Reference Material Part 2

HaKings

Contents

1 Floyd Warshall

```
#define MAX_V 400
void floydWarshall(Graph &g, int distance[MAX_V][MAX_V]) {
  FOR(i, 0, g.V-1)
    FOR(j, i, g.V)
      distance[i][j] = distance[j][i] = INF*(i != j);
  FOR(i, 0, g.V)
    FOR(j, 0, g.edges[i].size())
      distance[i][g.edges[i][j].to] = g.edges[i][j].weight;
  FOR(i, 0, g.V)
    FOR(j, 0, g.V)
      FOR(k, 0, g.V)
        distance[j][k] = min(distance[j][k], distance[j][i] + distance[i][k]);
}
```

2 Lists Graph

```
struct Edge {
  int to, weight;
    int backEdge, strong, type, visited; //optional
  Edge(int to, int weight = 1) : to(to), weight(weight), strong(0), visited(0) {}
};
struct Graph {
  int V; bool undirected;
  vector<vector<Edge> > edges;
  Graph(int v, bool undirected) : V(v), undirected(undirected) { edges.assign(V, vector<Edge>()); }
  void connect(int from, Edge edge) {
    edges[from].pb(edge);
    if(undirected) {
      int aux = edge.to;
      edge.to = from;
      edges[aux].pb(edge);
            edges[from].back().backEdge = edges[aux].size() - 1; //optional
            edges[aux].back().backEdge = edges[from].size() - 1; //optional
    }
  }
};
```

3 Matrix Graph

```
struct MatrixEdge {
  int weight;
  MatrixEdge(int weight = 1) : weight(weight) { }
};
struct MatrixGraph {
  int V; bool undirected;
  vector<vector<Edge> > edges;
  MatrixGraph(int v, bool undirected) : V(v), undirected(undirected) {
    edges.assign(V, vector<Edge>(V, Edge(0)));
  }
  void connect(int from, int to, Edge edge = Edge(1)) {
    edges[from][to] = edge;
    if(undirected) edges[to][from] = edge;
  }
};
```

4 Union Find

```
struct UnionFindDS {
  vi tree;
  UnionFindDS(int n) { FOR(i, 0, n) tree.pb(i); }
  int root(int i) { return tree[i] == i ? i : tree[i] = root(tree[i]); }
  bool connected(int i, int j) {return root(i) == root(j);}
  void connect(int i, int j) { tree[root(i)] = tree[root(j)]; }
};

struct UnionFindDS2 {
  vi tree, sizes;
  int N;
  UnionFindDS2(int n) : N(n) {
    tree.reserve(n);
    FOR(i, 0, n) tree[i] = i;
    sizes.assign(n, 1);
  }
  int root(int i) { return (tree[i] == i) ? i : (tree[i] = root(tree[i]));}
  int countSets() { return N;}
  int getSize(int i) { return sizes[root(i)];}
  bool connected(int i, int j) { return root(i) == root(j);}
  void connect(int i, int j) {
    int ri = root(i), rj = root(j);
    if(ri != rj) {
      N--;
      sizes[rj] += sizes[ri];
      tree[ri] = rj;
    }
  }
};
```

5 Interval Tree

```
#define LCHILD(n) ((n)->parent->left == (n))
class IntervalTree {
  struct Node {
```

```
  Node *left, *right, *parent;
  set<int> intervals;
  int key, area;
  bool isLeaf;
  void unLeaf(int k) {
    isLeaf = 0, key = k;
    left = new Node(this), right = new Node(this);
  }
  Node(Node *p) : parent(p), isLeaf(1), area(0), left(NULL), right(NULL) {}
  Node(int k, Node *p) : parent(p), area(0), left(NULL), right(NULL) { unLeaf(k); }
};
Node *root;
void insert(Node *node, int key) {
  Node *parent = find(node, key);
  if(parent->key == key) return;
  (key < parent->key ? parent->left : parent->right)->unLeaf(key);
}
void insert(Node *node, int interval, int a, int b, int imin, int imax) {
  if(a <= imin && b >= imax) { node->area = imax-imin; node->intervals.insert(interval); return; }
  if(a < node->key)
    insert(node->left, interval, a, b, imin, node->key);
  if(b > node->key)
    insert(node->right, interval, a, b, node->key, imax);
  if(node->intervals.size() == 0)
    node->area = (node->left ? node->left->area : 0) + (node->right ? node->right->area : 0);
}
Node * find(Node *node, int key) {
  if(key == node->key) { return node; }
  if(key < node->key) return !node->left->isLeaf ? find(node->left, key) : node;
  return !node->right->isLeaf ? find(node->right, key) : node;
}
void query(Node *node, int a, int b, int imin, int imax, set<int> &result) {
  if(!node) return;
  result.insert(node->intervals.begin(), node->intervals.end());
  if(a < node->key)
    query(node->left, a, b, imin, node->key, result);
  if(b >= node->key)
    query(node->right, a, b, node->key, imax, result);
}
void erase(Node *node, int interval, int a, int b, int imin, int imax) {
  if(a <= imin && b >= imax) {
    node->intervals.erase(interval);
    if(node->intervals.size() == 0)
      node->area = (node->left ? node->left->area : 0) + (node->right ? node->right->area : 0);
    return;
    }
    if(a < node->key)
      erase(node->left, interval, a, b, imin, node->key);
    if(b > node->key)
      erase(node->right, interval, a, b, node->key, imax);
    if(node->intervals.size() == 0)
        node->area = (node->left ? node->left->area : 0) + (node->right ? node->right->area : 0);
  }
  void dealloc(Node *node) { if(node->left) dealloc(node->left); if(node->right) dealloc(node->right); delete node; }
public:
  IntervalTree() : root(0) {}
  ~IntervalTree() { if(root) dealloc(root); }
  void insert(int key) { if(root) insert(root, key); else root = new Node(key, 0); }
  bool contains(int key) { return root && find(root, key)->key == key; }
  void insert(int interval, int a, int b) { insert(a); insert(b+1); insert(root, interval, a, b+1, -INF, INF); }
  set<int> query(int a, int b) { set<int> s; if(root) query(root, a, b, -INF, INF, s); return s; }
  void erase(int interval, int a, int b) { erase(root, interval, a, b+1, -INF, INF); }
  int getArea() { if(root) return root->area - 1; return 0; }
};
```

