ICPC Notebook

Sultan Halim - National University of Singapore

Contents

1 Data Structure and Basics

1.1 Bitmask

```
#define isOn(S, j) (S & (1 << j))
#define setBit(S, j) (S |= (1 << j))
#define clearBit(S, j) (S &= ~(1 << j))
#define toggleBit(S, j) (S ^= (1 << j))
#define lowBit(S) (S & (-S))
#define setAll(S, n) (S = (1 << n) - 1)

#define modulo(S, N) ((S) & (N - 1))   // returns S % N, where N is a power of 2
#define isPowerOfTwo(S) (!(S & (S - 1)))
#define nearestPowerOfTwo(S) ((int)pow(2.0, (int)((log((double)S) / log(2.0)) + 0.5)))
#define turnOffLastBit(S) ((S) & (S - 1))
#define turnOnLastZero(S) ((S) | (S + 1))
#define turnOffLastConsecutiveBits(S) ((S) & (S + 1))
#define turnOnLastConsecutiveZeroes(S) ((S) | (S - 1))

#define LSOne(S) (S & (-S))
```

1.2 Sam fenwick tree

```
class FenwickTree { private:
    vector<long long int> ft;
    int n;

  public:
    FenwickTree(int _n) : n(_n) {
      ft.assign(n+1, 0);
    }

    FenwickTree(const vector<int>& f) : n(f.size() - 1) {
      ft.assign(n+1, 0);
      for(int i = 1; i <= n; ++i) {
        ft[i] += f[i];
        if(i + LSOne(i) <= n) {
          ft[i + LSOne(i)] += ft[i];
        }
      }
    }

    long long int rsq(int b) {
      long long int sum = 0;
      for(; b; b -= LSOne(b)) {
        sum += ft[b];
      }
      return sum;
    }

    long long int rsq(int a, int b) {
      return rsq(b) - rsq(a);
    }

    void adjust(int k, long long int v) {
      for(; k <=n; k += LSOne(k)) {
        ft[k] += v;
      }
    }
};
```

1.3 Range fenwick tree

```
class FenwickTree {
private: vi ft1, ft2;
  int query(vi &ft, int b) {
    int sum = 0; for (; b; b -= LSOne(b)) sum += ft[b];
    return sum; }
  void adjust(vi &ft, int k, int v) {
    for (; k < (int)ft.size(); k += LSOne(k)) ft[k] += v; }
public:
  FenwickTree() {}
  FenwickTree(int n) { ft1.assign(n + 1, 0); ft2.assign(n+1, 0);}
  int query(int a) { return a * query(ft1, a) - query(ft2, a); }
  int query(int a, int b) { return query(b) - (a == 1 ? 0 : query(a-1)); }
  void adjust(int a, int b, int value){
    adjust(ft1, a, value);
    adjust(ft1, b+1, -value);
    adjust(ft2, a, value * (a-1));
    adjust(ft2, b+1, -1 * value * b);
  }
  int get(int n) {
    return query(n) - query(n-1);  }
};
```

1.4 UFDS

```
class UnionFind {
  public:
    vector<int> p, rank, setSize;
    int numSets;

  public:
    UnionFind(int N) {
      numSets = N;
      setSize.assign(N, 1);
      rank.assign(N, 0);
      p.assign(N, 0);
      for(int i = 0; i < N; ++i) {
        p[i] = i;
      }
    }

    int findSet(int i) {
      return (p[i] == i) ? i : (p[i] = findSet(p[i]));
    }

    bool isSameSet(int i, int j) {
      bool x = findSet(i) == findSet(j);
      return x;
    }

    void unionSet(int i, int j) {
      if(!isSameSet(i, j)) {
        numSets--;
        int x = findSet(i);
        int y = findSet(j);

        if(rank[x] > rank[y]) {
          p[y] = x; setSize[x] += setSize[y];
        } else {
          p[x] = y; setSize[y] += setSize[x];
          if(rank[x] == rank[y]) {
            rank[y]++;
          }
        }
      }
    }

    int numDisjoinSets() {
      return numSets;
    }

    int sizeOfSet(int i) {
      return setSize[findSet(i)];
    }
};
```

1.5 Segment tree

```
class SegmentTree {         // the segment tree is stored like a heap array
private: vi st, A;            // recall that vi is: typedef vector<int> vi;
  int n;
  int left (int p) { return p << 1; }     // same as binary heap operations
  int right(int p) { return (p << 1) + 1; }

  void build(int p, int L, int R) {                           // O(n log n)
    if (L == R)                                 // as L == R, either one is fine
      st[p] = L;                                              // store the index
    else {                                      // recursively compute the values
      build(left(p) , L              , (L + R) / 2);
      build(right(p), (L + R) / 2 + 1, R          );
      int p1 = st[left(p)], p2 = st[right(p)];
      st[p] = (A[p1] <= A[p2]) ? p1 : p2;
  } }

  int rmq(int p, int L, int R, int i, int j) {                  // O(log n)
    if (i >  R || j <  L) return -1; // current segment outside query range
    if (L >= i && R <= j) return st[p];               // inside query range

     // compute the min position in the left and right part of the interval
    int p1 = rmq(left(p) , L            , (L+R) / 2, i, j);
    int p2 = rmq(right(p), (L+R) / 2 + 1, R          , i, j);

    if (p1 == -1) return p2;   // if we try to access segment outside query
    if (p2 == -1) return p1;                              // same as above
    return (A[p1] <= A[p2]) ? p1 : p2; }          // as as in build routine

  int update_point(int p, int L, int R, int idx, int new_value) {
    // this update code is still preliminary, i == j
    // must be able to update range in the future!
    int i = idx, j = idx;

    // if the current interval does not intersect
    // the update interval, return this st node value!
    if (i > R || j < L)
      return st[p];

    // if the current interval is included in the update range,
    // update that st[node]
    if (L == i && R == j) {
      A[i] = new_value; // update the underlying array
      return st[p] = L; // this index
    }

    // compute the minimum pition in the
    // left and right part of the interval
    int p1, p2;
    p1 = update_point(left(p) , L              , (L + R) / 2, idx, new_value);
    p2 = update_point(right(p), (L + R) / 2 + 1, R          , idx, new_value);

    // return the pition where the overall minimum is
    return st[p] = (A[p1] <= A[p2]) ? p1 : p2;
  }

public:
  SegmentTree(const vi &_A) {
    A = _A; n = (int)A.size();              // copy content for local usage
    st.assign(4 * n, 0);            // create large enough vector of zeroes
    build(1, 0, n - 1);                                     // recursive build
  }

  int rmq(int i, int j) { return rmq(1, 0, n - 1, i, j); }   // overloading

  int update_point(int idx, int new_value) {
    return update_point(1, 0, n - 1, idx, new_value); }
};
```

1.6 Backtracking

```
/* 8 Queens Chess Problem */
#include <cstdlib>                                // we use the int version of 'abs'
#include <cstdio>
#include <cstring>
using namespace std;

int row[8], TC, a, b, lineCounter;                      // ok to use global variables

bool place(int r, int c) {
  for (int prev = 0; prev < c; prev++)       // check previously placed queens
     if (row[prev] == r || (abs(row[prev] - r) == abs(prev - c)))
       return false;                // share same row or same diagonal -> infeasible
 return true; }

void backtrack(int c) {
  if (c == 8 && row[b] == a) {
 // candidate sol, (a, b) has 1 queen
```

```
    printf("%2d %d", ++lineCounter, row[0] + 1);
     for (int j = 1; j < 8; j++) printf(" %d", row[j] + 1);
    printf("\n"); }
  for (int r = 0; r < 8; r++)                                         // try all possible row
      if (place(r, c)) {                          // if can place a queen at this col and row
        row[c] = r; backtrack(c + 1);
          // put this queen here and recurse
}     }

int main() {
  scanf("%d", &TC);
  while (TC--) {
     scanf("%d %d", &a, &b); a--; b--;                    // switch to 0-based indexing
    memset(row, 0, sizeof row); lineCounter = 0;
    printf("SOLN       COLUMN\n");
    printf(" #        1  2 3 4 5 6 7 8\n\n");
   backtrack(0);                          // generate all possible 8! candidate solutions
      if (TC) printf("\n");
} } // return 0;
```

1.7 Sparse Table

```
#define MAX_N 1000
#define LOG_TWO_N 10

class RMQ {
private:
  int _A[MAX_N], SpT[MAX_N][LOG_TWO_N];
public:
  RMQ(int n, int A[]) {
    for(int i = 0; i < n; i++) {
      _A[i] = A[i];
      SpT[i][0] = i;
    }
    for (int j = 1; (1 << j) <= n; j++)
      for (int i = 0; i + (1 << j) - 1 < n; i++)
        if(_A[SpT[i][j-1]] < _A[SpT[i+(1<<(j-1))][j-1]])
          SpT[i][j] = SpT[i][j-1];
        else
          SpT[i][j] = SpT[i+(1 << (j-1))][j-1];
  }

  int query(int i, int j) {
    int k = (int)floor(log((double)j-i+1) / log(2.0));
    if (_A[SpT[i][k]] <= _A[SpT[j-(1<<k)+1][k]]) return SpT[i][k];
    else return SpT[j-(1<<k)+1][k];
  }
};
```

1.8 LCA

```
inline void makeP(){
  int logg = lg[n];
  for (int j = 1; j <= logg; j++) {
    for (int i = 1; i <= n; i++) {
      if (P[i][j-1] != -1) {
        P[i][j] = P[P[i][j-1]][j-1];
      }
    }
  }
}

inline int LCA(int p, int q) {
  if (L[p] < L[q]) swap(p, q);
  int logg = lg[L[p]];
  for (int i = logg; i >= 0; i--) {
    if (L[p] - (1 << i) >= L[q]) {
      p = P[p][i]l
    }
  }
  if (p == q) return p;
  for (int i = logg; i >= 0; i--) {
    if (P[p][i] != -1 && P[p][i] != P[q][i]) {
      p = P[p][i];
      q = P[q][i];
    }
  }
  return P[p][0];
}
```

2 String

2.1 KMP

```
/*
Searches for the string w in the string s (of length k). Returns the
0-based index of the first match (k if no match is found). Algorithm
runs in O(k) time.
*/

#include <iostream>
#include <string>
#include <vector>

using namespace std;

typedef vector<int> VI;

void buildTable(string& w, VI& t)
{
  t = VI(w.length());
  int i = 2, j = 0;
  t[0] = -1; t[1] = 0;

  while(i < w.length())
  {
    if(w[i-1] == w[j]) { t[i] = j+1; i++; j++; }
    else if(j > 0) j = t[j];
    else { t[i] = 0; i++; }
  }
}

int KMP(string& s, string& w)
{
  int m = 0, i = 0;
  VI t;

  buildTable(w, t);
  while(m+i < s.length())
  {
    if(w[i] == s[m+i])
    {
      i++;
      if(i == w.length()) return m;
    }
    else
    {
      m += i-t[i];
      if(i > 0) i = t[i];
    }
  }
  return s.length();
}
```

2.2 Aho-Corasick

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

// {{{ Aho Corasick
namespace AhoCorasick {
  struct Node {
    int next[26];
    int parent;
    int fromParent;
    int fail;
    int wordCount;
  };

  int last;
  Node node[MAXNODE];

  void init() {
    last = 0;
    for (int i = 0; i < MAXNODE; i++) {
      for (int j = 0; j < 26; j++) {
        node[i].next[j] = -1;
      }
      node[i].parent = node[i].fail = -1;
      node[i].wordCount = 0;
    }
```

```
    node[0].fail = 0;
  }

  int addString(char *s, int offset = 'a') {
    int len = strlen(s);
    int cur = 0;
    for (int i = 0; i < len; i++) { //no C++11? BibleThump
      if (node[cur].next[s[i] - offset] == -1) {
        node[cur].next[s[i] - offset] = ++last;
        node[last].parent = cur;
        node[last].fromParent = s[i] - offset;
      }
      cur = node[cur].next[s[i] - offset];
    }
    node[cur].wordCount++;
    return cur;
  }

  void constructFail() {
    queue<int> q;
    q.push(0);
    while (!q.empty()) {
      int u = q.front();
      q.pop();
      for (int i = 0; i < 26; i++) {
        if (node[u].next[i] != -1) {
          q.push(node[u].next[i]);
        }
      }
      if (u == 0) {
        continue;
      }
      int v = node[node[u].parent].fail;
      int c = node[u].fromParent;
      while (v != 0 && node[v].next[c] == -1) {
        v = node[v].fail;
      }
      node[u].fail = node[v].next[c] != -1  && node[v].next[c] != u ? node[v].next[c] : 0;
    }
  }

  int search(char *s, int offset) {
    int len = strlen(s);
    int cur = 0;
    int match = 0;
    for (int i = 0; i < len; i++) {
      while (cur != 0 && node[cur].next[s[i] - offset] == -1) {
        assert(node[cur].fail != -1);
        cur = node[cur].fail;
      }
      if (node[cur].next[s[i] - offset] != -1) {
        cur = node[cur].next[s[i] - offset];
      }
      assert(cur != -1);
      match += node[cur].wordCount;
    }
    return match;
  }
}
// }}}
```

2.3 Hash

