Fast Fourier Transform(n log n)

```
#include<cassert>
#include<cstdio>
#include<cmath>
struct cpx
{
  cpx(){}
  cpx(double aa):a(aa),b(0){}
  cpx(double aa, double bb):a(aa),b(bb){}
  double a;
  double b;
  double modsq(void) const
  { return a * a + b * b; }
  cpx bar(void) const
  { return cpx(a, -b); }
};
cpx operator +(cpx a, cpx b)
{ return cpx(a.a + b.a, a.b + b.b); }
cpx operator *(cpx a, cpx b)
{ return cpx(a.a * b.a - a.b * b.b, a.a * b.b + a.b * b.a); }
cpx operator /(cpx a, cpx b)
{cpx r = a * b.bar();
return cpx(r.a / b.modsq(), r.b / b.modsq()); }
cpx EXP(double theta)
{  return cpx(cos(theta),sin(theta)); }
const double two_pi = 4 * acos(0);
void FFT(cpx *in, cpx *out, int step, int size, int dir) {
  if(size < 1) return;
  if(size == 1){
    out[0] = in[0];
    return;
  }
  FFT(in, out, step * 2, size / 2, dir);
  FFT(in + step, out + size / 2, step * 2, size / 2, dir);
  for(int i = 0 ; i < size / 2 ; i++)  {
    cpx even = out[i];
    cpx odd = out[i + size / 2];
    out[i] = even + EXP(dir * two_pi * i / size) * odd;
    out[i + size / 2] = even + EXP(dir * two_pi * (i + size /
2) / size) * odd;
  }
}
```

```
int main(void)
{
  printf("If rows come in identical pairs, then everything
works.\n");
  cpx a[8] = {0, 1, cpx(1,3), cpx(0,5), 1, 0, 2, 0};
  cpx b[8] = {1, cpx(0,-2), cpx(0,1), 3, -1, -3, 1, -2};
  cpx A[8];
  cpx B[8];
  FFT(a, A, 1, 8, 1);
  FFT(b, B, 1, 8, 1);
  for(int i = 0 ; i < 8 ; i++)
    printf("%7.2lf%7.2lf", A[i].a, A[i].b);
  printf("\n");
  for(int i = 0 ; i < 8 ; i++) {
    cpx Ai(0,0);
    for(int j = 0 ; j < 8 ; j++)
      Ai = Ai + a[j] * EXP(j * i * two_pi / 8);
    printf("%7.2lf%7.2lf", Ai.a, Ai.b);
  }
  printf("\n");
  cpx AB[8];
  for(int i = 0 ; i < 8 ; i++)
    AB[i] = A[i] * B[i];
  cpx aconvb[8];
  FFT(AB, aconvb, 1, 8, -1);
  for(int i = 0 ; i < 8 ; i++)
    aconvb[i] = aconvb[i] / 8;
  for(int i = 0 ; i < 8 ; i++){
    printf("%7.2lf%7.2lf", aconvb[i].a, aconvb[i].b);
  }
  printf("\n");
  for(int i = 0 ; i < 8 ; i++){
    cpx aconvbi(0,0);
    for(int j = 0 ; j < 8 ; j++)
      aconvbi = aconvbi + a[j] * b[(8 + i - j) % 8];
    printf("%7.2lf%7.2lf", aconvbi.a, aconvbi.b);
  }
  printf("\n");
  return 0;
}
```

Min Cut

```
// Number of vertices in given graph
```

```
#define V 6
int bfs(int rGraph[V][V], int s, int t, int parent[]){
    bool visited[V];
    memset(visited, 0, sizeof(visited));
    queue <int> q;
    q.push(s);
    visited[s] = true;
    parent[s] = -1;
    while (!q.empty()){
        int u = q.front();
        q.pop();
        for (int v=0; v<V; v++){
            if (visited[v]==false && rGraph[u][v] > 0){
                q.push(v);
                parent[v] = u;
                visited[v] = true;
            }
        }
    }
    return (visited[t] == true);
}
void dfs(int rGraph[V][V], int s, bool visited[]){
    visited[s] = true;
    for (int i = 0; i < V; i++)
       if (rGraph[s][i] && !visited[i])
           dfs(rGraph, i, visited);
}
void minCut(int graph[V][V], int s, int t){
    int u, v;
    int rGraph[V][V]; // rGraph[i][j] indicates residual
capacity of edge i-j
    for (u = 0; u < V; u++)
        for (v = 0; v < V; v++)
             rGraph[u][v] = graph[u][v];
    int parent[V];
    while (bfs(rGraph, s, t, parent)){
        int path_flow = INT_MAX;
        for (v=t; v!=s; v=parent[v]){
            u = parent[v];
            path_flow = min(path_flow, rGraph[u][v]);
        }
        for (v=t; v != s; v=parent[v]) {
            u = parent[v];
            rGraph[u][v] -= path_flow;
            rGraph[v][u] += path_flow;
        }
    }
    bool visited[V];
    memset(visited, false, sizeof(visited));
    dfs(rGraph, s, visited);
    for (int i = 0; i < V; i++)
      for (int j = 0; j < V; j++)
         if (visited[i] && !visited[j] && graph[i][j])
              cout << i << " - " << j << endl;
    return;}
call by minCut(graph [V][V],source, dest);
```

Ford-Fulkerson algo for Max flow (m * E)

```
// Number of vertices in given graph
#define V 6
bool bfs(int rGraph[V][V], int s, int t, int parent[]){
    bool visited[V];
    memset(visited, 0, sizeof(visited));
    queue <int> q;
    q.push(s);``````````````````
    visited[s] = true;
    parent[s] = -1;
    while (!q.empty()){
        int u = q.front();
        q.pop();
        for (int v=0; v<V; v++){
            if (visited[v]==false && rGraph[u][v] > 0){
                q.push(v);
                parent[v] = u;
                visited[v] = true;
            } } }
    return (visited[t] == true);
}
int fordFulkerson(int graph[V][V], int s, int t){
    int u, v;
    int rGraph[V][V];
    for (u = 0; u < V; u++)
        for (v = 0; v < V; v++)
             rGraph[u][v] = graph[u][v];
    int parent[V];
```

```
    int max_flow = 0;
    while (bfs(rGraph, s, t, parent)){
        int path_flow = INT_MAX;
        for (v=t; v!=s; v=parent[v]){
            u = parent[v];
            path_flow = min(path_flow, rGraph[u][v]);
        }
        for (v=t; v != s; v=parent[v]){
            u = parent[v];
            rGraph[u][v] -= path_flow;
            rGraph[v][u] += path_flow;
        }
        max_flow += path_flow;
    }
    return max_flow;
}
call by fordFulkerson(graph[V][V], source, dest);
```

Maximum Bipartite Matching (O(V*E))

```
#define M 6
#define N 6
bool bpm(bool bpGraph[M][N], int u, bool seen[], int matchR[]){
    for (int v = 0; v < N; v++){
        if (bpGraph[u][v] && !seen[v]){
            seen[v] = true;
            if (matchR[v] < 0 || bpm(bpGraph, matchR[v], seen,
matchR)){
                matchR[v] = u;
                return true;
            }
        }
    }
    return false;
}
int maxBPM(bool bpGraph[M][N]){
    int matchR[N];
    memset(matchR, -1, sizeof(matchR));
    int result = 0;
    for (int u = 0; u < M; u++){
        bool seen[N];
        memset(seen, 0, sizeof(seen));
        if (bpm(bpGraph, u, seen, matchR))
            result++;
    }
    return result;
}
call by maxBPM(graph[M][N]);
```