6 Splay Tree

```
#define LCHILD(n) ((n)->parent->left == (n))
template< typename K, typename Compare = less<K> >
class SplayTree {
  Compare compare;
  struct Node {
    Node *left, *right, *parent;
    K key;
    Node(K k, Node *p) : key(k), parent(p), left(0), right(0) {}
  };
  Node *root;
  void insert(Node *node, K key) {
    Node *parent = find(node, key);
    if(parent->key == key) return;
    (compare(key, parent->key) ? parent->left : parent->right) = new Node(key, parent);
  }
  Node * find(Node *node, K key) {
    if(key == node->key) { splay(node); return node; }
    if(compare(key, node->key)) return node->left ? find(node->left, key) : node;
    return node->right ? find(node->right, key) : node;
  }
  void erase(Node *node, K key) {
    node = find(node, key);
    if(node->key != key) return;
    if(node == root && !node->left && !node->right)
      root = 0, delete node;
    else if(node->left && node->right) {
      Node *pred = node->left;
      while(pred->right) pred = pred->right;
      swap(node->key, pred->key);
      if(pred != root) (LCHILD(pred) ? pred->parent->left : pred->parent->right) = pred->left ? pred->left : pred->right;
      if(pred->left || pred->right) (pred->left ? pred->left : pred->right)->parent = pred->parent;
      delete pred;
    } else {
      if(node == root) root = node->left ? node->left : node->right;
      else (LCHILD(node) ? node->parent->left : node->parent->right) = node->left ? node->left : node->right;
      if(node->left || node->right) (node->left ? node->left : node->right)->parent = node->parent;
      delete node;
    }
  }
  void leftRotate(Node *parent) {
    Node *child = parent->right;
    parent->right = child->left;
    if(child->left) child->left->parent = parent;
    child->parent = parent->parent;
    if(!parent->parent) root = child;
    else if(LCHILD(parent)) parent->parent->left = child;
    else parent->parent->right = child;
    child->left = parent;
    parent->parent = child;
  }
  void rightRotate(Node *parent) {
    Node *child = parent->left;
    parent->left = child->right;
    if(child->right) child->right->parent = parent;
    child->parent = parent->parent;
    if(!parent->parent) root = child;
    else if(!LCHILD(parent)) parent->parent->right = child;
    else parent->parent->left = child;
    child->right = parent;
    parent->parent = child;
  }
  void splay(Node *node) {
    while(root != node) {
      if(node->parent->parent) {
        if(LCHILD(node)) {
          if(LCHILD(node->parent))
            rightRotate(node->parent->parent), rightRotate(node->parent);
          else
            rightRotate(node->parent), leftRotate(node->parent);
        } else {
          if(LCHILD(node->parent))
            leftRotate(node->parent), rightRotate(node->parent);
          else
            leftRotate(node->parent->parent), leftRotate(node->parent);
        }
      } else if(LCHILD(node)) rightRotate(node->parent);
      else leftRotate(node->parent);
    }
  }
  void dealloc(Node *node) { if(node->left) dealloc(node->left); if(node->right) dealloc(node->right); delete node; }
public:
  SplayTree() : root(0) {}
  ~SplayTree() { if(root) dealloc(root); }
  void insert(K key) { if(root) insert(root, key); else root = new Node(key, 0); }
  void erase(K key) { if(root) erase(root, key); }
  bool contains(K key) { return root && find(root, key)->key == key; }
};
```

7 Divisors

Returns all divisors of N

```
void getDivisors(vii pf, int d, int index, vi &div) {
  if (index == pf.size()) {
    div.pb(d);
    return;
  }
  for (int i = 0; i <= pf[index].second; i++) {
    getDivisors(pf, d, index+1, div);
    d *= pf[index].first;
  }
  return;
}
```

8 ModNCR

```
int nCr(int n, int r, int MOD) {
  if (n-r < r)
    return nCr(n, n-r, MOD);
  int res = 1;
  FOR(i, 0, r) {
    res = res*(n-i)%MOD;
    res = res*mod_inverse(i+1, MOD)%MOD;
  }
  return res;
}
```

