Contents

1 Build and Snippet **1**
- 1.1 Snippet . . . 1
- 1.2 Sublime Build . . . 1

2 Data Structures **1**
- 2.1 Ordered Set . . . 1
- 2.2 BIT . . . 1
- 2.3 2D BIT . . . 1
- 2.4 Segment Tree . . . 1
- 2.5 Lazy Propagation . . . 2
- 2.6 Sparse Table . . . 3
- 2.7 DSU . . . 3
- 2.8 Merge Sort Tree . . . 4

3 Number Theory **4**
- 3.1 Big MOD . . . 4
- 3.2 Sieve . . . 4
- 3.3 Bitwise Sieve . . . 5
- 3.4 Eulars Totient Function . . . 5
- 3.5 Divisor . . . 5
- 3.6 nCr . . . 5
- 3.7 Combinatorics . . . 5
- 3.8 Chinese Reminder Theorem . . . 6
- 3.9 NOD and SOD . . . 6
- 3.10 Matrix . . . 7
- 3.11 Pollard rho . . . 7

4 Graph **8**
- 4.1 Bellman Ford . . . 8
- 4.2 Dijkstra . . . 8
- 4.3 K th shortest path . . . 8
- 4.4 KRUSKALS SPANNING TREE(MST) . . . 9
- 4.5 LCA Related . . . 9
- 4.6 STRONGLY CONNECTED COMPONENT . . . 10

5 String **10**
- 5.1 Hashing . . . 10
- 5.2 KMP . . . 11
- 5.3 Manacher . . . 11
- 5.4 Trie . . . 11

6 Divide and Conquer **11**
- 6.1 Max Subarray Sum . . . 11

7 DP **12**
- 7.1 Coin Change . . . 12
- 7.2 Knapsack . . . 12
- 7.3 SOS . . . 12

8 Equations and Formulas **14**
- 8.1 Catalan Numbers . . . 14
- 8.2 Stirling Numbers First Kind . . . 14
- 8.3 Stirling Numbers Second Kind . . . 14
- 8.4 Other Combinatorial Identities . . . 14
- 8.5 Different Math Formulas . . . 14
- 8.6 GCD and LCM . . . 14

1 Build and Snippet

1.1 Snippet

```
int main()
{
  mt19937_64 RNG(chrono::steady_clock::now().time_since_epoch().count());
  auto begin = std::chrono::high_resolution_clock::now();
  cin.tie(0)->ios_base::sync_with_stdio(0);
  int T = 1; // cin >> T;
  // cin.ignore(numeric_limits<streamsize>::max(), '\n');
  for(int testCase=1;testCase<=T;testCase++) {
    // cout << fixed << setprecision(1);
    // cout << "Case " << testCase << ": ";
    solve(testCase);
  }
  auto end = std::chrono::high_resolution_clock::now();
  auto elapsed = std::chrono::duration_cast<std::chrono::nanoseconds>(end - begin);
  cerr << "Time measured: " << elapsed.count() * 1e-9 << " seconds.\n";
  return 0;
}
```

1.2 Sublime Build

```
#for linux
  {
    "shell_cmd": "g++ $file -o $file_base_name && ./$file_base_name<input.txt> output.
        txt && rm $file_base_name",
    "working_dir": "$file_path",
    "selector": "source.c++"
  }
#for windows
  {
  "shell_cmd": "g++ -std=c++17 $file -o $file_base_name.exe && $file_base_name.exe <
      input.txt> output.txt && del $file_base_name.exe",
  "working_dir":"$file_path",
  "selector":"source.c++"
  }
```

2 Data Structures

2.1 Ordered Set

```
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;
#define ordered_multiset tree<int, null_type,less_equal<int>, rb_tree_tag,
    tree_order_statistics_node_update>
X.order_of_key(d) -> number of elements less than d in X
*X.find_by_order(d) -> number at index d in X
```

2.2 BIT

```
int a[N],bin[N];

void update(int idx, int val, int n){
  while(idx<=n){
    bin[idx]+=val;
    idx += idx&-idx;
  }
}
int helper(int idx){
    int sum = 0;
    while(idx>0){
        sum+=bin[idx];
        idx -= idx&-idx;
    }
    return sum;
}
int query(int l,int r){
    return helper(r)-helper(l-1);
}
```

2.3 2D BIT

```
//forest query problem
int a[10001][10001],bin[10001][10001];
void update(int x,int y, int val, int n){
    for(int i=x; i<=n; i += i&-i){
        for(int j=y; j<=n; j += j&-j){
            bin[i][j]+=val;
        }
    }
}
int query(int x,int y){
  int sum = 0;
  for(int i=x; i>0; i -= i&-i){
    for(int j=y; j>0; j -= j&-j){
      sum += bin[i][j];
    }
  }
  return sum;
// check --> query(x2,y2)-query(x2,y1-1)-query(x1-1,y2)+query(x1-1,y1-1)
}
```

