ACM ICPC Reference

University of Notre Dame

October 25, 2019

Contents

1 vimrc

```
syntax on
colors evening
set ai si noet ts=4 sw=4 sta sm nu so=7 t_Co=8
imap {<CR> {<CR>}<Esc>O
```

2 hashify.sh

```
#!/bin/bash
while IFS=$'\n' read -r line; do
    trim=$(echo "$line" | tr -d "[:space:]")
    md5=$(echo -n "${trim%%\/\/*}" | md5sum)
    md5=${md5:0:4}
    [ "${trim:~0}" == "$" ] && md5="@$md5@"
    echo "$md5 $line"
done
```

3 STL

3.1 Algorithms

```
a102 #include <algorithm>
d41d // sort, search, array
1bf7 sort(startaddress, endaddress)
e416 binary_search(startaddress, endaddress, valuetofind)
0c81 reverse(first_iterator, last_iterator)
74d6 *max_element (first_iterator, last_iterator)
e4a8 *min_element (first_iterator, last_iterator)
5324 accumulate(first_iterator, last_iterator, initial value of sum) //summation of vector elements
7c96 count(first_iterator, last_iterator,x) //occurrences of x
d278 find(first_iterator, last_iterator, x) // points to last if not found
9ea6 lower_bound(first_iterator, last_iterator, x) //first element in range [first, last) which has a value not
    less than x
d45e upper_bound(first_iterator, last_iterator, x)
213f arr.erase(position to be deleted) // erased element in vector
5c35 arr.erase(unique(arr.begin(),arr.end()),arr.end()) // erases the duplicate occurrences in sorted vector in a
    single line.
e161 next_permutation(first_iterator, last_iterator)
71d5 prev_permutation(first_iterator, last_iterator)
cc6d distance(first_iterator,desired_position) //very useful while finding the index.
f924 all_of(ar, ar+6, [](int x) { return x>0; }) ?//check every element for condition
3261 any_of() //check if there is an element with condition
ba64 none_of() //check if none
5caa copy_n(source, size, dest) //copy one array into another
39d0 iota(ar, ar+6, 20); // ar = 20, 21, 22, 23, 24, 25
d41d
d41d // partition
5aca partition(beg, end, condition)
5896 is_partitioned(beg, end, condition)
67d6 stable_partition(beg, end, condition) //order is preserved
febd partition_point(beg, end, condition) //returns iterator pointing to partion point
8cba partition_copy(beg, end, beg1, beg2, condition) // separate between true and false
d41d
```

3.2 Numeric

```
646d #include<valarray>
d41d // apply and sum
5d32 valarray<int> varr = { 10, 2, 20, 1, 30 };
4ac0 valarray<int> varr1 = varr.apply([](int x){return x=x+5;});
e7f1 varr1.sum()
d41d // min and max
5d32 valarray<int> varr = { 10, 2, 20, 1, 30 };
```

```
a6d5 varr.max()
ed45 varr.min()
d41d // shift and cshift
5d32 valarray<int> varr = { 10, 2, 20, 1, 30 };
3f80 valarray<int> varr1 = varr.shift(2); // {20, 1, 30, 0, 0}
92e2 varr1 = varr.cshift(-3); // {20, 1, 30, 10, 2}
d41d // swap
5d32 valarray<int> varr = { 10, 2, 20, 1, 30};
043d valarray<int> varr1 = { 2, 29, 3, 1, 41};
31f0 varr.swap(varr1);
d41d
```

3.3 List

```
c702 #include<list>
148f list <int> list1;
a5b7 list1.push_back(element)
8ba1 list1.push_front(element)
142f list1.front()
8a25 list1.back()
9947 list1.pop_front()
d803 list1.pop_back()
141d list1.reverse()
3722 list1.sort()
ecef list1.rbegin() // reverse iterator last element
2a73 list1.rend() //reverse iterator beggining of list
eee2 list1.empty() //empty 1 or not 0
ef20 list1.insert(pos_iter, ele_num, ele) //insert ele_num elements ar position pos_iter of value ele
0e66 list1.erase() //remove element
a68f list1.assign(count, value) //assigns new elements by replacing current elements
3e16 list1.remove(position)
d4bd list1.size()
a460 list1.unique(function)
d41d
```

3.4 Unordered Map

```
efe5 #include <unordered_map>
9418 unordered_map<string, double> umap;
7cae umap.at()
4bf0 umap.begin()
7f7d umap.end()
1d78 umap.bucket(key)
c529 umap.bucket_count()
1bbc umap.count(key)
ea39 umap.equal_range() //can be used as search
d41d
```

3.5 Map

```
7306 #include <map>
59ab map<int, int> map1;
c79e map1.begin()
8aac map1.end()
f588 map1.size()
f503 map1.max_size()
2401 map1.empty()
512c map.insert({key, element})
18b2 map.erase(pos)
9286 map.erase(const g) //remove the key value g
47ff map.clear() //remove all elements
```

3.6 Set

```
bd50 #include <set>
cb62 set <int, greater <int> > set1
c6c0 set1.begin()
12f9 set1.end()
00e1 set1.size()
66c6 set1.empty()
e375 set1.rbegin()
5a33 set1.rend()
70db set1.insert(const g)
75cf set1.erase(pos)
e9ef set1.erase(const g)
5a5b set1.clear()
4d8d set1.find(const g)
a286 set1.upper_bound(const g)
6b2c set1.lower_bound(const g)
ef74 set1.swap()
313a set1.emplace()
d41d
```

3.7 Vector

```
ee72 #include <vector>
82b2 vector<int> v1;
e483 v1.begin()
3e1a v1.end()
c689 v1.rbegin()
840a v1.rend()
4159 v1.size()
289d v1.max_size()
e09b v1.capacity()
7bea v1.resize(n)
c752 v1.empty() //whether is empty
cfe6 v1.shrink_to_fit(n) //reduce capacity and destroys all beyond
68ba v1.reserve(n) //capacity to be at least enough to contain n elements
e106 v1.at()
6345 v1.front()
cac5 v1.back()
026d v1.data()
49a3 v1.assign()
692a v1.push_back()
fa5e v1.pop_back()
8e7e v1.insert()
b498 v1.erase()
2ba5 v1.swap()
5434 v1.clear()
d5cf v1.emplace()
d41d
```

3.8 Stack

```
76cb #include <stack>
4ce3 stack <int> s;
161e s.empty()
e958 s.size()
d03c s.top()
b88a s.push(g)
cc35 s.pop()
```

3.9 Queue

```
1bba #include <queue>
26a5 queue <int> q;
5a9d queue <int> p;
2d70 q.empty()
439a q.size()
```

```
1e4e q.swap(p)
2a43 q.emplace()
3c63 q.front()
df4e q.back()
3be4 q.push(g)
a46b q.pop()
d41d
```

3.10 Deque

```
8467 #include <deque>
1e15 deque <int> d;
fcfe d.insert()
ddcc d.rbegin()
c62e d.rend()
5d62 d.begin()
7cdf d.end()
828d d.assign()
73e7 d.resize()
6a9b d.push_front()
e1e8 d.push_back()
e22a d.pop_front()
623b d.pop_back()
fe45 d.front()
36b8 d.back()
9088 d.clear()
8fe7 d.erase()
8069 d.empty()
23a4 d.size()
d41d
```