9 Base Conversion

```
string toBaseN(int num, int N) {
  string converted = num ? "" : "0";
  for(int div=abs(num); div; div /= N) {
    int value = div % N;
    converted = char(value > 9 ? value + ’A’ - 10 : value + ’0’) + converted;
  }
  return converted;
}
```

10 Roman Numerals

```
string fill(char c, int n) {
  string s;
  while(n--) s += c;
  return s;
}

string toRoman(int n) {
  if( n < 4 ) return fill( ’i’, n );
  if( n < 6 ) return fill( ’i’, 5 - n ) + "v";
  if( n < 9 ) return string( "v" ) + fill( ’i’, n - 5 );
  if( n < 11 ) return fill( ’i’, 10 - n ) + "x";
  if( n < 40 ) return fill( ’x’, n / 10 ) + toRoman( n % 10 );
  if( n < 60 ) return fill( ’x’, 5 - n / 10 ) + ’l’ + toRoman( n % 10 );
  if( n < 90 ) return string( "l" ) + fill( ’x’, n / 10 - 5 ) + toRoman( n % 10 );
  if( n < 110 ) return fill( ’x’, 10 - n / 10 ) + "c" + toRoman( n % 10 );
  if( n < 400 ) return fill( ’c’, n / 100 ) + toRoman( n % 100 );
  if( n < 600 ) return fill( ’c’, 5 - n / 100 ) + ’d’ + toRoman( n % 100 );
  if( n < 900 ) return string( "d" ) + fill( ’c’, n / 100 - 5 ) + toRoman( n % 100 );
  if( n < 1100 ) return fill( ’c’, 10 - n / 100 ) + "m" + toRoman( n % 100 );
  if( n < 4000 ) return fill( ’m’, n / 1000 ) + toRoman( n % 1000 );
  return "?";
}
```

11 Fenwick Tree

1 indexed not 0 indexed

```cpp
struct FenwickTree {
  vi ft;
  FenwickTree(int N) { ft.assign(N, 0); }
  int query(int to) { int sum = 0; while(to) sum += ft[to], to -= to&-to; return sum; }
  int query(int from, int to) { if(from > to) swap(to, from); return query(to) - query(from - 1); }
  void add(int i, int value) { while(i < int(ft.size())) ft[i] += value, i += i&-i;}
};

struct FenwickTree2D {
  vvi ft;
  FenwickTree2D(int R, int C) { ft.assign(R, vi(C, 0)); }
  int query(int r, int c) {
    int sum = 0;
    for(; r; r-=r&-r)
      for(int j=c; j; j-=j&-j)
        sum += ft[r][j];
    return sum;
  }
  int query(int r, int c, int R, int C) { if(R<r)swap(r,R); if(C<c)swap(c, C); return query(R, C) - query(r-1, C) - query(R, c-1) + query(r-1, c-1); }
  void add(int r, int c, int val) {
    for(; r<int(ft.size()); r+=r&-r)
      for(int j=c; j<int(ft.size()); j+=j&-j)
        ft[r][j] += val;
  }
};
```

12 Tree Hash

```cpp
const int INIT = 191, P1 = 701, P2 = 34943;

int hs(vector<vi> &children, int root) {
  int value = INIT;
  vi sub;
  FORC(children[root], it)
    sub.pb(hs(children, *it));
  sort(sub.begin(), sub.end());
  FORC(sub, it)
    value = ((value * P1) ^ *it) % P2;
  return value % P2;
}
```

13 Bellman Ford

```
vi bellmanFord(Graph &g, int source, bool &negativeCycle) {
  vi distanceTo(g.V, INF);
  distanceTo[source] = 0;
  FOR(i, 0, g.V-1)
    FOR(j, 0, g.V)
      FORC(g.edges[j], edge)
        distanceTo[edge->to] = min(distanceTo[edge->to], distanceTo[j] + edge->
            weight);
  //to detect negative weight cycles:
  FOR(i, 0, g.V)
    FORC(g.edges[i], edge)
      if(distanceTo[edge->to] > distanceTo[i] + edge->weight)
        negativeCycle = true;
  return distanceTo;
}
```

14 Bit Manipulation

```
#define turnOffLastBit(S) ((S) & (S - 1))
#define turnOnLastZero(S) ((S) | (S + 1))
#define turnOffLastConsecutiveBits(S) ((S) & (S + 1))
#define turnOnLastConsecutiveZeroes(S) ((S) | (S - 1))

int MSB(int x) {
  if(!x) return 0;
  int ans = 1;
  while(x>>1) x>>=1, ans<<=1;
  return ans;
}
```

15 Topological Sort

```
vi topologicalSort(Graph &g) {
  vi order, inDegree(g.V, 0);
  FOR(i, 0, g.V)
    FORC(g.edges[i], edge)
      inDegree[edge->to]++;
  FOR(i, 0, g.V)
    if(inDegree[i] == 0)
      order.pb(i);
  FOR(i, 0, order.size())
    FORC(g.edges[order[i]], edge)
      if(--inDegree[edge->to] == 0)
        order.pb(edge->to);
  return order;
}

void dfs(Graph &g, int currentVertex, vi &order, vi &visited) {
  visited[currentVertex] = true;
  FORC(g.edges[currentVertex], edge)
    if(!visited[edge->to])
      dfs(g, edge->to, order, visited);
  order.pb(currentVertex);
}

//Recursive version
vi topologicalSort2(Graph &g) {
  vi order, visited(g.V, 0);
  FOR(i, 0, g.V)
    if(!visited[i])
      dfs(g, i, order, visited);
  reverse(order.begin(), order.end());
  return order;
}
```