2.4 Segment Tree

```
class segmentTree{
  int *seg; int n;
  int merge(int a,int b){
```

```
    return max(a,b);
  }
  void build(int idx,int le, int ri,vector<int> &v){
    if(le==ri){
      seg[idx] = v[le]; return;
    }
    int mid=(le+ri)>>1;
    build(2*idx+1, le,mid,v);
    build(2*idx+2,mid+1,ri,v);
    seg[idx] = merge(seg[2*idx+1],seg[2*idx+2]);
  }
  void update(int idx, int le,int ri,int k,int val){
    if(le==ri){
      seg[idx] = val; return;
    }
    int mid = (le+ri) >> 1;
    if(k<=mid) update(2*idx+1,le,mid,k,val);
    else update(2*idx+2,mid+1,ri,k,val);
    seg[idx] = merge(seg[2*idx+1],seg[2*idx+2]);
  }
  int query(int idx,int le,int ri,int l,int r){
    if(l<=le && r>=ri) return seg[idx];
    if(r<le || l>ri) return 0; ///////identity
    int mid = (le+ri) >> 1;
    return merge(query(2*idx+1,le,mid,l,r) , query(2*idx+2,mid+1,ri,l,r));
  }
  public:
  segmentTree(int m,vector<int> &v){
    n = m;
    seg = new int [4*n];
    build(0,0,n-1,v);
  }
  void update(int k,int val){
    update(0,0,n-1,k,val);
  }
  int query(int l,int r){
    return query(0,0,n-1,l,r);
  }
};
//extra part: use of array compression with segment tree
void compressarray(){ //longest increasing and decreasing subsequence
  int n; cin>>n;
  int a[n],b[n];
  for(int i=0;i<n;i++){
    cin>>a[i];
    b[i] = a[i];
  }
  sort(b,b+n);
  int l[n], r[n]; //increasing and decreasing subsequence from each i[0,n-1]
  vector<int> v(n,0);
  segmentTree treel(n,v),treer(n,v);
  for(int i=0;i<n;i++){
    int pos = lower_bound(b,b+n,a[i])-b;
    int val = (treel.query(0,pos-1));
    l[i] = val;
    treel.update(pos,val+1);
  }
  for(int i=n-1;i>=0;i--){
    int pos = lower_bound(b,b+n,a[i])-b;
    int val = (treer.query(0,pos-1));
    r[i] = val;
    treer.update(pos,val+1);
  }
}
```

2.5 Lazy Propagation

```
template<typename T1,typename T2>
//T1 is main array and T2 is the data type of segment tree
class segtree{
  private:
  int n;
  T2 (*seg);
  T2 identity;
  int *lazy;
  T2 merge(T2 a,T2 b){
    return max(a,b);
  }
  void propagate(int idx,int val,int le,int ri){//change propagate function
      accordingly
    seg[idx] += (ri-le+1)*val;
    if(le != ri){
      lazy[2*idx+1] += val;
      lazy[2*idx+2] += val;
    }
    lazy[idx] = 0;
  }
  void build(int idx,int le, int ri,T1 a[]){
    if(le==ri){
      seg[idx] = a[le];     //how the main array will affect the segment tree
      return;
    }
    int mid=(le+ri)/2;
```

```
    build(idx*2+1,le,mid,a);
    build(idx*2+2,mid+1,ri,a);
    seg[idx] = merge(seg[2*idx+1],seg[2*idx+2]);
  }
  void update(int idx,int le,int ri,int l,int r,int val){
    if(lazy[idx]) propagate(idx,lazy[idx],le,ri);
    if(r<le || l>ri) return;
    if(le>=l && ri<=r){
      propagate(idx,val,le,ri); return;
    }
    int mid=(le+ri)/2;
    update(idx*2+1,le,mid,l,r,val);
    update(idx*2+2,mid+1,ri,l,r,val);
    seg[idx] = merge(seg[2*idx+1],seg[2*idx+2]);
  }

  T2 query(int idx,int le,int ri,int l,int r){
    if(lazy[idx]){
        propagate(idx,lazy[idx],le,ri);
    }
    if(l<=le && r>=ri) return seg[idx];
    if(r<le || l>ri) return identity;
    int mid=(le+ri)/2;
    return merge(query(idx*2+1,le,mid,l,r) , query(idx*2+2,mid+1,ri,l,r));
  }
  public:
  segtree(T1 arr[],int m,,T2 Identity){
    n=m;
    identity = Identity;
    seg = new T2[n*4];
    lazy = new int[n*4];
    memset(lazy,0,sizeof(lazy));
    build(0,0,n-1,arr);
  }
  void update(int l,int r,int val){
    update(0,0,n-1,l,r,val);
  }
  T2 query(int l,int r){
    return query(0,0,n-1,l,r);
  }
};
```

2.6 Sparse Table

```
LL sparseTable[MAXN][(LL)log2(MAXN) + 5];
void build(LL n, vector<LL> &v){
  for(LL i = 0; i < n; i++) sparseTable[i][0] = v[i];
  for(LL j = 1; (1 << j) <= n; j++) {
    for(LL i = 0; (i + (1 << j) - 1) < n; i++)
      sparseTable[i][j] = min(sparseTable[i][j - 1], sparseTable[i + (1 << (j - 1))][j
           - 1]);
  }
}
LL query(LL L, LL R){
  LL j = (LL)log2(R - L + 1);
  return min(sparseTable[L][j], sparseTable[R - (1 << j) + 1][j]);
}

// 2-DIMENTIONAL SPARSE TABLE FOR A MATRIX/GRID
ll grid[MAXN][MAXN];
ll sparseTable[MAXN][MAXN][10];
void build(ll n){
  for(ll k=0;(1 << k)<=n;k++) {
    for(ll i=0;i+(1 << k)-1 < n;i++) {
      for(ll j=0;j+(1 << k)-1 < n;j++) {
        if(!k) sparseTable[i][j][k] = grid[i][j];
        else {
          ll power = 1 << (k - 1);
          sparseTable[i][j][k] = max({sparseTable[i][j][k - 1], sparseTable[i + power][
              j][k - 1], sparseTable[i][j + power][k - 1], sparseTable[i + power][j +
              power][k - 1]});
        }
      }
    }
  }
}
ll query(ll i, ll j, ll s){
  ll k = log2(s);
  ll power = 1 << k;
  return max({sparseTable[i][j][k], sparseTable[i + s - power][j][k], sparseTable[i][j
       + s - power][k], sparseTable[i + s - power][j + s - power][k]});
}
void SPARSE_TABLE(){
  build(n);
  while(q--) {
    ll R, C, S; cin >> R >> C >> S;
    R--; C--;
    cout << query(R, C, S) << endl;
  }
}
```

