() < s.top() .get\_priority() I J

token.get\_priority() == s.top() .get\_priority() && token.is\_left\_assoc())) {

error = !apply\_op(operands, s.top()); s.pop();

)

if (!error) { s.push (token) ;

)

if (error) break;

)

while (!error && !s.empty()) {

error =·!apply\_op (operands, s.top ()); s.pop ();

)

error I = (operands.size() != 1) ;

if (error) {

**return O;**

)

return operands.top() ;

int main(void)

(

int result; string expr; **bool error;**

&

Tyr 'Y August 19, 2014

*(,-\:*

\ ;· *i*

·."*\_,,J*

infix.cc

.{ '-/I

· --

47

( ·

*----)*





*r*l*r*b- *11*

*r*

*/*

,-;-- \

-

/\

**Printed by Dart** ,

**Aug 19,** 14 21:36

int **mult.cc**

**Page 1/2**

/\*

* Integer multiplication/division without overflow
* Author: Howard Cheng
* Given a list of factors in the numerator (num, size n) and a list
* of factors in the denominator (dem, size m), ft returns the product
* of the numerator divided by the denominator. It is assumed that

\* the numerator is divisible by the denominator (ie. the result

* is an integer). Overflow will not occur as long as the final result

\* is representable.

\*/

#include <iostream>

**#include <Cas ert>**

using namespace std; int gcd(int a, int b)

{

**int r;**

while (b) (

**r = a % b;**

a = b;

b = r;

}

assert (a >= O ) ;

**return a;**

int mult(int A[], int n, int B[], int m)

{

int i, j , prod, d;

int count = O;

/\* **unnecessary if the two lists are positive \*/**

for (i = O ; i < n; i++} {

if (A[i] < 0) {

A[i] \*= -1;

**count++;**

for ( i = o ; i< m; i++) {

if (B[i] < 0 ) {

- B (i] \*= -1;

**count++;**

}

for (i = o ; i < n; i++) { for (j = O ; j < m; j++)

d = gcd(A[i), B (j) ) ;

A(i) /= d;

B [j ] /= d;

) .

prod = l;

for (i = O; i< n; i++) prod \*= A[i];

)

for (j = o ; j < m; j + + ) ( ·

assert(B[jl == 1);

)

reurn (count % 2 == 0) ? prod -prod;

int main(void)

Tuesday August 19, 2014 int\_muIt.cc **48/77**

{.

int A [1000], B [1000], n, m, i;

while (cin >> n >> m && n > O && m > 0 ) {

for (i = O ; **i < n;** i++) (

cin » A[i];

for ( i = O ;

}

**i < ffi j** i++) {

cin >> B (il ;

}

**caut << 11 prod** = << **mult(A,n,B, m)** << **endl ;**

11

return O ;

**Aug 19, 14 21:36**

**int mult.cc**

**Page 2/2**

Aug 19, 14 21:36 intersectTF.cc Page 1/2

*I\**

* **Line Intersection**
* Author: Howard Cheng
* **Reference:**
* CLRS, "Introduction to Algorithms", 2nd edition, pages 936-939.
* Given two lines specified by their endpoints (al, ail and (bl, b2),
* returns true if they intersect, and false otherwise. The intersection
* **point is not known.**

\*/

**#include <iostream>**

#include <cmath>

**using namespace std;**

/ \* how close to call. equal •/ canst double EPSILON = lE-8;

struct Point { double x, *y ;*

} ;

double direction(Point pl, Point p2, Point p3)

(

double xl = p3.x - pl.x double yl = p3.y - pl.y double x2 = p2.x - pl.x double y2 = p2.y - pl.y return xl\*y2 - x2\*yl;

}

i{nt on-segment(Point pl, Point p2, Point p3)

return ((pl.x <= p3 .x && p3 .x <= p2 .x) 1 1 (p2.x <= p3 .x && p3 .x <= pl .x)) &&

((p1.y <= p3.y && p3.y <= p2.yl I I 1p2.y <= p3.y && p3.y <= p1.yl l ,

int intersect(Point al, Point a2, Point bl, Point b2)

{

double dl = direction(bl, b2, al)

double d2 = direction(bl, b2, a2) double d3 = direction(al, a2, bl) double d4 = direction(al, a2, b2)

if ( ( (dl > EPSILON && d2 < -EPSILON) 1 1 (dl < -EPSILON && d2 > EPSILON) ) &&

((d3 > EPSILON && d4 < -EPSILON) (d3 < -EPSILON && d4 > EPSILON))) {

**return 1;**

) else (

return (fabs(dl) < EPSILON && on segment(bl, b2, al)) I I

I I

(fabs(d2) < EPSILON && on\_segment(bl, b2, a2)) (fabs(d3) < EPSILON && on segment(al, a2, bl)) (fabs(d4) < EPSILON && on=segment(al, a2, b2));

) }

int main(void)

{

**Point a, b 1 c, d;**

**int al, a2, a3, a4, as, a6, a7, a8;**



|  |  |
| --- | --- |
|  | Aug 19, 14 21:36 **intersectTF .cc** Page 2/2 |
|  | cout << **n yesu** << **endl;**  else (  cout << **11 No 11 << endl;**  **return O;** |

**while (cin >> al >> a2 >> a3** >> **a4 >> as** >> **a6 >> a7 >> as)** {

a.x = al; a.y = a2;

b.x = a3; b.y = a4;

c.x = as; *c .y* = a6;

d.x = a7; d.y = a8;

if (intersect(a, b, c, d) ) {

Tur v August 19, 2014

(5,

-, -'•...\_....\_....

interectTF.cc

-

·C*...* :*\_* 3:·,

4" .



r---'.\

-

\

(;/

/)

\:::;.,/ i

Printed by Darcy Best

Aug 19, 14 21:36 intersect\_circle **circle.cc** age 1/2

*II* Determines the point(s) of intersection if a circle and a circle

*II*

*II* Author: Darcy Best

*II* Date October l, 2010

*II* Source: http:lllocal.wasp.uwa.edu.au/-pbourkelgeometryl2circlel

*II* .

*II* Note: A circle of radius O must be considered independently.

*I I* See comments\_ in the implementation. ·

#include ciostream>

#include.ciomanip>

#include ccmath>

#include calgorithm>

**using namespace std;**

#define SQR (X) ( (X) \* (X) )

*II* How close to call equal canst double EPS = le-4;

bool dEqual(double x,double y)(

Aug 19, 14 21:36 **intersect circle\_circle.cc** Page 2/2

*II* Note that both are true -- It all depends on the problem

**return 1;**

else { return O;

double d = hypot(cl.x-c2.x,cl.y-c2.y);

*II* Check if the circles are exactly the same. if(dEqual(cl.x,c2.x) && dEqual(cl.y,c2.y) && dEqual(cl.r,c2.r))

return 3;

*II* The circles are disjoint if(d cl.r + c2.r + EPS)

**return O ;**

*II* One circle is contained inside the other -- No intersection if(d c abs(cl.r-c2.r) - EPS)

**return O;**

return fabs.(x-y) c EPS;

struct Point{

double a double h

Point P;

(SQR(cl.r) - SQR(c2.r) + SQR(d)) *I* (2\*d);

sqrt(abs(SQR(cl.r) - SQR(a)));

double x,y;

bool operatorc(const Point& a) canst{ if(dEqual(x,a.x))

P.x = cl.x + a I d \* (c2.x - cl.x);

P.y = cl.y + a I d \* (c2.y - cl.y);

return *y* c·a.y;

**return x < a.x;**

}

.

**ansl.x**

ansl.y

P.x + h I d \* (c2.y - cl.y);

P.y - h I d \* (c2.x - cl.x);

) ;

*II* Prints out the ordered pair. This also accounts for the negative o .

if(fabs(h) c EPS) return l;

void print(const Point& a){

**cout** << " ( " ;

if(fabs(a.x) c le-4)

ans2.x ans2.y

P.x - h I d \* (c2.y - cl.y);

P.y + h *I d \** (c2.x - cl.x);

**cout**

else

**"0 . 000 11 ;**

**return 2;**

**cout << a.x;**

**cout** << 11 , 11 ;

if(fabs(a.y) c le-4)

**cout << 11 0 . 00 0 11 ;**

else

**cout << a.y;**

**cout** << 11 ) 11 ;

struct Circle{ double r,x,y;

) ;

*II* Input:

*II* Two circles to intersect

*II*

*II* Output:

*II* Number of points of intersection points

*II* If 1 (or 2), then ansl (and ans2) contain those points.

*II* If 3, then there are infinitely many. (They're the same circle)

int intersect\_circle\_circle(Circle cl,Circle c2,Point& ansl,Point& ans2) (

*II* If we have two singular points if(fabs(cl.r) c EPS && fabs(c2.r) c EPS) (

if(dEqual(cl.x,c2.x) && dEqual(cl.y,c2.y)) {

int main() {

cout < fixed << setprecision(3)l Circle Cl,C2;

**Point al,a2;**

while(cin >> Cl.x >> Cl.y >> Cl.r >> C2.x >> C2.y >> C2.r) ( int num = intersect circle circle(Cl,C2,al,a2); switch(num) ( - -

**case o :**

**cout << 11 NO INTERSECTION 11 << endl ;**

break;

**case l :**

print(al); cout << endl; break;

case 2:

if (a2 c al) swap(al,a2);

print(al);print(a2);cout cc endl; break;

**case 3 :**

cout cc "THE CIRCLES ARE THE SAME" cc endl;

**break;**

)

**ansl.x = cl.x;**

ansl.y = cl.y;

*II* Here, you need to know what the intersection of two exact points is:

*II* "return 1;" - If the points intersect at only 1 point

*II* "return 3;" - If the circles are the same

**return O;**

Tuesday August 19, 2014 intersect\_circle\_circle.cc 50/77

Aug 19, 14 21:36 intersect **iline.cc** Page 1/2

/\*

* **2-D Line Intersection**
* Author: Howard Cheng
* Reference,
* <http://www.exaflop.org/docs/cgafaq/cgal.html>
* This routine takes two infinite lines specified by two points, and
* determines whether they intersect at one point, infinitely points,
* **or no poins. In the first case, the point of intersection is also**
* returned. The. points of a line must be different (otherwise,
* the line is not defined).

\* /

#include <iostream>

**#include <cmath>**

**#include <cassert>**

using namespace std;

*i •* how close to call equal •/ canst double EPSILON = lE-8;

struct Point { double x, y;

**return O ;**

} ,

/\* returns 1 if intersect at a point, o if not, -1 if the lines coincide \*/ i{nt intersect-iline(Point a, Point b, Point c, Point d, Point &p)

double r;

double denom, numl, num2;

assert((a.x != b.x I I a.y != b.y) && (c.x != d.x J I c.y != d.y));

numl = (a.y - c.y)\*(d.x - c.x) - (a.x - c.x)\*(d.y - c.y);

num2 = (a.y - c.y)\*(b.x - a.x) - (a.x - c.x)\*(b.y - a.y);

denom = (b.x - a.x)\*(d.y - c.y) - (b.y - a.y)\*(d.x - c.x);

if (fabs(denom) >= EPSILON) { r = numl *I* denom;

p.x = a.x + r\* (b.x - a.x);

p.y = a.y + r\* (b.y - a.y); return l;

) else {

if (fabs(numl) >= EPSILON) (

**return D ;**

) else (

**return -1;**

} )

int main (void)

(

Point a, b, c, d, p;

**int res ;**

while (cin >> a.x >> a.y >> b.x >> b.y >> c.x >> c.y >> d.x >> d.y) { res = intersect iline(a, b, c, d, p);

if (res == 1) {-

**1**

**cout << 11 Intersect at** (' **cc p.x** << 11

} else if (res == O) {

**cout << 11 D0n't intersect 11 << endl;**

} else {

**11**

**1 << p .y** << 11 ) 11 << **en.fl.l ;**

} **cout**

}

<< **Infinite**. **number of intersections" << endl;**

, 1ay August 19, 2014

Aug 19, 14 21:36

**intersect\_iline.cc**

Page 2/2

' <1::..:.:::::,-; --

:...-

intersect\_iline.cc ·:



;---· ' .'

Cl.:z:- *}*

|  |  |  |
| --- | --- | --- |
| Aug 19, 14 21:36 **intersect iline circle.cc** |  | Page 2/2 |
| ansl.y circCentre.y + (-D\*dx + abs(dy)\*sqrt(desc)) *I* dr;  ans2.x circCentre.x + (D\*dy - sgn\*dx\*sqrt(desc) ) *I* dr; ans2.y circCentre.y + (-D\*dx - abs(dy)\*sqrt(desc)) *I* dr;  **return 2;**  int main() { Line L; Circle C; Point al,a2;  cin >> L.pl.x >> L.pl.y >> L.p2.x >> L.p2.y;  **cin >> C.centre.x >> C.centre.y >> C.radius;**  int nurn = intersect iline circle(L,C,al,a2); if(nurn == 0) - -  cout << "NO INTERSECTION." << endl;  else if(nurn == 1)  **cout << ''ONE INTERSECTION:** ( 11 << **al . x** << 11 1 11 << **al .** y <<  else if(nurn == 2)  **cout << 11 TWO INTERSECTIONS:** ( 11 << **al .x** << 11 , 11 << **al .y** <<  << 11 ( " << **a2 .x** << 11 , 11 << **a2 . y** << 11 ) 11 << **endl ;**  **return O ;** | 11 11  )  11 ) 11 | << **endl ;** |

*(* '. \ 1

Printed by Darbs{

Aug 19, 14 21:36 intersect· iline circle.cc Page 1/2

*II* Determines the point(s) of intersection if a circle and a line

*II*

*II* Author: Darcy Best

*II* Date May 1, 2010

*II* Source, http:llmathworld.wolfram.comlcircle-Linelntersection.htrnl

#include <iostrearn>

#include <crnath>

**using namespace std;**

#define SQR (X) ((X) \* (X))

*II* How close to call equal canst double EPS = le-7;

bool dEqual(double x,double y){ return fabs(x-y) < EPS;

struct Point{ double x,y;

) ;

struct Line{ Point pl,p2;

} ;

struct Circle{ **Point centre;** double radius;

};

*II* Input of:

*II* - 2 distinct points on the line

*II* - The centre of the circle *II* - The radius of the circle *II* Output:

*II* Number of points of intersection points

*II* If 1 or 2, then ansl and ans2 contain those points.

int intersect\_iline\_circle( Line l,Circle c,Point& ansl,Point& ans2) {

Point pl = l.pl; Point p2 = l.p2;

**Point circCentre = c.centre;**

double rad = c.radius;

pl.x p2.x pl.y p2.y

**circCentre.x circCentre.x** circCentre.y circCentre.y

double dx double dy double dr double D

p2 .x - pl.x;

p2 .y - pl.y;

SQR(dx) + SQR(dy);

pl.x\*p2.y - p2.x\*pl.y;

double desc = SQR(rad)\*dr - SQR(D); if(dEqual(desc,O) ){

ansl.x = circCentre.x + (D\*dy) *I* dr; ansl.y = circCentre.y + (-D\*dx) *I* dr;

return l;

else if(desc < O) { return O;

double sgn = (dy < -EPS ? -1 1);

ansl.x = circCentre.x + (D\*dy + sgn\*dx\*sqrt(desc)) *I* dr;

Tuesday August 19, 2014

intersect\_iline\_circle.cc *52177*

C d;

}

else if (a.y == d.y) {

else if (b.y < c.y I I d.y < a.y) {

p = b ;

**P = a;**

else if (b.x < c.x I I d.x < a.xi {

u .. August 19, 2014

.

*};*

intersect\_line.cc .;( <

'. V.

,:i;·

5

*'1?*

(*'--·-'-<:!.;?,\_}­*

...

G

\', ·

*----.:-..:-i*

..,;;..'".

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Aug 19, 14 21:36 | intersect | line.cc |  | Page 1/2 |  | Aug 19, 14 21:36 |  | **intersect\_line.cc** | Page 2/2 |
| *I\**   * **2-D Line Intersection** * Author: Howard Cheng * **Reference:** * <http://www.exafl.op.org/docs/cgafaq/cgal.html> * This routine takes two line segments specified by endpoints, and   \* determines whether they intersect at one point, infinitely points,  \* or **no points. In the first case, the point of intersection is also**  \* returned. The endpoints of a line must be different (otherwise,  \* the line is not defined).  \*/  **#include <iostream>**  #include <cmath>  **#include ccassert>**  using namespace std;  /\* how close to call equal \*/ canst double EPSILON = lE-8;  struct Point { double x, y;  ) ;  /\* returns 1 if intersect at a point, O if not, -1 if the lines coincide \*/ int intersect line(Point a, Point b, Point c, Point d, Point &p)  { -  **Point t;**  double r, s;  double denom, numl, num2;  assert((a.x != b.x I I a.y != b.y) && (c.x != d.x J I c.y != d.y));  numl = (a.y - c.y)\*(d.x - c.x) - (a.x - c.x)\*(d.y - c.y);  num2 = (a.y - c.y)\*(b.x - a.xi - (a.x - c.x)\*(b.y - a.y);  denom = (b.x - a.x)\*(d.y - c.y) - (b.y - a.y)\*(d.x - c.x);  if {fabs(denom) >= EPSILON) { r = numl / denom;  s = num2 *I* denom;  if CO-EPSILON <= r && r <= l+EPSILON &&  0-EPSILON <= s && s <= l+EPSILON) {  /\* always do this part if we are interested in lines instead \*/  /\* of linta segments \*/  p.x = a.x + r\*(b.x - a.x);  p.y = a.y + r\*(b.y - a.y); return 1;  ) else {  **return O ;**  )  } else {  if (fabs(numl) >= EPSILON) { return O;  } else {  /\* I am not using "fuzzy comparisons" here, because the comparisons \*/  /\* are based on the input, not some derived quantities. You may \*/  *I\** want to change that if the input points are computed **somehow.** \*/  **/\*.two lines are the 11 same 11 • See if they overlap \*/**  if (a.x > b.x 1 1 (a.x == b.x && a.y > b.y)) {  t :a  a = b  b = t  )  if (c.x > d.x 1 1 (c.x = d.x && c.y > d.y)) { | | | | |  | t c;  d t ·  if (a.x == b.x) {  /\* vertical lines \*/  if (b .y =• C .y) {  p - b;  return l;  P = a; ..  **return 1;**  return O; else { **return -1;**  }  else (  if (b.x == c.x)  **return l;**  else if (a.x == d.x) {  **return 1;**  return O; else { **return -l;**  return -1;  } }  int main (void)  {  Point a, b, c, d, p;  **int res;**  **while {cin >> a.x >> a.y >> b.x >> b.y** >> c.x >> c.y >> d.x >> d.y) res = intersect line(a, b, c, d, p);  if (res == 1) {-  **cout << ''Intersect at** (11 << **p.x** << 11 II << **p.y** << ") II << **enl;**  else if (res == O) {  **cout << 11 D0n' t intersect'• < < endl ;**  ) else {  **cout** << **11 Infinite number of intersections 11 << endl;**  } )  return O; | | | |

:CT.-/

*(* -·.

josephus. cc Page 111

,

*(!* '\

*/*

. ---·•,,\

Printed by Darcy *}*

Aug 19, 14 21:36

*II*

*II* Josephus Problem

*II*

*II* Author: Darcy Best

*II* Date : September 4, 2010

*II*

*II* The Josephus problem:

*II* A group of n people are in a circle, and you start by killing

*II* person f. Then, you kill every kth person until only one person

*II* is left.

*II*

*II* Two implementations are given here (Note that neither depend on k):

*II* 1. Determine the survivor -- O(n) *II* 2. Determine the full killing order -- O(nA2) *II*

*II* If there are 17 people, with every 5th person killed (killing the

*II* 1st person first), the kill order is:

*II* l,6,ll,16,5,12,2,9,17,10,4,15,1 4,3,8,13,7 (survivor = 7)

*II*

*II* NOTE: This is. 1-based, not 0-based.

#include <iostream>

**using namespace std;**

const int MAX\_N = 100;

int survivor(int n,int £,int k)(

return (n==l ? 1 : (survivor(n-1,k,k) + (f-1)) % n + l);

void killOrder(int n,int f,int k,int A(]){ if(n == O) return;

A [OJ = O;

killOrder(n-1,k,k,A+l);

**for(int i=O;i<n;i++)**

A[i] = (A(i] + (f-1)) % n + l;

int main() (

int n,f,k,kOrder[MAX NJ ;

while( cin >> n >> f;> *k* && (n J J f I I k)){ killOrder(n,f,k,kOrder);

for(int i=O;i<n;i++)

**cout << kOrder[i] << endl;**

**cout << "Survivor:** 11 << **survivor(n, f i k) << endl;**

}

**return *O ;***

Aug 19, 14 21:36 **kmp.cc** Page 1/2

*I\**

* KMP String Matching

\*

* Author: Howard Cheng

\*

* The prepare\_pattern routine takes in the pattern you wish to search
* **f or, and perform some processing to give a " f ailure array 11 to be ·used**
* by the actual search. The complexity is linear in the length of the
* pattern.
* The find\_pattern routine takes in a string s, a pattern pat, and a
* vector T computed by prepare\_pattern. It returns the index of the
* first occurrence of pat in s, or -1 if it does not occur in s.
* The complexity is linear in the length of the string s.

\*

*\*I*

#include <iostream>

**#include <string>**

#include <vector>

#include <algorithm>

**using namespace std;**

void prepare\_pattern( const string &pat, vector<int> &T)

{

int n = pat.length(); T.resize(n+l); fill(T.begin(), T.end(), -1);

for (int i = l; i <= n; i++) { int pos = T[i-1);

while (pos != -1 && pat[pos] != pat[i-1) ) { pos = T[pos) ;

)

T(i] = pos + l;

int find pattern(const string &s, const string &pat, const vector<int> &T)

-

{

int Sp = 0, kp = 0;

int slen = s.length(), plen = pat.length(); while (sp < slen) {

while (kp != -1 && (kp == plen I J pat[kp) != s [spJ )) {

kp = T(kp);

)

**kp++; Sp++;**

if (kp == plen) {

*II* a match is found return ·sp - plen;

*II* if you want more than one match (i.e. all matches), do not return *II* in the above but rather record the location of the match. Continue *II* the loop with:

*II*

*II* kp = T[kpJ ;

) )

**return -1;**

int main (void)

{

**sring str, pat{**

while (cin >> str >> pat) { vector<int> T; prepare\_pattern( pat, T);

cout << "index = " << finq\_pattern(str, pat, T) << endl;

Tuesday August 19, 2014 josephus.cc, kmp.cc 54/77

Printed by Darcy Best

Aug 19, 14 21:36 kmp.cc Page 2/2 Aug 19, 14 21:36 **linsolve.cc** Page 1/3

)

**return O ;**

*I\**

* Solution of systems of linear equations
* Author: Howard Cheng
* **Reference:**

K.E. Atkinson. "An Introduction to Numerical Analysis."

1. nd Ed. , John

Wiley & Sons, 1988, pages 520-521.

\*

ISBN 0-471-62489-6

* + To solve the system Ax = b where A is an n x n matrix, first call
  + LU decomp on A to obtain its LU decomposition. Once the LU
  + decomposition is obtained, it can be used to solve linear systems with
  + the same coefficient matrix A but different vectors of b using the
  + LU\_solve routine. This routine is numerically stable (provided that
  + the original coefficient matrix has a small condition number).

\*

* + The inputs to LU\_decomp are the matrix A, the dimension n, an
  + output array pivot of n-1 elements such that pivot[i) = j means
* that rows i and j were swapped during the i-th step, and an output

\* parameter to return the determinant of the matrix. The function

* returns 1 if successful, and O if the matrix is singular. The
* matrix A is overwritten by its LU decomposition on return. If the
* matrix is. singular, the content of A should not be used (it represents
* intermediate results during the decomposition). ·
* The inputs to LU\_solve are the LU decomposition of A, the dimension

t n, the pivot array from LU\_decomp, and n-dimensional vectors b and

\* x. This function should be called only if the original matrix A

* has a small condition number. You can check this by checking that
* the determinant returned by LU decamp is not too close to 0. This is

\* only a crude check: you should-really be computing the condition number

\* of the matrix.

•/

#include <iostream>

#include <cmath>

using namespace std; const int MAX\_N = 10;

i{ nt LU-decomp(double A[MAX-NJ [MAX-NJ , int n, int pivot[), dounle &det)

double s[MAX NJ ; /\* factors used in implicit scaling •/ double c, t;-

int i, j , k; det = 1.O ;

*I\** compute s[il \*/

for (i = O; i < n; i++) { s[il = o .o;

for (j = o ; j -< n; j ++) {

if ((t = fabs(A[i) [j) ) ) > s[i)l

s [i) = t;

}

if (s[il == 0 .0) {

*!•* a row of zeroes: singular \*/ det = 0 .0 ;

return O ;

*I\** do the row reductions \*/ for (k = O; k < n-1; k++) { c = fabs (A[kl [kl /s [ki );

pivot[k) = k;

**for (i = k+l; i < n; i++)**

'----

Tu(' August 19, 2014

*,,> )*

. *' 1.:*

"'1'11:\ ,,,.

*( .*

' ·

-k· \.:!:::'......

kmp.c9, linsolve.cc

: )

*S* "*l*.*i*,*i* ; .· - '

·•.



*r- '*

*2''*

*c;,*



.' "')

(

*)*

-· Printed by Oarcy *(*Be*J*st*.1*

Aug 19, 14 21:36

**linsolve.cc**

Page 3/3

x[n-1] /= A[n-l•l [n-ll ;

for (i = n-2; i >= O ; i--) { for (j = i+l; j< n; j++) (

x[il -= A[iJ [jJ \* x[jl ;

x[i] *I=* A[il [il ;

)

int main(void)

{

double A[MAX NJ [MAX 'Nl , x[MAX NJ , b[MAX NJ ;

int pivot[MAX\_Nl ; - -/\* only n=1 is needed, but what the heck \*/ int n, i, j;

double det;

while (cin >> n && O < n && n <= MAX\_N)

**cout << "Enter A:** 11 << **endl ;**

for (i = O ; i < n; i++) { for (j = O ; j < n; j++) {

cin » A[iJ [j];

) )

**cout << "Enter b:** 11 ;

**for (i = O;** i< **n; i++)**

cin >> b [iJ ;

)

if (LU\_decomp(A, n, pivot, det)) { LU solve(A, n, pivot, b, x);

**coUt << "LU decomposition of A:" << endl;**

for {i = O; i< n; i++) {

for (j = O; j < n; j++) { cout << A(i] (jl << " ";

)

**cout << endl;**

)

**cout << 11 det** = << **det << endl;**

11

**cout <<**

**11 x** = 11 ;

for (i = O; *i* < n; i++)

**cout << x(iJ** << u " ·

)

**cout << endl;**

else {

**cout << is singular 11 << endl;**

**11 A**

)

return O;

Aug 19, 14 21:36 linsolve.cc Page 2/3

t = f abs (A (i] [k] /s [i] ) ;

if (t > c) {

C = t;

pivot[k] = i;

if (c == O l {

/\* pivot == 0: singular \*/ det = 0.0;

return O;

/\* do row exchange \*/

if (k != pivot[k]) { det \*= -1. O ;

for (j = k; j < n; j++)

t = A [k] [j] ;

A[k] [j ] = A[pivot[k]J [j]; A[pivot[kl l [jl = t;

t = s [kl ;

s [kl = s [pivot [kl l ;

s (pivet [kl l = t ;

) }

/\* do the row reduction •/ for (i = k+l; i < n; i++) (

A[il [kl /= A[kl [kl ;

for (j = k+l; j < n; j++) { A[i]'[jl -= A[il [kl A[kl [j ] ;

det \*= A [kl [kl ;

/\* note that the algorithm as state in the book is incorrect. The \*/

/\* following is need to ensure that the last row is not all O's. •/ *I\** (maybe the book is correct, depending on what you think it's \*/ *I\** supposed to do.) •/

if (A [n-ll [n-lJ == o. O) (- det = 0.0;

**return O ;**

else {

det \*= A [n-lJ [n-11 ;

**return l;**

void LU solve(double A[MAX NJ [MAX NJ , int n, int pivot[), double b[l ,

- double x[l l - -

**double t:;**

int i, j , k;

for (i = O; i < n; i++) {

X [i] = b [i] ;

}

for (k = O; k < n-1; k++)

/• swap if necessary \*/

if (k != pivot[kl ) { t = x[kJ ;

x[kl = x[pivot[kl J ; x[pivot[kl l = t;

for (i = k+l; i < n; i++) x[iJ '-= A[iJ [kl •x[kl ;

Tuesday August 19, 2014

Hnsolve.cc 56/77

Printed by Darcy Best Aug 19, 14 21 :36 **mincostmaxflowdense.cc** Page 1/3 Aug 19, 14 21:36 **mincostmaxflowdense.cc** Page 2/3

*I\**

* Min Cost Max Flow for Dense graphs
* Authors: Frank Chu, Igor Naverniouk
* http:llshygypsy.comltoolslmcmf3.cpp
* Min cost max flow \* (Edmonds-Karp relabelling + Dijkstra)

\*

* This implementacion takes a directed graph where each edge has a
* capacity ('cap') and a cost per unit of flow ('cost') and returns a

d [s) = o ;

par[s] = -n - l;

while (1) {

*II* find u with smallest d(u] int u = -1, bestD = Inf;

for (int i = o ; i < n; i++) {

if (par[i] < o && d[i] < bestD) bestD = d[u = i];

* maximum flow network of minimal cost ['feast') from s to t. }

if (bestD == Inf) break;

* PARAMETERS:

cap (global): adjacency matrix where cap[u] [v) is·the capacity

*II* relax edge (u,i) or (i,u) for all i;

* of the edge u->v. cap [u) (v] is o for non-existent edges. cost (global): a matrix where cost [u) [v] is the cost per unit

of flow along the edge u->v. If cap [u)[v] == O , cost [u][v] is

* ignored. ALL COSTS MUST BE NON-NEGATIVE!
  + n: the mlmber of vertices (( 0 , n-1] are considered as vertices).
  + **s: source vertex.**
  + t: sink.

par[u] = -par[u] - l;

for (int i = O; i < deg[u); i++l

*II* try undoing edge v->u int v = adj [u)(i];

if (par[v) >= O ) continue;

if (fnet [v] [u] && d [v] > Pot (u,v) - cost [v) [u]) ( d[v) = Pot( u, v ) - cost[v] [u);

* RETURNS:
  + the flow
  + the total cost through 'feast'
  + fnet contains the flow network. Careful: both fnet [u] (v] and fnet[v) [u] could be positive. Take the difference.
* COMPLEXITY:
* - **Worst case: O(nA2\*flow <? nA3\*fcost)**
* REFERENCE:

Edmonds, J . , Karp, R. "Theoretical Improvements in Algorithmic Efficieincy for Network Flow Problems".

