Stanford University ICPC Team Notebook (2015-16)

Contents

1 Numerical algorithms

1.1 Sieve

```
Input: an integer n > 1.
Let A be an array of Boolean values, indexed by integers 2 to n,
initially all set to true.

for i = 2, 3, 4, ..., not exceeding sqrt(n)
  if A[i] is true:
    for j = i2, i2+i, i2+2i, i2+3i, ..., not exceeding n:
      A[j] := false.

Output: all i such that A[i] is true.
```

1.2 Number theory (modular, Chinese remainder, linear Diophantine)

```
// This is a collection of useful code for solving problems that
// involve modular linear equations.  Note that all of the
// algorithms described here work on nonnegative integers.

#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

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) {
        while (b) { int t = a%b; a = b; b = t; }
        return a;
}

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

// (a^b) mod m via successive squaring
int powermod(int a, int b, int m)
{
        int ret = 1;
        while (b)
        {
                if (b & 1) ret = mod(ret*a, m);
                a = mod(a*a, m);
                b >>= 1;
        }
        return ret;
}

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

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

// Chinese remainder theorem (special case): find z such that
// z % m1 = r1, z % m2 = r2.  Here, z is unique modulo M = lcm(m1, m2).
// Return (z, M).  On failure, M = -1.
PII chinese_remainder_theorem(int m1, int r1, int m2, int r2) {
        int s, t;
```

```
        int g = extended_euclid(m1, m2, s, t);
        if (r1%g != r2%g) return make_pair(0, -1);
        return make_pair(mod(s*r2*m1 + t*r1*m2, m1*m2) / g, m1*m2 / g);
}

// Chinese remainder theorem: find z such that
// z % m[i] = r[i] for all i.  Note that the solution is
// unique modulo M = lcm_i (m[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 &m, const VI &r) {
        PII ret = make_pair(r[0], m[0]);
        for (int i = 1; i < m.size(); i++) {
                ret = chinese_remainder_theorem(ret.second, ret.first, m[i], r[i]);
                if (ret.second == -1) break;
        }
        return ret;
}

// computes x and y such that ax + by = c
// returns whether the solution exists
bool linear_diophantine(int a, int b, int c, int &x, int &y) {
        if (!a && !b)
        {
                if (c) return false;
                x = 0; y = 0;
                return true;
        }
        if (!a)
        {
                if (c % b) return false;
                x = 0; y = c / b;
                return true;
        }
        if (!b)
        {
                if (c % a) return false;
                x = c / a; y = 0;
                return true;
        }
        int g = gcd(a, b);
        if (c % g) return false;
        x = c / g * mod_inverse(a / g, b / g);
        y = (c - a*x) / b;
        return true;
}

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

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

        // expected: 95 451
        VI sols = modular_linear_equation_solver(14, 30, 100);
        for (int i = 0; i < sols.size(); i++) cout << sols[i] << " ";
        cout << endl;

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

        // expected: 23 105
        //           11 12
        PII ret = chinese_remainder_theorem(VI({ 3, 5, 7 }), VI({ 2, 3, 2 }));
        cout << ret.first << " " << ret.second << endl;
        ret = chinese_remainder_theorem(VI({ 4, 6 }), VI({ 3, 5 }));
        cout << ret.first << " " << ret.second << endl;

        // expected: 5 -15
        if (!linear_diophantine(7, 2, 5, x, y)) cout << "ERROR" << endl;
        cout << x << " " << y << endl;
        return 0;
}
```

1.3 Prime numbers

```
// O(sqrt(x)) Exhaustive Primality Test
#include <cmath>
#define EPS 1e-7
typedef long long LL;
bool IsPrimeSlow (LL x)
{
  if(x<=1) return false;
  if(x<=3) return true;
  if (!(x%2) || !(x%3)) return false;
  LL s=(LL)(sqrt((double)(x))+EPS);
  for(LL i=5;i<=s;i+=6)
  {
    if (!(x%i) || !(x%(i+2))) return false;
  }
  return true;
}
// Primes less than 1000:
//       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   101   103   107   109   113   127   131   137   139   149   151
//     157   163   167   173   179   181   191   193   197   199   211   223
//     227   229   233   239   241   251   257   263   269   271   277   281
//     283   293   307   311   313   317   331   337   347   349   353   359
//     367   373   379   383   389   397   401   409   419   421   431   433
//     439   443   449   457   461   463   467   479   487   491   499   503
//     509   521   523   541   547   557   563   569   571   577   587   593
//     599   601   607   613   617   619   631   641   643   647   653   659
//     661   673   677   683   691   701   709   719   727   733   739   743
//     751   757   761   769   773   787   797   809   811   821   823   827
//     829   839   853   857   859   863   877   881   883   887   907   911
//     919   929   937   941   947   953   967   971   977   983   991   997

// Other primes:
//    The largest prime smaller than 10 is 7.
//    The largest prime smaller than 100 is 97.
//    The largest prime smaller than 1000 is 997.
//    The largest prime smaller than 10000 is 9973.
//    The largest prime smaller than 100000 is 99991.
//    The largest prime smaller than 1000000 is 999983.
//    The largest prime smaller than 10000000 is 9999991.
//    The largest prime smaller than 100000000 is 99999989.
//    The largest prime smaller than 1000000000 is 999999937.
//    The largest prime smaller than 10000000000 is 9999999967.
//    The largest prime smaller than 100000000000 is 99999999977.
//    The largest prime smaller than 1000000000000 is 999999999989.
//    The largest prime smaller than 10000000000000 is 9999999999971.
//    The largest prime smaller than 100000000000000 is 99999999999973.
//    The largest prime smaller than 1000000000000000 is 999999999999989.
//    The largest prime smaller than 10000000000000000 is 9999999999999937.
//    The largest prime smaller than 100000000000000000 is 99999999999999997.
//    The largest prime smaller than 1000000000000000000 is 999999999999999989.
```

2 Data structures

2.1 Union-find set

```
int uf[MAX];

int find(int x) {
  if(uf[x] != x)
    uf[x] = find(uf[x]);
  return uf[x];
}
void un(int x, int y) {
  int x_ = find(x);
  int y_ = find(y);
  if(x_==y_) return;
  r[x_] = r[y_];
}

int main() {
  for(int i=0; i<n; i++) uf[i] = i;
  return 0;
}
```

