1 STL Useful Tips
1.1 Common libraries
1.2 I/O
1.3 Useful constant
1.4 Space waster
1.5 Tricks in cmath
1.6 Initialize array with predefined value
1.7 Modifying sequence operations
1.8 Merge
1.9 String
1.10 Heap
1.11 Sort
1.12 Permutations
1.13 Searching
1.14 Random algorithm
2 Number Theory
2.1 Prime number under 100
2.5 If prime number
2.6 Prime factorization
2.7 Leap year
2.8 Binary exponential
2.10 Factorial mod
2.11 Generate combinations
2.12 10-ary to m-ary
2.13 m-ary to 10-ary
2.14 Binomial coefficient
2.15 Catalan numbers
2.16 Eulerian numbers
2.17 Karatsuba algorithm in Java
2.18 Euler's totient function
2.19 Split plane
3 Searching Algorithms
3.1 Find rank k in array
3.2 KMP Algorithm
4 Dynamic Programming
4.1 0/1 Knapsack problems
4.2 Complete Knapsack problems
4.3 Longest common subsequence (LCS)
4.4 Longest increasing common sequence (LICS)
4.5 Longest Increasing Subsequence (LIS) 4.6 Maximum submatrix
4.7 Partitions of integers
4.8 Partitions of sets
5 Trees
5.1 Tree traversal
5.2 Depth and width of tree
6 Graph Theory 19
6.1 Graph representation
6.2 Flood fill algorithm
6.3 SPFA — shortest path
6.4 Floyd-Warshall algorithm – shortest path of all pairs
6.5 Prim — minimum spanning tree
6.6 Eulerian circuit
6.7 Topological sort
7 Individual Templates

1 STL Useful Tips

1.1 Common libraries

```
/*** Functions ***/
#include<algorithm>
#include<functional> // for hash
#include<climits> // all useful constants
#include<cmath>
#include<cstdio>

#include<cstdlib> // random
#include<ctime>
#include<iostream>
#include<sstream>
#include<iomanip> // right justifying
std::right and std::setw(width)
/*** Data Structure ***/
#include<deque> // double ended queue
#include<list>
#include<queue> // including priority_queue
#include<stack>
#include<string>
#include<vector>
```

1.2 I/O

```
// iostream and cstdio are both using I/O
streams
// However, they have different behavior,
// pay attention on them if you're using
them together.
// cin does not concern with '\n' at end of
each line
// however scanf or getline does concern
with '\n' at end of each line
// '\n' will be ignored when you use cin to
read char.
// when you use getline(cin, str) to read a
whole line of input
// please add an extra getline before
inputing if previous inputs are numbers
cin >> n;
getline(cin, str) // wasted getline
getline(cin, str) // real input string
```

1.3 Useful constant

```
INT_MIN
INT_MAX
LONG_MIN
LONG_MAX
LLONG_MIN
LLONG_MAX
(~0u) // infinity (for long and long long)
// use (~0u)>>2 for int.
```

1.4 Space waster

```
// consider to redefine data types to void
data range problem
#define int long long // make everyone long
long
#define double long double // make everyone
long double
// function definitions
#undef int // main must return int
int main(void)
#define int long long // redefine int
```

```
// rest of program
```

1.5 Tricks in cmath

```
// when the number is too large. use powl
instead of pow.
// will provide you more accuracy.
powl(a, b)
(int)round(p, (1.0/n)) // nth root of p
```

1.6 Initialize array with predefined value

```
// for 1d array, use STL fill_n or fill to
initialize array
fill(a, a+size_of_a, value) fill(arr,arr+10,4);
fill_n(a, size_of_a, value)
// for 2d array, if want to fill in 0 or -1
memset(a, 0, sizeof(a));
// otherwise, use a loop of fill or fill_n
through every a[i]
fill(a[i], a[i]+size_of_ai, value) // from 0
to number of row.
```

1.7 Modifying sequence operations

```
void copy(first, last, result);
void swap(a,b);
void swap(first1, last1, first2); // swap
range
void replace(first, last, old_value,
new_value); // replace in range
void replace_if(first, last, pred,
new_value); // replace in conditions
// pred can be represented in function
// e.x. bool IsOdd (int i) { return
((i%2)==1); }
void reverse(first, last); // reverse a
range of elements
void reverse_copy(first, last, result); //
copy a reverse of range of elements
void random_shuffle(first, last); // using
built-in random generator to shuffle array
```

1.8 Merge

```
// merge sorted ranges
void merge(first1, last1, first2, last2,
result, comp);
// union of two sorted ranges
void set_union(first1, last1, first2, last2,
result, comp);
// intersection of two sorted ranges
void set_interaction(first1, last1, first2,
last2, result, comp);
// difference of two sorted ranges
void set_difference((first1, last1, first2,
last2, result, comp);
```

1.9 String

```
// Searching
unsigned int find(const string &s2, unsigned
int pos1 = 0);
unsigned int rfind(const string &s2,
unsigned int pos1 = end);
unsigned int find_first_of(const string &s2,
unsigned int pos1 = 0);
unsigned int find_last_of(const string &s2,
unsigned int pos1 = end);
unsigned int find_first_not_of(const string
&s2, unsigned int pos1 = 0);
unsigned int find_last_not_of(const string
&s2, unsigned int pos1 = end);
// Insert, Erase, Replace
string& insert(unsigned int pos1, const
string &s2);
string& insert(unsigned int pos1, unsigned
int repetitions, char c);
string& erase(unsigned int pos = 0, unsigned
int len = npos);
string& replace(unsigned int pos1, unsigned
int len1, const string &s2);
string& replace(unsigned int pos1, unsigned
int len1, unsigned int repetitions, char c);
// String streams
stringstream s1;
int i = 22;
s1 << "Hello world! " << i;
cout << s1.str() << endl;
```

1.10 Heap

```
template <class RandomAccessIterator>
void push_heap (RandomAccessIterator first,
RandomAccessIterator last);
template <class RandomAccessIterator, class
Compare>
void push_heap (RandomAccessIterator first,
RandomAccessIterator last,
                                    Compare
comp);
                    template <class
RandomAccessIterator>
                    void pop_heap
(RandomAccessIterator first,
RandomAccessIterator last);
                    template <class
RandomAccessIterator, class Compare>
                    void pop_heap
(RandomAccessIterator first,
RandomAccessIterator last,
                                   Compare
comp);
                    template <class
RandomAccessIterator>
                    void make_heap
(RandomAccessIterator first,
RandomAccessIterator last);
                    template <class
RandomAccessIterator, class Compare>
                    void make_heap
(RandomAccessIterator first,
RandomAccessIterator last,
                                    Compare
comp );
                    template <class
RandomAccessIterator>
                    void sort_heap
(RandomAccessIterator first,
RandomAccessIterator last);
                    template <class
RandomAccessIterator, class Compare>
```

```
                    void sort_heap
(RandomAccessIterator first,
RandomAccessIterator last,
                                    Compare
comp);
                    template <class
RandomAccessIterator>
                    RandomAccessIterator
is_heap_until (RandomAccessIterator first,
                            RandomAccessIter
ator last);
                    template <class
RandomAccessIterator, class Compare>
                    RandomAccessIterator
is_heap_until (RandomAccessIterator first,
                            RandomAccessIter
ator last
                            Compare comp);
```

1.11 Sort

```
                    void sort(iterator
first, iterator last);
                    void sort(iterator
first, iterator last, LessThanFunction
comp);
                    void
stable_sort(iterator first, iterator last);
                    void
stable_sort(iterator first, iterator last,
LessThanFunction comp);
                    void
partial_sort(iterator first, iterator
middle, iterator last);
                    void
partial_sort(iterator first, iterator
middle, iterator last, LessThanFunction
comp);
                    bool is_sorted(iterator
first, iterator last);
                    bool is_sorted(iterator
first, iterator last,
LessThanOrEqualFunction comp);
// example for sort, if have array x,
start_index, end_index;
                    sort(x+start_index,
x+end_index);
                    /** sort a map **/
// You cannot directly sort a map<key type,
mapped data type>
// if you only want to sort in key type
// you can use insert method to copy map
into another map
// b.insert(make_pair(it->first, it->second)
/* it is a map iterator */
// this will result a map which sorts key
type in increasing order
// if you want to sort key type in
decreasing order, then declare your map as
// something like:
// map<char, int, greater<char> >
// if you want to sort based on key, you
need to copy the data to a vector
// where elements of vector are pair.
// you can define a PAIR type by using:
                    typedef pair<char, int>
PAIR;
// suppose this is the map
                    map<char, int> a;
// sort vector in decreasing order
                    bool cmp_by_value(const
PAIR& lhs, const PAIR& rhs)
{
    return lhs.second > rhs.second;
}
// sort key in increasing order
bool cmp_by_char(const PAIR& lhs, const
PAIR& rhs)
{
    return lhs.first < rhs.first;
}
// copy map data to vector
vector<PAIR> b(a.begin(), a.end());
// sort data
sort(b.begin(), b.end(), cmp_by_value);
// you can still call your data by
b[i].first and b[i].second.
// THE ABOVE CODES ARE EXAMPLE FOR SORTING A
MAP.
// PLEASE USE IT FOR YOUR OWN DEMANDS.
```