* This is a slight improvement of Frank Chu's implementation.

par[v] = -U-1;

*II* try edge u->v

if (£net[u] (v] < cap [u) [v] && d [v] > Pot (u,v) + cost [u][v)) { d[v] = Pot(u,v) + cost[u] [v] ;

par[v] = -u - l;

} }

*\*\*I*

#include <iostream>

#include <algorithm>

#include <climits>

**using namespace std;**

*II* the maximum number of vertices + 1

const int NN = 1024;

*II* adjacency matrix (fill this up)

for (int i = O; i < n; i++) { if (pi[i] < Inf) {

pi [i] += d[i];

} )

return par[t) >= O;

#undef Pot

int cap [NN) [NN] ; int mcmf( int n, int s, int t, int &feast )

*II* cost per unit of flow matrix (fill this up) int cost [NN] [NN];

*II* flow network and adjacency list

int fnet [NN] [NN) , adj [NN] [NN), deg (NN];

*II* Dijkstra's successor and depth

{ *II* build the adjacency list fill(deg, deg+NN, O);

for (int i = O; i < n; i++) { for (int j = o ; j < n; j++ ) {

if (cap[i] [j] 1 1 cap[j] [i]) adj [i] [deg [i]++] = j;

int par[NN], d[NN); *II* par[source]

*II* Labelling function int pi[NN];

**source;**

for (int i = O ; i < NN; i++) { fill (fnet [i], fnet [i]+NN, O ) ;

const int Inf = INT\_MAXl2;

*II* Dijkstra's using non-negative edge weights (cost + potential)

#define Pot (u,v) (d(u] + pi (u] - pi (v])

bool dijkstra(int n, int s, int t)

{

for (int i = O ; i. < n; i++) ( d[i) = Inf;

par[i] = -1;

*y* ,y August 19, 2014

\\..\_ . ·. );

}

fill(pi, pi+NN, 0);

int flow = feast = O;

*II* repeatedly, find a cheapest path from s to t while (dijkstra(n, s, t)) {

*II* get the bottleneck capacity

int bot = INT MAX;

for (int v = t, u = par[v]; v != s; u = par[v = u] ) { .

bot = min (bot, fnet [v] [u) ? fnet [v] [u] : (cap[u][v] - fnet [tl] [v]));

}

mincostmaxflowdense. cc

<..

"--7·1·

*i::·*

*!'* .*::::;1'\*

'--=-,'

·

...\_. . :..-·

*j*

-. .,,,...

,..v

*"'°'·{\......J*



'7,

Aug 19, 14 21:36

mincostmaxflowdense.cc

Page 3/3

*II* update the flow network

for (int v = t, u = par[v]; v != s; u = par[v = u]) (

if (fnet[v] [u)) {

fnet [v] [u] -= bot;

fcost -= bot \* cost [v] [u]; else {

fnet [u) [v] += bot;

fcost += bot \* cost [u][v] ;

} }

flow += bot;

return flow;

//---------------- EXAMPLE USAGE

**#include <iostream>**

using namespace std;

int main()

(

**int numV; cin >> numV**

for (int i O; i < NN; i++) { fill(cap[ ], cap[i]+NN, O);

**int m, a, b, c,** cp;

**int S, t;**

**cin >> m;**

**cin >> s >> t;**

*II* fill up cap with existing capacities.

*JI* if the edge u->v has capacity 6, set cap[u] [v) = 6.

// for each cap [u] [v] > o, set cost [u] [v] to the

*JI* cost per unit of flow along the edge i->v for (int i=O; i<m; i++) {

**cin >> a >> b >> *cp* >> c;**

cost(a] [bl = c; // cost[b] [a] = c;

cap[a] [bl = cp; *II* cap[b) [al = cp;

int feast;

int flow = mcmf( numv, s, t, feast );

**cout** << 11 **f low:** << **f low *<<* endl ;**

11

**cout** << 11 **cost :** << **f cost << endl ;**

11

return O;

,

-

*(* ,I

/ /

Aug 19, 14 21:36 **.mincostmaxflowsparse.cc**

*I\*\**

. -

Printed by *Darcy )*

Page 1/3

*JIIJJIIJ /IIII/IJ JJ IJJI/*

// MIN COST MAX FLOW //

\* *J /JJJIII//J//JJ JJJ/JJII*

* Authors: Frank Chu, Igor Naverniouk

*\*\*I*

*!\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**

* Min cost max flow \* (Edmonds-Karp relabelling + fast heap Dijkstra)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

* Takes a directed graph where each edge has a capacity ('cap') and a
* cost per unit of flow ('cost') and returns a maximum flow network
* of minimal cost ('fcost') from s to t. USE mcmf3.cpp FOR DENSE GRAPHS!
* PARAMETERS:

cap (global): adjacency matrix where cap[u) (v] is the capacity of the edge u->v. cap[u] (v] is o for non-existent edges.

* + cost (global): a matrix where cost[uJ (v) is the cost per unit
* of flow along the edge u->v. If cap (u][v) == O, cost [u) [v) is
* ignored. ALL COSTS MUST BE NON-NEGATIVE!
  + n: the number of vertices ((0, n-1] are considered as vertices).
* - **s : source vertex.**
  + t: sink.
* RETURNS:
  + the flow
  + the total cost through 'feast'
  + fnet contains the flow network. careful: both fnet[u) [v) and fnet [v] [u] could be positive. Take the difference.
* COMPLEXITY:
* - Worst case: O(m\*log(m)\*flow <? n\*m\*log(m)\*fcost)
* FIELD TESTING:
* - Valladolid 10594: Data Flow
* REFERENCE:

Edmonds, J. , Karp, R. "Theoretical Improvements in Algorithmic Efficieincy for Network Flow Problems".

* This is a slight improvement of Frank Chu's implementation.

\*\*/

#include <iostream>

#include <algorithm>

#include <climits> using namespace std;

*JI* the maximum number of vertices + 1

#define NN 1024

*JI* adjacency matrix (fill this up) int cap[NNJ [NN];

*II* cost per unit of flow matrix (fill this up) int cost [NN][NN) ;

*JI* flow network and adjacency list

int fnet[NN] (NN], adj [NN] [NN], deg[NN] ;

*J j* Dijkstra' s predecessor, depth and priority queue int par[NN), d(NN), q[NN], inq[NN] , qs;

*II* Labelling function int pi(NN] ;

#define Inf (INT MAX/2)

#define BUBL (\-

t - q[i]; q[i] = q[j); q[j) = t; \

t = inq[q[i)]; inq[q[i)] = inq[q[j]J ; inq(q[j)J = t; )

*II* Dijkstra' s using non-negative edge weights (cost + potential)

#define Pot\_(u,v) (d(u] + pi [u) - pi (v] )

Tuesday August 19, 2014 mincostmaxflowdense. cc, mincostmaxflowsparse. cc 58/77

Printed by Darcy Best Aug 19, 14 21:36 **mincostmaxflowsparse.cc** Page 2/3 Aug 19, 14 21:36 **mincostmaxflowsparse.cc** Page 3/3

bool dijkstra( int n, int s, int t )

{

f ll(d, d+NN, Inf);

f ll(par, par+NN, -1) ; f ll(inq, \_ inq+NN, -1);

d [s] = qs = o ; inq[q[qs++] = s] = O; par[s] = n;

*II* update the flow network

for (int v = t, u = par[v]; v l = s; u = par(v = u] )

if (fnet[v] [u] ) { fnet [v] [u] -= bot; feast -= bot \* cost [v] [u) ;

else { fnet [u) [v) += bot; fcost += bot \* cost [u] [v]; )

flow += bot;

return flow;

while (qs) {

*II* get the minimum from q and bubble down int u = q [O J ;

inq(u] = -1;

q[O] = q[--qs];

if ( qs ) inq[q[O]] = O;

for (int i = 0, j = 2\*i + l, t; j < qs; i= j, j if ( j + 1 < qs && d(q[j + l]] < d(q[j]J ) j++;

if (d[q[j]l >= d(q[i]]) break;

BUBL ;

*II* relax edge (u,i) or (i,u) for all i;

2\*i + 1) {

int main()

(

int numV;

**int m, a, b , c, cp; int s, t;**

**cin >> numV;**

**cin >> m;**

**cin >> s >> t;**

*II* fill up cap with existing capacities.

for (int k = O , v = adj(u][k]; k < deg[u]; v = adj(u] [++k] )

*II* try undoing edge v->u

if (fnet [v] [u] && d [v] > Pot (u,v) - cost [v] (u])

d [v] = Pot (u,v) - cost [v][par[v] = u] ;

*II* if the edge U->v has capacity 6, set cap(u] [v] 6.

*II* for each cap [u] [v] > o, set cost [u][v] to the

*II* cost per unit of flow along the edge u->v for (int i=O; i<m; i++] {

**cin >> a >> b >> cp >> c;**

*II* try using edge u->v

cost[a] [bl = c; *II*

cost[b] [a] = c;

if (fnet[u][v) < cap [u] [v] && d [v] > Pot (u,v) + cost [u] [v]) d [v] = Pot (u,v) + cost [par [v] = u] [v] ;

if (par[v] == Lt) (

*II* bubble up or decrease key

if( inq[v] < O ) { inq[q[qs] = v] = qs; qs++;

for( int i = inq[v], j = ( i - 1 *)12,* t; d(q[i]] < d(q[j]]; i = j, j = ( i - 1 *) 12*

BUBL;

cap[a] [bl = cp; *II* cap[b] (a] = cp;

int fcost;

int flow = mcmf( numV, s, t, feast );

**cout << 11 flow: 11 << flow < < endl; cout << ''cost:** " << **fcost << endl;**

**return O ;**

} .

for( int i = O; i c n; it+ ) if( pi(i] < Inf ) pi(i] += d[i] ; return par[t] >= O;

int mcmf( int n, int s, int t, int &feast )

{

*II* build the adjacency list fill(deg, deg+NN, O);

for (int i = O; i c n; i++ ) {

for (int j = O; j < n; j++)

if (cap[il [jJ 1 1 cap [j l [i]) adj [i] [deg[i]++] j;

}

for (int i = O; i < NN; i++) {

fill (fnet [il , fnet (i]+NN, 0);

)

fill(pi, pi+NN, 0);

int flow = fcost = O;

*II* repeatedly, find a cheapest path from s to t

while (dijkstra(n, s, t)) {

*I I* get the bottleneck capacity int bot = INT MA. ;

for (int v = t, 1J. = par[v]; v != s; u = par[v = u]) {

bot = min (bot, fnet [v](u] ? fnet (v] (u] : ( cap (u][v] - fnet [u][v] ));

*Tue{* . August 19, 2014

I

031

mincostmaxflowsparse.cc *j*

-J

5g;..

·:,'-/

''···1, *:i*

..>,

'·•.,\_.

*(*

CC, .· l

Aug 19, 14 21:36 mst.cc Page 1/3

*I\**

* Implementation of Kruskal's Minimum Spanning Tree Algorithm

\*

* Author: Howard Cheng
* This is a routine to find the minimum spanning tree. It takes as
* input:

\*

uf[res2] .p = resl;

)

else {

uf[resl] .p = res2;

if (uf[resl) .rank == uf[res2) .rank) { uf[res2] .rank++;

)

**return true;**

)

return false;

private:

**int howMany;**

UF\* uf;

int find(UF uf[J , int x)

if (uf[x].p != x) {

uf[x] .p = find(uf, uf[x) .p);

*II tor* internal use

)

return uf(x].p;

) ;

class Edge { public:

Edge(int i=-1, int j=-1, double weight=O) {

**Vl = i ;**

v2 = j ;

w = weight;

}

bool operator<(const Edge& e) canst { return w < e.w;

*I\** two endpoints of edge

*\* I*

) ;

**int vl, v2;**

double w;

*I\** weight, can be double instead of int *\* I*

double mst(int n. int m, Edge elist[], int index(], int& size)

{

UnionFind uf(n);

sort(elist, elist+m); double w = 0.0;

**size = *D i***

for (int i = O; i < m && size < n-1; i++) { int cl = uf.find(elist[il .vl);

int c2 = uf.find(elist[i) .v2); if (cl != c2) {

index(size++) = i; w += elist[i] .w; uf.merge(cl, c2);

) )

**return w;**

int main (void)

{

**cout << fixed << setprecision(2};**

int n;

**cin >> n;**

double\* x new double[n]; double\* *y* new double[n];

Aug 19, 14 21:36

**mst.cc**

Page 2/3

*II*



\:::::;,



Printed by Darcy

(-.j'

-

Bes?'

n: number of vertices m: number of edges

elist: an array of edges (if (u,v) is in the list, there is no need for (v,u) to be in, but it wouldn' t hurt, as long as the weights are the same).

* The following are returned:

index: an array of indices that shows which edges from elist are in the minimum spanning tree. It is assumed that its size is at least n-1.

**size: the number of edges selected in 11 index 11 • If this is not** n-1, the graph is not connected and we have a "minimum spanning forest."

* The weight of the MST is returned as the function return value.
* The run time of the algorithm is O(m log m).
* Note: the elements of elist may be reordered.
* Modified by Rex Forsyth using C++ Aug 28, 2003
* This version defines the unionfind and edge as classes and provides
* constructors. The edge class overloads the < operator. So the sort does
* not use a \* cmp function. It uses dynamic arrays.

*•I*

#include <cmath>

#include <iostream>

#include <iomanip>

#include <Cstdlib>

**#include <cassert>**

#include <algorithm> using namespace std;

**class UnionFil"ld**

{

struct UF { int p; int rank; );

public:

UnionFind (int n) { *II* constructor howMany = n;

uf = new UF[howMany);

for (int i = O; i < howMany; i++) { uf[i) .p = i;

uf [i) .rank = O;

-UnionFind() { delete (] uf;

int find(int x) { return find(uf.x); *II* for client use bool merge(int x, int y)

**int resl, res2;** resl = find(uf, x); res2 = find(uf, y); if (resl I = res2) {

if (uf[resl] .rank > uf[res2).rank) {

Tuesday August 19, 2014 mst.cc

60/77



Aug 19, 14 21:36 mst.cc Page 3/3

**int\* index** new int[n];

for (int i O; i < n; i++) cin » x[i) » y[i) ; Edge\* elist = new Edge [n\*n);

Aug 19, 14 21:36 **muIt.cc**

*I\**

* Multiplicationidivision without overflow
* Author, Howard Cheng

Printed by Darcy Best

Page 1/1

**int k = o ;**

**for (int i = O;** i < **n; i++)**

for (int j = i+l; j < n; j++)

elist[k++] = Edge(i,j,hypot(x[i]-x[ j], y[i)-y[j)) );

int t; *II* number of edges in the mst

cout << mst(n, k, elist, index, t) << endl; return O;

* Given a list of, factors in the numerator (num, size n) and a list
* **of factors in the denominator {dem, size m), it returns the product**
* of the numerator divided by the denominator, while reducing the
* result as soon as it is larger than some BOUND.

*\* I*

**#include <iostream>**

#include <Gassert>

using namespace std;

canst int BOUND = (1 << 16);

double mult(double num[J , int n, double dem[], int m)

{

int i, j;

double prod = 1.0;

i= j = O ;

while H < n 1 1 j < m) {

if (prod >= BOUND && j < m) ( prod *I=* dem(j++);

else if (i < n) {

**prod \*= num(i++];**

else { assert(j < m);

prod *I=* dem(j++];

return prod; int main (void)

{

double A[lOOO], B[lOO O];

**int n, m, i ;**

while (cin >> n >> m && n > O && m > 0) {

for **(i = O ; i < n;** i++) {

cin >> A[i);

}

for {i = O; **i < m;** i++) {

cin >> B(i];

}

**cout << "prod** " << mult (A, n, B, m) << endl; return O;

*1* iay August 19, 2014

*f :::: .*

*'<.,;:;;;;:.!*

*.........::......·*

l3;

mst.cc, mult.cc *r'f*

*. i*

··'"

*l'' \*

,...,,.-..,, v





2:· ' . '

Aug 19, 14 21:36 networkflow.cc Page 1/3

*I\**

* Network Flow
* Author, Howard Cheng

\*

* The routine network flow() finds the maximum flow that can be
* pushed from the source (s) to the sink (t) in a flow network

Aug 19, 14 21:36

-Graph()

{

delete[] nbr;

.;

*I,* ')

*/*

**networkflow.cc**

*\*

Printed by Dar:

Page 2/3

* (i.e. directed graph with capacities on the edges). The maximum
* flow is returned. Note that the graph is modified. If you wish,to
* **recover the flow on an edge, it is in the "flow" field, as long as**
* is\_real is set to true.
* Note: if you have an undirected network. simply call add\_edge twice
* with an edge in both directions (same capacity). Note that 4 edges
* will be added (2 real edges and 2 residual edges). To discover the
* actual flow between two vertices u and v, add up the flow of all

·• real edges from u to v and subtract all the flow of real edges from

* v to u. (In fact, for a residual edge the flow is always O in this
* implementation. )

\*

* This code can also be used for bipartite matching by setting up an
* appropriate flow network.

\*

* The code here assumes an adjacency list representation since most
* problems requiring network flow have sparse graphs.
* This is the basic augmenting path algorithm and it is not the most
* efficient. But it should be good enough for most programming contest
* problems. The complexity is O(f m) where f is the size of the flow
* and m is tthe number of edges. This is good if you know that f
* is small, but can be exponential if f is large.

*\*I*

#include <iostream>

#include <algorithm>

#include <vector>

**#-include <list>**

#include <cassert> using namespace std;

struct Edge;

typedef list<Edge>::iterator Edgelter;

struct Edge

**int to;**

int cap; int flow;

bool is real; Edgeiter partner;

Edge(int t, int c, bool real = true)

: to(t), cap(c), flow(O), is\_real(real)

{);

int residual() canst

{

return cap - flow;

)

) ;

struct Graph { list<Edge> \*nbr; **int num nodes;** Graph(int n)

num\_nodes(n)

nbr = new list<Edge>[num\_nodes] ;

Tuesday August 19, 2014

*II* note: this routine adds an edge to the graph with the specified capacity, *II* as well as a residual edge. There is no check on duplicate edge, so it *II* is possible to add multiple edges (and residual edges) between two

*II* vertices

void add\_edge(int u, int v, int cap)

{

nbr[u) .push\_front(Edge(v, cap)); nbr[v) .push\_front(Edge(u, o, false));

nbr[v] .begin()->partner = nbr[u] .begin();

nbr[u] .begin()->partner = nbr[v] .begin();

)

} ;

void.push path(Graph &G, int s, int t, canst vector<Edgeiter> &path, int flow)

-

{

for (int i = O; s != t; i++) if (path[i)->is real) {

oath[i)->flow-+= flow; path[i]->partner->cap += flow; else {

path[i)->cap -= flow;

path[i)->partner->flow -= flow;

s = path[i]->to;

*II* the path is stored in a peculiar way for efficiency: path[i] is the

*II* i-th edge taken in the path.

int augmenting path(const Graph &G, int s, int t, vector<Edgeiter> &path, vector<bool> &visited, int step = O)

if (s == t) (

return -1;

)

for (Edgeiter it = G.nbr(s].begin(); it != G.nbr[s].end(); ++it)

**int V = it->tO;**

if (it->residual () > 0 && !visited [v] ) { path[step] = it;

visited[v] = true;

int flow = augmenting path(G, v, t, path, visited, step+l); if (flow == -1) (

return it->residual(); else if (flow > 0) (

return min (flow, it-·>residual());

**return O ;**

*II* note that the graph is modified

int network flow(Graph &G, int s, int t)

-

{

vector<bool> visited(G.num nodes); vector<Edgeiter> path(G.num\_nodes); int flow = 0, f;

do {

fill(visited.begin(), visited.end(), false);

if ((f = augmenting path(G, s, t, path, visited)) > 0) {

push\_path(G,' s, t, path, f);

networkflow.cc *62177*



Aug 19, 14 21:36 **networkflow2.cc**

Aug 19, 14 21:36

networkflow.cc

Page 3/3

flow += f;

)

while ( f > 0);

return flow;

**int main(void)**

{

Graph G (1o a) ;

int s, t, u, v, cap, flow;

**cin >> s >> t;**

**while (cin > > u > > v >> cap)**

) G.add-edge(u, v, cap);

flow = network flow(G, s, t);

**cout << "maximUm flow == << flow < < endl;**

**11**

**return o ;**

*!•*

* Network Flow (Relabel-to-front)

\*

* Author: Howard Cheng

Printed by Darcy Best

. Page 1/3

* The routine network flow() finds the maximum flow that can be
* pushed from the source (s) to the sink (t) in a flow network
* (i.e. directed graph with capacities on the edges). The maximum
* flow is returned. The flow is given in the flow array (look for
* positive flow).
* The complexity of this algorithm is O(nA3), which is good if the
* graph is small but the maximum flow can be large. Since the
* algorithm is O(nA3) we are going to use the adjacency matrix
* representation.

\*

\*/

#include <iostream>

**#include <list>**

**#include <cassert> using namespace std;**

canst int MAX\_NODE = 102;

v{oid clear-graph(int graph[MAX-NODE] [MAX-NODE], int n) for (int i = O ; i < n; i++) {

for (int j = O ; j < n;\_ j++) { graph[il [j] = O ;

) }

void push(int graph[MAX\_NODE] [MAX\_NODE], int flow[MAX\_NODE] [MAX\_NODE], int e [], int u, int v)

int cf = graph [u] [v] - flow [u)[v) ;

int d = (e[u] < cf) ? e[u] : cf; flow [u] [v] += d;

flow[v] (u] = -flow(u] [v] ; e[u] -= d;

e [v] += d;

void relabel(int graph[MAX NODE] [MAX NODE] , int flow[MAX NODE] [MAX NODE), int n, int h[J, int u) - - -

h[u) = -1;

for (int v = O; v < n; v++) {

if (graph (u] [v] - flow [u] [v] > 0 &&

(h[u] == -1 1 1 1 + h[v] < h[u]))

h[u] = 1 + h[v) ;

} }

assert(h[u] >= O);

Tu(;? · ·August 19, 2014

\:::.,..,

'"-,w."£"" .,·

*0::))*

'·..-. :.--

void discharge(int graph[MAX NODE]·[MAX NODE), int flow[MAX NODE) [MAX NODE], int n, int e( , int h(] list<int>& NU, - - **list<int>: :iterator &current r int u)**

while (e(u] > 0) {

if (current == NU.end{)) { relabel(graph, flow, n, h, u); current = NU.begin( );

} else {

**int v = \*current;**

networkflow .cc, networkflow2.cc

*( e):*

*Bf , '\*

*· ,\. }*



*r-'' .*

*)/*

,- *rJ*

*/*

*/*

-.,,\_\\

*(* t

Printed by Darcy B /

Aug 19, 14 21:36 networkflow2.cc Page 2/3

} }

int main (void)

(

int graph[MAX\_NODE) [MAX\_NODE];

**int s, t;**

**int n, m, u, v, c;**

int flow[MAX NODE] [MAX NODE];

int maxflow;-

-

while (cin >> n && n > 0) clear graph(graph, n); **cin > m >> s >> t;** while (m-- > 0) (

graph[u) [v] = c;

**cin** >> u >> **v >> c;**

)

ma."Cflow = network flow(graph, flow, n, S, t) ; **cout << ''f low =** ,.-<< **maxf low << endl ;** print\_flow(flow, n);

return O;

Aug 19, 14 21:36

**networkflow2.cc**

Page 3/3

if (graph[u] [v] - flow [u) [v] > O && h [u] == h [v] + 1) { push(graph, flow, e, u, v);

else (-

**++currerit;**

} }

int network flow(int graph[MAX NODE] [MAX NODE], int flow[MAX NODE] [MAX NODE] ,

- **int n, int s,-int t)** - - -

int e(MAX NODE]' h[MAX NODE];

int u, v,-oh; -

list<int> N[MAX NODE], L;

list<int>: :iterator current[MAX\_NODE] , p;

for (u = O; u < n; u++) h[u] = e(u] = O ;

}

for (u = O; u c n; u++) (

for (v = O; v < n; v++) flow [u] [v] = 0;

if (graph[u] [v] > 0 1 1 graph[v] [u] > 0) {

N(u] .push front(v);

J -

}

h[s] = n;

for (u = O; u c n; u++) { if (graph [s] (ul > O) {

e [u] = flow [s] [u] = graph [s] [u] ;

*e* [s] += flow [u](s] = -graph (s] (u];

}

if (u != s && u != t) {

L.push front(u);

-

}

current[u] = N[u] .begin();

p = L.begin();

while (p != L.end())

u = \*p; oh = h(u];

discharge(graph, flow, n, e, h, N[u], current[u], u);

if (h(u] > oh) {

L.erase(p); L.push\_front(u); p = L .begin ();

++p;

**int maxflow = O;**

for (u = O; u c n; u++) {

if (flow(s] [u] > 0) { maxflow += flow [s] [u] 1

)

**return maxflow;**

void print flow (int flow [MAX NODE] [MAX NODE] , int n)

- - -

{

for (int i = O;'i < n; i++) (· for (int j = O; j < n; j ++) (

if (flow[i]( j ] > 0) ·(-

**cout < < i**'<< 11 -> 11 << j << " : 11 << **f low [i) [j] << endl ;**

Tuesday August 19, 2014 networkflow2.cc 64/77

Aug 19, 14 21:37 pointpoly.cc Page 1/2 Aug 19, 14 21:37 **pointpoly.cc** Page 2/2

/ \*

\* Point-in-polygon test

\* Author: Howard Cheng

\* Reference:

<http://www.exaflop.org/docs/cgafaq/cga2.html>

*I\** now check if it's on the boundary \*/

·for (i = O; i < n-1; i++) {

if (ccw(poly[i], poly(i+l], p) == CNEITHER) return BOUNDARY;

}

\* **Given a polygon as a list of n vertices, and a point, it returns**

\* whether the point is in the polygon or not.