2.2 Lowest common ancestor

```
const int max_nodes, log_max_nodes;
int num_nodes, log_num_nodes, root;

vector<int> children[max_nodes];        // children[i] contains the children of node i
int A[max_nodes][log_max_nodes+1];      // A[i][j] is the 2^j-th ancestor of node i, or -1 if that
      ancestor does not exist
int L[max_nodes];                       // L[i] is the distance between node i and the root

// floor of the binary logarithm of n
int lb(unsigned int n)
{
    if(n==0)
```

```
        return -1;
    int p = 0;
    if (n >= 1<<16) { n >>= 16; p += 16; }
    if (n >= 1<< 8) { n >>=  8; p +=  8; }
    if (n >= 1<< 4) { n >>=  4; p +=  4; }
    if (n >= 1<< 2) { n >>=  2; p +=  2; }
    if (n >= 1<< 1) {           p +=  1; }
    return p;
}

void DFS(int i, int l)
{
    L[i] = l;
    for(int j = 0; j < children[i].size(); j++)
        DFS(children[i][j], l+1);
}

int LCA(int p, int q)
{
    // ensure node p is at least as deep as node q
    if(L[p] < L[q])
        swap(p, q);

    // "binary search" for the ancestor of node p situated on the same level as q
    for(int i = log_num_nodes; i >= 0; i--)
        if(L[p] - (1<<i) >= L[q])
            p = A[p][i];

    if(p == q)
        return p;

    // "binary search" for the LCA
    for(int i = log_num_nodes; i >= 0; i--)
        if(A[p][i] != -1 && A[p][i] != A[q][i])
        {
            p = A[p][i];
            q = A[q][i];
        }

    return A[p][0];
}

int main(int argc,char* argv[])
{
    // read num_nodes, the total number of nodes
    log_num_nodes=lb(num_nodes);

    for(int i = 0; i < num_nodes; i++)
    {
        int p;
        // read p, the parent of node i or -1 if node i is the root

        A[i][0] = p;
        if(p != -1)
            children[p].push_back(i);
        else
            root = i;
    }

    // precompute A using dynamic programming
    for(int j = 1; j <= log_num_nodes; j++)
        for(int i = 0; i < num_nodes; i++)
            if(A[i][j-1] != -1)
                A[i][j] = A[A[i][j-1]][j-1];
            else
                A[i][j] = -1;

    // precompute L
    DFS(root, 0);


    return 0;
}
```

2.3 SegTree

```
t int MAX = 2e5+10;
template<typename T, typename OP> struct seg_tree{
    OP op;
    T *t;
    int n;
    int L, R;
    T *v;
    T build(int l, int r, int i){
        if (l == r) return t[i] = op(v[l]);
        int m = (l+r)/2;
        T lb = build(l, m, 2*i+1);
        T rb = build(m+1, r, 2*i+2);
        return t[i] = op(lb, rb);
    }
    T query(int l, int r, int i){
        if (L <= l && r <= R) return t[i];
        int m = (l+r)/2;
        if (m >= R) return query(l, m, 2*i+1);
        if (m < L) return query(m+1, r, 2*i+2);
        return op(query(l, m, 2*i+1), query(m+1, r, 2*i+2));
    }

    T query(int L_, int R_){
        L = L_;
        R = R_;
        return query(0, n-1, 0);
    }
    T update(int l, int r, int i, int k){
        if (l == r) return t[i] = op(v[k]);
        int m = (l+r)/2;
        if (m >= k) return t[i] = op(t[2*i+2], update(l, m, 2*i+1, k));
        if (m < k) return t[i] = op(t[2*i+1], update(m+1, r, 2*i+2, k));
        return t[i] = op(update(l, m, 2*i+1, k), update(m+1, r, 2*i+2, k));
    }
    T update(int k){
        return update(0, n-1, 0, k);
    }
    seg_tree(T *v_, int n_){
        t = new T[4*MAX+1];
        v = v_;
        n = n_;
        build(0, n-1, 0);
    }
    seg_tree(){}
};

// For range sum
struct sum_ll{
    ll operator()(const ll& l){return l;}
    ll operator()(const ll& l, const ll &r){return l+r;}
};

ll v[MAX];
seg_tree<ll, sum_ll> st(v);
```

3 DP

3.1 Max1DRangeSum

```
// Calcula 1D Max Range Sum
int sum = 0, ans = 0;
for(int i=0; i<n; i++) {
  sum += v[i];
  ans = max(ans, sum);
  if(sum < 0) sum = 0;
}
cout << ans << endl;
```

3.2 Max2DRangeSum

```
int max_area = 0;
for(int left = 0; left<s; left++){
  vector<int> temp(s, 0);
  for(int right = left; right<s; right++) {
    for(int i=0; i<s; i++)
      temp[i] += m[i][right];

    max_area = max(max_area, kadane(temp));
  }
}
cout << max_area << endl;
```

3.3 Longest increasing subsequence

```
#include <iostream>
#include <vector>
```

```
// Binary search (note boundaries in the caller)
int CeilIndex(std::vector<int> &v, int l, int r, int key) {
        while (r-l > 1) {
        int m = l + (r-l)/2;
        if (v[m] >= key)
                r = m;
        else
                l = m;
        }

        return r;
}

int LongestIncreasingSubsequenceLength(std::vector<int> &v) {
        if (v.size() == 0)
                return 0;

        std::vector<int> tail(v.size(), 0);
        int length = 1; // always points empty slot in tail

        tail[0] = v[0];
        for (size_t i = 1; i < v.size(); i++) {
                if (v[i] < tail[0])
                        // new smallest value
                        tail[0] = v[i];
                else if (v[i] > tail[length-1])
                        // v[i] extends largest subsequence
                        tail[length++] = v[i];
                else
                        // v[i] will become end candidate of an existing subsequence or
                        // Throw away larger elements in all LIS, to make room for upcoming grater
                              elements than v[i]
                        // (and also, v[i] would have already appeared in one of LIS, identify the
                              location and replace it)
                        tail[CeilIndex(tail, -1, length-1, v[i])] = v[i];
        }

        return length;
}

int main() {
        std::vector<int> v{ 2, 5, 3, 7, 11, 8, 10, 13, 6 };
        std::cout << "Length of Longest Increasing Subsequence is " <<
              LongestIncreasingSubsequenceLength(v) << 'n';
return 0;
}
```

3.4 01 Knapsack

```
/ A Dynamic Programming based solution for 0-1 Knapsack problem
#include<stdio.h>

// A utility function that returns maximum of two integers
int max(int a, int b) { return (a > b)? a : b; }

// Returns the maximum value that can be put in a knapsack of capacity W
int knapSack(int W, int wt[], int val[], int n)
{
   int i, w;
   int K[n+1][W+1];

   // Build table K[][] in bottom up manner
   for (i = 0; i <= n; i++)
   {
       for (w = 0; w <= W; w++)
       {
           if (i==0 || w==0)
               K[i][w] = 0;
           else if (wt[i-1] <= w)
                 K[i][w] = max(val[i-1] + K[i-1][w-wt[i-1]],  K[i-1][w]);
           else
                 K[i][w] = K[i-1][w];
       }
   }

   return K[n][W];
}

int main()
{
    int val[] = {60, 100, 120};
    int wt[] = {10, 20, 30};
    int  W = 50;
    int n = sizeof(val)/sizeof(val[0]);
    printf("%d", knapSack(W, wt, val, n));
    return 0;
}
```