16 Shortest Path in a DAG

```
vi shortestPath(Graph &g) {
  vi order = topologicalSort(g);
  vi distanceTo(g.V, 0);
  FOR(i, 0, g.V) {
    int cv = order[i];
    FORC(g.edges[cv], edge) {
      if(distanceTo[edge->to] == 0)
        distanceTo[edge->to] = INF;
      distanceTo[edge->to] = min(distanceTo[edge->to], edge->weight +
          distanceTo[cv]);
    }
  }
  return distanceTo;
}
```

17 Edge Property Check

```
#define UNVISITED 0
#define EXPLORED 1 //visited but not completed
#define VISITED 2 //visited and completed
#define TREE 0 // Edge from explored to unvisited
#define BACK 1 // Edge that is part of a cycle (not including bidirectional
    edges). From explored to explored
#define FORWARD 2 // Edge from explored to visited
void dfs3(Graph &g, int cv, vi &parent, vi &state) {
  state[cv] = EXPLORED;
  FORC(g.edges[cv], edge)
    if(state[edge->to] == UNVISITED) {
      edge->type = TREE;
      parent[edge->to] = cv;
      dfs3(g, edge->to, parent, state);
    } else if(state[edge->to] == EXPLORED)
      edge->type = BACK; //if(edge->to == parent[cv]) //bidirectional
    else if(state[edge->to] == VISITED)
      edge->type = FORWARD;
  state[cv] = VISITED;
}

void edgeProperties(Graph &g) {
  vi state(g.V, UNVISITED), parent(g.V, 0);
  FOR(i, 0, g.V)
    if(state[i] == UNVISITED)
      dfs3(g, i, parent, state);
}
```

18 Cycle Finding

// x[i] = f(x[i-1])

```
int f(int i) { return (7*i+5)%12; }
ii floydCycleFinding(int x0) {
  int tortoise = f(x0), hare = f(f(x0));  //Encontrar el primer xi = x2i
  while (tortoise != hare) { tortoise = f(tortoise); hare = f(f(hare)); }
  int mu = 0; hare = x0;   //Encontrar mu usando el rango i
  while (tortoise != hare) { tortoise = f(tortoise); hare = f(hare); mu++; }
  int lambda = 1; hare = f(tortoise);   //Encontrar lambda teniendo mu
  while (tortoise != hare) { hare = f(hare); lambda++; }
  return ii(mu, lambda);
}
```

19 Fibbonacci

```
int fibn(int n) { //max 91
  double goldenRatio = (1+sqrt(5))/2;
  return round((pow(goldenRatio, n+1) - pow(1-goldenRatio, n+1))/sqrt(5));
}

int fibonacci(int n) {
  Matrix m = CREATE(2, 2);
  m[0][0] = 1, m[0][1] = 1, m[1][0] = 1, m[1][1] = 0;
  Matrix fib0 = CREATE(2, 1);
  fib0[0][0] = 1, fib0[1][0] = 1; //fib0 y fib1
  Matrix r = multiply(pow(m, n), fib0);
  return r[1][0];
}
```

20 Fast IO

```
const int BUFFSIZE = 10240;
char BUFF[BUFFSIZE + 1], *ppp = BUFF;
int RR, CHAR, SIGN, BYTES = 0;
#define GETCHAR(c) { \
  if(ppp-BUFF==BYTES && (BYTES==0 || BYTES==BUFFSIZE)) { BYTES = fread(BUFF,1,BUFFSIZE,stdin); ppp=BUFF; } \
  if(ppp-BUFF==BYTES && (BYTES>0 && BYTES<BUFFSIZE)) { BUFF[0] = 0; ppp=BUFF; } \
  c = *ppp++; \
}
#define DIGIT(c) (((c) >= '0') && ((c) <= '9'))
#define MINUS(c) ((c)== '-')
#define GETNUMBER(n) { \
  n = 0; SIGN = 1; do { GETCHAR(CHAR); } while(!(DIGIT(CHAR) || MINUS(CHAR))); \
  if(MINUS(CHAR)) { SIGN = -1; GETCHAR(CHAR); } \
  while(DIGIT(CHAR)) { n = ((n<<3) + (n << 1)) + CHAR-'0'; GETCHAR(CHAR); } if(SIGN == -1) { n = -n; } \
}
```

21 Binomial Coefficients

max n=61

```
int nCr(int n, int r) {
  int res = 1;
  FOR(i, 0, r) res = res*(n-i)/(i+1);
  return res;
}

#define MAXN 68
long long pascal[MAXN][MAXN];
void buildPascal() {
  FOR(n, 0, MAXN)
    FOR(r, 0, n+1)
      pascal[n][r] = (r == 0 || r == n) ? 1 : pascal[n-1][r-1] + pascal[n-1][r];
}
```