\* One has the option to define the behavior on the boundary.

if (ccw(poly(n-1) , poly(O], p) return BOUNDARY;

CNEITHER) {

\*/ .

#include ciostream>

#include ccmath>

#include ccassert>

/\* finally check if it's inside \*/

for (i = 0, j = n-1; i < -n; j = i++) {

if (((poly[i] *.y* c= p.y && p.y c poly(j] .y) I I

(poly(j] *.y* <= p.y && p.y c poly[i).y)) &&

(p.x c (poly(j] .x - poly(i] .x) \* (p.y - poly(il .y)

*I* (poly[j] .y - poly[i) .y) + poly[i) .x))

**C '= ! C;**

**using namespace std;** }

**return c;**

*I\** how close to call egual \*/

canst double EPSILON = lE-8;

/\* what should be returned on the boundary? \*/ canst bool BOUNDARY = true;

struct Point { double x, y;

} ;

/\* **counterclockwise, clockwise, or undefined \*/**

enum Orientation {CCW, CW, CNEITHER};

int main(void)

{ .

Point \*polygon, p; int n;

**int i ;**

while (cin >> n && n > O) polygon = new Point[nl ; assert(polygon);

for (i = O; i c n; i++) {

cin >> polygon[i).x >> polygon[i) .y;

Orientation ccw(Point a, Point b, Point c)

(

double dxl = b.x - a.x double dx2 = c.x - b.x double dyl = b.y - a-. y double dy2 = c.y - b.y double tl = dy2 \* dxl;

double t2 = dyl \* dx2;

if (fabs (tl - t2) c EPSILON) {

if (dxl \* dx2 c O 1 1 dyl \* dy2 c 0 ) {

if (dxl\*dxl + dyl\*dyl >= dx2\*dx2 + dy2\*dy2 - EPSILON) return CNEITHER;

} else ( return CW;

}

while (cin >> p.x >> p.y) {

if (point\_in\_poly( polygon, n, pl l {

**cout << 11 yes";**

} else (

**caut << 11 no 11 ;**

} .

**cout << endl;**

}

delete[) polygon;

}

**return O ;**

}

} else { return CCW;

}

else if (tl > t2) ( return CCW;

} else {

return CW;

J }

b{ool point-in-poly(Point poly[), int n, Point p)

**int i , j , C O;**

/\* first check to see if point is one of the vertices \*/ for ( i = O ; ic n; i++) {

if (fabs(p.x - poly(i] .x) C EPSILON && fabs(p.y - poly(i] .y) C EPSILON) { return BOUNDARY;



':day August 19, 2014

"""--,if-: -

-

',,c: ./

pointpoly.cc

\_ .*'*.\_*--=.."/)\* ·•

i, ·,

*'*· *i)*

.....r,:.c





,

W

G*r"·*.*.* 6' ;- l

*(* ," *i*

*\:::::::Y /* .

Aug 19, 14 21:37 polygon\_ **inter.cc** Page 1/5

*I\**

\* Convex Polygon Intersection

\* Author: Howard Cheng

\* This routine takes two convex polygon, and returns the intersection

* which is also convex. If the intersection contains less than
* **3 points, it is considered empty.**

*\*I*

**#include <iostream>**

#include <iomanip>

#include <cmath>

#include <algorithm>

#include <cassert> using namespace std;

*- I\** how close to call equal •/

const double EPSILON = lE-8;

struct Point { double x, y;

} ,

Printed by Darcy Best

Aug 19, 14 21:37 **polygon inter.cc** Page 2/5

hull[OJ = polygon[oJ ;

**return 1;**

/\* find the first point: min y, and then min x \*/ start\_p = polygon[O) ;

best *i* = O;

for i = l; i < n; i++) {

if ((polygon [i) .y < start p .y) 1 1

{polygon[i] .y == start\_p.y && polygon[ i) .x < start\_p.x)) { start\_p = polygon[i);

**best\_i = i;**

) }

polygon[best\_i) = polygon[ O) ; polygon[ O) = start\_p;

/\* get simple closed polygon •/ sort(polygon+l, polygon+n, ccw\_cmp);

/\* do convex hull \*/

**count = O;**

hull[count) = polygon[count]; count++; hull[count) = polygon[count] ; count++; for (i = 2; i < n; i++) (

while (count > 1 &&

canst bool BOUNDARY = true;

/\* ounterclockwise, clockwise, or undefined \*/ enum Orientation {ccw, CW, CNEITHER};

*!•* Gl;bal point for computing convex hull \*/

}

ccw(hull[count-2), hull[count-1], polygon[i) )

*I\** pop point •/

**count--;**

}

hull[count++) = polygon[ i];

CW) { ,

Point start\_p;

Orientation ccw(Point a, Point b, Point c)

{

double dxl = b.x - a.x

double dx2 = c.x -.b.x double dyl = b.y - a.y

double dy2 = c.y - b.y double tl = dy2 \* dxl;

double t2 = dyl • dx2;

if (fabs(tl - t2) < EPSILON) {

if (ctx1 • ctx2 < o I I dy1 • dy2 < o) {

if (dxl\*dxl + dyl\*dyl >= dx2\*dx2 + dy2\*dy2 - EPSILON) { return CNEITHER;

} else { return CW;

}

} else {

**return CCW;**

**return count;**

bool point in\_poly(Point poly(], int n, Point p)

-

{

**int i, j , c = O ;**

/\* first check to see if point is one of the vertices \*/ for (i = O; i < n; i++) {

if (fabs(p.x - poly(i] .x} < EPSILON && fabs(p.y - poly[i) .y) < EPSILON} ( return BOUNDARY;

/\* now check if it's on the boundary \*/ for (i = 0 : i < n-1; i++) {

if (ccwlpoly[ij , poly[i+l] , p) == CNEITHER) return BOu"NDARY;

}

}

else if (tl > t2) ( return CCW;

else {

if (ccw(poly[n-1), poly[O], p) return BOUNDARY;

CNEITHER) (

return CW;

} }

bool ccw\_cmp(const Point &a, canst Point &b)

{

return ccw(start\_p, a, b) == CCW;

}

/\* finally check if it's inside \*/

**for Ci = o , j = n-1; i < n; j ""' i++)** {

if (((poly(i] .y <= p.y.&& p.y < poly[j) .y) 1 1

(poly[j].y <= p.y && p.y < poly[i) .y)) &&

(p.x < (poly[j].x - poly[i) .x) \* (p.y - poly[i] .y)

*I* (poly[j] .y - poly[il .y) + poly[i] .x))

**C = *! C j***

}

int convex\_hull(Point polygon[ ), int n, Point hull[]) (

**int count, best\_i, i;**

if (n == 1) {

Tuesday August 19, 2014

**return c;**

*I\** retui-ns 1 if intersect at a point, o if not, -1 if the lines coincide \*/ int intersectlirie(Point a, Point b, Point c, Point d, Point &p)

\_ polygon\_inter.cc

66/77

Aug 19, 14 21:37 eo\_lygon\_i nter.cc Page 3/5

double r, s;

double denom, numl, num2;

numl *=* (a.y - c.y)\*(d.x - c.x) - (a.x - c.x)\*(d.y - c.y);

num2 = (a.y - c.y)\*(b.x - a.x) - (a.x - c.x)\*(b.y - a.y);

denom = (b.x - a.x)\*(d.y - c.y) - (b.y - a.y)\*(d.x - c.x);

if (fabs(denom) >= EPSILON) {

**r = numl / denom;**

s = num2 *I* denom;

if (-EPSILON <= r && r <• l+EPSILON && -EPSILON <= s && s <= l+EPSILON) (

p.x *=* a.x + r '(b.x - a.x);

P·Y • - Y· + r•(b.y - a.y); return 1;

) else (

**return O ;**

)

) else {

if (fabs(numl) >= EPSILON) (

**return o ;**

) else (

**return -1;**

) )

)

int intersect polygon(Point polyl[ ), int nl, Point poly2[), int n2,

- Point \*&out) ·

Point \*newpoly, p;

**char \*used;**

**int new n = nl + n2 + nl\*n2;**

**int couilt, i, i2, j ,** j2, **new\_caunt; int n;**

newpoly = new Point[new\_n]; out = new Point[new n]; used = new char[new-n];

assert(newpoly && out && used); count = O ;

fill(used, used+new\_n, O);

for (i *=* a ; i < nl; i++) (

if (point\_in\_poly( poly2, n2, polyl[i])) ( newpoly[count++l *=* polyl(i];

) )

for (i *=* o ; i < n2; i++) (

if (point\_in\_pol.y(polyl, nl, poly2 [i])) ( newpoly[count++] = poly2[i);

J )

for (i *=* O ; i < nl; i++) {

i2 = (i+l == nl) ? o : i+l; for (j = o; j < n2; j ++) {

j2 *=* (j+l *==* n2) ? O : j+l;

if (intersect\_line (polyl(i], polyl (i2], poly2 (j], poly2 (j2], p)

**newpoly{count++] = p;**

== 1) (

) )

if (count >= 3) {

**n *=* convex\_hull{newpoly, count, out);**

if (n < 3) {

delete[] out;

n = **O ;**

*T\J* ·,y August 19, 2014

J '•... •.

' .'

*]*

' •\.l1\,,.;,..'··.

··'

polygon\_inter.cc 5·

'b·

"·

''\*·,,fJ

······?

Aug 19, 14 21:37

}

else { delete[] out; n = O ;

**polygon\_inter.cc**

Page 4/5

/\* eliminate dltplicates \**I*

for (i = O ; i < n-1; i++) {

for ( j = i+l; j < n; j++) ("

if (out(i] .x == out[j] .x && out(i] .y used[j) = l;

) )

)

j = O;

out[j].y) {

**new count = O;**

**for-(i = O; i < n;·i++)**

if (!used[i]) { out[new\_count++] = out[i];

**n = new\_count;**

delete[] newpoly; delete(] used; **return n;**

i{nt read-poly(Point \*&poly)

**int n, i ;**

**cin "> > n;**

if (n == O )

**return O ;**

)

poly = new Point(n]; assert (poly);

for ( i = O ; i < n; i++) (

cin >> poly[i) .x >> poly[i) .y;

}

**reb.trn n;**

int main(void)

{

Point \*polyl, \*poly2, \*intersection;

**int nl, n2, n3, i;**

while ((nl = read\_poly(polyl) )) n2 = read poly(poly2);

n3 = intersect polygon(polyl, nl, poly2, n2, intersection); delete[] polyl;

delete[] poly2; if (n3 >= 3) {

for (i = O; i< n3; i++) (

cout << fixed << setprecision(2);

cout << " (" << intersection (il.x << " " << intersection [il .*y*

<< " ) II *i*

)

**cout << endl;** delete(] intersection; else {

cout << "Empty Intersection" << endl;

**return O;**



*(* ,, )



r*r* ; ( )

- \

*/* :sJ.

Printed by Darcy s

/

Aug 19, 14 21:37

polygon inter.cc

,Page 5/5

Aug 19, 1421:37

**ratlinsolve.cc**

Page 1/3

*II*

//. **Performs guassian elimination over the rationals.**

*II* Date September 22, 2010

*II* pair<int,int> means first = numerator, second = denominator

#include <iostream>

#include <iomanip>

#include ·<cstdlib> using namespace std;

#define pii pair<int,int> canst int MAX\_N = 100;

**pii \*r\_rn,m\_m;**

void print(pii x) {

**if(x.second** == 1)

**cout << x.first;**

else

*II* Author: Darcy Best

*II*

**cout << x.first** <<

11 / 11

<< **x.second;**

void print(pi A[MAX NJ (MAX NJ ,int m,int nl { for(int i=O i<m;i++){ -

for(int j O;j<n;j++) { cout << setw(S); print(A( iJ [j] ) ;

}

cout << endl;

)

cout << endl;

void read(pii& x){ cin >> x.first; char ch;

if(cin.peek(l == '/')

**cin >> ch >> x.second;**

else

**x.second** = l;

int gcd(int a,int b) ( while (b) {

int r = a % b;

a = b ;

b = r;

}

**return a;**

pii reduce(pii a)( if(a.first == OJ {

a.second = l; ·

) else { if(a.second < O) {

a.first \*= -1;

**a.second \*= -1;**

)

a.first *I=* g;

int g = gcd(abs(a.first) ,a.second);

a-second *I=* g;

return a;

pii operator\*(pii a,pii b){

Tuesday August 19, 2014

polygon\_inter. cc. ratlinsolve.cc 68/77

Aug 19, 14 21:37 ratlinsolve.cc Page 2/3

} return reduce(pii(a.first\*b. first,a.second\*b.second));

*pii* operator+(pii *a, pii* b) (

} return reduce(pii(a.first\*b.second+b.first\*a.second,a.second\*b.second));

void multRow(pii& x){

x = **x \* m m;**

-

)

void addMultRow(pii& x) (

} x = x + (m-m • (\*r-m++));

int rowReduction {pii A [MAX N] [MAX N] , int rows, int cols){

**int rank = o ;** - -

for(int C=O;c<cols;c++) { for(int r=rank;r<rows;r++) {

if(A[r] [cl .first){

if(r.!= rank) // Swap rows

swap ranges(A[rank],A[rank]+cols,A[r)); if(c - cols-1) *II* Inconsistent

**return -1;**

*II* Make first entry 1

m m = pii (A [rank] [c) .second,A [rank) (cl .first) ; for each(A(rank)+c+l,A(rank)+cols,multRow); A(rank) [c) = pii (1,0 ) ;

for(int i-(arb?rank+l:O) ;i<rows;i++) if(A(i) [cl .first && i != rank){

*II* Make the other rows 0

m m = pii(-A[i) [c] .first,A[i] [c] .second); r-m = A[rank)+c+l;

for each(A[i)+c+l,A[i]+cols,addMultRow); A(il[c] = pii (0,l);

} ) }

)

**rank++;**

break;

**return rank;**

int main() ( int C=O;

**int T,m,n,rank;**

pii A[MAX\_N] [MAX\_N];

while (cin » T &,& T)( if (C++)

cout << endl;

**cout << 11 Solution for Matrix System** # 11 << **T << endl ; cin >> n >> m;**

**for(int i= O;i<m;i++)**

for(int j=O;j<=n;j++) read(A[i] [j]);

if( (rank = rowReduction(A,m,n+l)) < 0)(

**cout << "No Solution." << endl;**

} else {

if(rank != n){

**cout << "Infinitely many solutions containing** 11 << **n-rank** << 11 **arbitrary**

**constants. 11 << endl;**

} else [

for(int i=O;i<n;i++) {

} cout << "x[" << i+l <<· "] = ";print(A[i] [n]); cout << endl;

Aug 19, 14 21:37

**ratlinsolve.cc**

Page 3/3

} } }

return O;

'. .rJay August 19, 2014

-........!

*{* . ratlinsolve.cc

- -v\

/.

*··-.. L*



*r"'*

*6/*

l,/

.)

\

,./

*/*

*(* J/

Printed by Darc t

-

Aug 19, 14 21:37 roman\_ numerals.cc Page 1/1

*II* Converts Roman Numerals to Arabic Numbers (and vice versa)

*II*

*II* Author: Darcy Best

*II* Date September 5, 2010

*II*

*II* If you are given a valid integer (0 < x < 4000), then it will give

Aug 19, 14 21:37 **sec.cc** Page 1/2

*II* Compresses a directed graph into its strongly connected components

*II*

*II* Author: Darcy Best

*II* Date : October 1, 2010

*II*

*II* A set of nodes is "strongly connected" if for any pair of nodes in

*II* the standard roman numeral representation of it. Note that if you give *II* the set, there is a path from u to *v* AND from v to u.

*II* it a number such that x >= 4000, then it will just append as many

*II* as needed.

*II*

**11 M 11 s**

*II*

*II* Compressing a graph into its strongly connected components means

*II* converting each strongly connected component into a super-node.

*II* If you are given a valid roman numberal, then it will give you the answer

*II* as a base 10 number.

#include <iostream>

#include <String>

#include <map>

using namespace std;

**canst string Roman [13 )** = { **"M" , u cM" , 11 D 11 *I* 11 CD 1 ' 11 C 11 *I* 11 XC" *I* 11 L11 *I* 11 XL 11 *I* 11 X 11 ' 11 X 11 ' 11 V 11 *I* 11IV 0**

**'**

*I* **U r i/ } *j***

canst int Arabic[ l3) = {1000,900,500,400,l00,90,S0,40,l0,9,5,4,1] ;

string toRoman( int x) (

**string roman;**

for(int i=O;i<l3;i++) while(x >= Arabic[ i)){

x -= Arabic[ i);

roman += Roman(i];

**return roman;**

int toint(string s){ int Ll,L2,ind=O,ans=O; while(ind < 13)(

Ll = s.length();

L2 = Roman[ind) .length();

*II*

*II* We then build a "compressed" graph made with the super-nodes. We *II* add an edge in the compressed graph between U and V if there is a *II* vertex u in U and v in V such that there was an edge from u to v in *II* the original graph. The compressed graph will be a Directed Acyclic *II* Graph (DAG}, .and the list of components will be in REVERSE

*II* topological order.

*II*

*II* If you are only concerned with the number of strongly connected

*II* components, you do not need to build the graph. See comments below

*II* on how to remove the sec graph.

*II*

*II* The complexity of this algorithm is O(I VI + I E!).

*II*

#include <iostream>

#include <algorithm>

#include <Stack>

#include <cassert>

**#include <vector>**

using namespace std;

const int MAX\_NODES 100005; struct Graph{

int numNodes;

vector<int> adj [MAX NODES) ;

if(s.substr(O,min(Ll,L2) ) ans += Arabic[ ind]; s.erase(O,min(Ll,L2)); else { ·

**ind++;**

**return ans;**

int main() { **char c; int X;**

Roman[ind]) (

void clear(){ - numNcides = O;

for{int i=O;i<MAX NODES;i++) adj [i) .clear() ,-

J

void add edge{int u,int v) {

if{find{adj [u).begin(),adj [u) .end(),v) adj [u).push\_back{v};

}; )

int po[MAX\_NODES) ,comp[MAX\_NODES) 1

adj [u) .end() )

string s;

*II* Checks to see if the line is Roman Numerals or Arabic Numbers,

*II* then converts to the opposite. while(cin >> c){

cin.putback(c);

if(c >= '0' && C <= '9')(

**cin >> x;**

cout << toRoman(x) << endl; else {

**cin >> s;**

void DFS(int v,const Graph& G,Graph& G scc,int& C, stack<int>& P,stack<int>& S) {-

po[v) = C++;

1. push{v); P.push{v);

for(unsigned int i=O;i<G.adj [v].size();i++} { int w = G.adj [v] [i];

if{po[w) == -1){

DFS(w,G,G scc,C,P,S};

else if(comp[w) == -1)(

while{!P.empty{} && {po[P.top{)) > po[w] ))

**cout << toint {s) < < endl;**

. J

J

P.pop();

)

return *D ;*

Tuesday August 19, 2014

if (!P.empty() && P.top() == v) { while{! S.empty()) {

int t = S.top{} ;

S.pop(),

comp[t]·= G\_scc.numNodes;

roman\_numerals. cc, sec.cc *70177*

|  |
| --- |
| Aug 19, 14 21:37 **simplex.cc** Page 1/2 |
| #include <algorithm>  using namespace std;  canst int MAX CONSTRAINTS = 100;  canst int MAX-VARS = 100;  const int MAXM = MAX CONSTRAINTS + l; const int MAXN = MAX=VARS + l;  canst double EPS = le-9; canst double INF = l.OIO.O;  double A[MAXM] [MAXN];  int basis[MAXM), out[MAXN);  void pivot(int m, int n, int a, int b)  {  **int i, j;**  **for Ci = O; i <= m; i++)**  if (i != a)  for (j = O; j <= n; j++)  if ( j != bl  A[i) [j) -= A[a) [j) \* A[i] [bl *I* A[a) [bl ; for (j = O; j <= n; j++)  if (j != b) Ala] [j) *I=* A[a) [bl ;  for (i = O; i <= m; i++l  if (i != a) A[i] [bl = -A[i) [bl *I* A[a) [bl ; A[a) [bl = 1 *I* A(a] [bl ;  swap (basis[al , out [bl l ;  d{ouble simplex(int . m, int n , double C[)[MAXN), double X [ J )  int i, j, ii, jj;  **for (i = l; i <= m; i++)**  copy(C[i), C[i)+n+l, A[i)); for (j = O; j <= n; j++)  A[O] [j] = -C[O) [j);  **for (i = O; i <= m; i++)**  basis [i] = -i;  for (j = O; j <= n; j++l out[j] = j;  for (;;l {  **for (i = ii = l; 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) >= -EPSl break; for (j = jj = a ; j < n; j++ )  if (A[ii) [j) < A[ii) [jj] - EPS 1 1  (A[ii] [j] < A[ii] [jj] - EPS && out[i] < out[j])l  j j = j ;  if (A[ii] [jj] >= -EPS) return -INF; pivot (m, n, ii, jjl ;  }  for (;;) {  for (j = j j = a; j < n; j++ )  if (A[O] [j] < A[O] [jj] 1 1 (A[O][j] == A[O] [jj) && out[j] < out[jj]l l  j j = j ;  if (A[O] [jj] > -EPS) break;  **for (i=l, ii=O; i <= m; i++)**  if ((A[i] [jj]>EPSl &&  (!ii \ \ (A[i] [n]IA[i] [jj] < A[ii] [n]IA[ii] [jj)-EPS) \ \  ((Ali) [n]IA[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);  ) |
| fill(X, X+n, Ol ; |

*i*

simplex.cc *?'f ·J)*

;" \..'-':?\

v

·-..·'1.j;:'J j)

Aug 19, 14 21:37

if(t == v)

break;

**sec.cc**

Page 2/2

}

G scc.numNodes++; P-:-pop();

int SCC(const Graph&, G,Graph& G\_scc) ( G sec.clear();

int C=l; stack<int> P,S;

fill(po,po+G.numNodes,-1); fill(comp,comp+G.numNodes,-1) for(int i=O;i<G.numNodes;i++)

if (po[i) == -.1)

DFS(i,G,G\_scc,C,P,S);

*II* You do not need this if you are only interested

*II* strongly connected components.

for(int i=O;i<G.numNodes;i++) {

for(unsigned int j=O;j<G. adj [i) .size();j++) { int w = G.adj [i][j];

if(comp[i] != comp[w]) G\_scc.add\_edge(comp[ i] ,comp[w]);

} }

in the number of

return G\_scc.numNodes;

*II* avoid Runtime Error. Graph G,G\_scc;

int main() {

*II* Declare these as a global variable if MAX NODES is large to

-

**int**

**U1 v, m,n;**

**int n sec;**

while cin >> n >> m && (n I \ ml ){ G.clear() ;

G.numNodes = n; for(int i=O;i<m;i++l {

**cin >> u >> *v ;***

}

G.add-edge(u,v);

n\_scc = SCC(G,G\_sccl ;

cout << " # of Strongly Connected Components: " << n\_scc << endl;

**return O ;**

~~Tr~~ y August 19, 2014

sec.cc,

...

*;*

*·,,••..,!••/*

\

•.::,..\_,

·'·*I*·

C

/

Printed by Darcy*U*

Aug 19, 14 21:37 **str rotation period.cc**

1

*I\**

* Page 1/1
  + Finding the lexicographically least rotation of a string, and finding
  + the smallest period of a string.

\*

* + Author: Sumudu Fernando

\*

* + Given a string, the algorithm can be used to compute two things:

\*

* + a) the position at which the lexicographically least rotation starts.

If there are ties, give the first position. .

b) the length of the shortest substring such that the original string is a concatenation of copies of that substring

* + Complexity: O(n) where n length of the string
  + Tested on: 719

\* 10298

ACPC 2011 H

Glass Beads Power Strings

Let's call a SPaDE a SPaDE

\*/

simplex.cc

Page 2/2

for (i = l; i c= m; i++) if (basis[i] >= 0)

X [basis [i]J = A [iJ (n) ;

return A[O] [n] ;

**#include <iostream>**

#include ciomanip>

int main (void)

{

.

double C[MAXM) (MAXNJ ' X[MAX\_VARSJ ;

C [OJ ( 0)

C [l] [OJ

C[2][O]

C [3] [OJ

-1; C[O) (lJ = -3; C[OJ [2] = O; C[O] [3J = 0 ;

-2; C (l] (l] = -3; C[l) (2) = -6 ; C(l] [3] = 250;

-1; C[2J (1) = -5 ; C [2) [2] = -5; C [2] [3) = 4 00;

-59; C [3) [1) = -35 ;

C [3] [2] = -160 ; C [3] (3] = 30 ;

double val = simplex(2, 2, C, X);

cout cc fixed cc setprecision(3);

**cout** .<< **11 val** = << **val << endl;**

11

cout c< "X [OJ = " << X [OJ cc endl; cout <c "X[l) = " cc X[l] <c endl;

*II* cout << "X [2] = " << X [2] << endl;

return O;

*(*

'

*,/*

*/*

#include <iostream>

#include cstring>

#include <algorithm>

using namespace std;

*II* pos = position of the start of the lexicographically least rotation

*II* period = the period

voi.d compute (string s, int &pas, int &period)

{

**S += S;**

int len = s.length(); int i = O; j = l;

for (int k = O; i+k < len && j+k < len; .k++) {

if (s [i+k] > s[j+k]) { i = max(i+k+l, j+l); k = -1;

else if (s[i+k] c s[j+k]) j = max(j+k+l, i+l);

k = -1;

) )

pas = min(i, j);

period = (i > j) ? i - j j - i;

int main(voidi

{

string s;

while (cin >> s) {

int pos, period; **compute(s, pos, period);** int n = s.length();

s += **s;**

cout c< "least rotation = " cc s.substr(pos, n) cc endl;

**cout** '<< **"period** = " << **s . substr ( 0 , period) << endl ;**

return D ;

Tuesday August 19, 2014 simplex.cc, str\_rotation\_period.cc 72/77

Aug 19, 14 21:37 **suffixarray.cc** Page 1/4 Aug 19, 14 21:37 **suffixarray.cc** Page 2/4

*I\**

* Suffix array
* Author, Howard Cheng
* **References:**

**Manber, U. and Myers, G. 11 Suffix Arrays: a New Method for On-line**

String Searches." SIAM Journal on Computing. 22(5) p. 935-948, 1993.

1. Kasai, G. Lee, H. Arimura, S. Arikawa, and K. Park. "Linear-time

* **Longest-common-prefix Computation in Suffix Arrays and Its** Applications." Proc. 12th Annua1·conference on Combinatorial Pattern Matching, LNCS 2089, p. 181-192, 2001

#define GetI () (SA12 [t] < nO ? SA12 [t] \* 3 + l (SA12 (t] - nO) \* 3 + 2)

void sarray\_int(int\* s, int\* SA, int n, int K) {

int n0=(n+2)/3, nl=(n+l)/3, n2=n/3, n02=nO+n2; .

int\* sl2 = new int(n02 + 3]; sl2[n02]= sl2[n02+1]= sl2[n02+2]=0; int\* SA12 = new int[n02 + 3]; SA12[no2]=SA12[n02+l]=SA12[n02+2]=0;

int\* so = new int[nO]; int\* SAO = new int[nO];

for (int i=O, j=O; i < n+(nO-nl); i++) if (i%-3 != O) sl2[j++J = i;

J. Karkkainen and P. Sanders. Simple linear work suffix array construction. In Proc. 13th International Conference on Automata, Languages and Programming, Springer, 2003

radixPass(sl2 , SA12, s+2, n02, K) radixPass(SA12, sl2 , s+l, n02, K) radixPass(s12 , SA12, s n02, K)

* The build\_sarray routine takes in a string str of n characters (null­
* terminated), and construct an array sarray. Optionally, you can also

**int name = o, co**

for (int i = O;

= -1, cl = -1, c2 = -1; i < n02; i++) {

* construct an lcp array from the sarray computed. The properties if (s[SA12 [i]] != co I I s[SA12[i)+l] != cl I I s[SA12[i]+2] != c2) {

**are:**

- If p = sarray[i], then the suffix of str starting at p (i.e.

str[p ..n-1] is the i-th suffix when all the suffixes are sorted in lexicographical order

**name-f.+; co**

}

if (SA12 [i) %­

else

s [SA12 [i)); cl *=* s [SA12 [i]+l] ; c2 = s [SA12 [i]+2];

3 == 1) { s12 [SA12 [i]/3] = **name;**

{ s12 [SA12 [i)/3 + no] = **name;**

NOTE, the empty suffix is not included in this list, so sarray[O] -! = n.

if (name < n02) (

sarray\_int(s12, SA12, n02, name);

* - lcp[i] contains the length of the longest common prefix of the suffixes pointed to by sarray[i-1] and sarray[i]. lcp[O] is defined to be 0.