3.5 CoinChange

```
// C program for coin change problem.
#include<stdio.h>

int count( int S[], int m, int n )
{
    int i, j, x, y;

    // We need n+1 rows as the table is constructed
    // in bottom up manner using the base case 0
    // value case (n = 0)
    int table[n+1][m];

    // Fill the enteries for 0 value case (n = 0)
    for (i=0; i<m; i++)
        table[0][i] = 1;

    // Fill rest of the table entries in bottom
    // up manner
    for (i = 1; i < n+1; i++)
    {
        for (j = 0; j < m; j++)
        {
            // Count of solutions including S[j]
            x = (i-S[j] >= 0)? table[i - S[j]][j]: 0;

            // Count of solutions excluding S[j]
            y = (j >= 1)? table[i][j-1]: 0;

            // total count
            table[i][j] = x + y;
        }
    }
    return table[n][m-1];
}

// Driver program to test above function
int main()
{
    int arr[] = {1, 2, 3};
    int m = sizeof(arr)/sizeof(arr[0]);
    int n = 4;
    printf(" %d ", count(arr, m, n));
    return 0;
}
```

4 Graph

4.1 Bipartite check

```
queue<int> q; q.push(s);
vi color(V, INF); color[s] = 0;
bool isBartite = true;
while(!q.empty() & isBipartite) {
  int u = q.front(); q.pop();
  for(int j=0; j<(int)Adj.size(); j++) {
    ii v = AdjList[u][j];
    if(color[v.first] == INF) {
      color[v.first] = 1 - color[u];
      q.push(v.first);
    }
    else if(color[v.first] == color[u]) {
      isBipartite = false;
      break;
    }
  }
}
```

4.2 TopoSort

```
vector<int> resp;
// rola priority queue tb
```

```
queue<ii> q;
for(int i=1; i<=n; i++)
  if(degree[i].first == 0)
    q.push(degree[i]);

while(!q.empty()) {
  ii e = q.front(); q.pop();
  resp.push_back(e.second);
  for(auto it : adj[e.second])
    if( --degree[it].first == 0 )
      q.push(degree[it]);
}
```

4.3 ArtiBridge

```
void articulationPointAndBridge(int u) {
  dfs_low[u] = dfs_num[u] = dfsNumberCounter++; // dfs_low[u] <= dfs_num[u]
        for (int j = 0; j < (int)AdjList[u].size(); j++) {
        ii v = AdjList[u][j];
    if (dfs_num[v.first] == UNVISITED) {
        dfs_parent[v.first] = u;
      if (u == dfsRoot) rootChildren++;

      articulationPointAndBridge(v.first);
      if (dfs_low[v.first] >= dfs_num[u])
                              articulation_vertex[u] = true;
      if (dfs_low[v.first] > dfs_num[u])
                printf(" Edge (%d, %d) is a bridge\n", u, v.first);

      dfs_low[u] = min(dfs_low[u], dfs_low[v.first]); // update dfs_low[u]
    }
                else if (v.first != dfs_parent[u]) // a back edge and not direct cycle
                        dfs_low[u] = min(dfs_low[u], dfs_num[v.first]); // update dfs_low[u]
  }
}
// inside int main()
dfsNumberCounter = 0; dfs_num.assign(V, UNVISITED); dfs_low.assign(V, 0);
dfs_parent.assign(V, 0); articulation_vertex.assign(V, 0);
printf("Bridges:\n");
for (int i = 0; i < V; i++) {
        if (dfs_num[i] == UNVISITED) {
        dfsRoot = i; rootChildren = 0; articulationPointAndBridge(i);
                articulation_vertex[dfsRoot] = (rootChildren > 1);
        }
}
printf("articulation points: \n");
for(int i=0; i < V; i++) {
        if(articulation_vertex[i])
                printf(" Vertex %d\n", i);
}
```

4.4 Tarjan

```
vi dfs_num, dfs_low, S, visited;
void tarjanSCC(int u) {
  dfs_low[u] = dfs_num[u] = dfsNumberCounter++; // dfs_low[u] <= dfs_num[u]
  S.push_back(u); // stores u in a vector based on order of visitation
  visited[u] = 1;
  for (int j = 0; j < (int)AdjList[u].size(); j++) {
    ii v = AdjList[u][j];
    if (dfs_num[v.first] == UNVISITED)
      tarjanSCC(v.first);
    if (visited[v.first]) // condition for update
      dfs_low[u] = min(dfs_low[u], dfs_low[v.first]); }
  if (dfs_low[u] == dfs_num[u]) { // if this is a root (start) of an SCC
    printf("SCC %d:", ++numSCC); // this part is done after recursion
    while (1) {
      int v = S.back(); S.pop_back(); visited[v] = 0;
      printf(" %d", v);
      if (u == v)
      break;
    }
    printf("\n");
  }
}

dfs_num.assign(V, UNVISITED); dfs_low.assign(V, 0); visited.assign(V, 0);
dfsNumberCounter = numSCC = 0;
for (int i = 0; i < V; i++)
        if (dfs_num[i] == UNVISITED)
                tarjanSCC(i);
```

4.5 Minimum spanning trees

```
vi taken;
priority_queue<ii> pq; // priority queue to help choose shorter edgei

process(int vtx) { // so, we use -ve sign to reverse the sort order
  taken[vtx] = 1;
  for (int j = 0; j < (int)AdjList[vtx].size(); j++) {
    ii v = AdjList[vtx][j];
    if (!taken[v.first]) pq.push(ii(-v.second, -v.first));
  }
}

taken.assign(V, 0);
process(0);
mst_cost = 0;
while(!pq.empty()) {
        ii front = pq.top(); pq.pop();
        u = -front.second, w = -front.first;
        if(!taken[u])
                mst_cost += w, process(u);
}
```