Convex Hull (Graham Scan) = O(n log n)

```
class Point    {
public:
    int x, y;
    bool operator < (Point b) {
        if (y != b.y)
            return y < b.y;
        return x < b.x;
    }
};
Point pivot;
int ccw(Point a, Point b, Point c) {
    int area = (b.x - a.x) * (c.y - a.y) - (b.y - a.y) * (c.x -
a.x);
    if (area > 0)
        return -1;
    else if (area < 0)
        return 1;
    return 0;
}
int sqrDist(Point a, Point b)  {
    int dx = a.x - b.x, dy = a.y - b.y;
    return dx * dx + dy * dy;
}
bool POLAR_ORDER(Point a, Point b)  {
    int order = ccw(pivot, a, b);
    if (order == 0)
        return sqrDist(pivot, a) < sqrDist(pivot, b);
    return (order == -1);
}

stack<Point> grahamScan(Point *points, int N)    {
    stack<Point> hull;

    if (N < 3)
        return hull;
    int leastY = 0;
    for (int i = 1; i < N; i++)
```

```
        if (points[i] < points[leastY])
            leastY = i;
    Point temp = points[0];
    points[0] = points[leastY];
    points[leastY] = temp;
    pivot = points[0];
    sort(points + 1, points + N, POLAR_ORDER);
    hull.push(points[0]);
    hull.push(points[1]);
    hull.push(points[2]);
    for (int i = 3; i < N; i++) {
        Point top = hull.top();
        hull.pop();
        while (ccw(hull.top(), top, points[i]) != -1)   {
            top = hull.top();
            hull.pop();
        }
        hull.push(top);
        hull.push(points[i]);
    }
    return hull;
}
int main()  {
    Point points[] = {{0, 0}, {1, 1}, {2, 2}, {3, -1}};
    int N = sizeof(points)/sizeof(points[0]);

    stack<Point> hull = grahamScan(points, N);
    while (!hull.empty())   {
        Point p = hull.top();
        hull.pop();
        printf("(%d, %d)\n", p.x, p.y);
    }
    return 0;
}
```

KMP // 0-indexed, sp contains the starting points

```
#define sz 300005
int lps[sz]; string pat, main; vector <int> sp;
void fillLps()
{
      int l = pat.length();
      int len = 0;
      int i = 1;
      while(i < l)
      {
            if(pat[i] == pat[len])
                  len++, lps[i] = len, i++;
            else
            {
                  if(len != 0)
                        len = lps[len-1];
                  else
                        lps[i] = 0, i++;
            }
      }
}
void kmp()
{
      int m = pat.length();
      int n = main.length();
      int i = 0, j = 0;
      while(i < n)
      {
            if(pat[j] == main[i])
                  i++, j++;
            if(j == m)
                  sp.push(i-j), j = lps[j-1];
            else if(i < n and pat[j] != main[i])
            {
                  if(j != 0)
                        j = lps[j-1];
                  else
                        i++;
            }
      }
}
```

1Dsparse

```
const int k = 16;
const int N = 1e5;
const int ZERO = 1e9 + 1; // min(ZERO, x) = min(x, ZERO) = x
(for any x)

int table[N][k + 1]; // k + 1 because we need to access
table[r][k]
```

```
int Arr[N];
int n; //array size
void build() //All 0-indexed
{
    for(int i = 0; i < n ; i++)
        table[i][0] = Arr[i];
    for(int j = 1; j <= k; j++) {
        for(int i = 0; i <= n - (1 << j); i++)
            table[i][j] = min(table[i][j - 1], table[i + (1 <<
(j - 1))][j - 1]);
    }
}

void query(int L, int R)
{
    int answer = ZERO;
    for(int j = k; j >= 0; j--) {
        if(L + (1 << j) - 1 <= R) {
            answer = min(answer, table[L][j]);
            L += 1 << j;
        }
    }
    cout << answer << endl;
}
```

2Dsparse

```
#define sz 1003
int n,m,mat[sz][sz]; // 0-indexed
int sparse[sz][sz][12][12];

int get_max_four(int a,int b,int c,int d)
{
      return max(max(a,b),max(c,d));
}

void build_sparse(int n,int m)
{
      for(int x=0;x<n;x++)
            for(int y=0;y<m;y++)
                  sparse[x][y][0][0]=mat[x][y];
      for(int j=1;(1<<j) <= m;j++)
            for(int x=0;x<n;x++)
                  for(int y=0;y+(1<<j)-1<m;y++)
                        sparse[x][y][0][j]=max(sparse[x][y][0]
[j-1],sparse[x][y+(1<<(j-1))][0][j-1]);
      for(int i=1;(1<<i) <= n;i++)
            for(int x=0;x+(1<<i)-1<n;x++)
                  for(int y=0;y<m;y++)
                        sparse[x][y][i][0]=max(sparse[x][y][i-
1][0],sparse[x+(1<<(i-1))][y][i-1][0]);
      for(int i=1;(1<<i) <= n;i++)
            for(int j=1;(1<<j) <= m;j++)
                  for(int x=0;x+(1<<i)-1<n;x++)
                        for(int y=0;y+(1<<j)-1<m;y++)
                              sparse[x][y][i]
[j]=get_max_four(sparse[x][y][i-1][j-1],sparse[x+(1<<(i-1))][y]
[i-1][j-1],sparse[x][y+(1<<(j-1))][i-1][j-1],sparse[x+(1<<(i-
1))][y+(1<<(j-1))][i-1][j-1]);
}

int query_max(int x1,int y1,int x2,int y2)
{
      int k=pre_log[x2-x1+1];
      int l=pre_log[y2-y1+1];
      int ans=get_max_four(sparse[x1][y1][k][l],sparse[x2 -
(1<<k) + 1][y1][k][l],sparse[x1][y2 - (1<<l) + 1][k]
[l],sparse[x2 - (1<<k) + 1][y2 - (1<<l) + 1][k][l]);
      return ans;
}
```

Binary indexed tree

```
int BIT[1000], a[1000], n; //1-indexed BIT
void update(int x, int delta)
{
      for(; x <= n; x += x&-x)
        BIT[x] += delta; //increment by delta
}
int query(int x)
{
     int sum = 0;
     for(; x > 0; x -= x&-x)
        sum += BIT[x];
     return sum;
}
```

Date

```
string dayOfWeek[] = {"Mon", "Tue", "Wed", "Thu", "Fri", "Sat",
"Sun"};
// converts Gregorian date to integer (Julian day number)
int dateToInt (int m, int d, int y){
  return
    1461 * (y + 4800 + (m - 14) / 12) / 4 +
    367 * (m - 2 - (m - 14) / 12 * 12) / 12 -
    3 * ((y + 4900 + (m - 14) / 12) / 100) / 4 +
    d - 32075;
}
// converts integer (Julian day number) to Gregorian date:
month/day/year
void intToDate (int jd, int &m, int &d, int &y){
  int x, n, i, j;

  x = jd + 68569;
  n = 4 * x / 146097;
  x -= (146097 * n + 3) / 4;
  i = (4000 * (x + 1)) / 1461001;
  x -= 1461 * i / 4 - 31;
  j = 80 * x / 2447;
  d = x - 2447 * j / 80;
  x = j / 11;
  m = j + 2 - 12 * x;
  y = 100 * (n - 49) + i + x;
}
// converts integer (Julian day number) to day of week
string intToDay (int jd){
  return dayOfWeek[jd % 7];
}
int main (int argc, char **argv){
  int jd = dateToInt (3, 24, 2004);
  int m, d, y;
  intToDate (jd, m, d, y);
  string day = intToDay (jd);

  // expected output:
  //    2453089
  //    3/24/2004
  //    Wed
  cout << jd << endl
    << m << "/" << d << "/" << y << endl
    << day << endl;
}
```