```
// {{{ Hash
template<int pr, int MOD> class Hash {
  int n;
  vector<int> p, v;
  void init() {
    p.resize(n);
    p[0] = 1;
    for (int i = 1; i < n; i++) {
      p[i] = (int)(1LL * p[i - 1] * pr % MOD);
      v[i] = (int)(1LL * v[i - 1] * pr % MOD) + v[i];
      if (v[i] >= MOD) v[i] -= MOD;
    }
  }
public:
  Hash() {}
  Hash(string s) {
    n = (int)s.length();
    v.resize(n);
    for (int i = 0; i < n; i++) {
      v[i] = s;
    }
    init();
  }
  Hash(const vector<int> &v) {
    this->v = v;
    n = (int)v.size();
    for (auto &x : this->v) {
      x %= MOD;
      if (x < 0) x += MOD;
    }
    init();
  }
  int get(int l, int r) {
    int res = v[r] - (l > 0 ? (int)(1LL * v[l - 1] * p[r - l + 1] % MOD) : 0);
    if (res >= MOD) res -= MOD;
    if (res < 0) res += MOD;
    return res;
  }
};
// }}}
```

2.4 SuffixArray

```
// Source: http://codeforces.com/contest/452/submission/7269543
// Efficient Suffix Array O(N*logN)

// String index from 0
// Usage:
// string s;
// SuffixArray sa(s);
// Now we can use sa.SA and sa.LCP
struct SuffixArray {
    string a;
    int N, m;
    vector<int> SA, LCP, x, y, w, c;

    SuffixArray(string _a, int m) : a(" " + _a), N(a.length()), m(m),
            SA(N), LCP(N), x(N), y(N), w(max(m, N)), c(N) {
        a[0] = 0;
        DA();
        kasaiLCP();
        #define REF(X) { rotate(X.begin(), X.begin()+1, X.end()); X.pop_back(); }
        REF(SA); REF(LCP);
        a = a.substr(1, a.size());
        for(int i = 0; i < SA.size(); ++i) --SA[i];
        #undef REF
    }

    inline bool cmp (const int a, const int b, const int l) { return (y[a] == y[b] && y[a + l] == y[b
         + l]); }

    void Sort() {
        for(int i = 0; i < m; ++i) w[i] = 0;
        for(int i = 0; i < N; ++i) ++w[x[y[i]]];
        for(int i = 0; i < m - 1; ++i) w[i + 1] += w[i];
        for(int i = N - 1; i >= 0; --i) SA[--w[x[y[i]]]] = y[i];
    }

    void DA() {
        for(int i = 0; i < N; ++i) x[i] = a[i], y[i] = i;
        Sort();
        for(int i, j = 1, p = 1; p < N; j <<= 1, m = p) {
            for(p = 0, i = N - j; i < N; i++) y[p++] = i;
            for (int k = 0; k < N; ++k) if (SA[k] >= j) y[p++] = SA[k] - j;
            Sort();
            for(swap(x, y), p = 1, x[SA[0]] = 0, i = 1; i < N; ++i)
                x[SA[i]] = cmp(SA[i - 1], SA[i], j) ? p - 1 : p++;
        }
    }

    void kasaiLCP() {
        for (int i = 0; i < N; i++) c[SA[i]] = i;
        for (int i = 0, j, k = 0; i < N; LCP[c[i++]] = k)
            if (c[i] > 0) for (k ? k-- : 0, j = SA[c[i] - 1]; a[i + k] == a[j + k]; k++);
            else k = 0;
    }
};
```

3 Graph Basic

3.1 Dijkstra

```
int main() {
  int V, E, s, u, v, w;
  vector<vii> AdjList;
```

```cpp
  AdjList.assign(V, vii()); // assign blank vectors of pair<int, int>s to AdjList
  for (int i = 0; i < E; i++) {
    scanf("%d %d %d", &u, &v, &w);
    AdjList[u].push_back(ii(v, w));                                    // directed graph
  }

  // Dijkstra routine
  vi dist(V, INF); dist[s] = 0;                          // INF = 1B to avoid overflow
  priority_queue< ii, vector<ii>, greater<ii> > pq; pq.push(ii(0, s));
                                      // ^to sort the pairs by increasing distance from s
  while (!pq.empty()) {                                                       // main loop
    ii front = pq.top(); pq.pop();          // greedy: pick shortest unvisited vertex
    int d = front.first, u = front.second;
    if (d > dist[u]) continue;      // this check is important, see the explanation
    for (int j = 0; j < (int)AdjList[u].size(); j++) {
      ii v = AdjList[u][j];                              // all outgoing edges from u
      if (dist[u] + v.second < dist[v.first]) {
        dist[v.first] = dist[u] + v.second;                     // relax operation
        pq.push(ii(dist[v.first], v.first));
  } } }  // note: this variant can cause duplicate items in the priority queue

  for (int i = 0; i < V; i++) // index + 1 for final answer
    printf("SSSP(%d, %d) = %d\n", s, i, dist[i]);

  return 0;
}
```

3.2 Bellman Ford

```cpp
int main() {
  int V, E, s, a, b, w;
  vector<vii> AdjList;

  AdjList.assign(V, vii()); // assign blank vectors of pair<int, int>s to AdjList
  for (int i = 0; i < E; i++) {
    scanf("%d %d %d", &a, &b, &w);
    AdjList[a].push_back(ii(b, w));
  }

  // Bellman Ford routine
  vi dist(V, INF); dist[s] = 0;
  for (int i = 0; i < V - 1; i++)   // relax all E edges V-1 times, overall O(VE)
    for (int u = 0; u < V; u++)                           // these two loops = O(E)
      for (int j = 0; j < (int)AdjList[u].size(); j++) {
        ii v = AdjList[u][j];         // we can record SP spanning here if needed
        dist[v.first] = min(dist[v.first], dist[u] + v.second);          // relax
      }

  bool hasNegativeCycle = false;
  for (int u = 0; u < V; u++)                                 // one more pass to check
    for (int j = 0; j < (int)AdjList[u].size(); j++) {
      ii v = AdjList[u][j];
      if (dist[v.first] > dist[u] + v.second)                    // should be false
        hasNegativeCycle = true;        // but if true, then negative cycle exists!
    }
  printf("Negative Cycle Exist? %s\n", hasNegativeCycle ? "Yes" : "No");

  if (!hasNegativeCycle)
    for (int i = 0; i < V; i++)
      printf("SSSP(%d, %d) = %d\n", s, i, dist[i]);

  return 0;
}
```

3.3 MCBM

```cpp
vector<vi> AdjList;
vi match, vis;                                                       // global variables

int Aug(int l) {                             // return 1 if an augmenting path is found
  if (vis[l]) return 0;                                             // return 0 otherwise
  vis[l] = 1;
  for (int j = 0; j < (int)AdjList[l].size(); j++) {
    int r = AdjList[l][j];
    if (match[r] == -1 || Aug(match[r])) {
      match[r] = l; return 1;                                       // found 1 matching
  } }
  return 0;

    // no matching
}

int main() {
  int V = 5, Vleft = 3;                                             // we ignore vertex 0
  AdjList.assign(V, vi());
  AdjList[1].push_back(3); AdjList[1].push_back(4);
  AdjList[2].push_back(3);

  int MCBM = 0;
  match.assign(V, -1);     // V is the number of vertices in bipartite graph
  for (int l = 0; l < Vleft; l++) {              // Vleft = size of the left set
    vis.assign(Vleft, 0);                             // reset before each recursion
    MCBM += Aug(l);
  }
  printf("Found %d matchings\n", MCBM);   // the answer is 2 for Figure 4.42

  return 0;
}
```

4 Graph

4.1 2-SAT

```cpp
// VAR(x), NOT(x)
// TwoSat(int varCount)
// bool TwoSat.solve() -> vector<int> TwoSat.value
// {{{ 2-SAT
using graph = vector<vi>;
inline int VAR(int x) { return 2 * x; }
inline int NOT(int x) { return x ^ 1; }

struct TwoSat {
  int n;
  vi value;
  graph g, grev;
  TwoSat(int n) : n(2 * n), g(2 * n) {}

  void addImplication(int x, int y) {
    g[x].emplace_back(y);
    g[NOT(y)].emplace_back(NOT(x));
  }

  void addOr(int x, int y) {
    g[NOT(x)].emplace_back(y);
    g[NOT(y)].emplace_back(x);
  }

  void dfs(int u, graph &g, vi &visit, vi &order) {
    visit[u] = 1;
    for (int v : g[u]) {
      if (!visit[v]) {
        dfs(v, g, visit, order);
      }
    }
    order.push_back(u);
  }

  bool solve() {
    grev.assign(n, vi());
    for (int i = 0; i < n; i++) {
      for (int j : g[i]) {
        grev[j].push_back(i);
      }
    }
    vi visited(n, false);
    vi order;
    for (int i = 0; i < n; i++) {
      if (!visited[i]) {
        dfs(i, g, visited, order);
      }
    }
    reverse(ALL(order));
    vi repr(n);
    fill(ALL(visited), false);
    value.assign(n, -1);
```

```
    for (int u : order) {
      if (!visited[u]) {
        vi tmp;
        dfs(u, grev, visited, tmp);
        for (int v : tmp) {
          repr[v] = tmp[0];
          if (value[v] == -1) {
            value[v] = 1;
            value[NOT(v)] = 0;
          }
        }
      }
    }
    for (int i = 0; i < n; i++) {
      if (repr[i] == repr[NOT(i)]) {
        return false;
      }
    }
    return true;
  }
};
// }}}
```

4.2 Biconnected Component

```
// Input graph: vector< vector<int> > a, int n
// Note: 0-indexed
// Usage: BiconnectedComponent bc; (bc.components is the list of components)

struct BiconnectedComponent {
  vector<int> low, num, s;
  vector< vector<int> > components;
  int counter;

  BiconnectedComponent() : num(n, -1), low(n, -1), counter(0) {
    for (int i = 0; i < n; i++)
      if (num[i] < 0)
        dfs(i, 1);
  }

  void dfs(int x, int isRoot) {
    low[x] = num[x] = ++counter;
    if (a[x].empty()) {
      components.push_back(vector<int>(1, x));
      return;
    }
    s.push_back(x);

    for (int i = 0; i < a[x].size(); i++) {
      int y = a[x][i];
      if (num[y] > -1) low[x] = min(low[x], num[y]);
      else {
        dfs(y, 0);
        low[x] = min(low[x], low[y]);

        if (isRoot || low[y] >= num[x]) {
          components.push_back(vector<int>(1, x));
          while (1) {
            int u = s.back();
            s.pop_back();
            components.back().push_back(u);
            if (u == y) break;
          }
        }
      }
    }
  }
};
```

4.3 Bridge Articulation

```
// Assume already have undirected graph vector< vector<int> > G with V vertices
// Vertex index from 0
// Usage:
// UndirectedDfs tree;
// Then you can use tree.bridges and tree.cuts
struct UndirectedDfs {
    vector<int> low, num, parent;
    vector<bool> articulation;
    int counter, root, children;

    vector< pair<int,int> > bridges;
    vector<int> cuts;