4.6 Kruskal's algorithm

```
/*
Uses Kruskal's Algorithm to calculate the weight of the minimum spanning
forest (union of minimum spanning trees of each connected component) of
a possibly disjoint graph, given in the form of a matrix of edge weights
(-1 if no edge exists). Returns the weight of the minimum spanning
forest (also calculates the actual edges - stored in T). Note: uses a
disjoint-set data structure with amortized (effectively) constant time per
union/find. Runs in O(E*log(E)) time.
*/

#include <iostream>
#include <vector>
#include <algorithm>
#include <queue>

using namespace std;

typedef int T;

struct edge
{
  int u, v;
  T d;
};

struct edgeCmp
{
  int operator()(const edge& a, const edge& b) { return a.d > b.d; }
};

int find(vector <int>& C, int x) { return (C[x] == x) ? x : C[x] = find(C, C[x]); }

T Kruskal(vector <vector <T> >& w)
{
  int n = w.size();
  T weight = 0;

  vector <int> C(n), R(n);
  for(int i=0; i<n; i++) { C[i] = i; R[i] = 0; }

  vector <edge> T;
  priority_queue <edge, vector <edge>, edgeCmp> E;

  for(int i=0; i<n; i++)
    for(int j=i+1; j<n; j++)
      if(w[i][j] >= 0)
      {
        edge e;
        e.u = i; e.v = j; e.d = w[i][j];
        E.push(e);
      }

  while(T.size() < n-1 && !E.empty())
  {
    edge cur = E.top(); E.pop();

    int uc = find(C, cur.u), vc = find(C, cur.v);
    if(uc != vc)
    {
```

```
      T.push_back(cur); weight += cur.d;

      if(R[uc] > R[vc]) C[vc] = uc;
      else if(R[vc] > R[uc]) C[uc] = vc;
      else { C[vc] = uc; R[uc]++; }
    }
  }

  return weight;
}

int main()
{
  int wa[6][6] = {
    { 0, -1, 2, -1, 7, -1 },
    { -1, 0, -1, 2, -1, -1 },
    { 2, -1, 0, -1, 8, 6 },
    { -1, 2, -1, 0, -1, -1 },
    { 7, -1, 8, -1, 0, 4 },
    { -1, -1, 6, -1, 4, 0 } };

  vector <vector <int> > w(6, vector <int>(6));

  for(int i=0; i<6; i++)
    for(int j=0; j<6; j++)
      w[i][j] = wa[i][j];

  cout << Kruskal(w) << endl;
  cin >> wa[0][0];
}
```

4.7 Fast Dijkstra's algorithm

```
// Implementation of Dijkstra's algorithm using adjacency lists
// and priority queue for efficiency.
//
// Running time: O(|E| log |V|)

vii adj[MAX];

memset(dist, inf, sizeof(dist));
priority_queue< ii, vector<ii>, greater<ii> > pq;
pq.push(ii(0,source));
while(!pq.empty()) {
  ii front = pq.top(); pq.pop();
  int d = front.first, u = front.second;
  if(d>dist[u]) continue;
  for(int j=0; j<adj[u].size(); j++) {
    ii v = adj[u][j];
    if(dist[u] + v.second < dist[v.first]) {
      dist[v.first] = dist[u] + v.second;
      pq.push(ii(dist[v.first], v.first));
    }
  }
}
```

4.8 Bellman-Ford shortest paths with negative edge weights

```
// This function runs the Bellman-Ford algorithm for single source
// shortest paths with negative edge weights.  The function returns
// false if a negative weight cycle is detected.  Otherwise, the
// function returns true and dist[i] is the length of the shortest
// path from start to i.
//
// Running time: O(|V|^3)
//
//   INPUT:   start, w[i][j] = cost of edge from i to j
//   OUTPUT:  dist[i] = min weight path from start to i
//            prev[i] = previous node on the best path from the
//                      start node

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

using namespace std;

typedef double T;
typedef vector<T> VT;

typedef vector<VT> VVT;

typedef vector<int> VI;
typedef vector<VI> VVI;

bool BellmanFord (const VVT &w, VT &dist, VI &prev, int start){
  int n = w.size();
  prev = VI(n, -1);
  dist = VT(n, 1000000000);
  dist[start] = 0;

  for (int k = 0; k < n; k++){
    for (int i = 0; i < n; i++){
      for (int j = 0; j < n; j++){
        if (dist[j] > dist[i] + w[i][j]){
          if (k == n-1) return false;
          dist[j] = dist[i] + w[i][j];
          prev[j] = i;
        }
      }
    }
  }

  return true;
}
```

4.9 FloydWarshall

```
for(int k=0; k < V; k++)
        for(int i=0; i < V; i++)
                for(int j=0; j < V; j++)
                        AdjMat[i][j] = min(AdjMat[i][j], Adj[i][k] + AdjMat[k][j]);
```

4.10 Eulerian path

```
struct Edge;
typedef list<Edge>::iterator iter;

struct Edge
{
        int next_vertex;
        iter reverse_edge;

        Edge(int next_vertex)
                :next_vertex(next_vertex)
                { }
};

const int max_vertices = ;
int num_vertices;
list<Edge> adj[max_vertices];           // adjacency list

vector<int> path;

void find_path(int v)
{
        while(adj[v].size() > 0)
        {
                int vn = adj[v].front().next_vertex;
                adj[vn].erase(adj[v].front().reverse_edge);
                adj[v].pop_front();
                find_path(vn);
        }
        path.push_back(v);
}

void add_edge(int a, int b)
{
        adj[a].push_front(Edge(b));
        iter ita = adj[a].begin();
        adj[b].push_front(Edge(a));
        iter itb = adj[b].begin();
        ita->reverse_edge = itb;
        itb->reverse_edge = ita;
}
```

5 NetFlow

5.1 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).

#include<cstdio>
#include<vector>
#include<queue>
using namespace std;
typedef long long LL;

struct Edge {
  int u, v;
  LL cap, flow;
  Edge() {}
  Edge(int u, int v, LL cap): u(u), v(v), cap(cap), flow(0) {}
};

struct Dinic {
  int N;
  vector<Edge> E;
  vector<vector<int>> g;
  vector<int> d, pt;

  Dinic(int N): N(N), E(0), g(N), d(N), pt(N) {}

  void AddEdge(int u, int v, LL cap) {
    if (u != v) {
      E.emplace_back(u, v, cap);
      g[u].emplace_back(E.size() - 1);
      E.emplace_back(v, u, 0);
      g[v].emplace_back(E.size() - 1);
    }
  }

  bool BFS(int S, int T) {
    queue<int> q({S});
    fill(d.begin(), d.end(), N + 1);
    d[S] = 0;
    while(!q.empty()) {
      int u = q.front(); q.pop();
      if (u == T) break;
      for (int k: g[u]) {
        Edge &e = E[k];
        if (e.flow < e.cap && d[e.v] > d[e.u] + 1) {
          d[e.v] = d[e.u] + 1;
          q.emplace(e.v);
        }
      }
    }
    return d[T] != N + 1;
  }

  LL DFS(int u, int T, LL flow = -1) {
    if (u == T || flow == 0) return flow;
    for (int &i = pt[u]; i < g[u].size(); ++i) {
      Edge &e = E[g[u][i]];
      Edge &oe = E[g[u][i]^1];
      if (d[e.v] == d[e.u] + 1) {
        LL amt = e.cap - e.flow;
        if (flow != -1 && amt > flow) amt = flow;
        if (LL pushed = DFS(e.v, T, amt)) {
          e.flow += pushed;
          oe.flow -= pushed;
          return pushed;
        }
      }
    }
    return 0;
  }

  LL MaxFlow(int S, int T) {
    LL total = 0;
    while (BFS(S, T)) {
      fill(pt.begin(), pt.end(), 0);
      while (LL flow = DFS(S, T))
        total += flow;
    }
    return total;
  }
};