4 Geometry

4.1 Base

```
d41d // typedef double cood; cood eps = 1e-8; // risky: XXX, untested: TODO
00a0 const double pi = acos(-1.);
ccb5 template<typename T> inline T sq(T x) { return x*x; }
87bc struct vec {
b86a ▹ cood x, y;
6e4f ▹ vec () : x(0), y(0) {} vec (cood a, cood b) : x(a), y(b) {}
741a ▹ inline vec operator - (vec o) { return {x - o.x, y - o.y}; }
ff7e ▹ inline vec operator + (vec o) { return {x + o.x, y + o.y}; }
b6dd ▹ inline vec operator * (cood o) { return {x * o, y * o}; }
2711 ▹ inline vec operator / (cood o) { return {x / o, y / o}; }
6ac9 ▹ inline cood operator ^ (vec o) { return x * o.y - y * o.x; }
83dd ▹ inline cood operator * (vec o) { return x * o.x + y * o.y; }
46ef ▹ inline cood cross (vec a, vec b) { return ((*this)-a) ^ ((*this)-b); } // |(this)a||(this)b|sen(angle)
cbad ▹ inline cood inner (vec a, vec b) { return ((*this)-a) * ((*this)-b); } // |(this)a||(this)b|cos(angle)
cddd ▹ inline double angle (vec a, vec b) { return atan2(cross(a,b),inner(a,b)); } // ccw angle from (this)a to
    (this)b in range [-pi,pi]
e4d3 ▹ inline int ccw (vec a, vec b) { cood o = cross(a,b); return (eps < o) - (o < -eps); } // this is to the
    (1 left, 0 over, -1 right) of ab
2e1f ▹ inline int dir (vec a, vec b) { cood o = inner(a,b); return (eps < o) - (o < -eps); } // a(this) is to
    the (1 same, 0 none, -1 opposite) direction of ab
5d26 ▹ inline cood sq (vec o = vec()) { return inner(o,o); }
e7cf ▹ inline double nr (vec o = vec()) { return sqrt(sq(o)); } //$
4e72 ▹ inline vec operator ~ () { return (*this)/nr(); }
f149 ▹ inline vec proj (vec a, vec b) { return a + (b-a)*(a.inner((*this),b) / a.sq(b)); } // projects this onto
    line ab
1664 ▹ inline vec rotate (double a) { return vec(cos(a) * x - sin(a) * y, sin(a) * x + cos(a) * y); } // ccw by
    a radians
3206 ▹ inline vec rot90 () { return vec(-y,x); } // rotate(pi/2)$
2810 ▹ bool in_seg (vec a, vec b) { return ccw(a,b) == 0 && dir(a,b) <= 0; } // tips included
5e56 ▹ double dist2_lin (vec a, vec b) { return a.sq(b) <= eps ? sq(a) : double(::sq(cross(a,b)))/a.sq(b); } //
    see cir.has_inter_lin
```

```
8831 	double dist2_seg (vec a, vec b) { return a.dir((*this),b) == (b.dir((*this),a)) ? dist2_lin(a,b) :
    min(sq(a),sq(b)); }
436b 	inline bool operator == (const vec & o) const { return abs(x-o.x) <= eps && abs(y-o.y) <= eps; }
5522 	inline bool operator < (const vec & o) const { return (abs(x-o.x)>eps)?(x < o.x):(y > o.y); } // lex
    compare (inc x, dec y)
d41d 	// full ccw angle strict compare beginning upwards (this+(0,1)) around (*this)
d41d 	// incresing distance on ties, this is the first
69ad 	bool compare (vec a, vec b) {
a482 		if ((*this < a) != (*this < b)) return *this < b;
bdb1 		int o = ccw(a,b); return o?o>0:((a == *this && !(a == b)) || a.dir(*this,b) < 0);
cbb1 	}
2145 }; //$
bafe struct lin { // line
6143 	vec p; cood c; // p*(x,y) = c
1105 	lin () {} lin (vec a, cood b) : p(a), c(b) {}
d036 	lin (vec s, vec t) : p((s-t).rot90()), c(p*s) {}
5c8b 	inline lin parll (vec v) { return lin(p,v*p); }
1263 	inline lin perp () { return lin(p.rot90(),c); }
3838 	vec inter (lin o) { if (vec(0,0).ccw(p,o.p) == 0) throw 1; cood d = (p^o.p); return vec((c*o.p.y -
    p.y*o.c)/d,(o.c*p.x - o.p.x*c)/d); }
1375 	bool contains (vec v) { return abs(p*v - c) <= eps; }
eda5 	vec at_x (cood x) { return vec(x,(c-p.x*x)/p.y); }
c0fb 	vec at_y (cood y) { return vec((c-y*p.y)/p.x,y); }
e1ef 	double sign_dist (vec v) { return double(p*v - c)/p.nr(); }
2145 }; //$
3236 struct cir { // circle
b6d3 	vec c; cood r;
126a 	cir () {} cir (vec v, cood d) : c(v), r(d) {}
c118 	cir (vec u, vec v, vec w) { // XXX untreated degenerates
0fb6 		vec mv = (u+v)/2; lin s(mv, mv+(v-u).rot90());
bf5f 		vec mw = (u+w)/2; lin t(mw, mw+(w-u).rot90());
a0c4 		c = s.inter(t); r = c.nr(u);
cbb1 	}//$
9e54 	inline bool contains (vec w) { return c.sq(w) <= sq(r) + eps; } // border included
0549 	inline bool border (vec w) { return abs(c.sq(w) - sq(r)) <= eps; }
1cd6 	inline bool has_inter (cir o) { return c.sq(o.c) <= sq(r + o.r) + eps; } // borders included
376d 	inline bool has_border_inter (cir o) { return has_inter(o) && c.sq(o.c) + eps >= sq(r - o.r); }
8ab4 	inline bool has_inter_lin (vec a, vec b) { return a.sq(b) <= eps ? contains(a) : sq(c.cross(a,b)) <=
    sq(r)*a.sq(b) + eps; } // borders included XXX overflow
9bf7 	inline bool has_inter_seg (vec a, vec b) { return has_inter_lin(a,b) && (contains(a) || contains(b) ||
    a.dir(c,b)*b.dir(c,a) != -1); } // borders and tips included XXX overflow
7abe 	inline double arc_area (vec a, vec b) { return c.angle(a,b)*r*r/2; } // smallest arc, ccw positive
f967 	inline double arc_len (vec a, vec b) { return c.angle(a,b)*r; } // smallest arc, ccw positive$
771f 	pair<vec,vec> tan (vec v) { // XXX low precision
84ec 		if (contains(v) && !border(v)) throw 0;
2894 		cood d2 = c.sq(v); double s = sqrt(d2 - r*r); s = (s==s)?s:0;
0f70 		double al = atan2(r,s); vec t = (~(c-v));
3a69 		return pair<vec,vec>(v + t.rotate(al)*s, v + t.rotate(-al)*s);
cbb1 	}//$
c56f 	pair<vec,vec> border_inter (cir o) {
c4d4 		if (!has_border_inter(o) || o.c == (*this).c) throw 0;
2b40 		double a = (sq(r) + o.c.sq(c) - sq(o.r))/(2*o.c.nr(c));
b647 		vec v = (o.c - c)/o.c.nr(c); vec m = c + v * a;
65b9 		double h = sqrt(sq(r) - sq(a)); h = h!=h?0:h;
440c 		return pair<vec,vec>(m + v.rot90()*h, m - v.rot90()*h);
cbb1 	}//$
5182 	pair<vec,vec> border_inter_lin (vec a, vec b) { // first is closest to a than second
c6e7 		if (a.sq(b) <= eps) { if (border(a)) return pair<vec,vec>(a,a); throw 0; }
40f6 		if (a.dir(b,c) == -1) swap(a,b);
45ab 		if (!has_inter_lin(a,b)) throw 0;
5cb6 		double d2 = c.dist2_lin(a,b); vec p = (b-a)/a.nr(b);
0aca 		double h = sqrt(r*r - d2); h = h!=h?0:h;
ddf2 		double y = sqrt(c.sq(a) - d2); y = y!=y?0:y;
5539 		return pair<vec,vec>(a + p*(y-h), a + p*(y+h));
cbb1 	}//$
be35 	double triang_inter (vec a, vec b) { // ccw oriented, this with (c,a,b)
53ba 		if (c.sq(a) > c.sq(b)) return -triang_inter(b,a);
148a 		if (contains(b)) return c.cross(a,b)/2;
7434 		if (!has_inter_seg(a,b)) return arc_area(a,b);
```

```
773a 		pair<vec,vec> itr = border_inter_lin(b,a); // order important
12a9 		if (contains(a)) return c.cross(a,itr.first)/2 + arc_area(itr.first,b);
c2f4 		return arc_area(a,itr.second) + c.cross(itr.second,itr.first)/2 + arc_area(itr.first,b);
cbb1 	}
2145 }; //$
a71b bool inter_seg (vec a, vec b, vec c, vec d) {
2397 	if (a.in_seg(c, d) || b.in_seg(c, d) || c.in_seg(a, b) || d.in_seg(a, b)) return true;
bbbd 	return (c.ccw(a, b) * d.ccw(a, b) == -1 && a.ccw(c, d) * b.ccw(c, d) == -1);
cbb1 }
e0fd double dist2_seg (vec a, vec b, vec c, vec d){return inter_seg(a,b,c,d)?0.:min({ a.dist2_seg(c,d),
     b.dist2_seg(c,d), c.dist2_seg(a,b), d.dist2_seg(a,b) });}
```