    UndirectedDfs() : low(V, 0), num(V, -1), parent(V, 0), articulation(V, false),
            counter(0), children(0) {
        for(int i = 0; i < V; ++i) if (num[i] == -1) {
            root = i; children = 0;
            dfs(i);
            articulation[root] = (children > 1);
        }
        for(int i = 0; i < V; ++i)
            if (articulation[i]) cuts.push_back(i);
    }
private:
    void dfs(int u) {
        low[u] = num[u] = counter++;
        for(int j = 0; j < G[u].size(); ++j) {
            int v = G[u][j];
            if (num[v] == -1) {
                parent[v] = u;
                if (u == root) children++;
                dfs(v);
                if (low[v] >= num[u])
                    articulation[u] = true;
                if (low[v] > num[u]) bridges.push_back(make_pair(u, v));
                low[u] = min(low[u], low[v]);
            } else if (v != parent[u])
                low[u] = min(low[u], num[v]);
        }
    }
};
```

4.4 Dinic Max Flow

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

//Add Edge : MaxFlow.addEdge(int from, int to, F capacity)
//Solve : MaxFlow.calcMaxFlow(int source, int sink)
// {{{ Dinic Max Flow<F=int>
template<typename F = int>
struct DinicMaxFlow {
  struct Edge {
    int to, rev;
    F flow, cap;
  };

  const F INF_FLOW = numeric_limits<F>::max() / 2;
  const int INF_DIST = 1e9;

  int source, sink;
  vector<vector<Edge>> graph;
  vector<int> dist, ptr;

  DinicMaxFlow(int N) : graph(N), dist(N), ptr(N) {}

  void addEdge(int from, int to, F cap) {
    if (from == to) {
      return;
    }
    Edge e1 { to, SZ(graph[to]), 0, cap };
    Edge e2 { from, SZ(graph[from]), 0, 0 };
    graph[from].push_back(e1);
    graph[to].push_back(e2);
  }

  bool buildLevelGraph() {
    fill(begin(dist), end(dist), INF_DIST);
    queue<int> q;
    dist[source] = 0;
    q.push(source);
    while (!q.empty()) {
      int u = q.front();
      q.pop();
      for (const auto &e : graph[u]) {
        if (e.flow < e.cap && dist[e.to] > dist[u] + 1) {
          dist[e.to] = dist[u] + 1;
          q.push(e.to);
        }
      }
    }
    return dist[sink] != INF_DIST;
  }

  F dfs(int u, F bottleneck) {
    if (u == sink || bottleneck == 0) {
      return bottleneck;
    }
    for (int &i = ptr[u]; i < SZ(graph[u]); i++) {
      auto &e = graph[u][i];
```

```
      auto &b = graph[e.to][e.rev];
      if (dist[e.to] != dist[u] + 1 || e.flow >= e.cap) {
        continue;
      }
      F pushed = dfs(e.to, min(e.cap - e.flow, bottleneck));
      if (pushed > 0) {
        e.flow += pushed;
        b.flow -= pushed;
        return pushed;
      }
    }
    return 0;
  }

  F calcMaxFlow(const int source, const int sink) {
    this->source = source;
    this->sink = sink;
    F mf = 0;
    while (buildLevelGraph()) {
      fill(begin(ptr), end(ptr), 0);
      F pushed;
      while ((pushed = dfs(source, INF_FLOW))) {
        mf += pushed;
      }
    }
    return mf;
  }
};
// }}}
```

4.5 Directed MST

```
#include "../../template.h"

const int maxe = 100111;
const int maxv = 100;

// Index from 0
// Running time O(E*V)
namespace chuliu {
    struct Cost;
    vector<Cost> costlist;

    struct Cost {
        int id, val, used, a, b, pos;
        Cost() { val = -1; used = 0; }
        Cost(int _id, int _val, bool temp) {
            a = b = -1; id = _id; val = _val; used = 0;
            pos = costlist.size(); costlist.push_back(*this);
        }
        Cost(int _a, int _b) {
            a = _a; b = _b; id = -1; val = costlist[a].val - costlist[b].val;
            used = 0; pos = costlist.size(); costlist.push_back(*this);
        }
        void push() {
            if (id == -1) {
                costlist[a].used += used;
                costlist[b].used -= used;
            }
        }
    };

    struct Edge {
        int u, v;
        Cost cost;
        Edge() {}
        Edge(int id, int _u, int _v, int c) {
            u = _u; v = _v; cost = Cost(id, c, 0);
        }
    } edge[maxe];

    int n, m, root, pre[maxv], node[maxv], vis[maxv], best[maxv];

    void init(int _n) {
        n = _n; m = 0;
        costlist.clear();
    }

    void add(int id, int u, int v, int c) {
        edge[m++] = Edge(id, u, v, c);
    }

    int mst(int root) {
        int ret = 0;

        while (true) {
            REP(i, n) best[i] = -1;

            REP(e, m) {
                int u = edge[e].u, v = edge[e].v;
                if ((best[v] == -1 || edge[e].cost.val < costlist[best[v]].val) && u != v) {
                    pre[v] = u;
                    best[v] = edge[e].cost.pos;
                }
            }

            REP(i, n) if (i != root && best[i] == -1) return -1;

            int cntnode = 0;
            memset(node, -1, sizeof node); memset(vis, -1, sizeof vis);

            REP(i, n) if (i != root) {
                ret += costlist[best[i]].val;
                costlist[best[i]].used++;

                int v = i;
                while (vis[v] != i && node[v] == -1 && v != root) {
                    vis[v] = i;
                    v = pre[v];
                }

                if (v != root && node[v] == -1) {
                    for (int u = pre[v]; u != v; u = pre[u]) node[u] = cntnode;
                    node[v] = cntnode++;
                }
            }

            if (cntnode == 0) break;

            REP(i, n) if (node[i] == -1) node[i] = cntnode++;

            REP(e, m) {
                int v = edge[e].v;
                edge[e].u = node[edge[e].u];
                edge[e].v = node[edge[e].v];
                if (edge[e].u != edge[e].v) edge[e].cost = Cost(edge[e].cost.pos, best[v]);
            }

            n = cntnode;
            root = node[root];
        }

        return ret;
    }

    vector<int> trace() {
        vector<int> ret;
        FORD(i, costlist.size()-1,0) costlist[i].push();
        REP(i, costlist.size()) {
            Cost cost = costlist[i];
            if (cost.id != -1 && cost.used > 0) ret.push_back(cost.id);
        }
        return ret;
    }
}

int main() {
}
```

4.6 Eulerian Cycle

```
// directed!
// careful with empty graph!
// {{{ eulerianTour
template<typename T>
bool eulerianTour(const vector<vector<T>> &g, vi &tour) {
  int n = SZ(g);
  int edgeCount = 0;
  vi inDeg(n), outDeg(n);
  for (int i = 0; i < n; i++) {
    outDeg[i] = SZ(g[i]);
    for (auto &it : g[i]) {
      inDeg[it.fi]++;
      edgeCount++;
    }
  }
  int out = -1, in = -1;
  for (int i = 0; i < n; i++) {
    if (inDeg[i] == outDeg[i] + 1) {
      debug printf("finish vertex %d\n", i);
      if (out == -1) {
        out = i;
      } else {
        return false;
      }
```

```
    } else if (outDeg[i] == inDeg[i] + 1) {
      debug printf("start vertex %d\n", i);
      if (in == -1) {
        in = i;
      } else {
        return false;
      }
    } else if (outDeg[i] != inDeg[i]) {
      return false;
    }
  }
  // either zero or both
  if ((in != -1) ^ (out != -1)) {
    return false;
  }
  if (in == -1) {
    for (int i = 0; i < n; i++) {
      if (outDeg[i] != 0) {
        in = out = i;
        break;
      }
    }
    // empty graph?
    if (in == -1) {
      in = out = 0;
    }
  }
  tour.clear();
  stack<int> st;
  vi visit(n);
  vi ptr(n);
  st.push(in);
  while (!st.empty()) {
    int u = st.top();
    visit[u] = true;
    if (ptr[u] == SZ(g[u])) {
      tour.push_back(u);
      st.pop();
    } else {
      st.push(g[u][ptr[u]++].fi);
    }
  }
  reverse(ALL(tour));
  return SZ(tour) == edgeCount + 1;
}
// }}}
```

4.7 Edmonds Blossom

```
// General matching on graph

const int maxv = 1000;
const int maxe = 50000;

// Index from 1
// Directed
struct EdmondsLawler {
    int n, E, start, finish, newRoot, qsize, adj[maxe], next[maxe], last[maxv], mat[maxv], que[maxv],
          dad[maxv], root[maxv];
    bool inque[maxv], inpath[maxv], inblossom[maxv];

    void init(int _n) {
        n = _n; E = 0;
        for(int x=1; x<=n; ++x) { last[x] = -1; mat[x] = 0; }
    }
    void add(int u, int v) {
        adj[E] = v; next[E] = last[u]; last[u] = E++;
    }
    int lca(int u, int v) {
        for(int x=1; x<=n; ++x) inpath[x] = false;
        while (true) {
            u = root[u];
            inpath[u] = true;
            if (u == start) break;
            u = dad[mat[u]];
        }
        while (true) {
            v = root[v];
            if (inpath[v]) break;
            v = dad[mat[v]];
        }
        return v;
    }
    void trace(int u) {
        while (root[u] != newRoot) {
            int v = mat[u];

            inblossom[root[u]] = true;
            inblossom[root[v]] = true;

            u = dad[v];
            if (root[u] != newRoot) dad[u] = v;
        }
    }
    void blossom(int u, int v) {
        for(int x=1; x<=n; ++x) inblossom[x] = false;

        newRoot = lca(u, v);
        trace(u); trace(v);

        if (root[u] != newRoot) dad[u] = v;
        if (root[v] != newRoot) dad[v] = u;

        for(int x=1; x<=n; ++x) if (inblossom[root[x]]) {
            root[x] = newRoot;
            if (!inque[x]) {
                inque[x] = true;
                que[qsize++] = x;
            }
        }
    }
    bool bfs() {
        for(int x=1; x<=n; ++x){
            inque[x] = false;
            dad[x] = 0;
            root[x] = x;
        }
        qsize = 0;
        que[qsize++] = start;
        inque[start] = true;
        finish = 0;

        for(int i=0; i<qsize; ++i) {
            int u = que[i];
            for (int e = last[u]; e != -1; e = next[e]) {
                int v = adj[e];
                if (root[v] != root[u] && v != mat[u]) {
                    if (v == start || (mat[v] > 0 && dad[mat[v]] > 0)) blossom(u, v);
                    else if (dad[v] == 0) {
                        dad[v] = u;
                        if (mat[v] > 0) que[qsize++] = mat[v];
                        else {
                            finish = v;
                            return true;
                        }
                    }
                }
            }
        }
        return false;
    }
    void enlarge() {
        int u = finish;
        while (u > 0) {
            int v = dad[u], x = mat[v];
            mat[v] = u;
            mat[u] = v;
            u = x;
        }
    }
    int maxmat() {
        for(int x=1; x<=n; ++x) if (mat[x] == 0) {
            start = x;
            if (bfs()) enlarge();
        }

        int ret = 0;
        for(int x=1; x<=n; ++x) if (mat[x] > x) ++ret;
        return ret;
    }
} edmonds;
```

4.8 Hungarian Assignment

```
const int MN = 3210;
const int inf = 1000111000;

// Vertex: 1 --> nx, 1 --> ny
// cost >= 0
// fx[x] + fy[y] <= cost[x][y] for all x, y
// Min cost matching

struct Hungary {
    int nx, ny, cost[MN][MN], fx[MN], fy[MN], matx[MN], which[MN], dist[MN];
    bool used[MN];
```

```
    void init(int _nx, int _ny) {
        nx = _nx; ny = _ny;
        memset(fx, 0, sizeof fx);
        memset(fy, 0, sizeof fy);
        memset(used, false, sizeof used);
        memset(matx, 0, sizeof matx);
        for(int i=0; i<=nx; ++i) for(int j=0; j<=ny; ++j) cost[i][j] = inf;
    }

    void add(int x, int y, int c) { cost[x][y] = min(cost[x][y],c); }

    int mincost() {
        for(int x=1; x<=nx; ++x) {
            int y0 = 0; matx[0] = x;
            for(int y=0; y<=ny; ++y) { dist[y] = inf + 1; used[y] = false; }

            do {
                used[y0] = true;
                int x0 = matx[y0], delta = inf + 1, y1;

                for(int y=1; y<=ny; ++y) if (!used[y]) {
                    int curdist = cost[x0][y] - fx[x0] - fy[y];
                    if (curdist < dist[y]) {
                        dist[y] = curdist;
                        which[y] = y0;
                    }
                    if (dist[y] < delta) {
                        delta = dist[y];
                        y1 = y;
                    }
                }

                for(int y=0; y<=ny; ++y) if (used[y]) {
                    fx[matx[y]] += delta;
                    fy[y] -= delta;
                } else dist[y] -= delta;

                y0 = y1;
            } while (matx[y0] != 0);

            do {
                int y1 = which[y0];
                matx[y0] = matx[y1];
                y0 = y1;
            } while (y0);
        }
//      return -fy[0]; // n  u  lu n     m  b  o  c   b   gh p     y
        int ret = 0;
        for(int y=1; y<=ny; ++y) {
            int x = matx[y];
            if (cost[x][y] < inf) ret += cost[x][y];
        }
        return ret;
    }
} hungary;
```