// BEGIN CUT
// The following code solves SPOJ problem #4110: Fast Maximum Flow (FASTFLOW)

int main()
{
  int N, E;
  scanf("%d%d", &N, &E);
  Dinic dinic(N);
  for(int i = 0; i < E; i++)
  {
    int u, v;
    LL cap;
    scanf("%d%d%lld", &u, &v, &cap);
    dinic.AddEdge(u - 1, v - 1, cap);
    dinic.AddEdge(v - 1, u - 1, cap);
  }
  printf("%lld\n", dinic.MaxFlow(0, N - 1));
  return 0;
}

// END CUT
```

5.2 Karp

```
int res[MAX_V][MAX_V], mf, f, s, t; // global variables
vi p; // p stores the BFS spanning tree from s
void augment(int v, int minEdge) { // traverse BFS spanning tree from s->t
        if (v == s) { f = minEdge; return; } // record minEdge in a global var f
  else if (p[v] !+ -1) {
                augment(p[v], min(minEdge, res[p[v]][v]));
                resp[p[v]][v] -= f;
                res[v][p[v]] += f;
        }
}

mf = 0; // Max flow
while(1) {
        f = 0;
        vi dist(MAX_V, INF);
        dist[s] = 0;
        queue<int> q;
        q.push(s);
        p.assign(MAX_V, -1); // record the bfs spanning tree, from s to t
        while(!q.empty()) {
                int u = q.front(); q.pop();
                if(u==t) break; // stop if we already reach sick t
                for(int v = 0; v < MAX_V; v++) {
                        if(res[u][v] > 0 && dist[v] = INF)
                                dist[v] = dist[u] + 1; q.push(v), p[v] = u;
                }
        }
        augment(t, INF); // find the min edge weight f in this path if any
        if(f==0) break; // we cannot send any more flow ('f' = 0), terminate
        mf+=f;  // we can still send a flow, increase te max flow
}
```

6 String

6.1 Longest common subsequence

```
/*
Calculates the length of the longest common subsequence of two vectors.
Backtracks to find a single subsequence or all subsequences. Runs in
O(m*n) time except for finding all longest common subsequences, which
may be slow depending on how many there are.
*/

#include <iostream>
#include <vector>
#include <set>
```

```cpp
#include <algorithm>

using namespace std;

typedef int T;
typedef vector<T> VT;
typedef vector<VT> VVI;

typedef vector<int> VI;
typedef vector<VI> VVI;

void backtrack(VVI& dp, VT& res, VT& A, VT& B, int i, int j)
{
  if(!i || !j) return;
  if(A[i-1] == B[j-1]) { res.push_back(A[i-1]); backtrack(dp, res, A, B, i-1, j-1); }
  else
  {
    if(dp[i][j-1] >= dp[i-1][j]) backtrack(dp, res, A, B, i, j-1);
    else backtrack(dp, res, A, B, i-1, j);
  }
}

void backtrackall(VVI& dp, set<VT>& res, VT& A, VT& B, int i, int j)
{
  if(!i || !j) { res.insert(VI()); return; }
  if(A[i-1] == B[j-1])
  {
    set<VT> tempres;
    backtrackall(dp, tempres, A, B, i-1, j-1);
    for(set<VT>::iterator it=tempres.begin(); it!=tempres.end(); it++)
    {
      VT temp = *it;
      temp.push_back(A[i-1]);
      res.insert(temp);
    }
  }
  else
  {
    if(dp[i][j-1] >= dp[i-1][j]) backtrackall(dp, res, A, B, i, j-1);
    if(dp[i][j-1] <= dp[i-1][j]) backtrackall(dp, res, A, B, i-1, j);
  }
}

VT LCS(VT& A, VT& B)
{
  VVI dp;
  int n = A.size(), m = B.size();
  dp.resize(n+1);
  for(int i=0; i<=n; i++) dp[i].resize(m+1, 0);

  for(int i=1; i<=n; i++)
    for(int j=1; j<=m; j++)
    {
      if(A[i-1] == B[j-1]) dp[i][j] = dp[i-1][j-1]+1;
      else dp[i][j] = max(dp[i-1][j], dp[i][j-1]);
    }

  VT res;
  backtrack(dp, res, A, B, n, m);
  reverse(res.begin(), res.end());
  return res;
}

set<VT> LCSall(VT& A, VT& B)
{
  VVI dp;
  int n = A.size(), m = B.size();
  dp.resize(n+1);
  for(int i=0; i<=n; i++) dp[i].resize(m+1, 0);
  for(int i=1; i<=n; i++)
    for(int j=1; j<=m; j++)
    {
      if(A[i-1] == B[j-1]) dp[i][j] = dp[i-1][j-1]+1;
      else dp[i][j] = max(dp[i-1][j], dp[i][j-1]);
    }
  set<VT> res;
  backtrackall(dp, res, A, B, n, m);
  return res;
}

int main()
{
  int a[] = { 0, 5, 5, 2, 1, 4, 2, 3 }, b[] = { 5, 2, 4, 3, 2, 1, 2, 1, 3 };
  VI A = VI(a, a+8), B = VI(b, b+9);
  VI C = LCS(A, B);

  for(int i=0; i<C.size(); i++) cout << C[i] << " ";
  cout << endl << endl;

  set <VI> D = LCSall(A, B);
  for(set<VI>::iterator it = D.begin(); it != D.end(); it++)
  {
    for(int i=0; i<(*it).size(); i++) cout << (*it)[i] << " ";
    cout << endl;
  }
}
```

6.2 Edit Distance

```cpp
// A Dynamic Programming based C++ program to find minimum
// number operations to convert str1 to str2
#include<bits/stdc++.h>
using namespace std;

// Utility function to find the minimum of three numbers
int min(int x, int y, int z)
{
    return min(min(x, y), z);
}

int editDistDP(string str1, string str2, int m, int n)
{
    // Create a table to store results of subproblems
    int dp[m+1][n+1];

    // Fill d[][] in bottom up manner
    for (int i=0; i<=m; i++)
    {
        for (int j=0; j<=n; j++)
        {
            // If first string is empty, only option is to
            // isnert all characters of second string
            if (i==0)
                dp[i][j] = j;  // Min. operations = j

            // If second string is empty, only option is to
            // remove all characters of second string
            else if (j==0)
                dp[i][j] = i; // Min. operations = i

            // If last characters are same, ignore last char
            // and recur for remaining string
            else if (str1[i-1] == str2[j-1])
                dp[i][j] = dp[i-1][j-1];