4.2 Advanced

```
484c cir min_spanning_circle (vec * v, int n) { // n
f1ea 	srand(time(NULL)); random_shuffle(v, v+n); cir c(vec(), 0); int i,j,k;
b11a 	for (i = 0; i < n; i++) if (!c.contains(v[i]))
e5b6 		for (c = cir(v[i],0), j = 0; j < i; j++) if (!c.contains(v[j]))
a47c 			for (c = cir((v[i] + v[j])/2,v[i].nr(v[j])/2), k = 0; k < j; k++) if (!c.contains(v[k]))
3dd3 					c = cir(v[i],v[j],v[k]);
807f 	return c;
cbb1 }//$
d45c int convex_hull (vec * v, int n, int border_in) { // nlg | border_in (should border points stay?)
4f17 	swap(v[0], *min_element(v,v+n)); int s, i;
f37e 	sort(v+1, v+n, [&v] (vec a, vec b) { int o = b.ccw(v[0], a); return (o?o==1:v[0].sq(a)<v[0].sq(b)); });
a69c 	if (border_in) {
9492 		for (s = n-1; s > 1 && v[s].ccw(v[s-1],v[0]) == 0; s--);
0bb0 		reverse(v+s, v+n);
cbb1 	}
c497 	for (i = s = 0; i < n; i++) if (!s || !(v[s-1] == v[i])) {
cea9 		for (; s >= 2 && v[s-1].ccw(v[s-2],v[i]) >= border_in; s--);
ceca 		swap(v[s++],v[i]);
cbb1 	}
0478 	return s;
cbb1 }//$
79b9 int monotone_chain (vec * v, int n, int border_in) { // nlg | border_in (should border points stay?)
5031 	vector<vec> r; sort(v, v+n); n = unique(v, v+n) - v;
d885 	for (int i = 0; i < n; r.pb(v[i++])) while (r.size() >= 2 && r[r.size()-2].ccw(r.back(),v[i]) <=
     -border_in) r.pop_back();
dd80 	r.pop_back(); unsigned int s = r.size();
c19d 	for (int i = n-1; i >= 0; r.pb(v[i--])) while (r.size() >= s+2 && r[r.size()-2].ccw(r.back(),v[i]) <=
     -border_in) r.pop_back();
a255 	return copy(r.begin(), r.end() - (r.size() > 1), v) - v;
cbb1 }//$
f80f double polygon_inter (vec * p, int n, cir c) { // signed area
2eae 	return inner_product(p, p+n-1, p+1, c.triang_inter(p[n-1],p[0]), std::plus<double>(), [&c] (vec a, vec b)
     { return c.triang_inter(a,b); });
cbb1 }//$
3214 int polygon_pos (vec * p, int n, vec v) { // lg | p should be simple (-1 out, 0 border, 1 in)
6c2a 	int in = -1; // it's a good idea to randomly rotate the points in the double case, numerically safer
6033 	for (int i = 0; i < n; i++) {
2bca 		vec a = p[i], b = p[i?i-1:n-1]; if (a.x > b.x) swap(a,b);
c9e9 		if (a.x + eps <= v.x && v.x < b.x + eps) { in *= v.ccw(a,b); }
c3b1 		else if (v.in_seg(a,b)) { return 0; }
cbb1 	}
091d 	return in;
cbb1 }//$
271f int polygon_pos_convex (vec * p, int n, vec v) { // lg(n) | (-1 out, 0 border, 1 in) TODO
a868 	if (v.sq(p[0]) <= eps) return 0;
088f 	if (n <= 1) { return 0; } if (n == 2) { return v.in_seg(p[0],p[1])?0:-1; }
2ceb 	if (v.ccw(p[0],p[1]) < 0 || v.ccw(p[0],p[n-1]) > 0) return -1;
fcfd 	int di = lower_bound(p+1,p+n-1,v, [&p](vec a,vec v) { return v.ccw(p[0],a) > 0; }) - p;
adf3 	if (di == 1) return v.ccw(p[1],p[2]) >= 0?0:-1;
cfa4 	return v.ccw(p[di-1],p[di]);
cbb1 }//$
d41d // v is the pointset, w is auxiliary with size at least equal to v's
bf98 cood closest_pair (vec * v, vec * w, int l, int r, bool sorted = 0) { // nlg | r is exclusive TODO (AC on
     cf, no test)
```

```
91d7 	if (l + 1 >= r) return inf;
900b 	if (!sorted) sort(v+l,v+r,[](vec a, vec b){ return a.x < b.x; });
89cd 	int m = (l+r)/2; cood x = v[m].x;
1a44 	cood res = min(closest_pair(v,w,l,m,1),closest_pair(v,w,m,r,1));
d046 	merge(v+l,v+m,v+m,v+r,w+l,[](vec a, vec b){ return a.y < b.y; });
2dd0 	for (int i = l, s = l; i < r; i++) if (sq((v[i] = w[i]).x - x) < res) {
ad96 		for (int j = s-1; j >= l && sq(w[i].y - w[j].y) < res; j--)
c3b1 			res = min(res, w[i].sq(w[j]));
1991 		w[s++] = v[i];
cbb1 	}
b505 	return res;
cbb1 }//$
ac2e double union_area (cir * v, int n) { // n^2lg | XXX joins equal circles TODO (AC on szkopul, no tests)
c765 	struct I { vec v; int i; } c[2*(n+4)];
cf66 	srand(time(NULL)); cood res = 0; vector<bool> usd(n);
dd83 	cood lim = 1./0.; for (int i = 0; i < n; i++) lim = min(lim, v[i].c.y - v[i].r - 1);
0b02 	for (int i = 0, ss = 0; i < n; i++, ss = 0) {
dc37 		vec fp = v[i].c + vec(0,v[i].r).rotate(rand()); // rotation avoids corner on cnt initialization
6e87 		int cnt = 0, eq = 0;
578e 		for (int j = 0; j < n; j++) {
df48 			cnt += (usd[j] = v[j].contains(fp));
2311 			if (!v[i].has_border_inter(v[j])) continue;
8daa 			if (v[i].c == v[j].c) eq++;
4e6b 			else {
e59e 				pair<vec,vec> r = v[i].border_inter(v[j]);
0782 				c[ss++] = {r.first, j}; c[ss++] = {r.second, j};
cbb1 			}
cbb1 		}
d21b 		vec d = vec(v[i].r,0); for (int k = 0; k < 4; k++, d = d.rot90()) c[ss++] = {v[i].c + d, i};
85d3 		int md = partition(c,c+ss,[v,i,fp](I a){return a.v.ccw(v[i].c,fp) > 0;}) - c;
19c7 		sort(c,c+md,[v,i](I a,I b){return a.v.ccw(v[i].c,b.v) < 0;});
7430 		sort(c+md,c+ss,[v,i](I a,I b){return a.v.ccw(v[i].c,b.v) < 0;});
56cd 		for (int j = 0; j < ss; j++) {
2b5e 			if (c[j].i != i) { cnt -= usd[c[j].i]; usd[c[j].i] = !usd[c[j].i]; cnt += usd[c[j].i]; }
b115 			vec a = c[j].v, b = c[(j+1)%ss].v;
7c4a 			cood cir = abs(v[i].arc_area(a,b) - v[i].c.cross(a,b)/2), tra = abs((b.x-a.x)*(a.y+b.y-2*lim)/2);
e20b 			cood loc = (a.x<b.x)?cir-tra:tra+cir; res += (cnt==eq)?loc/eq:0;
cbb1 		}
cbb1 	}
b505 	return res;
cbb1 }//$
4ede pii antipodal (vec * p, int n, vec v) { // lg(n) | extreme segments relative to direction v TODO
d41d 	// po: closest to dir, ne: furthest from dir
3bd9 	bool sw = ((p[1]-p[0])*v < 0);
d189 	if (sw) v = vec(0,0) - v; // lower_bound returns the first such that lambda is false
0303 	int md = lower_bound(p+1, p+n, v, [p] (vec & a, vec v) { return (a-p[0])*v > eps; }) - p; // chain
     separation
25f1 	int po = lower_bound(p, p+md-1, v, [p,n] (vec & a, vec v) { return (p[(&a+1-p)%n]-a)*v > eps; }) - p; //
     positive
9dc9 	int ne = (lower_bound(p+md, p+n, v, [p,n] (vec & a, vec v) { return (p[(&a+1-p)%n]-a)*v <= eps; }) -
     p)%n; // negative
5703 	if (sw) swap(po,ne);
ef0b 	return pii(po,ne);
cbb1 }//$
34e2 int mink_sum (vec * a, int n, vec * b, int m, vec * r) { // (n+m) | a[0]+b[0] should belong to sum, doesn't
     create new border points TODO
8d81 	if (!n || !m) { return 0; } int i, j, s; r[0] = a[0] + b[0];
de54 	for (i = 0, j = 0, s = 1; i < n || j < m; s++) {
1ab0 		if (i >= n) j++;
1dc4 		else if (j >= m) i++;
4e6b 		else {
4f09 			int o = (a[(i+1)%n]+b[j%m]).ccw(r[s-1],a[i%n]+b[(j+1)%m]);
e43c 			j += (o >= 0); i += (o <= 0);
cbb1 		}
f5b4 		r[s] = a[i%n] + b[j%m];
cbb1 	}
162b 	return s-1;
cbb1 }//$
9e65 int inter_convex (vec * p, int n, vec * q, int m, vec * r) { // (n+m) | XXX
```

```
2d76     int a = 0, b = 0, aa = 0, ba = 0, inflag = 0, s = 0;
2a6c     while ((aa < n || ba < m) && aa < n+n && ba < m+m) {
b977         vec p1 = p[a], p2 = p[(a+1)%n], q1 = q[b], q2 = q[(b+1)%m];
35b2         vec A = p2 - p1, B = q2 - q1;
1479         int cross = vec(0,0).ccw(A,B), ha = p1.ccw(p2,q2), hb = q1.ccw(q2,p2);
c6e0         if (cross == 0 && p2.ccw(p1,q1) == 0 && A*B < -eps) {
507b             if (q1.in_seg(p1,p2)) r[s++] = q1;
5e83             if (q2.in_seg(p1,p2)) r[s++] = q2;
ce58             if (p1.in_seg(q1,q2)) r[s++] = p1;
526a             if (p2.in_seg(q1,q2)) r[s++] = p2;
7b25             if (s < 2) return s;
e2a8             inflag = 1; break;
5e6d         } else if (cross != 0 && inter_seg(p1,p2,q1,q2)) {
f420             if (inflag == 0) aa = ba = 0;
2b81             r[s++] = lin(p1,p2).inter(lin(q1,q2));
37fd             inflag = (hb > 0) ? 1 : -1;
cbb1         }
5499         if (cross == 0 && hb < 0 && ha < 0) return s;
0872         bool t = cross == 0 && hb == 0 && ha == 0;
c0ec         if (t ? (inflag == 1) : (cross >= 0) ? (ha <= 0) : (hb > 0)) {
9873             if (inflag == -1) r[s++] = q2;
1146             ba++; b++; b %= m;
9d97         } else {
5c98             if (inflag == 1) r[s++] = p2;
5ecb             aa++; a++; a %= n;
cbb1         }
cbb1     }
c1b2     if (inflag == 0) {
3880         if (polygon_pos_convex(q,m,p[0]) >= 0) { copy(p, p+n, r); return n; }
115c         if (polygon_pos_convex(p,n,q[0]) >= 0) { copy(q, q+m, r); return m; }
cbb1     }
fc37     s = unique(r, r+s) - r;
2629     if (s > 1 && r[0] == r[s-1]) s--;
0478     return s;
cbb1 }//$
03ae bool isear (vec * p, int n, int i, int prev[], int next[]) { // aux to triangulate
7630     vec a = p[prev[i]], b = p[next[i]];
2d9f     if (b.ccw(a,p[i]) <= 0) return false;
578e     for (int j = 0; j < n; j++) {
97eb         if (j == prev[i] || j == next[i]) continue;
0ef9         if (p[j].ccw(a,p[i]) >= 0 && p[j].ccw(p[i],b) >= 0 && p[j].ccw(b,a) >= 0) return false;
0639         int k = (j+1)%n;
2898         if (k == prev[i] || k == next[i]) continue;
a537         if (inter_seg(p[j],p[k],a,b)) return false;
cbb1     }
8a6c     return true;
cbb1 }
1851 int triangulate (vec * p, int n, bool ear[], int prev[], int next[], int tri[][3]) { // O(n^2) | n >= 3
d14e     int s = 0, i = 0;
78d0     for (int i = 0, prv = n-1; i < n; i++) { prev[i] = prv; prv = i; next[i] = (i+1)%n; ear[i] =
       isear(p,n,i,prev,next); }
6b3b     for (int lef = n; lef > 3; lef--, i = next[i]) {
ced7         while (!ear[i]) i = next[i];
e7a9         tri[s][0] = prev[i]; tri[s][1] = i; tri[s][2] = next[i]; s++; // tri[i][0],i,tri[i][1] inserted
e0c0         int c_prev = prev[i], c_next = next[i];
c354         next[c_prev] = c_next; prev[c_next] = c_prev;
84b6         ear[c_prev] = isear(p,n,c_prev,prev,next); ear[c_next] = isear(p,n,c_next,prev,next);
cbb1     }
bc1d     tri[s][0] = next[next[i]]; tri[s][1] = i; tri[s][2] = next[i]; s++; // tri[i][0],i,tri[i][1] inserted
0478     return s;
cbb1 }
```

4.3 3D

```
f61c const double pi = acos(-1);
d41d // typedef double cood; cood eps = 1e-6; // risky: XXX, untested: TODO
3f73 struct pnt { // TODO it's not tested at all :)
5e43     cood x, y, z;
```

```
cf2f     pnt () : x(0), y(0), z(0) {} pnt (cood a, cood b, cood c) : x(a), y(b), z(c) {}
4e90     inline pnt operator - (pnt o) { return pnt(x - o.x, y - o.y, z - o.z); }
2b18     inline pnt operator + (pnt o) { return pnt(x + o.x, y + o.y, z + o.z); }
7470     inline pnt operator * (cood o) { return pnt(x*o, y*o, z*o); }
8194     inline pnt operator / (cood o) { return pnt(x/o, y/o, z/o); }
a269     inline cood operator * (pnt o) { return x*o.x + y*o.y + z*o.z; } // inner: |this||o|*cos(ang)
079c     inline pnt operator ^ (pnt o) { return pnt(y*o.z - z*o.y, z*o.x - x*o.z, x*o.y - y*o.x); } // cross: oriented normal to the plane containing the two vectors, has norm |this||o|*sin(ang)
a2ea     inline cood operator () (pnt a, pnt b) { return (*this)*(a^b); } // mixed: positive on the right-hand rule (thumb=this,index=a,mid=b)
d41d
f500     inline cood inner (pnt a, pnt b) { return (a-(*this))*(b-(*this)); }
4114     inline pnt cross (pnt a, pnt b) { return (a-(*this))^(b-(*this)); } // its norm is twice area of triangle
fa90     inline cood mixed (pnt a, pnt b, pnt c) { return (a-(*this))(b-(*this),c-(*this)); } // 6 times the oriented area of thetahedra
d41d
4f78     inline cood sq (pnt o = pnt()) { return inner(o,o); }
113b     inline double nr (pnt o = pnt()) { return sqrt(sq(o)); }
6edf     inline pnt operator ~ () { return (*this)/nr(); }
d41d
11c0     inline bool in_seg (pnt a, pnt b) { return cross(a,b).sq() <= eps && inner(a,b) <= eps; } // tips included
a6b7     inline bool in_tri (pnt a, pnt b, pnt c) { return abs(mixed(a,b,c)) <= eps && cross(a,b)*cross(b,c) >= -eps && cross(a,b)*cross(c,a) >= -eps; } // border included$
d41d
7c26     inline pnt proj (pnt a, pnt b) { return a + (b-a)*a.inner(b,(*this))/a.sq(b); }
3a26     inline pnt proj (pnt a, pnt b, pnt c) { pnt n = a.cross(b,c); return (*this) - n*(n*((*this)-a))/n.sq(); }
d41d
8fbb     inline double dist2_lin (pnt a, pnt b) { return cross(a,b).sq()/a.sq(b); }
1880     inline double dist2_seg (pnt a, pnt b) { return a.inner(b,(*this))*b.inner(a,(*this)) <= eps ? min(sq(a),sq(b)) : dist2_lin(a,b); }
39c1     inline double dist_pln (pnt a, pnt b, pnt c) { return abs((~a.cross(b,c))*((*this)-a)); }
5bc2     inline double dist2_tri (pnt a, pnt b, pnt c) { pnt p = proj(a,b,c); return p.in_tri(a,b,c) ? sq(p) : min({ dist2_seg(a,b), dist2_seg(b,c), dist2_seg(c,a) }); }
2145 };
eb48 inline cood area (pnt a, pnt b, pnt c) { return abs(a.cross(b,c).nr()) / 2; }
a6c7 inline cood vol (pnt a, pnt b, pnt c, pnt d) { return abs(a.mixed(b,c,d)) / 6; } // thetahedra
084a pnt inter_lin_pln (pnt s, pnt t, pnt a, pnt b, pnt c) { pnt n = a.cross(b,c); return s + (t-s)*(n*(a-s))/(n*(t-s)); } //$
fabc struct sph { // TODO it's also not tested at all
af42     pnt c; cood r;
390f     sph () : c(), r(0) {} sph (pnt a, cood b) : c(a), r(b) {}
baaf     inline pnt operator () (cood lat, cood lon) { return c + pnt(cos(lat)*cos(lon), sin(lon), sin(lat))*r; } // (1,0,0) is (0,0). z is height.
171a     inline double area_hull (double h) { return 2.*pi*r*h; }
60a4     inline double vol_hull (double h) { return pi*h/6 * (3.*r*r + h*h); }
2145 };
```