2.7 DSU

```
struct DSU {
```

```
  vector <int> par, size;
  DSU(int n) : par(n), size(n) {
      fill(size.begin(), size.end(), 1);
      iota(par.begin(), par.end(), 0);
  }
  int root(int node){
      if(par[node] == node) return node;
      return par[node] = root(par[node]);
  }
  bool same(int x, int y){
      return root(x) == root(y);
  }
  void merge(int x, int y){
      x = root(x), y = root(y);
      if(x == y) return;
      if(size[x] < size[y]) swap(x, y);
      size[x] += size[y], par[y] = x;
  }
};
```

2.8 Merge Sort Tree

```
const int inf = 1e9 + 7;
const int N = 5e4 + 7;
int n, m;
int a[N];
vector<int> tree[N * 4];
vector<int> merge(vector<int> a,vector<int> b){
  int i=0, j=0, sza=a.size(), szb=b.size();
  vector<int> ret;
  while (i < sza or j < szb){
    if (i == sza) ret.emplace_back(b[j++]);
    else if (j == szb) ret.emplace_back(a[i++]);
    else{
      if (a[i] < b[j]) ret.emplace_back(a[i++]);
      else ret.emplace_back(b[j++]);
    }
  }
  return ret;
}
void build(int v, int l, int r){
  if (l == r) return void(tree[v] = {a[l]});
  int m = l + (r - l) / 2;
  build(v * 2, l, m);
  build(v * 2 + 1, m + 1, r);
  tree[v] = merge(tree[v * 2], tree[v*2+1]);
}
int query(int v, int l, int r, int ql,int qr, int k){
  if (r < ql or qr < l) return 0;
  if (l >= ql and r <= qr){
    int lo = 0, hi = (int)tree[v].size() - 1, pos = -1;
    while (lo <= hi){
      int mid = lo + (hi - lo) / 2;
      if (tree[v][mid] > k) hi = mid - 1;
      else pos = mid, lo = mid + 1;
    }
    return pos + 1;
  }
  else{
      int m = l + (r - l) / 2;
      return query(v * 2, l, m, ql, qr, k) + query(v * 2 + 1, m + 1, r, ql, qr, k);
  }
}
void update(int v, int b, int e, int i,int x){
  if (b == e) return void(tree[v] = {x});
  int mid = b + (e - b) / 2;
  if (mid < i) update(v * 2 + 1, mid + 1, e, i, x);
  else update(v * 2, b, mid, i, x);
  tree[v] = merge(tree[v * 2], tree[v*2+1]);
}
//build(1, 0, n - 1),query(1, 0, n - 1, l - 1, r - 1, k)
```

3 Number Theory

3.1 Big MOD

```
LL bigmod(LL x, LL n, LL mod) {
  if(n == -1) n = mod - 2;
  LL ans = 1;
  while(n) {
    if((n & 1)) ans = (ans * x) % mod;
    n >>= 1;
    x = (x * x) % mod;
  }
  return ans;
}
```

3.2 Sieve

```
mod = {1500000007, 1500000013, 1500000023, 1500000057, 1500000077};

const int N = 1e6;
int divisor[N+1];
void sieve(){
 for(int i=1;i<=N;i++) divsor[i]=i;
 for(int i=2;i<=N;i+=2) divisor[i]=2;
```

```
 for(int i=3;i<=N;i+=2){
  if(divisor[i]==i){
   for(int j=i*i;j<=N;j+=i){
    if(divisor[j]==j) divisor[j]=i;
   }
  }
 }
}
```

3.3 Bitwise Sieve

```
const int nmax = 1e8+1;
int mark[(nmax>>6)+1];
vector<int> primes;
#define isSet(n,pos) (bool)((n) & (1LL<<(pos)))
#define Set(n,pos) ((n) | (1LL<<(pos)))
void sieve(int n){
 for(int i=3;i*i<=n;i+=2){
  if(isSet(mark[i>>6],(i>>1) & 31)==0){
   for(int j=i*i;j<=n;j+=(i<<1))
    mark[j>>6]=Set(mark[j>>6],(j>>1) & 31);
  }
 }
 primes.push_back(2);
 for(int i=3;i<=n;i+=2){
  if(isSet(mark[i>>6],(i>>1) & 31)==0)
   primes.push_back(i);
 }
}
```

3.4 Eulars Totient Function

```
int phi(int n){
 int ret=n;
 for(int i=2;i*i<=n;i++){
  if(n%i==0){
   while(n%p==0) n/=i;
   ret -= ret/i;
  }
 }
 if(n>1) ret -= ret/n;
 return ret;
}
void phi_in_range(){
 int N=1e6, phi[N+1];
 for(int i=0;i<=N;i++) phi[i] = i;
 for(int i=2;i<=N;i++){
  if(phi[i] != i) continue;
  for(int j=i;j<=N;j+=i){
   phi[j] -= phi[j]/i;
  }
 }
}

#some important properties of phi
phi(p) = p-1 ,where p is a prime number
phi(a*b) = phi(a)*phi(b) ,where a and b are co-prime
phi(a*b) = phi(a)*phi(b)*(gcd(a,b)/phi(gcd(a,b))) ,for any number
phi(p^k) = p^k - p^(k-1) ,where p is a prime number, '^' indicates power
Sum of values of totient functions of all divisors of n is equal to n.
```

3.5 Divisor

```
// calculate divisor in range[1,n]
LL sum_in_range(LL n) {
  return n * (n + 1) / 2;
}
LL sum_all_divisors(LL n) {
  LL ans = 0;
  for(LL i=1;i*i<=n;i++) {
    LL hello = i * (n / i - i + 1);
    LL world = sum_in_range(n / i) - sum_in_range(i);
    ans += hello + world;
  }
  return ans;
}
```