- To see whether a pattern P occurs in str, you can look for it as

for (int i = O; else

for (int i = O;

i < n02; i++) sl2 [SA12 [i]J **i + 1;**

i < n02; i++) SA12 [s12 [i] - 1) = i;

the prefix of a suffix. This can be done with a binary search in O (IP I log n) time.. Call find() to return a pair <L, R> such that

all occurrences of the pattern are at positions sarray(i] with

**L <= i < R. If L == R then there is no match.**

\* The construction of the suffix array takes O(n) time.

for (int i=O, j=O; i < n02; i++) if (SA12 [i] < no) so [ j ++ )

**radixPass(sO, SAO, s, no, K);**

for (int p=O, t=nO-nl, k=O; k < n; k++) { int i = GetI ();

int j =. SAO[p]; if (SA12[tl < no

3\*SA12 [i);

•/

#include <iostream>

**\*include <iomanip>**

#include <string>

leq(s[i], sl2[SA12[t] + nO], s[j], sl2[j/3]) : leq(s[i),s[i+l] ,sl2"[SA12 [t)-nO+l], s [j ) ,s [j+l] ,sl2 (j/3+n0) ))

SA[k) = i; t++;

if (t == n02) {

#include <algorithm>

**#include <climits>**

for (k++; p < nO;

}

p++, k++) SA[k] SAO [pl";

using namespace std; bool leq(int al, int a2,

(

int bl, int b2)

else {

SA[k] = j; p++;

if (p == no) {

for (k++; t < n02;

t++, k++) SA[k) = GetI();

return(al < bl I I al == bl && a2 <= b2);

J l

delete [) sl2; delete (] SA12; delete (] SAO; delete [] so;

bool leq(int al, **int a2, int a3,**

{

int bl, int b2, int b3)

return(al < bl I \

)

al == bl && leq(a2,a3, b2,b3));

v{oid build-sarray(string str, int sarray(J )

int n = str.length();

**void radixPass { int\* a , int\* b , \_int\* r, int n, int K}**

{

int• c = new int[K + 1]; fill(c, c+K+l, C);

if (n <= 1) {

for (int i = O ; i < n; i++) { sarray[i) = i; .

**for {int i = O; i < n;**

for (int i = O, sum = O;

i++) c[r[a[i]])++;

i <= K; i++) {

}

**return;**

int t = c[i]; c[i] = **sum; sum += t. ;**

)

for (int i = O; i < n; delete [] c;

*r'* 4ay August 19, 2014

i++) b[c[r[a[i])]++J a [i] ; int \*s = new int[n+3]; int \*SA = new int[n+3];

suffixarray. cc

·\

*·-.... ..?·*

'

. *( ,,;*

- -

*·*.*,*ii.*.*:*\*:*\_*:1*\_}*\

.,.....

*r--·*

-·

r-

Aug 19, 14 21:37

for (int i = D; i c n; i++) (

Aug 19, 14 21:37

·**suffixarray.cc**

Page 4/4

if (pattern < str.substr(sarray[mid] , p)) ( hi = mid;

else {

**lo = mid;**

} } .

R = hi;

if (L > R) R = L;

return make\_pair(L, R);

int main{void)

{

**string str;**

int sarray[lOO), lcp[lOO]; unsigned int i;

while (cin >> str) (

build sarray{str, sarray); compute\_lcp(str, sarray, lcp);

for (i = 0; i < str.length(); i++) {

**cout << setw (3 ) << i** << ' << **setw (2) << lcp [i]** <<

11 : 1

11 , 11

<< str.substr(sarray[i) , str.length()-sarray[ i]) << endl;

}

**return O ;**

*c;,*

suffixarray.cc



Page 3/4

*y* --

\:::;,/ '

,,

Printed by Dare

} s[i) = (int)str[i) - CHAR-MIN + 1; s[n) = s[n+l) = s[n+2) = O;

sarray int(s, SA, n, 256); copy(SA, SA+n, sarray);

delete[] s, delete[] SA;

void compute lcp(string str, int sarray[J , int lcp[l )

-

(

int n = str.length(); int \*rank = new int[n];

for (int i = O; i c n; i++) rank[sarray[il l = i;

J

int h = O;

for (int i = O; i c n; i++) ( int k = rank[i];

if (k == 0 ) (

lcp[k] = -1;

else {

int j = sarray[k-1];

while (i + h c n && j + h < n && str[i+h] str[j+hl ) {

**h++;**

}

lcp [kl = h;

)

if {h > O )·

h--;

} )

lcp[O] = O; delete[] rank;

paircint,int> find(const string &str, const int sarray[),

canst string &pattern) ·

int n = str.length(), p = pattern.length(); int L, R;

if {pattern <= str.substr(sarray[O), p)) {

L = O;

else if (pattern > str.substr(sarray[n-1] , p)) (

**L = n;**

else (

int lo = 0 , hi = n-1; while (hi - lo > 1) {

int mid = lo + (hi - lo)/2;

if (pattern <= str.substr(sarray[mid] , p)) (

hi = **mid;** else ( lo = mid,

)

L = hi;

if (pattern < str.substr(sarray[O] , p)) { R = O;

else if (pattern >= str.substr( sarray[n-1] , p)) ( R = n;

else {

int lo = 0, hi = n-1; while (hi - lo > 1) {

int mid = lo + (hi - lo)/2;

Tuesday August 19, 2014

suffixarray.cc 74/77

Aug 19, 14 21:37 top\_sort.cc Page 1/2

*I\**

* Topological sort
* Author: Howard Cheng
* Given a directed acyclic graph, the topological\_sort routine
* returns a vector of integers that gives the vertex number (0 to n-1)

\* such that if there is a path from vl to v2, then vl occurs earlier

* than v2 in the order. Note that the topological sort result is not
* **necessarily unique.**
* **topological sort returns true if there is no cycle. Otherwise it**
* returns false and the sorting is unsuccessful.
* The complexity is O(n + m).

*\*I*

#include <iostream>

#include <algbrithm>

**#include <Vector>**

#include <queue>

**using namespace std;**

typedef int Edge;

order.clear() ;

while (!q.empty()) { int v = q.front(); q.pop (); order.push back(v);

for (int i-= O; i *<* G.nbr[v] .size(); i++) (

if (--indeg[G-.nbr[v] [i]J == 0) (

q.P,ush(G.nbr(v] (i]);

return order.size() **G.num\_nodes;**

int main (void)

{

**int n, m;**

while (cin » n ».m && <n I r ml l Graph G(n);

for (int i = O; i < m; i++) int u,.v;

**cin >> u >> v;**

I Printed by Darcy Best

typedef vector<Edge>::iterator Edgeiter;

struct Graph { vector<Edge> \*nbr; **int num nodes;** Graph(int n)

num\_nodes( ni

{

nbr = new vector<Edge>[ num nodes];

-

)

-Graph()

{

delete[] nbr;

*II* note: There is no check on dupliGate edge, so it is possible to

*II* add multiple edges between two vertices

void add\_edge(int u, int v)

(

) nbr(u) .push-back(Edge(v));

);

) G.add-edge(u, v);

vector<int> order;

if (topological sort(G, order)) for (int i = O; i < n; i++) (

**if Ci) cout** << ' ';

cout << order[i];

)

cout << endl; else {

**cout << " there is a· cycle 11 < < endl ;**

}

**return o ;**

b{ool topological-sort(const Graph &G, vector<int> &order) vector<int> indeg(G.num nodes);

.

fill(indeg.begin(), indeg.end(), 0);

**for (int** i = **O; i < G.num\_nodes;** i++) { for {int j = O; j *<* G.nbr[i] .size(); j++ )

indeg[G.nbr[i] (j]]++;

*II* use a priority queue if you want to get a topological sort order

*II* with ties broken by lexicographical ordering

**queuecint> q;**

for (int i = O; i < G.num\_nodes; i++) (

if (indeg[i] == O) (

q.push(i);

*I {*

*ay* August 19, 2014

top\_sort.cc \)

\.... ...., j ·./1

Aug 19, 14 21:37

**top\_sort.cc**

Page 2/2

...\_...,,,. /



-....,

; (L '

'

*,,.-:l*

*r::-* •..;\

)

*c)i*

Printed by Darcy Best

Aug 19, 14 21:37 **vecsum.cc** Page 1/2

*I\**

* Largest subvector sum
* Author: Howard Cheng
* **Reference: Programming Pearl, page 74**
* Given an array of integers, we find the continguous subvector that
* gives the aximum sum. If all entries are negative, it returns
* **an empty vector with sum = 0.**
* If we want the subvector to be nonempty, we should first scan for the
* largest element in the vector (1-element subvector) and combine the
* **result in this routine.**
* The sum is returned, as well as the start and the end position
* (inclusive). If start > end, then the subvector is empty.

*\*I*

**#include <iostream>**

**#include <Casser >**

using namespace std;

int vecsum( int v[J , int n, int &start, int &end)

{

Aug 19, 14 21:37 ·

unionfind.cc

Page 1/1

*II* UnionFind cla.ss -- based on Howard .Cheng's C code for UnionFinq

*II* Modified to use C++ by Rex Forsyth, Oct 22, 2003

*II*

.

*II* find -- return index of x in the UnionFind

*II* Constuctor -- builds a UnionFind object of size n and initializes it

.

*II* merge -- updates relationship between x and y in the UnionFind

class UnionFind

{

struct UF { int p; int rank; };

public:

UnionFind(int n) {

**howMany = n;**

*II* constructor

uf = new UF[howMany] ;

for (int i = O; i < howMany; i++) { uf[i] .p = i; .

uf[i) .rank = O;

-UnionFind() { delete[] uf;

int find(int x) { return find(uf,x); )

bool merge(int x, int y)

**int resl, res2;** resl = find(uf, x); res2 = find(uf, y);

if (resl != res2) (

if (uf[resl] .rank > uf[res2] .rank) uf[res2) .p = resl;

*II* for client use

}

else {

uf[resl) .p = res2;

if (uf[resl) .rank == uf[res2) .rank) { uf[res2) .rank++;

l

**return true;**

)

**return false;**

private:

int howMany; UF\* uf;

int find(UF uf[J , int x) {

// recursive funcion for internal use

if (uf[x] .p != x) (

uf[x) .p = find(uf, uf[x) .p);

)

return uf[x) .p;

l ;

**int max.val = O; int max end = O;**

**int max-end start, max end end;**

**int i ;** - - - -

start = max end start = O;

end = max end end =·-1; for (i = O; *C<* n; i++) (

if (v[i) + max\_end >= 0) (. max end = v[i) + max end; max-end end = i; - else { -

**max end start = i+l;** max-end-end = -1; **max=end-= O ;**

if (maxval < max\_end) {

**start = max end start; end = max efld efld; maxval = max end;**

**else if (maxVal == max end)**

*I\** put whatever preferences we have for a tie *\*I*

*I\** **eg. longest subvector, and then the one that starts the earliest \*/**

if (max\_end\_end - max\_end\_start > end - start I I

**(max end end - max end start == end - start** &&

max=end=start < start ) (

**start = max end start;**

end = max end end; .

**maxval = ffiax\_ena;**

**return maxval;**

int main(void)

{

**int n; int \*V;** int i;

**int sum, start, end;**

Tuesday August 19, 2014

unionfind.cc. vecsum.cc 76/77

while (cin >> n *t.&,* n > 0)

Aug 19, 14 21:37

vecsum.cc

Page 2/2

v = new int [n] ;

assert(v);

for (i = O; ic n;· i++)

**cin** :> > v [i) ;

}

**sum = vecsum(v, n, start, end);**

**cout << "Maximum sum 11 < < sum << 11 from 11 << start << 11 to 11 < < end <<**

<< **endl;**

delete[] v;

return o;

11 11

Printed by Darcy Best

'

1

|  |
| --- |
| Aug 19, 14 21:39 **Table of Content** Page 1/1 |
| Table of Contents  l *2sat.cc ..........* ... sheets 1 to 1 ( l) pages 1- 2 132 lines   1. areapoly.cc......... sheets 2 to 2 { 1) pages 3- 3 54 lines 2. *asc subseq.cc .* ...... sheets 2 to 3 ( 2) pages 4- 6 J.51 lines 4 bellmanford.cc ...... sheets 4 to 5 ( 2) pages 7- 9 l.41 lines   5 *bfs path. cc.........* sheets 5 to 6 ( 2) pages 10- 12 l.42 lines  .6 *bicomp.cc., .........* sheets 7 to 7 ( 1) pages 13- 14 J.36 lines   1. *bigint. cc...........* sheets 8 to 13 ( 6) pages 15- 25 755 lines 2. *binsearch. cc........* sheets 13 to 14 ( 2) pages 26- 27 l.02 lines   9 ccw.cc .............. sheets 14 to 15 ( 2) pages 28- 29 76 lines   1. *circle 3pts.cc ......* sheets 15 to 15 ( 1) pages 30- 30 64 lines 2. *common-subseq.cc ....* sheets 16 to 16 ( l) pages 31- 32 85 lines 3. *convex=hull.cc ...*... sheets 17 to 18 ( 2) pages 33- 35 J.83 lines 13 era.cc.............. sheets 18 to 19 ( 2) pages 36- 37 134 lines 14 date.cc ............. sheets 19 to 20 ( 2) pages 38- 40 204 lines 4. *dijkstra.cc ....*..... sheets 21 to 22 ( 2) pages 41- 43 J.66 lines 5. *dijkstra\_sparse.cc ..* sheets 22 to 23 ( 2) pages 44- 46 l.43 lines 17 *dist3D.* cc........... sheets 24 to 26 ( 3) pages 47.- 51 281 lines 6. *dist\_line.cc ...*..... sheets 26 to 26 ( l) pages 52- 52 47 lines 7. *euclid.cc..*......... sheets 27 to 27 ( l) pages 53- 53 45 lines 8. *eulerphi.cc ....*·..... sheets 27 to 28 ( 2) pages 54- 55 84 lines 9. *eulertour.cc .*....... sheets 28 to 30 ( 3) pages 56- 59 213 lines 22 *exp.cc ..:...........* sheets 30 to 30 ( l) pages 60- 60 42 lines 10. *expmod.cc......* ...... sheets 31 to 31 ( l) pages 61- 61 44 lines 11. *exteuclid.cc........* sheets 31 to 31 ( l) pages 62- 62 70 lines 12. *factor.cc ...........* sheets 32 to 32 ( 1) pages 63- 64 71 lines 13. *factor\_large.cc .....* sheets 33 to 34 ( 2) pages 65- 68 249 lines 14. *fenwicktree.cc ..:.*.. sheets 35 to 36 ( 2) pages 69- 71 J.47 lines 28· *fflinsolve.cc .......* sheets 36 to 37 ( 2) pages 72- 73 l.28 lines 29 floyd.cc ............ sheets 37 to 38 ( 2) pages 74- 75 84 lines 15. *floyd path.cc .......* sheets 38 to 39 ( 2) pages 76- 77 l.28 lines 16. *frac2dec.cc ...*...... sheets 39 to 40 ( 2) pages 78- 79 88 lines 17. *fraction.cc .........* sheets 40 to 42 ( 3) pages so- 83 230 lines 18. *greatcircle.cc.* ..... sheets 42 to 43 ( 2) pages 84- 85 84 lines 34 *heron.cc ...........* .·sheets 43 to 43 ( 1) pages 86- 86 37 lines   35 *lmngarian.cc ........* sheets 44 to 45 ( 2) pages 87- 90 211 lines 36 *infix.cc ..:.........* sheets 46 to 47 ( 2) pages 91- 94 221 lines   1. *int\_mult.cc .........* sheets 48 to 48 ( 1) pages 95- 96 85 lines 2. *intersectTF. cc...... sheets* 49 to 49 ( 1) pages 97- 98 77 lines 3. *intersect\_circle\_circle. cc* sheets SO to so ( 1) pages 99-100 136 lines 4. *intersect\_iline.cc ..* sheets 51 to 51 ( l) pages 101-102 73 lines 5. *intersect iline circle.cc* sheets 52 to 52 ( l) pages 103-104 97 lines 6. *intersect=line.cc...* sheets 53 to 53 ( 1) pages 105-106 124 lines 7. *joseplms.cc .........* sheets 54 to 54 ( l) pages 107-107 50 lines 44 *kmp. c:c ..............* sheets 54 to 55 ( 2) pages 108-109 73 lines 8. *linsolve.cc .........* sheets 55 to 56 ( 2) pages 110-112 191 lines 9. *mincostmaxflowdense. cc* sheets 57 to ·55 ( 2) pages 113-115 190 lines 10. *mincostmaxflowsparse.cc* sheets 58 to 59 ( 2) pages 116-118 178 lines 48 mst.cc .............. sheets 60 to 61 ( 2) pages 119-121 l.56 lines 49 *mult.cc .............* sheets 61 to 61 ( l) pages 122-122 55 lines   SO *networkflow.cc: ......* sheets 62 to 63 ( 2) pages 123-125 163 lines   1. *'networkflow2.cc.....* sheets 63 to 64 ( 2) pages 126-128 165 lines 2. *pointpoly.cc ........* sheets 65 to 65 ( l) pages 129-130 116 lines 3. *polygon\_inter. cc....* sheets 66 to 68 ( 3) pages 131-135 278 lines 4. *ratlinsolve. cc......* sheets 68 to 69 ( 2) pages 136-138 J.43 lines 5. *roman\_numerals.cc ...* sheets 70 to 70 ( l) pages 139-139 66 lines 56 *sec.cc ..............* sheets 70 to 71 ( 2) pages 140-141 121 lines 6. *simplex.cc..........* sheets 71 to 72 ( 2) pages 142-143 98 lines 7. *str\_rotation\_period.cc* sheets 72 to 72 ( l) pages 144-144 62 lines 8. *suffixarray.cc ......* sheets 73 to 74 ( 2) pages 145-148 237 lines 9. *top\_sort.cc..*....... sheets 75 to 75 ( l) pages 14'9-150 l.ll lines 10. *unionfind.cc ...*..... sheets 76 to 76 ( 1) pages 151-151 59 lines 62 *vecsum.cc...........* sheets 76 to 77 ( 2) pages 152-153 84 lines |
|  |

*Tu(* · August 19, 20'14



··

I J

'-.\_,,.,;,

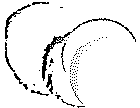
*i* vecsum.cc, Table of Content i

\ ·

*T* :::::1)

/





Theoretical Computer Science Cheat Sheet

Definitions Series

*J ( n)* = *O( g( n) )* iff 3 positive c, *n0* such that *ti=*

*ti3*

*n2(n*

1)2 .

0 *:S f ( n)* :S *cg( n) 'r/n no.*

*n( n +* 1) fi2 = *n( n +* 1)(2n + 1) = +

*f (n)* = *D( g( n) )* iff 3 positive·c, *no* such that

·i=l

2 ,

i=l

6 ' i=l 4

*f ( n)* 2: *cg( n)* 0 *\fn no .* In general:

*f ( n)* = *G( g( n) )* iff *f (n)* = *O( g( n) )* and }*n* :im = -

1-[en + 1rn+1 - 1- z*n*:((i + 1Jm+ 1 - r+1 - *( ni* + 1)r) J

m + l .

*f ( n)* = *D( g( n)).* i=l . i=l

*f ( n)* = *o( g(n) )* iff lim

oc *f ( n)/ g( n)* = 0.

n*L*-1 ·m 1 *Lm* (*·m* + I) *B* m+l-h,

11 i = -*m +*- *k*, *k n* .

1

lim *an = a* iff Ve > 0, 3no such that i=l *k=O*

*/an* - *a/* < *E, 'r/n* 2: *no .* Geometric series:

n-;,oc

*L .* I

sup S least *b* E IR such that *b* 2: *s ,*

*C i=* 1, c'· ---

*L i C*

/ cl < 1,

*n*

}:c; =

cn+l - 1

00 *oc*

C = --

c -1 '

- 1 - c ' 1 - *c'*

*\f .s* E S. ·i=O i=O i=l

*00*

.(Sf'

inf S greatest *b* E IR such that *b* :S

• *i ncn+2* - *( n +* l)c11+l + c

*}:-ic*

11

z:· ; C */cl <* 1.

*s, Vs*

E S.

t=O

= (c - 1)2 ' c / 1, . *-ic* = (1 - c)2 ,

t=O

lim inf *an* liin inf{ai Ii2: *n,i* E N}. Harmonic series:

rt-+tX; *noo* 11 1 *t B·* \_ *n( n* + 1)*H* \_ *n( n* - I)

*Hn = }: -:- ,*

i,. - ? " 4 .

lim sup a.,, Jim sup{a; I i2: *n,·i* E N}. *"t*

i=l i=l

n-+cx:i n-->oo

() Combinations: Size *k* sub- *}: H , = ( n + l)Hn - n,*

*n C )* . (n + l) ( 1 )

*11*

sets of a size *n* set. i=l

[r J Stirling numbers (1st kind):

*m* H; = . *m* + 1 *Hn+l* - *m* + 1 ·

. '

Arrangements of an *n* ele-

*k - ( n - k )! k! '* 3. *k* = *n - k* '

1. *( n) n!* 2. *t* (z)· = 2", ('1) ( *n* )

ment set into *k* cycles. *k=O*

g'. } Stirling numbers (2nd kind): *(n)* +

4. *(n)* = (n - 1), 5. = (n -1) (n -1) ,

*k k* k -1 *k k* . k - 1

Partitions of an *n* element

set into *k* no11-empty sets. 6.

1. 1st order Eulerian numbers:

*( n) (m) (n) ( n - k )* 7. ('· k) = *( r +:+* 1),

11

*rn k* = *k m - k* '

Permutations 1r11r2 . , . *1r11* on 8

*t ( k )* c- + l )

9 (7') ( *s* ) (+ *s)*

{1, 2, . , . , n} with *k* ascents.

'

*k=O*

m = m + l '

* + *k n - k* = *n* '

GJ 2nd order Eulerian numbers. 10. (;) = (-ll c -·z - 1),

11,

{n} = {1*n*1'} = 1,

'Cf'

*(*

*Cn* Catalan Numbers: Binary

{n }- ?n-1 - 1 {n} {n-1} {n - 1}

trees with *n* + 1 vertices. 12. 2 - *..J* , 13; *k = k k* + k - 1 '

1

14. [] = (n - 1)!, 15. [*n*2 *]*

- (·*n* - 1)*'*.*H* n-1,

16. [*r,.]* = 1, 17. *[ n]* { *n* },

18. [zJ = (n - 1>['n:1] + [z = l 1

[{ n }= [ n ]](#_TOC_250001)

[= (n) '](#_TOC_250000)

20, [ ;] = n!,

21. C,, =

1-(2n)'

n - 1 *n - 1* 2 *n +* 1 *n*

*n* . *k k:*

22. (*;)* = \n: 1) = l,

23.

(:) = (n - ·- k )'

24.

(n*k* ) = *( k* + 1)(n *k*- 1) + *( n* - *k )* (n*k*--11),

25. ( ) = G if *k* = 0, 26. \ 7) = 2" - *n* -1,

27.

(n) = 311 - *(n* + 1)2" + c- + l),

otherwise 2 2

28. *x"* = *t*(') *(X k).* 29. (;) = (n ;l) (m + l - k)"(-1)\ 30. m!{ }= (;) (n m),

*k=O* '

31. ( n ) = *t { 1 } ( n - k) ( -1)"-k-mk!,* 32. {;}= 1, 33. {:}= 0 for *n ¥* 0,

*m k=O k m*

34. ;}= *(k* + l){n :1

*k* '

}

t

+ (2n - 1- k){;= },

35.

*t* {n} =·(2n)!l

" -

- {*X :*n }= { }

e+ \ l - *k).*

37.

{ :}= ( ) {}=

*k=O*

L}(m+

2"

1r-k,

*(\*

\ - -

Theoretical Computer Science Cheat Sheet

Identities Cont. Trees

38. (;i:] = (:] ( ) = *[!]nn-k* = n!ti(!],

40. ·{ n }= *L* (n) {*k* + 1}(-lt-k,

*m* m + l

*k k*

39. *[ x :n]* = {}(\k),

41. (;] = ( ;] ( )(-1r-\

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

Kraft inequal­ ity: If the depths

42. {m +;+ l} *= f k{ n! k },*

*k=O*

43. [m +;+ l] = *tk( n+ k)[ n :k] ,*

*k=O*

of the leaves of a binary tree are

44. (;) = {::}[:J (-1r-\ 45. (n -m)!(;) = [::J{!}(-1r-\

for *n ;:::: m,*

d1, , , ·, dn:

*n*

*I: z-d ,* 1,

46. {*n -nm* }-

*k*

(m*m+*-*nk)* (m*n+*+*kn)* [m *k*+ *k ]* '

47. [*n -nm*J = L...., (m*m+*-*nk)* (m*n+*+ *kn)* {*m k*+ k },

i=l

and equality holds



48. {e:m}(e m) = { } { n k} (:),

49. *[ e :m]* c m) = *[:][n:k ]* (;).

*k*

only . if every in­ ternal node has 2

sons. '

Master method:

Recurrences *F'/*

Generating functions:

*T(n)* = *aT(n/ b)* + *f ( n),* a ;:::: l, b > 1

If 3E > 0 such that *f* (n) = O(n10gb a-e)

then

*T(n)* = 8(nlogb *a ).*

If *J( n)* = 8(n10gb a) then

*T ( n)* = e(n10gb *a* log2 *n).*

l(T(n) - *3T ( n/ 2)* = *n)*

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

3log2 n-l (T(2) - 3T(l) = 2)

Let *m* = log2 *n.* Summing the left side we get *T( n)* - 3mT(l) = *T( n)* - 3m ==

1. Multiply both sides of the equa­ tion by *xi.*
2. Sum both sides over all i. for which the equation is valid.
3. Choose a generating .function

*G( x ).* Usually G(x) = I: o *xig;.*

1. Rewrite the equation in terms of the generating function *G( x ).*

If :lt > 0 such that *f ( n)* = n(n10g.a+<),

*T( n)* - *nk* where *k* = log

3 1.58496.

and 3c < 1 such that *af ( n/ b) cf ( n)*

for large *n,* then

*T( n)* = 8(f (n)).

Substitution (example): Consider the following recurrence .

. 2' 2

T;+1 = 2 · *T;* , *Ti* = 2.

Note that *T;* is always a power of two.

Let *t;* = log2 *T;.* Then we have

t;+1 = 2i + 2t;, t1 = 1.

Let *u;* = *ti/ 2;.* Dividing both sides of the previous equation by 2H 1 we get

*t;+1 2i ti*

2H1 = *2i+i* + 2i .

Substituting we find

2,

2

Summing the right side we get

m-1 m-1

*L*; I:(

3i = *n* ) i.

*i=O* i=O

Let c =!. Then we have

m-1 ( m 1)

*n L e; = n* \:1

t=O

= 2n(clog2 *n* - 1)

= 2n(c(k-l}lo c n - 1)

= *2nk* - *2n,*

and so *T(n)* = 3n.k' - 2n. Full history re­ currences can often be changed to limited history ones (example): Consider

i-1

1. Solve for *G( x ).*
2. The coefficient of :1/ in *G(x )* is *91.*

Example:

9i+1 = *2g,* + 1, *.9o* = 0.