5 Graphs

5.1 DFS

```
d41d // const int N = ;
a301 vector<int> adj[N]; // adj[i] is the adjacency list of vertex i
a692 bool visited[N]; // visited[i] is true iff vertex i was visited
d41d
58bf void add_edge(int u, int v) {
cc97     adj[u].push_back(v);
d41d     // if graph is undirected:
d41d     // adj[v].push_back(u);
cbb1 }
d41d
9e49 void dfs_util(int u) {
2a95     visited[u] = true;
895e     for (int v : adj[u])
4373         if (!visited[v])
9518             dfs_util(v);
cbb1 }
```

```
d41d
4635 void dfs(int n) {
8300     for (int i = 0; i < n; i++)
88e7         visited[i] = false;
8300     for (int i = 0; i < n; i++)
b1c5         if (!visited[i])
9370             dfs_util(i);
cbb1 }
```

5.2 BFS

```
d41d // const int N = ;
d41d
a301 vector<int> adj[N]; // adj[i] is the adjacency list of vertex i
a692 bool visited[N]; // visited[i] is true iff vertex i was visited
77e0 int dist[N]; // dist[i] is distance from start to vertex i
d41d
58bf void add_edge(int u, int v) {
cc97     adj[u].push_back(v);
d41d     // if graph is undirected:
d41d     // adj[v].push_back(u);
cbb1 }
d41d
a20b void bfs(int start) {
26a5     queue<int> q;
7292     q.push(start);
fe25     visited[start] = true;
4c87     dist[start] = 0;
d41d
14d7     while (!q.empty()) {
be15         int u = q.front(); q.pop();
d41d
3722         for (int v : adj[u]) {
451a             if (!visited[v]) {
2a1e                 q.push(v);
e758                 visited[v] = true;
8a03                 dist[v] = dist[u] + 1;
cbb1             }
cbb1         }
cbb1     }
cbb1 }
d41d
```

5.3 Dinic

```
d41d //typedef int num; const int N = ; const int M = * 2; const num eps = 0;
582d struct dinic {
656d     int hd[N], seen[N], qu[N], lv[N], ei[N], to[M], nx[M]; num fl[M], cp[M]; int en = 2; int when = 0;
1233     bool bfs(int s, int t) {
876c         seen[t] = ++when; lv[t] = 0; int ql = 0, qr = 0; qu[qr++] = t;
a872         while(ql != qr) {
036d             t = qu[ql++]; ei[t] = hd[t]; if(s == t) return true;
9a44             for(int e = hd[t]; e; e = nx[e]) if(seen[to[e]] != when && cp[e ^ 1] - fl[e ^ 1] > eps) {
d4fb                 seen[to[e]] = when;
de5c                 lv[to[e]] = lv[t] + 1;
f0ff                 qu[qr++] = to[e];
cbb1             }
cbb1         }
d1fe         return false;
cbb1     }
a444     num dfs(int s, int t, num f) {
f449         if(s == t) return f;
cebe         for(int &e = ei[s]; e; e = nx[e]) if(ei[to[e]] && seen[to[e]] == when && cp[e] - fl[e] > eps &&
    lv[to[e]] == lv[s] - 1)
7004             if(num rf = dfs(to[e], t, min(f, cp[e] - fl[e]))) {
805c                 fl[e] += rf;
5226                 fl[e ^ 1] -= rf;
```

```
2cb7                return rf;
cbb1            }
bb30        return 0;
cbb1    }
d41d    // public $
de22    num max_flow(int s, int t) {
6cb2        num fl = 0;
1c5e        while (bfs(s, t)) for(num f; (f = dfs(s, t, numeric_limits<num>::max())); fl += f);
e508        return fl;
cbb1    }
5a3f    void add_edge(int a, int b, num c, num rc=0) {
d03a        to[en] = b; nx[en] = hd[a]; fl[en] = 0; cp[en] = c; hd[a] = en++;
2f94        to[en] = a; nx[en] = hd[b]; fl[en] = 0; cp[en] = rc; hd[b] = en++;
cbb1    }
7415    void reset_flow() { memset(fl, 0, sizeof(num) * en); }
ae0a    void init(int n=N) { en = 2; memset(hd, 0, sizeof(int) * n); } // resets all
2145 };
```

5.4 MinCost MaxFlow

```
d41d //typedef int val; // type of flow
d41d //typedef int num; // type of cost
d41d //const int N = , M = * 2; const num eps = 0;
1854 struct mcmf {
933f    int es[N], to[M], nx[M], en = 2, par[N], seen[N], when, qu[N];
ef55    val fl[M], cp[M], flow; num cs[M], d[N], tot;
d0cc    val spfa(int s, int t) {
104f        when++; int a = 0, b = 0;
e0c6        for(int i = 0; i < N; i++) d[i] = numeric_limits<num>::max();
3518        d[s] = 0; qu[b++] = s; seen[s] = when;
9841        while(a != b) {
32d9            int u = qu[a++]; if(a == N) a = 0; seen[u] = 0;
a86f            for(int e = es[u]; e; e = nx[e]) if(cp[e] - fl[e] > val(0) && d[u] + cs[e] < d[to[e]] - eps) {
1ee4                d[to[e]] = d[u] + cs[e]; par[to[e]] = e ^ 1;
85b7                if(seen[to[e]] < when) { seen[to[e]] = when; qu[b++] = to[e]; if(b == N) b = 0; }
cbb1            }
cbb1        }
8e2a        if(d[t] == numeric_limits<num>::max()) return false;
91fe        val mx = numeric_limits<val>::max();
bf09        for(int u = t; u != s; u = to[par[u]])
05dc            mx = min(mx, cp[par[u] ^ 1] - fl[par[u] ^ 1]);
6de0        tot += d[t] * val(mx);
bf09        for(int u = t; u != s; u = to[par[u]])
ae7b            fl[par[u]] -= mx, fl[par[u] ^ 1] += mx;
b9aa        return mx;
cbb1    }
d41d    // public $
8662    num min_cost(int s, int t) {
3b69        tot = 0; flow = 0;
e66e        while(val a = spfa(s, t)) flow += a;
126a        return tot;
cbb1    }
457a    void add_edge(int u, int v, val c, num s) {
1d08        fl[en] = 0; cp[en] = c; to[en] = v; nx[en] = es[u]; cs[en] = s; es[u] = en++;
8015        fl[en] = 0; cp[en] = 0; to[en] = u; nx[en] = es[v]; cs[en] = -s; es[v] = en++;
cbb1    }
8537    void reset_flow() { memset(fl, 0, sizeof(val) * en); }
451f    void init(int n) { en = 2; memset(es, 0, sizeof(int) * n); } // XXX must be called
2145 };
```

5.5 Cycle Cancelling

```
d41d //typedef int val; // type of flow
d41d //typedef int num; // type of cost
d41d //const int N = ; const int M = * 2; const val eps = 0;
afb2 struct cycle_cancel {
```

```
e36d   int hd[N], seen[N], qu[N], lv[N], ei[N], to[M], nx[M], ct[N], par[N]; val fl[M], cp[M], flow; num cs[M],
       d[N], tot; int en = 2, n; int when = 0;
1233   bool bfs(int s, int t) {
876c       seen[t] = ++when; lv[t] = 0; int ql = 0, qr = 0; qu[qr++] = t;
a872       while(ql != qr) {
036d           t = qu[ql++]; ei[t] = hd[t]; if(s == t) return true;
9a44           for(int e = hd[t]; e; e = nx[e]) if(seen[to[e]] != when && cp[e ^ 1] - fl[e ^ 1] > eps) {
d4fb               seen[to[e]] = when;
de5c               lv[to[e]] = lv[t] + 1;
f0ff               qu[qr++] = to[e];
cbb1           }
cbb1       }
d1fe       return false;
cbb1   }
e4d9   val dfs(int s, int t, val f) {
f449       if(s == t) return f;
cebe       for(int &e = ei[s]; e; e = nx[e]) if(ei[to[e]] && seen[to[e]] == when && cp[e] - fl[e] > eps &&
       lv[to[e]] == lv[s] - 1)
9fe1           if(val rf = dfs(to[e], t, min(f, cp[e] - fl[e]))) {
805c               fl[e] += rf;
5226               fl[e ^ 1] -= rf;
2cb7               return rf;
cbb1           }
bb30       return 0;
cbb1   }
5cbe   bool spfa() {
e2f3       when++; int a = 0, b = 0, u;
91bc       for(int i = 0; i < n; i++) { d[i] = 0; qu[b++] = i; seen[i] = when; ct[i] = 0; }
9841       while(a != b) {
b492           u = qu[a++]; if(a == N) a = 0; seen[u] = 0;
d627           if(ct[u]++ >= n + 1) { a--; break; }
ccce           for(int e = hd[u]; e; e = nx[e]) if(cp[e] - fl[e] > val(0) && d[u] + cs[e] < d[to[e]] - eps) {
1ee4               d[to[e]] = d[u] + cs[e]; par[to[e]] = e ^ 1;
85b7               if(seen[to[e]] < when) { seen[to[e]] = when; qu[b++] = to[e]; if(b == N) b = 0; }
cbb1           }
cbb1       }
5c28       if(a == b) return false;
02be       val mn = numeric_limits<val>::max();
be15       when++;
ef15       for(; seen[u] != when; u = to[par[u]]) seen[u] = when;
82c8       for(int v = u; seen[v] != when + 1; v = to[par[v]]) {
6e6b           seen[v] = when + 1;
0f6a           mn = min(mn, cp[par[v] ^ 1] - fl[par[v] ^ 1]);
cbb1       }
cb73       for(int v = u; seen[v] == when + 1; v = to[par[v]]) {
7618           seen[v] = 0;
c37c           fl[par[v]] -= mn;
c905           fl[par[v] ^ 1] += mn;
cbb1       }
8a6c       return true;
cbb1   }
2b0e   val max_flow(int s, int t) {
e7a0       val fl = 0;
036d       while (bfs(s, t)) for(val f; (f = dfs(s, t, numeric_limits<val>::max())); fl += f);
e508       return fl;
cbb1   }
d41d   // public $
8662   num min_cost(int s, int t) {
94a7       flow = max_flow(s, t);
6c9f       while(spfa());
ed25       tot = 0;
112e       for(int i = 2; i < en; i++)
b951           if(fl[i] > 0)
dae8               tot += fl[i] * cs[i];
126a       return tot;
cbb1   }
8537   void reset_flow() { memset(fl, 0, sizeof(val) * en); }
457a   void add_edge(int u, int v, val c, num s) {
d321       fl[en] = 0; cp[en] = c; to[en] = v; nx[en] = hd[u]; cs[en] = s; hd[u] = en++;
```

```
f081         fl[en] = 0; cp[en] = 0; to[en] = u; nx[en] = hd[v]; cs[en] = -s; hd[v] = en++;
cbb1     }
bfc4     void init(int n) { this->n = n; en = 2; memset(hd, 0, sizeof(int) * n); } // XXX must be called
2145 };
```