1.12 Permutations

```
bool next_permutation(iterator first,
iterator last);
bool next_permutation(iterator first,
iterator last, LessThanOrEqualFunction
comp);
bool prev_permutation(iterator first,
iterator last);
bool prev_permutation(iterator first,
iterator last, LessThanOrEqualFunction
comp);
```

1.13 Searching

```
// will return address of iterator, call
result as *iterator;
iterator find(iterator first, iterator last,
const T &value);
iterator find_if(iterator first, iterator
last, const T &value, TestFunction test);
bool binary_search(iterator first, iterator
last, const T &value);
bool binary_search(iterator first, iterator
last, const T &value,
LessThanOrEqualFunction comp);
```

1.14 Random algorithm

```
srand(time(NULL));
// generate random numbers between [a,b)
rand() % (b - a) + a;
// generate random numbers between [0,b)
rand() % b;
// generate random permutations
random_permutation(anArray, anArray + 10);
random_permutation(aVector, aVector + 10);
```

2 Number Theory

2.1 Prime number under 100

```
// there are 25 numbers
2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37,
41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83,
89, 97
```

2.5 If prime number

```
bool prime(int n)
{
    if (n<2) return false;
    if (n<=3) return true;
    if (!(n%2) || !(n%3)) return false;
    for (int i=5; i*i<=n; i+=6)
        if (!(n%i) || !(n%(i+2))) return
false;
    return true;
}
```

2.6 Prime factorization

```
// smallest prime factor of a number.
function factor(int n)
{
    int a;
    if (n%2==0)
        return 2;
    for (a=3; a<=sqrt(n); a++++)
    {
        if (n%a==0)
            return a;
    }
    return n;
}
// complete factorization
int r;
while (n>1)
{
r = factor(n);
    printf("%d", r);
    n /= r;
}
```

2.7 Leap year

```
bool isLeap(int n)
{
    if (n%100==0)
        if (n%400==0) return true;
        else return false;
    if (n%4==0) return true;
    else return false;
}
```

2.8 Binary exponiential

```
int binExpIte(int a,int b){
    int ans=1;
    while(b){
        if(b&1){
            ans*=a;
        }
        a*=a;
        b>>=1;
    }
    return ans;
}
```

2.9 a^b mod p

```
long powmod(long base, long exp, long
modulus)
{
    base %= modulus;
    long result = 1;
    while (exp > 0)
    {
        if (exp & 1) result = (result *
base) % modulus;
        base = (base * base) % modulus;
        exp >>= 1;
    }
    return result;
}
```

2.10 Factorial mod

//n! mod p

```
int factmod (int n, int p)
{
    long long res = 1;
    while (n > 1)
    {
        res = (res * powmod (p-1, n/p, p)) %
p;
        for (int i=2; i<=n%p; ++i)
            res=(res*i) %p;
        n /= p;
    }
    return int (res % p);
}
```

2.11 Generate combinations

```
// n>=m, choose M numbers from 1 to N.
void combination(int n, int m)
{
    if (n<m) return ;
    int a[50]= {0};
    int k=0;
    for (int i=1; i<=m; i++) a[i]=i;
    while (true)
    {
        for (int i=1; i<=m; i++)
            cout << a[i] << " ";
        cout << endl;
        k=m;
        while ((k>0) && (n-a[k]==m-k)) k--;
        if (k==0) break;
        a[k]++;
        for (int i=k+1; i<=m; i++)
            a[i]=a[i-1]+1;
    }
}
```

2.12 10-ary to m-ary

```
char a[16]=
{’0’,’1’,’2’,’3’,’4’,’5’,’6’,’7’,’8’,’9’,
             ’A’,’B’,’C’,’D’,’E’,’F’
            };
            string tenToM(int n, int m)
{
    int temp=n;
    string result="";
    while (temp!=0)
    {
        result=a[temp%m]+result;
        temp/=m;
    }
    return result;
}
```

2.13 m-ary to 10-ary

```
string num="0123456789ABCDE";
           int mToTen(string n, int m)
{
    int multi=1;
    int result=0;
    for (int i=n.size()-1; i>=0; i--)
    {
        result+=num.find(n[i])*multi;
        multi*=m;
    }
    return result;
}
```

2.14 Binomial coefficient

```
#define MAXN 100 // largest n or m
long binomial_coefficient(n,m) // compute n
choose m
int n,m;
{
    int i,j;
    long bc[MAXN][MAXN];
    for (i=0; i<=n; i++) bc[i][0] = 1;
    for (j=0; j<=n; j++) bc[j][j] = 1;
    for (i=1; i<=n; i++)
        for (j=1; j<i; j++)
            bc[i][j] = bc[i-1][j-1] + bc[i-
1][j];
    return bc[n][m];
}
```

2.15 Catalan numbers

```
Cn =
    nX−1
    k=0
      CkCn−1−k =
          1
          n + 1
          n
          k

          (1)
```

The first terms of this sequence are 2, 5, 14, 42, 132, 429, 1430 when C0 = 1. This is the number of ways to build a balanced formula from n sets of left and right parentheses. It is also the number of triangulations of a convex polygon, the number of rooted binary tress on n + 1 leaves and the number of paths across a lattice which do not rise above the main diagonal.

2.16 Eulerian numbers

```
        n
        k

        = k
          n − 1
          k

          + (n − k + 1)
          n − 1
          k − 1

          (2)
```

```
// This is the number of permutations of
length n with exactly k ascending sequences
or runs.
// Basis: k=0 has value 1
#define MAXN 100 // largest n or k
          long eularian(n,k)
          int n,m;
{
    int i,j;
    long e[MAXN][MAXN];
    for (i=0; i<=n; i++) e[i][0] = 1;
    for (j=0; j<=n; j++) e[0][j] = 0;
    for (i=1; i<=n; i++)
        for (j=1; j<i; j++)
            e[i][j] = k*e[i-1][j] + (i-
j+1)*e[i-1][j-1];
    return e[n][k];
}
```

2.17 Karatsuba algorithm in Java

```
// fast algorithm to find multiplication of
two big numbers.
import java.math.BigInteger;
import java.util.Random;
class Karatsuba
{
    private final static BigInteger ZERO =
new BigInteger("0");
    public static BigInteger
karatsuba(BigInteger x, BigInteger y)
    {
        int N = Math.max(x.bitLength(),
y.bitLength());
        if (N <= 2000) return x.multiply(y);
        N=(N/2)+(N %2);
        BigInteger b = x.shiftRight(N);
        BigInteger a =
x.subtract(b.shiftLeft(N));
        BigInteger d = y.shiftRight(N);
        BigInteger c =
y.subtract(d.shiftLeft(N));
        BigInteger ac = karatsuba(a, c);
        BigInteger bd = karatsuba(b, d);
        BigInteger abcd =
karatsuba(a.add(b), c.add(d));
        return
ac.add(abcd.subtract(ac).subtract(bd).shiftL
eft(N)).add(bd.shiftLeft(2*N));
    }
    public static void main(String[] args)
    {
        long start, stop, elapsed;
        Random random = new Random();
        int N = Integer.parseInt(args[0]);
        BigInteger a = new BigInteger(N, 
random);
        BigInteger b = new BigInteger(N, 
random);
        start = System.currentTimeMillis();
        BigInteger c = karatsuba(a, b);
        stop = System.currentTimeMillis();
        System.out.println(stop - start);
        start = System.currentTimeMillis();
        BigInteger d = a.multiply(b);
        stop = System.currentTimeMillis();
        System.out.println(stop - start);
        System.out.println((c.equals(d)));
    }
}
```

2.18 Euler's totient function

```
// the positive integers less than or equal 
to n that are relatively prime to n.
int phi (int n)
{
    int result = n;
    for (int i=2; i*i<=n; ++i)
        if(n %i==0)
        {
            while(n %i==0)
                n /= i;
            result -= result / i;
        }
    if (n > 1)
        result -= result / n;
    return result;
}
```

2.19 Split plane

n lines can split a plane in (n+1)n
2 + 1 sub-regions.