3.6 nCr

```
namespace com{
    LL fact[N], inv[N], inv_fact[N];
    void init(){
        fact[0] = inv_fact[0] = 1;
        for(int i = 1; i < N; i++){
            inv[i] = i == 1 ? 1 : (LL) inv[i - mod%i] * (mod/i + 1) % mod;
            fact[i] = (LL) fact[i-1] * i % mod;
            inv_fact[i] = (LL) inv_fact[i-1] * inv[i] % mod;
        }
    }
    LL C(int n,int r){
        return (r < 0 or r > n) ? 0 : fact[n] * inv_fact[r] % mod * inv_fact[n-r] %
             mod;
    }
}
```

3.7 Combinatorics

```
/* Given n boxes, each box has cnt[i] different (distinct) items,
   you can take only 1 object from each box. how many different combinations
   of choices are there */
ll call(ll box, ll take, vector <ll> &cnt){
    vector < vector <int>> DP(box+1, vector <int> (take+2));
    dp[0][0] = 1, dp[0][1] = cnt[0];
    for(int s = 0; s <= take; s++){
        for(int idx = 0; idx < box; idx++){
            dp[idx+1][s] = ( dp[idx+1][s] + dp[idx][s]);
            dp[idx+1][s+1] = (dp[idx+1][s+1] + dp[idx][s] * cnt[idx+1][s]);
        }
    }
    return dp[box-1][take];
}
```

3.8 Chinese Reminder Theorem

```
using LL = long long;
using PLL = pair <LL,LL>;
// given a, b will find solutions for, ax + by = 1
tuple <LL,LL,LL> EGCD(LL a, LL b){
    if(b == 0) return {1, 0, a};
    else{
        auto [x,y,g] = EGCD(b, a%b);
        return {y, x - a/b*y,g};
    }
}
// given modulo equations, will apply CRT
PLL CRT(vector <PLL> &v){
    LL V = 0, M = 1;
    for(auto &[v, m]:v){ //value % mod
        auto [x, y, g] = EGCD(M, m);
        if((v - V) % g != 0)
            return {-1, 0};
        V += x * (v - V) / g % (m / g) * M, M *= m / g;
        V = (V % M + M) % M;
    }
    return make_pair(V, M);
}
```

3.9 NOD and SOD

```
namespace sieve{
  const int N = 1e7;
  vector <int> primes;
  int spf[N+5], phi[N+5], NOD[N+5], cnt[N+5], POW[N+5];
  bool prime[N+5];
  int SOD[N+5];
void init(){
  fill(prime+2, prime+N+1, 1);
  SOD[1] = NOD[1] = phi[1] = spf[1] = 1;
  for(LL i=2;i<=N;i++){
    if(prime[i]) {
      primes.push_back(i), spf[i] = i;
      phi[i] = i-1;
      NOD[i] = 2, cnt[i] = 1;
      SOD[i] = i+1, POW[i] = i;
    }
    for(auto p:primes){
      if(p*i>N or p > spf[i]) break;
      prime[p*i] = false, spf[p*i] = p;
      if(i%p == 0){
        phi[p*i]=p*phi[i];
        NOD[p*i]=NOD[i]/(cnt[i]+1)*(cnt[i]+2), cnt[p*i]=cnt[i]+1;
        SOD[p*i]=SOD[i]/SOD[POW[i]]*(SOD[POW[i]]+p*POW[i]),POW[p*i]=p*POW[i];
        break;
      } else {
        phi[p*i]=phi[p]*phi[i];
        NOD[p*i]=NOD[p]*NOD[i], cnt[p*i]=1;
        SOD[p*i]=SOD[p]*SOD[i], POW[p*i]=p;
      }
    }
  }
}
// O(lg n factorization for small numbers upto sieve)
map <ULL,int> fast_factorize(ULL n){
    map <ULL,int> ans;
    for(;n>1;n/=spf[n])
        ans[spf[n]]++;
    return ans;
}
// factorization for big numbers
// using pollard rho might be better
map <ULL,int> factorize(ULL n){
    int cnt = 0;
    map <ULL,int> ans;
    for(auto p:primes){
        if(p*p>n) break;
        for(;n%p==0;n/=p)
            ans[p]++;
    }
    if(n!=1) ans[n]++;
    return ans;
}
```

```
  // only for large numbers
  int number_of_div(ULL n){
      if(n < 1) return 0;
      auto A = factorize(n);
      int ans = 1;
      for(auto [p,cnt]:A)
          ans *= cnt+1;
      return ans;
  }
  ULL sum_of_div(ULL n){
      if(n < 1) return 0;
      ULL ans = 1, ppow;
      for(ULL p:primes){
          if(p*p > n) break;
          for(ppow=p; n%p==0; n/=p,ppow*=p);
          ans *=(ppow-1)/(p-1);
      }
      return n == 1 ? ans: ans*(1+n);
  }
  ULL PHI(ULL n){
      ULL ans = n;
      for(auto [p,cnt]:factorize(n))
          ans = ans/p*(p-1);
      return ans;
  }
}
```

3.10 Matrix

```
int n;
struct Matrix{
  vector<vector<LL>> Mat = vector<vector<LL>>(n, vector<LL>(n));
   // memset(Mat,0,sizeof(Mat));
  Matrix operator*(const Matrix &other){
   Matrix product;
   for (int i = 0; i < n; i++){
     for (int j = 0; j < n; j++){
       for (int k = 0; k < n; k++){
         LL temp = ((Mat[i][k] % mod)*(other.Mat[k][j]%mod))%mod;
         product.Mat[i][j] = (product.Mat[i][j] % mod + temp % mod) % mod;
       }
     }
   }
   return product;
  }
};
Matrix MatExpo(Matrix a, int p){
  Matrix product;
  for (int i = 0; i < n; i++)
    product.Mat[i][i] = 1;
  while (p > 0){
    if (p % 2) product = product * a;
    p /= 2;
    a = a * a;
  }
  return product;
}
```