22 FactMOD

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

23 Range OR

```
int rangeOR(int A, int B) {
  int value = 0;
  for(int i=1<<(sizeof(int)-1); i; i >>= 1) {
    value <<= 1;
    value += A/i&1 || B/i&1 || A/i != B/i;
  }
  return value;
}
```

24 Longest Increasing Subsequence

```
vi longestIncreasingSubsequence(vi v) {
  vii best;
  vi parent(v.size(), -1);
  FOR(i, 0, v.size()) {
    ii item = ii(v[i], i);
    vii::iterator it = upper_bound(best.begin(), best.end(), item);
    if (it == best.end()) {
      parent[i] = (best.size() == 0 ? -1 : best.back().second);
      best.pb(item);
    } else {
      parent[i] = parent[it->second];
      *it = item;
    }
  }
  vi lis;
  for(int i=best.back().second; i >= 0; i=parent[i])
    lis.pb(v[i]);
  reverse(lis.begin(), lis.end());
  return lis;
}
```

25 Longest Common Subsequence

```
string LCS(string a, string b) {
  int n = a.length(), m = b.length();
  int D[n][m];
  char c[n][m];
  FOR(i, 0, n)
    FOR(j, 0, m)
      if(a[i] == b[j]) {
        D[i][j] = i&&j ? D[i-1][j-1] + 1 : 1;
        c[i][j] = a[i];
      }
      else {
        c[i][j] = (i ? D[i-1][j] : 0) >= (j ? D[i][j-1] : 0);
        D[i][j] = max(i ? D[i-1][j] : 0, j ? D[i][j-1] : 0);
      }
  string lcs;
  while(n-- && m--) {
    if(c[n][m] == 0) n++;
    else if(c[n][m] == 1) m++;
    else lcs = c[n][m] + lcs;
  }
  return lcs;
}
```

26 Maximum Subarray

```
int maximumSubarray(int numbers[], int N) {
  int maxSoFar = numbers[0], maxEndingHere = numbers[0];
  FOR(i, 1, N) {
    if(maxEndingHere < 0) maxEndingHere = numbers[i];
    else maxEndingHere += numbers[i];
    maxSoFar = max(maxEndingHere, maxSoFar);
  }
  return maxSoFar;
}
int maxCircularSum (int a[], int n) {
   int max_kadane = maximumSubarray(a, n);
   int max_wrap  =  0;
   FOR(i, 0, n) {
        max_wrap += a[i];
        a[i] = -a[i];
   }
   max_wrap = max_wrap + maximumSubarray(a, n);
   return (max_wrap > max_kadane)? max_wrap : max_kadane;
}
```

27 Subsequence Counter

Regresa cuantas veces subseq es subsequence de seq

```
int subseqCounter(string seq, string subseq) {
  int n = seq.length(), m = subseq.length();
  vi sub(m, 0);
  FOR(i, 0, n)
```

```
    for(int j = m-1; j >= 0; j--)
      if(seq[i] == subseq[j]) {
        if(j == 0) sub[0]++;
        else sub[j] += sub[j-1];
            }
  return sub[m-1];
}
```

28 Edit Distance

```
int editDistance(string A, string B) {
  int n = A.length(), m = B.length();
  int dist[n+1][m+1];
  dist[0][0] = 0;
  FOR(i, 1, n+1) dist[i][0] = i;
  FOR(j, 1, m+1) dist[0][j] = j;
  FOR(i, 1, n+1)
    FOR(j, 1, m+1)
      dist[i][j] = min(dist[i-1][j-1] + (A[i-1] != B[j-1]), min(dist[i-1][j] +
          1, dist[i][j-1] + 1));
  return dist[n][m];
}
```

29 Quicksort

```
void quickSort(int arr[], int left, int right) {
  int pivot = arr[(left+right)/2];
  int i = left, j = right;
  while(i <= j) {
    while(arr[i] < pivot) i++;
    while(arr[j] > pivot) j--;
    if(i<=j) swap(arr[i++], arr[j--]);
  }
  if(left < j) quickSort(arr, left, j);
  if(i < right) quickSort(arr, i, right);
}
```

30 Mergesort

```
int merge(int array[], int low, int mid, int high) {
  int inversions = 0;
  int sorted[high-low+1];
  int p1 = low, p2 = mid+1, psorted = 0;
  while(p1 <= mid && p2 <= high) {
    if(array[p1] <= array[p2])
      sorted[psorted++] = array[p1++];
    else {
      sorted[psorted++] = array[p2++];
      inversions += mid-p1+1;
    }
  }
  while(p1 <= mid) sorted[psorted++] = array[p1++];
  while(p2 <= high) sorted[psorted++] = array[p2++];
  FOR(i, low, high+1) array[i] = sorted[i-low];
  return inversions;
}

//returns the number of inversions
int mergeSort(int array[], int low, int high) {
  if(low < high) {
    int mid = (low + high)/2;
    int inversions = mergeSort(array, low, mid) + mergeSort(array, mid+1, high);
    return inversions + merge(array, low, mid, high);
  }
  return 0;
}
```