            // If the last character is different, consider all
            // possibilities and find the minimum
            else
                dp[i][j] = 1 + min(dp[i][j-1],  // Insert
                                   dp[i-1][j],  // Remove
                                   dp[i-1][j-1]); // Replace
        }
    }

    return dp[m][n];
}

// Driver program
int main()
{
    // your code goes here
    string str1 = "sunday";
    string str2 = "saturday";

    cout << editDistDP(str1, str2, str1.length(), str2.length());

    return 0;
}
```

7 Miscellaneous

7.1 Hamiltonian Cycle

```cpp
/* C/C++ program for solution of Hamiltonian Cycle problem
   using backtracking */
#include<stdio.h>

// Number of vertices in the graph
#define V 5

void printSolution(int path[]);
```

```
/* A utility function to check if the vertex v can be added at
   index 'pos' in the Hamiltonian Cycle constructed so far (stored
   in 'path[]') */
bool isSafe(int v, bool graph[V][V], int path[], int pos)
{
    /* Check if this vertex is an adjacent vertex of the previously
       added vertex. */
    if (graph [ path[pos-1] ][ v ] == 0)
        return false;

    /* Check if the vertex has already been included.
      This step can be optimized by creating an array of size V */
    for (int i = 0; i < pos; i++)
        if (path[i] == v)
            return false;

    return true;
}

/* A recursive utility function to solve hamiltonian cycle problem */
bool hamCycleUtil(bool graph[V][V], int path[], int pos)
{
    /* base case: If all vertices are included in Hamiltonian Cycle */
    if (pos == V)
    {
        // And if there is an edge from the last included vertex to the
        // first vertex
        if ( graph[ path[pos-1] ][ path[0] ] == 1 )
            return true;
        else
          return false;
    }

    // Try different vertices as a next candidate in Hamiltonian Cycle.
    // We don't try for 0 as we included 0 as starting point in in hamCycle()
    for (int v = 1; v < V; v++)
    {
        /* Check if this vertex can be added to Hamiltonian Cycle */
        if (isSafe(v, graph, path, pos))
        {
            path[pos] = v;

            /* recur to construct rest of the path */
            if (hamCycleUtil (graph, path, pos+1) == true)
                return true;

            /* If adding vertex v doesn't lead to a solution,
               then remove it */
            path[pos] = -1;
        }
    }

    /* If no vertex can be added to Hamiltonian Cycle constructed so far,
       then return false */
    return false;
}

/* This function solves the Hamiltonian Cycle problem using Backtracking.
  It mainly uses hamCycleUtil() to solve the problem. It returns false
  if there is no Hamiltonian Cycle possible, otherwise return true and
  prints the path. Please note that there may be more than one solutions,
  this function prints one of the feasible solutions. */
bool hamCycle(bool graph[V][V])
{
    int *path = new int[V];
    for (int i = 0; i < V; i++)
        path[i] = -1;

    /* Let us put vertex 0 as the first vertex in the path. If there is
       a Hamiltonian Cycle, then the path can be started from any point
       of the cycle as the graph is undirected */
    path[0] = 0;
    if ( hamCycleUtil(graph, path, 1) == false )
    {
        printf("\nSolution does not exist");
        return false;
    }

    printSolution(path);
    return true;
}

/* A utility function to print solution */
void printSolution(int path[])
{
    printf ("Solution Exists:"
            " Following is one Hamiltonian Cycle \n");
    for (int i = 0; i < V; i++)
        printf(" %d ", path[i]);

    // Let us print the first vertex again to show the complete cycle
    printf(" %d ", path[0]);
    printf("\n");
}

// driver program to test above function
int main()
{
   /* Let us create the following graph
      (0)--(1)--(2)
       |   / \   |
       |  /   \  |
       | /     \ |
      (3)-------(4)    */
   bool graph1[V][V] = {{0, 1, 0, 1, 0},
                        {1, 0, 1, 1, 1},
                        {0, 1, 0, 0, 1},
                        {1, 1, 0, 0, 1},
                        {0, 1, 1, 1, 0},
                       };

    // Print the solution
    hamCycle(graph1);

   /* Let us create the following graph
      (0)--(1)--(2)
       |   / \   |
       |  /   \  |
       | /     \ |
      (3)       (4)    */
    bool graph2[V][V] = {{0, 1, 0, 1, 0},
                        {1, 0, 1, 1, 1},
                        {0, 1, 0, 0, 1},
                        {1, 1, 0, 0, 0},
                        {0, 1, 1, 0, 0},
                       };

    // Print the solution
    hamCycle(graph2);

    return 0;
}
```

7.2 Permutation

```
// An optimized version that uses reverse instead of sort for
// finding the next permutation

// A utility function to reverse a string str[l..h]
void reverse(char str[], int l, int h)
{
   while (l < h)
   {
       swap(&str[l], &str[h]);
       l++;
       h--;
   }
}

// Print all permutations of str in sorted order
void sortedPermutations ( char str[] )
{
    // Get size of string
    int size = strlen(str);

    // Sort the string in increasing order
    qsort( str, size, sizeof( str[0] ), compare );

    // Print permutations one by one
    bool isFinished = false;
    while ( ! isFinished )
    {
        // print this permutation
        printf ("%s \n", str);

        // Find the rightmost character which is smaller than its next
        // character. Let us call it 'first char'
        int i;
        for ( i = size - 2; i >= 0; --i )
           if (str[i] < str[i+1])
              break;

        // If there is no such chracter, all are sorted in decreasing order,
        // means we just printed the last permutation and we are done.
        if ( i == -1 )
            isFinished = true;
        else
        {
            // Find the ceil of 'first char' in right of first character.
            // Ceil of a character is the smallest character greater than it
```

```
            int ceilIndex = findCeil( str, str[i], i + 1, size - 1 );

            // Swap first and second characters
            swap( &str[i], &str[ceilIndex] );

            // reverse the string on right of 'first char'
            reverse( str, i + 1, size - 1 );
        }
    }
}
```

7.3 Lexicographic Rank

```
#include <stdio.h>
#include <string.h>

// A utility function to find factorial of n
int fact(int n)
{
    return (n <= 1)? 1 :n * fact(n-1);
}

// A utility function to count smaller characters on right
// of arr[low]
int findSmallerInRight(char* str, int low, int high)
{
    int countRight = 0, i;

    for (i = low+1; i <= high; ++i)
        if (str[i] < str[low])
            ++countRight;

    return countRight;
}

// A function to find rank of a string in all permutations
// of characters
int findRank (char* str)
{
    int len = strlen(str);
    int mul = fact(len);
    int rank = 1;
    int countRight;

    int i;
    for (i = 0; i < len; ++i)
    {
        mul /= len - i;

        // count number of chars smaller than str[i]
        // fron str[i+1] to str[len-1]
        countRight = findSmallerInRight(str, i, len-1);

        rank += countRight * mul ;
    }

    return rank;
}

// Driver program to test above function
int main()
{
    char str[] = "string";
    printf ("%d", findRank(str));
    return 0;
}
```

8 Pedro

8.1 exp matrix