3.11 Pollard rho

```
namespace rho{
  inline LL mul(LL a, LL b, LL mod) {
    LL result = 0;
    while (b) {
      if (b & 1) result = (result + a) % mod;
      a = (a + a) % mod;
      b >>= 1;
    }
    return result;
  }
  inline LL bigmod(LL num,LL pow,LL mod){
    LL ans = 1;
    for( ; pow > 0; pow >>= 1, num = mul(num, num, mod))
      if(pow&1) ans = mul(ans,num,mod);
    return ans;
  }
  inline bool is_prime(LL n){
    if(n < 2 or n % 6 % 4 != 1) return (n|1) == 3;
    LL a[] = {2, 325, 9375, 28178, 450775, 9780504, 1795265022};
    LL s = __builtin_ctzll(n-1), d = n >> s;
    for(LL x: a){
      LL p = bigmod(x % n, d, n), i = s;
      for( ; p != 1 and p != n-1 and x % n and i--; p = mul(p, p, n));
      if(p != n-1 and i != s) return false;
    }
    return true;
  }
  LL f(LL x, LL n) {
    return mul(x, x, n) + 1;
  }
  LL get_factor(LL n) {
    LL x = 0, y = 0, t = 0, prod = 2, i = 2, q;
    for( ; t++ %40 or __gcd(prod, n) == 1; x = f(x, n), y = f(f(y, n), n) ){
      (x == y) ? x = i++, y = f(x, n) : 0;
```

```
    prod = (q = mul(prod, max(x,y) - min(x,y), n)) ? q : prod;
  }
  return __gcd(prod, n);
 }
 void _factor(LL n, map <LL, int> &res) {
  if(n == 1) return;
  if(is_prime(n)) res[n]++;
  else {
    LL x = get_factor(n);
    _factor(x, res);
    _factor(n / x, res);
  }
 }
 map <LL, int> factorize(LL n){
  map <LL, int> res;
  if(n < 2)  return res;
  LL small_primes[] = {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 };
  for (LL p: small_primes)
    for( ; n % p == 0; n /= p, res[p]++);
  _factor(n, res);
  return res;
 }
}
```

4 Graph

4.1 Bellman Ford

```
void bellmanford(int n, int m, vector<int> edge[], int dist[], int src){
 fill(dist, dist + n, INT_MAX);
 dist[src] = 0;
 int i, j, k;
 vector<int> v;
 for (i = 0; i < n; i++){
  for (j = 0; j < m; j++) {
    v = edge[j];
    if (dist[v[1]] > dist[v[0]] + v[2])
     dist[v[1]] = dist[v[0]] + v[2];
  }
 }
 for (j = 0; j < m; j++){ // For checking negative loop
    v = edge[j];
    if (dist[v[1]] > dist[v[0]] + v[2]){
       fill(dist, dist + n, INT_MIN); // Negative loop detected
       return;
    }
 }
}
```

4.2 Dijkstra

```
struct node {
 int to;
 LL weight;
 bool operator<(const node &a) const {
  return weight > a.weight;
 }
};
vector<node> adj[N];
void dijkstra(int src, vector<LL> &dist, vector<int> &parent) {
 parent.assign(n + 1, -1);
 priority_queue<node> pq;
 pq.push({src, 0});
 dist[src] = 0;
 parent[src] = -1;
 while(!pq.empty()) {
  auto cur = pq.top(); pq.pop();
  for(auto next : adj[cur.to]) {
   if(dist[next.to] > dist[cur.to] + next.weight) {
    dist[next.to] = dist[cur.to] + next.weight;
    pq.push({next.to, dist[next.to]});
    parent[next.to] = cur.to;
   }
  }
 }
}
```

4.3 K th shortest path

```
void K_shortest(int n,int m){
 int st,des,k,u,v;
 LL w;
 scanf("%d%d%d", &st,&des,&k);
 st--,des--;
 vector <vector<pii > > edges(n);
 for(int i=0;i<m;i++){
  scanf("%d%d%lld", &u,&v,&w);
  u--,v--;
  edges[u].push_back({w,v});
 }
 vector < vector <LL> > dis(n,vector <LL> (k+1,1e8));
 vector <int> vis(n);
 priority_queue <pii ,vector <pii >, greater< pii > > q;

 q.emplace(0LL,st);
```

```
 while(!q.empty()){
  v = q.top().second, w = q.top().first;
  q.pop();
  if(vis[v]>=k) continue;
  // for the varient, check if this path is greater than previous, if not, continue
  //if(vis[v]>0 && w == dis[v][vis[v]-1]) continue;
  dis[v][vis[v]] = w;
  vis[v]++;
  for(auto nd:edges[v]){
   q.emplace(w+nd.first,nd.second);
  }
 }
 LL ans = dis[des][k-1];
 if(ans==1e8) ans = -1;
 printf("%lld\n", ans);
}
```

4.4 KRUSKALS SPANNING TREE(MST)

```
struct info {
  int u, v, w;
};
int parent[N], sz[N];
void initialize(int n) {
  for(int i=0;i<=n;i++) {
    parent[i] = i;
    sz[i] = 1;
  }
}
int root(int x) {
  if(parent[x] == x) return x;
  return parent[x] = root(parent[x]);
}
void merge(int x, int y) {
  int p = root(x);
  int q = root(y);
  if(sz[p] > sz[q]) {
    parent[p] = parent[q];
    sz[p] += sz[q];
    sz[q] = 0;
  }
  else {
    parent[q] = parent[p];
    sz[q] += sz[p];
    sz[p] = 0;
  }
}
// v should be already sorted according to weight
LL kruskal(int n, vector<info> &v) {
  initialize(n);
  LL cost = 0;
  for(auto cur : v) {
    if(root(cur.u) != root(cur.v)) {
      cost += cur.w;
      merge(cur.u, cur.v);
    }
  }
  return cost;
}
// PRIM'S SPANNING TREE (MST)
bool marked[N];
vector<PII> adj[N]; // adj[from] -> {weight, to}

int prim(int n, int src) {
  for(int i=1;i<=n;i++) marked[i] = false;
  priority_queue<PII, vector<PII>, greater<PII> > pq;
  int cost = 0;
  pq.push({0, src});
  while(!pq.empty()) {
    PII cur = pq.top(); pq.pop();
    if(marked[cur.second]) continue;
    cost += cur.first;
    marked[cur.second] = true;
    cc++;
    for(auto next : adj[cur.second]) {
      if(!marked[next.second]) pq.push(next);
    }
  }
  return cost;
}
```