31 Fast Exponentiation

```
double fastPow(double a, int n) {
  if(n == 0) return 1;
  if(n == 1) return a;
  double t = fastPow(a, n>>1);
  return t*t*fastPow(a, n&1);
}
```

32 Binary Search

```
const int UPPERBOUND = 0, LOWERBOUND = 1, ANY = 2;
int binarySearch(int array[], int searchValue, int left, int right, int type = ANY) {
  int leftBound = left, rightBound = right;
  while(left <= right) {
        int mid = (left+right)/1;
    if(searchValue > array[mid]) left = mid+1;
    else if (searchValue < array[mid]) right = mid-1;
    else {
            if(type == UPPERBOUND) {
                if(mid == rightBound || array[mid+1] != array[mid])
                    return mid;
                left = mid+1;
            } else if(type == LOWERBOUND) {
                if(mid == leftBound || array[mid-1] != array[mid])
```

```
                    return mid;
                right = mid-1;
            } else {
                return mid;
            }
    }
  }
  return -1;
}
```

33 LCM

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

34 Binary Heap

```
template <typename T>
struct Heap {
  vector<T> tree;
  int last;
  Heap(int size) : last(1) { tree.assign(size+1, 0); }
  void push(T n) {
    tree[last++] = n;
    for(int i=last-1; i != 1 && tree[i>>1] < tree[i]; i>>=1)
      swap(tree[i], tree[i>>1]);
  }
  void pop() {
    swap(tree[--last], tree[1]);
    for(int i=1; ((i<<1) < last && tree[i] < tree[i<<1]) || ((i<<1)+1 < last && tree[i] < tree[(i<<1)+1]);) {
      int k = ((i<<1) + ((i<<1)+1 < last && tree[(i<<1)+1] > tree[i<<1]));
      swap(tree[i], tree[k]);
      i=k;
    }
  }
  int top() { return tree[1]; }
  bool empty() { return last == 1; }
  bool size() { return last - 1; }
};
```

35 Simplex

Two-phase simplex algorithm for solving linear programs of the form maximize $c^T x$ subject to $Ax <= b$ $x >= 0$ INPUT: A – an m x n matrix b – an m-dimensional vector c – an n-dimensional vector x – a vector where the optimal solution will be stored OUTPUT:

value of the optimal solution (infinity if unbounded above, nan if infeasible) To use this code, create an LPSolver object with A, b, and c as arguments. Then, call Solve(x).

```
#include <limits>
typedef long double DOUBLE;
typedef vector<DOUBLE> VD;
typedef vector<VD> VVD;
typedef vector<int> VI;
const DOUBLE EPS = 1e-9;
struct LPSolver {
  int m, n;
  VI B, N;
  VVD D;
  LPSolver(const VVD &A, const VD &b, const VD &c) :
      m(b.size()), n(c.size()), N(n+1), B(m), D(m+2, VD(n+2)) {
      for (int i = 0; i < m; i++) for (int j = 0; j < n; j++) D[i][j] = A[i][j];
      for (int i = 0; i < m; i++) { B[i] = n+i; D[i][n] = -1; D[i][n+1] = b[i]; }
      for (int j = 0; j < n; j++) { N[j] = j; D[m][j] = -c[j]; }
      N[n] = -1; D[m+1][n] = 1;}
  void Pivot(int r, int s) {
      for (int i = 0; i < m+2; i++) if (i != r)
          for (int j = 0; j < n+2; j++) if (j != s)
      D[i][j] -= D[r][j] * D[i][s] / D[r][s];
      for (int j = 0; j < n+2; j++) if (j != s) D[r][j] /= D[r][s];
      for (int i = 0; i < m+2; i++) if (i != r) D[i][s] /= -D[r][s];
      D[r][s] = 1.0 / D[r][s];
      swap(B[r], N[s]);}
  bool Simplex(int phase) {
      int x = phase == 1 ? m+1 : m;
      while (true) {
          int s = -1;
          for (int j = 0; j <= n; j++) {
      if (phase == 2 && N[j] == -1) continue;
      if (s == -1 || D[x][j] < D[x][s] || D[x][j] == D[x][s] && N[j] < N[s]) s = j;
          }
          if (D[x][s] >= -EPS) return true;
          int r = -1;
          for (int i = 0; i < m; i++) {
      if (D[i][s] <= 0) continue;
      if (r == -1 || D[i][n+1] / D[i][s] < D[r][n+1] / D[r][s] ||
              D[i][n+1] / D[i][s] == D[r][n+1] / D[r][s] && B[i] < B[r]) r = i;
          }
          if (r == -1) return false;
          Pivot(r, s);}}
  DOUBLE Solve(VD &x) {
      int r = 0;
      for (int i = 1; i < m; i++) if (D[i][n+1] < D[r][n+1]) r = i;
      if (D[r][n+1] <= -EPS) {
          Pivot(r, n);
          if (!Simplex(1) || D[m+1][n+1] < -EPS) return -numeric_limits<DOUBLE>::infinity();
          for (int i = 0; i < m; i++) if (B[i] == -1) {
      int s = -1;
      for (int j = 0; j <= n; j++)
          if (s == -1 || D[i][j] < D[i][s] || D[i][j] == D[i][s] && N[j] < N[s]) s = j;
      Pivot(i, s);}}
      if (!Simplex(2)) return numeric_limits<DOUBLE>::infinity();
      x = VD(n);
      for (int i = 0; i < m; i++) if (B[i] < n) x[B[i]] = D[i][n+1];
      return D[m][n+1];}};
int main() {
    const int m = 4;
    const int n = 3;
    DOUBLE _A[m][n] = {
        { 6, -1, 0 },
        { -1, -5, 0 },
        { 1, 5, 1 },
        { -1, -5, -1 }
    };
    DOUBLE _b[m] = { 10, -4, 5, -5 };
    DOUBLE _c[n] = { 1, -1, 0 };
    VVD A(m);
    VD b(_b, _b + m);
    VD c(_c, _c + n);
    for (int i = 0; i < m; i++) A[i] = VD(_A[i], _A[i] + n);
    LPSolver solver(A, b, c);
    VD x;
    DOUBLE value = solver.Solve(x);
    cerr << "VALUE:␣"<< value << endl;
    cerr << "SOLUTION:";
    for (size_t i = 0; i < x.size(); i++) cerr << "␣" << x[i];
    cerr << endl;
    return 0;
}
```