Eucledian

```
typedef vector<int> VI;
typedef pair<int,int> PII;

// return a % b (positive value)
int mod(int a, int b) {
  return ((a%b)+b)%b;
}

// computes gcd(a,b)
int gcd(int a, int b) {
  int tmp;
  while(b){a%=b; tmp=a; a=b; b=tmp;}
  return a;
}

// computes lcm(a,b)
int lcm(int a, int b) {
  return a/gcd(a,b)*b;
}

// returns d = gcd(a,b); finds x,y such that d = ax + by
int extended_euclid(int a, int b, int &x, int &y) {
  int xx = y = 0;
  int yy = x = 1;
  while (b) {
    int q = a/b;
    int t = b; b = a%b; a = t;
    t = xx; xx = x-q*xx; x = t;
    t = yy; yy = y-q*yy; y = t;
  }
  return a;
}

// finds all solutions to ax = b (mod n)
VI modular_linear_equation_solver(int a, int b, int n) {
```

```
  int x, y;
  VI solutions;
  int d = extended_euclid(a, n, x, y);
  if (!(b%d)) {
    x = mod (x*(b/d), n);
    for (int i = 0; i < d; i++)
      solutions.push_back(mod(x + i*(n/d), n));
  }
  return solutions;
}

// computes b such that ab = 1 (mod n), returns -1 on failure
int mod_inverse(int a, int n) {
  int x, y;
  int d = extended_euclid(a, n, x, y);
  if (d > 1) return -1;
  return mod(x,n);
}

// Chinese remainder theorem (special case): find z such that
// z % x = a, z % y = b.  Here, z is unique modulo M =
lcm(x,y).
// Return (z,M).  On failure, M = -1.
PII chinese_remainder_theorem(int x, int a, int y, int b) {
  int s, t;
  int d = extended_euclid(x, y, s, t);
  if (a%d != b%d) return make_pair(0, -1);
  return make_pair(mod(s*b*x+t*a*y,x*y)/d, x*y/d);
}

// Chinese remainder theorem: find z such that
// z % x[i] = a[i] for all i.  Note that the solution is
// unique modulo M = lcm_i (x[i]).  Return (z,M).  On
// failure, M = -1.  Note that we do not require the a[i]'s
// to be relatively prime.
PII chinese_remainder_theorem(const VI &x, const VI &a) {
  PII ret = make_pair(a[0], x[0]);
  for (int i = 1; i < x.size(); i++) {
    ret = chinese_remainder_theorem(ret.second, ret.first,
x[i], a[i]);
    if (ret.second == -1) break;
  }
  return ret;
}

// computes x and y such that ax + by = c; on failure, x = y =-
1
void linear_diophantine(int a, int b, int c, int &x, int &y) {
  int d = gcd(a,b);
  if (c%d) {
    x = y = -1;
  } else {
    x = c/d * mod_inverse(a/d, b/d);
    y = (c-a*x)/b;
  }
}

int main() {

  // expected: 2
  cout << gcd(14, 30) << endl;

  // expected: 2 -2 1
  int x, y;
  int d = extended_euclid(14, 30, x, y);
  cout << d << " " << x << " " << y << endl;

  // expected: 95 45
  VI sols = modular_linear_equation_solver(14, 30, 100);
  for (int i = 0; i < (int) sols.size(); i++) cout << sols[i]
<< " ";
  cout << endl;

  // expected: 8
  cout << mod_inverse(8, 9) << endl;

  // expected: 23 56
  //           11 12
  int xs[] = {3, 5, 7, 4, 6};
  int as[] = {2, 3, 2, 3, 5};
```

```
  PII ret = chinese_remainder_theorem(VI (xs, xs+3), VI(as,
as+3));
  cout << ret.first << " " << ret.second << endl;
  ret = chinese_remainder_theorem (VI(xs+3, xs+5), VI(as+3,
as+5));
  cout << ret.first << " " << ret.second << endl;

  // expected: 5 -15
  linear_diophantine(7, 2, 5, x, y); //7x + 2y = 5
  cout << x << " " << y << endl;

}
```

FastIO

```
template <typename T>
inline void fi(T *a)
{
    register char c=0;
    while (c<33) c=getchar_unlocked();
    *a=0;
    int tmp = 0;
    while (c>33)
    {
        if ( c == 45 ) tmp = 1;
        else *a=*a*10+c-'0';
        c=getchar_unlocked();
    }
    if ( tmp == 1 ) *a = 0-(*a);
}
//usage : fi(&a);
```

Lazy propogation

```
//Lazy template for Range Sum Query and Range Update
const int mxn = 1e5;
const int height = ceil(log2(mxn)) + 1;
const int mx_tree = (1<<height);
ll arr[mxn+1], tree[mx_tree], lazy[mx_tree]; // 1-indexed arr
void build(ll node, ll start, ll end)
{
      if(start == end)
            tree[node] = arr[start];
      else
      {
            int mid = (start+end)/2;
            build(2*node, start, mid);
            build(2*node+1, mid+1, end);
            tree[node] = tree[2*node] + tree[2*node+1];
      }
}
//sample query - update_range(1, 1, n, l, r, val)
void update_range(ll node, ll start, ll end, ll l, ll r, ll
val)//[l-r]+=val
{
      if(lazy[node] != 0)
      {
            tree[node] += (end-start+1) * lazy[node];
            if(start != end)
                  lazy[2*node] += lazy[node], lazy[2*node+1]
+= lazy[node];
            lazy[node] = 0;
      }
      if(start>end || start>r || end<l) //outside
            return;
      if(start>=l && end<=r) //Completely inside
      {
            tree[node] += (end-start+1) * val;
            if(start != end)
                  lazy[2*node] += val, lazy[2*node+1] += val;
            return;
      }
      ll mid = (start+end)/2;
      update_range(2*node, start, mid, l, r, val);
      update_range(2*node+1, mid+1, end, l, r, val);
      tree[node] = tree[2*node] + tree[2*node+1];
}
//sample - query_range(1, 1, n, l, r)
ll query_range(ll node, ll start, ll end, ll l, ll r)//[l-r]Sum
{
      if(start>end || start>r || end<l)
            return 0;
      if(lazy[node] != 0)
      {
            tree[node] += (end-start+1) * lazy[node];
            if(start != end)
```

```
                  lazy[2*node] += lazy[node], lazy[2*node+1]
+= lazy[node];
            lazy[node] = 0;
      }
      if(start>=l && end<=r) //Completely inside
            return tree[node];
      ll mid = (start+end)/2;
      ll left = query_range(2*node, start, mid, l, r);
      ll right = query_range(2*node+1, mid+1, end, l, r);
      return (left+right);
}
```

LCA

```
#define LN 17  // <O(N logN, O(logN)>

int depth[100005], pa[LN][100005]; //0-indexed
int LCA(int u, int v) {
      if(depth[u] < depth[v]) swap(u,v);
      int diff = depth[u] - depth[v];
      for(int i=0; i<LN; i++) if( (diff>>i)&1 ) u = pa[i][u];
      if(u == v) return u;
      for(int i=LN-1; i>=0; i--) if(pa[i][u] != pa[i][v]) {
            u = pa[i][u];
            v = pa[i][v];
      }
      return pa[0][u];
}

void dfs(int cur, int prev, int _depth=0) {
      pa[0][cur] = prev;
      depth[cur] = _depth;
      for(int i=0; i<g[cur].size(); i++)
            if(g[cur][i] != prev){
                  g[cur][i];
                  dfs(g[cur][i], cur, _depth+1);
            }
}

main()
{
      dfs(root, -1);
      for(int i=1; i<LN; i++)
            for(int j=0; j<n; j++)
                  if(pa[i-1][j] != -1)
                        pa[i][j] = pa[i-1][pa[i-1][j]];
}
```