3 Searching Algorithms

3.1 Find rank k in array

```
int find(int l, int r, int k)
{
    int i=0,j=0,x=0,t=0;
    if (l==r) return a[l];
    x=a[(l+r)/2];
    t=a[x];
    a[x]=a[r];
    a[r]=t;
    i=l-1;
    for (int j=l; j<=r-1; j++)
        if (a[j]<=a[r])
        {
            i++;
            t=a[i];
            a[i]=a[j];
            a[j]=t;
        }
    i++;
    t=a[i];
    a[i]=a[r];
    a[r]=t;
    if (i==k) return a[i];
    if (i<k) return find(i+1, r,k);
    return find(l, i-1, k);
}
```

3.2 KMP Algorithm

```
#include <iostream>
#include <string>
#include <vector>
using namespace std;
typedef vector<int> VI;
void buildTable(string& w, VI& t)
{
    t = VI(w.length());
    int i = 2, j = 0;
    t[0] = -1;
    t[1] = 0;
    while(i < w.length())
    {
        if(w[i-1] == w[j])
        {
            t[i] = j+1;
            i++;
            j++;
        }
        else if(j > 0) j = t[j];
        else
        {
            t[i] = 0;
            i++;
        }
    }
}
int KMP(string& s, string& w)
{
    int m = 0, i = 0;
    VI t;
    buildTable(w, t);
    while(m+i < s.length())
    {
        if(w[i] == s[m+i])
        {
            i++;
            if(i == w.length()) return m;
        }
        else
        {
            m += i-t[i];
            if(i > 0) i = t[i];
        }
    }
    return s.length();
}
int main(void)
{
    string a = (string) "The example above 
illustrates the general technique for 
assembling "+
               "the table with a minimum of 
fuss. The principle is that of the overall 
search: "+
               "most of the work was already 
done in getting to the current position, so 
very "+
               "little needs to be done in
leaving it. The only minor complication is
that the "+
               "logic which is correct late
in the string erroneously gives non-proper
"+
               "substrings at the beginning.
This necessitates some initialization
code.";
    string b = "table";
    int p = KMP(a, b);
    cout << p << ": " << a.substr(p,
b.length()) << " " << b << endl;
    return 0;
}
```

4 Dynamic Programming

4.1 0/1 Knapsack problems

```
#include<iostream>
using namespace std;
int f[1000]= {0};
             int n=0, m=0;
             int main(void)
{
    cin >> n >> m;
    for (int i=1; i<=n; i++)
    {
        int price=0, value=0;
        cin >> price >> value;
        for (int j=m; j>=price; j--)
            if (f[j-price]+value>f[j])
                f[j]=f[j-price]+value;
    }
    cout << f[m] << endl;
    return 0;
}
```

4.2 Complete Knapsack problems

```
#include<iostream>
using namespace std;
int f[1000]= {0};
             int n=0, m=0;
             int main(void)
{
    cin >> n >> m;
    for (int i=1; i<=n; i++)
    {
        int price=0, value=0;
        cin >> price >> value;
        for (int j=price; j<=m; j++)
            if (f[j-price]+value>f[j])
                f[j]=f[j-price]+value;
    }
    cout << f[m] << endl;
    return 0;
}
```

4.3 Longest common subsequence (LCS)

```
int dp[1001][1001];
int lcs(const string &s, const string &t)
{
    int m = s.size(), n = t.size();
    if (m == 0 || n == 0) return 0;
    for (int i=0; i<=m; ++i)
        dp[i][0] = 0;
    for (int j=1; j<=n; ++j)
        dp[0][j] = 0;
    for (int i=0; i<m; ++i)
        for (int j=0; j<n; ++j)
            if (s[i] == t[j])
                dp[i+1][j+1] = dp[i][j]+1;
            else
                dp[i+1][j+1] =
max(dp[i+1][j], dp[i][j+1]);
    return dp[m][n];
}
```

4.4 Longest increasing common sequence (LICS)

```
#include<iostream>
using namespace std;
int a[100]= {0};
            int b[100]= {0};
            int f[100]= {0};
            int n=0, m=0;
            int main(void)
{
    cin >> n;
    for (int i=1; i<=n; i++) cin >> a[i];
    cin >> m;
    for (int i=1; i<=m; i++) cin >> b[i];
    for (int i=1; i<=n; i++)
    {
        int k=0;
        for (int j=1; j<=m; j++)
        {
            if (a[i]>b[j] && f[j]>k) k=f[j];
            else if (a[i]==b[j] && k+1>f[j])
f[j]=k+1;
        }
    }
    int ans=0;
    for (int i=1; i<=m; i++)
        if (f[i]>ans) ans=f[i];
    cout << ans << endl;
    return 0;
}
```

4.5 Longest Increasing Subsequence (LIS)

```
#include<iostream>
using namespace std;
int n=0;
      int a[100]= {0}, f[100]= {0}, x[100]=
{0};
      int main(void)
{
    cin >> n;
    for (int i=1; i<=n; i++)
    {
        cin >> a[i];
        x[i]=INT_MAX;
    }
    f[0]=0;
    int ans=0;
    for(int i=1; i<=n; i++)
    {
        int l=0, r=i;
        while (l+1<r)
        {
            int m=(l+r)/2;
            if (x[m]<a[i]) l=m;
            else r=m;
// change to x[m]<=a[i] for non-decreasing
case
        }
        f[i]=l+1;
        x[l+1]=a[i];
        if (f[i]>ans) ans=f[i];
    }
    cout << ans << endl;
    return 0;
}
```

4.6 Maximum submatrix

```
// URAL 1146 Maximum Sum
#include<iostream>
using namespace std;
int a[150][150]= {0};
                 int c[200]= {0};
                 int maxarray(int n)
{
    int b=0, sum=-100000000;
    for (int i=1; i<=n; i++)
    {
        if (b>0) b+=c[i];
        else b=c[i];
        if (b>sum) sum=b;
    }
    return sum;
}
int maxmatrix(int n)
{
    int sum=-100000000, max=0;
    for (int i=1; i<=n; i++)
    {
        for (int j=1; j<=n; j++)
            c[j]=0;
        for (int j=i; j<=n; j++)
        {
            for (int k=1; k<=n; k++)
                c[k]+=a[j][k];
            max=maxarray(n);
            if (max>sum) sum=max;
        }
    }
    return sum;
}
int main(void)
{
    int n=0;
    cin >> n;
    for (int i=1; i<=n; i++)
        for (int j=1; j<=n; j++)
            cin >> a[i][j];
    cout << maxmatrix(n);
    return 0;
}
```

4.7 Partitions of integers

```
#define MAXN 100 // largest n or m
long int_coefficient(n,k) // compute f(n,k)
int n,m;
{
    int i,j;
    long f[[MAXN][MAXN];
                 f [1][1] = 1;
                 for (i=0; i<=n; i++)
f[i][0] = 0;
                 for (i=1; i<=n; i++)
                 for (j=1; j<i; j++)
                 if (i-j <= 0)
                 f[i][j] = f[i][k-1];
                 else
                 f[i][j] = f[i-j][k]+f[i][k-
1];
                 return f[n][k];
      }
```

4.8 Partitions of sets

Number of ways to partition n + 1 items into k sets.

n
k

= k

n − 1
k

+

n − 1
k − 1

(3)

where

n
1

=

n
n

= 1 (4)

5 Trees

5.1 Tree traversal

```
            int L[100]= {0};
int R[100]= {0};
            void DLR(int m)
{
    cout << m << " ";
    if (L[m]!=0) DLR(L[m]);
    if (R[m]!=0) DLR(R[m]);
}
void LDR(int m)
{
    if (L[m]!=0) LDR(L[m]);
    cout << m << " ";
    if (R[m]!=0) LDR(R[m]);
}
void LRD(int m)
{
    if (L[m]!=0) LRD(L[m]);
    if (R[m]!=0) LRD(R[m]);
    cout << m << " ";
}
int main(void)
{
    cin >> n;
    for (int i=1; i<=n; i++)
        cin >> L[i] >> R[i];
    DLR(1);
    cout << endl;
    LDR(1);
    cout << endl;
    LRD(1);
    cout << endl;
    return 0;
}
```

5.2 Depth and width of tree

```
#include <iostream>
#include <queue>
#include <stack>
using namespace std;
int l[100]= {0};
            int r[100]= {0};
            stack<int> mystack;
            int n=0;
            int w=0;
            int d=0;
            int depth(int n)
{
    if (l[n]==0 && r[n]==0)
        return 1;
    int depthl=depth(l[n]);
    int depthr=depth(r[n]);
    int dep=depthl>depthr ? depthl:depthr;
    return dep+1;
}
void width(int n)
{
    if (n<=d)
    {
        int t=0,x;
        stack<int> tmpstack;
        while (!mystack.empty())
        {
            x=mystack.top();
            mystack.pop();
            if (x!=0)
            {
                t++;
                tmpstack.push(l[x]);
                tmpstack.push(r[x]);
            }
        }
        w=w>t?w:t;
        mystack=tmpstack;
        width(n+1);
    }
}
int main(void)
{
    cin >> n;
    for (int i=1; i<=n; i++)
        cin >> l[i] >> r[i];
    d=depth(1);
    mystack.push(1);
    width(1);
    cout << w << " " << d << endl;
    return 0;
}
```