4.9 Min Cost Max Flow

```
// Min Cost Max Flow - SPFA
// Index from 0
// Lots of double comparison --> likely to fail for double
// Example:
// MinCostFlow mcf(n);
// mcf.addEdge(1, 2, 3, 4);
// cout << mcf.minCostFlow() << endl;

template<class Flow=int, class Cost=int>
struct MinCostFlow {
    const Flow INF_FLOW = 1000111000;
    const Cost INF_COST = 1000111000111000LL;

    int n, t, S, T;
    Flow totalFlow;
    Cost totalCost;
    vector<int> last, visited;
    vector<Cost> dis;
    struct Edge {
        int to;
        Flow cap;
        Cost cost;
        int next;
        Edge(int to, Flow cap, Cost cost, int next) :
                to(to), cap(cap), cost(cost), next(next) {}
    };
    vector<Edge> edges;

    MinCostFlow(int n) : n(n), t(0), totalFlow(0), totalCost(0), last(n, -1), visited(n, 0), dis(n, 0)
            {
        edges.clear();
    }

    int addEdge(int from, int to, Flow cap, Cost cost) {
        edges.push_back(Edge(to, cap, cost, last[from]));
        last[from] = t++;
        edges.push_back(Edge(from, 0, -cost, last[to]));
        last[to] = t++;
        return t - 2;
    }

    pair<Flow, Cost> minCostFlow(int _S, int _T) {
        S = _S; T = _T;
        SPFA();
        while (1) {
            while (1) {
                REP(i,n) visited[i] = 0;
                if (!findFlow(S, INF_FLOW)) break;
            }
            if (!modifyLabel()) break;
        }
        return make_pair(totalFlow, totalCost);
    }

private:
    void SPFA() {
        REP(i,n) dis[i] = INF_COST;
        priority_queue< pair<Cost,int> > Q;
        Q.push(make_pair(dis[S]=0, S));
        while (!Q.empty()) {
            int x = Q.top().second;
            Cost d = -Q.top().first;
            Q.pop();
            // For double: dis[x] > d + EPS
            if (dis[x] != d) continue;
            for(int it = last[x]; it >= 0; it = edges[it].next)
                if (edges[it].cap > 0 && dis[edges[it].to] > d + edges[it].cost)
                    Q.push(make_pair(-(dis[edges[it].to] = d + edges[it].cost), edges[it].to));
        }
        REP(i,n) dis[i] = dis[T] - dis[i];
    }

    Flow findFlow(int x, Flow flow) {
        if (x == T) {
            totalCost += dis[S] * flow;
            totalFlow += flow;
            return flow;
        }
        visited[x] = 1;
        Flow now = flow;
        for(int it = last[x]; it >= 0; it = edges[it].next)
            // For double: fabs(dis[edges[it].to] + edges[it].cost - dis[x]) < EPS
            if (edges[it].cap && !visited[edges[it].to] && dis[edges[it].to] + edges[it].cost == dis[x
                    ]) {
                Flow tmp = findFlow(edges[it].to, min(now, edges[it].cap));
                edges[it].cap -= tmp;
                edges[it ^ 1].cap += tmp;
                now -= tmp;
                if (!now) break;
            }
        return flow - now;
    }

    bool modifyLabel() {
        Cost d = INF_COST;
        REP(i,n) if (visited[i])
            for(int it = last[i]; it >= 0; it = edges[it].next)
                if (edges[it].cap && !visited[edges[it].to])
                    d = min(d, dis[edges[it].to] + edges[it].cost - dis[i]);

        // For double: if (d > INF_COST / 10)     INF_COST = 1e20
        if (d == INF_COST) return false;
        REP(i,n) if (visited[i])
            dis[i] += d;
        return true;
    }
};
```

4.10 Strongly Connected Component

```
// Assume that already have directed graph vector< vector<int> > G with V vertices
// Index from 0
// Usage:
// DirectedDfs tree;
// Now you can use tree.scc
```

```
struct DirectedDfs {
    vector<int> num, low, current, S;
    int counter;
    vector< vector<int> > scc;

    DirectedDfs() : num(V, -1), low(V, 0), current(V, 0), counter(0) {
        REP(i,V) if (num[i] == -1) dfs(i);
    }

    void dfs(int u) {
        low[u] = num[u] = counter++;
        S.push_back(u);
        current[u] = 1;
        REP(j, G[u].size()) {
            int v = G[u][j];
            if (num[v] == -1) dfs(v);
            if (current[v]) low[u] = min(low[u], low[v]);
        }
        if (low[u] == num[u]) {
            scc.push_back(vector<int>());
            while (1) {
                int v = S.back(); S.pop_back(); current[v] = 0;
                scc.back().push_back(v);
                if (u == v) break;
            }
        }
    }
};
```

4.11 LRFlow

```
#include "Dinic.cpp"

//Construct : LRFlow(int N, int originalSource, int originalSink)
//Add Edge : LRFlow.addEdge(int from, int to, F lowerBound, F upperBound) []
//Solve: LRFlow.calcLRFlow()
// {{{ LR FLow<F=int, MF=DinicMaxFlow<F>>
template<typename F = int, typename MF = DinicMaxFlow<F>>
struct LRFlow {
  MF mf;
  F lowerBoundSum = 0;
  int n, realSource, realSink;

  LRFlow(int n, int src, int snk) : mf(n + 2), n(n + 2) {
    realSource = this->n - 2;
    realSink = this->n - 1;
    mf.addEdge(snk, src, mf.INF_FLOW);
  }

  void addEdge(int from, int to, F lowerBound, F upperBound) {
    assert(lowerBound <= upperBound);
    lowerBoundSum += lowerBound;
    mf.addEdge(from, to, upperBound - lowerBound);
    mf.addEdge(realSource, to, lowerBound);
    mf.addEdge(from, realSink, lowerBound);
  }

  pair<F, bool> calcLRFlow() {
    F maxFlow = mf.calcMaxFlow(realSource, realSink);
    return make_pair(maxFlow, maxFlow == lowerBoundSum);
  }
};
// }}}
```

4.12 Hopcroft Karp

```
#include <algorithm>
#include <iostream>

using namespace std;

const int MAXN1 = 50000;
const int MAXN2 = 50000;
const int MAXM = 150000;

int n1, n2, edges, last[MAXN1], prev[MAXM], head[MAXM];
int matching[MAXN2], dist[MAXN1], Q[MAXN1];
bool used[MAXN1], vis[MAXN1];

void init(int _n1, int _n2) {
    n1 = _n1;
    n2 = _n2;
    edges = 0;
    fill(last, last + n1, -1);
}

void addEdge(int u, int v) {
    head[edges] = v;
    prev[edges] = last[u];
    last[u] = edges++;
}

void bfs() {
    fill(dist, dist + n1, -1);
    int sizeQ = 0;
    for (int u = 0; u < n1; ++u) {
        if (!used[u]) {
            Q[sizeQ++] = u;
            dist[u] = 0;
        }
    }
    for (int i = 0; i < sizeQ; i++) {
        int u1 = Q[i];
        for (int e = last[u1]; e >= 0; e = prev[e]) {
            int u2 = matching[head[e]];
            if (u2 >= 0 && dist[u2] < 0) {
                dist[u2] = dist[u1] + 1;
                Q[sizeQ++] = u2;
            }
        }
    }
}

bool dfs(int u1) {
    vis[u1] = true;
    for (int e = last[u1]; e >= 0; e = prev[e]) {
        int v = head[e];
        int u2 = matching[v];
        if (u2 < 0 || !vis[u2] && dist[u2] == dist[u1] + 1 && dfs(u2)) {
            matching[v] = u1;
            used[u1] = true;
            return true;
        }
    }
    return false;
}

int maxMatching() {
    fill(used, used + n1, false);
    fill(matching, matching + n2, -1);
    for (int res = 0;;) {
        bfs();
        fill(vis, vis + n1, false);
        int f = 0;
        for (int u = 0; u < n1; ++u)
            if (!used[u] && dfs(u))
                ++f;
        if (!f)
            return res;
        res += f;
    }
}

int main() {
    init(2, 2);

    addEdge(0, 0);
    addEdge(0, 1);
    addEdge(1, 1);

    cout << (2 == maxMatching()) << endl;
}
```

4.13 Heavy Light Decomposition

```
template <class T, int V>
class HeavyLight {
  int parent[V], heavy[V], depth[V];
  int root[V], treePos[V];
  SegmentTree<T> tree;

  template <class G>
  int dfs(const G& graph, int v) {
    int size = 1, maxSubtree = 0;
    for (int u : graph[v]) if (u != parent[v]) {
      parent[u] = v;
      depth[u] = depth[v] + 1;
      int subtree = dfs(graph, u);
      if (subtree > maxSubtree) heavy[v] = u, maxSubtree = subtree;
      size += subtree;
```

```
    }
    return size;
  }

  template <class BinaryOperation>
  void processPath(int u, int v, BinaryOperation op) {
    for (; root[u] != root[v]; v = parent[root[v]]) {
      if (depth[root[u]] > depth[root[v]]) swap(u, v);
      op(treePos[root[v]], treePos[v] + 1);
    }
    if (depth[u] > depth[v]) swap(u, v);
    op(treePos[u], treePos[v] + 1);
  }

public:
  template <class G>
  void init(const G& graph) {
    int n = graph.size();
    fill_n(heavy, n, -1);
    parent[0] = -1;
    depth[0] = 0;
    dfs(graph, 0);
    for (int i = 0, currentPos = 0; i < n; ++i)
      if (parent[i] == -1 || heavy[parent[i]] != i)
        for (int j = i; j != -1; j = heavy[j]) {
          root[j] = i;
          treePos[j] = currentPos++;
        }
    tree.init(n);
  }

  void set(int v, const T& value) {
    tree.set(treePos[v], value);
  }

  void modifyPath(int u, int v, const T& value) {
    processPath(u, v, [this, &value](int l, int r) { tree.modify(l, r, value); });
  }

  T queryPath(int u, int v) {
    T res = T();
    processPath(u, v, [this, &res](int l, int r) { res.add(tree.query(l, r)); });
    return res;
  }
};
```

4.14 Gomory-Hu

```
// tree which represents all-pair min-cut
// min-cut of u-v = minimum edge in path from u to v in tree
// combine the following function with usual max flow routine
// also because we need to reset the flow graph during each max flow, // don't forget to add variable
     in struct to record initial capacity
void buildGomoryHu() {
  for(int i = 2 ; i <= n ; i++)
    parent[i] = 1;
  for(int i = 2 ; i <= n ; i++) {
    int minCut = maxFlow(i,parent[i]);
    tree[parent[i]].pb({i,minCut});
    // the one below is not necessary if we want to make a rooted tree tree[i].pb({parent[i],minCut});
    for(int j = i + 1 ; j <= n ; j++) {
      if(dist[j] != -1 && parent[j] == parent[i]) {
        parent[j] = i;
      }
    }
  }
}
```

4.15 Online-Bridge

```
// O(N + M)
int psetBridge[N], psetComp[N]; // UF super vertice, UF tree int par[N];
bool seen[N];
int size[N];
int bridge;
int n;
void reset() {
  for(int i = 0 ; i <= n ; i++) {
    psetBridge[i] = psetComp[i] = i;
    par[i] = -1;
    seen[i] = 0;
    size[i] = 1;
  }
  bridge = 0;
}
int findsB(int x) {
  if(x == -1) return -1;
  return x == psetBridge[x] ? x : psetBridge[x] = findsB(psetBridge[x]);
}
int findsC(int x) {
  x = findsB(x);
  return x == psetComp[x] ? x : psetComp[x] = findsC(psetComp[x]);
}
void makeRoot(int x) { // make super vertice which contains x root of its tree
  x = findsB(x);
  int root = x;
  int chld = -1;
  while(x != -1) {
    int papa = findsB(par[x]);
    par[x] = chld;
    psetComp[x] = root;
    chld = x;
    x = papa;
  }
  size[root] = size[chld];
}
void mergePath(int u,int v) { // remove bridge between u and v in tree vector<int> vu,vv;
  int lca = -1;
  while(1) {
    if(u != -1) {
      u = findsB(u);
      vu.push_back(u);
      if(seen[u]) {
        lca = u;
        break;
      }
      seen[u] = 1;
      u = par[u];
    }
    if(v != -1) {
      v = findsB(v);
      vv.push_back(v);
      if(seen[v]) {
        lca = v;
        break;
      }
      seen[v] = 1;
      v = par[v];
    }
  }
  for(int i = 0 ; i < 2 ; i++) {
    vector<int> &proc = i ? vv : vu;
    for(int x : proc) {
      psetBridge[x] = lca;
      if(x == lca) {
        break;
      }
      bridge--;
    }
    for(int x : proc)
      seen[x] = 0;
  }
}
void addEdge(int u,int v) {
  u = findsB(u);
  v = findsB(v);
  if(u != v) {
    int uu = findsC(u);
    int vv = findsC(v);
    if(uu != vv) {
      bridge++;
      if(size[uu] > size[vv]) {
        swap(u,v);
        swap(vv,uu);
      }
      makeRoot(u);
      par[u] = psetComp[u] = v;
      size[vv] += size[u];
    } else
      mergePath(u,v);
  }
}
```