```
#include<bits/stdc++.h>

using namespace std;

#define sd(a) scanf("%d", &a)
#define sd2(a,b) scanf("%d %d", &a, &b)
#define sd3(a,b,c) scanf("%d %d %d", &a, &b, &c)
#define sd4(a,b,c,d) scanf("%d %d %d %d", &a, &b, &c, &d)
#define sll(a) scanf("%lld", &a)
#define sll2(a,b) scanf("%lld %lld", &a, &b)
#define sll3(a,b,c) scanf("%lld %lld %lld", &a, &b, &c)
#define sll4(a,b,c,d) scanf("%lld %lld %lld %lld", &a, &b, &c, &d)
#define sc(a) scanf("%c", &a)
#define slf(a) scanf("%lf", &a)
#define slf2(a,b) scanf("%lf %lf", &a, &b)
#define slf3(a,b,c) scanf("%lf %lf %lf", &a, &b, &c)
#define slf4(a,b,c,d) scanf("%lf %lf %lf %lf", &a, &b, &c, &d)

#define FORN(i,n) for(int i = 0; i < n; i++)
#define all(v) v.begin(), v.end()
#define in(a,b) ( (b).find(a) != (b).end())
#define pb push_back
#define mp make_pair
#define fi first
#define se second

#define BUFF ios::sync_with_stdio(false);

#define MOD 1000000007
#define MAXD 2

typedef long long int lld;
typedef pair<int,int> ii;
typedef vector<int> vi;
typedef vector<ii> vii;

class MTX{
    public:
    int dim;
    lld M[MAXD][MAXD];
    MTX(int n, int k):dim(n){
        for(int i = 0; i < dim; i++)
        for(int j = 0; i < dim; i++)M[i][j] = k;
    }
    MTX operator *(const MTX &N) const{
        MTX r = MTX(dim,0);
        for(int i = 0; i < dim; i++){
            for(int j = 0; j < dim; j++){
                for(int k = 0; k < dim; k++){
                    r.M[i][j] += M[i][k]*N.M[k][j];
                    r.M[i][j] %= MOD;
                }
            }
        }
        return r;
    }

};

MTX ID = MTX(2,0);

MTX prodM(MTX M, int n){
    if(n == 0) return ID;
    else if(n == 1) return M;
    else if(n%2 == 0) return prodM(M*M, n/2);
    else return M*prodM(M*M, (n-1)/2);
}

int main(){
    MTX mtx = MTX(2,0);

    mtx.M[0][0] = 0;
    mtx.M[0][1] = 3;
    mtx.M[1][0] = 1;
    mtx.M[1][1] = 2;

    ID.M[0][0] = 1;
    ID.M[1][1] = 1;

    int n;
    sd(n);

    mtx = prodM(mtx,n);

    printf("%d\n",mtx.M[0][0]);
    return 0;
}
```

8.2 UF

```
#include<bits/stdc++.h>

using namespace std;

#define sd(a) scanf("%d", &a)
#define sd2(a,b) scanf("%d %d", &a, &b)
#define sd3(a,b,c) scanf("%d %d %d", &a, &b, &c)
```

```
#define sd4(a,b,c,d) scanf("%d %d %d %d", &a, &b, &c, &d)
#define sll(a) scanf("%lld", &a)
#define sll2(a,b) scanf("%lld %lld", &a, &b)
#define sll3(a,b,c) scanf("%lld %lld %lld", &a, &b, &c)
#define sll4(a,b,c,d) scanf("%lld %lld %lld %lld", &a, &b, &c, &d)
#define sc(a) scanf("%c", &a)
#define slf(a) scanf("%lf", &a)
#define slf2(a,b) scanf("%lf %lf", &a, &b)
#define slf3(a,b,c) scanf("%lf %lf %lf", &a, &b, &c)
#define slf4(a,b,c,d) scanf("%lf %lf %lf %lf", &a, &b, &c, &d)

#define FORN(i,n) for(int i = 0; i < n; i++)
#define all(v) v.begin(), v.end()
#define in(a,b) ( (b).find(a) != (b).end())
#define pb push_back
#define mp make_pair
#define fi first
#define se second

#define BUFF ios::sync_with_stdio(false);

#define MAXN 100100

typedef pair<int,int> ii;
typedef vector<int> vi;
typedef vector<ii> vii;

int parent[MAXN];
int Rank[MAXN];

void makeSets(int n){
    for(int i = 0; i < n; i++){
        parent[i] = i;
        Rank[i] = 0;
    }
}

int find(int i){
    if(parent[i] != i){
        parent[i] = find(parent[i]);
    }
    return parent[i];
}

bool Union(int x, int y){
    int xRoot = find(x);
    int yRoot = find(y);

    if(xRoot == yRoot) return false;

    if(Rank[x] < Rank[y]){
        parent[x] = parent[y];
    }
    else if(Rank[y] < Rank[x]){
        parent[y] = parent[x];
    }
    else{
        parent[y] = parent[x];
        Rank[x]++;
    }
    return true;
}

int main(){
    int n,m;
    sd2(n,m);
    makeSets(n);
    int u,v;
    bool possible = true;
    for(int i = 0; i < m; i++){
        sd2(u,v);
        u--;
        v--;
        if(!Union(u,v)) possible = false;
    }
    if(!possible) printf("NO\n");
    else{
        for(int i = 1; i < n; i++){
            if(parent[i] != parent[i-1]){
                possible = false;
                break;
            }
        }
        if(possible) printf("YES\n");
        else printf("NO\n");
    }
    return 0;
}
```

8.3 HELPASHU

```
#include<bits/stdc++.h>
#include <ext/numeric>

using namespace std;

#define sd(a) scanf("%d", &a)
#define sd2(a,b) scanf("%d %d", &a, &b)
#define sd3(a,b,c) scanf("%d %d %d", &a, &b, &c)
#define sd4(a,b,c,d) scanf("%d %d %d %d", &a, &b, &c, &d)
#define sll(a) scanf("%lld", &a)
#define sll2(a,b) scanf("%lld %lld", &a, &b)
#define sll3(a,b,c) scanf("%lld %lld %lld", &a, &b, &c)
#define sll4(a,b,c,d) scanf("%lld %lld %lld %lld", &a, &b, &c, &d)
#define sc(a) scanf("%c", &a)
#define slf(a) scanf("%lf", &a)
#define slf2(a,b) scanf("%lf %lf", &a, &b)
#define slf3(a,b,c) scanf("%lf %lf %lf", &a, &b, &c)
#define slf4(a,b,c,d) scanf("%lf %lf %lf %lf", &a, &b, &c, &d)

#define FORN(i,n) for(int i = 0; i < n; i++)
#define all(v) v.begin(), v.end()
#define in(a,b) ( (b).find(a) != (b).end())
#define pb push_back
#define mp make_pair
#define fi first
#define se second