Manacher

```
#include<bits/stdc++.h>
using namespace std;
int manacher_algorithm(string x)
{
    string y="#";
    int l=x.length();
    int i=0;
    int len=0;
    int p[10001]={0}; //array for preprocessing
    int c=0,r=0,i_mirror=0;
    for(i=0;i<l;i++) //insert special character # between
characters of a string
    {
        y+=x[i];
        y+="#";
    }
    len=y.length();
    c=1,r=1,i_mirror=0;
    for(i=1;i<len;i++)
    {
        i_mirror=2*c-i;
        p[i]=(r>i)?min(r-i,p[i_mirror]):0;
        while(i-p[i]-1>=0 && p[i]+i-1<len && (y[p[i]+i-1]==y[i-
1-p[i]])) //palindrome expands past right edge
        {
            p[i]+=1;
        }
        if(i+p[i]>r) //reassgining center of palindrome at p[i]
        {
            r=p[i]+i;
            c=i;
        }
    }
    int max_val=INT_MIN;
    for(i=0;i<len;i++) //finding maximum value of p[i]
    {
        if(p[i]>max_val)
```

```
            max_val=p[i];
    }
    int counter=0;
    for(i=0;i<len;i+=1)  //finding number of occurences of
max_val in p[]
        if(p[i]==max_val)
            counter++;
    cout<<max_val-1<<" "<<counter<<endl; //print the length and
number of occurrences
}
```

Matrix exp

```
#define ITERATE_MATRIX(w) for(int r = 0; r < ( w ) ; ++r) \
                                                for(int c = 0; c <
( w ) ; ++c)
template < class T , int N >
struct M {
      vector < T > m[N];

      M () { ITERATE_MATRIX(N) m[r].pb(0); }

      static M id ()
      {
            M I ; for( int i = 0; i < N ; ++i ) I.m[i][i] = 1;
return I ;
      }

      M operator *( const M & rhs ) const
      {
            M out ;
            ITERATE_MATRIX(N) for(int i = 0; i < N ; ++i)
            out.m [r][c] += m[r][i] * rhs.m[i][c];
            return out ;
      }

      M raise ( ll n ) const
      {
            if ( n == 0) return id () ;
            if ( n == 1) return * this ;
            auto r = (* this ** this ) . raise ( n / 2) ;
            return ( n %2 ? * this * r : r ) ;
      }
};

int main()
{
      M <int, 2> obj;
      // Set values
      auto x = obj.raise(2);
      for(int i = 0 ; i < 2 ; i++)
            for(int j = 0 ; j < 2 ; j++)
                  cout << x.m[i][j] << " ";
      return 0;
}
```

Suffix array

```
/*
Suffix array O(n lg^2 n)
LCP table O(n)
*/
#define REP(i, n) for (int i = 0; i < (int)(n); ++i)

const int MAXN = 1 << 21;
string S;
int N, gap;
int sa[MAXN], pos[MAXN], tmp[MAXN], lcp[MAXN];

bool sufCmp(int i, int j)
{
      if (pos[i] != pos[j])
            return pos[i] < pos[j];
      i += gap;
      j += gap;
      return (i < N && j < N) ? pos[i] < pos[j] : i > j;
}

void buildSA()
{
      N = S.length();//The suffix array for "banana" is {5, 3,
1, 0, 4, 2}
      REP(i, N) sa[i] = i, pos[i] = S[i];
      for (gap = 1;; gap *= 2)
```

```
	{
		sort(sa, sa + N, sufCmp);
		REP(i, N - 1) tmp[i + 1] = tmp[i] + sufCmp(sa[i],
sa[i + 1]);
		REP(i, N) pos[sa[i]] = tmp[i];
		if (tmp[N - 1] == N - 1) break;
	}
}

void buildLCP() // It stores the lengths of the LCP between
pairs of consecutive suffixes in the suffix array.
{
	for (int i = 0, k = 0; i < N; ++i) if (pos[i] != N - 1)
	{
		for (int j = sa[pos[i] + 1]; S[i + k] == S[j + k];)
		++k;
		lcp[pos[i]] = k;
		if (k)--k;
	}
}
```

Aho-Corasick

```
const int no_of_key = 5;
const int max_key = 100;
const int MAXS = no_of_key*max_key + 10; // Max number of
states in the matching machine.
                                // Should be equal to the sum of
the length of all keywords.
const int MAXC = 26; // Number of characters in the alphabet.
int out[MAXS];
int f[MAXS]; // Failure function
int g[MAXS][MAXC]; // Goto function, or -1 if fail.
int buildMatchingMachine(const vector<string> &words, char
lowestChar = 'a', char highestChar = 'z')
{
    memset(out, 0, sizeof out);
    memset(f, -1, sizeof f);
    memset(g, -1, sizeof g);
    int states = 1; // Initially, we just have the 0 state
    for (int i = 0; i < words.size(); ++i)
    {
        const string &keyword = words[i];
        int currentState = 0;
        for (int j = 0; j < keyword.size(); ++j)
        {
            int c = keyword[j] - lowestChar;
            if (g[currentState][c] == -1)  // Allocate a new
node
                g[currentState][c] = states++;
            currentState = g[currentState][c];
        }
        out[currentState] |= (1 << i); // There's a match of
keywords[i] at node currentState.
    }
    for (int c = 0; c < MAXC; ++c)
        if (g[0][c] == -1)
            g[0][c] = 0;
    queue<int> q;
    for (int c = 0; c <= highestChar - lowestChar; ++c)
        if (g[0][c] != -1 && g[0][c] != 0)
            f[g[0][c]] = 0, q.push(g[0][c]);
    while (q.size())
    {
        int state = q.front();
        q.pop();
        for (int c = 0; c <= highestChar - lowestChar; ++c)
        {
            if (g[state][c] != -1)
            {
                int failure = f[state];
                while (g[failure][c] == -1)
                    failure = f[failure];
                failure = g[failure][c];
                f[g[state][c]] = failure;
                out[g[state][c]] |= out[failure]; // Merge out
values
                q.push(g[state][c]);
            }
        }
    }
    return states;
}
int findNextState(int currentState, char nextInput, char
lowestChar = 'a')
```

```
{
    int answer = currentState;
    int c = nextInput - lowestChar;
    while (g[answer][c] == -1) answer = f[answer];
    return g[answer][c];
}
int main()
{
    vector<string> keywords;
    keywords.push_back("he");
    keywords.push_back("she");
    keywords.push_back("hers");
    keywords.push_back("his");
    string text = "ahishers";
    buildMatchingMachine(keywords, 'a', 'z');
    int currentState = 0;
    for (int i = 0; i < text.size(); ++i)
    {
       currentState = findNextState(currentState, text[i],
'a');
       if (out[currentState] == 0) continue; // Nothing new,
let's move on to the next character.
       for (int j = 0; j < keywords.size(); ++j)
       {
           if (out[currentState] & (1 << j))
           { // Matched keywords[j]
               cout << "Keyword " << keywords[j] << " appears
from "
                    << i - keywords[j].size() + 1 << " to " <<
i << endl;
           }
       }
   }
   return 0;
}
```