5 Math Basic

5.1 Big Integer

```
import java.io.*;
```

```java
import java.util.*;
import java.util.Scanner;
import java.math.BigInteger;

class BigInteger2 {
  public static void main(String[] args) throws Exception {
    BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
    PrintWriter pw = new PrintWriter(new BufferedWriter(new OutputStreamWriter(System.out)));
    while(true) {
        String N = br.readLine();
          if (N == null) break;
        BigInteger BN = new BigInteger(N);
        BigInteger BRN = new BigInteger(
              new StringBuffer(BN.toString()).reverse().toString());
          pw.printf("%s is ", N);
          if (!BN.isProbablePrime(10)) // 10 is enough
              pw.printf("not prime.\n");
          else if (!BN.equals(BRN) && BRN.isProbablePrime(10))
              pw.printf("emirp.\n")
          else
              pw.printf("prime.\n")
        }
      pw.close();
    }
}

import java.util.Scanner; // Scanner class is inside package java.util
import java.math.BigInteger; // BigInteger class is inside package java.math

class Main { /* UVa 10925 - Krakovia, 0.732s in Java */
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
      int caseNo = 1;
    while (true) {
         int N = sc.nextInt(), F = sc.nextInt(); // N bills, F friends
          if (N == 0 && F == 0) break;
      BigInteger sum = BigInteger.ZERO; // BigInteger has this constant ZERO
         for (int i = 0; i < N; i++) { // sum the N large bills
          BigInteger V = sc.nextBigInteger(); // for reading next BigInteger!
              sum = sum.add(V); // this is BigInteger addition
          }
        System.out.println("Bill #" + (caseNo++) + " costs " +
              sum + ": each friend should pay " + sum.divide(BigInteger.valueOf(F)));
        System.out.println(); // the line above is BigInteger division
} } }                                          // divide the large sum to F friends
```

5.2 Prime

```cpp
ll _sieve_size;
bitset<10000010> bs;     // 10^7 should be enough for most cases
vi primes;     // compact list of primes in form of vector<int>

// first part

void sieve(ll upperbound) {                  // create list of primes in [0..upperbound]
  _sieve_size = upperbound + 1;                              // add 1 to include upperbound
  bs.set();

      // set all bits to 1
  bs[0] = bs[1] = 0;                                                                  //
      except index 0 and 1
  for (ll i = 2; i <= _sieve_size; i++) if (bs[i]) {
      // cross out multiples of i starting from i * i!
    for (ll j = i * i; j <= _sieve_size; j += i) bs[j] = 0;
    primes.push_back(i);   // also add this vector containing list of primes
} }                                                                           // call this
     method in main method

bool isPrime(ll N) {                                // a good enough deterministic prime tester
  if (N < _sieve_size) return bs[N];                              // O(1) for small primes
  for (int i = 0; i < (int)primes.size(); i++)
    if (N % primes[i] == 0) return false;
  return true;                                      // it takes longer time if N is a large prime!
}                                            // note: only work for N <= (last prime in vi "primes")^2

// second part

vi primeFactors(ll N) {     // remember: vi is vector of integers, ll is long long
  vi factors;                                       // vi `primes' (generated by sieve) is optional
  ll PF_idx = 0, PF = primes[PF_idx];          // using PF = 2, 3, 4, ..., is also ok
  while (N != 1 && (PF * PF <= N)) {      // stop at sqrt(N), but N can get smaller
    while (N % PF == 0) { N /= PF; factors.push_back(PF); }       // remove this PF
      PF = primes[++PF_idx];                                                    // only
      consider primes!
  }
  if (N != 1) factors.push_back(N);         // special case if N is actually a prime
  return factors;                  // if pf exceeds 32-bit integer, you have to change vi
}

// third part

ll numPF(ll N) {
  ll PF_idx = 0, PF = primes[PF_idx], ans = 0;
  while (N != 1 && (PF * PF <= N)) {
    while (N % PF == 0) { N /= PF; ans++; }
    PF = primes[++PF_idx];
  }
  if (N != 1) ans++;
  return ans;
}

ll numDiffPF(ll N) {
  ll PF_idx = 0, PF = primes[PF_idx], ans = 0;
  while (N != 1 && (PF * PF <= N)) {
      if (N % PF == 0) ans++;                                                  // count this pf
       only once
    while (N % PF == 0) N /= PF;
      PF = primes[++PF_idx];
  }
  if (N != 1) ans++;
  return ans;
}

ll sumPF(ll N) {
  ll PF_idx = 0, PF = primes[PF_idx], ans = 0;
  while (N != 1 && (PF * PF <= N)) {
    while (N % PF == 0) { N /= PF; ans += PF; }
      PF = primes[++PF_idx];
  }
  if (N != 1) ans += N;
  return ans;
}

ll numDiv(ll N) {
  ll PF_idx = 0, PF = primes[PF_idx], ans = 1;                    // start from ans = 1
  while (N != 1 && (PF * PF <= N)) {
      ll power = 0;
                                                                                    //
       count the power
    while (N % PF == 0) { N /= PF; power++; }
     ans *= (power + 1);                                                     // according to
       the formula
      PF = primes[++PF_idx];
  }
  if (N != 1) ans *= 2;                        // (last factor has pow = 1, we add 1 to it)
  return ans;
}

ll sumDiv(ll N) {
  ll PF_idx = 0, PF = primes[PF_idx], ans = 1;                    // start from ans = 1
  while (N != 1 && (PF * PF <= N)) {
      ll power = 0;
    while (N % PF == 0) { N /= PF; power++; }
     ans *= ((ll)pow((double)PF, power + 1.0) - 1) / (PF - 1);                // formula
      PF = primes[++PF_idx];
  }
  if (N != 1) ans *= ((ll)pow((double)N, 2.0) - 1) / (N - 1);              // last one
  return ans;
}

ll EulerPhi(ll N) {
  ll PF_idx = 0, PF = primes[PF_idx], ans = N;                    // start from ans = N
  while (N != 1 && (PF * PF <= N)) {
      if (N % PF == 0) ans -= ans / PF;                         // only count unique factor
    while (N % PF == 0) N /= PF;
      PF = primes[++PF_idx];
  }
  if (N != 1) ans -= ans / N;
                                                                            // last factor
  return ans;
}
```

5.3 Pollard Rho

```cpp
#include <cstdio>
using namespace std;

#define abs_val(a) (((a)>=0)?(a):-(a))
typedef long long ll;

ll mulmod(ll a, ll b, ll c) { // returns (a * b) % c, and minimize overflow
  ll x = 0, y = a % c;
  while (b > 0) {
      if (b % 2 == 1) x = (x + y) % c;
       y = (y * 2) % c;
       b /= 2;
```

```
    }
  return x % c;
}

ll gcd(ll a, ll b) { return !b ? a : gcd(b, a % b); }             // standard gcd

ll pollard_rho(ll n) {
  int i = 0, k = 2;
  ll x = 3, y = 3;                              // random seed = 3, other values possible
  while (1) {
      i ++;
      x  = (mulmod(x, x, n) + n - 1) % n;                              // generating function
      ll  d = gcd(abs_val(y - x), n);                                          // the key insight
      if  (d != 1 && d != n) return d;                    // found one non-trivial factor
      if  (i == k) y = x, k *= 2;
} }

int main() {
  ll n = 2063512844981574047LL;       // we assume that n is not a large prime
  ll ans = pollard_rho(n);                    // break n into two non trivial factors
  if (ans > n / ans) ans = n / ans;                    // make ans the smaller factor
  printf("%lld %lld\n", ans, n / ans);   // should be: 1112041493 1855607779
} // return 0;
```

6 Math

6.1 Number theory

```
// 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;
}
```

6.2 Chinese Remainder Theorem

```
bool linearCongruences(const vector<int> &a, const vector<int> &b,
        const vector<int> &m, int &x, int &M) {
   int n =      a.size();
   x = 0; M = 1;
   REP(i, n) {
      int a_ = a[i] % M, b_ = b[i] - a[i] * x, m_ = m[i];
      int y, t, g = extgcd(a_, m_, y, t);
      if (b_ % g) return false;
      b_ /= g; m_ /= g;
      x += M * (y * b_ % m_);
      M *= m_;
   }
   x = (x + M) % M;
   return true;
}
```

6.3 Fast Fourier Transform

```
// {{{ FFT Generic
template<typename T>
struct FFTTrait {
  static T getRoot(int, bool) { assert(false); }
};

template<>
struct FFTTrait<complex<double>> {
  static complex<double> getRoot(int len, bool rev) {
    double ang = 2 * M_PI / len;
    return rev ? complex<double>{cos(ang), sin(ang)} : complex<double>{cos(-ang), sin(-ang)};
  }
};

template<typename T = complex<double>, typename Trait = FFTTrait<T>>
struct FFT {
  const static int MAXLEN = 1 << 20;
  static void reorder(vector<T> &data) {
    int n = SZ(data);
    assert(n > 0 && (n & (n - 1)) == 0);
    int bitCount = __builtin_ctz(n);
    for (int i = 0; i < n; i++) {
      int j = 0;
      for (int b = 0; b < bitCount; b++) {
        if (i & (1 << b)) {
          j |= 1 << (bitCount - 1 - b);
        }
      }
      if (i < j) {
        swap(data[i], data[j]);
      }
    }
  }
  static void fix(vector<T> &data) {
    int n = 1;
    while (n < SZ(data)) n *= 2;
    n *= 2;
    data.resize(n);
  }
  static void fft(vector<T> &data, bool rev) {
    reorder(data);
    int n = SZ(data);
    for (int len = 2; len <= n; len <<= 1) {
      static T roots[MAXLEN];
      int len2 = len >> 1;
      T w = Trait::getRoot(len, rev);
      roots[0] = T(1);
      for (int i = 1; i < len2; i++) {
        roots[i] = roots[i - 1] * w; // binary? can be precomputed
      }
      for (int i = 0; i < n; i += len) {
        for (int j = 0; j < len2; j++) {
          T p = data[i + j];
          T q = roots[j] * data[i + j + len2];
          data[i + j] = p + q;
          data[i + j + len2] = p - q;
        }
      }
    }
    if (rev) {
      for (int i = 0; i < n; i++) {
        data[i] /= n;
      }
    }
  }
};
// }}}
```

6.4 Sqrt Mod

```
// Jacobi Symbol (m/n), m, n  0  and n is odd
// (m/n)==1    x^2 == m (mod n) solvable, -1 unsolvable
#define NEGPOW(e) ((e) % 2 ? -1 : 1)
int jacobi(int a, int m) {
   if (a == 0) return m == 1 ? 1 : 0;
   if (a % 2)  return NEGPOW((a-1)*(m-1)/4)*jacobi(m%a, a);
   else return NEGPOW((m*m-1)/8)*jacobi(a/2, m);
}
int invMod(int a, int m) {
   int x, y;
   if (extgcd(a, m, x, y) == 1) return (x + m) % m;
   else                         return 0; // unsolvable
}
// No solution when: n(p-1)/2 = -1 mod p
int sqrtMod(int n, int p) { //find x: x2 = n (mod p) p is prime
   int S, Q, W, i, m = invMod(n, p);
   for (Q = p - 1, S = 0; Q % 2 == 0; Q /= 2, ++S);
   do { W = rand() % p; } while (W == 0 || jacobi(W, p) != -1);
   for (int R = powMod(n, (Q+1)/2, p), V = powMod(W, Q, p); ;) {
      int z = R * R * m % p;
      for (i = 0; i < S && z % p != 1; z *= z, ++i);
      if (i == 0) return R;
      R = (R * powMod(V, 1 << (S-i-1), p)) % p;
   }
}
int powMod (int a, int b, int p) {
        int res = 1;
        while (b)
                if (b & 1)
                        res = int (res * 1ll * a % p),  --b;
                else
                        a = int (a * 1ll * a % p),  b >>= 1;
        return res;
}
```