Multiply and sum:

*L .9i+i x'* = *L 2g,xi* + *L x;·.*

i>O i>O i>O

We coose *G( x )* I:i>O *xigi ,-*Rewrite 01

in terms of *G( x ):* -

*G( x )* - *.9o* = 2G(x) + *L X;.*

i2:0

Simplify:

. *G( x )* = 2G(x) + \_1\_.

*X* 1-*X*

*·u*i+l - 1 + ui,

2

*u*1 - 1

*T;* = 1+ *L Tj , To =* 1.

Solve for G(x):

*X*

which is simply *·u;* = i/2. So we find that *Ti* has the closed form *T;* = 2i2'-1 .

Note that

*j=O*

*( x )* = (1- x)(l - 2x) ·

the following recurrence

TH1 = 1+ *r:rj .*

*G( x*. *) = x* ( 2 --

1-)

Summing factors (example): Consider

*T( n)* = *3T ( n/ 2) +n,* T(l) = 1.

Rewrite so that all terms involving *T*

Subtracting we find

*i*

*j=O*

i i-1

Expand this using partial fractions:

l - 2x 1-x

= *x ( 2 L 2i xi* - *I: xi)*

are on tlie left side

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

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

*Ti+l - T;* = 1+ *LTj* - 1- *LTj*

*j =O j=O*

*= T;.*

i;::,:o i;::,:o

= *L(i+l* - l)xi+1

.

;;::,:o ·

*I*

scope"

And so *T;+1* = *2T;* = 2H1.

So *g;* = 2i -1.

\ *i)*

. *,t/*

(p

Theoretical Computer Science Cheat Sheet

2

*r,* 3.14159,

*e* 2.71828, 1' 0.57721,

rp = 1+-/5 1.61803,

¢= l-2-/5 -.61803

Probability

*6* - 42' 8 - -30 l 10 - 66.

Continuous distributions: If

Pr(a < *X* < b] = *p( x ) dx ,*

*lb*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 2i | | | *Pi* | General |
| 1 2  2 4  3 8  4 16  5 32 | | | 2  3  5  7  11 | Bernoulli Numbers (Bi = 0, odd *·i i=* 1):  *B*0 - *1*' *B*1 *- -z1 , B*2 - 6*1* > *B*4 - -310,  *B - 1 B -* i B \_ s  Change of base, quadratic formula:  loga *X -b* ± *,/b2* - *4ac* |
| *logb* a; = -*--b* ,  6 64 13 1*oga 2a*  7 128 17 · Euler's number *e:*  8 256 19  *9* 51 23 lim (1 + !:.) *n* = *e:r .*  2,048 31  4,096 37 (1+ *l.*)" = *e* - :\_ + - 0 (2-) .  1,j 8,192 41 Harmonic muubers:  14 16,384 43 1 3 11 25 137 49 363 761 7129  15 32,768. 47  16 65,536 53 *Inn* < *Hn* < In n + 1,  17 131,072 59  18 262,144 61 *Hn* = ln n + -y + o(;,)·  19 524,288 67 Factorial, Stirling's approximation:  20 1,048,576 71 1, 2, 6. 24. 120. 720. 5040, -10320. 362880, . . .  21 2,097,152 73  22 4,19°1,304 79 *n!* = hm(;)n (1+ e(;,)).  23 8,388,608 83 Ackermann's fonction and inverse:  25 33,554,432 97 *a(-i , j )* = {:i- 2) :i  26 67,108,864 101  a(i) = min{j I *a( j, j )* :::\_ *i}.* | | | | |
| 268,435,456 107 Binomial distribution:  29 536,870,912 109 *(;) p"'q"-\ q* = 1- *p,* | | | | |
| 30 | 1,073,741,824 | 113 |  |  |
| 31 | 2,147,483,648 | 127 |  | E(X] = *tk( ) pkqn-k np.* |
| 32 | 4,294,967,296 | 131 |  | *k=1* |
|  | Pascal 's Triangle |  |  | Poisson distribution:  *e->.).,k* |
| 1 Pr(X = kJ = ' E(X] *= A.*  1 1 Normal (Gaussian) distribution:.  1 2 1 . 1 *-( x-µ)2* /20"2 E(X] = *µ.*  1 3 3 1 *p* (*x* ) *=* v *2~~·rr~~cr e* ,  14 6 4 l The "coupon collector": vVe are given a  15 10 10 5 l random coupon each day, and there are *n*  different types of c:oupons. The dist.ribn­  1 6 15 20 15 6 1 tion of coupons is uniform. The expected  17 21 35 35 21 7 1 number of days to pass before we to col­  I8 28 56 70 56 28 8 1 lect all *n* types is  . 1 *9* 36 84 126 126 84 36 9 1  *.Ji* 20 210 252 210 120 45 10 1 | | | | |

then *p* is the probability density function of

*X.* If .

Pr(X *< a] = P( a),*

then *P* is the distribution fm1ction of *X .* If

*P* and *p* both exist then

¥10]

1,o2,1 29

e = l + l2 + l6 + .2..4L + \_i2L0 + ···

n--+oo *n*

(1 + i)" < *e* < (1 + tJ'+l.

*P( a)*

= *1 p( x ) dx.*

Expectation: If *X*

is discrete

E(g(X)]

= *L g( x )* Pr[X = *x:).*

*X*

24 16,777,216

*RI* 134,217,728

89

103

" *2n* 24n2 *n3*

1 2 ' 6' 12' 60' 20 1 140 ' 280 ' 2520 1 ' ..

1,

*a(-i* - 1, *a.(-i, j* - 1)) *i,j* 2: 2

If *X* continuous then

E[g(X)) == *l!( x )p( x ) dx* = *[ j <x ) dP(i).*

Variance, standard deviation:

VAR(XJ = E[X2] - E[XJ 2 •

*er =* JVAR(XJ .

For events *A* and *B:*

Pr[A *V B)* = Pr(AJ + Pr(B] - Pr(A *I\ B)*

Pr[A *I\* BJ = Pr[AJ · Pr[BJ,

iff *A* and *B* are independent.

Pr[A!BJ = Pr(A *I\* BJ

Pr[B]

For random variables *X* and *Y:*

E(X · *Y)* = E(X] · E[YJ;

if *X* and *Y* are independent.

E(X + Y] = E(X] + *E[Y J,*

E(cXJ = *c E( X ).*

Bayes' theorem:

Pr[X = kj =

*Pr [BI Ai]* Pr(A;j

Pr[AdBJ = '2: '=

1

*=* Inclusion-exclusion:

*n* n

Pr(AiJ *Pr [BI Ai ) .*

Pr *[* V*xi ]* = LPr[Xd +

*i=1 i=l*

I:(-1l+1 *L* Pr [ *J\ X;; ].*

*n k*

*k=2*

Moment inequalities:

*i;<···<ik j=1*

1

Pr [IX! 2 A E(Xl] J·

Pr *[I X* - E[XJI 2: *A · er]* '.S *A ·.*

Geometric distribution:

Pr[X = kj = *Pl- 1, q* = 1-*p,*

E(-·VJ \_ *Lk-.pq*h,-1\_- -1.

*k=l p*

'*(*' .*(*\_,

"--

Trigonometry

Theoretical Computer Science Cheat Sheet

Matrices More Trig.

Multiplication: *C*

*n*

*b* (cos *e'* sin e)

*A C ll*

(-1,0) (1,0)

*C = A · B , ci,j* = *L a;,k bk ,j·*

k=l

Determinants: det *A =/:-* 0 iff *A* is non-singular.

det A · *B* = det A · det B,

*A* C *B*

Law of cosines:

C *a* .

*n* 2 *2 2*

*B* {0,-1)

Pythagorean theorem:

c2 = A2 *+ B2 .*

Definitions:

det A = LITsign(-rr)ai,rr(i)·

7r i=l

2 x 2 and 3 x 3 determinant:

I; !I = *ad* - *be,*

*a b*

c = *a +b -2ab* cos *C.*

Area:

*A = lhc*

2 '

= iabsin C,

sin *a = A/C, csc a* = *C/ A,*

sin *a A*

tan a = -- = -

cos *a = B/ C,*

sec *a = Cf B,*

cot *a =*-cos *a* = *B*

- -

*d e* ? l =gl : *; ) - h i* ;) +ii :)

c2 sin A sin B

- 2sin C Heron's formula:

cos *a B '*

sin *a A·*

*aei* + *bf g* + *cdh*

Area, radius of inscribed circle:

*g h*

1 *AB*

*z AB , A+ B + c·*

Permanents:

- *ceg* - *f ha* - *ibd.*

*n*

*A* = *J S* · *Sa* · *Sb* · *Sc ,*

*s = !( a + b + c),*

*Sa = S* - *a,*

Identities: sm x = --,

. 1

*csc x*

*cos x* = --,

1

*sec x*

perm *A = LIT* ai,1r(i)·

7T' i=l

Hyperbolic Functions

*Sb = S* - *b,*

*Sc = S* - C.

tan :.c; = -

1-,

cot x

sin2 *X* + cos2 *X* = 1,

Definitions: -:z:

*e"'* - *e*

*e"'* + *e-x*

More identities:

. *x 0-cos x*

l+ tan 2 *:c* = sec2 *x ,*

1+ cot2 *x* = csc2 *x ,*

sinh x = 2 '

*e"' - e-x*

cosh x = 2 '

1

sm z = v 2 )

sin.i· = cos (J - x) , sin x = sin( rr - *x ),*

c1 >ii .r: = - cos(1r - *x ),* tan *x* = cot (f - *x )* ,

*cot. ,1:* = -cot(1r - *x ),* csc *x* = qot f - cot *x ,*

tanh x = *e"'* + *e-x '*

1

sech *x* = cosh *x* '

Identities:

csch *x* = sinh *x* '

1

coth x = tanh x ·

cos f = y

tan =

os x

,

1. *cos x*

1+ *cos x*

sin(x ± *y )* = sin x *cos y ±* cos *x* sin *y ,*

cosh2 *x* - sinh2 *x* = 1,

tanh2 *x* + sech2 *x* = 1,

1- *cos x*

= sin x I

cos( *x* ± *y )* = cos *x* cos *y* =i= sin *x* sin *y ,*

ccith2 *x* - csch2 *x* = 1,

cosh(-x) = cosh x,

sinh(-x) = -sinh ::c,

tanh( -x) = -tanh *x ,*

sin x

l x '

=

-

l + cos x *I*

"*1*·

tan(x ± y) =

tan x ± tarr y ,

1=i= tan *x* tan *y*

sinh(x + *y )* = sinh *x* cosh *y* + cosh x sinh *y ,*

cot =

1-*cos x'*

1+ *cos x*

cot (*x ± y* ) = cot *x* cot *y* =i= 1

cosh(x + *y )* = cosh x cosh y + sinh x sinh y,

cot x ± cot y , sin *2x* = 2 sin *x* cos *x ,.*

sin 2x =

sinh *2x* = 2 sinh *x* cosh *x ,*

sin x sin x

- 1-cos x '

cos 2x = cos2 *X* - sin2 *x ,*

cos *2x* = ?. c082 *X* - 1,

cosh *2x* = cosh2

*x* +sinh2 *x,*

*e•*.*x* - *e-ix*

sin x = 2i 1

cos 2x = 1- sin2 *x ,*

tan 2x = -2 tan x-

- 2

l-tan2 *x*

cos 2x = 1+ tan2 *x'*

cot2 *x* - 1 cot 2x = 2 cot *x* '

cosh x + sinh x *= ex ,* cosh x - sinh *x* = e-x ,

(cosh x + sinh x)n = cosh nx +sinh *nx,* n E Z,

2 sinh2 f = cosh x -1, 2 cosh2 f = cosh x + 1.

*eix* + *e-ix*

*cos x* = 2

*eix* \_ e-ia:

tan x = -i*e* . + . ,

1- tan *x'*

*'·x*

*e--ix*

sin(x +*y )* sin(x -*y )* = sin2 *x* - sin2 *y,*

*cos( x* + *y ) cos( x* - *y )* = cos2 *x* - sin2 *y.*

Euler's equation:

ei:z: = *cos x* + isin x, ei?r = -1.

*e* sin e *cos e* tan e . . . in mathematics

0 0 1 0 you don't under-

*1r* 1 V3 V3 stand things, you

6 2 2 3 just get used to

*e2ix* -1

- - ---

- e2i:z: + 1'

. sinhix sm *x*

= --.-,

'I.

cos *x* = cosh *ix,*

:> I I *(*

v2.02 ©1994 by Steve Seiden I *r.* v'3 1 *.Fi* ·

4 2 2

- J . von Neumann

· ---- - i h.

7r v'2 v'2 1 them.

I

"'"'..... i. *,:*.......

3 2 ""

*\.I*

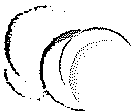
·- I I 1

··- -,:"' ' "··..---·---·---, -----· I

7f 1 *t.*

"u' 00 ).. /;/*•I*





Number Theory

Theoretical Computer Science Cheat Sheet

Graph Theory



The Chinese remainder theorem: There ex­

Definitions:

Notation:

ists a number *C* such that:

*Loop* An edge connecting a ver­ tex to itself.

*E(G)*

*V(G)*

Edge set Vertex set

*C :: rn* mod mn

if m; and *m1* are relatively prime for *-i # j.*

Euler's function: *cp( ;r; )* is the number of positive integers less than *X* relatively

prime to *x.* If rr=l *Pt* is the prime fac­

torization of *x* then

"

*cf>( x )* = *IT Pt -1 (p;* - 1).

*Directed Simple*

*Walk Trail Path*

*Connected*

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

A sequence *voe1v1* . . . *ee-ve ,* A walk with distinct edges. A trail with distinct vertices.

A graph where there exists a path between any two vertices.

*c(G)*

G(SJ

deg(-v)

*(G)*

*6(G)*

*x(G)*

*xE(G)*

*ac*

*Kn*

**Kn1,n2**

r(k, *f )*

Niunber of components

Induced subgraph Degree of *v* Maximum degree Minimum degree Chroinatic number

Edge- chromatic number Complement graph Complete graph

Complete bipartite gTaph Ramsey number

((]*(* . .

i=l

*Component* A maximal connected

. subgraph.

Geometry

* + · · .ler's theorem: If *a* and *b* are relatively

prime then

1= *a,f,( b)* mod *b.*

Fermat's theorem:

=

1 a.P-l mod *p.*

*Tree Free tree DAG*

*Eulerian*

A connected acyclic graph. A tree with no root.

Directed acyclic graph. Graph with a trail visiting each edge exactly once.

Projective coordinates: triples

*(x,y, z),* not all x, *y* and *z* zero.

*( x , y, z )* = *( ex, cy, cz)* Ve *=I=* 0.

Cartesian Projective

The Eu'clidea.11 algorithm: if *a* > *b* are in­

*Hamiltonian* Graph with a cycle visiting

each vertex exactly once.

*(x,y) (x,y, 1)*

*y* = *rnx* + *b* (*rn,* -t *b)*

tegers then

gcd(a, *b )* = gcd(a mod b, *b).*

If rr 1JJf ' is the prime factorization of *X*

*Cut*

A set of edges whose re- moval increases the num,

ber of components.

*X* = *C* (1, 0, -c)

Distance formula, *Lp* and *L00*

* + - metric:

v( +

then

*n e 1 +I* l

IT

*S( x ) = "\' cl = P,* - .

D . *p* 1

dl:r i.=l i-

*Cut-set* A minimal cut.

*Cid edge* A size 1 cut.

*k-Connected* A graph connected with

the removal of any *k* - 1

,.-c- -c-c -

*x* 1 - xo)2 (Y1 - Yo )2 ,

[lx1 - *xol P* + IY1 - *Yo*IP] l/p,

lirn [1;1:1 - *xol P* + IY1 - *Yo*I P] l/p.

Perfect Numbers: *x* is an even perfect mun­ ber ilf *X* = 2"-1(2n -l) and 2n -l is prime.

: Wilson's theorem: *n* is a prime iff

(n -1)! = -l mod n.

((r '1ius in1m·sion: if i= 1.

·,·· *C)* \_ 0 if *·i* is not square-free.

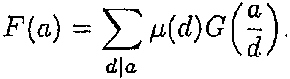
*t t* - (-1*Y* if iis the product of

*r* distinct primes.

If

*G( a)* = *L F( d ),*

*dla*

then

Prime numbers:

ln ln n

*Pn* = n ln n + n ln ln n *- n +n-­*

In n

+ o , *n* )

(

In n '

*n* · *n* 2!n

*rr(n) = -* + --+ --

In n (ln n)2 (lnn)3

+ 0((ln .)4 ) ·

vertices.

*k- Tottgh VS V,S =/=* 0 we have

*k* · *c( G* - *S)* ::; *181.*

*k-Regular* A graph where all vertices have degree *k.*

*k-Factor* A k-regular spanning

subgraph.

*Matching* A set of edges, no two of

which are adjacent.

*Cliq-ue* A set of vertices, all of

which are adjacent.

*Ind. set* A set of vertices, none of

which are adjacent.

*Vertex cover* A set of vertices which cover all edges.

*Planar graph* A graph which can be em­

beded in the plane.

*Plane graph* An embedding of a planar

graph.

*L* deg(-v) = *2rn.*

t•EV

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

*f* ::; *2n* - 4, . *m. 3n* - 6.

Any planar graph has a vertex with de­ gree ::; 5.

**p-HX;**

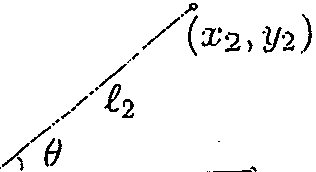
Area of triangle *( xo ,Yo ), ( x1 ,Y1 )*

and (x2 , y2 ):

.-1 abs IX1 - *Xo Yl* - *·vo* I .

2 *Xz* - *Xo Yz* -*Yo*

Angle formed by three points:



(0, 0) l'1 (x1, Y1)

*cos e* = *(x1,Y r )* · *(x2 ,Y2 ) .*

f1l'2

Line through two poiuts *( xo , Yo)*

and (x1, y1):

*X y* 1

*xo Yo* 1 = 0.

X1 Y1 1

Area of circle, volume of sphere:

4 3

*V* = .

 *31rr*

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

- Issac Newton

*r*--;\ -

*d( sec u) du*

* 1. . = tan u *sec·ud -* ,

*X X*

*d( csc u) du*

* 1. . *- cot u csc u-,f* ''



=

*X d x*

*d(* arcsin *u)*

=

15.

I *cl-u* ,

16. *d(* arccos *u)* = -1 *du*

*dx* v 1- u2 *dx*

1 . d(arctan *u)* = \_1\_ *du ,*

7

*dx* v'l - *u2 dx* '

d(arccot u) -1 *du*

18

*d(*

19.

1+ u2 *dx*

arcsec *u)* = 1-*clu* ,

*dx* uv l - u2 *dx*

' · = 1+ *u2 dx '*

1. d(arccsc u) = -1 *du ,*

*uv1I u2 dx*

d(sinh *u)* h *d-u*

1. . = cos *ud-*,

*X*

23. d(tanh *u)* = h2 *udu* ,

. sec *d-*

*X*

cl(sech *·u)* h *du*

d(cosli u) \_ . h *du*

*·x* - sm *dx*

22. *u* ,

d(coth u) \_ \_ h2 *du dx* - csc *&c*

24. *u* ,

d(csch *u) du.*

25. = -sec11 u tan *u*

*d-*,

1. . = - csch *u* coth *·ud -*,

*dX X*

*d(* arcsinh *u)* 1 *du*

1. *dx* = v'l + u2 *dx '*

*X X*

1. d(arccosh u) = 1 *cl·u*

*d x* v'u2 - 1*dx '*

29.

-

31.

*d(* arctanh *u)*

*cl(* arcsech *u)*

1 *du*

= --,

-1 *du*

= ,

30. d(arccoth u) = *\_l\_' du*

1. d(arccsch u) = -1 *du \_*

*\cj*

*d x* 1 - u2 *dx*

*dx* u2 - 1*dx '* 0\

*dx*

Integrals:

*·uv l* - u2 *dx*

*dx l ul v' I* + u2 *d x* ·

* 1. *j cu.d x* = c *j udx ,* 2. *j (-u* + *v) dx = j u dx* + *j v dx ,*

' 1

*J* n + l

*n f -I ,*

1. *j .!.clx* = ln x,
2. *J ex dx* = *eX,*

3, xn *dx* = --xn+l,

*J* l + x

*X*

*J dx dx*

*dx*

6. -2

= arctan x,

* 1. *udv dx* = *uv* - *j ·ud--udx ,*
  2. *J sin x dx = - cos x ,*

10. *j* tan x *d x* = -ln Icos *xi ,*

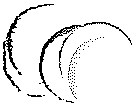
12. *j* sec x dx = ln l sec x + tan xl ,

* 1. *j* cos *x dx* = sin x, 11. *j cot x d x* = ln I cos *xi ,*

1. *j csc x dx = In l csc x + cot xl ,*
2. *j* arcsin *!dx* = arcsin ;+ ..,/a2 - x2 , *a >* 0,

|  |  |
| --- | --- |
| Theoretical Computer Science Cheat Sheet | |
| 7r | Calculus |
| Wallis' identity:  7r = 2 · 2 · 2 ·4 · 4 · 6 · 6 ···  --------  1.3 . 3 . 5 . 5 . 7 .. .  Brouncker's continued fraction expansion:  12  *1*= 1+ 2 + - 32  2  2+--5,,--  2+  Gregrory's series:  {= 1 - }+ t - t +!- .. .  Newton's series:  " 1 I I ·3  6 = 2 + 2 . 3 . 23 + 2 . 4 · 5 · 25 + · · ·  Sharp's series:  :tt =\_l (i \_ \_\_ I 1 )  6 v3 31 · 3 + 32 ·5 - 33 . 7 + . . .  Euler's series:  ,r2 \_ l 1. + 1 1 + 1  6- l2 + 22 32 + 42 *?*+ . . .  8rr\_- !2 +"§12 + 512 + 7l2 + 912 + ...  2  rr 2 l l l 1 + l  l2 - l2 - 22 + 32 - 42 52 - . . . | Derivatives:  *d( cu) du d( u* + *v)* \_ *du dv d( uv)* \_ *dv du*  1. *-d-= cd-'* 2· *dx* - *dx* + *dx '* 3' - *udx* + *v dx '*  *X X*  4 d(un ) \_ n-1*du* 5. *d( u/ v)* \_ *v ( ) - u( t)* , 6. *d( ec" )* = *cecu du* ,  . *- nu dx ' d-* - 2 *d-* -  *X V X d x*  7. *d( cu)* = (ln c)c" du , 8  . d(ln u) *= .!\_ du*  *d x d x dx u d x '*  d(sin *u)* \_ *du d( cos u)* \_ . *d-u*  9. *dx - COS U dx '* 10. . - - sm *udx* ,  d(tan *u)* 2 *du d( cot u)* \_ 2 . *du*  11. . = sec *u dx* , 12. . - csc *ud-*,  *X X* |
|  |
| Partial Fractions |
| L,:,I: N(:r;) and *D( x )* be polynomial fun.c­ tions of *:c.* We can break down *N ( :1: ) / D( x )* using partial fraction expan­ sir.111. Fi rst, if the degree of *N* is greater than or equal to the degree of *D,* divide *N* by *D,* obtaining  *N( x ) N'( x )*  *D( x)* = Q(x) + D(x) '  where the degree of *N'* is less than that of  *D.* Second, factor *D( x ).* Use the follow­ ing rules: For a non-repeated factor:  *N ( x ) A N'( x )*  (*x----a'-)-D-'- (-x )* = *x - a* + *D( x )* '  where  *N ( x ) ]* .  *A = [ D( x ) x=a*  For a repeated factor:  *N ( x )* \_ *Ak* +*N'( x )*  - *L..,* (x - *a)m-k D( x )* '  *k=O*  where 1 [ *dk* (N(x) )] .  *Ak* = k! *dxk D( x ) x=a* |
| The reasonable man adapts himself to the world; the unreasonable persists in trying to adapt the world to himself. Therefore all progress depends on the unreasonable.  - George Bernard Shaw |

I I



\*(*S.

15. *j*

17. *j*

Theoretical Computer Science Cheat Sheet

Calculus Cont.

arc.cos *';;\_ clx* = arc.cos - *-J*a2 - *x 2 , a >* 0,

16. *j*

arctan *;dx* = *x* arctan ;- % ln(a2 + :r:2 ), *a >* O,

*sin2 ( ax)d x* = 1 *( ax* - *sin( ax) cos( ax) ) ,*

20

18. *j cos2 ( ax)d x* = *la ( ax + sin( ax) cos( ax) ) ,*

19. *j*

21.

*J* sm.

sec2 *x dx* = tan *x ,*

" *x &c* = - sinn-l x c.os x: + *n*-

1*J* . -

smn *- x d:1.:,*

20. *j* csc2 *x dx:* = -cot :1;,

*J*

1*J*

*cosn-- :c d:r ,*

23.

tann :t chc =

.

tann-l *x J*

*n*

*n*

22.

cosn *x dx* =

cosn-l x sin x + *n*-

- tann -2 x dx, *n /* I,

?

24. *cot n x dx* = *n* \_ - cot"-- *x dx , n -:j:.*l,

*J*

cotn-l *x J*

*n*

*n*

*J*

25. sec"' *x clx* =

*J*

tan x sec"-1 *x n* - 2 *j*

*11* - 1

1

+ -- secn-2 .r *d:c , n /* 1,

1 . c.sc71 *x dx* = -

*J*

*cot x cscn-l x n - 21*

n - 1

n - 1

+ -- cscn -2 *x &r:,*

*n ,f:.* 1,

27. sinh *x dx* = cosh *x,*

*J*

28. cosh:c *dx* = sinh *x ,*

*J*

,, . n - 1 n - 1

29. / tanh x cl:r = ln / cosh x /, 30. / coth x dx = ln / sinh x/, 31. / sech x d:c = arctan sinh x, 32. / csch :r dx = ln / tanh f / ,

33. *j* sinh2 *x d:r.* = :i sinh(2x) *- ix,* 34. *j* cosh2 *x clx* = i sinh(2x) + *ix ,* 35. *j sech2 x clx* = tanh ::r.,

36. *j* arcsinh *dx* = ::t arcsinh - *-Jx 2* + a2 .' *a >* 0, 37. *j* arctanh *dx* = *x* arctanh + % ln */a 2* - *x2 / ,*

*x* arccosh :'.. - *J*x2 + *a2 ,* if arccosh > 0 and *a* > 0,

*J* {

38. arcc.osh *'£dx* =

*a x* arccosh - + *J*x2 + a2 , if arccosh ;< 0 and *a* > o·,

*a*

39. *J .dx* = ln *(x* + v'a.2 +:r2) , *a >* 0,

*J a2* + *:c2*

*J dx*

40. --- = arc.tan £,

+

*a.2* ;1;2 *a a*

*a. >* 0, 41. / · *1*a2 - x2 *clx* = .:J'. · *1a2* - *x 2* + 02 a.resin £

*a >* 0,

42.

V 2 Y 2 *a '*

((1(

*j ((i2 - x2 ) 312 d,r.* = *(5a2* - 2:.r.:2 ) /a.2 - x2 + 3 ·a.resin . *a >* 0,

·, -· ' - . *J*

*j*

*J a2 - x2*

*clx*

= arcsin £

*a '*

*a* > 0, 44

•

,a,2 - x:2

= J\_ ln

*2a a - x '*

*/ a + x* /

45. 2 *dx I* = *x* ,

*( a* - x2)3 2 *a2 ,/ a2 - x2 ·*

*j*

46. *j J a2* ±:r:2 *ch: = {Ja2* ± *x2* ± *a;* 111 /:1:+ *J az* ± x2 / *!* 47.

*dx* = ln */ x* + *J*x2 - a2 / , *a* > 0,

. *J x2* - a2

*J*

*I*

48 · *Ia:1./:b:1:* = In *I a:bx I* '

*2( 3bx* - *2a)( a* + *bx) 312*

49. . *:"C J a.+ bx dx* = lSb2 ,

*j /a'+bx dx* = *2 J a + bx + ct / clx ,*

*j x dx* = -1 ln / *v'a+bx* -

*,/al*

*, a >* 0,

5.