5.6 Hungarian

```
d41d //const int N = ; typedef ll num; const num eps = ;
d41d // Solves minimum perfect matching in an n by n bipartite graph with edge costs in c
d41d // y and z will be such that y[i] + z[j] <= c[i][j] and sum of y and z is maximum
55ad struct hungarian {
2f6a     int n, MA[N], MB[N], PB[N], mn[N], st[N], sn; bool S[N], T[N];
6cc1     num c[N][N], d[N], y[N], z[N];
cd49     bool increase(int b) {
03dd         for (int a = PB[b];;) {
9ae2             int n_b = MA[a];
1ba8             MB[b] = a; MA[a] = b;
8f2f             if(n_b == -1) break;
5af0             b = n_b; a = PB[b];
cbb1         }
8a6c         return true;
cbb1     }
3a3b     bool visit(int a) {
cdb1         S[a] = true;
f580         for(int b = 0; b < n; b++) {
367c             if(T[b]) continue;
e782             if(c[a][b] - y[a] - z[b] < d[b] - eps) { d[b] = c[a][b] - y[a] - z[b]; mn[b] = a; }
3f25             if(c[a][b] - eps <= y[a] + z[b]) {
b46d                 T[b] = true; PB[b] = a; st[sn++] = b;
f8ab                 if(MB[b] == -1) return increase(b);
cbb1             }
cbb1         }
d1fe         return false;
cbb1     }
415c     bool update_dual() {
2f63         int mb = -1, b; num e;
f135         for(b = 0; b < n; b++) if(!T[b] && (mb == -1 || d[b] < d[mb])) mb = b;
04ff         for(e = d[mb], b = 0; b < n; b++)
3c42             if(T[b]) z[b] -= e;
6435             else d[b] -= e;
a915         for(int a = 0; a < n; a++)
cbbc             if(S[a]) y[a] += e;
eabc         PB[mb] = mn[mb];
7dcf         if(MB[mb] == -1) return increase(mb);
e309         st[sn++] = mb; T[mb] = true;
d1fe         return false;
cbb1     }
c4db     void find_path() {
2cc3         int a; for(a = 0; MA[a] != -1; a++);
0351         memset(S, 0, sizeof S); memset(T, 0, sizeof T);
e0c6         for(int i = 0; i < N; i++) d[i] = numeric_limits<num>::max();
7160         sn = 0; if(visit(a)) return;
6679         while(true) {
1f3f             if(sn) { if(visit(MB[st[--sn]])) break; }
6656             else if(update_dual()) break;
cbb1         }
cbb1     }
7e1e     void reset_all() {
52b4         for(int i = 0; i < n; i++) { y[i] = *min_element(c[i], c[i] + n); z[i] = 0; }
e517         for(int i = 0; i < n; i++) MA[i] = MB[i] = -1;
cbb1     }
d41d     // public $
957f     num min_match() { // set n and c then call this function
b989         reset_all(); num all = 0;
c13f         for(int i = 0; i < n; i++) find_path();
fe8e         for(int a = 0; a < n; a++) all += c[a][MA[a]];
64a8         return all;
cbb1     }
```

```
2145 };
```

5.7 Bridges/Cut-Vertices

```
d41d // Finds bridges and cut vertices
d41d // Receives:
d41d // N: number of vertices
d41d // l: adjacency list
d41d // Gives:
d41d // vis, seen, par (used to find cut vertices)
d41d // ap - 1 if it is a cut vertex, 0 otherwise
d41d // brid - vector of pairs containing the bridges
045d typedef pair<int, int> PII;
d41d
060e int N;
b087 vector <int> l[MAX];
d7bf vector <PII> brid;
9938 int vis[MAX], seen[MAX], par[MAX], ap[MAX];
6d2f int cnt, root;
d41d
6c18 void dfs(int x){
6229  if(vis[x] != -1)
505b    return;
7144  vis[x] = seen[x] = cnt++;
d41d
1e7d  int adj = 0;
724e  for(int i = 0; i < (int)l[x].size(); i++){
bea7    int v = l[x][i];
d1b9    if(par[x] == v)
5e2b      continue;
3c6e    if(vis[v] == -1){
062a      adj++;
20c6      par[v] = x;
6b41      dfs(v);
1214      seen[x] = min(seen[x], seen[v]);
5ec3      if(seen[v] >= vis[x] && x != root)
927e        ap[x] = 1;
d41c      if(seen[v] == vis[v])
0818        brid.push_back(make_pair(v, x));
cbb1    }
4e6b    else{
e09a      seen[x] = min(seen[x], vis[v]);
c943      seen[v] = min(seen[x], seen[v]);
cbb1    }
cbb1  }
63df  if(x == root) ap[x] = (adj>1);
cbb1 }
d41d
ece6 void bridges(){
f342  brid.clear();
faad  for(int i = 0; i < N; i++){
6939    vis[i] = seen[i] = par[i] = -1;
628e    ap[i] = 0;
cbb1  }
a018  cnt = 0;
9723  for(int i = 0; i < N; i++)
2bec    if(vis[i] == -1){
1c5a      root = i;
1e5d      dfs(i);
cbb1    }
cbb1 }
```

5.8 Strongly Connected Components

```
bf05 struct SCC {
e1bd    int V, group_cnt;
52c2    vector<vector<int> > adj, radj;
```

```
16c0     vector<int> group_num, vis;
eac6     stack<int> stk;
d41d
d41d     // V = number of vertices
8aca     SCC(int V): V(V), group_cnt(0), group_num(V), vis(V), adj(V), radj(V) {}
d41d
d41d     // Call this to add an edge (0-based)
b873     void add_edge(int v1, int v2) {
09e7        adj[v1].push_back(v2);
eb77        radj[v2].push_back(v1);
cbb1     }
d41d
f543     void fill_forward(int x) {
efc4        vis[x] = true;
4e75        for (int i = 0; i < adj[x].size(); i++) {
720f           if (!vis[adj[x][i]]) {
7306              fill_forward(adj[x][i]);
cbb1           }
cbb1        }
b527        stk.push(x);
cbb1     }
d41d
4462     void fill_backward(int x) {
b364        vis[x] = false;
4dfa        group_num[x] = group_cnt;
b46b        for (int i = 0; i < radj[x].size(); i++) {
4b86           if (vis[radj[x][i]]) {
e3ce              fill_backward(radj[x][i]);
cbb1           }
cbb1        }
cbb1     }
d41d
d41d     // Returns number of strongly connected components.
d41d     // After this is called, group_num contains component assignments (0-based)
26d6     int get_scc() {
e3e8        for (int i = 0; i < V; i++) {
4d29           if (!vis[i]) fill_forward(i);
cbb1        }
a231        group_cnt = 0;
07d1        while (!stk.empty()) {
5862           if (vis[stk.top()]) {
5c3a              fill_backward(stk.top());
0b14              group_cnt++;
cbb1           }
f3f7           stk.pop();
cbb1        }
893b        return group_cnt;
cbb1     }
2145 };
```

5.9 Stable Marraige Problem

```
d41d // Gale-Shapley algorithm for the stable marriage problem.
d41d // madj[i][j] is the jth highest ranked woman for man i.
d41d // fpref[i][j] is the rank woman i assigns to man j.
d41d // Returns a pair of vectors (mpart, fpart), where mpart[i] gives the partner of man i, and fpart is
     analogous
d474 pair<vector<int>, vector<int> > stable_marriage(vector<vector<int> >& madj, vector<vector<int> >& fpref) {
6f98    int n = madj.size();
48a7    vector<int> mpart(n, -1), fpart(n, -1);
7789    vector<int> midx(n);
c37b    queue<int> mfree;
6033    for (int i = 0; i < n; i++) {
1581       mfree.push(i);
cbb1    }
ca8c    while (!mfree.empty()) {
ac1d       int m = mfree.front(); mfree.pop();
eca5       int f = madj[m][midx[m]++];
```

```
4777        if (fpart[f] == -1) {
85a1            mpart[m] = f; fpart[f] = m;
8862        } else if (fpref[f][m] < fpref[f][fpart[f]]) {
fa31            mpart[fpart[f]] = -1; mfree.push(fpart[f]);
85a1            mpart[m] = f; fpart[f] = m;
9d97        } else {
2b67            mfree.push(m);
cbb1        }
cbb1    }
2a98    return make_pair(mpart, fpart);
cbb1 }
```

6 Structures

6.1 Ordered Set

```
7747 #include <ext/pb_ds/assoc_container.hpp>
30f4 #include <ext/pb_ds/tree_policy.hpp>
0d73 using namespace __gnu_pbds;
4519 template <typename tA, typename tB=null_type> using ord_set = tree<tA, tB, less<tA>, rb_tree_tag,
    tree_order_statistics_node_update>;
d41d // map: tA -> tB with the less<tA> comparison function
d41d // can be used as a normal map
d41d // s.find_by_order(k) :: returns iterator to the k-th element (0-indexed) (or s.end())
d41d // s.order_of_key(x) :: returns how many elements are strictly less than x
```

6.2 Treap

```
d41d //const int N = ; typedef int num;
5463 num X[N]; int en = 1, Y[N], sz[N], L[N], R[N];
8b25 void calc (int u) { // update node given children info
d4c7    sz[u] = sz[L[u]] + 1 + sz[R[u]];
d41d    // code here, no recursion
cbb1 }
234f void unlaze (int u) {
e39f    if(!u) return;
d41d    // code here, no recursion
cbb1 }
ee5e void split_val(int u, num x, int &l, int &r) { // l gets <= x, r gets > x
754f    unlaze(u); if(!u) return (void) (l = r = 0);
4bc1    if(X[u] <= x) { split_val(R[u], x, l, r); R[u] = l; l = u; }
81a7    else { split_val(L[u], x, l, r); L[u] = r; r = u; }
aaa8    calc(u);
cbb1 }
9374 void split_sz(int u, int s, int &l, int &r) { // l gets first s, r gets remaining
754f    unlaze(u); if(!u) return (void) (l = r = 0);
e06d    if(sz[L[u]] < s) { split_sz(R[u], s - sz[L[u]] - 1, l, r); R[u] = l; l = u; }
f524    else { split_sz(L[u], s, l, r); L[u] = r; r = u; }
aaa8    calc(u);
cbb1 }
c870 int merge(int l, int r) { // els on l <= els on r
67f0    unlaze(l); unlaze(r); if(!l || !r) return l + r; int u;
7801    if(Y[l] > Y[r]) { R[l] = merge(R[l], r); u = l; }
ae90    else { L[r] = merge(l, L[r]); u = r; }
0ffd    calc(u); return u;
cbb1 }
500b void init(int n=N-1) { // XXX call before using other funcs
7d1c    for(int i = en = 1; i <= n; i++) { Y[i] = i; sz[i] = 1; L[i] = R[i] = 0; }
8c5a    random_shuffle(Y + 1, Y + n + 1);
cbb1 }
```

