**Disjoint Union Find**

void initialize( int Arr[ ], int N){

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

Arr[ i ] = i ;

size[ i ] = 1;

}

}

int root(int Arr[ ],int i){

while(Arr[ i ] != i)

i = Arr[ i ];

return i;

}

bool isTheSameSet(int A,int B){

if( root(A)==root(B) )

return true;

else

return false;

}

void union(int Arr[ ],int size[ ],int A,int B){

int root\_A = root(A);

int root\_B = root(B);

if(size[root\_A] < size[root\_B ]){

Arr[ root\_A ] = Arr[root\_B];

size[root\_B] += size[root\_A];

}

else{

Arr[ root\_B ] = Arr[root\_A];

size[root\_A] += size[root\_B];

}

}

**Fill and setfill**

fill(adj, adj+n, value); // for 1 dimensional array

for(auto i: adj)

fill(i, i + n, value); // for 2 dimensional array

\*/

cout<<setfill(0)<<setw(10)<<55;//000000000055

cout<<setbase(16)<<255;// ff only for base 8 and 16

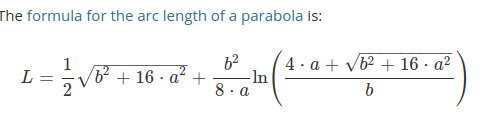
**maths**

log(x), log10(x);

cout<<setprecision(p)<<fixed<<val;// for p decimal point after point

cout<<setprecision(p)<<val;// for p decimal place

b-> horizontal distance, and a-> is vertical distance



bit operator

unsigned char a = 5, b = 9; // a = 5(00000101), b = 9(00001001)

printf("a = %d, b = %d\n", a, b);

printf("a&b = %d\n", a&b); // The result is 00000001

printf("a|b = %d\n", a|b);// The result is 00001101

printf("a^b = %d\n", a^b); // The result is 00001100

printf("~a = %d\n", a = ~a);// The result is 11111010

printf("b<<1 = %d\n", b<<1);// The result is 00010010

printf("b>>1 = %d\n", b>>1);// The result is 00000100

ios\_base::sync\_with\_stdio(false);

cin.tie(0);

\_\_gcd(x,y);

transform(str.begin(), str.end(), str.begin(), ::tolower);

**datastructure**

**vector**

vector<int> v;

v.push\_back(3);

for (auto x : v) {

cout << x << "\n";

}

v.pop\_back();

cout << v.back() << "\n";

vector<int> v = {2,4,2,5,1};

vector<int> v(10);// size 10, initial value 0

vector<int> v(10, 5);// size 10, initial value 5

sort(v.begin(), v.end());

reverse(v.begin(), v.end());

random\_shuffle(v.begin(), v.end());

find (vec.begin(), vec.end(), value);

**array**

sort(a, a+n);

reverse(a, a+n);

random\_shuffle(a, a+n);

**string**

string a = "hattivatti";

string c = b.substr(3,4);//tiva

int d = b.find(‘t’); // 2

**set**

set<int> s;

s.insert(3);

s.erase(3);

set<int> s = {2,5,6,8};

set<int>::iterator it = s.begin();

auto it = s.begin();

cout << \*it << "\n";

for (auto it = s.begin(); it != s.end(); it++)

cout << \*it << "\n";

auto it = s.end(); it--;//last element iterator

auto it = s.find(x);//returns an iterator

if (it == s.end())

// x is not found

**multiset**

multiset<int> s;

s.insert(5);

s.insert(5);

s.insert(5);

cout << s.count(5) << "\n"; // 3

s.erase(5);

cout << s.count(5) << "\n"; // 0

s.erase(s.find(5));

cout << s.count(5) << "\n"; // 2

**map**

map<string,int> m;

m["monkey"] = 4;

map<string,int> m;

cout << m["aybabtu"] << "\n"; // 0

for (auto x : m)

cout << x.first << " " << x.second << "\n";

\*\* accessing element of set, map, multiset & multimap is log(n)

\*\* accessing element of unordered\_set, unordered\_map, unordered\_multiset & unordered\_multimap is log(n)

**bitset**

bitset<10> s;

s[1] = 1;

bitset<10> s(string("0010011010")); // from right to left

cout << s.count() << "\n"; // 4 , returns the number of ones in the bitset:

bitset<10> a(string("0010110110"));

bitset<10> b(string("1011011000"));

cout << (a&b) << "\n"; // 0010010000

cout << (a|b) << "\n"; // 1011111110

cout << (a^b) << "\n"; // 1001101110

**deque**

deque<int> d;

d.push\_back(5); // [5]

d.push\_back(2); // [5,2]

d.push\_front(3); // [3,5,2]

d.pop\_back(); // [3,5]

d.pop\_front(); // [5]

**common functions for all datastructures**

x.size();

x.erase(value); x.erase(pointer);

x.erase(initial\_pointer, final\_pointer);

x.count(v);

x.clear();

find(initial\_pointer, final\_pointer, value);

lower\_bound ( x ) returns an iterator to the smallest element in the

set whose value is at least x , and the function upper\_bound ( x ) returns an iterator to the smallest element in the set whose value is larger than x .

**priority\_queue**

Insertion and removal take O (log n ) time, and retrieval takes O (1) time.

priority\_queue<int> q;

q.push(3);

q.push(5);

q.push(7);

q.push(2);

cout << q.top() << "\n"; // 7

q.pop();

cout << q.top() << "\n"; // 5

q.pop();

q.push(6);

cout << q.top() << "\n"; // 6

q.pop();

priority\_queue<int,vector<int>,greater<int>> q;//that supports finding and removing the smallest element

// count number bit 1’s in a number

int countSetBits(long long n)

{

unsigned int count = 0;

while (n)

{

count += n & 1;

n >>= 1;// n= n>>1;

}

return count;

}

\*\* number of odd elemnts in a pascal triangle at nth column is

powl(2, x); where x = countSetBits(n)

if (b&(1<<i)) // check whether the nth bit is 0 or 1

vector<int> permutation;

for (int i = 0; i < n; i++) {

permutation.push\_back(i);

}

do {

// process permutation

} while (next\_permutation(permutation.begin(),permutation.end()));

**n quens**

void search(int y) {

if (y == n) {

count++;

return;

}

for (int x = 0; x < n; x++) {

if (column[x] || diag1[x+y] || diag2[x-y+n-1]) continue;

column[x] = diag1[x+y] = diag2[x-y+n-1] = 1;

search(y+1);

column[x] = diag1[x+y] = diag2[x-y+n-1] = 0;

}

}

int d, x, y;

void extendedEuclid(int A, int B) {

if(B == 0) {

d = A;

x = 1;

y = 0;

}

else {

extendedEuclid(B, A%B);

int temp = x;

x = y;

y = temp - (A/B)\*y;

}

}

int modInverse(int A,int M)

{

A=A%M;

for(int B=1;B<M;B++)

if((A\*B)%M)==1)

return B;

}

int d,x,y;

int modInverse(int A, int M)

{

extendedEuclid(A,M);

return (x%M+M)%M; //x may be negative

}

//O(log(max(A,M)))

**used only when M is prime**

int modInverse(int A,int M)

{

return modularExponentiation(A,M-2,M);

}