LIS (n logn)

```
vector<int> v; //Take input in this

vector<int> ans; //Only for getting  the ans

int main()
{
    int n;
    cin>>n;
    for (int i = 0; i < n; i++) {
        int x;
        scanf("%d", &x);
        v.push_back(x);
    }
    for (int i = 0; i < n; i++)
    {
        int p = lower_bound(ans.begin(), ans.end(), v[i]) -
ans.begin();
        if (p == ans.size())
            ans.push_back(v[i]);
        else
            ans[p] = v[i];
    }
    for(int i=0;i<ans.size();i++)
        cout<<ans[i]<<" ";
    cout<<"\n";
    cout<<ans.size();
    return 0;
}//dp[i] = max(dp[j] + 1);//a[j] < a[i]
```

Djikstra

```
#define inf INT32_MAX;

typedef pair<int, int> P;

int* dijkstra(vector<P> *adj, int n, int s)
{
        int *dist;
        dist = new int[n+1];
        for(int i=1;i<=n;i++)
                dist[i] = inf;
        dist[s] = 0;
        queue<int> q;
        q.push(s);
        while(!q.empty())
```

```
        {
                int ver = q.front();
                q.pop();
                for(int i = 0; i<adj[ver].size(); i++)
                {
                        int nextv = adj[ver][i].first;
                        int cost = adj[ver][i].second;
                        if(dist[ver] + cost < dist[nextv])
                        {
                                dist[nextv] = dist[ver] + cost;
                                q.push(nextv);
                        }
                }
        }
        return dist;
}
```

Min Cost Bipartite matching

```
//////////////////////////////////////////////////////////////////
///////
// Min cost bipartite matching via shortest augmenting paths
//
// This is an O(n^3) implementation of a shortest augmenting
path
// algorithm for finding min cost perfect matchings in dense
// graphs.  In practice, it solves 1000x1000 problems in around
1
// second.
//
//   cost[i][j] = cost for pairing left node i with right node
j
//   Lmate[i] = index of right node that left node i pairs with
//   Rmate[j] = index of left node that right node j pairs with
//
// The values in cost[i][j] may be positive or negative.  To
perform
// maximization, simply negate the cost[][] matrix.
//////////////////////////////////////////////////////////////////
///////
typedef vector<double> VD;
typedef vector<VD> VVD;
typedef vector<int> VI;

double MinCostMatching(const VVD &cost, VI &Lmate, VI &Rmate) {
  int n = int(cost.size());
  // construct dual feasible solution
  VD u(n);
  VD v(n);
  for (int i = 0; i < n; i++) {
    u[i] = cost[i][0];
    for (int j = 1; j < n; j++) u[i] = min(u[i], cost[i][j]);
  }
  for (int j = 0; j < n; j++) {
    v[j] = cost[0][j] - u[0];
    for (int i = 1; i < n; i++) v[j] = min(v[j], cost[i][j] -
u[i]);
  }
  // construct primal solution satisfying complementary
slackness
  Lmate = VI(n, -1);
  Rmate = VI(n, -1);
  int mated = 0;
  for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++) {
      if (Rmate[j] != -1) continue;
      if (fabs(cost[i][j] - u[i] - v[j]) < 1e-10) {
      Lmate[i] = j;
      Rmate[j] = i;
      mated++;
      break;
      }
    }
  }
  VD dist(n);
  VI dad(n);
  VI seen(n);
  // repeat until primal solution is feasible
  while (mated < n) {
    // find an unmatched left node
    int s = 0;
    while (Lmate[s] != -1) s++;
    // initialize Dijkstra
```

```
    fill(dad.begin(), dad.end(), -1);
    fill(seen.begin(), seen.end(), 0);
    for (int k = 0; k < n; k++)
      dist[k] = cost[s][k] - u[s] - v[k];
    int j = 0;
    while (true) {
      // find closest
      j = -1;
      for (int k = 0; k < n; k++) {
      if (seen[k]) continue;
      if (j == -1 || dist[k] < dist[j]) j = k;
      }
      seen[j] = 1;

      // termination condition
      if (Rmate[j] == -1) break;

      // relax neighbors
      const int i = Rmate[j];
      for (int k = 0; k < n; k++) {
      if (seen[k]) continue;
      const double new_dist = dist[j] + cost[i][k] - u[i] -
v[k];
      if (dist[k] > new_dist) {
        dist[k] = new_dist;
        dad[k] = j;
      }
      }
    }
    // update dual variables
    for (int k = 0; k < n; k++) {
      if (k == j || !seen[k]) continue;
      const int i = Rmate[k];
      v[k] += dist[k] - dist[j];
      u[i] -= dist[k] - dist[j];
    }
    u[s] += dist[j];
    // augment along path
    while (dad[j] >= 0) {
      const int d = dad[j];
      Rmate[j] = Rmate[d];
      Lmate[Rmate[j]] = j;
      j = d;
    }
    Rmate[j] = s;
    Lmate[s] = j;

    mated++;
  }
  double value = 0;
  for (int i = 0; i < n; i++)
    value += cost[i][Lmate[i]];
  return value;
}
```

Push Relabel Flow(max flow)

```
// Adjacency list implementation of FIFO push relabel maximum
flow
// with the gap relabeling heuristic.  This implementation is
// significantly faster than straight Ford-Fulkerson.  It
solves
// random problems with 10000 vertices and 1000000 edges in a
few
// seconds, though it is possible to construct test cases that
// achieve the worst-case.
//
// Running time:
//     O(|V|^3)
//
// INPUT:
//     - graph, constructed using AddEdge()
//     - source
//     - sink
//
// OUTPUT:
//     - maximum flow value
//     - To obtain the actual flow values, look at all edges
with
//       capacity > 0 (zero capacity edges are residual edges).
typedef long long LL;
struct Edge {
  int from, to, cap, flow, index;
  Edge(int from, int to, int cap, int flow, int index) :
    from(from), to(to), cap(cap), flow(flow), index(index) {}
```

```
};
struct PushRelabel {
  int N;
  vector<vector<Edge> > G;
  vector<LL> excess;
  vector<int> dist, active, count;
  queue<int> Q;
  PushRelabel(int N) : N(N), G(N), excess(N), dist(N),
active(N), count(2*N) {}
  void AddEdge(int from, int to, int cap) {
    G[from].push_back(Edge(from, to, cap, 0, G[to].size()));
    if (from == to) G[from].back().index++;
    G[to].push_back(Edge(to, from, 0, 0, G[from].size() - 1));
  }
  void Enqueue(int v) {
    if (!active[v] && excess[v] > 0) { active[v] = true;
Q.push(v); }
  }
  void Push(Edge &e) {
    int amt = int(min(excess[e.from], LL(e.cap - e.flow)));
    if (dist[e.from] <= dist[e.to] || amt == 0) return;
    e.flow += amt;
    G[e.to][e.index].flow -= amt;
    excess[e.to] += amt;
    excess[e.from] -= amt;
    Enqueue(e.to);
  }
  void Gap(int k) {
    for (int v = 0; v < N; v++) {
      if (dist[v] < k) continue;
      count[dist[v]]--;
      dist[v] = max(dist[v], N+1);
      count[dist[v]]++;
      Enqueue(v);
    }
  }
  void Relabel(int v) {
    count[dist[v]]--;
    dist[v] = 2*N;
    for (int i = 0; i < G[v].size(); i++)
      if (G[v][i].cap - G[v][i].flow > 0)
      dist[v] = min(dist[v], dist[G[v][i].to] + 1);
    count[dist[v]]++;
    Enqueue(v);
  }
  void Discharge(int v) {
    for (int i = 0; excess[v] > 0 && i < G[v].size(); i++)
Push(G[v][i]);
    if (excess[v] > 0){
      if (count[dist[v]] == 1)
      Gap(dist[v]);
      else
      Relabel(v);
    }
  }
  LL GetMaxFlow(int s, int t) {
    count[0] = N-1;
    count[N] = 1;
    dist[s] = N;
    active[s] = active[t] = true;
    for (int i = 0; i < G[s].size(); i++) {
      excess[s] += G[s][i].cap;
      Push(G[s][i]);
    }
    while (!Q.empty()) {
      int v = Q.front();
      Q.pop();
      active[v] = false;
      Discharge(v);
    }
    LL totflow = 0;
    for (int i = 0; i < G[s].size(); i++) totflow += G[s]
[i].flow;
    return totflow;
  }
};
```