6.3 Envelope

```
d41d // typedef ll num; const num eps = 0;
d41d // XXX double: indicates operations specific to integers, not precision related
```

```
d79f template<typename line> struct envelope {
5e0f     deque<line> q; num lo,hi; envelope (num _lo, num _hi) : lo(_lo), hi(_hi) {}
01ca     void push_front (line l) { // amort. O(inter) | l is best at lo or never
a86b         if (q.size() && q[0](lo) < l(lo)) return;
89b8         for (num x; q.size(); q.pop_front()) {
cc18             x = (q.size()<=1?hi:q[0].inter(q[1],lo,hi)-1); // XXX double (-1)
4202             if (l(x) > q[0](x)) break;
cbb1         }
45bc         q.push_front(l);
cbb1     }
f644     void push_back (line l) { // amort. O(inter) | l is best at hi or never
0334         if (q.size() && q[q.size()-1](hi) <= l(hi)) return;
b71c         for (num x; q.size(); q.pop_back()) {
4e80             x = (q.size()<=1?lo:q[q.size()-2].inter(q[q.size()-1],lo,hi));
1747             if (l(x) >= q[q.size()-1](x)) break;
cbb1         }
5e56         q.push_back(l);
cbb1     }
e732     void pop_front (num _lo) { for (lo=_lo; q.size()>1 && q[0](lo) > q[1](lo); q.pop_front()); } // amort. O(n)
218a     void pop_back (num _hi) { for (hi=_hi; q.size()>1 && q[q.size()-2](hi) <= q[q.size()-1](hi); q.pop_back()); } // amort. O(n)
7155     line get (num x) { // O(lg(R))
e32f         int lo, hi, md; for (lo = 0, hi = q.size()-1, md = (lo+hi)/2; lo < hi; md = (lo+hi)/2)
c1fb             if (q[md](x) > q[md+1](x)) { lo = md+1; }
b029             else { hi = md; }
adf9         return q[lo];
cbb1     }
2145 };
b3a6 struct line { // inter = O(1)
7bd4     num a,b; num operator () (num x) const { return a*x+b; }
2417     num inter (line o, num lo, num hi) { return abs(o.a-a)<=eps?((b<o.b)?hi+1:lo):min(hi+1,max(lo,(o.b-b-(o.b-b<0)*(a-o.a-1))/(a-o.a) + 1)); }
2145 };
16ed struct generic_line { // inter = O(lg(R))
7bd4     num a,b; num operator () (num x) const { return a*x+b; }
3cfe     num inter (generic_line o, num lo, num hi) { // first point where o strictly beats this
ca4f         for (num md = lo+((++hi)-lo)/2; lo < hi; md = lo+(hi-lo)/2) { // XXX double
760b             if ((*this)(md)<=o(md)) { lo = md+1; } // XXX double
b029             else { hi = md; }
cbb1         }
2532         return lo;
cbb1     }
2145 };
11a2 template<typename line> struct full_envelope { // XXX ties are broken arbitrarily
85c9     vector<envelope<line> > v; full_envelope(envelope<line> c) : v({c}) {} // v.reserve(30);
6aed     void add (line l) { // amort. O(lg(n)*inter)
8cca         envelope<line> cur(v.back().lo,v.back().hi); cur.push_back(l);
bb4a         while (!v.empty() && v.back().q.size() <= cur.q.size()) {
ce29             deque<line> aux; swap(aux,cur.q); int i = 0, j = 0;
31d2             for (; i < aux.size(); i++) {
542d                 for (; j < v.back().q.size() && v.back().q[j](cur.hi) > aux[i](cur.hi); j++)
0015                     cur.push_back(v.back().q[j]);
70a1                 cur.push_back(aux[i]);
cbb1             }
a0e7             for (; j < v.back().q.size(); j++) cur.push_back(v.back().q[j]);
deff             v.pop_back();
cbb1         }
026e         v.push_back(cur);
cbb1     }
7155     line get (num x) { // O(lg(n)lg(R)) | pop_back/pop_front can optimize
9351         line a = v[0].get(x);
ad67         for (int i = 1; i < (int) v.size(); i++) {
bcbe             line b = v[i].get(x);
ad0f             if (b(x)<a(x)) a = b;
cbb1         }
3f53         return a;
cbb1     }
2145 };
```

6.4 Centroid

```
0eca vector<int> adj[N]; int cn_sz[N], n;
c864 vector<int> cn_chld[N]; int cn_dep[N], cn_dist[20][N]; // removable
ace4 void cn_setdist (int u, int p, int depth, int dist) { // removable
989e     cn_dist[depth][u] = dist;
59dd     for (int v : adj[u]) if (p != v && cn_sz[v] != -1) // sz = -1 marks processed centroid (not dominated)
4ce5         cn_setdist(v, u, depth, dist+1);
cbb1 }
e897 int cn_getsz (int u, int p) {
08c9     cn_sz[u] = 1;
59dd     for (int v : adj[u]) if (p != v && cn_sz[v] != -1)
b2f6         cn_sz[u] += cn_getsz(v,u);
37a9     return cn_sz[u];
cbb1 }
912c int cn_build (int u, int depth) {
28a0     int siz = cn_getsz(u,u); int w = u;
0168     do {
9847         u = w;
a786         for (int v : adj[u]) if (cn_sz[v] != -1 && cn_sz[v] < cn_sz[u] && cn_sz[v] + cn_sz[v] >= siz)
9a13             w = v;
06ba     } while (u != w); // u becomes current centroid root
094e     cn_setdist(u,u,depth,0); // removable, here you can iterate over all dominated tree
32c2     cn_sz[u] = -1; cn_dep[u] = depth;
5cff     for (int v : adj[u]) if (cn_sz[v] != -1) {
1df5         int w = cn_build(v, depth+1);
2e31         cn_chld[u].pb(w); // removable
cbb1     }
03f4     return u;
cbb1 }
```

6.5 Link Cut Tree

```
d41d //const int N = ; typedef int num;
8db1 int en = 1, p[N], sz[N], pp[N]; bool lzswp[N];
c7d4 int C[N][2]; // {left, right} children
fc41 inline void calc(int u) { // update node given children info
5665     sz[u] = sz[C[u][0]] + 1 + sz[C[u][1]];
d41d     // code here, no recursion
cbb1 }
93d8 inline void unlaze(int u) {
e39f     if(!u) return;
a2c4     if(lzswp[u]) {
3550         swap(C[u][0], C[u][1]);
20b7         if(C[u][0]) lzswp[C[u][0]] ^= 1;
8917         if(C[u][1]) lzswp[C[u][1]] ^= 1;
53e1         lzswp[u] = 0;
cbb1     }
cbb1 }
0584 int rotate(int u, int dir) { // pulls C[u][dir] up to u and returns it
05db     int v = C[u][dir];
5b77     swap(pp[v], pp[u]);
2116     C[u][dir] = C[v][!dir];
6c8a     if(C[u][dir]) p[C[u][dir]] = u;
ed1d     C[v][!dir] = u; p[v] = p[u];
b9c1     if(p[v]) C[p[v]][C[p[v]][1] == u] = v;
6967     p[u] = v; calc(u); calc(v);
6dc7     return v;
cbb1 }
3ca5 void unlz_back(int u) { if(!u) return; unlz_back(p[u]); unlaze(u); }
81a1 void splay(int u) { // pulls node u to root
c46d     unlz_back(u);
bdd0     while(p[u]) {
2a84         int v = p[u], w = p[p[u]];
c76a         int du = (C[v][1] == u);
```

```
448e         if(!w) { rotate(v, du); assert(!p[u]); }
4e6b         else {
d499             int dv = (C[w][1] == v);
4780             if(du == dv) { rotate(w, dv); assert(C[v][du] == u); rotate(v, du); }
e576             else { rotate(v, du); assert(C[w][dv] == u); rotate(w, dv); }
cbb1         }
cbb1     }
cbb1 }
a7c2 int find_sz(int u, int s) { // returns s-th node (0-index)
d9d5     unlaze(u);
3939     while(sz[C[u][0]] != s) {
da07         if(sz[C[u][0]] < s) { s -= sz[C[u][0]] + 1; u = C[u][1]; }
afa2         else u = C[u][0];
d9d5         unlaze(u);
cbb1     }
49a4     splay(u); return u;
cbb1 }
498d int new_node() {
a2cc     int i = en++; assert(i < N);
bea5     pp[i] = C[i][0] = C[i][1] = p[i] = 0;
0db4     lzswp[i] = 0; sz[i] = 1; return i;
cbb1 }
c538 int access(int u) {
10c3     if(!u) return u;
6d13     splay(u);
f206     if(int v = C[u][1]) { p[v] = 0; pp[v] = u; C[u][1] = 0; }
aaa8     calc(u);
566b     while(pp[u]) {
0068         int w = pp[u]; splay(w);
33f4         if(int v = C[w][1]) { p[v] = 0; pp[v] = w; }
db1d         C[w][1] = u; p[u] = w; pp[u] = 0; calc(w); splay(u);
cbb1     }
03f4     return u;
cbb1 }
0782 int find_root(int u) { // root o u's tree
29bf     access(u);
3980     while(C[u][0]) { unlaze(u = C[u][0]); }
c607     access(u); return u;
cbb1 }
4d88 int get_parent(int u) { // u's parent, rootify might change it
29bf     access(u);
c6f1     if(!C[u][0]) return pp[u];
e123     unlaze(u = C[u][0]);
323c     while(C[u][1]) unlaze(u = C[u][1]);
c607     access(u); return u;
cbb1 }
c63a void link(int u, int v) { // adds edge from u to v, v must be root
961c     if(find_root(u) == find_root(v)) return;
78b9     access(u); access(v);
612a     assert(C[v][0] == 0 && pp[v] == 0 && sz[v] == 1); // v must be root
8e1a     C[u][1] = v; p[v] = u; calc(u);
cbb1 }
d41d // XXX cut + rootify require get_parent, cut unlinks u from parent, rootify makes u root
e166 void cut(int u) { access(u); assert(C[u][0]); p[C[u][0]] = 0; C[u][0] = 0; calc(u); }
1cea void rootify(int u) { access(u); lzswp[u] = 1; access(u); }
b59a void init() { en = 1; } // XXX initialize
```

6.6 Binary Indexed Tree

```
d41d // Binary indexed tree supporting binary search.
7148 struct BIT {
1a88     int n;
1160     vector<int> bit;
d41d     // BIT can be thought of as having entries f[1], ..., f[n]
d41d     // which are 0-initialized
f6ad     BIT(int n):n(n), bit(n+1) {}
d41d     // returns f[1] + ... + f[idx-1]
d41d     // precondition idx <= n+1
```

```
f3df    int read(int idx) {
a604       idx--;
11e1       int res = 0;
89ae       while (idx > 0) {
8e68          res += bit[idx];
23b5          idx -= idx & -idx;
cbb1       }
b505       return res;
cbb1    }
d41d    // returns f[idx1] + ... + f[idx2-1]
d41d    // precondition idx1 <= idx2 <= n+1
c747    int read2(int idx1, int idx2) {
a3a4       return read(idx2) - read(idx1);
cbb1    }
d41d    // adds val to f[idx]
d41d    // precondition 1 <= idx <= n (there is no element 0!)
9c0d    void update(int idx, int val) {
29b7       while (idx <= n) {
eabd          bit[idx] += val;
ff79          idx += idx & -idx;
cbb1       }
cbb1    }
d41d    // returns smallest positive idx such that read(idx) >= target
724e    int lower_bound(int target) {
6182       if (target <= 0) return 1;
c05c       int pwr = 1; while (2*pwr <= n) pwr*=2;
b1a4       int idx = 0; int tot = 0;
e438       for (; pwr; pwr >>= 1) {
b489          if (idx+pwr > n) continue;
4075          if (tot + bit[idx+pwr] < target) {
1438             tot += bit[idx+=pwr];
cbb1          }
cbb1       }
354e       return idx+2;
cbb1    }
d41d    // returns smallest positive idx such that read(idx) > target
855f    int upper_bound(int target) {
a7a2       if (target < 0) return 1;
c05c       int pwr = 1; while (2*pwr <= n) pwr*=2;
b1a4       int idx = 0; int tot = 0;
e438       for (; pwr; pwr >>= 1) {
b489          if (idx+pwr > n) continue;
b0eb          if (tot + bit[idx+pwr] <= target) {
1438             tot += bit[idx+=pwr];
cbb1          }
cbb1       }
354e       return idx+2;
cbb1    }
2145 };
```