6 Graph Theory

6.1 Graph representation

```
// The most common way to define graph is to
use adjacency matrix
// example:
// (1) (2) (3) (4) (5)
// (1) 2 0 5 0 0
// (2) 4 2 0 0 1
// (3) 3 0 0 1 4
// (4) 6 9 0 0 0
// (5) 1 1 1 1 5
// it’s always a square matrix.
// suppose a graph has n nodes, if given
exactly adjacency matrix
for (int i=1; i<=n; i++)
for (int j=1; i<=n; j++)
{
    cin << a[i][j] << endl;
    }
// Usually will go like this representation
in data
// start_node end_node weight
// suppose m lines
for (int i=1; i<=m; i++)
{
int x=0, y=0, t=0;
cin >> x >> y >> t;
a[x][y]=t;
// if undirected graph
    a[y][x]=t;
}
// another variant: on the ith line, has
data as
// end_node weight
// when you read data, you can assign matrix
as
a[i][x]=t;
// if undirected graph
        a[x][i]=t;
// Initialization of graph !!!IMPORTANT
// Depends on usage, normally initialize as
0 for all elements in matrix.
// so that 0 means no connection, non-0
means connection
// (for problem without weight, use weight
as 1)
// If weights are important in this context
(especially searching for path)
```

```
// Initialize graph as infinity for all
elements in matrix.
// Another way to store graph is Adjacency
list
// No space advantage if using array
(unknown maximum number for in-degree).
// Big space advantage if using dynamic data
structure (like list, vector).
// each row represent a node and its
connectivity.
// we don’t need it so much due to it’s
search efficiency.
// let’s define a node as
        struct Node
{
    int id; // node id
    int w; // weight
};
// suppose n nodes and m lines of inputs as
// start_node end_node weight
// assume using <vector> in this example
// g is a vector, and each element of g is
also a vector of Node
for (int i=1; i<=m; i++)
{
int x=0, y=0, t=0;
cin >> x >> y >> t;
Node temp;
temp.id=y;
temp.w=t;
g[x].push_back(temp);
// if undirected
    temp.id=x;
    g[y].push_back(temp);
}
// Note that you don’t need this node
structure if graph has only connectivity
information.
/**** Special Structure ****/
// Special structure here is usually not a
typical graph, like city-blocks, triangles
// They are represented in 2-d array and
shows weights on nodes instead of edges.
// Note that in this case travel through
edge has no cost, but visit node has cost.
// Triangles: Read data like this
// 1
// 1 2
// 4 2 7
// 7 3 1 5
// 6 2 9 4 6
for (int i=1; i<=n; i++)
for (int j=i; j<=n; j++)
    cin >> a[i][j];
// Simple city-blocks: it’s just like first
form of adjacency matrix, but this time
// represents weights on nodes, may not be
square matrix.
// 1 2 4 5 6
// 2 4 5 1 3
// 4 5 2 3 6
        for (int i=1; i<=n; i++)
            for (int j=1; <=m; j++)
                cin >> a[i][j];
// More complex data structures: typical
city-block structure may has some
constraints on
// questions, but it has no boundaries.
However, some questions requires to form a
maze.
// In these cases, data structures can be
very flexible, it totally depends on how the
question
// presents the data. A usual way is to
record it’s adjacent blocks information:
                struct Block
        {
            bool l[4]; // if has 8 neighbors
then use bool l[8];
// label them as your favor, e.x.
// 1 1 2 3
// 4 x 2 8 x 4
// 3 7 6 5
// true if there is path, false if there is
boundary
// other informations (optional)
                int weight;
                int component_id;
// etc.
            };
// Note that usually we use array from index
1 instead of 0 because sometimes
// you need index 0 as your boundary, and
start from index 1 will give you
// advantage on locating nodes or positions
```

6.2 Flood fill algorithm

```
//component(i) denotes the
//component that node i is in
void flood_fill(new_component)
do
    num_visited = 0
        for all nodes i
            if component(i) = -2
               num_visited = num_visited + 1

    component(i) = new_component

            for all neighbors j of node i

                if component(j) = nil

                    component(j) = -2

                      until num_visited = 0


void find_components()


num_components = 0
```

```
        for all nodes i

            component(node i) = nil

                    for all nodes i

                        if component(node i)
is nil

                            num_components =
num_components + 1

                                    componen
t(i) = -2


flood_fill(component num_components)
```

6.3 SPFA — shortest path

```
int q[3001]= {0}; // queue for node
                                          in
t d[1001]= {0}; // record shortest path from
start to ith node
                                          bo
ol f[1001]= {0};
                                          in
t a[1001][1001]= {0}; // adjacency list
                                          in
t w[1001][1001]= {0}; // adjacency matrix
                                          in
t main(void)
                        {
                            int n=0, m=0;
                            cin >> n >> m;
                            for (int i=1;
i<=m; i++)
                                {
                                    int x=0,
y=0, z=0;
                                    cin >> x
>> y >> z; // node x to node y has weight z
                                    a[x][0]+
+;
                                    a[x][a[x
][0]]=y;
                                    w[x][y]=
z;
                                    /*
                                    // for
undirected graph
                                    a[x][0]+
+;
                                    a[y][a[y
][0]]=x;
                                    w[y][x]=
z;
                                    */
                                }
                                int s=0,
e=0;
                                cin >> s >>
e; // s: start, e: end
                                SPFA(s);
                                cout << d[e]
<< endl;
                                return 0;
                            }
void SPFA(int v0)
{
    int t,h,u,v;
    for (int i=0; i<1001; i++) d[i]=INT_MAX;
    for (int i=0; i<1001; i++) f[i]=false;
    d[v0]=0;
    h=0;
    t=1;
    q[1]=v0;
    f[v0]=true;
    while (h!=t)
    {
        h++;
        if (h>3000) h=1;
        u=q[h];
        for (int j=1; j<=a[u][0]; j++)
        {
            v=a[u][j];
            if (d[u]+w[u][v]<d[v]) // change
to > if calculating longest path
            {
                d[v]=d[u]+w[u][v];
                if (!f[v])
                {
                    t++;
                    if (t>3000) t=1;
                    q[t]=v;
                    f[v]=true;
                }
            }
        }
        f[u]=false;
    }
}
```

6.4 Floyd-Warshall algorithm – shortest path of all pairs

```
// map[i][j]=infinity at start
void floyd()
{
    for (int k=1; k<=n; k++)
        for (int i=1; i<=n; i++)
            for (int j=1; j<=n; j++)
```

```
                if (i!=j && j!=k && i!=k)
                    if
(map[i][k]+map[k][j]<map[i][j])
                        map[i][j]=map[i][k]+
map[k][j];
}
```

6.5 Prim – minimum spanning tree

```
int d[1001]= {0};
             bool v[1001]= {0};
             int a[1001][1001]= {0};
             int main(void)
{
    int n=0;
    cin >> n;
    for (int i=1; i<=n; i++)
    {
        int x=0, y=0, z=0;
        cin >> x >> y >> z;
        a[x][y]=z;
    }
    for (int i=1; i<=n; i++)
        for (int j=1; j<=n; j++)
            if (a[i][j]==0) a[i][j]=INT_MAX;
    cout << prim(1,n) << endl;
}
int prim(int u, int n)
{
    int mst=0,k;
    for (int i=0; i<d.length; i++)
d[i]=INT_MAX;
    for (int i=0; i<v.length; i++)
v[i]=false;
    d[u]=0;
    int i=u;
    while (i!=0)
    {
        v[i]=true;
        k=0;
        mst+=d[i];
        for (int j=1; j<=n; j++)
            if (!v[j])
            {
                if (a[i][j]<d[j])
d[j]=a[i][j];
                if (d[j]<d[k]) k=j;
            }
        i=k;
    }
    return mst;
}
```

6.6 Eulerian circuit

```
// USACO Fence
#include<iostream>
using namespace std;
int f[100]= {0}, ans[100]= {0};
            bool g[100][100]= {0}, v[100]=
{0};
            int n=0, m=0, c=0;
            void dfs(int k)
{
    for (int i=1; i<=n; i++)
        if (g[k][i])
        {
            g[k][i]=false;
            g[i][k]=false;
            dfs(i);
        }
    m++;
    ans[m]=k;
}
int main(void)
{
    cin >> n >> m;
    for (int i=1; i<=m; i++)
    {
        int x=0, y=0;
        g[x][y]=true;
        g[y][x]=true;
        f[x]++;
        f[y]++;
    }
    m=0;
    int k1=0;
    for (int i=1; i<=n; i++)
    {
        if (f[i]%2==1) k1++;
        if (k1>2)
        {
            cout << "error" << endl;
            return 0;
        }
        if (f[i]%2 && c==0) c=i;
    }
    if (c==0) c=1;
    dfs(x);
    for (int i=m; i>=1; i--) cout << ans[i]
<< endl;
    return 0;
}
```

6.7 Topological sort

```
// Find any solution of topological sort.
#include<iostream>
using namespace std;
int f[100]= {0}, ans[100]= {0};
            bool g[100][100]= {0}, v[100]=
{0};
            int n=0, m=0;
            void dfs(int k)
{
    int i=0;
    v[k]=true;
    for (int i=1; i<=n; i++)
        if (g[k][i] && !v[i]) dfs(i);
    m++;
    ans[m]=k;
}
int main(void)
{
    cin >> n >> m;
    for (int i=1; i<=m; i++)
    {
        int x=0, y=0;
        cin >> x >> y;
        g[y][x]=true;
    }
    m=0;
    for (int i=1; i<=n; i++)
        if (!v[i]) dfs(i);
    for (int i=1; i<=n; i++) cout << ans[i]
<< endl;
    return 0;
}
```

// Find the order of topological sort is dictionary minimum

```
#include<iostream>
using namespace std;
int f[100]= {0}, ans[100]= {0};
            bool g[100][100]= {0}, v[100]=
{0};
            int n=0, m=0;
            int main(void)
{
    cin >> n >> m;
    for (int i=1; i<=m; i++)
    {
        int x=0, y=0;
        cin >> x >> y;
        g[x][y]=true;
        f[y]++;
    }
    for (int i=1; i<=n; i++)
    {
        for (int j=1; j<=n; j++)
        {
            if (f[j]==0 && !v[j]) break;
            if (f[j]!=0)
            {
                cout << "error" << endl;
                return 0;
            }
            ans[i]=j;
            v[j]=true;
            for (int k=1; k<=n; k++)
                if (g[j][k]) f[k]--;
        }
    }
    for (int i=1; i<=n; i++) cout << ans[i]
<< endl;
    return 0;
}
```