*X* · *:t*

I

*a + b:1:*

51.

-

* *Ja* + *bx* J2 *J a* + *bx* +

*,/a*

52. *j J* a2 - *x2 dx* = *J*a2 - *x2* - *a* ln

*a + J* a2 - x21 ·

53. *j x J a2 - x2 dx = -H a2 - x 2 ) 312 ,*

*X X*

54. *j x2 J a2 - x 2 dx = H 2x2 - a2 ) J a2 - x2 +* 4 arc.sin ,

*, x dx*

*a. >* 0,

57.

55. *j dx* = -l!n / a + ,/a2 - x21 ,·

*J a2* -:z:2 *a X* .

*J x dx* 2

*J*

*2*

*j J a.2* \_ *x2*

-/a.2 *- x z ,*

a2 \_

2 - u.,

56. =

-*J*-;::=

x:2:;: = \_ £ /*a2* -;1;2 + , arc.sin *iE.. a >* 0,

1. *j ..J a2* + *x2 dx* = *J*a2 + *x2* - *a* ln I*a* + *J* a2 + x2 I,

*X X*

1. *J xz* - *aZ dx* = *J x* *2* -*a2* - a arccos ..!!:... *a >* 0,

*X* /i / '

*- J x J xz ± a2 dx = i( x2 ± a2 )3 f2 ,*

61'·

*d x*-- 1111 *I x*

') I'

*J x J x2 + a2 a a + ,/a 2 + x-*

r> \

Theoretical Computer Science Cheat Sheet Calculus Cont.

Finite Calculus

62. *J* ,.,.,/,*=*.,.*d\_*2*x*\_ *n2* - *a*1 arccos *r;ay , a >* 0,

63. *j dx* - -v'x2±a2

Difference, shift operators:

*x dx* - *x2 ,.jx2* ± a2 - =F *a2 x* '

*J*

*6.f ( x )* = *f ( x* + 1) - *f ( x ) ,*

64. .*I J x2* ± a2 = *J x2* ± a2,

65. *,.jx2* ± a2 *d* - (x2 + a2)3/2

I

*x* =F ,

*E f ( x )* = *f ( x*

+ I) .

*x 4* -

*j* 4aJ

*3a2x*

Fundamental Theorem:

*d* 1 1 *2a.x* + *b* - yb2 - *4ac*

66. *ax'+;,*+c - *v'b'* - *4ac* n *2ax* + *b+ v'b'-*

if *b'* > *4a,,*

*f ( x )* = *6.F( x )* ¢'? *L f ( x )ox* = *F ( x )* + *C.*

*b b-l*

{ 2 arctan *2 ax* + *b* 2

I::: *f*

*( x )ox* = *Lf* (i).

v4ac - *b2 ,/4ac* - b2 ) if b

*1n ! 2ax + b + 2vaJ ax2 + bx + cl ,*

*J*

67 *dx* \_ y a

< *4ac,*

if *a >* 0,

*a* i=a

Differences:

*bi.( cu)* = *cbi.u, ( u* + *v)* = *bi.u* + *bi.v ,*

*bi.( uv)* = *ubi.v* + *E vu,*

· *-/ ax2 + bx +* c - {

1 . *-2ax* - *b*

*1*

arcsm

*2*

y -a *,/b*

. ,

- *4ac*

if *a <* 0, ·

(x!!:.) = *nxn- ,* -

*J J 2ax* + *b* V *4ax* - *b2 J dx*

*( Hx )* = x- 1

=

,

bi.(2:r ) = 2:i: , f )

*( X )* = ( *X* ) . ,.,

*68. ax2* + *bx* + c *d x* = -

.

-- *ax2* + *bx* + c + " ,

4*a 8a*

bi.(c"' )

Sums:

(c - l)cX,

m m-1

69. *j x dx* = *vax2* + *bx +* c \_ \_!:\_ *j dx*

*-/ ax2 + bx +c a 2a* " · ' '

--ln *1 2-/c-/ax2 + bx + c + bx + 2c l* ,

if C > 0,

*I: cu ox* = *c I:u ox ,*

*I:( u* + *v) ox = I:u ox* + *I:v ox, I:·ubi.v ox* = U'(l - *I: E v6.u ox,*

70. *I dx* = VC *x*

n+l

0*'""' ",,'.nu'X* --:mr.+.J-)

I:x-l *Ox = Hx ,*

. *x -/ ax 2 + bx + c*

1 . *bx + 2c*

*r-:.* arcsm ,

{

v *-c l xl -/b2* - *4ac*

if C < 0,

I:*c"' 0:1:* = cl ,

I:(:,) *O:r:* = (m:1) .

71. *j* :r:3 *J*x2 + *a2 dx* = (*ixz* - fi5(t2) *( x2* + a2)3/2 '

1. ./*x"* sin(ax) *dx* = *-ixn* cos(ax) + *j xn-l* cos(ax) *dx ,*
2. ./ *xn cos( ax) d x* = *itc"* sin(ax) - ./ *xn-l* sin(ax) *dx ,*

*J*

Falling Factorial Powers:

*xn* = *x( :t* - 1) ·· · *( x* - *n* + 1), *n* > 0,

*XQ =* 1,

*n* 1

*x-* = (*x + l*) *· · ·* (*x +* I*n*I) , *n* < o.

*x* = *x.!!!.( x* - m)!!..

1. *j . xneax dx* =*-xnea,·* -·

*a-*

*x"-l eax dx ,*

Rising Factorial Powers:

*xn* = *x( x* + 1) ·.. *( x* + *n* - 1), *n* > 0, *\_)*



1.

1. *j* xn *ln( ax)·d x* = *xn+l* ( ln(ax) - 1 ) ,

*n* + 1 *( n* + 1)2

*J*

*x0* = 1,

- 1

1. *J* xn(ln axr *dx* = *-xn+1*

*-(*

*lnaxr* --

*m*

- xn (ln ax)m-l *dx .*

xn =· *n* < 0

(x -1) - ··(x - jnl) ' '

*xi =*

*x2* -

*. n + I* · n + l

*x!.*

*x1+ x!.*

xr

*:1:2* - xr

*xn+m* = *xm( x* +*m)n.*

Conversion:

*xn* = (-l)"(-x)11 = *( x* - *n* + l)n

= 1/(x + l)-n ,

*x3* =

*x4* =

*x5* =

*xr* =

x2 =

*x1+* 3x1+ *xl. xi+* 6x1+ 7*x1* + *x!.*

*xi* + 15xi + *25xi* + 10x1+ *xl.*

xl

*x2 + xi*

=

*x!.* =

*xl* =

*x3* - 3x2 + x1

*x 4* - *6x3* + *7x2* - x1

*xi* - 15x4 + 25x3 - 10x2 + x1

xl

*x z* - xl

*xn* = (-1)"(-x)!!· = *( x + n - 1)1!.*

. = 1/(x - I).=.!!.,

xn = *t { :}xk =* t{;}(-1)"-k:i:'<,

*t*[:]

k=l k=l

*x!l* = *( -I)n-k xk,* .

*x3* =

*xi=*

*XS =*

*x 3* + 3x2 + 2x1

*x 4* + *6x 3* + llx2 + 6x1

*x 5* + 10x4 + 35x3 + 50x2 + 24x 1

*x1=*

*xi = xi =*

*x 3* - 3x2 + 2x1

*x 4* - *6x3* + llx2 - 6x1

*x 5 10x4* + 35x3 - 50x 2 + 24x 1

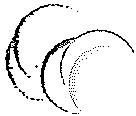
*xn* = *t [ n] xk .*

*k*

'"'

J

k=l



Taylor's series:

Theoretical Computer Science Cheat Sheet Series

Ordinary power se1:ies:

'*x - a*)"*-*

+ + '

*J ( x )* = *f ( a) (:i::* - *a)J' ( ci)*

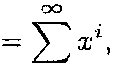
2

co (*x a* ' *·*

*f" ( ci)* + · · · = *L ) j(il ( a).*

*A( x )* = *I:ai xi.*

. i=O '

Expans10ns: ·

1

= 1+ *x* + x2 + a,3 + x4 + . . .

i=O

Exponential power series:

**oc *x i***

1-x

1

1-*ex*

i=O

00

 = *I:ci xi,*

i=O

00

*A( x )* = *L ai ·*

,:=O 2.

Dirichlet power series:

1 = 1+ *x"* + *x2n* + *x3n* + . . .

*X* = *x* + 2x2 + *3x3* + 4x4 + . . .

= *I: xni ,*

*i=O*

oc

= · *i*

*i=O*

*A( x )* = *L.,* i .

i=l



*ix*

Binomial theorem:

*( x* + *Yt* = *t(;)x"-kyk .*

(1 - *x ) 2*

*k dn* ( 1 )

*X dxn* 1 *- x*

= *X* +

*L....,'l X '*

2nx2 + *3nx3* + 4nx4 + . . . = *Linxi'*

=

i=O

**oc. x,:**

*k=O*

Difference of like pow0rs:

n-1

ln(l + *x )*

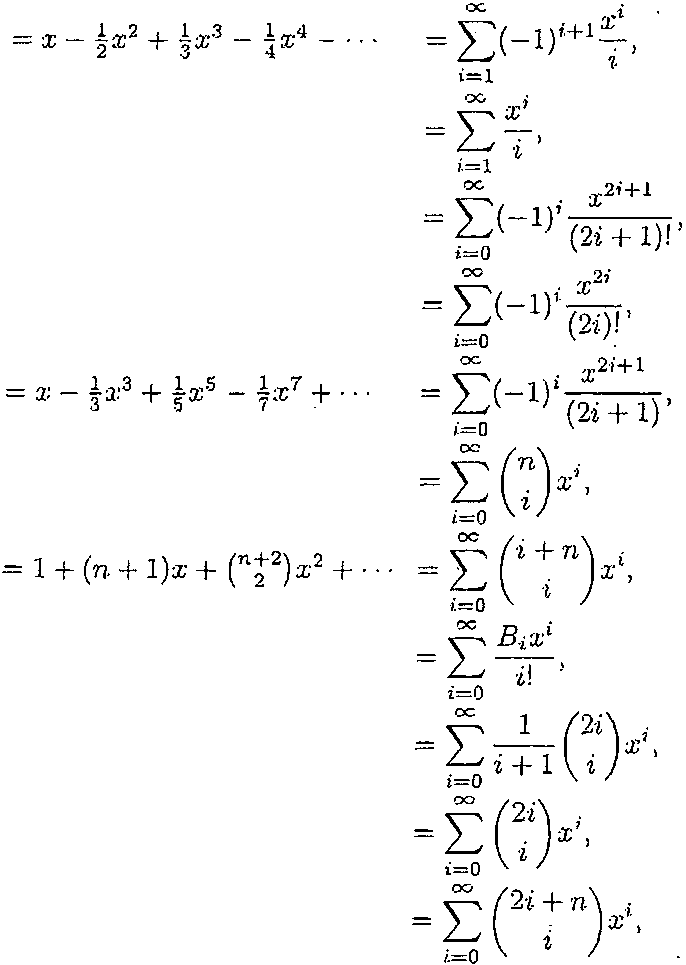
= 1+ *X* + i:1:2 + *ix3* + . . .

= *L -::T ,*

i=O *t .*

*xn - y"* = *( x - y ) Lxn-1-k yk .*

*k=O*

For ordinary power series:

oc

1

- -

In -·

1- x

*sin x*

cbs a;

= *x* + *!x 2* + *3-x 3* + *± x-t* + · · ·

= *x* - *.*3*l*!*.x3* + .5l!.:r.s - .7l!.a:.7 + . . .

= 1\_ *l..x2* 1...7,4 \_ *l..x6*

+ + . . .

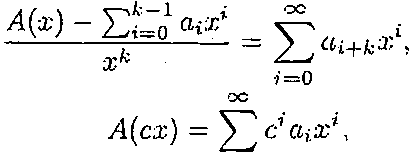
2! 4! .... 6!

i=O

OC,

:1: :r = L...., *a;-k X* ,

*,.,A(.* ) . ;

*i=k*



(1 *+ xt*

= 1+ *n:1.·* + ii(n-l) x2 + . . .

2

*i=O*

OC,

*L(i*

*A'( x )* =

i=O

OC,

+ *l)a.;+1 xi.*

1

*X*

*e:r* - 1

1

-(1- /1*- 4x)*

*2x* .

1

*J I - 4x,*

1 (1-*Jl=-Tx )* n

= 1- *lx* + /2 *x2* - 70 *x'l* + . . .

= 1+ *x* + *2:c2* + *5x3* + . . .

= 1+*·x* + 2x2 + 6x·3 + . . .

4 11 2

*.cA'( x )* = *I:·ia;x\*

i=l

(

*J A x ) d x* = *-a;:-i-1x*i*,*

1.=l

*A( x)* + *A( -x) 2i*

? = L...., *a2; X* ,

- *i=D*

*A( x )* - *A( -x)* 2i+J

2 = L...., *a2i+1X* ·

i=O

*J I - 4x* 2::c

= 1+ (2 *+n)x* + ( 1) x

+ . . .

Summation: If *bi* = )=o *a;* then

1 1

- -

-ln -

1-x 1-x

oc

= *x* + *lx2* + 11x3 + *2s x4* + . .. '= *H,xi ,*

2 6 12 L...., •

·i=l

*B( x )* =

Convolution:

1

1\_ *x A( x ).*

2

2 1- *X*

L...., . '

*A(::c )B( x )* = f (t.*aj b,-j)*

-1 (ln 1-)

*X*

=*l*2 *x*,*2* +*l*4 *x*,*3* +.

*2* 3

12-11''.'4 + . . .

*4*

= *Hi-l Xi*

·i=2 t

oc

*i=D j=D*

*xi.*

1 - *X* - *x 2*

= *x* + *x*

+ 2x

+ *3x*

+ · ··

= *L F;xi,*

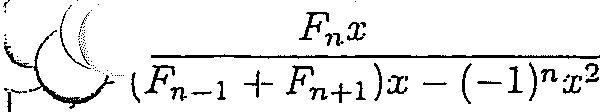
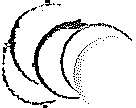
i=O co

= *L Fni Xi .*

i-0

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

- Leopold Kronecker



*,("*

Expansions:

Theoretical Computer Science Cheat Sheet Series

Escher's Knot

1 Ii 1-

(l - x)n+l 1- x

*xn*

(1n 1 xr

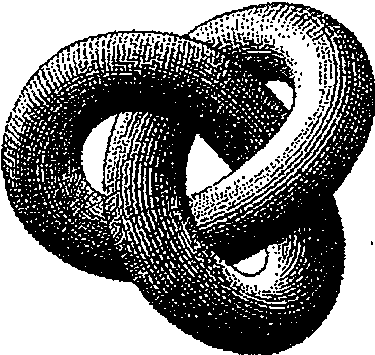
= f*( Hn+i* - *Hn )* (n + *i)x i,*

i=O i

= f [ ]xi,

i=O i

( *)-n* = {:}xi,

*( e'"* - lf = f { i}n!xi



i.=O *n* i! '

f (

=

= f [ i ] n!t '

i=O *n* i.

*x cot x*

-4)iB2;X2i

i=O (2i)! '

tan *x*

00

= L(-l)i-1

22·( 20-·' -

l)E2iX2i-l

((x)

00

- *!,-;* 1

i=l (2i)!

'

1 = µ(i)

*( ( x* - 1)

*iX* >

•=l

f

= ¢(i)

((x)

*( ( x )*

*L\_; ·x* >

i=l i

= IT . 1 \_ ,

((x)

i=l

i'" ,

Stieltjes Integrntion

(2(:.c:)

*p*

OO d(i)

= . 7 where *d(n)* = Edin 1,

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

*lbG( x ) d F( x )*

((x)((x - 1)

= S(i) where *S( n)* = *E din d,*

*x'*

i=l

exists. If *a* ::; *b* ::; c then

*,[ G( x ) d F( x)* = *1b G( x ) dF ( x )* + *G( x ) d F( x).*

*le*

((2n)

22 jB2n l 2n

-- (2n)! 1f ,

n-l

*n* E N

l

If the integrals involved exist

*t*

*X* oo

. 1 (4; - 2)B2,·x2i

*1b ( G( x ) + H ( x ) ) d F ( x )* = *.lb G( x ) d F ( x )* + *lo. H ( x ) d F ( x),*

sin x

= L(-1)'- (2i)!

*i=O*

*1b 1b f b*

( 1 - )n

= *-n( 2i + n* - 1)! i

*G( x ) cl( F( x )* + *H ( x ) )* =

*G( x ) d F ( x )* + *la G( x ) d H ( x ) ,*

*2x* i=O

\_ *L*oc

:*t* f*.*(*n* -J- 1..)! *X* ,

2'·;2 sin :h!. .

*1b c · G( x ) d F( x ) =*

*f b t*

*la G( ;) cl( c · F( x ) ) = c la G( x ) d F( x )\_*

,,., sin x

- . ·1

4 *X* ' *, lb lb*

i=l i.

v - (4'i)!

00

*G( x ) d F( x )* = *G( b)F( b)* - *G( a)F( a)* -

F(x) *dG( x ).*

.arc:m.

)2

\_ '\"' *x* i*,*

- 16i y'2(2i)!(2·i + 1)!

•=0

x

Ifthe integrals involved exist, and *F* possesses a derivative *F'* at ever point in [a, b] then · /

( i=O

\_ , ,\_ .

x2;

- '' .

*1b G( :.c:) clF( x )* = *lb G( x)F' ( x ) dx.*

= '\o"c ' 4i·i!2

Cramer's Rule

If we have equations:

*a1,1X1* + a1,2X2 + · · · + *a1,n Xn* = *b1*

a2,1X1 + a2,2X2 + · · ·+ *a2,n X n* = b2

*an,1X1* + *an,2 X 2* + · · ·+ *an,n Xn* = *bn*

Let *A = ( ai,j )* and *B* be the column matrix *( b;).* Then there is a unique solution iff det *A -of* 0. Let *A;* be *A* with column ireplaced by *B.* Then

det A;

*X;* = det A ·

Improvement makes strait roads, but the crooked roads without Improvement, are roads of Genius.

- William Blake (The Marriage of Heaven and Hell)

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

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

*n* = *Fk ,* + *Fk 2* + · · ·+ *Fk= ,* where *ki* 2'. ki+ 1 + 2 for all i, 1 ::; i< *m* and *km* 2:: 2.

Fibonacci Numbers

1, l,2,3,5,8,13,21,34,55,89,...

Definitions:

*F;* = F;-1-I-F;-2, *Fo* = F1 = 1,

*F\_;* = (-l)i-i F;,

*Fi =* v1'5 (¢i -¢Ai) ,

Cassini's identity: for i> 0:

*Fi+1Fi-1* - F;2 = (-lf

Additive rule:

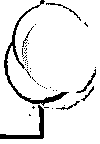
*Fn+k* = *Fk Fn+l* -1- *Fk-1Fn , F2n* = *Fn Fn+l* + *Fn-l Fn ,*

Calculation by matrices:

Fn-2 Fn-1) = (0 l )n

(

*Fn-1 Fn* 1 1



- ">:< u'"" nvi.ca,:;w.,,:: - V t:f/SlOll 1.,:::, (A•!j April 20, 2007 1

1 Notations

" The symbol co,,,, for const.

* The symbol *.n* for *function returned value.*

e Template class parameters lead by outlined character. For example: 11', lKey, !Compare.

2.2.2 Members & Operators X::X();

X::X(<•nst X&);

X::-X();

X& X::operator=(c0n,t X&);

S:iterator S::erase(S::const..itcrator *firni ,*

*.n post erased* S::const..iterator last); void S::push\_back(const S::valuc\_type& :i:); void S::pop\_back();

S::refcrence S::front();

void *II* move *x's [xFirst,xLast) before pos*

list,::splice (it.era.tor *pos,*

list,('lI')& *x,*

it,erntor xFirst,

itera.tor xLa.st); D"1'7.2

void list::remove(con,t 1l'& *vaiue);*

Interpreted in template definition context.

e Sometimes class, typename dropped.

* Templa.te class parameters dropped,

X::iterator

X::const..itera.tor X::itcrator

X::begin();

X::begin()

X::end();

**const .**

S::const\_reforence S::front() const :

S::reforence S::back(); S::c:onst\_refcrenc:e S::back() canst ;

void list::remove\_if (lP'redicate *vred );*

*II a fter call:* 'ef *this* iterator *1' , \*)) i= \*(1'* + 1) void list::unique(); *II remove repeats*

thus C sometimes used inst.ea.cl of C('lI').

'" "See exa.mple" by o.,-, its output by ,;;,11111t-.

X::const.\_itenitor X::reverse.iterator

X::end()

X::rbegin();

**consl ;**

2.4 Vector

void *II as before bi.rt, -,binPred(\*V , \*(JJ* + 1))

tist,::unique(BinaryPredicate *binPred.);*

2 Containers

X::const\_reverse\_itera.tor

X::reverse.iterator

X::rbegin()

X::rend();

**const ,**

#include <vector>

*JI -Assuming both this and x sorted*

void list.::merge(list('lI')& :r.);

X::const ..reverse\_iterator

X::rend() =**const .**

*II merge and assume sorted by cmp*

2.1 Pair

#include <utility>

X::size\_type X::sizc\_type. boo!

void

X::size() const ;

X::n1ax\_size() con,,;

X::empty() canst ;

X::swap(X& x);

Sec also 2.2 and 2.3.

t.cmplate(class 1l',

class *A.* lloc=allocator) class vector;

sizc\_type vector::capacity() con,,;

void list::merge(list(1l')& :r., Compare cmp);

void list::reverse(); void list.::sort();

void X::clear();

templatc(class 1'1, class 1'2) struct pair {

'.['1 first; 1['2 second; pair() {}

pa.ir(const 'Jl'l& a, co,,st 'JI'2& b):

first(a), second(b) {} };

void vector::reserve(size\_type *11-);*

vector::reference .

void list::sort(Compare

*cmp*);

2.2.3 Comparison Operators

vector::operator[J (size\_type i);

* 1. Sorted

Herc A any of

Associative

Let, X *v, w.* X may also be pair (2.1).

vector::const\_rcfcrnnce vector::operator[] (size\_typc i) ;

{set, multiset, map, multimap}.

2.1.1 Types

***V* == 1.Ll**

**11** < *·w*

tl <= *w*

*V* !: ***W***

*V* > *W*

*V* >= *W*

I& 7.1.

2.5 Deq1ie

2.7.1 Types

For A=(multi]sct, columns a.re the same .

pair::first\_type pair: :second\_type

2.1.2 Functions & Operators See also 2.2.3.

All done lexicographically and .nbool.

2.3 Sequence Containers

S is any of {vector, deque, list}

#include <deque>

template(class 'JI',

class A.lloc=allocator) class deqtie;

A::key\_type A::value\_type

A::keycompare A::value\_co111pare

2.7.2 Constructors A::A(Compare c=Compare())

pair(1l'l,'li'2)

make\_pair(con,t 1I'1&, canst 1I'2&);

2 .2 Containers -Common

2.3.1 Constructors 5::5(5::size\_type n,

con,t S::value\_typc& *t);*

Has all of vector functionality (see *2 A).* void dcque::push\_front(const 'JI'& x); void dequc::pop..front();

A::A(A::const,\_itcrator

A::const.\_iterator

Compare

2.7.3 Members

first,

last,

c=Compare());

Here X is any of

{ve·ctor, deque, list,

set, multiset, map, multimap}

2.2.1 Types

S::S(S::coist\_iterator *first,*

S::const\_iterator last); *IG!"7.2. 7.3*

2.3.2 Members

$::iterator *II inserted copy*

2.6 List

#include <list>

t.cmplate(class 1l',

class A.lloc=a.llocato1) class list;

A::lrnycompare A::key \_co111p()· const ; A::valuc\_comparc A::value\_comp() i

A::iterator

A::insert(A::iterator. *hint,*

con,, A::va.lue\_typo& *va.l);*

X::value\_type X::reference X::consLreference X::iterator X::const \_iterator

X::reverse\_iterator

S::insert(S::iterator

**canst S::va.lue\_type&**

S::iterat.or *II inserted copy*

S::insert(S::iterator

S::size..type

canst S::value\_t.ype&

S::iterator *II inserted copy*

before,

*val);*

before, *nVal\_ val );*

See also 2.2 and 2.3. void list::pop\_front();

void list,::push..front(= T& :r,);

void *II move all x (&x # this) before pos*

list,::splice(iterat.or *pos,* list(1l')& *x );* 11'.?7.2

void A::insert(A::itcrator *first,*

·A::iterator last); A::size\_type *II # erased* A::erase( A::kcy\_type& *k );*

void A::erase(A::iterat.or *JJ* );

void A::erase(A::iterator *first,*

X::const reverse\_iterator

X::difference\_type

S::insert(S::iterator

S::const\_iterator

before, first,

void *II move x's xE/emPos before pos*

list::splice (iterator *pos,*

A::sizc..type

A::iterat.or /a. t);

X::size\_type

S::const\_iterator

*last);*

list.(11"'} &. *x,*

A::count(<on,t A::key\_typc& *k )* con,t ;

Iterators reference value\_t,ype (Sec 6).

S:itcrator S::erase(S::iterator *vosition):*

it.ern.t.or xEJemPos); 111N' 7.2

A::itcrator A::find.( A::key\_type& 1..) canst ;

Y · '111 Medini © 1998-2007

.. \:::)

*\..::;;*

*(,*.. .

*::5/i*

*·-..::::J;*

\* http: *I*/www .medini .org/stllstl .htrf'

'::7\

*·"d)*

1 votarn .medin;'1i. l .com

- )





. . *r* ' *V*··, STL Quick Reference - Version 1.29 (A4] April "''\7

I l

o..

0.1 *)* '-··· "' '

A::iteratr , 2.10.2 Members multimap::const..iterator void

A::lower\_bound( A::key\_type& k) . multimap::lower\_bound( queue::push( Container::v-d.lue\_type& x);

A::iterator A::upper\_bound(< 0

•"

A::key\_type& k) conrt ;

map::map(

Compare& cmp=Compare());

con,t multimap::key\_type& k) con,t ; j void queue::pop();

multimap::const..iterator

pair(A::iterator, A::iterator) // see *4.3.1*

pair(map::iterator, boo!) // *bool* = if *new*

multimap::upper \_bound(

con,t tContainer::value.type&

queue::front() ;

A::equaLrange(co•st A::key\_type& k) co"";

""

map::insert(c0

map::value\_type& :z:);

multimap::key\_type& *k )* con,,;

pair(multimap: :constJterator,

tContainer::value.type& queue::front();

* 1. Set

#include <set>

'JI'& map:operator [](con,,map::key\_type&);

map::const\_iterator map::lower\_bound(

const m.ap::key\_type& *k )* Const ;

map::const..iterator map::upper \_bound(

const map::key\_type& *k )* con.st j

multimap::const..iterator)

multimap::equaLrange(

const multimap::key\_type& k) con,t ;

3 Container Adaptors

c •" Container::value\_type& queue::back() const ;

Container::value\_type& queue::back();

0

Comparision Operators boo! operator= =(con•t queue& *qO*

con,t queue& *ql );*

pair(map::const..iterator, map::const..iterator) map::equaLrange(

template(class lKey,

class <Compare=less(lKey), class Alloc=allocator)

class set;

3.1

Stack Adaptor

boo] operator<(con,t queue& *qO,*

con,t queue& *ql );*

See also 2.2 and 2.7. map::key\_type& *k )* const ;

set::set(c00 , Compare& cmp=Compare());

#include <stack>

3.3 Priority Queue

pair(set::iterator, boo!) // bool = *if new*

Example

#include <queue>

set::insert(c0

""

set::value\_type& *x* );

typedef map<string, int> MSI; MSI nam2num;

nam2num.insert(MSI: :value\_type ("one" , 1)) ;

Default constructor. Container must have

template(class 'JI',

class Container=deque('f ) ) class stack;

t.emplate(class 'JI',

class Container=vector('JI'), class Compare=less('ll') )

class priority \_queue;

2.9 Multiset

#include <multiset.h>

nam2num.insert (MSI: :value\_type ("t"o", 2) ) ; nam2nwn. insert (MSI: :value\_type("three" , 3) ) ; int n3 = nam2num("oe11 ] + nam2num["tYo 11 ) ; cout << n3 << " called 11

back O , push\_back () , pop\_back () . So vector, list and deque can be used. ·

boo! stack::empty() con,t j

template(class lKey,

class Compare=less(lKey), class Alloc=allocator)

class multiset;

;

f or (MSI: :const\_iterator i = nam2num.begin() ;

i != nam2num.end0 ; ++i)

if ( (\*i) . second = n3)

{cout « (\*i) .first « endl;}

Container::size\_type st.ack::size() ;

void

stack::push(conrt tContainer::value.type& x);

void stack::pop();

Container must provide random access iterator and have empty() , size() , f ront () , push..back O and pop\_backO . So vector and deque can be used.