4.5 LCA Related

```
using vec = vector <int>;
using mat = vector <vector <int>>;
namespace ta {
  int time;
  void dfs(int u, int p, vec &par, vec &lvl, Tree &T) {
    for(int e: T[u]) {
      int v = T(e).to(u);
      if(v == p) continue;
      par[v] = u, lvl[v] = lvl[u] + 1;
      dfs(v, u, par, lvl, T);
    }
```

```
  }
  mat ancestorTable(vec &par) {
    int n = par.size(), sz = __lg(n) + 1;
    mat anc(sz, par);
    for(int k = 1; k < sz; k++) {
      for(int i = 0; i < n; i++) {
          anc[k][i] = anc[k - 1][anc[k - 1][i]];
      }
    }
    return anc;
  }
  int getAncestor(int u, int ht, mat &anc) {
    int sz = anc.size();
    for (int k = 0; k < sz; k++) {
        if (ht >> k & 1) u = anc[u][k];
    }
    return u;
  }
  bool isAncestor(int u, int par, vec &st, vec &en) {
      return st[par] <= st[u] and en[par] >= en[u];
  }
  int subtreeSize(int u, vec &st, vec &en) {
      return en[u] - st[u] + 1 >> 1;
  }
  int lca(int u, int v, vec &lvl, mat &anc) {
    if (lvl[u] > lvl[v]) swap(u, v);
    for (int k = anc.size() - 1; ~k; k--) {
        if (lvl[u] + (1 << k) <= lvl[v]) v = anc[k][v];
    }
    if (u == v) return u;
    for (int k = anc.size() - 1; ~k; k--) {
        if (anc[k][u] != anc[k][v]) u = anc[k][u], v = anc[k][v];
    }
    return anc[0][u];
  }
  int dis(int u, int v, vec &lvl, mat &anc) {
    int g = lca(u, v, lvl, anc);
    return lvl[u] + lvl[v] - 2 * lvl[g];
  }
}
```

4.6 STRONGLY CONNECTED COMPONENT

```
vii adj[MAXN], rev_adj[MAXN], components[MAXN];
vii visited(MAXN), toposorted;
ll cnt = 0;
void dfs(ll now)
{
  visited[now] = 1;
  for(auto next : adj[now]) {
      if(!visited[next]) dfs(next);
  }
  toposorted.pb(now);
}
void dfs2(ll now){
  visited[now] = 1;
  components[cnt].pb(now);
  for(auto next : rev_adj[now]) {
      if(!visited[next]) dfs(next);
  }
}
void strongly_connected_component(ll n, ll m){
  visited.assign(n + 1, 0);
  for(ll i=0;i<m;i++) {
    ll a, b; cin >> a >> b;
    adj[a].pb(b);
    rev_adj[b].pb(a);
  }
  for(ll i=1;i<=n;i++) {
      if(!visited[i]) dfs(i);
  }
  reverse(all(toposorted));
  visited.assign(n + 1, 0);
  for(auto now : toposorted) {
      if(!visited[now]) {
          dfs2(now); cnt++;
      }
  }
}
```

5 String

5.1 Hashing

```
void generate_hash(const string &s){
  LL n = s.size(), p = 31, m = 1e9 + 9;
  vector<LL> p_pow(n), h(n + 1, 0);
  p_pow[0] = 1;
  for(LL i = 1; i < n; i++) p_pow[i] = (p_pow[i - 1] * p) % m;
  // generate
  for(int i = 0; i < n; i++) h[i + 1] = (h[i] + (s[i] - 'a' + 1) * p_pow[i]) % m;
}
LL get_substring_hash(LL l, LL r) {
  LL cur_h = (h[r] + m - h[l - 1]) % m;
  cur_h = (cur_h * p_pow[n - l - 1]) % m;
```

```
  return cur_h;
}
```

5.2 KMP

```
template<typename T>
class calckmp {
  void computelps(vector<int> &lps,T &b,int m){
    int len=0,i=1;
    lps[0]=0;
    while(i<m){
      if(b[i]==b[len]){
          len++; lps[i]=len; i++;
      }else{
        if(len) len = lps[len-1];
        else lps[i]=0, i++;
      }
    }
  }
public:
  int match(T &a,T &b){
    int n=a.size();
    int m=b.size();
    vector<int> lps(m);
    int cnt=0;
    computelps(lps,b,m);
    int i=0, j=0;
    while(n-i >= m-j){
      if(a[i]==b[j]) i++, j++;
      if(j==m){
        cnt++;
        j=lps[j-1];
      }else if(i<n && a[i] != b[j]){
        if(j) j=lps[j-1];
        else i++;
      }
    }
    return cnt;
  }
};
```

5.3 Manacher

```
vector<int. manacher(string str){
  int i, j, k, l=str.size(), n=l<<1;
  vector<int> pal(n);
  for(i=0,j=0,k=0; i<n; j=max(0,j-k),i+=k){
    while(j<=i && (i+j+1)<n && str[(i-j)>>1] == str(i+j+1)>>1]) j++;
    for(k=1, pal[i]=j; k<=i && k<=pal[i] && (pal[i]-k) != pal[i-k]; k++){
      pal[i+k] = min(pal[i-k],pal[i]-k);
    }
  }
  pal.pop_back();
  return pal;
}
```