6.8 Dijkstra

```
const int N = 1e5 + 100;
vector<pii> gra[N];
ll dis[N];
int par[N];
int main()
{
    int node, edge; cin >> node >> edge;
    rep(i, edge)
    {
        int a, b, w; cin >> a >> b >> w;
        gra[a].emplace_back(b, w);
        gra[b].emplace_back(a, w);
    }
    priority_queue<pii> pq;
    rep(i, node) dis[i] = 1e18 + 100;
    int src = 1;
    dis[src] = 0;
    par[1] = -1;
    pq.push ({ -dis[src], src});
    while (pq.size() > 0)
    {
        auto t = pq.top();
        pq.pop();
        int u = t.ss, d = -t.ff;
        if (dis[u] < d) continue;
        for (auto it : gra[u])
        {
            int v = it.ff, w = it.ss;

            if (dis[v] > dis[u] + w)
            {
                dis[v] = dis[u] + w;
                pq.push({ -dis[v], v});
                par[v] = u;
            }
        }
    }


    vector<int>path;
    int xx = node;
    while (xx != -1)
    {
        path.em(xx);
        xx = par[xx];
        if (xx == 0)
        {
            cout << -1;
            return 0;
        }
    }
    reverse(all(path));
    for (int it : path) cout << it << ' ';
}
```

7 Individual Templates

7.1

///Basics

```
->cout << n << " " << flush;
->map<int,int>m={{1,2},{3,3},{2,1},{5,1}};
  it1 = m.lower_bound(5);
    if(it1==m.end()) cout<<"Not found\n";
```

///No. of Triangle from different lengths

```
for(i=0;i<(n-2);i++){
            for(j=i+1;j<(n-1);j++){
                int t = a[j]+a[i];
                auto itr =
upper_bound(a.begin()+j,a.end(), t);
                int idx=itr-a.begin();
                if(*itr==t && itr!=a.end()){
                    ++idx, sum+=(n-idx);
                }
                else if((*itr>t)&&
itr!=a.end()){
                    sum+=(n-idx);
                }
            }
        }
        cout<<sum<<'\n';
```

///Binary Search

```
Return the indx less than or equal to the
element:
int lower_bound(int a[],int lo,int hi,int x)
{
    if(a[0]>x) return 0;
    int mid=(lo+hi+1)/2;
    if(lo>=hi) return mid+1;
    else if(a[mid]<=x) return
lower_bound(a,mid,hi,x);
    else return lower_bound(a,lo,mid-1,x);
}
```

///Number Theory

///GCD

```
int gcd(int a,int b){
   if(b==0) return a;
   return gcd(b,a%b);
}
```

//PRIME FACTORIZATION

```
vector<int>prime_factors;
 for(int i=2;(i*i)<=n;i++){
      while(n%i==0){
           prime_factors.push_back(i);
           n/=i;
      }
  }
  if(n>1)
      prime_factors.push_back(n);
```

///Sieve,highest prime,lowest prime

```
const int N=1e7+10;
vector<bool>isPrime(N,1);
vector<int>hp(N,0), lp(N,0);   ///hp=highest
prime ///lp=lowest prime
int main()
{
    isPrime[0]=isPrime[1]=false;
    for(int i=2;i<N;i++){
        if(isPrime[i]==true){
            hp[i]=lp[i]=i; ///As, i is prime
            for(int j=2*i;j<N;j+=i){
                isPrime[j]=false, hp[j]=i;
                if(lp[j]==0)  lp[j]=i;
            }
        }
    }
    int n; cin>>n;
    vector<int>prime_factors;
    while(n>1){
        int prime_factor=hp[n];
        while(n%prime_factor==0){
prime_factors.push_back(prime_factor);
            n/=prime_factor;
        }
    }
    for(int factor:prime_factors)
        cout<<factor<<" ";
    ///prime factorization : O(log(n))
}
```

///BFS

```
vector<int>adj[100];
int visited[100],int par[100],int dis[100];
void bfs(int s){
    queue<int>q;
    q.push(s);
    visited[s]=1;
    par[s]=-1;
    dis[s]=0;
    while(q.size()!=0){
        int u = q.front();
        q.pop();
        for(int i=0; i<adj[u].size(); i++){
            int v = adj[u][i];
            if(visited[v]==0){
                q.push(v);
                visited[v]=1;
                par[v]=u;
                dis[v]=dis[u]+1;
            }
        }
    }
}

void path(int x){
    if(x==-1)   return;
    path(par[x]);
    cout<<x<<" ";
}

int main(){
    int n,e,s;
    cin>>n>>e;
    while(e--){
        int x,y; cin>>x>>y;
        adj[x].push_back(y);
        adj[y].push_back(x);
    }
    cin>>s;
    bfs(s);
    while(1){
        cout<<"Enter the vertex whose
shortest path and shortest distance from
"<<s<<" to be displayed: ";
        int x;
        cin>>x;
        cout<<"Shortest distance from
"<<s<<" to "<<x<<" is: "<<dis[x]<<endl;
        cout<<"Shortest path from "<<s<<" to
"<<x<<" is: ";
        path(x);
        cout<<endl;
    }
}
///Biparitite
const int N=1000;
int adj[N][N];
int n,e;
bool isBicolored(int s){

    int colorArray[n];
    for(int i=0;i<n;i++)
        colorArray[i]=-1;   ///initially no
color;

    queue<int>q;
    q.push(s);
    colorArray[s]=1;    ///assigning first
color

    while(!q.empty()){
        int senior = q.front();
        q.pop();

        if(adj[senior][senior]==1)  ///self
loop
            return false;

        for(int i=0;i<n;i++){
            int junior=i;
            if(adj[senior][junior]==1){

if(colorArray[junior]==colorArray[senior])
///successor(child/junior) having same color
                    return false;
                ///if(colorArray[junior]!=-
1) continue;    ///not same color but have a
color
                else
if(colorArray[junior]==-1){        ///No
color assigned
                    q.push(junior);

colorArray[junior]=!colorArray[senior];
///assigning diff color
                }
            }
        }
    }
    return true;
}
///All possible ways of a problem
void bfs()
{
    queue<int>q;
    q.push(x);
    vis[x]=1;
    dis[x]=0;
    while(!q.empty()){
        int senior=q.front();
        q.pop();
        for(int i=1;i<=2;i++){
            int junior;
            if(i==1) junior=(2*senior);
            else    junior=senior-1;
            if(junior<=0 || junior>10000)
continue;
            if(!vis[junior]){
                q.push(junior);
                vis[junior]=1;
                dis[junior]=dis[senior]+1;

                if(junior==y) return;
            }
        }
    }
}

int main()
{
    cin>>x>>y;
    bfs();
    cout<<dis[y]<<"\n";
}
///Guilty Prince
string lands[10000];
int row,column,counts;
bool vis[10000][10000];
void dfs(int r,int c)
{
    if(r<0 || r>=row || c<0 || c>=column ||
lands[r][c]=='#' || vis[r][c]==1)
        return;

    vis[r][c]=1, counts++;
    dfs(r-1,c);   ///up
    dfs(r+1,c);   ///down
    dfs(r,c-1);   ///left
    dfs(r,c+1);   ///right
}
int main()
{
    counts=0;
    cin>>column>>row;
    for(int i=0;i<row;i++){
        cin>>lands[i];
    }
    for(int i=0;i<row;i++){
        for(int j=0;j<column;j++){
            vis[i][j]=0;
        }
    }
    for(int i=0;i<row;i++){
        for(int j=0;j<column;j++){
            if(lands[i][j]=='@')
                dfs(i,j);
        }
    }
    cout<<counts<<"\n";
}
///Two farthest node
vector<int>adj[30001];
map<pair<int,int>,int>weight;
```

```
map<int,int>vis,dis;
void dfs(int node)
{
    vis[node]=1;
    for(int i=0;i<adj[node].size();i++){
        int child=adj[node][i];
        if(vis[child]==1) continue;

dis[child]+=dis[node]+weight[{node,child}];
        dfs(child);
    }
}
void reset()
{
    for(int i=0;i<30001;i++){
        adj[i].clear();
    }
    dis.clear(),weight.clear(),vis.clear();
}
int main()
{
    int t; cin>>t;
    for(int p=1;p<=t;p++)
    {
        int n,u,v,w; cin>>n;
        for(int i=0;i<n-1;i++){
            cin>>u>>v>>w;
            adj[u].push_back(v);
            adj[v].push_back(u);

            weight[{u,v}]=w;
            weight[{v,u}]=w;
        }
        dfs(0);
        int max_dis=0,farthestVertex;
        map<int,int>::iterator i;
        for(i=dis.begin();i!=dis.end();i++){
            if(i->second>max_dis){
                max_dis=i->second;
                farthestVertex=i->first;
            }
        }

        vis.clear();
        dis.clear();

        dfs(farthestVertex);
        max_dis=0;
        for(i=dis.begin();i!=dis.end();i++){
            if(i->second>max_dis){
                max_dis=i->second;
            }
        }
        cout<<"Case "<<p<<":
"<<max_dis<<"\n";
        reset();
    }
}
```

```
///SEGMENT TREE
/// first u have to build
const int N = 3e5 + 10;
int tree[N << 2];
int arr[N];

void build(int u, int i, int j)
{
      if (i == j) /// leap node
      {
            tree[u] = arr[i];
            return;
      }

      int mid = (i + j) >> 1;

      build(2 * u, i, mid); /// left child
      build(2 * u + 1, mid + 1, j); ///
right child

      tree[u] = tree[2 * u] ^ tree[2 * u +
1]; /// build as per required
}

void update(int u, int i, int j, int idx,
int x)
{
      if (i == j)
      {
            tree[u] ^= x; /// here is update
as per required
            return;
      }

      int mid = (i + j) >> 1;

      if (idx <= mid) update(2 * u, i, mid,
idx, x);
      else update(2 * u + 1, mid + 1, j,
idx, x);

      tree[u] = tree[2 * u] ^ tree[2 * u +
1]; /// ja change hoise , se jonno range
gulao update korte hocche
}

int query(int u, int i, int j, int b, int e)
{
      if (e < i or j < b) return 0; /// out
of required range
      if (i >= b and j <= e) return tree[u];
/// range is full inside in required range

      int mid = (i + j) >> 1;

      int left = query(2 * u, i, mid, b, e);
      int right = query(2 * u + 1, mid + 1,
j, b, e);

      return  left ^ right; /// here is
operation as per require
}
```