6.7 Segment Tree

```
d41d // This is set up for range minimum queries, but can be easily adapted for computing other quantities.
d41d // To enable lazy propagation and range updates, uncomment the following line.
d41d // #define LAZY
1a25 struct Segtree {
1a88 ▹ int n;
1d95 ▹ vector<int> data;
dea8 #ifdef LAZY
8cdd #define NOLAZY 2e9
b869 #define GET(node) (lazy[node] == NOLAZY ? data[node] : lazy[node])
c68a ▹ vector<int> lazy;
8c16 #else
0458 #define GET(node) data[node]
f2ee #endif
c5b1 ▹ void build_rec(int node, int* begin, int* end) {
c22b ▹ ▹ if (end == begin+1) {
10c5 ▹ ▹ ▹ if (data.size() <= node) data.resize(node+1);
```

```
4669 			data[node] = *begin;
9d97 		} else {
8e29 			int* mid = begin + (end-begin+1)/2;
409c 			build_rec(2*node+1, begin, mid);
3780 			build_rec(2*node+2, mid, end);
807b 			data[node] = min(data[2*node+1], data[2*node+2]);
cbb1 		}
cbb1 	}
7a43 #ifndef LAZY
3167 	void update_rec(int node, int begin, int end, int pos, int val) {
c22b 		if (end == begin+1) {
a677 			data[node] = val;
9d97 		} else {
9685 			int mid = begin + (end-begin+1)/2;
6035 			if (pos < mid) {
1474 				update_rec(2*node+1, begin, mid, pos, val);
9d97 			} else {
a049 				update_rec(2*node+2, mid, end, pos, val);
cbb1 			}
807b 			data[node] = min(data[2*node+1], data[2*node+2]);
cbb1 		}
cbb1 	}
8c16 #else
dfc5 	void update_range_rec(int node, int tbegin, int tend, int abegin, int aend, int val) {
297c 		if (tbegin >= abegin && tend <= aend) {
1a83 			lazy[node] = val;
9d97 		} else {
3841 			int mid = tbegin + (tend - tbegin + 1)/2;
7ef7 			if (lazy[node] != NOLAZY) {
2d31 				lazy[2*node+1] = lazy[2*node+2] = lazy[node]; lazy[node] = NOLAZY;
cbb1 			}
ca4d 			if (mid > abegin && tbegin < aend)
a566 				update_range_rec(2*node+1, tbegin, mid, abegin, aend, val);
d7ab 			if (tend > abegin && mid < aend)
f8e6 				update_range_rec(2*node+2, mid, tend, abegin, aend, val);
ef60 			data[node] = min(GET(2*node+1), GET(2*node+2));
cbb1 		}
cbb1 	}
f2ee #endif
b241 	int query_rec(int node, int tbegin, int tend, int abegin, int aend) {
297c 		if (tbegin >= abegin && tend <= aend) {
c377 			return GET(node);
9d97 		} else {
dea8 #ifdef LAZY
7ef7 			if (lazy[node] != NOLAZY) {
04fe 				data[node] = lazy[2*node+1] = lazy[2*node+2] = lazy[node]; lazy[node] = NOLAZY;
cbb1 			}
f2ee #endif
3841 			int mid = tbegin + (tend - tbegin + 1)/2;
a3e6 			int res = INT_MAX;
ca4d 			if (mid > abegin && tbegin < aend)
52fa 				res = min(res, query_rec(2*node+1, tbegin, mid, abegin, aend));
d7ab 			if (tend > abegin && mid < aend)
1071 				res = min(res, query_rec(2*node+2, mid, tend, abegin, aend));
b505 			return res;
cbb1 		}
cbb1 	}
d41d
d41d 	// Create a segtree which stores the range [begin, end) in its bottommost level.
b5e6 	Segtree(int* begin, int* end): n(end - begin) {
db18 		build_rec(0, begin, end);
dea8 #ifdef LAZY
c57d 		lazy.assign(data.size(), NOLAZY);
f2ee #endif
cbb1 	}
d41d
7a43 #ifndef LAZY
d41d 	// Call this to update a value (indices are 0-based). If lazy propagation is enabled, use
	update_range(pos, pos+1, val) instaed.
```

```
a69b     void update(int pos, int val) {
0f2e         update_rec(0, 0, n, pos, val);
cbb1     }
8c16 #else
d41d     // Call this to update range [begin, end), if lazy propagation is enabled. Indices are 0-based.
ff71     void update_range(int begin, int end, int val) {
52d3         update_range_rec(0, 0, n, begin, end, val);
cbb1     }
f2ee #endif
d41d     // Returns minimum in range [begin, end). Indices are 0-based.
cfb8     int query(int begin, int end) {
c8b1         return query_rec(0, 0, n, begin, end);
cbb1     }
2145 };
```

7 Strings

7.1 Suffix Tree

```
4623 namespace sf {
d41d // const int NS = ; const int N = * 2;
1506 int cn, cd, ns, en = 1, lst;
f48b string S[NS]; int si = -1;
08ad vector<int> sufn[N]; // sufn[si][i] no do sufixo S[si][i...]
3c9e struct node {
a322     int l, r, si, p, suf;
d3ca     map<char, int> adj;
499b     node() : l(0), r(-1), suf(0), p(0) {}
2a9f     node(int L, int R, int S, int P) : l(L), r(R), si(S), p(P) {}
a577     inline int len() { return r - l + 1; }
48b2     inline int operator[](int i) { return S[si][l + i]; }
9eae     inline int& operator()(char c) { return adj[c]; }
fbe2 } t[N];
ea71 inline int new_node(int L, int R, int S, int P) { t[en] = node(L, R, S, P); return en++; }
e33b void add_string(string s) {
9a02     s += '$'; S[++si] = s; sufn[si].resize(s.size() + 1); cn = cd = 0;
c5eb     int i = 0; const int n = s.size();
f90a     for(int j = 0; j < n; j++)
fb3e         for(; i <= j; i++) {
8d90             if(cd == t[cn].len() && t[cn](s[j])) { cn = t[cn](s[j]); cd = 0; }
465b             if(cd < t[cn].len() && t[cn][cd] == s[j]) {
c4d2                 cd++;
ce02                 if(j < s.size() - 1) break;
4e6b                 else {
aafd                     if(i) t[lst].suf = cn;
ac68                     for(; i <= j; i++) { sufn[si][i] = cn; cn = t[cn].suf; }
cbb1                 }
7ced             } else if(cd == t[cn].len()) {
0a2a                 sufn[si][i] = en;
0467                 if(i) t[lst].suf = en; lst = en;
aff4                 t[cn](s[j]) = new_node(j, n - 1, si, cn);
02c2                 cn = t[cn].suf; cd = t[cn].len();
9d97             } else {
f287                 int mid = new_node(t[cn].l, t[cn].l + cd - 1, t[cn].si, t[cn].p);
12ed                 t[t[cn].p](t[cn][0]) = mid;
5201                 if(ns) t[ns].suf = mid;
0467                 if(i) t[lst].suf = en; lst = en;
0a2a                 sufn[si][i] = en;
cb00                 t[mid](s[j]) = new_node(j, n - 1, si, mid);
7bfa                 t[mid](t[cn][cd]) = cn;
07fe                 t[cn].p = mid; t[cn].l += cd; cn = t[mid].p;
5967                 int g = cn? j - cd : i + 1; cn = t[cn].suf;
c197                 while(g < j && g + t[t[cn](S[si][g])].len() <= j) {
6fea                     cn = t[cn](S[si][g]); g += t[cn].len();
cbb1                 }
71c3                 if(g == j) { ns = 0; t[mid].suf = cn; cd = t[cn].len(); }
f90d                 else { ns = mid; cn = t[cn](S[si][g]); cd = j - g; }
```

```
cbb1 			}
cbb1 		}
cbb1 	}
2145 };
```

7.2 Z-function

```
2a61 void Z(char s[], int n, int z[]) { // z[i] = |lcp(s,s[i..n])|
fc15 	for(int i = 1, m = -1; i < n; i++) {
d69b 		z[i] = (m != -1 && m + z[m] >= i)?min(m + z[m] - i, z[i - m]):0;
8a63 		while (i + z[i] < n && s[i + z[i]] == s[z[i]]) z[i]++;
bbe8 		if (m == -1 || i + z[i] > m + z[m]) m = i;
cbb1 	}
cbb1 }
```

8 Math

8.1 Linear System Solver

```
d41d //const int N = ;
d41d
46cc double a[N][N];
3793 double ans[N];
d41d
d41d // sum(a[i][j] * x_j) = a[i][n] for 0 <= i < n
d41d // stores answer in ans and returns det(a)
c42a double solve(int n) {
f99b 	double det = 1;
6033 	for(int i = 0; i < n; i++) {
0268 		int mx = i;
197a 		for(int j = i + 1; j < n; j++)
b83d 			if(abs(a[j][i]) > abs(a[mx][i]))
672f 				mx = j;
28c6 		if(i != mx) {
e83f 			swap_ranges(a[i], a[i] + n + 1, a[mx]);
0143 			det = -det;
cbb1 		}
997e 		if(abs(a[i][i]) < 1e-6); // singular matrix
2f40 		det *= a[i][i];
94fe 		for(int j = i + 1; j < n; j++) {
12fe 			for(int k = i + 1; k <= n; k++)
ea32 				a[j][k] -= (a[j][i] / a[i][i]) * a[i][k];
efbc 			a[j][i] = 0;
cbb1 		}
cbb1 	}
45bd 	for(int i = n - 1; i >= 0; i--) {
7634 		ans[i] = a[i][n];
197a 		for(int j = i + 1; j < n; j++)
9b00 			ans[i] -= a[i][j] * ans[j];
35e5 		ans[i] /= a[i][i];
cbb1 	}
7a32 	return det;
cbb1 }
```