#define BUFF ios::sync_with_stdio(false);

#define MAXN 100100

typedef long long int lld;
typedef pair<int,int> ii;
typedef vector<int> vi;
typedef vector<ii> vii;
typedef vector<vi> vvi;

int segTree[4*MAXN];
int v[MAXN];

void build(int node, int b, int e){
    if(b == e){
        segTree[node] = v[b]%2;
        return;
    }

    int m = (b+e)/2;

    build(2*node + 1, b, m);
    build(2*node + 2, m+1, e);

    segTree[node] = segTree[2*node + 1] + segTree[2*node + 2];
}

void update(int node, int b, int e, int idx, int value){
    if(b == e){
        v[idx] = value;
        segTree[node] = value%2;
        return;
    }

    int m = (b+e)/2;

    if(idx <= m){
        update(2*node + 1, b, m, idx, value);
    }
    else{
        update(2*node + 2, m+1, e, idx, value);
    }

    segTree[node] = segTree[2*node + 1] + segTree[2*node + 2];
}

int query(int node, int b, int e, int l, int r){
    if(b > e || b > r || e < l){
        return 0;
    }

    if(b >= l && e <= r) return segTree[node];

    int m = (b+e)/2;
    int qleft = query(2*node + 1, b, m, l, r);
    int qright = query(2*node + 2, m+1, e, l, r);

    return qleft + qright;
}
```

```
int main(){
    int n,q,l,r,o;

    sd(n);
    for(int i = 0; i < n; i++){
        sd(v[i]);
    }

    build(0, 0, n-1);

    sd(q);
    for(int i = 0; i < q; i++){
        sd3(o,l,r);

        if(o == 0){
            update(0,0, n-1, l-1, r);
        }
        else if(o == 1){
            printf("%d\n", (r-l+1) - query(0, 0, n-1, l-1, r-1));
        }
        else if(o == 2){
            printf("%d\n", query(0, 0, n-1, l-1, r-1));
        }
    }

    return 0;
}
```

8.4 ROYCOINB

```
#include<bits/stdc++.h>
#include <ext/numeric>

using namespace std;

#define sd(a) scanf("%d", &a)
#define sd2(a,b) scanf("%d %d", &a, &b)
#define sd3(a,b,c) scanf("%d %d %d", &a, &b, &c)
#define sd4(a,b,c,d) scanf("%d %d %d %d", &a, &b, &c, &d)
#define sll(a) scanf("%lld", &a)
#define sll2(a,b) scanf("%lld %lld", &a, &b)
#define sll3(a,b,c) scanf("%lld %lld %lld", &a, &b, &c)
#define sll4(a,b,c,d) scanf("%lld %lld %lld %lld", &a, &b, &c, &d)
#define sc(a) scanf("%c", &a)
#define slf(a) scanf("%lf", &a)
#define slf2(a,b) scanf("%lf %lf", &a, &b)
#define slf3(a,b,c) scanf("%lf %lf %lf", &a, &b, &c)
#define slf4(a,b,c,d) scanf("%lf %lf %lf %lf", &a, &b, &c, &d)

#define FORN(i,n) for(int i = 0; i < n; i++)
#define all(v) v.begin(), v.end()
#define in(a,b) ( (b).find(a) != (b).end())
#define pb push_back
#define mp make_pair
#define fi first
#define se second

#define BUFF ios::sync_with_stdio(false);

#define MAXN 1000100

typedef long long int lld;
typedef pair<int,int> ii;
typedef vector<int> vi;
typedef vector<ii> vii;
typedef vector<vi> vvi;

ii minMax[4*MAXN];
int segTree[4*MAXN];
int lazy[4*MAXN];


void build(int node, int b, int e){
    if(lazy[node] != 0){
        segTree[node] += lazy[node];

        if(b != e){
            lazy[2*node + 1] += lazy[node];
            lazy[2*node + 2] += lazy[node];
        }
        lazy[node] = 0;
    }

    if(b == e){
        minMax[node] = mp(segTree[node], segTree[node]);
        return;
    }

    int m = (b+e)/2;

    build(2*node + 1, b, m);
    build(2*node + 2, m+1, e);

    minMax[node] = mp(min(minMax[2*node + 1].fi,minMax[2*node + 2].fi), max(minMax[2*node + 1].se,
          minMax[2*node + 2].se));
}

int query(int node, int b, int e, int l, int r, int x){
    if(minMax[node].se < x) return 0;
    if(x < minMax[node].fi) return (e-b+1);

    if(b > e || b > r || e < l) return 0;
    if(b == e) return (minMax[node].fi >= x) ? 1 : 0;
    int m = (b+e)/2;

    int q1 = query(2*node + 1, b, m, l, r, x);
    int q2 = query(2*node + 2, m+1, e, l, r, x);

    return q1 + q2;
}

void updateRange(int node, int b, int e, int l, int r){
    //printf("%d [%d, %d] (%d, %d)\n",node, b, e, l, r);
    if(lazy[node] != 0){
        segTree[node] += lazy[node];

        if(b != e){
            lazy[2*node + 1] += lazy[node];
            lazy[2*node + 2] += lazy[node];
        }
        lazy[node] = 0;
    }

    if(b > e || b > r || e < l) return;
    if(b >= l && e <= r){
        segTree[node]++;
        if(b != e){
            lazy[2*node + 1]++;
            lazy[2*node + 2]++;
        }
        return;
    }
    int m = (b+e)/2;

    updateRange(2*node + 1, b, m, l, r);
    updateRange(2*node + 2, m+1, e, l, r);

    segTree[node] = segTree[2*node + 1] + segTree[2*node + 2];
}

int queryRange(int node, int b, int e, int l, int r, int x){

    if(b > e || b > r || e < l){
        return 0;
    }
    if(lazy[node] != 0){
        segTree[node] += lazy[node];

        if(b != e){
            lazy[2*node + 1] += lazy[node];
            lazy[2*node + 2] += lazy[node];
        }
        lazy[node] = 0;
    }

    /*if(b == e) {
            // printf("%d [%d, %d]  = %d\n",node, b, e, segTree[node]);
        return (segTree[node] >= x) ? 1 : 0;
    }
    */

    if(b >= l && e <= r){

    }

    int m = (b+e)/2;
    int q1 = queryRange(2*node + 1, b, m, l, r, x);
    int q2 = queryRange(2*node + 2, m+1, e, l, r, x);

    return q1+q2;
}

int main(){
    int n,m,l,r,x;

    sd2(n,m);
    for(int i = 0; i < m; i++){
        sd2(l,r);
        updateRange(0, 0, n-1, l-1, r-1);
```

```
}
build(0,0,n-1);
int q;
sd(q);
for(int i = 0; i < q; i++){
    sd(x);
    printf("%d\n", query(0,0,n-1,0,n-1,x));
}
```

```
    return 0;
}
```