///HASHING

```
#include <bits/stdc++.h>
#define ff first
#define ss second
#define mp make_pair
using namespace std;
typedef long long LL;
typedef pair<LL, LL> PLL;
const PLL M=mp(1e9+7, 1e9+9);   ///Should be
large primes
const LL base=347;              ///Should be
a prime larger than highest value
const int N = 1e6+7;            ///Highest
length of string
ostream& operator<<(ostream& os, PLL hash) {
    return os<<"("<<hash.ff<<",
"<<hash.ss<<")";
}
PLL operator+ (PLL a, LL x)     {return
mp(a.ff + x, a.ss + x);}
PLL operator- (PLL a, LL x)     {return
mp(a.ff - x, a.ss - x);}
PLL operator* (PLL a, LL x)     {return
mp(a.ff * x, a.ss * x);}
PLL operator+ (PLL a, PLL x)    {return
mp(a.ff + x.ff, a.ss + x.ss);}
PLL operator- (PLL a, PLL x)    {return
mp(a.ff - x.ff, a.ss - x.ss);}
PLL operator* (PLL a, PLL x)    {return
mp(a.ff * x.ff, a.ss * x.ss);}
PLL operator% (PLL a, PLL m)    {return
mp(a.ff % m.ff, a.ss % m.ss);}

PLL power (PLL a, LL p) {
    if (p==0)   return mp(1,1);
    PLL ans = power(a, p/2);
    ans = (ans * ans)%M;
    if (p%2)    ans = (ans*a)%M;
    return ans;
}
///Magic!!!!!!!
PLL inverse(PLL a)  {
    return power(a, (M.ff-1)*(M.ss-1)-1);
}
PLL pb[N];      ///powers of base mod M
PLL invb;
///Call pre before everything
void hashPre() {
    pb[0] = mp(1,1);
    for (int i=1; i<N; i++)
        pb[i] = (pb[i-1] * base)%M;
    invb = inverse(pb[1]);
}
///Calculates Hash of a string
PLL Hash (string s) {
    PLL ans = mp(0,0);
    for (int i=0; i<s.size(); i++)
        ans=(ans*base + s[i])%M;
    return ans;
}
///appends c to string
PLL append(PLL cur, char c) {
    return (cur*base + c)%M;
}
///prepends c to string with size k
PLL prepend(PLL cur, int k, char c) {
    return (pb[k]*c + cur)%M;
}
///replaces the i-th (0-indexed) character
from right from a to b;
PLL replace(PLL cur, int i, char a, char b)
{
    cur = (cur + pb[i] * (b-a))%M;
    return (cur + M)%M;
}
///Erases c from the back of the string
PLL pop_back(PLL hash, char c) {
    return (((hash-c)*invb)%M+M)%M;
}
///Erases c from front of the string with
size len
PLL pop_front(PLL hash, int len, char c) {
    return ((hash - pb[len-1]*c)%M+M)%M;
}
///concatenates two strings where length of
the right is k
PLL concat(PLL left, PLL right, int k) {
    return (left*pb[k] + right)%M;
}

///Calculates hash of string with size len
repeated cnt times
///This is O(log n). For O(1), pre-calculate
inverses
PLL repeat(PLL hash, int len, LL cnt) {
    PLL mul = (pb[len*cnt] - 1) *
inverse(pb[len]-1);
    mul = (mul%M+M)%M;
    PLL ans = (hash*mul)%M;

    if (pb[len].ff == 1)    ans.ff =
hash.ff*cnt;
    if (pb[len].ss == 1)    ans.ss =
hash.ss*cnt;
    return ans;
}

///Calculates hashes of all prefixes of s
including empty prefix
vector<PLL> hashList(string s) {
    int n = s.size();
    vector<PLL> ans(n+1);
    ans[0] = mp(0,0);

    for (int i=1; i<=n; i++)
        ans[i] = (ans[i-1] * base + s[i-
1])%M;
    return ans;
}
```

```
///Calculates hash of substring s[l..r] (1 indexed)
PLL substringHash(const vector<PLL> &hashlist, int l, int r) {
    int len = (r-l+1);
    return ((hashlist[r] - hashlist[l-1]*pb[len])%M+M)%M;
}
///Solves LightOJ 1255-Substring Frequency
///You are given two strings A and B. You have to find
///the number of times B occurs as a substring of A.
char buffer[N];
int main()
{
    hashPre();
    int t;
    scanf("%d", &t);
    for (int cs=1; cs<=t; ++cs)
    {
        string a, b;
        scanf("%s", buffer); a = buffer;
        scanf("%s", buffer); b = buffer;
        int na = a.size(), nb = b.size();
        PLL hb = Hash(b);
        vector<PLL> ha = hashList(a);
        int ans = 0;
        for (int i=1; i+nb-1<=na; i++)
            if (substringHash(ha, i, i+nb-1) == hb)  ans++;
        printf("Case %d: %d\n", cs, ans);
    }
}
```

/// TRIE

```
const int N = 1e6 + 100;
int tot_node = 1;
int to[N][26];
int add(string &s) {
    int cur = 1; // root node
    for(int i = 0; i < s.size(); i++) {
        int c = s[i]-'a';
        if(!to[cur][c]) to[cur][c] = ++tot_node;
        cur = to[cur][c];
    }
    return cur; // leaf node where this string ends
}
///
/// KMP PI TABLE (FUCKING MATERIALS)
vector<int> prefix_function(string s) /// this will  return kmp pi table
{
    int n = s.size();
    vector<int> pi(n);/// pi[0] = 0, as per kmp condition
    for (int i = 1; i < n; i++)
    {   /// j = prefix length and end at j-1
        int j = pi[i - 1]; /// max prefix matched at i-1
        while (j > 0 and s[i] != s[j]) j = pi[j - 1];
        if (s[i] == s[j]) ++j;
        pi[i] = j;
    }
    return pi;
}
```

/// DIGIT DP

```
//  How many numbers x are there in the range a to b, where the digit d occurs exactly k times in x?
int a, b, d, k;
vii digit;
int n;
int dp[30][30][3];
/// DP[p][c][f] = Number of valid numbers <= b from this state
/// p = current position from left side (zero based)
/// c = number of times we have placed the digit d so far
/// f = the number we are building has already become smaller than b? [0 = no, 1 = yes]
int call(int idx, int cnt, int f)
{
    if (cnt > k) return 0;
    if (idx >= n) return cnt == k;
    if (dp[idx][cnt][f] != -1) return dp[idx][cnt][f];
    int limit;
    if (!f) limit = digit[idx];
    /// Digits we placed so far matches with the prefix of b
    /// So if we place any digit > num[pos] in the current position, then the number will become greater than b
    else limit = 9;
    /// The number has already become smaller than b. We can place any digit now.
    int xx = 0;
    /// Try to place all the valid digits such that the number doesn't exceed b
    for (int i = 0; i <= limit; i++)
    {
        int cnt1 = 0;
        int ff = f;
        if (!f and i < limit) ff = 1;/// The number is getting smaller at this position
        if (i == d) cnt1 = 1;
        xx += call(idx + 1, cnt + cnt1, ff);
    }

    return dp[idx][cnt][f] = xx;
}
int solve(int x)
{
    mem(dp, -1);
    digit.clear();
    while (x)
    {
        digit.em(x % 10);
        x /= 10;
    }
    reverse(all(digit));
    /// Stored all the digits of x in num
for simplicity
    n = digit.size();
    return call(0, 0, 0);
}
signed main()
{
#ifndef ONLINE_JUDGE
    freopen("input.txt", "r", stdin);
    freopen("output.txt", "w", stdout);
#endif
    cin >> a >> b >> d >> k;
    cout << solve(b) - solve(a - 1) <<
endll;
}
/////
/// SEGMENT TREE LAZY
const int N = 1e5 + 100;
int tree[N << 2], lz[N << 2];
void propagate(int u, int st, int en)
{
      if (!lz[u]) return;
      tree[u] += lz[u] * (en - st + 1);

      if (st != en)
      {
            lz[2 * u] += lz[u];
            lz[2 * u + 1] += lz[u];
      }
      lz[u] = 0;
}
void update(int u, int st, int en, int l,
int r, int x)
{
      propagate(u, st, en);
      if (r < st or en < l) return;
      else if (st >= l and en <= r)
      {
            lz[u] += x;
            propagate(u, st, en);
      }
      else
      {
            int mid = (st + en) >> 1;
            update(2 * u, st, mid, l, r, x);
            update(2 * u + 1, mid + 1, en,
l, r, x);
            tree[u] = tree[2 * u] + tree[2 *
u + 1];
      }
}
int query(int u, int st, int en, int l, int
r)
{
      propagate(u, st, en);
      if (r < st or en < l) return 0;
      else if (st >= l and en <= r) return
tree[u];
      else
      {
            int mid = (st + en) >> 1;
            int left = query(2 * u, st, mid,
l, r);
            int right = query(2 * u + 1, mid
+ 1, en, l, r);
            return left + right;
      }
}


// dp print
int dp[60][1500];
int dir[60][1500];

int knap(int i,int now)
{
    if(i>=cap)
        return 0;
    if(dp[i][now] != -1)
        return dp[i][now];


    int t1=0,t2=0;

    if(now + arrw[i] <= n)
        t1 = arrc[i] +
knap(i+1,now+arrw[i]);
    t2 = knap(i+1,now);

    if(t1>t2)
        dir[i][now] = 1;
    else
        dir[i][now] = 2;

    return dp[i][now] = max(t1,t2);
}

vector <int> pri;

void print(int i,int now)
{
    if(dir[i][now] == -1)
        return;

    if(dir[i][now] == 1)
    {
        pri.push_back(i);
        print(i+1,now+arrw[i]);
    }
    else
        print(i+1,now);
}
```