6.5 Brent, Miller Rabin

```
#include "../Prime/RabinMiller.h"

long long mul(long long a, long long b, long long mod) {
    if (b == 0) return 0;
    if (b == 1) return a % mod;
    long long mid = mul(a, b >> 1, mod);
    mid = (mid + mid) % mod;
    if (b & 1) return (mid + a) % mod;
    else return mid;
}

long long brent(long long n) {
    if (n == 1) return 1;
    if (!(n & 1)) return 2;
    if (!(n % 3)) return 3;

    const int p[3] = {1, 3, 5};
    long long y, q, x, ys, g, my = 3;
    int i, j, k, m, r, c;

    for (i = 0; i < my; ++i) {
        y = 1; r = 1; q = 1; m = 111; c = p[i];

        do {
            x = y; k = 0;
            for (j = 1; j <= r; ++j) y = (mul(y, y, n) + c) % n;
            do {
                ys = y;
                for (j = 1; j <= min(m, r-k); ++j) {
                    y = (mul(y, y, n) + c) % n;
```

```
                q = mul(q, abs(x - y), n);
            }
            g = __gcd(q, n); k += m;
        } while (k < r && g < 2);
        r <<= 1;
    } while (g < 2);

    if (g == n)
        do {
            ys = (mul(ys, ys, n) + c) % n;
            g = __gcd(abs(x - ys), n);
        } while (g < 2);

    if (g != n) return g;
  }
  return n;
}
```

7 Geometry

7.1 Rectangle in Rectangle

```
// Checks if rectangle of sides x,y fits inside one of sides X,Y
// Not tested with doubles but should work fine :)
// Code as written rejects rectangles that just touch.
bool rect_in_rect(int X, int Y, int x, int y) {
   if (Y > X) swap(Y, X);
   if (y > x) swap(y, x);
   double diagonal = sqrt(double(X)*X + double(Y)*Y);
   if (x < X && y < Y) return true;
   else if (y >= Y || x >= diagonal) return false;
   else {
      double w, theta, tMin = PI/4, tMax = PI/2;
      while (tMax - tMin > EPS) {
         theta = (tMax + tMin)/2.0;
         w = (Y-x*cos(theta))/sin(theta);
         if (w < 0 || x * sin(theta) + w * cos(theta) < X) tMin = theta;
         else tMax = theta;
      }
      return (w > y);
   }
}
```

7.2 Geometry Basic

```
#define EPS 1e-6

double DEG_to_RAD(double d) { return d * M_PI / 180.0; }
double RAD_to_DEG(double r) { return r * 180.0 / M_PI; }

inline int cmp(double a, double b) {
    return (a < b - EPS) ? -1 : ((a > b + EPS) ? 1 : 0);
}

struct Point {
    double x, y;
    Point(double x = 0.0, double y = 0.0) : x(x), y(y) {}

    Point operator + (Point a) { return Point(x+a.x, y+a.y); }
    Point operator - (Point a) { return Point(x-a.x, y-a.y); }
    Point operator * (double k) { return Point(x*k, y*k); }
    Point operator / (double k) { return Point(x/k, y/k); }

    double operator * (Point a) { return x*a.x + y*a.y; } // dot product
    double operator % (Point a) { return x*a.y - y*a.x; } // cross product

    int cmp(Point q) const { if (int t = ::cmp(x,q.x)) return t; return ::cmp(y,q.y); }

    #define Comp(x) bool operator x (Point q) const { return cmp(q) x 0; }
    Comp(>) Comp(<) Comp(==) Comp(>=) Comp(<=) Comp(!=)
    #undef Comp

    Point conj() { return Point(x, -y); }
    double norm() { return x*x + y*y; }

    // Note: There are 2 ways for implementing len():
    // 1. sqrt(norm()) --> fast, but inaccurate (produce some values that are of order X^2)
    // 2. hypot(x, y) --> slow, but much more accurate
    double len() { return sqrt(norm()); }

    Point rotate(double alpha) {
        double cosa = cos(alpha), sina = sin(alpha);
        return Point(x * cosa - y * sina, x * sina + y * cosa);
    }
};

int ccw(Point a, Point b, Point c) {
    return cmp((b-a)%(c-a),0);
}

double angle(Point a, Point o, Point b) { // angle AOB
    a = a - o; b = b - o;
    return acos((a * b) / sqrt(a.norm() * b.norm()));
}

// Distance from p to Line ab (closest Point --> c)
double distToLine(Point p, Point a, Point b, Point &c) {
    Point ap = p - a, ab = b - a;
    double u = (ap * ab) / ab.norm();
    c = a + (ab * u);
    return (p-c).len();
}

// Distance from p to segment ab (closest Point --> c)
double distToLineSegment(Point p, Point a, Point b, Point &c) {
    Point ap = p - a, ab = b - a;
    double u = (ap * ab) / ab.norm();
    if (u < 0.0) {
        c = Point(a.x, a.y);
        return (p - a).len();
    }
    if (u > 1.0) {
        c = Point(b.x, b.y);
        return (p - b).len();
    }
    return distToLine(p, a, b, c);
}

struct Line {
    double a, b, c;
    Point A, B; // Added for polygon intersect line. Do not rely on assumption that these are valid

    Line(double a, double b, double c) : a(a), b(b), c(c) {}

    Line(Point A, Point B) : A(A), B(B) {
        a = B.y - A.y;
        b = A.x - B.x;
        c = - (a * A.x + b * A.y);
    }
    Line(Point P, double m) {
        a = -m; b = 1;
        c = -((a * P.x) + (b * P.y));
    }
    double f(Point A) {
        return a*A.x + b*A.y + c;
    }
};

bool areParallel(Line l1, Line l2) {
    return cmp(l1.a*l2.b, l1.b*l2.a) == 0;
}

bool areSame(Line l1, Line l2) {
    return areParallel(l1 ,l2) && cmp(l1.c*l2.a, l2.c*l1.a) == 0;
}

bool areIntersect(Line l1, Line l2, Point &p) {
    if (areParallel(l1, l2)) return false;
    double dx = l1.b*l2.c - l2.b*l1.c;
    double dy = l1.c*l2.a - l2.c*l1.a;
    double d  = l1.a*l2.b - l2.a*l1.b;
    p = Point(dx/d, dy/d);
    return true;
}

void closestPoint(Line l, Point p, Point &ans) {
    if (fabs(l.b) < EPS) {
        ans.x = -(l.c) / l.a; ans.y = p.y;
        return;
    }
    if (fabs(l.a) < EPS) {
        ans.x = p.x; ans.y = -(l.c) / l.b;
        return;
    }
    Line perp(l.b, -l.a, - (l.b*p.x - l.a*p.y));
    areIntersect(l, perp, ans);
}

void reflectionPoint(Line l, Point p, Point &ans) {
    Point b;
    closestPoint(l, p, b);
    ans = p + (b - p) * 2;
```

```
}
```

7.3 Geometry Circle

```
struct Circle : Point {
    double r;
    Circle(double x = 0, double y = 0, double r = 0) : Point(x, y), r(r) {}
    Circle(Point p, double r) : Point(p), r(r) {}

    bool contains(Point p) { return (*this - p).len() <= r + EPS; }
};

// Find common tangents to 2 circles
// Helper method
void tangents(Point c, double r1, double r2, vector<Line> & ans) {
    double r = r2 - r1;
    double z = sqr(c.x) + sqr(c.y);
    double d = z - sqr(r);
    if (d < -EPS)  return;
    d = sqrt(fabs(d));
    Line l((c.x * r + c.y * d) / z,
           (c.y * r - c.x * d) / z,
           r1);
    ans.push_back(l);
}
// Actual method: returns vector containing all common tangents
vector<Line> tangents(Circle a, Circle b) {
    vector<Line> ans; ans.clear();
    for (int i=-1; i<=1; i+=2)
        for (int j=-1; j<=1; j+=2)
            tangents(b-a, a.r*i, b.r*j, ans);
    for(int i = 0; i < ans.size(); ++i)
        ans[i].c -= ans[i].a * a.x + ans[i].b * a.y;

    vector<Line> ret;
    for(int i = 0; i < (int) ans.size(); ++i) {
        bool ok = true;
        for(int j = 0; j < i; ++j)
            if (areSame(ret[j], ans[i])) {
                ok = false;
                break;
            }
        if (ok) ret.push_back(ans[i]);
    }
    return ret;
}

// Circle & line intersection
vector<Point> intersection(Line l, Circle cir) {
    double r = cir.r, a = l.a, b = l.b, c = l.c + l.a*cir.x + l.b*cir.y;
    vector<Point> res;

    double x0 = -a*c/(a*a+b*b),  y0 = -b*c/(a*a+b*b);
    if (c*c > r*r*(a*a+b*b)+EPS) return res;
    else if (fabs(c*c - r*r*(a*a+b*b)) < EPS) {
        res.push_back(Point(x0, y0) + Point(cir.x, cir.y));
        return res;
    }
    else {
        double d = r*r - c*c/(a*a+b*b);
        double mult = sqrt (d / (a*a+b*b));
        double ax,ay,bx,by;
        ax = x0 + b * mult;
        bx = x0 - b * mult;
        ay = y0 - a * mult;
        by = y0 + a * mult;

        res.push_back(Point(ax, ay) + Point(cir.x, cir.y));
        res.push_back(Point(bx, by) + Point(cir.x, cir.y));
        return res;
    }
}

double commonCircleArea(Circle c1, Circle c2) { //return the common area of two circle
    double d = hypot(c1.x-c2.x, c1.y-c2.y), area;
    if (c1.r+c2.r <= d) area = 0;
    else if (c2.r+d <= c1.r) area = (c1.r * c1.r - c2.r * c2.r) * M_PI;
    else if (c1.r+d <= c2.r) area = (c2.r * c2.r - c1.r * c1.r) * M_PI;
    else {
        double p1 = c2.r * c2.r * acos((d*d + c2.r*c2.r - c1.r*c1.r) / (2*d*c2.r));
        double p2 = c1.r * c1.r * acos((d*d + c1.r*c1.r - c2.r*c2.r) / (2*d*c1.r));
        double p3 = 0.5 * sqrt((-d+c2.r+c1.r) * (d+c2.r-c1.r) * (d+c1.r-c2.r) * (d+c1.r+c2.r));
        area = p1 + p2 - p3;
    }
    return area;
}

// Given 2 circles: (0, 0, r) and (c2.x, c2.y, c2.r)
// Intersections are intersections with line Ax + By + C where
// A = -2 * c2.x
// B = -2 * c2.y
// C = c2.x^2 + c2.y^2 + r^2 - c2.r^2
```