5.4 Trie

```
struct node {
    bool endmark;
    node* next[26 + 1];
    node()
    {
        endmark = false;
        for (int i = 0; i < 26; i++)
            next[i] = NULL;
    }
} * root;
void insert(char* str, int len){
    node* curr = root;
    for (int i = 0; i < len; i++) {
        int id = str[i] - 'a';
        if (curr->next[id] == NULL)
            curr->next[id] = new node();
        curr = curr->next[id];
    }
    curr->endmark = true;
}
bool search(char* str, int len){
  node* curr = root;
  for (int i = 0; i < len; i++) {
    int id = str[i] - 'a';
    if (curr->next[id] == NULL) return false;
    curr = curr->next[id];
  }
  return curr->endmark;
}
void del(node* cur){
  for (int i = 0; i < 26; i++)
    if (cur->next[i]) del(cur->next[i]);
  delete (cur);
}
```

6 Divide and Conquer

6.1 Max Subarray Sum

```
void maxSubArraySum(int a[], int size)
{
    vector<int> dp(size, 0);
    dp[0] = a[0];
    int ans = dp[0];
    for (int i = 1; i < size; i++) {
        dp[i] = max(a[i], a[i] + dp[i - 1]);
        ans = max(ans, dp[i]);
    }
    cout << ans;
}
```

7 DP

7.1 Coin Change

```
void coin(){ //given different types of coin how many way number x can be formed?
    int n,x,mod=1e9+7; cin>>n>>x;
    int a[n], dp[x+1]={};
    for(int i=0;i<n;i++){
        cin>>a[i];
        if(a[i]<=x) dp[a[i]]=1;
    }
    for(int i=1;i<=x;i++){
        for(int j=0;j<n;j++){
            if(i>= a[j]){
                dp[i] += dp[i-a[j]];
                dp[i] %= mod;
            }
        }
    }
    cout<<dp[x]<<ln;
}
```

7.2 Knapsack

```
/*
for 1 ---->
    1<=N<=100
    1<=W<=105
    1<=wi<=W
    1<=vi<=1e9
for 2 ---->
    1<=N<=100
    1<=W<=1e9
    1<=wi<=W
    1<=vi<=1e3
*/
int n,W,v[101],w[101],dp[101][N];

int ks(int i,int W){
    if(i>=n) return 0;
    if(dp[i][W]!=-1) return dp[i][W];
    if(W<w[i]) return dp[i][W]=ks(i+1,W);
    else return dp[i][W]=max(ks(i+1,W),ks(i+1,W-w[i])+v[i]);
}
void solve(){
    cin>>n>>W;
    for(int i=0;i<n;i++){
        cin>>w[i]>>v[i];
    }
    cout<<ks(0,W)<<ln;
}

int knpsk2(int i,int val){
    if(val==0) return 0;
    if(i>=n) return INT_MAX;
    if(dp[i][val]!=-1) return dp[i][val];
    int b = ks(i+1,val);
    if(val-v[i]>=0) b = min(b,ks(i+1,val-v[i]) + w[i]);
    return dp[i][val] = b;
}
void solve(){
    cin>>n>>W;
    for(int i=0;i<n;i++){
        cin>>w[i]>>v[i];
    }
    int ans=0, sm=accumulate(v,v+n,0LL);;
    for(int j=sm;j>=0;j--){
        if(ks(0,j)<=W){
            cout<<j<<ln;
            break;
        }
    }
}
```

7.3 SOS

```
/*
Given a fixed array A of 2^N integers, we need to calculate for all x function F(x) =
    Sum of all A[i] such that x&i = i, i.e., i is a subset of x.
*/
//iterative version
for(int mask = 0; mask < (1<<N); ++mask){
 dp[mask][-1] = A[mask]; //handle base case separately (leaf states)
```

```
 for(int i = 0;i < N; ++i){
  if(mask & (1<<i))
   dp[mask][i] = dp[mask][i-1] + dp[mask^(1<<i)][i-1];
  else
   dp[mask][i] = dp[mask][i-1];
 }
 F[mask] = dp[mask][N-1];
}
//memory optimized, super easy to code.
for(int i = 0; i<(1<<N); ++i)
 F[i] = A[i];
for(int i = 0;i < N; ++i) for(int mask = 0; mask < (1<<N); ++mask){
 if(mask & (1<<i))
  F[mask] += F[mask^(1<<i)];
}
```

8 Equations and Formulas

8.1 Catalan Numbers

$C_n = \frac{1}{n+1}\binom{2n}{n}$ $C_0 = 1, C_1 = 1$ and $C_n = \sum_{k=0}^{n-1} C_k C_{n-1-k}$

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 i.e. non-intersecting chords.
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.
Number of permutations of $1, \ldots, n$ that avoid the pattern 123 (or any of the other patterns of length 3); that is, the number of permutations with no three-term increasing sub-sequence. For $n = 3$, these permutations are 132, 213, 231, 312 and 321.