Dinic

```
// Adjacency list implementation of Dinic's blocking flow
algorithm.
// This is very fast in practice, and only loses to push-
relabel flow.
//
// Running time:
```

```
//     O(|V|^2 |E|)
//
// INPUT:
//     - graph, constructed using AddEdge()
//     - source and sink
//
// OUTPUT:
//     - maximum flow value
//     - To obtain actual flow values, look at edges with
capacity > 0
//       (zero capacity edges are residual edges).
typedef long long LL;
struct Edge {
  int from, to, cap, flow, index;
  Edge(int from, int to, int cap, int flow, int index) :
    from(from), to(to), cap(cap), flow(flow), index(index) {}
  LL rcap() { return cap - flow; }
};

struct Dinic {
  int N;
  vector<vector<Edge> > G;
  vector<vector<Edge *> > Lf;
  vector<int> layer;
  vector<int> Q;

  Dinic(int N) : N(N), G(N), Q(N) {}

  void AddEdge(int from, int to, int cap) {
    if (from == to) return;
    G[from].push_back(Edge(from, to, cap, 0, G[to].size()));
    G[to].push_back(Edge(to, from, 0, 0, G[from].size() - 1));
  }

  LL BlockingFlow(int s, int t) {
    layer.clear(); layer.resize(N, -1);
    layer[s] = 0;
    Lf.clear(); Lf.resize(N);

    int head = 0, tail = 0;
    Q[tail++] = s;
    while (head < tail) {
      int x = Q[head++];
      for (int i = 0; i < G[x].size(); i++) {
        Edge &e = G[x][i]; if (e.rcap() <= 0) continue;
        if (layer[e.to] == -1) {
          layer[e.to] = layer[e.from] + 1;
          Q[tail++] = e.to;
        }
        if (layer[e.to] > layer[e.from]) {
          Lf[e.from].push_back(&e);
        }
      }
    }
    if (layer[t] == -1) return 0;

    LL totflow = 0;
    vector<Edge *> P;
    while (!Lf[s].empty()) {
      int curr = P.empty() ? s : P.back()->to;
      if (curr == t) { // Augment
        LL amt = P.front()->rcap();
        for (int i = 0; i < P.size(); ++i) {
          amt = min(amt, P[i]->rcap());
        }
        totflow += amt;
        for (int i = P.size() - 1; i >= 0; --i) {
          P[i]->flow += amt;
          G[P[i]->to][P[i]->index].flow -= amt;
          if (P[i]->rcap() <= 0) {
            Lf[P[i]->from].pop_back();
            P.resize(i);
          }
        }
      } else if (Lf[curr].empty()) { // Retreat
        P.pop_back();
        for (int i = 0; i < N; ++i)
          for (int j = 0; j < Lf[i].size(); ++j)
            if (Lf[i][j]->to == curr)
              Lf[i].erase(Lf[i].begin() + j);
      } else { // Advance
        P.push_back(Lf[curr].back());
      }
    }
```

```
    return totflow;
  }

  LL GetMaxFlow(int s, int t) {
    LL totflow = 0;
    while (LL flow = BlockingFlow(s, t))
      totflow += flow;
    return totflow;
  }
};
```

Min Cost Max Flow

```
// Implementation of min cost max flow algorithm using
adjacency
// matrix (Edmonds and Karp 1972).  This implementation keeps
track of
// forward and reverse edges separately (so you can set cap[i]
[j] !=
// cap[j][i]).  For a regular max flow, set all edge costs to
0.
//
// Running time, O(|V|^2) cost per augmentation
//     max flow:           O(|V|^3) augmentations
//     min cost max flow:  O(|V|^4 * MAX_EDGE_COST)
augmentations
//
// INPUT:
//     - graph, constructed using AddEdge()
//     - source
//     - sink
//
// OUTPUT:
//     - (maximum flow value, minimum cost value)
//     - To obtain the actual flow, look at positive values
only.
typedef vector<int> VI;
typedef vector<VI> VVI;
typedef long long L;
typedef vector<L> VL;
typedef vector<VL> VVL;
typedef pair<int, int> PII;
typedef vector<PII> VPII;

const L INF = numeric_limits<L>::max() / 4;

struct MinCostMaxFlow {
  int N;
  VVL cap, flow, cost;
  VI found;
  VL dist, pi, width;
  VPII dad;

  MinCostMaxFlow(int N) :
    N(N), cap(N, VL(N)), flow(N, VL(N)), cost(N, VL(N)),
    found(N), dist(N), pi(N), width(N), dad(N) {}

  void AddEdge(int from, int to, L cap, L cost) {
    this->cap[from][to] = cap;
    this->cost[from][to] = cost;
  }

  void Relax(int s, int k, L cap, L cost, int dir) {
    L val = dist[s] + pi[s] - pi[k] + cost;
    if (cap && val < dist[k]) {
      dist[k] = val;
      dad[k] = make_pair(s, dir);
      width[k] = min(cap, width[s]);
    }
  }

  L Dijkstra(int s, int t) {
    fill(found.begin(), found.end(), false);
    fill(dist.begin(), dist.end(), INF);
    fill(width.begin(), width.end(), 0);
    dist[s] = 0;
    width[s] = INF;

    while (s != -1) {
      int best = -1;
      found[s] = true;
      for (int k = 0; k < N; k++) {
        if (found[k]) continue;
        Relax(s, k, cap[s][k] - flow[s][k], cost[s][k], 1);
```

```
        Relax(s, k, flow[k][s], -cost[k][s], -1);
        if (best == -1 || dist[k] < dist[best]) best = k;
      }
      s = best;
    }

    for (int k = 0; k < N; k++)
      pi[k] = min(pi[k] + dist[k], INF);
    return width[t];
  }

  pair<L, L> GetMaxFlow(int s, int t) {
    L totflow = 0, totcost = 0;
    while (L amt = Dijkstra(s, t)) {
      totflow += amt;
      for (int x = t; x != s; x = dad[x].first) {
        if (dad[x].second == 1) {
          flow[dad[x].first][x] += amt;
          totcost += amt * cost[dad[x].first][x];
        } else {
          flow[x][dad[x].first] -= amt;
          totcost -= amt * cost[x][dad[x].first];
        }
      }
    }
    return make_pair(totflow, totcost);
  }
};
```