36 Graph Cut Inference

Special-purpose 0,1 combinatorial optimization solver for problems of the following by a reduction to graph cuts: minimize $sum_ipsi_i(x[i])x[1]...x[n]in0,1 + sum_{i<j}phi_{ij}(x[i],x[j]))$ where $psi_i : 0,1 -- > R$ $phi_{ij} : 0,1x0,1 -- > R$ such that $phi_{ij}(0,0) + phi_{ij}(1,1) <= phi_{ij}(0,1) + phi_{ij}(1,0)(*)$ This can also be used to solve maximization problems where the direction of the inequality in (*) is reversed. INPUT: phi – a matrix such that $phi[i][j][u][v] = phi_{ij}(u,v)$ psi – a matrix such that $psi[i][u] = psi_i(u)$ x – a vector where the optimal solution will be stored OUTPUT: value of the optimal solution To use this code, create a GraphCutInference object, and call the DoInference() method. To perform maximization instead of minimization, ensure that #define MAXIMIZATION is enabled.

```
typedef vector<int> VI;
typedef vector<VI> VVI;
typedef vector<VVI> VVVI;
typedef vector<VVVI> VVVVI;
const int INF = 1000000000;
// comment out following line for minimization
#define MAXIMIZATION
struct GraphCutInference {
    int N;
    VVI cap, flow;
    VI reached;
    int Augment(int s, int t, int a) {
        reached[s] = 1;
        if (s == t) return a;
        for (int k = 0; k < N; k++) {
            if (reached[k]) continue;
            if (int aa = min(a, cap[s][k] - flow[s][k])) {
        if (int b = Augment(k, t, aa)) {
            flow[s][k] += b;
            flow[k][s] -= b;
            return b;}}}
        return 0;}
    int GetMaxFlow(int s, int t) {
        N = cap.size();
        flow = VVI(N, VI(N));
        reached = VI(N);
        int totflow = 0;
        while (int amt = Augment(s, t, INF)) {
            totflow += amt;
            fill(reached.begin(), reached.end(), 0);}
        return totflow;}
    int DoInference(const VVVVI &phi, const VVI &psi, VI &x) {
        int M = phi.size();
        cap = VVI(M+2, VI(M+2));
        VI b(M);
        int c = 0;
        for (int i = 0; i < M; i++) {
            b[i] += psi[i][1] - psi[i][0];
            c += psi[i][0];
            for (int j = 0; j < i; j++)
        b[i] += phi[i][j][1][1] - phi[i][j][0][1];
            for (int j = i+1; j < M; j++) {
        cap[i][j] = phi[i][j][0][1] + phi[i][j][1][0] - phi[i][j][0][0] - phi[i][j][1][1];
        b[i] += phi[i][j][1][0] - phi[i][j][0][0];
        c += phi[i][j][0][0];}}
#ifdef MAXIMIZATION
        for (int i = 0; i < M; i++) {
            for (int j = i+1; j < M; j++)
        cap[i][j] *= -1;
            b[i] *= -1;}
        c *= -1;
#endif
        for (int i = 0; i < M; i++) {
            if (b[i] >= 0) {
        cap[M][i] = b[i];
            } else {
        cap[i][M+1] = -b[i];
        c += b[i];}}
        int score = GetMaxFlow(M, M+1);
        fill(reached.begin(), reached.end(), 0);
        Augment(M, M+1, INF);
        x = VI(M);
        for (int i = 0; i < M; i++) x[i] = reached[i] ? 0 : 1;
        score += c;
#ifdef MAXIMIZATION
        score *= -1;
#endif
        return score;}};
int main() {
    // solver for "Cat vs. Dog" from NWERC 2008
    int numcases;
    cin >> numcases;
    for (int caseno = 0; caseno < numcases; caseno++) {
        int c, d, v;
        cin >> c >> d >> v;
        VVVVI phi(c+d, VVVI(c+d, VVI(2, VI(2))));
        VVI psi(c+d, VI(2));
        for (int i = 0; i < v; i++) {
            char p, q;
            int u, v;
            cin >> p >> u >> q >> v;
            u--; v--;
            if (p == ’C’) {
        phi[u][c+v][0][0]++;
        phi[c+v][u][0][0]++;
            } else {
        phi[v][c+u][1][1]++;
        phi[c+u][v][1][1]++;
            }
        }
        GraphCutInference graph;
        VI x;
        cout << graph.DoInference(phi, psi, x) << endl;}
    return 0;
}
```