8.2 Stirling Numbers First Kind

The Stirling numbers of the first kind count permutations according to their number of cycles (counting fixed points as cycles of length one).
$S(n, k)$ counts the number of permutations of n elements with k disjoint cycles.
$S(n, k) = (n-1) \cdot S(n-1, k) + S(n-1, k-1),\ where,\ S(0, 0) = 1, S(n, 0) = S(0, n) = 0$ $\sum_{k=0}^{n} S(n, k) = n!$
The unsigned Stirling numbers may also be defined algebraically, as the coefficient of the rising factorial:

$$x^{\bar{n}} = x(x+1)...(x+n-1) = \sum_{k=0}^{n} S(n, k)x^k$$

Lets $[n, k]$ be the stirling number of the first kind, then

$$\left[{}_{n}\overset{n}{-}{}_{k}\right] = \sum_{0 \le i_1 < i_2 < i_k < n} i_1 i_2 .... i_k.$$

8.3 Stirling Numbers Second Kind

Stirling number of the second kind is the number of ways to partition a set of n objects into k non-empty subsets.
$S(n, k) = k \cdot S(n-1, k) + S(n-1, k-1),\ where\ S(0, 0) = 1, S(n, 0) = S(0, n) = 0$ $S(n, 2) = 2^{n-1} - 1$ $S(n, k) \cdot k! =$ number of ways to color n nodes using colors from 1 to k such that each color is used at least once.
An r-associated Stirling number of the second kind is the number of ways to partition a set of n objects into k subsets, with each subset containing at least r elements. It is denoted by $S_r(n, k)$ and obeys the recurrence relation. $S_r(n+1, k) = kS_r(n, k) + \binom{n}{r-1} S_r(n-r+1, k-1)$
Denote the n objects to partition by the integers $1, 2, \ldots, n$. Define the reduced Stirling numbers of the second kind, denoted $S^d(n, k)$, to be the number of ways to partition the integers $1, 2, \ldots, n$ into k nonempty subsets such that all elements in each subset have pairwise distance at least d. That is, for any integers i and j in a given subset, it is required that $|i - j| \ge d$. It has been shown that these numbers satisfy, $S^d(n, k) = S(n-d+1, k-d+1), n \ge k \ge d$

8.4 Other Combinatorial Identities

$\binom{n}{k} = \frac{n}{k}\binom{n-1}{k-1}$

$\sum_{i=0}^{k} \binom{n+i}{i} = \sum_{i=0}^{k} \binom{n+i}{n} = \binom{n+k+1}{k}$

$n, r \in N, n > r, \sum_{i=r}^{n} \binom{i}{r} = \binom{n+1}{r+1}$

If $P(n) = \sum_{k=0}^{n} \binom{n}{k} \cdot Q(k)$, then,

$$Q(n) = \sum_{k=0}^{n} (-1)^{n-k} \binom{n}{k} \cdot P(k)$$

If $P(n) = \sum_{k=0}^{n} (-1)^k \binom{n}{k} \cdot Q(k)$, then,

$$Q(n) = \sum_{k=0}^{n} (-1)^k \binom{n}{k} \cdot P(k)$$

8.5 Different Math Formulas

Picks Theorem : $A = i + b/2 - 1$
Deragements : $d(i) = (i-1) \times (d(i-1) + d(i-2))$

$$\frac{n}{ab} \quad - \quad \left\{\frac{b'n}{a}\right\} \quad - \quad \left\{\frac{a'n}{b}\right\} \quad + \quad 1$$

8.6 GCD and LCM

if m is any integer, then $\gcd(a + m \cdot b, b) = \gcd(a, b)$
The gcd is a multiplicative function in the following sense: if a_1 and a_2 are relatively prime, then $\gcd(a_1 \cdot a_2, b) = \gcd(a_1, b) \cdot \gcd(a_2, b)$.
$\gcd(a, \mathrm{lcm}(b, c)) = \mathrm{lcm}(\gcd(a, b), \gcd(a, c))$.
$\mathrm{lcm}(a, \gcd(b, c)) = \gcd(\mathrm{lcm}(a, b), \mathrm{lcm}(a, c))$.
For non-negative integers a and b, where a and b are not both zero, $\gcd(n^a - 1, n^b - 1) = n^{\gcd(a,b)} - 1$

$\gcd(a, b) = \sum_{k|a \text{ and } k|b} \phi(k)$

$\sum_{i=1}^{n} [\gcd(i, n) = k] = \phi\left(\frac{n}{k}\right)$

$\sum_{k=1}^{n} \gcd(k, n) = \sum_{d|n} d \cdot \phi\left(\frac{n}{d}\right)$

$\sum_{k=1}^{n} x^{\gcd(k,n)} = \sum_{d|n} x^d \cdot \phi\left(\frac{n}{d}\right)$

$\sum_{k=1}^{n} \frac{1}{\gcd(k, n)} = \sum_{d|n} \frac{1}{d} \cdot \phi\left(\frac{n}{d}\right) = \frac{1}{n} \sum_{d|n} d \cdot \phi(d)$

$\sum_{k=1}^{n} \frac{k}{\gcd(k, n)} = \frac{n}{2} \cdot \sum_{d|n} \frac{1}{d} \cdot \phi\left(\frac{n}{d}\right) = \frac{n}{2} \cdot \frac{1}{n} \cdot \sum_{d|n} d \cdot \phi(d)$

$\sum_{k=1}^{n} \frac{n}{\gcd(k, n)} = 2 * \sum_{k=1}^{n} \frac{k}{\gcd(k, n)} - 1$, for $n > 1$

$\sum_{i=1}^{n} \sum_{j=1}^{n} [\gcd(i, j) = 1] = \sum_{d=1}^{n} \mu(d) \lfloor \frac{n}{d} \rfloor^2$

$\sum_{i=1}^{n} \sum_{j=1}^{n} \gcd(i, j) = \sum_{d=1}^{n} \phi(d) \lfloor \frac{n}{d} \rfloor^2$

$\sum_{i=1}^{n} \sum_{j=1}^{n} i \cdot j [\gcd(i, j) = 1] = \sum_{i=1}^{n} \phi(i) i^2$

$F(n) = \sum_{i=1}^{n} \sum_{j=1}^{n} \mathrm{lcm}(i, j) = \sum_{l=1}^{n} \left(\frac{\left(1 + \lfloor \frac{n}{l} \rfloor\right)\left(\lfloor \frac{n}{l} \rfloor\right)}{2} \right)^2 \sum_{d|l} \mu(d) l d$