7.4 Geometry Polygon

```
typedef vector< Point > Polygon;

// Convex Hull: returns minimum number of vertices. Should work when N <= 1 and when all points
// are collinear
struct comp_hull {
    Point pivot;
    bool operator() (Point q,Point w) {
        Point Q = q - pivot, W = w - pivot;
        double R = Q % W;
        if (cmp(R,0)) return R < 0;
        return cmp(Q*Q,W*W) < 0;
    }
};
Polygon convex_hull(Polygon p) { // minimum vertices
    int j = 0, k, n = p.size();
    Polygon r(n);
    if (!n) return r;
    comp_hull comp;
    comp.pivot = *min_element(p.begin(),p.end());
    sort(p.begin(),p.end(),comp);
    for(int i = 0; i < n; ++i) {
        while (j > 1 && ccw(r[j-1],r[j-2],p[i]) <= 0) j--;
        r[j++] = p[i];
    }
    r.resize(j);
    if (r.size() >= 2 && r.back() == r.front()) r.pop_back();
    return r;
}

// Area, perimeter, centroid
double signed_area(Polygon p) {
    double area = 0;
    for(int i = 0; i < p.size(); i++) {
        int j = (i+1) % p.size();
        area += p[i].x*p[j].y - p[j].x*p[i].y;
    }
    return area / 2.0;
}
double area(const Polygon &p) {
    return fabs(signed_area(p));
}
Point centroid(Polygon p) {
    Point c(0,0);
    double scale = 6.0 * signed_area(p);
    for (int i = 0; i < p.size(); i++){
        int j = (i+1) % p.size();
        c = c + (p[i]+p[j])*(p[i].x*p[j].y - p[j].x*p[i].y);
    }
    return c / scale;
}
double perimeter(Polygon P) {
    double res = 0;
    for(int i = 0; i < P.size(); ++i) {
        int j = (i + 1) % P.size();
        res += (P[i] - P[j]).len();
    }
    return res;
}
// Is convex: checks if polygon is convex. Assume there are no 3 collinear points
bool is_convex(const Polygon &P) {
    int sz = (int) P.size();
    if (sz <= 2) return false;
    int isLeft = ccw(P[0], P[1], P[2]);
    for (int i = 1; i < sz; i++)
        if (ccw(P[i], P[(i+1) % sz], P[(i+2) % sz]) * isLeft < 0)
            return false;
    return true;
}

// Inside polygon: O(N). Works with any polygon
// Does not work when point is on edge. Should check separately
bool in_polygon(const Polygon &P, Point pt) {
    if ((int)P.size() == 0) return false;
    double sum = 0;
    for (int i = 0; i < (int)P.size(); i++) {
        Point Pj = P[(i+1) % P.size()];
        if (ccw(pt, P[i], Pj) > 0)
            sum += angle(P[i], pt, Pj);
        else sum -= angle(P[i], pt, Pj);
```

```
    }
    return fabs(fabs(sum) - 2*M_PI) < EPS;
}

// Check point in convex polygon, O(logN)
// Source: http://codeforces.com/contest/166/submission/1392387
// On edge --> false
#define Det(a,b,c) ((double)(b.x-a.x)*(double)(c.y-a.y)-(double)(b.y-a.y)*(c.x-a.x))
bool in_convex(vector<Point>& l, Point p){
    int a = 1, b = l.size()-1, c;
    if (Det(l[0], l[a], l[b]) > 0) swap(a,b);
    // Allow on edge --> if (Det... > 0 || Det ... < 0)
    if (Det(l[0], l[a], p) >= 0 || Det(l[0], l[b], p) <= 0) return false;
    while(abs(a-b) > 1) {
        c = (a+b)/2;
        if (Det(l[0], l[c], p) > 0) b = c; else a = c;
    }
    // Alow on edge --> return Det... <= 0
    return Det(l[a], l[b], p) < 0;
}


// Cut a polygon with a line. Returns one half.
// To return the other half, reverse the direction of Line l (by negating l.a, l.b)
// The line must be formed using 2 points
Polygon polygon_cut(Polygon P, Line l) {
    Polygon Q;
    for(int i = 0; i < P.size(); ++i) {
        Point A = P[i], B = (i == P.size()-1) ? P[0] : P[i+1];
        if (ccw(l.A, l.B, A) != -1) Q.push_back(A);
        if (ccw(l.A, l.B, A)*ccw(l.A, l.B, B) < 0) {
            Point p; areIntersect(Line(A, B), l, p);
            Q.push_back(p);
        }
    }
    return Q;
}

// Find intersection of 2 polygons
// Helper method
bool intersect_1pt(Point a, Point b,
    Point c, Point d, Point &r) {
    double D =  (b - a) % (d - c);
    if (cmp(D, 0) == 0) return false;
    double t =  ((c - a) % (d - c)) / D;
    double s = -((a - c) % (b - a)) / D;
    r = a + (b - a) * t;
    return cmp(t, 0) > 0 && cmp(t, 1) < 0 && cmp(s, 0) > 0 && cmp(s, 1) < 0;
}
Polygon convex_intersect(Polygon P, Polygon Q) {
    const int n = P.size(), m = Q.size();
    int a = 0, b = 0, aa = 0, ba = 0;
    enum { Pin, Qin, Unknown } in = Unknown;
    Polygon R;
    do {
        int a1 = (a+n-1) % n, b1 = (b+m-1) % m;
        double C = (P[a] - P[a1]) % (Q[b] - Q[b1]);
        double A = (P[a1] - Q[b]) % (P[a] - Q[b]);
        double B = (Q[b1] - P[a]) % (Q[b] - P[a]);
        Point r;
        if (intersect_1pt(P[a1], P[a], Q[b1], Q[b], r)) {
            if (in == Unknown) aa = ba = 0;
            R.push_back( r );
            in = B > 0 ? Pin : A > 0 ? Qin : in;
        }
        if (C == 0 && B == 0 && A == 0) {
            if (in == Pin) { b = (b + 1) % m; ++ba; }
            else            { a = (a + 1) % m; ++aa; }
        } else if (C >= 0) {
            if (A > 0) { if (in == Pin) R.push_back(P[a]); a = (a+1)%n; ++aa; }
            else       { if (in == Qin) R.push_back(Q[b]); b = (b+1)%m; ++ba; }
        } else {
            if (B > 0) { if (in == Qin) R.push_back(Q[b]); b = (b+1)%m; ++ba; }
            else       { if (in == Pin) R.push_back(P[a]); a = (a+1)%n; ++aa; }
        }
    } while ( (aa < n || ba < m) && aa < 2*n && ba < 2*m );
    if (in == Unknown) {
        if (in_convex(Q, P[0])) return P;
        if (in_convex(P, Q[0])) return Q;
    }
    return R;
}

// Find the diameter of polygon.
// Rotating callipers
double convex_diameter(Polygon pt) {
    const int n = pt.size();
    int is = 0, js = 0;
    for (int i = 1; i < n; ++i) {
        if (pt[i].y > pt[is].y) is = i;
        if (pt[i].y < pt[js].y) js = i;
    }
    double maxd = (pt[is]-pt[js]).norm();
    int i, maxi, j, maxj;
    i = maxi = is;
    j = maxj = js;
    do {
        int jj = j+1; if (jj == n) jj = 0;
        if ((pt[i] - pt[jj]).norm() > (pt[i] - pt[j]).norm()) j = (j+1) % n;
        else i = (i+1) % n;
        if ((pt[i]-pt[j]).norm() > maxd) {
            maxd = (pt[i]-pt[j]).norm();
            maxi = i; maxj = j;
        }
    } while (i != is || j != js);
    return maxd; /* farthest pair is (maxi, maxj). */
}

// Closest pair
// Source: e-maxx.ru
#define upd_ans(x, y) {}
#define MAXN 100
double mindist = 1e20; // will be the result
void rec(int l, int r, Point a[]) {
    if (r - l <= 3) {
        for (int i=l; i<=r; ++i)
            for (int j=i+1; j<=r; ++j)
                    upd_ans(a[i], a[j]);
        sort(a+l, a+r+1); // compare by y
        return;
    }

    int m = (l + r) >> 1;
    int midx = a[m].x;
    rec(l, m, a), rec(m+1, r, a);
    static Point t[MAXN];
    merge(a+l, a+m+1, a+m+1, a+r+1, t); // compare by y
    copy(t, t+r-l+1, a+l);

    int tsz = 0;
    for (int i=l; i<=r; ++i)
        if (fabs(a[i].x - midx) < mindist) {
            for (int j=tsz-1; j>=0 && a[i].y - t[j].y < mindist; --j)
                upd_ans(a[i], t[j]);
            t[tsz++] = a[i];
        }
}

// Pick theorem
// Given non-intersecting polygon.
// S = area
// I = number of integer points strictly Inside
// B = number of points on sides of polygon
// S = I + B/2 - 1
```

7.5 Smallest Enclosing Circle

```
// Smallest enclosing circle:
// Given N points. Find the smallest circle enclosing these points.
// Amortized complexity: O(N)

struct SmallestEnclosingCircle {
    Circle getCircle(vector<Point> points) {
        assert(!points.empty());

        random_shuffle(points.begin(), points.end());
        Circle c(points[0], 0);
        int n = points.size();

        for (int i = 1; i < n; i++)
            if ((points[i] - c).len() > c.r + EPS)
            {
                c = Circle(points[i], 0);
                for (int j = 0; j < i; j++)
                    if ((points[j] - c).len() > c.r + EPS)
                    {
                        c = Circle((points[i] + points[j]) / 2, (points[i] - points[j]).len() / 2);
                        for (int k = 0; k < j; k++)
                            if ((points[k] - c).len() > c.r + EPS)
                                c = getCircumcircle(points[i], points[j], points[k]);
                    }
            }

        return c;
    }

    Circle getCircumcircle(Point a, Point b, Point c) {
        double d = 2.0 * (a.x * (b.y - c.y) + b.x * (c.y - a.y) + c.x * (a.y - b.y));
        assert(fabs(d) > EPS);
```

```
        double x = (a.norm() * (b.y - c.y) + b.norm() * (c.y - a.y) + c.norm() * (a.y - b.y)) / d;
        double y = (a.norm() * (c.x - b.x) + b.norm() * (a.x - c.x) + c.norm() * (b.x - a.x)) / d;
        Point p(x, y);
        return Circle(p, (p - a).len());
    }
};
```

7.6 Geometry Complex

```
// add : a + b
// scalar multi : r * a
// dot product : (conj(a) * b).x
// cross product : (conj(a) * b).y
// squared distance : norm(a - b)
// euclidean distance : abs(a - b)
// angle of elevation : arg(b - a)
// slope of line (a, b) : tan(arg(b - a))
// polar to cartesian : polar(r, theta)
// rotation about the origin : a * polar(1.0, theta)
// rotation about pivot p : (a - p) * polar(1.0, theta) + p
// angle ABC: abs(remainder(arg(a - b) - arg(c - b), 2.0 * M_PI)) distance: [-PI, PI]
// project p onto vector v : v * dot(p, v) / norm(v)
// project p onto line(a, b) : a + (b - a) * dot(p - a, b - a) / norm(b - a)
// reflect p across line (a, b) : a + conj((p - a) / (b - a)) * (b - a)
// relect p across line (a, b) : a + conj((p - a) / (b - a)) * (b - a)
// intersection of line(a, b) and (p, q) :
point intersection(point a, point b, point c) {
  double c1 = cross(p - a, b - a), c2 = cross(q - a, b - a);
  return (c1 * q - c2 * p) / (c1 - c2); //undefined if parallel
}
// read:
template<class T>
istream &operator>>(istream &is, complex<T> &p) {
  T value;
  is >> value;
  p.real(value);
  is >> value;
  p.imag(value);
  return is;
}
```

7.7 Latitude Longitude

```
/*
Converts from rectangular coordinates to latitude/longitude and vice
versa. Uses degrees (not radians).
*/

#include <iostream>
#include <cmath>

using namespace std;

struct ll
{
  double r, lat, lon;
};

struct rect
{
  double x, y, z;
};

ll convert(rect& P)
{
  ll Q;
  Q.r = sqrt(P.x*P.x+P.y*P.y+P.z*P.z);
  Q.lat = 180/M_PI*asin(P.z/Q.r);
  Q.lon = 180/M_PI*acos(P.x/sqrt(P.x*P.x+P.y*P.y));

  return Q;
}

rect convert(ll& Q)
{
  rect P;
  P.x = Q.r*cos(Q.lon*M_PI/180)*cos(Q.lat*M_PI/180);
  P.y = Q.r*sin(Q.lon*M_PI/180)*cos(Q.lat*M_PI/180);
  P.z = Q.r*sin(Q.lat*M_PI/180);

  return P;
}

int main()
{
  rect A;
  ll B;

  A.x = -1.0; A.y = 2.0; A.z = -3.0;

  B = convert(A);
  cout << B.r << " " << B.lat << " " << B.lon << endl;

  A = convert(B);
  cout << A.x << " " << A.y << " " << A.z << endl;
}
```

7.8 Delaunay

```
#include <iostream>
#include <cstdio>
#include <algorithm>
#include <vector>
#include <complex>
#include <list>
#include <array>
#include <set>
using namespace std;

typedef long double ld;
typedef complex<ld> Pnt;
typedef array<int, 2> Edge;
typedef array<int, 3> Tri;
typedef pair<Pnt, ld> Circ;
const ld oo = 1e9;
const ld eps = 1e-6;
int n;

Circ compute_circumcircle(Pnt a, Pnt b, Pnt c) {
  b -= a;
  c -= a;
  auto center = (b*norm(c)-c*norm(b)) / (b*conj(c)-c*conj(b));
  return {center+a, norm(center)};
}

vector<Tri> delaunay(vector<Pnt> pnts) {
  pnts.push_back({-oo, -oo});
  pnts.push_back({oo, -oo});
  pnts.push_back({oo, oo});
  pnts.push_back({-oo, oo});
  list<pair<Tri, Circ>> tcs;
  tcs.push_back({{n, n+1, n+2}, {0, 2*oo*oo}});
  tcs.push_back({{n+2, n+3, n}, {0, 2*oo*oo}});
  for (int i = 0; i < n; i++) {
    set<Edge> edges;
    for (auto it = begin(tcs); it != end(tcs); ) {
      auto circ = it->second;
      ld d = norm(pnts[i] - circ.first) - circ.second;
      if (d < -eps) {
        auto tri = it->first;
        auto j = *prev(end(tri));
        for (int k: tri) {
          auto ite = edges.find({k, j});
          if (ite == end(edges)) {
            edges.insert({j, k});
          } else {
            edges.erase(ite);
          }
          j = k;
        }
        it = tcs.erase(it);
      } else {
        it++;
      }
    }
    int j0 = (*begin(edges))[0];
    for (int j = j0; ; ) {
      int k = (*edges.lower_bound({j, 0}))[1];
      Circ circ = compute_circumcircle(pnts[i], pnts[j], pnts[k]);
      tcs.push_back({{i, j, k}, circ});
      j = k;
      if (j == j0) break;
    }
  }
  vector<Tri> tris;
  for (auto& tc: tcs) {
    auto& tri = tc.first;
    bool flag = false;
    for (int j: tri) {
      flag = flag || j >= n;
```

```
    }
    if (flag) continue;
    tris.push_back(tri);
  }
  return tris;
}
```