Max Bipartite Matching

```
#define MAX 100001
#define NIL 0
#define INF (1<<28)
vector< int > G[MAX];
int n, m, match[MAX], dist[MAX];
// n: number of nodes on left side, nodes are numbered 1 to n
// m: number of nodes on right side, nodes are numbered n+1 to
n+m
// G = NIL[0] U G1[G[1---n]] U G2[G[n+1---n+m]]
```

```
bool bfs() {
  int i, u, v, len;
  queue< int > Q;
  for(i=1; i<=n; i++) {
      if(match[i]==NIL) {
          dist[i] = 0;
          Q.push(i);
      }
      else dist[i] = INF;
  }
  dist[NIL] = INF;
  while(!Q.empty()) {
      u = Q.front(); Q.pop();
      if(u!=NIL) {
          len = G[u].size();
          for(i=0; i<len; i++) {
              v = G[u][i];
              if(dist[match[v]]==INF) {
                  dist[match[v]] = dist[u] + 1;
                  Q.push(match[v]);
              }
          }
      }
  }
  return (dist[NIL]!=INF);
}

bool dfs(int u) {
  int i, v, len;
  if(u!=NIL) {
```

```
      len = G[u].size();
      for(i=0; i<len; i++) {
          v = G[u][i];
          if(dist[match[v]]==dist[u]+1) {
              if(dfs(match[v])) {
                  match[v] = u;
                  match[u] = v;
                  return true;
              }
          }
      }
      dist[u] = INF;
      return false;
  }
  return true;
}

int hopcroft_karp() {
  int matching = 0, i;
  // match[] is assumed NIL for all vertex in G
  while(bfs())
      for(i=1; i<=n; i++)
          if(match[i]==NIL && dfs(i))
              matching++;
  return matching;
}
```

Fenwick Tree with range updates

```
#define LSOne(S) (S & (-S))
typedef long long ll;
// B1 and B2 are two fenwick trees
// Original array entries are assumed to be 0
// and only updates are stored.
ll B1[100005], B2[100005];
// Array size
int N;
// Point query
// Returns value at position b in the array for ft = B1
// Returns value to be subtracted from query(B1, b) * b for ft
= B2
ll query(ll* ft, int b)    {
   ll sum = 0;
   for (; b; b -= LSOne(b)) sum += ft[b];
   return sum;
}
// Range query: Returns the sum of all elements in [1...b]
ll query(int b) {
      return query(B1, b) * b - query(B2, b);
}
// Range query: Returns the sum of all elements in [i...j]
ll range_query(int i, int j)   {
      return query(j) - query(i - 1);
}
// Point update: Adds v to the value at position k in the array
// ft is the fenwick tree which represents that array
void update(ll* ft, int k, ll v) {
   for (; k <= N; k += LSOne(k)) ft[k] += v;
}
// Range update: Adds v to each element in [i...j]
void range_update(int i, int j, ll v)    {
   update(B1, i, v);
   update(B1, j + 1, -v);
```

```
    update(B2, i, v * (i - 1));
    update(B2, j + 1, -v * j);
}
int main()  {
    int T, C, p, q, cmd;
    ll v;
    scanf("%d", &T);
    while (T--)    {
        // C -> No. of operations
        scanf("%d %d", &N, &C);
        memset(B1, 0, (N+1) * sizeof(ll));
        memset(B2, 0, (N+1) * sizeof(ll));
        while (C--)    {
              scanf("%d %d %d", &cmd, &p, &q);
              // cmd is 0 for a range update and 1 for a range
query
              if (cmd == 0)   {
                          scanf("%lld", &v);
                          range_update(p, q, v);
              } else
                          printf("%lld\n", range_query(p, q));
        }
    }
       return 0;
}
```

Order Statistic Tree

```
typedef long long ll;
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;
typedef tree<
double,
int,
less<double>,
rb_tree_tag,
tree_order_statistics_node_update> map_t;
int main() {
      map_t s;
      s.insert(make_pair(12, 1012));
      s.insert(make_pair(505, 1505));
      s.insert(make_pair(30, 1030));
      cout << s.find_by_order(1)->second << '\n';
      return 0;
}
```

Fibonacci

```
#define REP(i,n) for (int i = 1; i <= n; i++)
typedef long long ll;
map<long long, long long> F;

ll m=1000000007;
long long f(long long n) {
        if (F.count(n))
                return F[n];
        long long k = n / 2;
        if (n % 2 == 0) { // n=2*k
                return F[n] = (f(k) * f(k) + f(k - 1) * f(k -
1)) % m;
        } else { // n=2*k+1
                return F[n] = (f(k) * f(k + 1) + f(k - 1) *
f(k)) % m;
        }
}
int main()
{
    F[0] = F[1] = 1;
    ll n;cin >> n; // This answers the term n
    cout << f(n-1);
```

```
}
```

Segment Tree with Index

```
#define MN INT32_MIN
#define MX INT32_MAX
#define minindex(i,j) (arr[mintree[i]]<arr[mintree[j]]) ? (i):
(j)
#define maxindex(i,j) (arr[maxtree[i]]>arr[maxtree[j]]) ? (i):
(j)
const int mxn = 1e5;
const int height = ceil(log2(mxn)) + 1;
const int mx_tree = (1<<height);
int arr[mxn+5];
int maxtree[(mx_tree+5], mintree[mx_tree+5];
void buildmax(int node,int start, int end)
{
        if(start==end)
                maxtree[node]=start;
        else
        {
                int mid=(start+end)/2;
                buildmax(2*node, start, mid);
                buildmax(2*node+1, mid+1, end);

maxtree[node]=maxtree[maxindex(2*node,2*node+1)];
        }
}
void updatemax(int node, int start, int end, int idx, int val)
{
        if(start==end)
                arr[idx]=val, maxtree[node]=start;
        else
        {
                int mid=(start+end)/2;
                if(start<=idx && idx<=mid)
                        updatemax(2*node, start, mid, idx,
val);
                else
                        updatemax(2*node+1, mid+1, end, idx,
val);
                maxtree[node]=maxtree[maxindex(2*node,
2*node+1)];
        }
}
int querymax(int node, int start, int end, int l, int r)
{
        if(r<start || l>end)
                return 0;
        if(l<=start && r>=end)
                return node;
        int mid=(start+end)/2;
        int lt=querymax(2*node, start, mid, l, r);
        int rt=querymax(2*node+1, mid+1, end, l, r);
        return maxindex(lt,rt);
}
//For Maxtree, arr[0] = MN, maxtree[0] = 0
//For Mintree, arr[mxn+1] = MX, mintree[0] = mxn+1 & replace
maxindex with minindex
```

Rabin Miller Primality test

```
#define s(n)                            scanf("%lld",&n)

#define sc(n)                           scanf("%c",&n)
#define sl(n)                           scanf("%lld",&n)
#define sf(n)                           scanf("%Lf",&n)
#define ss(n)                           scanf("%s",n)
#define maX(a,b)                         ( (a) > (b) ? (a) : (b))
// Useful constants
#define INF                             (int)1e9
#define EPS                             1e-9

// Useful hardware instructions
#define bitcount                        __builtin_popcount
#define gcd                             __gcd

// Useful container manipulation / traversal macros
#define forall(i,a,b)                   for(long long i=a;i<b;i++)
#define foreach(v, c)                   for( typeof( (c).begin()) v
= (c).begin();  v != (c).end(); ++v)
#define all(a)                          a.begin(), a.end()
```

```
#define in(a,b)                         ( (b).find(a) != (b).end())
#define pb                              push_back
#define fill(a,v)                        memset(a, v, sizeof a)
#define sz(a)                           ((int)(a.size()))
#define mp                              make_pair
int abse(int a)
{
    if(a>0)return a;
    return -a;
}
typedef  long long ULL;
ULL mulmod(ULL a, ULL b, ULL c){
        ULL x = 0,y = a%c;

        while(b>0){
                if(b&1) x = (x+y)%c;
                y = (y<<1)%c;
                b >>= 1;
        }

        return x;
}

ULL pow(ULL a, ULL b, ULL c){
        ULL x = 1, y = a;

        while(b>0){
                if(b&1) x = mulmod(x,y,c);
                y = mulmod(y,y,c);
                b >>= 1;
        }

        return x;
}
bool isPrime(ULL p, int it){
        if(p<2) return false;
        if(p==2) return true;
        if((p&1)==0) return false;

        ULL s = p-1;
        while(s%2==0) s >>= 1;

        while(it--){
                ULL a = rand()%(p-1)+1,temp = s;
                ULL mod = pow(a,temp,p);

                if(mod==-1 || mod==1) continue;

                while(temp!=p-1 && mod!=p-1){
                        mod = mulmod(mod,mod,p);
                        temp <<= 1;
                }

                if(mod!=p-1) return false;
        }

        return true;
}
int main()
{
    ULL test,n;
    test=10000;
    s(test);
    forall(i,0,test)
    {
        s(n);
        {if(isPrime(n,5))
        printf("%lld\n",n);
    }
    return 0;
}
```