8.2 Simplex

```
d41d //typedef long double dbl;
bec0 const dbl eps = 1e-6;
d41d //const int N = , M = ;
d41d
79ee struct simplex {
0643 	int X[N], Y[M];
6b50 	dbl A[M][N], b[M], c[N];
e268 	dbl ans;
14e0 	int n, m;
```

```
a00d ▷ dbl sol[N];
d41d
c511 ▷ void pivot(int x,int y){
eb91 ▷ ▷ swap(X[y], Y[x]);
c057 ▷ ▷ b[x] /= A[x][y];
8300 ▷ ▷ for(int i = 0; i < n; i++)
7f61 ▷ ▷ ▷ if(i != y)
d311 ▷ ▷ ▷ ▷ A[x][i] /= A[x][y];
3fa2 ▷ ▷ A[x][y] = 1. / A[x][y];
94f7 ▷ ▷ for(int i = 0; i < m; i++)
a325 ▷ ▷ ▷ if(i != x && abs(A[i][y]) > eps) {
6856 ▷ ▷ ▷ ▷ b[i] -= A[i][y] * b[x];
f90a ▷ ▷ ▷ ▷ for(int j = 0; j < n; j++)
6739 ▷ ▷ ▷ ▷ ▷ if(j != y)
8c78 ▷ ▷ ▷ ▷ ▷ ▷ A[i][j] -= A[i][y] * A[x][j];
e112 ▷ ▷ ▷ ▷ A[i][y] = -A[i][y] * A[x][y];
cbb1 ▷ ▷ ▷ }
8c7e ▷ ▷ ans += c[y] * b[x];
8300 ▷ ▷ for(int i = 0; i < n; i++)
7f61 ▷ ▷ ▷ if(i != y)
bec1 ▷ ▷ ▷ ▷ c[i] -= c[y] * A[x][i];
0997 ▷ ▷ c[y] = -c[y] * A[x][y];
cbb1 ▷ }
d41d
d41d ▷ // maximize sum(x[i] * c[i])
d41d ▷ // element a
d41d ▷ //   sum(a[i][j] * x[j]) <= b[i] for 0 <= i < m (Ax <= b)
d41d ▷ //   x[i] >= 0 for 0 <= i < n (x >= 0)
d41d ▷ // (n variables, m constraints)
d41d ▷ // stores the answer in ans and returns optimal value
59d9 ▷ dbl solve(int n, int m) {
1f59 ▷ ▷ this->n = n; this->m = m;
f1bf ▷ ▷ ans = 0.;
b1c6 ▷ ▷ for(int i = 0; i < n; i++) X[i] = i;
3e36 ▷ ▷ for(int i = 0; i < m; i++) Y[i] = i + n;
6679 ▷ ▷ while(true) {
ee39 ▷ ▷ ▷ int x = min_element(b, b + m) - b;
988b ▷ ▷ ▷ if(b[x] >= -eps)
c2be ▷ ▷ ▷ ▷ break;
49a2 ▷ ▷ ▷ int y = find_if(A[x], A[x] + n, [](dbl d) { return d < -eps; }) - A[x];
6f8c ▷ ▷ ▷ if(y == n) throw 1; // no solution
7fb4 ▷ ▷ ▷ pivot(x, y);
cbb1 ▷ ▷ }
6679 ▷ ▷ while(true) {
f802 ▷ ▷ ▷ int y = max_element(c, c + n) - c;
b7b6 ▷ ▷ ▷ if(c[y] <= eps) break;
d6b5 ▷ ▷ ▷ int x = -1;
06d7 ▷ ▷ ▷ dbl mn = 1. / 0.;
94f7 ▷ ▷ ▷ for(int i = 0; i < m; i++)
5877 ▷ ▷ ▷ ▷ if(A[i][y] > eps && b[i] / A[i][y] < mn)
832b ▷ ▷ ▷ ▷ ▷ mn = b[i] / A[i][y], x = i;
ff22 ▷ ▷ ▷ if(x == -1) throw 2; // unbounded
7fb4 ▷ ▷ ▷ pivot(x, y);
cbb1 ▷ ▷ }
d094 ▷ ▷ memset(sol, 0, sizeof(dbl) * n);
94f7 ▷ ▷ for(int i = 0; i < m; i++)
cff4 ▷ ▷ ▷ if(Y[i] < n)
09d7 ▷ ▷ ▷ ▷ sol[Y[i]] = b[i];
ba75 ▷ ▷ return ans;
cbb1 ▷ }
2145 };
```

9 Number Theory

9.1 Extended Euclidean Algorithm

```
c25f int egcd(int a, int b, int& x, int& y) { // a*x + b*y = gcd(a, b) [Bezout's Theorem]
```

```
8273 ▹ if (b == 0) return x = 1, y = 0, a;
98d1 ▹ int xx, yy;
0c0d ▹ int g = egcd(b, a % b, xx, yy);
512d ▹ x = yy;
a9d0 ▹ y = xx - (a / b) * yy;
96b5 ▹ return g;
cbb1 }
```

9.2 Miller-Rabin

```
a288 llu llrand() { llu a = rand(); a<<= 32; a+= rand(); return a;}
0a9c int is_probably_prime(llu n) {
8dbf    if (n <= 1) return 0;
2373    if (n <= 3) return 1;
7de1    llu s = 0, d = n - 1;
66b4    while (d % 2 == 0) {
90f4       d/= 2; s++;
cbb1    }
6b3a    for (int k = 0; k < 64; k++) {
12c0       llu a = (llrand() % (n - 3)) + 2;
dc17       llu x = exp_mod(a, d, n);
1181       if (x != 1 && x != n-1) {
f0ea          for (int r = 1; r < s; r++) {
708d             x = mul_mod(x, x, n);
61d9             if (x == 1)
bb30                return 0;
68b2             if (x == n-1)
c2be                break;
cbb1          }
34bc          if (x != n-1)
bb30             return 0;
cbb1       }
cbb1    }
6a55    return 1;
cbb1 }
```

9.3 Diofantine

```
d41d // find all solutions in the form ax + by = c
d41d
080f void shift_solution(int & x, int & y, int a, int b, int cnt) {
526a    x += cnt * b;
fdfb    y -= cnt * a;
cbb1 }
d41d
f0f5 int find_all_solutions(int a, int b, int c, int minx, int maxx, int miny, int maxy) {
ede5    int x, y, g;
7968    if (!find_any_solution(a, b, c, x, y, g))
bb30       return 0;
fe72    a /= g;
ee2d    b /= g;
d41d
0750    int sign_a = a > 0 ? +1 : -1;
f8be    int sign_b = b > 0 ? +1 : -1;
d41d
ab53    shift_solution(x, y, a, b, (minx - x) / b);
5969    if (x < minx)
8a96       shift_solution(x, y, a, b, sign_b);
6bcc    if (x > maxx)
bb30       return 0;
57f8    int lx1 = x;
d41d
9870    shift_solution(x, y, a, b, (maxx - x) / b);
6bcc    if (x > maxx)
f6f8       shift_solution(x, y, a, b, -sign_b);
eb5e    int rx1 = x;
d41d
```

```
7672    shift_solution(x, y, a, b, -(miny - y) / a);
a697    if (y < miny)
bf53       shift_solution(x, y, a, b, -sign_a);
a1de    if (y > maxy)
bb30       return 0;
8e42    int lx2 = x;
d41d
e322    shift_solution(x, y, a, b, -(maxy - y) / a);
a1de    if (y > maxy)
b156       shift_solution(x, y, a, b, sign_a);
481c    int rx2 = x;
d41d
473e    if (lx2 > rx2)
e723       swap(lx2, rx2);
2b9f    int lx = max(lx1, lx2);
037c    int rx = min(rx1, rx2);
d41d
f0c5    if (lx > rx)
bb30       return 0;
ebb8    return (rx - lx) / abs(b) + 1;
cbb1 }
```

10 Notes

10.1 Modular Multiplicative Inverse

- If $gcd(a, m) = 1$, then let $ax + my = gcd(a, m) = 1$ (Bezout's Theorem). Then $ax \equiv 1 \pmod{m}$.
- If $gcd(a, m) = 1$, then $a \cdot a^{\phi(m)-1} \equiv 1 \pmod{m}$ (Euler's Theorem).
- If m is prime, then $\phi(m) = m - 1$, so $a * a^{m-2} \equiv 1 \pmod{m}$.

10.2 Chinese Remainder Theorem

We are given $N = n_1 n_2 \cdots n_k$ where n_i are pairwise coprime. We are also given $x_1 \cdots x_k$ such that $x \equiv x_i \pmod{n_i}$. Let $N_i = N/n_i$. There exists M_i and m_i such that $M_i N_i + m_i n_i = 1$ (Bezout). Then, there is only one solution x, given by: $x = \sum_{i=1}^{k} a_i M_i N_i$

10.3 Euler's Totient Function

Positive integers up to a given integer n that are relatively prime to n. $\varphi(n) = n \prod_{p|n} \left(1 - \frac{1}{p}\right)$ where the product is over the distinct prime numbers dividing n.

10.4 Möebius

If $F(n) = \sum_{d|n} f(d)$, then $f(n) = \sum_{d|n} \mu(d) F(n/d)$.

10.5 Burnside

Let $A\colon GX \to X$ be an action. Define:

- $w :=$ number of orbits in X.
- $S_x := \{g \in G \mid g \cdot x = x\}$
- $F_g := \{x \in X \mid g \cdot x = x\}$

Then $w = \frac{1}{|G|} \sum_{x \in X} |S_x| = \frac{1}{|G|} \sum_{g \in G} |F_g|$.

10.6 Catalan Number

C_n is solution for:

- Number of correct bracket sequence consisting of n opening and n closing brackets.
- The number of rooted full binary trees with $n+1$ leaves (vertices are not numbered). A rooted binary tree is full if every vertex has either two children or no children.
- The number of ways to completely parenthesize $n+1$ factors.
- The number of triangulations of a convex polygon with $n+2$ sides (i.e. the number of partitions of polygon into disjoint triangles by using the diagonals).
- The number of ways to connect the $2n$ points on a circle to form n disjoint chords.
- The number of non-isomorphic full binary trees with n internal nodes (i.e. nodes having at least one son).
- The number of monotonic lattice paths from point $(0,0)$ to point (n,n) in a square lattice of size $n \times n$, which do not pass above the main diagonal (i.e. connecting $(0,0)$ to (n,n)).
- Number of permutations of length n that can be stack sorted (i.e. it can be shown that the rearrangement is stack sorted if and only if there is no such index $i < j < k$, such that $a_k < a_i < a_j$).
- The number of non-crossing partitions of a set of n elements.
- The number of ways to cover the ladder $1 \dots n$ using n rectangles (The ladder consists of n columns, where i^{th} column has a height i).

Recursive:

$$C_0 = C_1 = 1$$

$$C_n = \sum_{k=0}^{n-1} C_k C_{n-1-k}, n \geq 2$$

Analytical:

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

10.7 Landau

There is a tournament with outdegrees $d_1 \leq d_2 \leq \ldots \leq d_n$ iff:

- $d_1 + d_2 + \ldots + d_n = \binom{n}{2}$
- $d_1 + d_2 + \ldots + d_k \geq \binom{k}{2} \quad \forall 1 \leq k \leq n.$

In order to build it, let 1 point to $2, 3, \ldots, d_1 + 1$ and repeat recursively.

10.8 Erdös-Gallai

There is a simple graph with degrees $d_1 \geq d_2 \geq \ldots \geq d_n$ iff:

- $d_1 + d_2 + \ldots + d_n$ is even
- $\sum_{i=1}^{k} d_i \leq k(k-1) + \sum_{i=k+1}^{n} \min(d_i, k) \quad \forall 1 \leq k \leq n.$

In order to build it, connect 1 with $2, 3, \ldots, d_1 + 1$ and repeat recursively.

10.9 Gambler's Ruin

In a game in which we win a coin with probability p and lose a coin with probability $q := 1 - p$, the game stops when we win B ou lose A coins. Then $Prob(\text{win B}) = \frac{1-(p/q)^B}{1-(p/q)^{A+B}}$.

10.10 Extra

- $Fib(x+y) = Fib(x+1)Fib(y) + Fib(x)Fib(y-1)$