7.2

```
// typedef long long int;
```

```
const int MX = 1e6+5;
#define pii pair<int, int>



template<typename T>
bool comp(T a, T b){//sort by descending
    return a > b;
}
```

optimized Sieve(finds (n+1)th prime)

```
vector<int> nth_prime;
bitset<MX> visited;
void optimized_prime(){
    nth_prime.push_back(2);
    for(int i=3; i<MX; i+=2){
            if(visited[i])
                continue;
            nth_prime.push_back(i);
            if(1ll*i*i > MX)
                continue;
            for(int j = i*i; j< MX; j+= i+i)
                visited[j] = true;
    }
}
```

stores smallest prime divisor of every num from 1 to x

```
int spf[MX];
void sieve(){
    for(int i=1; i<MX; ++i)
        spf[i] = i;

    for(int i=2; i*i<MX; ++i){
        if(spf[i] != i) continue;
        for(int j=i*i; j<MX; j += i){
            if(spf[j]==j)
                spf[j] = i;
        }
    }
}
map<int, int> mp; //prime factorization
void factorize(int n)
{
    while(n != 1){
        mp[spf[n]]++;
        n /= spf[n];
    }
}
```

when phi(1) to phi(n) is neeeded

```
int phi[MX];
//bitset<MX> visited;// declared before in
optimized SIEVE
void sieve_phi(){
    for(int i=1; i<MX; ++i) phi[i] = i;

    visited[1] = 1;
    for(int i=2; i<MX; ++i){
        if(!visited[i]){
            for(int j = i; j<MX; j+=i){
                visited[j] = 1;
                phi[j] = phi[j]/i*(i-1);
            }
        }
    }
}
```

when only phi(n) is needed

```
int phi(int n){ //O(sqrt(n))
    int res = n;

    for(int p=2; p*p<=n; ++p){
        if(n%p== 0){
            while(n%p == 0)
                n /= p;
            res -= res/p;
        }
    }
    if(n>1) res -= res/n;
    return res;
}
```

claculate nCR start

```
typedef long long LL;
const LL MOD = 1e9+7;
const LL MAX = 2e5+5;

vector<LL> fact(MAX), inv(MAX);
void factorial(){
    fact[0] = 1;
    for(LL i=1; i<MAX; i++)
        fact[i] = (i*fact[i-1])%MOD;
}
LL bigmod(LL a, LL n, LL M=MOD){
    LL res = 1;
    while(n){
        if(n&1) res = (res*a)%M;
        a = (a*a)%M, n /= 2;
    }
    return res;
}
void inverse(){
    for(int i=0; i<MAX; ++i)
        inv[i] = bigmod(fact[i], MOD-2);
}
LL C(LL a, LL b){
      if(a<b or a<0 or b<0) return 0;
    LL de = (inv[b]*inv[a-b])%MOD;
    return (fact[a]*de)%MOD;
}
//call factorial() and inverse() from main
function
// end nCR

LL ModInv(int a, int M){    //M is prime
    return bigmod(a, M-2, M);
}
```

7.3

Knight Moves

```
int X[8]={2,1,-1,-2,-2,-1,1,2};
int Y[8]={1,2,2,1,-1,-2,-2,-1};
```

//bit count in O(1)

```
int BitCount(unsigned int u){
     unsigned int uCount;
     uCount = u - ((u >> 1) & 033333333333)
- ((u >> 2) & 011111111111);
     return ((uCount + (uCount >> 3)) &
030707070707) % 63;
}
```

Matrix Exponentiation

```
// A technique of computing a number raised
to a square matrix in a fast and efficient
manner.
// Uses properties of exponentiation and
binary numbers for fast computation.
//
// Running time:
// O(m^3*log(n)) where m is the size of the
matrix and n is the power the matrix is
being raised to.
//
// INPUT:
// - size of matrix m
// - the matrix A
// - the power n
// - modulo value mod
//
// OUTPUT:
// - the matrix A^n (all values mod m)
//

#include<bits/stdc++.h>
using namespace std;
typedef long long LL;

LL arr[60][60],res[60][60],tmp[60][60],m;

void matMul (LL a[][60], LL b[][60], LL mod)
{
    for(int i=0; i<m; i++)
        for(int j=0; j<m; j++)
        {
            tmp[i][j] = 0;
            for(int k=0; k<m; k++)
            {
                tmp[i][j] +=
(a[i][k]*b[k][j])%mod;
                tmp[i][j] %= mod;
            }
        }
}

void power(LL n, LL mod)
{
    for(int i=0; i<m; i++)
        for(int j=0; j<m; j++)
            if(i==j) res[i][j] = 1;
            else res[i][j] = 0;

    while(n)
    {
        if(n&1)
        {
            matMul(res,arr,mod);
            for(int i=0; i<m; i++)
                for(int j=0; j<m; j++)
res[i][j] = tmp[i][j];
            n--;
        }
        else
        {
            matMul(arr,arr,mod);
            for(int i=0; i<m; i++)
                for(int j=0; j<m; j++)
arr[i][j] = tmp[i][j];
            n/=2;
        }
    }
}

// BEGIN CUT
// The following code solves SPOJ problem
#MPOW: Power of Matrix
int main()
{
    ios_base::sync_with_stdio(false);
cin.tie(NULL); cout.tie(NULL);

//freopen("input.txt","r",stdin);freopen("ou
tput.txt","w",stdout);

    LL t=1, n, mod=1e9+7; cin>>t;
    while(t--)
    {
        cin>>m>>n;
        for(int i=0; i<m; i++)
            for(int j=0; j<m; j++)
cin>>arr[i][j];

        power(n,mod);

        for(int i=0; i<m; i++)
        {
            for(int j=0; j<m; j++)
cout<<res[i][j]<<" ";
            cout<<"\n";
        }
    }

    return 0;
}
// END CUT
```

7.4 Given an undirected graph G with n nodes and m edges.We are required to find in it all the connected components,
i.e, several groups of vertices such that within a group each vertex can be reached from another and no path exists between different groups.

```
// O(n+m)

int n;
vector<int> g[MAXN];
bool used[MAXN];
vector<int> comp;
void dfs(int v)
{
    used[v] = true;
    comp.push_back(v);
    for (size_t i = 0; i < (int)g[v].size();
++i)
    {
        int to = g[v][i];
        if (!used[to])
            dfs(to);
    }
}

void find_comps(){
    for (int i = 0; i < n; ++i)
        used[i] = false;
    for (int i = 0; i < n; ++i){
        if (!used[i]){
            comp.clear();
            dfs(i);
            cout << "Component:";
            for (size_t j = 0; j <
comp.size(); ++j)
                cout << ' ' << comp[j];
            cout << endl;
        }
    }
}
```