Iterative Trie

```
#define end _end

#define next _nxt

const int MaxN = 500500;

int sz = 0;
```

```
int next[27][MaxN];
int end[MaxN];
bool created[MaxN];
void insert (string &s) {
        int v = 0;
        for (int i = 0; i < s.size(); ++i) {
                int c = s[i] - 'a';
                if (!created[next[c][v]]) {
                        next[c][v] = ++sz;
                        created[sz] = true;
                }
                v = next[c][v];
        }
        ++end[v];
}
bool search (string tmp) {
        int v = 0;

        for (int i = 0; i < tmp.size(); ++i) {
                int c = tmp[i] - 'a';
                if (!created[next[c][v]])
                        return false;
                v = next[c][v];
        }
        return end[v] > 0;
}
int main () {
        string keys[] = {"hi", "hello", "you", "ekta", "me"};
        string output[] = {"NO", "YES"};

        for (int i = 0; i < 5; ++i)
                insert (keys[i]);

        cout << output[search ("my")] << endl;
        cout << output[search ("me")] << endl;

        return 0;
}
```

GaussJordan.cc 13/27

```
// Gauss-Jordan elimination with full pivoting.

//
// Uses:
//   (1) solving systems of linear equations (AX=B)
//   (2) inverting matrices (AX=I)
//   (3) computing determinants of square matrices
//
// Running time: O(n^3)
//
// INPUT:    a[][] = an nxn matrix
//           b[][] = an nxm matrix
//
// OUTPUT:   X      = an nxm matrix (stored in b[][])
//           A^{-1} = an nxn matrix (stored in a[][])
//           returns determinant of a[][]

#include <iostream>
#include <vector>
#include <cmath>

using namespace std;

const double EPS = 1e-10;

typedef vector<int> VI;
typedef double T;
typedef vector<T> VT;
typedef vector<VT> VVT;

T GaussJordan(VVT &a, VVT &b) {
  const int n = a.size();
  const int m = b[0].size();
  VI irow(n), icol(n), ipiv(n);
  T det = 1;

  for (int i = 0; i < n; i++) {
    int pj = -1, pk = -1;
    for (int j = 0; j < n; j++) if (!ipiv[j])
```

```
      for (int k = 0; k < n; k++) if (!ipiv[k])
        if (pj == -1 || fabs(a[j][k]) > fabs(a[pj][pk])) { pj =
j; pk = k; }
    if (fabs(a[pj][pk]) < EPS) { cerr << "Matrix is singular."
<< endl; exit(0); }
    ipiv[pk]++;
    swap(a[pj], a[pk]);
    swap(b[pj], b[pk]);
    if (pj != pk) det *= -1;
    irow[i] = pj;
    icol[i] = pk;

    T c = 1.0 / a[pk][pk];
    det *= a[pk][pk];
    a[pk][pk] = 1.0;
    for (int p = 0; p < n; p++) a[pk][p] *= c;
    for (int p = 0; p < m; p++) b[pk][p] *= c;
    for (int p = 0; p < n; p++) if (p != pk) {
      c = a[p][pk];
      a[p][pk] = 0;
      for (int q = 0; q < n; q++) a[p][q] -= a[pk][q] * c;
      for (int q = 0; q < m; q++) b[p][q] -= b[pk][q] * c;
    }
  }

  for (int p = n-1; p >= 0; p--) if (irow[p] != icol[p]) {
    for (int k = 0; k < n; k++) swap(a[k][irow[p]], a[k]
[icol[p]]);
  }

  return det;
}

int main() {
  const int n = 4;
  const int m = 2;
  double A[n][n] = { {1,2,3,4},{1,0,1,0},{5,3,2,4},{6,1,4,6} };
  double B[n][m] = { {1,2},{4,3},{5,6},{8,7} };
  VVT a(n), b(n);
  for (int i = 0; i < n; i++) {
    a[i] = VT(A[i], A[i] + n);
    b[i] = VT(B[i], B[i] + m);
  }

  double det = GaussJordan(a, b);

  // expected: 60
  cout << "Determinant: " << det << endl;

  // expected: -0.233333 0.166667 0.133333 0.0666667
  //           0.166667 0.166667 0.333333 -0.333333
  //           0.233333 0.833333 -0.133333 -0.0666667
  //           0.05 -0.75 -0.1 0.2
  cout << "Inverse: " << endl;
  for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++)
      cout << a[i][j] << ' ';
    cout << endl;
  }

  // expected: 1.63333 1.3
  //           -0.166667 0.5
  //           2.36667 1.7
  //           -1.85 -1.35
  cout << "Solution: " << endl;
  for (int i = 0; i < n; i++) {
    for (int j = 0; j < m; j++)
      cout << b[i][j] << ' ';
    cout << endl;
  }
}
```

Suffix Tree

```
#define fpos adla

const int inf = 1e9;
const int maxn = 1e4;
char s[maxn];
map<int, int> to[maxn];
int len[maxn], fpos[maxn], link[maxn];
int node, pos;
int sz = 1, n = 0;
```

```
int make_node(int _pos, int _len)
{
    fpos[sz] = _pos;
    len [sz] = _len;
    return sz++;
}

void go_edge()
{
    while(pos > len[to[node][s[n - pos]]])
    {
        node = to[node][s[n - pos]];
        pos -= len[node];
    }
}

void add_letter(int c)
{
    s[n++] = c;
    pos++;
    int last = 0;
    while(pos > 0)
    {
        go_edge();
        int edge = s[n - pos];
        int &v = to[node][edge];
        int t = s[fpos[v] + pos - 1];
        if(v == 0)
        {
            v = make_node(n - pos, inf);
            link[last] = node;
            last = 0;
        }
        else if(t == c)
        {
            link[last] = node;
            return;
        }
        else
        {
            int u = make_node(fpos[v], pos - 1);
            to[u][c] = make_node(n - 1, inf);
            to[u][t] = v;
            fpos[v] += pos - 1;
            len [v] -= pos - 1;
            v = u;
            link[last] = u;
            last = u;
        }
        if(node == 0)
            pos--;
        else
            node = link[node];
    }
}

int main()
{
    len[0] = inf;
    string s;
    cin >> s;
    int ans = 0;
    for(int i = 0; i < s.size(); i++)
        add_letter(s[i]);
    for(int i = 1; i < sz; i++)
        ans += min((int)s.size() - fpos[i], len[i]);
    cout << ans << "\n";
}
```