See also 2.2 and 2.7. multiset::multiset(

@ 1111• 3 called three Container::v-d.lue\_type& stack::top() conn ;

Mostly implemented as *heap.*

const Compare& cmp=Compare() );

multiset::multiset(

2.11 Multimap

#include <mult.imap.h>

void Container::value.typ& stack::top();

Comparision Operators

1

3.3.1 Constructors

explicit priority queue::priority \_queue(

linputiterator *first,*

linputiterator *last,*

template(class lKey, class 'JI',

boo! operator==(C0"'

stack& *so,*

Compare& comp=tCompare());

const Compare& cmp=Compare() ); multiset::itera.tor // *inserted copy*

multiset::insert(con,, multiset::va.lue\_type& *x* );

2.10 Map

#include <map>

template(class lKey, class 'JI',

class Compare=less(lKey), class A lloc=allocator)

class map;

See also 2.2 and 2.7.

2.10.1 Types

class tCompare=less(lKey), class Alloc=allocator)

class multimap; See also 2.2 and 2.7.

* + 1. Types multimap::value\_type // pair(co"" lKey,'JI')
    2. Members

multimap::multimap(

const Compare& cmp=Compare());

multimap::multimap( linputiterator *first:,*

linputiterator )a.st,

con,t stack& *sl );*

boo! operator<(const stack& *sO,*

conrt stack& *sl );*

3.2 Queue Adaptor

#include <queue>

templa.te(class 'JI',

class tContainer=deque ('JI') )

class queue;

Default constructor. Container must have empty() , size () , back (), f ront () , push...back () and pop..f ront (). So list and deque can be used.

boo! queue::empty() const ;

priority\_queue: :priority \_queue( linputiterator *first,* linput!terator *last,*

con,t Compare& comp=Compare());

3.3.2 Members

boo! priority\_queue::empty() const ;

Container::size\_type priority.queue::s1ze() const ;

const Container::value\_type& priority\_queue::top() con,t ;

Container::value\_type& priority\_queue: :top(); void priority\_queue::push(

con>t Container::value\_type& x);

void priority\_queue::pop();

map::value\_type // pair(c0

""

lKey,'JI')

cons•Compare& cmp=Compare());

Container::siziLtypce queue::size() con";

*No* compa.rision operators.

Yotam Medini © 1998-2007

\*[http://www.medini.org](http://www.medini.org/) /stl/stl.html

181 yotam.medini©gma il.com

*,.J* ...\_ .,\_, \olUlCK i-1..ererence - Version 1.29 !Ml April 20, 2007 3

1. Algorithms

#include <algorithm>

STL algorithms use iterator type parameters. Their *names* suggest their category (Sec 6.1).

*II r, bi-pointing to first binPred-mismatch* pair(Ilnput!teratorl, Ilnputlterator2) 1nismatch(Ilnputiteratorl *ilr:;tl ,*

Ilnputiteratorl Ja.st1, [nputiterator2 flrst2,

i[)) utputlterator *II \;f sf* E *Sk*

transform(llnputlteratorl

[nputlteratorl linputiterator2 I[)) utputlterator

*r;* = bop(s}, s;)

firstl, *lastl .* first*2,* result,

lForwardlterator /,/ *as above but using pred*

remove.:if (lForwardlterator *first:,*

IB'orwardlterator *last,*

lP'redicate pred);

i[)) utputlterator *II "' past last copied*

For abbreviation, the clause -

|  |  |  |  |
| --- | --- | --- | --- |
| Itemplate (class IFoo, . . .) j is dropped. The  outlined leading character can suggest the template context.  Note: When looking at *tv\ro* sequences: S1 = [first1, *last1 )* and *S2* = *[first:2 ,* ?) or S2 = [?, *lastz )* - caller is responsible that function will not overflow *S2.*  4.1 Query Algorithms  !Function *II f not changing [ first, last)*  for\_each([nputlterator first,  [nputiterator *last,*  lFunctin *f);* I&' *7.4* [nputlterator *II first i so i==last or \*i==val* find([nputlterator *first,* | boo!  equal(linputlteratorl *firstl ,* [nputiteratorl *lastl ,* [nputiterator2 *first2 );*  bool  equal(linputlteratorl first I, linputlteratorl *lastl ,* Ilnputlterator2 *first2,*  lBl inaryPredicate hinPred);  *II [ firstz , lastz) [ first1 ,* last1)  !Forwardlteratorl ·  search(IForwarditerator 1 *first 1,* lForwardlteratorl *lastl ,* lForwarditerator2 first2,  lForwarditerator2 *last2 );*  *II [ firstz , last2 ) binPred [ first1, last1 )* | void replac'e(IForwarditerator *first:,*  IForwardlterator *last,*  **const 1['&** *oldVal,*  **const 1f** & nenrVaJ);  void  replace\_if (IForwarditerator *fi rst,*  IForwarditerator *last.*  lP'redicate& *pred.*  **const J'*,S.;;*** newVa.J);  i[))utputiterator *II* "' *result2* + *#[ first, last)*  replace\_copy([nputlterator first,  llnputlterator *last,*  I[)) utputiterator *result,*  const '['& *old Val,*  const '['& new'\.·a.J); | [nputiterator *last,* ·  (Ql utputlterator *result,*  i[)) utputiterator *II as above but using pred*  linputiterator *last,*  lP'redicate pred); remove *conseci,tiw* (binPrcd-) duplicates. T IForwarditerator *1,1 ["',last) gets repetitions*  IForwarditerator *la.. t);*  IForwarditerator *J,I as above but using binPr* |
| [nputiterator *last,*  const '[' val); *n.i'7.2*  [nputiterator *II first i so i==!ast or* pred *(i )*  find\_if ([nputiterator first,  [nputiterator *last,*  lP'redicate pred); 11,/f' 7. 7 IForwarditerator *II first duplicate* adjacent..find(IForwarditerator *first,*  lForwarditeratc:,r *last);* | IForwarditeratorl search(IForwardlteratorl *first.I ,*  IForwarditeratorl *last 1,* IForwardlterator2 *iirst2,* lForwardlterator2 *last2,*  lBl inaryPredicate· *binPred );*  4.2 Mutating Algorithms  I[)) utputlterator *II* "' *first2* + (/ast1 - *first1 )*  copy(Ilnputlterator first], | i[))utputiterator *II as above but using pred*  replace\_copy \_if (Ilnputlterator first:,  Ilnputiterator *la.st,*  i[))utputlterator *result,*  .lP'redicate& pred,  '['& newVa./);  void fill(lForwardlterator *firsi ,*  lForwarditerator *last,*  **1'& *va.lue ):*** | lForwarditerator last,  Ill\ inaryPredicate binPrcd); I[)) utputlterator *II* "' *past last copied* unique\_copy(linputiterator *first,*  linputlterator *last,*  I[)) utputlterator result,  cons< '['& *rosul* t) ;  i[)) utputlterator *II as above but using binPr*  unique\_copy([nputiterator first,  Ilnputlterator la.st, |

JIB inaryPredicate binPred):

lBl inaryOperation

bop);

remove\_copy(llnputlterator first,

const '['& vaJue);

remove\_copy\_if (linputiterator *firs·l,*

I[)) utputiterator *result,*

All variants of unique template functions useful! after sort (See 4.3), .

unique(IB'orwarditerator *first,*

unique(IForwarditerator *first,*

hus

*ed*

IForwarditerator *II first binPred-duplicate*

adjacent\_find (IForwarditerator *first,*

Ilnputiterator *lastl ,*

*ed*

I[)) utputiterator *result,*

IForwarditerator *last,*

lBlinaryPredicate binPred);

i[)) utputiterator *first2).;*

*II* "' *last2* - *(Jast1* - *first1 )*

void fi.11..n(lForwardlterator first,

Size n,

**canst 'JI'& *value);***

void

lBl inaryPredicate binPred);

void *II n* = *# equal val*

count(IForwarditerator first;, IF'orwarditerator *last,* **canst ']f *val)***

Size& n);

void *II n* = *# satisfying pred*

lBl idirectiona1Iterator2 copy\_backward(

1. Bl idirectionaliteratorl JIB idirectionaliterator 1 JIB idirectionallterator2

void swap('['& x, '['& y);

*fiwtl , lastl , last2 );*

void */I by calling gen()*

generate(lForwardlterator *fin;t.,*

IForwarditerator last,

<G enerator gen);

void *I/ n calls to gen()*

reverse(llll idirectionallterator first,

JIB idirectionaliterator *last);*

(()J utputiterator *II "' past last copied*

reverse\_copy (JIB idirectionallterator

* + JIB idirectionaliterator I[)) utputlterator

*firsi. last,* result):

counLif (IForwarditerator *first,*

lForwardlterator *last,*

1Forwardlterator2 *II "' firstz* + *#[ first1 ,last1 )*

swap\_ranges(IForwardlteratorl *firstl ,*

generate\_n(IForwarditerator

Size

*first,*

*11,*

void *II with first moved to middle*

rotate(JForward!terator .first,

lP'redicate pred,

Size& 11);

*II* "' *bi-pointing to first ! =*

·pair([nputiteratorl, [nputiterator2) n1ismatch([nputlteratorl *ffrst.1 ,*

llnputlteratorl *lastl ,*

[nputiterator2 *fir1:1t2);*

lForwardlteratorl *lastl ,*

lForwarditerator2 *[irsi-2 );* · i[))utputlterator *II* "' *result+ (Jast1* - *first1 )* transforn1(linputiterator *first,*

linputiterator *last,*

I[)) utputlterator *result,*

\UnaryOperation *op); «.r7.6*

<Generator gen);

All variants of remove ctnd unique return it.erator t-o *new end* or *past la.st copied.* lForwardlterator *I/ {"',last) is all value* ren1ove(IForwarditerator *firsi,*

lForwarditerator *last,*

const '['& value):

IForwarditerator middle,

lForwardlterator *last);* ·

i[)) utputlterator *II first to middle* position rotate\_copy(IForwarditerator first,

IForwarditerator *middle,*

lForwardlterator last,

i[)) utputlterator result);

Y · ·m Medini © 1998-2007

$ http ://www .medini .org/stl/stl.ht

W yotam .medint0il. com

\.'I .

\*.\_j*\

- l

. - ' /

' *\_) )*



4 *(* >", ,

\ *1,,,-* .....,\ **STL** Q . l R *c*

'u "; ! 1 UlC ( eierence

·

- Vers10n

1.29

[M)

.-,,

April ( \',7

void ' .,• 'c""'' ,.\_

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Tr - tit \_ctopy ,,\_ lower\_bound and upper\_bound return. includes(llnputlteratorl *firstl*  impu era or *w'St* '  llnputlterator *last,'* pair(lForwarditerator,lForwarditerator) Ilnputlteratorl *ll ,* | | | | |
| lRandomAccessiterator | *resultFirst,* | equaLrange(lForwarditerator :first, | nputlterator2 | *first2,* |

ranclom\_shuffie( lRandomAccessiterator *first*

,

lRandornAccessiterator partial sort (

equal..range returns iterators pair that boo! // S1 2 *S2*

lRandomAccessiterator result);

void // *rand() returns* double *in* [O, 1)

random\_shuffle( · lRandomAccess!terator resultLast, lForwarditerator *last,* nputiteratorZ *last2);*

boo! // *as above but using comp*

lRandomAccessiterator *first ,* Compare comp); .']['& value);

includes(llnputiteratorl *firstl ,*

lJRR andomAccessiterator *last*,

Let *n* = *pos1*·t·*10n* - *ii*rst, nth\_e eme· nt

pair (JForwarditerator,lForwarditerator)

llnptitlteratorl Jastl,

andomGenerator rand); partitions [ii.z:s, *last)* into: *L* = *[first, position),*

e,,,, *R* = *fpos1t1011* + 1, *last)* such that

equal\_range(JForwarditerator *first,*

JForwardlterator *last,*

llnputlterator2 lirst2, Hnputlterator2 *last2,*

JIB idirectionallterator // *begin with* true

partition(JIB idirectionallterator *first,*

JIB idirectionallterator *last,*

IP'redicate pred);

JIB idirectionaliterator // *begin with* true

*'<fl* E *L,'<Ir* E *R l 'j,*en *S r.*

void nth\_element(

lRandomAccessiterator *first,*

lRandomAccesslterator *position,*

7.5

con,t ']['& *value,*

Compare comp);

Compare *comp);*

Outputlterator // S1 U *S2, .r\past end*

seLunion(llnputlteratorl *firstl ,*

llnputlteratorl *lastl ,*

stable\_partition(

JIB idir.ectionaliterator JIB idirectionaliterator

*ffrst;, last,*

!RandomAccesslterator *la,;t);*

void // *as above but using comp( ei ,* ej) nth\_element(

4.3.2 Merge

Assuming S1 = [:6rst1, *last1)* and

*Sz* = *[firstz ,* Jast2) are sorted, stably merge them

llnputlterator2 *flrst2,* llnputlterator2 *last2,* Output Iterator *result);*

lP'redicate

*pred );*

lRandomAccessiterator

!RandomAccesslterator

*first,*

position,

into *[result, result + N)* where *N* = IS1I + ISzl.

((}utputlterator

Outputiterator // *as above but using comp*

seLunion(llnputlteratorl *firstl ,*

4.3 Sort and Application

lRandomAccessiterator Compare

*last,*

comp);

merge(llnputiteratorl

l!nput!teratorl

*first ], lastl ,*

linputIeratorl *last],*

llnputiterator2 flrst2,

void sort(lRandomAccessiterator

lRandomAccessiterator

void sort(lRandomAccessiterator

lRandomAccessiterator

*first,*

*last);*

*first , last,*

4.3.1 Binary Search boo!

binary..search(JForwarditerator *first,*

llnput!terator2 llnputlterator2 ((}utputiterator

((}utputiterator

*first2, last2,* result);

llnputlterator2 *last2,*

Outputiterator *result,*

Compare comp);

Outputlterator // S1 n *S2, .r,past end*

!Gi"7.3 Compare

void

comp);

lForwardlterator *last,*

canst 'll'& *value);*

merge(llnputlteratorl *nrstl ,*

llnputiteratorl *lastl ,*

seLintersection(llnputiterator 1 *:firstl ,*

llnputlteratorl *lastl,*

stable\_sort(lRandomAccessiterator *first,*

lRandomAccessiterator *last);*

void

stable\_sort(lRandomAccesslterator *first,*

lRandomAccessiterator *last,*

Compare comp);

void // *{first,mid. dle} sorted.*

partial..sort( // *[middle.fast) eq-greater* lRandomAccessiterator *first,* lRandomAccessiterator middle,

boo!

binary..search(JForwarditerator

lForwardlterator

**const 'Jr&**

Compare

JForwardlterator

lower\_bound(lForwarditerator

JForwarditerator

**eonst 'Jr&**

Forwa.rdltera.tor

*first, last, value,* comp);

first, *last,* value);

Hnputiterator2 *first2,* ITnputlterator2 *last2,* Outputlterator result, Compare comp);

void // *ranges {first,middle) [middle,last)*

inplace..merge( // *into {first,/ast)* JIB idirectionallterator *first,* JIBidirectionallterator *middle,* JIB idirectionallterator *last);*

void // *as above but using comp*

l!nputiterator2 *Jirst2,*

llnputlterator2 *last2,*

Outputiterator *result);*

Output!terator // *as above but using comp*

seLintersection(llnputlteratorl *flrstl ,*

llnputiteratorl *lastl ,* llnputiterator2 *first2,* llnputlterator2 *Jast2,* ((}utputiterator *result,* Compare comp);

lRandomAccesslterator *last);*

void // as *above but using comp( ei ,* ej)

lower\_bound(JForwarditerator *first,*

IForwardlterator *last,*

inplace\_merge(

JIB idirectionallterator

*first,*

Outputiterator // S1 \ *S2, .r\past end*

seLdifference(Ilnputiterator 1 *first ] ,*

partiaLsort (

lRandomAccessiterator *first,*

lRandomAccessiterator *middle,*

IForwardlterator

= 1'&

Compare

value, comp);

JIBidirectionallterator JIBidirectionaliterator Compare

*middle, last, comp);*

llnput!terator 1 *lastl ,* llnput!terator2 *first2,* ITnputiterator2 *last2,*

lRandomAccessiterator *last,*

Compare comp);

lRandomAccessiterator // *post last sorted*

partial..sorLcopy(

upper\_bound(lForwarditerator

lForwardlterator

conn ']['&

Forwa.rditerator

*first, last,* value);

4.3.3 Functions on Sets

Can work on *sorted associcative* containers (see 2.7). For multiset the interpretation of -

Outputiterator result);

Outputlterator // *as above but using comp*

set\_difference(llnputlteratorl *firstl ,*

llnput!teratorl *lastl ,*

l!nputiterator

l!nputiterator

*first,*

*last,*

upper \_bound(lForward!terator *fiwt,*

lForwarditerator *last,*

*union, intersection* and *d:ifference* is by:

*maximum, minimum* and *substraction* of

llnputiterator2 *first2,*

llnputiterator2 *last2,*

lRandomAccessiterator lRandomAccessiterator

*resultFirst, resultLast);*

conrt ']['&

Compare

value, comp);

occun-ences respectably.

Let *S;* = (first;, *last-;)* for ·i = 1, 2.

((} utputiterator *result,*

Compare *comp);*

Yotam Medini © 1998-2007

[\*http:](http://www.medini.org/stl/stl.html)/[/www.medini.org/stl/stl.html](http://www.medini.org/stl/stl.html)

18l yotam.medini©gmail.com

o .1..1.., \.lUICK 1,.eterence - Version 1.29 IMl April 20, 2007 5

IO> utputiterator *II S1!!.S2, past end*

set\_synunetric\_difference(

. ITnputiteratorl flrst1,

][nputlteratorl *lastl ,*

4.3.5 Min and Max

const 'Jl'& nlin(<•nst 'Jl'& *xO,* gig 'll'& *xl* );

4.3.7 Lexicographic Order

bool lexicographicaLcompare(

IO> utput!terator *II r k* = *s1,.* s1.,-1 *for k >* 0 adjacent\_difference( *II ro* = *so*

ITnputiteratoi:' flrsJ:,

][nputlterator last,

][nputlterator2 *fi.rst2,* linputiterator2 *last2,* iO) utputiterator *result);*

con,t ']['& min(con,t ']['&

const 1['&

Compare

xO,

*xl ,*

comp);

][nputiterator 1 *first* 1,

Ilnputlteratorl *lastl ,* linput!terator2 flr:,t2, linputlterator2 *last2 ):*

*I[))* utputlte;rator result);

IO> utput!terator *If as above but using binop*

adjacent\_differ,ence(

iO) utputlterator *II as above but using comp*

set...symmetric\_difference(

c••" ']['& max( ']['& :XO, const ']['& *xl );*

bool lexicographicaLcompare(

linputiterator first,

. ][nputlteratorl *fastl ,* linputlteratorl *lastl ,* ITnputiterator2 *fi.rst2,*

ITnputiterator2 *last2,*

con,t 'll'& max(const 'll'&

const 1['& Compare

lForward!terator

*xO,*

*xl , comp);*

ITnputiteratorl *fii t.1,*

linput!teratorl *lastl ,*

][nputiterator2 *Brst2,*

ITnputiterator2 *last2,*

ITnputiterator *last,*

iO) utput!terator *result,*

JIB inaryOperation bin *op);*

IO> utputiterator *result,*

Compare *comp);*

4.3.4 Heap

void *II (f ast* - 1) *is pushed*

push\_heap(lR.andomAccesslterator

*first,*

n1in\_ele1nent(]Forward!terator

lForward!terator

lForwarditerator n1in\_element(]Forwarditerator

JForwarditerator

Compare

*first,*

*last);*

tirst,

*last,*

*comp);*

Compare *comp);*

4.4 Computational

#include <numeric>

'll' *II L[ first,/ast)* 1&7.6

1. Function Objects

#include <functional>

lR.andomAccesslterator void *II as ab\_ove but using comp* pushJieap(lR.andomAccessiteratcir

lR.andomAccessiterator

*last);*

first,

*last,*

lForwarditerator 1nax\_elem.ent(lForwarditerator

lForwarditerator

*fi.i'St,*

*last);*

acci.lmulate(ITnputiterator

linputiterator

']['

'][' *II as above but using binop*

*fil'st, last, initVa.I);*

Derived unary objects:

t.emplate(class A rg, class lR.esult) struct unary\_function {

typeclcf.Arg argument\_type; typedef ]Result resulLtype;}

Compare void *II first is popped*

pop J1eap(lR.andomAccessit erator

lR.andomAccess!terator void *II as above but using comp* pop-11eap(lR.andomAccesslterator

lR.andomAccesslterator

*comp);*

fi.rst,

*last);*

*fi.rst,*

*last,*

lForwarditerator

mrucelement (JForwarditerator

JForwarditerator Compare

4.3.6 Permutations

*fi.l'st,*

*last, .*

comp);

templat.e(class Argl, class Arg2, class !Result)

struct binary \_:function {

typedef Argl firsLargumenLtype; typedef Arg2 seco'nd\_argumenLtype; typedef lR.esult resulLtype;l

accumulate(linput!terator *first,*

linputlterator *last,*

'][' *i11itVal,*

JIB inaryOperation binop);

'Jl' *II L;* e} x er *for* ef. E *Sk,( k* = 1,2) inner\_prod uct(linput!teratorl *fi.rstl ,*

linputlteraforl *lastl ,*

struct. negate(']['); struct logicaLnot('ll');

o.-r 7.6

Compare

comp);

To get all permutations, start with ci.scending sequence encl with descending.

linputiterator2 *flrst2,*

void *II {first.last) arbitrary ordered*

boo! *II*

*..r,,*

*iff available*

'll' init111,I);

makeJieap(lR.andomAccessiterator

lR.andomAccessiterator void *II as above but using comp* n1akeJieap(lR.andomAccesslterator

lR.andomAccessiterator

Compare

first,

*last);*

*lirsl , last , comp);*

nexLpermutation(

JIB idirectionallter:ator *first,*

1lll idirectionaliterator *last);*

boo] *II as above but using comp*

11ext\_pern1utatio11(

'][' *II Similar, using "2:,(sum) and* X *mult*

inner\_prod i.lct(linputlteratorl *first.I ,*

][nputlteratorl *lastl ,*

linputlterator2 firsf:2,

'][' initVa.J.

JIB inaryOperation sum,

Following clerivocl templat,e objcct.s accept two operands. Result obvious by the name.

st.n1ct. plus(1f');

struct minus('][');

void *II sort the [first.last) heap*

sort J1eap(lR.andomAccesslt erat or

*first,*

JIB idirectionallterator

JR idirectionaliterator Compare

first:,

last, comp);

JIBinaryOperation mult);

IO> utputitrator . *II rk* = *"2:,fir +k* e;

st,ruct multiplies([); struct divides(1f'i;

lR.andomAccessiterator

*last);*

boo] *II*

*r,,*

*iff available*

***1=11rst***

partiaLsum(ITnputlterator fi.r:<l,

stn1ct modulus('1r');

struct equaLto('l[');

void *II as above but using comp*

prev

\_permutation(

ITnputiterator *last,*

struc:t. noLequaLto(1f');

sortJieap(lR.andomAccessiterator

lR.andomAccessiterator

<Compare

*first, last , comp);*

JIB idirectional!terator *fil'st,*

lB\ idirection.al!terator *last):*

boo! *II as above but using comp*

IO> utputiterator result):'

IO> utput!terator *II as above but using binop*

partiaLsum(

struct greater('}(');

struct. less('ll');

struct greater\_equal ('ll');

y, · '11Meclini © 1998-2007

prev\_permutation(

JIB idirectionaliterat or JIB idirectionaliterator Compare

first,, *last,* comp);

][nput!terator

linputiterator

*I[))* utput!terator JIB inaryOperation

first,

*last, result,* binop):

struct less\_equal('l['); strnct logicaLand ('Jr);

stn1ct logicaLor (1f');

yotam .medirly\_\_')il.com

*\: )*

\.'=".)

6

\*http:l/[www.medini.org1stl](http://www.medini.org1stl/) /stl.htrr

*j* l



5.1 -Ju:t

,c- , . . \_,\_

*) )* , STL Qmck Reference - Vers1011 1.29 fMJ April(J

nction Adaptors j j template(class ((J)peration) j j In table follows requirements check list for / 6.2 Stream Iterators

class binder2nd: public

5.1.l Negators

template(class lP'redicate)

class unary\_negate : public

unary..function (IP'redicate::argument\_type, boo!);

unary\_negate::unary \_negate( lP'redicate pred);

boo! *II negate pred*

unary...negate::operator()( lP'redicate::argument\_type x);

unary\_negate(lP'redicate) notl(con,t lP'redicate *pred );*

template(class IP'redicate)

class binary\_negate : public binary\_function (

lP'redicate: :first\_argumenLtype, lP'redicate: :second\_argumenLtype); boo!);

unary\_function(

((J)peration::first ..argumenLtype, OJ peration: :result\_type);

binder2nd::binder2nd(

con>t OJperation& *op*

'°"" ((J)peration::seconcLargument\_typc y)

*II* argument-type *from* unaryJunction ((J) peration::result\_type

binder2nd::operator() (

con,, binder2nd::argument\_type x);

binder2nd(((J)peration)

bind2nd( ((J)peration& op, ']['& x);

IE!i' 7.7.

5.1.3 · Pointers to Functions

template(class Arg, class !Result)

class pointer\_to\_unary..function : public unary\_function(Ar[,\_esult);

pointer\_to\_unary ..function(Arg, Result) ptr..fun(lR.esult(\*x)(.Arg));

templa.te(class 11',

class ][)) jstance=ptrdi1Lt) class istream\_iterator :

pubic iterator(inputJtera.tor\_tag, 11', ll}) jstance};

*II end of stream* llili'7.4

istream\_iterator: :istream\_iterator();

istream\_iterator: :istream...iterator ( istream& *s);* 1Eii'7.4

istrea.mJtera.tor::istream\_iterator (

.istream..iterator('f , [J) jstance)&); istreamJterator::-istream\_iterator ();

con,t 11'& istream\_itera.tor::operator\*() con>t ;

istream\_itera.tor& *II Read and store* 'f *value*

istreamJterator::operator+ +() cons,;

boo! *II*

Expression; Requirements

!

O

F

X O

X u

might be singular

•

X(a)

:::::,X(a) == a

•

•

\*a=t # \*X(a)=t

•

X u(a)

X u=a

:::::, u == a

•

.•

u copy of a

a==b

equivalence relation

•

•

a!=b

#!(a==b)

•

•

r = a

:> r == a

•

\*a

convertible to T.

a==b # \*a==\*b

•

•

\*a=t

(for *forward,* if X mutable)

•

++r

•

•

•

••

•

r++

convert.ible to X&;

#{X x=r;++r;return x;}

•

•

•

\*++r

**\*r++**

convertible to T

•

•

•

result is dereferenceablc or

*past-the-end.* &:r == &:++r

convertible to const X& convertible to X&; r==s# ++r==++s

•

binary...negate::binary\_negate( lP'redicate *pred );*

boo] *II negate pred*

binary...negate::operator()( IP'redicate::first\_argument\_type *x* lP'redicate: :second\_argumenLtype y);

binary\_negate (lP'redicate) not2(<0nn lP'redicate pred);

5.1.2 Binders

template(class OJ peration) class binderlst: public

unary..function(

OJ peration: :second\_argument.\_type, OJ peration::result\_type};

binderlst::hinderlst(

**c:onst l(Dperation& op)**

con" ((J)peration::first\_argumenLtype y);

*II* argument\_type *from* unary\_function ((J) peration: :result\_type binderlst::operator() (

con,t binderlst::argument\_type x);

binderlst(OJ peration)

bindlst(con,t ((J)peration& *op,* con,t 'f & x);

templa.te<class Argl, class Arg2, class !Result>

class pointer \_to\_binary \_function :

public bina.ry..function(Argl, .Arg2,

!Result);

pointer\_to\_binary ..function (Argl, Arg2,

!Result) ptr\_fun(lResult(\*x)(.Argl, Arg2));

1. Iterators

#include <iterator>

6.1 Iterators Categories

Here, we will use:

X iterator type. a, b iterator values.

r iterator reference (X&; r).

t a value type T.