7.5 SCC

```
const int N = 1002;
vector<int> adj[N], rev[N];
bitset<N> vis;
int n, m;
int comp[N]; // stores nth node is
includedto which scc_no
void DFS1(int node, stack<int> &TS){
    vis[node] = true;
    for (int child : adj[node])
        if (!vis[child])
            DFS1(child, TS);
    TS.push(node);
}
void DFS2(int node, const int cc_no,
vector<int> &vec){
    vis[node] = true;
    comp[node] = cc_no;
    vec.push_back(node);
    for (int child : rev[node])
        if (!vis[child])
            DFS2(child, cc_no,
                 vec);
}
auto SCC(){
    vis.reset();
    stack<int> TS;
    for (int i = 1; i <= n; ++i)
        if (!vis[i])
            DFS1(i, TS);
    // finding the SCCs using TopSort
    vis.reset();
    int cc_no = 1;
    vector<vector<int>> components;
    while (!TS.empty())
    {
        int idx = TS.top();
        TS.pop();
        if (!vis[idx])
        {
            vector<int> vec;
            DFS2(idx, cc_no++, vec);
            components.push_back(vec);
        }
    }
    return components;
}

signed main(){
    cin >> n >> m;
    for (int i = 0; i < m; ++i){
        int u, v;
        cin >> u >> v;
        // --u, --v;
        adj[u].push_back(v);
        rev[v].push_back(u);
    }
    auto res = SCC();
    int sz = res.size(), scc_no = 1;
    cout << "No. of SCC: " << sz << '\n';
    for (auto x : res)
    {
        cout << "SCC no." << scc_no++ << "
includes nodes : ";
        for (auto y : x) cout<<y<<' ';
        cout << '\n';
    }
}
```

no. of ways and min cost of connecting the sccs

```
const int MOD = 1e9 + 7, N = 1e5 + 2, INF =
1e18 + 2;
int n, m, comp[N];
vector<int> adj[N], rev[N];
bitset<N> vis;
void DFS1(int u, stack<int> &TS){
    vis[u] = true;
    for (int v : adj[u])
        if (!vis[v])
            DFS1(v, TS);
    TS.push(u);
}
void DFS2(int u, const int scc_no, int
&min_cost, int &ways, vector<int> &cost){
    vis[u] = true;
    comp[u] = scc_no;
    for (int v : rev[u])
        if (!vis[v])
        {
            if (min_cost == cost[v])
                ++ways;
            else if (min_cost > cost[v])
            {
                ways = 1;
                min_cost = cost[v];
            }
            DFS2(v, scc_no, min_cost, ways,
                 cost);
        }
}
signed main(){
    FIO cin >> n;
    vector<int> cost(n + 1);
    for (int i = 1; i <= n; ++i)
        cin >> cost[i];
    cin >> m;
    while (m--){
        int u, v;
        cin >> u >> v;
        adj[u].push_back(v);
        rev[v].push_back(u);
    }
    int tot = 0, ways = 1;
    stack<int> TS;
    for (int i = 1; i <= n; ++i)
        if (!vis[i])
            DFS1(i, TS);
    vis.reset();
    int scc_no = 0;
    while (!TS.empty()){
        int u = TS.top();
        TS.pop();
        if (!vis[u]){
            int tmp_cst = cost[u], tmp_ways
= 1;
            DFS2(u, ++scc_no, tmp_cst,
                 tmp_ways, cost);
            tot += tmp_cst;
            ways = (ways * tmp_ways) % MOD;
        }
    }
    cout << tot << ' ' << ways;
}
```

7.6 sqrt decomposition(MO’s Algo)

```
// https://www.spoj.com/problems/DQUERY/
#include <bits/stdc++.h>
using namespace std;
const int SIZE_1 = 1e6 + 10, SIZE_2 = 3e4 +
10;
class query{
public:
    int l, r, indx;
};

int block_size, cnt = 0;
int frequency[SIZE_1], a[SIZE_2];
void add(int indx){
    ++frequency[a[indx]];
    if (frequency[a[indx]] == 1)
        ++cnt;
}
void sub(int indx){
    --frequency[a[indx]];
    if (frequency[a[indx]] == 0)
        --cnt;
}
bool comp(query a, query b){
    if (a.l / block_size == b.l /
block_size)
        return a.r < b.r;
    return a.l / block_size < b.l /
block_size;
}
signed main(){
    int n; cin >> n;
    for(int i = 0; i < n; ++i) cin>>a[i];

    int q; cin >> q;
    int ans[q] = {};
    query Qur[q];
    for (int i = 0; i < q; ++i){
        int l, r; cin>>l>>r;

        Qur[i].l = l - 1;
        Qur[i].r = r - 1;
        Qur[i].indx = i;
    }
    block_size = sqrt(n); // sqrt(q) dileo
hobe, but n is more accurate
    sort(Qur, Qur + q, comp);

    int ML = 0, MR = -1;
    for(int i = 0; i < q; ++i) {
        int L = Qur[i].l;
        int R = Qur[i].r;

        // fixing right pointer
        while (MR < R) add(++MR);
        while (MR > R) sub(MR--);
        // fixiing left pointer
        while (ML < L) sub(ML++);
        while (ML > L) add(--ML);

        ans[Qur[i].indx] = cnt;
    }
    for (int i = 0; i < q; ++i)
        cout << ans[i] << '\n';
}
```

7.7 Meet in the middle

```
#include <bits/stdc++.h>
using namespace std;
int les_equal(vector<int> &s, int key){
    int siz = s.size();
    int lo = 0, hi = siz - 1, ans = 0;
    while (hi >= lo){
        int mid = lo + (hi - lo) / 2;
        if (s[mid] <= key){
            ans = max(ans, mid);
            lo = mid + 1;
        }
        else hi = mid - 1;
    }
    return ans;
}
signed main(){
    FIO int n, n1, n2, t;
    cin >> n >> t;

    n1 = (n + 1) / 2;
    n2 = n / 2;

    int a1[n1]; for(int &i: a1) cin>>i;
    int a2[n2]; for(int &i: a2) cin>>i;

    vector<int> set1, set2;
    for(int mask=0; mask < (1<<n1); ++mask){
        int temp_sum = 0;
        for (int i = 0; i < n1; ++i){
            int f = 1 << i;
            if (f & mask)
                temp_sum += a1[i];
        }
        set1.push_back(temp_sum);
    }
    for(int mask=0; mask < (1<<n2); ++mask){
        int temp_sum = 0;
        for (int i = 0; i < n2; ++i){
            int f = 1 << i;
            if (f & mask)
                temp_sum += a2[i];
        }
        set2.push_back(temp_sum);
    }
    sort(set2.begin(), set2.end());

    // for(auto itr: set2) cout<<itr<<' ';
    // cout<<'\n';
    // for(auto itr: set1) cout<<itr<<' ';
    // cout<<'\n';

    int ans = 0;
    for (auto it : set1){
        int left = t - it;
        if (left < 0) continue;

        int indx = les_equal(set2, left);
        int temp_sum_set2 = (indx != -1 ?
(it + set2[indx]) : 0);
        if (temp_sum_set2 <= t)
            ans = max(ans, temp_sum_set2);
    }
    cout<<ans;
}
```

7.8 PIE(inclusion - exclusion)

```
#include <bits/stdc++.h>
using namespace std;

inline int LCM(int a, int b){
    return a * b / __gcd(a, b);
}

int PIE(int div[], int n, int num){
    int sum = 0;

    for(int msk=1; msk < (1<<n); ++msk){
        int bit_cnt = 0;
        int cur_lcm = 1;

        for (int i = 0; i < n; ++i){
            if (msk & (1 << i)){
                ++bit_cnt;
                cur_lcm = LCM(cur_lcm,
div[i]);
            }
        }

        int cur = num / cur_lcm;
        if (bit_cnt & 1) sum += cur;
        else sum -= cur;
    }
    return num - sum;
}

signed main(){
    int n, m;
    while (cin >> n >> m){
        int a[m];
        for(int &i : a)cin >> i;

        cout << PIE(a, m, n) << '\n';
    }
}
```

```
{
"cmd": ["g++.exe","-std=c++14", "${file}",
"-o", "${file_base_name}.exe", "&&" ,
"${file_base_name}.exe<inputf.in>outputf.i
n"],
"shell":true,
"working_dir":"$file_path",
"selector":"source.cpp"
}
```

topic : Expected Value

If the probability that your candidate will win is strictly greater than W%, print

GET A CRATE OF CHAMPAGNE FROM THE BASEMENT!

If your candidate has no chance of winning, Print RECOUNT!

Otherwise, print PATIENCE, EVERYONE!

```
#include <bits/stdc++.h>
using namespace std;

#define err 1e-15
double dp[103][103]; // dp[confirmed
votes][unknown votes]

signed main(){
    dp[0][0] = 1.0;
    for(int i = 1; i < 101; ++i){
        for(int j=0; j <= i; ++j){
            dp[i][j] += 0.5 * dp[i - 1][j];
            // confirmed vote increased but
the vote didn't go to my favour
            dp[i][j + 1] += 0.5 * dp[i -
1][j];
            // confirmed vote increased and
went to my favour
        }
    }
    TC{
        int n, a, b, w;
        cin >> n >> a >> b >> w;
        int un_c = n - (a + b);
        int flag = 101;
        for(int i=0; i <= un_c; ++i){
            if(2*a + 2*i > n){
                flag = i;
                break;
            }
        }
        double sum = 0;
        for (int i = flag; i <= un_c; ++i)
            sum += dp[un_c][i];

        sum *= 100;
        sum -= err;
        if (sum > w)
            cout<<"GET A CRATE OF CHAMPAGNE
FROM THE BASEMENT !\n";
        else if(flag==101)
cout<<"RECOUNT!\n";
        else cout<<"PATIENCE,EVERYONE!\n";
    }
}
```