7.9 Convex hull

```
// Compute the 2D convex hull of a set of points using the monotone chain
// algorithm.  Eliminate redundant points from the hull if REMOVE_REDUNDANT is
// #defined.
//
// Running time: O(n log n)
//
//   INPUT:   a vector of input points, unordered.
//   OUTPUT:  a vector of points in the convex hull, counterclockwise, starting
//            with bottommost/leftmost point

#include <cstdio>
#include <cassert>
#include <vector>
#include <algorithm>
#include <cmath>
// BEGIN CUT
#include <map>
// END CUT

using namespace std;

#define REMOVE_REDUNDANT

typedef double T;
const T EPS = 1e-7;
struct PT {
  T x, y;
  PT() {}
  PT(T x, T y) : x(x), y(y) {}
  bool operator<(const PT &rhs) const { return make_pair(y,x) < make_pair(rhs.y,rhs.x); }
  bool operator==(const PT &rhs) const { return make_pair(y,x) == make_pair(rhs.y,rhs.x); }
};

T cross(PT p, PT q) { return p.x*q.y-p.y*q.x; }
T area2(PT a, PT b, PT c) { return cross(a,b) + cross(b,c) + cross(c,a); }

#ifdef REMOVE_REDUNDANT
bool between(const PT &a, const PT &b, const PT &c) {
  return (fabs(area2(a,b,c)) < EPS && (a.x-b.x)*(c.x-b.x) <= 0 && (a.y-b.y)*(c.y-b.y) <= 0);
}
#endif

void ConvexHull(vector<PT> &pts) {
  sort(pts.begin(), pts.end());
  pts.erase(unique(pts.begin(), pts.end()), pts.end());
  vector<PT> up, dn;
  for (int i = 0; i < pts.size(); i++) {
    while (up.size() > 1 && area2(up[up.size()-2], up.back(), pts[i]) >= 0) up.pop_back();
    while (dn.size() > 1 && area2(dn[dn.size()-2], dn.back(), pts[i]) <= 0) dn.pop_back();
    up.push_back(pts[i]);
    dn.push_back(pts[i]);
  }
  pts = dn;
  for (int i = (int) up.size() - 2; i >= 1; i--) pts.push_back(up[i]);

#ifdef REMOVE_REDUNDANT
  if (pts.size() <= 2) return;
  dn.clear();
  dn.push_back(pts[0]);
  dn.push_back(pts[1]);
  for (int i = 2; i < pts.size(); i++) {
    if (between(dn[dn.size()-2], dn[dn.size()-1], pts[i])) dn.pop_back();
    dn.push_back(pts[i]);
  }
  if (dn.size() >= 3 && between(dn.back(), dn[0], dn[1])) {
    dn[0] = dn.back();
    dn.pop_back();
  }
  pts = dn;
#endif
}

// BEGIN CUT
// The following code solves SPOJ problem #26: Build the Fence (BSHEEP)

int main() {
  int t;
  scanf("%d", &t);
  for (int caseno = 0; caseno < t; caseno++) {
    int n;
    scanf("%d", &n);
    vector<PT> v(n);
    for (int i = 0; i < n; i++) scanf("%lf%lf", &v[i].x, &v[i].y);
    vector<PT> h(v);
    map<PT,int> index;
    for (int i = n-1; i >= 0; i--) index[v[i]] = i+1;
    ConvexHull(h);

    double len = 0;
    for (int i = 0; i < h.size(); i++) {
      double dx = h[i].x - h[(i+1)%h.size()].x;
      double dy = h[i].y - h[(i+1)%h.size()].y;
      len += sqrt(dx*dx+dy*dy);
    }

    if (caseno > 0) printf("\n");
    printf("%.2f\n", len);
    for (int i = 0; i < h.size(); i++) {
      if (i > 0) printf(" ");
      printf("%d", index[h[i]]);
    }
    printf("\n");
  }
}

// END CUT
```

8 Miscellaneaous

8.1 Calendar Java

```
/*
Constructors:
GregorianCalendar()
GregorianCalendar(int year, int month, int dayOfMonth)
GregorianCalendar(int year, int month, int dayOfMonth, int hour, int minute)
Methods:
add(int field, int amount)
get(int field)
set(int field, int value)
Fields:
GregorianCalendar.YEAR
GregorianCalendar.MONTH
GregorianCalendar.WEEK_OF_YEAR
GregorianCalendar.DAY_OF_MONTH
GregorianCalendar.DAY_OF_WEEK
GregorianCalendar.DATE
*/
```

8.2 Dynamic Convex Hull DP (Max)

```
const ll is_query = -(1LL<<62);
struct Line {
    ll m, b;
    mutable function<const Line*()> succ;
    bool operator<(const Line& rhs) const {
        if (rhs.b != is_query) return m < rhs.m;
        const Line* s = succ();
        if (!s) return 0;
        ll x = rhs.m;
        return b - s->b < (s->m - m) * x;
    }
};
struct HullDynamic : public multiset<Line> { // will maintain upper hull for maximum
    bool bad(iterator y) {
        auto z = next(y);
        if (y == begin()) {
            if (z == end()) return 0;
            return y->m == z->m && y->b <= z->b;
        }
        auto x = prev(y);
        if (z == end()) return y->m == x->m && y->b <= x->b;
        return (x->b - y->b)*(z->m - y->m) >= (y->b - z->b)*(y->m - x->m);
    }
    void insert_line(ll m, ll b) {
        auto y = insert({ m, b });
        y->succ = [=] { return next(y) == end() ? 0 : &*next(y); };
        if (bad(y)) { erase(y); return; }
        while (next(y) != end() && bad(next(y))) erase(next(y));
```

```
        while (y != begin() && bad(prev(y))) erase(prev(y));
    }
    ll eval(ll x) {
        auto l = *lower_bound((Line) { x, is_query });
        return l.m * x + l.b;
    }
};
```

8.3 Floyd Cycle Finding

```
pii floyd(int x0) {
  int tortoise = f(x0), hare = f(f(x0));
  while (tortoise != hare) {
    tortoise = f(tortoise);
    hare = f(f(hare));
  }
  int mu = 0; hare = 0;
  while (tortoise != hare) {
    tortoise = f(tortoise);
    hare = f(hare);
    mu++;
  }
  int lambda = 1;
  hare = f(tortoise);
  while (tortoise != hare) {
    hare = f(hare);
    lambda++;
  }
  return mp(mu, lambda);
}
```

8.4 Josephus

```
/*
case k = 2 -> if n = 2^m + k and 0 <= k < 2^m, then f(n) =
2k+1
general case (0-based) -> f(n, k) = (f(n - 1, k) + k) mod n,
f(1, k) = 0
*/
```

8.5 Knight Shortest Path

```
// knight shortest path
int f(int x1, int y1, int x2, int y2)
{
 int dx=abs(x2-x1);
 int dy=abs(y2-y1);
 int lb=(dx+1)/2;
 lb>?=(dy+1)/2;
 lb>?=(dx+dy+2)/3;
 while ((lb%2)!=(dx+dy)%2) lb++;
 if (abs(dx)==1 && dy==0) return 3;
 if (abs(dy)==1 && dx==0) return 3;
 if (abs(dx)==2 && abs(dy)==2) return 4;
 return lb;
}
```

8.6 DP Knuth

```
// http://codeforces.com/blog/entry/8219
// Original Recurrence:
//   dp[i][j] = min(dp[i][k] + dp[k][j]) + C[i][j]   for k = i+1..j-1
// Necessary & Sufficient Conditions:
//   A[i][j-1] <= A[i][j] <= A[i+1][j]
//   with A[i][j] = smallest k that gives optimal answer
// Also applicable if the following conditions are met:
//   1. C[a][c] + C[b][d] <= C[a][d] + C[b][c] (quadrangle inequality)
//   2. C[b][c] <= C[a][d]                     (monotonicity)
//   for all a <= b <= c <= d
// To use:
//   Calculate dp[i][i] and A[i][i]
//
//   FOR(len = 1..n-1)
//     FOR(i = 1..n-len) {
//       j = i + len
//       FOR(k = A[i][j-1]..A[i+1][j])
//         update(dp[i][j])
//     }

// OPTCUT
#include "../template.h"

const int MN = 2011;
int a[MN], dp[MN][MN], C[MN][MN], A[MN][MN];
int n;

int main() {
    while (cin >> n) {
        FOR(i,1,n) {
            cin >> a[i];
            a[i] += a[i-1];
        }
        FOR(i,1,n) FOR(j,i,n) C[i][j] = a[j] - a[i-1];

        FOR(i,1,n) dp[i][i] = 0, A[i][i] = i;

        FOR(len,1,n-1)
            FOR(i,1,n-len) {
                int j = i + len;
                dp[i][j] = 2000111000;
                FOR(k,A[i][j-1],A[i+1][j]) {
                    int cur = dp[i][k-1] + dp[k][j] + C[i][j];
                    if (cur < dp[i][j]) {
                        dp[i][j] = cur;
                        A[i][j] = k;
                    }
                }
            }
        cout << dp[1][n] << endl;
    }
}
```

8.7 C++ Template

```
#ifdef __DEBUG
#define debug if (true)
#define _GLIBCXX_DEBUG
#else
#define debug if (false)
#endif

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

#define mp make_pair
#define pb push_back
#define fi first
#define se second
#define ALL(a) begin(a), end(a)
#define SZ(a) ((int)(a).size())

typedef long long ll;
typedef pair<int, int> pii;
typedef vector<int> vi;

int main() {
  ios::sync_with_stdio(false);
  cin.tie(nullptr);

  return 0;
}
```

8.8 Java Template

```
import java.util.*;
import java.io.*;

class Main {
  FastScanner in;
  PrintWriter out;

  public void solve() throws IOException {

  }

  public void run() {
    try {
```

```
      in = new FastScanner(System.in);
      out = new PrintWriter(System.out);

      solve();

      out.close();
    } catch (IOException e) {
      e.printStackTrace();
    }
  }

  class FastScanner {
    BufferedReader br;
    StringTokenizer st;

    FastScanner(InputStream is) {
      br = new BufferedReader(new InputStreamReader(is));
    }

    FastScanner(File f) {
      try {
        br = new BufferedReader(new FileReader(f));
      } catch (FileNotFoundException e) {
        e.printStackTrace();
      }
    }

    String next() {
      while (st == null || !st.hasMoreTokens()) {
        try {
          st = new StringTokenizer(br.readLine());
        } catch (IOException e) {
          e.printStackTrace();
        }
      }
      return st.nextToken();
    }

    int nextInt() {
      return Integer.parseInt(next());
    }
  }

  public static void main(String[] arg) {
    new Main().run();
  }
}
```

8.9 pika

```
#!/usr/bin/env bash
set -e
if [ "$2" == "-d" ]; then
        g++ -o "$1" "$1.cpp" -std=c++14 -D_GLIBCXX_DEBUG -D__DEBUG -g
else
        g++ -o "$1" "$1.cpp" -std=c++14 -O2 -Wall
fi
echo "COMPILE OK"
time "./$1" < "$1.in"
```

9 SHK

9.1 SHK C++

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

using namespace std;
  const double PI = acos(-1);
   #define INF 1E9
   #define endl '\n'
   #define pb push_back
   // A lot of typedefs
    typedef long long ll;
   // struct
    struct mystruct {
        int counter;
      };
   //custom hashing
    struct custom_hash {
       inline std::size_t operator()(const std::pair<int,int> & v) const {
              return v.first*31+v.second*7;
           }
      };
   // pq/set custom comparator, will get reversed (at least in pq)
    class mycomp {
       public:
        bool operator() (mystruct a, mystruct b) {
           return a.counter > b.counter;
           }
      };
   // sort custom comparator
    bool customcompare(mystruct a, mystruct b) {
       return a.counter > b.counter;
      }
int main ()
{
  ios::sync_with_stdio(false);
  cin.tie(nullptr);
  return 0;
}
```

9.2 SHK Python

```
import sys

class SamInput(object):
  def __init__(self):
     self.inp = []
     for i in sys.stdin:
          i  = i.replace("\n", "")
          j  = list(i.split())
        self.inp.append(j)

  def readln(self):
      if (len(self.inp) == 0) :
        return False
     else:
        return str.join(" ", self.inp.pop(0))

  def read(self):
      if (len(self.inp) == 0):
        return False
     while (len(self.inp[0]) == 0):
        self.inp.pop(0)
         if (len(self.inp) == 0):
           return False
     return self.inp[0].pop(0)
```

9.3 SHK Shortcuts

```
Comment highlighted code
Ctrl + Shift + C
Uncomment highlighted code
Ctrl + Shift + X

Indent block.
Tab
Dedent block.
Shift + Tab

Line cut.
Ctrl + L
Line copy.
Ctrl + Shift + T

g++ x.cpp -std=c++11 -o x
x.exe < test.txt
```