Imposed by empty struct ta.gs.

6.1.1 Input, Output, Forward

struct input-1terator\_tag {}llili' 7.8 struct output\_iterator \_tag {} struct forward.iterator \_tag

n

*all end-of-streams are equal*

operator= =ccon,t istream.itera.tor,

canst istream.itera.tor);

template(class 11')

class ostream\_iterator :

public iterator(out:i:mt.iterator\_ta.g, void, . . . );

*II I f delim -f* O *add a fter each write*

ostream\_iterator: :ostream...iterator( ostream& *s,*

c,n,, char\* *delim=O);*

ostream\_iterator::ostream\_iterator (

,on,t ostream..iterator s);

ostream\_itera.tor& *II Assign* & *write (\*o=t)*

ostreain\_iterator::operator\*O ;

ostrea.111-itera.tor& ostream\_itera.tor::operator=(

con,t ostream.iterator *s );*

ch

ostream\_it,erator& *II No-op*

ostream.iterator: :operator++();

ostream\_iterator& */I No-op*

ostream..iterator::operator+ +(int);

IE!i' 7.4.

Yotam Medini © 1998-2007

$h[ttp;//www.medini.org](http://www.medini.org/) /stl/stl.html

181 yotam.medini©gm ail.com

""'.v•• •'·'-'•'-'LvU\,v - V tlti.lUll l..:l'::J [A1J April 20, 2007 7

6.3 Typedefa & Adaptors

* 1. mplate(C<!,tegory, 1',

JD) istance=ptrdifLt,

Pointer= 1'\*, lReference= 1'&) class iterator {

* + - !Category iterator\_category;

1[' value\_type;

lD> ista;,,ce difference\_type; lP'ointer pointer; lReference reference;}

6.3.1 Traits template([)

class iterator\_traits { [::iterator\_category

iterator\_category;

Denote

RI = reverse\_iterator

All = lRandomAccessiterator.

Abbreviate:

typedef Rl<All, 1',

lReference, lD> istance) self;

*II Default constructor => singular value*

self::RI();

explicit *II Adaptor Constructor*

self::Rl(Alli);

All self::base(); *II adpatee's position*

*II so that: &\*( Rl(i ) )* == *&\*(i-1)* !Reference self::operator\*();

self *II position to* & *return basc()-1*

Rl::operator+ +();

6.3.3 Insert Iterators

templat !(class Container)

class back\_insert\_iterator : public out,puLitcrat.or;

templatc(class Container}

class fronLinserLiterator : public output\_itcrator;

template( class Container) class insert\_iterator :

public output..it.erat.or;

Herc '][' will denote the Container::value\_type.

Constructors

explicit // 3 Container::push\_back(<onrt 1['*&)*

back..inscrt..i terator::back\_inscrt.\_itera.tor(

**7** Examples

**7.1** Vector

*II* safe get

int vi(const vector<unsigned>& v, int i)

{return(i < (int)v.sizeO ? (int)v[i) : -1);}

*I!* safe set

**void vin(vector<int>& v, unsigned i , int n)** {

int nAdd = i - v,size O + i;

if (nAdd>O) v.insert(v.end(). nAdd, n); else v[i) = n;

}

7.2 List Splice

void lShow(ostreamk as, canst list<int>& 1) { **ostrE?am\_iterator<int> osi (os,** 0 ' ) ; copy(l.begin (), l.end(), osi); os<<endl;}

1

**void lmShow(ostream& os, canst char\* msg,**

[::value\_type

l:difference-type

value\_type; differern;:e\_type;

self& *II return old position and move*

Container& x);

**const list<int>& 1,**

canst list<int>& m) {

[::pointer

pointer;

Rl::operator++(int.); *II to* base()-1

explicit */I* 3 Container::pusJ1-fro11t(const ']['&)

front..inscrt\_itcrator::front\_inscrt.\_itcra.tor(

**os << msg << (m . size () ?**

lShow(os, l);

**0 :\n**

**11** 11 11

; ) ;

Lreforence reference;}

Pointer specilaizations: i& 7.8

self *I/ position to* & *return base()+l*

Rl::operator- -();

Container& x);

*II* 3 <Container::i11sert(,<0

""

1['*&)*

if (m.size( ))lShow(os, m); }*II* lmShow

list<int>: :iterator p(list<int>& l, int val)

self& *II return old position and move*

template(1')

class iterator\_traits(1'\*) { rando1n\_access iterator\_tag

iterator\_category

1['

ptrclifLt 1'\*

']['&

value\_type; difference\_type; pointer;

reference;}

Rl::operatoi:--(int); *II to base()+l*

bool *II* <=> *sO.base()* == *sl.base()*

operator==(<onst self& *sO,* const self& *sl );*

reverse\_iterator Specific

self *II returned value positioned at base()-11*

reverse.\_iterator: :operator+(

**Il))istance n)** ;

template(1')

class iterator\_traits (<0"" 1'\*) { random\_access\_itera.tor\_ta.g

iterator\_category ;

1' value\_type;

= 1r\* pointer;

ptrdHLt di.fference\_type;

= 1!'& reference;}

self& *II change* & *return position to base()-11*

reverse..itcrator::operator+=(lD> istance *11);*

self *II returned value positioned at* base()+n reversc..iterator::operator-(

JD) istance 11) 5.2!'11 ;

insert..iterator::insert...iterat.or(

Container *x,*

Container::itcrator i);

Denote ·

l nslter = back\_insert \_iterator insFunc = push\_back

iterMa ker = back\_inserter 1&7.4

or

l nslter = fronLinsert\_iterator insFu nc = push\_front

iterMa ker = front\_inserter

or

l nslter = insert\_iterator

insFunc = insert

Mi=mber Functions & Operators l nslter& *II calls x.insFunc(val)*

{return find(l.begin (), l.end(), val);}

static int prim[]= {2, 3, 5, 7};

static int pert[]= {6. 28, 496};

**canst list<int> 1Primes(prim+O, prim+4) ;** canst list<int> lPerfects (perf+O, perf+3) ; list<int> l(lPrimes) , m(lPerfects) ;

**lmShow- (cout, "primes & perf ects 11 , 1. m) ;**

l.splice(l.begin(), m);

**lrnShow (cout , 11 splice(l .beg, m) " , 1, m) ;**

**l = lPrimes; m = lPerfects;**

l.splice (l.begin (), m, p(m, 28));

**lmShow(cout, "splice(l .beg, m , -2s) 0 , 1, m) i**

m.erase(m.begin (), m.end()); *II* <=>m.clear( )

**1 = lPrimes;**

l.splice(p(l, 3), l, p(l, 5));

**lmShow(cout , "5 before 311 , 1, m) ;**

1 = lPrimes;

l.splice(l.begin (), l, p(l, 7), l.end());

self& */I change* & *return position to base()+11*

l nslter::operator=:(< 00'' 1!'& *va.l );*

l nslter& *I/ return \*this*

**lmShow (cout , "tail to head**

**= lPrimes;**

**11 ,**

**l , m) ;**

6.3.2 Reverse Iterator Transform [-i./,'j) ,\_, Li - 10,,-i - 1).

rcvcrse\_iterator: :operator-=(lD> ist ance 11);

lReference *II \* (\*this* + n)

reverse\_iterator: :operator[](lD>istance n);

lD> istance // *rO.baso()* - *rl.base()*

template(llter)

class reverse\_iterator : public iterator( iterator\_traits(llter): :iterator\_category, iterator\_traits (llter)::value-type,

iterator\_traits (l!1:er)::difference-type,

iterator\_traits(llter): :pointcr, iterator\_t.raits {llter)::reference);

l nslter::operator\*();

lnsli:er& *II noaop, just return \*this*

l nslter::operator+ +();

l nslter& *II no-op, just return \*this*

l nslter::operator++(int);

Template Function

**1**

l.splice(l.end(), 1, l.begin (), p(l, 3));

**lmShow (cout. "head to tail 11 , l , m) ;**

,;511111..

primes & perfects:

2 3 5 7

6 28 496

" self& *rO,* self& *rl );*

operator(<00

self *II 11 + r.basc()*

operator-(lD>istance *11,* canst self& r);

boo! // *rO.basc()* < *rl.basc()*

operator<(< "''self& rO, self& *rl );*

0

lnslter // *return* /ns/ter(Container) *(x)*

iterMa ker(IContainer& x);

*II return* i11serUterator(Container) *(x. i)* i:nscrt,..itcra.t.or(Container) inserter(Container& *x,* llterator i):

splice(l.beg, m): 6 28 496 2 3 5 7

splice(l.beg, m, -2a):

28 2 3 5 7

6 496

5 before 3: 2 5 3 7

tail to head: 7 2 3 5

head to ·tail: 3 5 7 2

*t·*.'ll Medini © 1998-2007

*\ !*

......... ....

\*<http://www.medini.org/>stl/stl.ht

·'--... ..,,.

161 yotam.medin y..Jil.com

' *J j*



8 *r* . \_- **STL** Quick Reference - Version 1.29 lMJ April

o

-- ;- -- t,- -/

7.3 Compare Object So.rt I c-o.0os,-o.500J I **7.7** Iterator **and Binder** I 7.8 Iterator **Traits**

(0.309,-0.951)

class ModN {

public:

ModN(unsigned m) : \_m(m) {}

bool operator () (const unsigned& uO,

const unsigned& ul)

{return ( (uO % \_m) < (ul % \_m) ) ;}

**private: unsigned \_m;**

}; *II* ModN

**ostrean;i\_i terator<unsigned> oi (cout .** 11 ") ;

unsigned q[6];

for (int n=6, i=-1; i>=O; n=i--) q [i] = n\*n\*n\*n;

**cout<<"four-powers:** 11 ;

copy(q + 0, q + 6, oi) ;

f or (unsigned b=lO; b<=lOOO; b \*• 10) {

**vector<unsigned> sq(q + 0, q + 6) ;** sort (sq.beginO , sq.end() , ModN (b) ) ; **cout<<endl<< 11 sort rood "<<set"W(4)<<b<< 11** copy (sq.begin() , sq.end() , oi) ;

} **cout << endl;**

|  |  |
| --- | --- |
| @1111  **f our-poto1ers:** | 16 81 256 625 1296 |
| sort mod 10: | 81 625 16 256 1296 |
| sort mod 100: | 16 625 256 81 1296 |
| sort mod 1000: | 16 Bl 256 1296 625 |

7.4 Stream Iterators

void unitRoots(int n) {

**cout << 11 unit** 11 << **n << 11 -roots: 11 << endl i**

**vector<complex<float> > roots;**

float arg = 2.\*M\_Pil(float)n;

complex<f loat> r , rl = polar( (f loat) l . , arg) ;

**for Cr = rl; --n; r \*= rl)**

7.5 Binary Search

*II* first 5 Fibonacci

static int f b5[J = {1, 1, 2, 3, 5}; f or (int n = O; n <= 6; ++n) {

**pair<int\*, int\*> p** =

equal\_range (f b5, f b5+5, n) ;

**cout<< n** <<11 :("<< **p.f irst-fbS** <<','

« p.second-f b5 «")

if (n==3 l I n=6) cout « endl;

}

@Ill

0:[0,0) 1:(0,2) 2:(2,3) 3:(3,4)

4: [4,4) 5: [4,5) 6: [5,5)

7.6 Transform & Numeric

**template <class T>**

class AbsPvr : public unary\_f unction<T, T> { public:

AbsPirr(T p) : \_p(p) {}

T operator() ( const *Tit* x) const

{ return poy(f abs(x) , \_p) ; }

**private: T \_p;**

}; *II* AbsPYr

template<typename Inpiter> float normNP(Inpiter xb, Inpiter xe, float pl {

**vector<float> vf ;**

**transform(xb, xe, back\_inserter(vf) ,**

AbsPYr<f loat>(p >·O. ? p : 1.)) ; return( (p > 0.)

? pow(accumulate(vf .begin() , vf .end() , O .) , 1.lp)

\*(max\_element (vf .begin() , vf . end() ) ) ) ;

*II* self -re:f ering int

**class Interator : public iterator<input\_iterator\_tag, int, size\_t>** { **int \_n;**

public:

Interator(int n=O) : \_n(n) {}

int operator\*() const {return \_n;}

**Interator& operator++()** {

++**\_n; return •this;** }

Interator operator++(int) { Interator t (\*this) ;

**++\_n; return t;}**

}; *II* Interator

bool operator==(const lnterator& iO,

**canst Interator&c: i1)**

{ **return (\*iO ==- •i1) ;** }

**bool operator!= (const Interator& iO ,**

**const Interatork i1)**

{ return !(iO == ill ; }

**struct Fermat: public binary\_function<int, int , bool>** {

Fermat (int p=2) : n (p) {}

**int n;**

int nPoYer(int t) const { *II* t-n

**int i=n, tn=l;**

1Jhile (i--) tn \*= t; return tn; } *II* nP01Jer

int nRoot (int t) const {

return (int)poY(t +.1, 1.ln) ; } int xNyN(int x, int y) const {

return(nPoyer(x)+nPoYer(y)) ; }

bool operator () (int x, int y) const int zn = xNyN(x, y) , z = nRoot (zn) ; return(= == nPoYer(z) ) ; }

}; *II* Fermat

f or (int n=2; n<=Mp; ++n) {

template <class Itr>

**typename iterator\_ traits<Itr>: :value\_ type**

mid(Itr b, Itr e, input\_iterator\_tag) {

**cout << "mid(general) : \n" i**

Itr bm (bl ; bool next = f alse;

**f or** ( ; **b != e; ++b, next = !next)** {

if (next) { ++bm; }

}

**return •bm;**

} *II* mid<input>

template <class Itr>

**typename iterator\_traits<Itr>: :value\_type**

mid(Itr b, Itr e,

**random\_access\_iterator\_tag)** {

**cout << "mid(randorn) :\n11 ;**

Itr bm = b + (e - b) *12;*

**return \*bm;**

} *II* mid<random>

**template <class Itr>**

typename iterator\_traits<Itr>: :value\_type mid(Itr b, tr el {

typename

**iterator\_traits<Itr>: :iterator\_category t;**

mid(b, e, t) ;

} *II* mid

template <class Ctr>

void f illmid (Ctr& ctr) { static int perf ects[5J

{6, 14, 496, 8128, 33550336},

\*pb = &perf ects[OJ ;

ctr.insert (ctr.end() , pb, pb + 5) ; int m = mid(ctr.beginO , ctr.end() ) ; **cout << "mid= 11 << m << "\n11 ;**

} *II* f illmid

roots.push\_back(r) ;

copy(roots.hegin () , roots.end() ,

**ostream\_iterator<complex<float> >(cout,**

"\n") ) ;

} *II* unitRoots

**{of stTeam. 0( 11primes.txt 11 ) ; o << 11 2 3 511 ;}**

**ifstream pream( 11 primes.txt 11 ) ;**

**vector<int> Pi**

} *II* normNP

float distNP (const float\* x, canst float• y, unsigned n, float p) {

vector<float> dif f ;

**transtorm(x, x + n, y, back\_inserter(diff ) ,**

minus<f],oat> () ) ;

return normNP(diff .begin() , diff .end() , pl ;

} *II* distNP

Fermat fermat (n) ;

**for (int x=l; x<Mx; ++x)** {

binderlst<Fermat>

f x = bindlst (f ermat , x) ;

Interator iy (x) , iyEnd(My) ;

while ((iy = f ind\_i:f (++iy, iyEnd, fx))

!= iyEnd) {

**int y = •iy,**

z = f ermat .nRoot (f ermat.xNyN(x, y) ) ;

list<int> l; f illmid(l) ;

@1111•

mid (general) : mid=496

mid (random) :

mid=496

**vector<int> v;**

f illmid(v) ;

**istrearn\_iterator<int> priter(pream);**

**istream\_iterator<int> eosi;**

copy(priter, eosi, back\_inserter(p) ) ;

f or\_each(p.begin() , p.end() , unitRoots) ;

float x3y4 [) = {3 . , 4. , O .};

float z12 0 = {O., 0., 12.};

float p[] = {1., 2., M\_PI, 0.};

**cout << x** <<

« y «

<< **z** <<

if (n>2)

<< **n** << " + "

<< **n** << II;**U**

<< **n << endl;**

@111111

f or (int i=O; i<4; ++i) {

f loat d = distNP(x3y4, z12, 3, p[i]) ;

**cout << 11Fermat is Qrong** !" << **endl;**

}

**unit 2-roots:**

(-1. 000 ,-0 .000)

**cout <<**

}

**11 d\_{.. << p[i]** << 11 }=11

<< **d << endl j** }

}

**unit 3-roots:**

(-0.500 ,0.866)

(-0.500,-o.866)

**unit 5-roo-cs:**

(0.309,0.951)

(-0.809 ,0.588)

Yotam Medini © 1998-2007

@1111.

d\_{1}=19 d\_{2}=13

d\_{3.14159}=12.1676 d\_{0}=12

(6)1111.

3-2 + 4-2 = 5-2

5-2 + 1r2 = 13-2

5-2 + a-2 = 10-2

r2 + 24-2 = 25-2

\*http://[www.medini.org/](http://www.medini.org/) stl/stl.html

l8i yotam.medini©gmail. com

POLYMORPHISM:

Single Inheritance with "extends" class A( J

class B extends. A{ abstract class. C ( ) class D extends C { ) class E extends D Abstract methods abstract class F {

abstract int bla{);

class G extends F (

int bla() ( //required method return S;

Multiple Inhe:i:·itance of interfaces with "impl.ements" (fiel.ds not inherited) interface H {

void methodl'.. ();

boolean methodB (int arg);

interface I extends H{ void methodC ();

interface K ()

class *J* extends F implements I, K {

int bla o· { return S; ) //required from F

void methodA() () //required from H boolean metbodB( int a) ( //req from A

return l;

void methodC() () //required from I

Type inference:

1. X = new B(); //OK
2. y = new A(); //Not OK
3. z = new C (); *I* /Cannot instantiate abstract

//Method calls care about right hand type

{the instant.iated object)

*I* /Compiler checks depend on left ha11d type

GENERICS:

class MyClass<T> T value;

T getValue() ( return value; J

class ExampleTwo<A,B> A x;

B y;

class ExampleThree<A extends List<B>,B> ( A list;

B head;

)

//Note the extends keyword here applies as **Well to inte-rf aces , so A can be an interface** that extends List<B>

0/

JAVA **COLLECTIONS:**

List<T>: Similar to arrays

ArrayList<T>: Slow insert into middle

//ArrayList has fast random access LinkedList<T>: slow random access

//LinkedList fast as queue/stack Stack: Removes and adds from end

List Usage: boolean add{T e);

void clear(); //empties boolean contains{Object o); T get{int index);

T remove{int index); boolean remove{Object o);

//remove uses comparator

T set{int index, E val); Int size();

List Traversal:

for{int i=Oi<x.size() ;i++)

//use x.get(i);

//Assuming List<T>:

for (T e : x) (

//use e

Queue<T>: Remove end, Insert beginning LinkedList implements Queue

Queue Usage:

T element(); // does not remove boolean offer(T o); //adds

T peek(); //pike element T poll{); //removes

T remove(); //like poll

*Traversal :* for (T e : x) {)

Set<T>: uses Comparable<T> for uniqueness 'TreeSet<T>, items are .sorted ·

HashSet<T>, not sorted, no order LinkedHashSet<T>, ordered by insert Usage like list: add, remove, size *Tri3versal :* for (T e : x) {)

Map<K,V>: Pa.irs where keys are unique HashMap K,V>, no order LinkedHashMap.<K,'V> ordered by insert TreeMap<K,V> so'rted by keys

v get {K key);

Set<K> keySet(); //set of keys V put(K key, V val.ue);

V remove {K key); Int size(};

Collection<V> values(); //all. values

*Traversal : f or-ea ch* ,,,; *keyset/ val ues*

*(*

**java .. util.PriorityQueue<T>**

A queue that is always automatically sorted using the comparable function of an object

public static void main(String[J args) ( Comparator<String> crop= new LenCmp(); PriorityQueue<String> queue =

new PriorityQueue<String>(lO, cmp); queue.add{"short");

queue.add{"very long indeed"); queue.add{"medium") ;

while {queue.size() != 0) System.out.println(queue: remove());

class LenCmp implements Comparator<String> public int compare{String x, String y) (

return x.length() - y.length();

)

**java .. util .. Collections algorithms**

Sort Example:

//Assuming List<T> x

Collections. sort(x); //sorts with comparator Sort Using Comparator:

Collections. sort(x, new Comparator<T>( public int compareTo(T a, T b) (

//calculate which is first

//return -1, O, or 1 for oraer: return someint;

Example.of two dimensional. array sort: public static void main{final String[ ) a) {

final String[)[) data = new String[][] new String[ ] { "20090725", "A" ) ,

new String[] ( "20090726", "B" ),

new String [J ( "20090727", "C" ),

new String[] ( "20090728", "D" ) ) ;

Arrays.sort{data,

new Comparator<String [J > () { public int compare (final String[]

entryl, final String[] entry2) {

final String timel = entryl[O] ; final String time2 = entry2[0] ; return timel.compareTo(time2);

)

) ) ;

for (final String[] s : data) { System.out.println{s[OJ +" "+s[l]);

More col.l.ections static methods: Collections .max( ... ); //returns maximum Collections .min ( ... ); *I* /returns maximum Collections.copy( A, B); //A list into B Collectioris reverse( A ); //if A is list

'-,·\

*('-.•..* '

® "'Kapparate, © 2013 Kapparate Operating

}

*u'·-<\_J*

*) j*

\ \..:\_ j

*Ci*..*.• CJ*



THE JAVA LANGUAGE CHEAT SHEET

Primitive Types:

INTEGER: byte(8bit) ,short(l6bit),int(32bit) ,. long(64bit),DECIM:float(32bit) ,double(64bit)

,OTHER: boolean( lbit), char (Unicode)

IF STATEMENTS:

if( boolean\_value else if( bool ) else if( ..etc ) else

STATEMENTS STATEMENTS STATEMENTS STATEMENTS

c/ 0

CLASS/OBJECT TYPES:

INSTANTIATION:

public class Ball (//only 1 public per file

.,

//STATIC FIELDS/METHODS

private static- int numBalls = 0;

HEX:OxlAF,BINARY:Ob00101, LONG:8888888888888L

**CHAR EXAMPLES:. 'a' , ' \n' , ' \t'** , '\' ' , ' \ \ ' *,'\11 '*

//curly brackets

LOOPS:

optional if one line

public static int getNumBalls( ) ( return numBalls;

Primitive Operators

Assignment Operator: = (ex: int a=5,b=3; ) Binary Operators (two arguments) : + - *\* I %* Unary Operators: + - ++ --

Boolean Not Operator (Unary) : ! Boolean Binary: == != > >= < <= Boolean Binary Only: && I I

Bitwise Operators: - & A -1 << >> >>>

Ternary Operator: bool?valtrue:valfalse;

Casting, Conversion

int X (int)5.5; //works for numeric types int x = Integer.parseint( "l23");

float y = Float.parseFloat( "l.5");

int x = Integer.parseint("7A",16); //froroHex String hex = Integer.toString(99,16);//toHex

*I* /Previous l"ines work w/ binary, other bases

java.util.Scanner, input, output

Scanner sc = new Scanner(System.in);

int i = sc.nextint(); //stops at whitespace String line = sc.nextLine(); //whole line System.out.println( "bla"); //stdout System.err.print( "bla"); //stderr,no newline

java.lang.Number types

Integer x = 5; double y = x.doubleVlue{); double y = (double)x.intValue( };

//Many other methods for Long, Double, etc

java.lang.String Methods

**//Operator +, e.g. "fat"+,.,cat" -> "fatcat"**

boolean equals(String other); int length() ;

char charAt(int i);

String substring(int i, int j); //j not incl boolean contains (String sub);-

boole.an startsWith (String pre); boolean endsWith(String post);

int indexOf(String p); //-1 if not found int indexOf(String p, int il ; //start at i int compareTo(String t); ·

//"a".compareTo("b") -> -1

String replaceAll-(String str, String find); String[) split(String delim);

StringBuffer, StringBuilder

StringBuffer is synchronized StringBuilder (Use StringBuilder unless multithreaded) Use tlie .a\_pend ( xyz ) methods to concat toString{) converts back to String

java.lang.Math

Math. abs (NUM) ,Math.ceil (NUM),Math.floor (NUM)

,Math.log(NUM) ,Math.max(A,B) ,Math.min(C,D), Math.pow (A,B) ,Math.round(A) ,Math.random()

while( bool ) ( STATEMENTS ) for(INIT;BOOL;UPDATE) ( STATEMENTS )

//lINIT 2BOOL 3STATEMENTS 4UPDATE 5->Step2

do( STATEMENTS )while( bool );

//do loops run at least once before checking break; //ends enclosing loop (exit·loop) continue; //jumps to bottom of loop

ARRAYS:

int[] x = new int[lOJ ; //ten zeros int[][] x = new int[5][5); //5 by 5 matrix int[] X = {1,2,3,4);

x.length; //int expression length of array int(][] x = ({1,2),{3,4,5)); //ragged array String[] y = new String[lO]; //10 nulls

//Note that object types are null by default

//loop through arr.ay:

for(int i=O;i<arrayname.length;i++)

//use arrayname[i);

//for-each loop through array int[) X = (10,20,30,40);

for(int v : x) (

*/ I v* cycles between 10,20,30,40

//Loop through \_ragged arrays: for(int i=O;i<x.length;i++)

for(int j=O;j<x(i].length;j++)

//CODE HERE

//Note, multi-dim arrays can have nulls

//in many places, especially object arrays: Integer[)[) x = {(1,2}, (3,null),null);

FUNCTI'ONS / METHODS:

**Static Declarations:**

public static int functionname ( ) private static double functionname ( ... ) static void functionname( J

Instance Declarations:

public void functionname( ) private int functionname( ) Arguments, Return Statement:

int myfunc(int argO, String argl)

return 5; //type matches *int* myfunc

//Non-void methods must return before ending

//Recursive functions should have an if

//statement base-case that returns at once

public static final int BALLRADIUS = 5;

//INSTANCE FIELDS

**private int x, y,** vx, vy;

public boolean randomPos = false;

//CONSTRUCTORS

public Ball(int x, int y, int vx, int vy)

(

this.x = x; this.y = y; this.vx = vx; this.vy = vy; numBalls++;

Ball() (

x = Math.random()\*lOO; y = Math.random()\*200; randomPos = true;

//INSTANCE METHODS

public int getX() ( return x; ) public int getY ()( return y; )

public int getVX(} ( return vx; public int getVY (){ return v·y",

public void move() { x+=vx; y+=vy; public boolean touching(Ball other)

float dx = x-other.x;

float dy \_= y-other.y; float rr = BALLRADIUS;

return Math. sqrt(dx\*dx+dy\*dy)<rr;

//Example Usage:

public static void main(String[ ] args) { Ball x = new Ball(S,10,2,2);

Ball y = new Ball();

List<Ball> balls = new ArrayList<Ball>(); balls.add(x); balls.add{y) ;

for(Ball b : balls) { for(Ball o : balls) {

if(b != o) ( //compares references boolean touch = b.touching(o);

® 2013 Kapparate, © 2013 .Kapparate Operating LLC