37 Notes

```
printf("%ld\n", strtol("222", 0, x)); //base x to long
~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
regmatch_t matches[1];
regcomp(&reg, pattern.c_str(), REG_EXTENDED|REG_ICASE);
if(regexec(&reg, str.c_str(), 1, matches, 0) == 0)
cout << "match" << endl;
regfree(&reg);
~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
template <typename T>
string toString(T n) { ostringstream ss; ss << n; return ss.str(); }

template <typename T>
T toNum(const string &Text) { istringstream ss(Text); T result; return ss >>
    result ? result : 0; }
~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
vector<int> v(3, 5); //init vector to {5, 5, 5}
int arr[] = {2, 3, 4};
vector<int> v(arr, arr+3); //init vector to array
~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
int* it = lower_bound(arr, arr+N, searchValue)
if(it == arr+N) cout << "not␣found" << endl;
else cout << "found␣" << *it << "␣at␣index␣" << it-arr << endl;
lower_bound: finds first that does not compare less than val.
upper_bound: finds first that compares greater than val.
~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
int arr[] = {1, 2, 3}
reverse(arr, arr+N); //reverses the array, arr = {3, 2, 1}
sort(arr+N, arr) //reverse sort
partial_sort(arr, arr+k, arr+N) //partially sorts the array time: klog(N)
~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
struct Point
{
  double x, y;
  string id;
};

Point origin = {0, 0, "origin"};
Point points[3] = {{3.4, 2.1, "myPoint1"},
    {2.4, 7.2, "myPoint2"},
    {4.1, 8.1, "myPoint3"}};
~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
#include <algorithm>
int arr[] = {0, 1, 2, 3, 4};
next_permutation(arr, arr+5); //0, 1, 2, 4, 3
next_permutation(arr, arr+5); //0, 1, 3, 2, 4
prev_permutation(...)
~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
#include <map>
```

```
#include <set>
//check if it contains an item
myMap.count(item);
mySet.count(item);
~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
//When sorting small structs, for example:
struct Team
{
  int goldMedals;
  int silverMedals;
  int bronzeMedals;
};
//sort by gold, then silver then bronze
//instead of defining a comparison function, another way is to:

typedef pair<int, pair<int, int> > Team;

Team teams[10];
teams[0] = make_pair(4, make_pair(2, 6));
...

sort(teams, teams + 10);
//drawback: all variables will be sorted ascending or descending
~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
#include <iomanip>
cout << fixed << setprecision(3) << 23.2341 << endl; //23.234 //formats forever until changed
cout.setwidth(8); //only for the next cout
cout << 2355 << endl; //"2355" -> "    2355"
cout.fill("-"); //forever until changed again
cout << 2355 << endl; //"2355" -> "------2355"
cout.setwidth(10);
cout << left << 2355 << endl; //"2355" -> "2355------"
~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
scanf:

%d -> base10 int | %d+
%o -> base8 int  | %d+
%x -> base16 int | %d+
%a -> base10 or base16 double | ex. 123, 34.24, 5464.324e+3, 53423E+2, 0x242.435, base16 if preceded by 0x
%c -> char or array of chars | ex. scanf("%c", &mychar) -> 'a', scanf("%4c", mycharptr) -> "asdf" (\0 not included)
%s -> string
matching: scanf("abc%d", &myint) with input: "ab34␣ascz24␣abc345" would store 345 in myint, use %% to match %
%*d means match an int but dont store it in a parameter
%3d means match an integer but read only the 3 first characters
%lld stores in a long long int %d matches int, more specifiers are:
%le long double
```

```
~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
class comparator
{
  bool operator()(int a, int b)
  {
    return a < b;
  }
};
priority_queue<int, vector<int>, comparator> pq;
~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
Bellman-Ford for solving system of inequalities of the type x_i - x_j <= c
create a node for every x
create a source node
create a zero weight edge from s to every other node
for every inequality x_i - x_j <= c, make an edge from i to j of weight c
run bellman ford starting at s
the value for x_i is d_i
if there was a negative weight cycle, the system is inconsistent
~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
max_element(arr, arr+N); //returns a pointer to the element
min_element(arr, arr+N);

//arrays must be sorted, can be used with set and map
merge() //same as set union but allows duplicates
set_union()//A + B
set_intersection()//A intersection B
set_difference()// A - B
set_symmetric_difference() // A^B
parameters: (begin1, end1, begin2, end2, begin_result); //returns a pointer to the end of the result, and the result is stored [begin_result to end_result)

accumulate(arr, arr+N, (long long)0); //add all elements by default, function can be specified, 0 is the initial value
double product = accumulate(all(v), double(1), multiplies<double>());
//plus, minus, divides, modulus, negate, equal_to, custom functions implemented same way as priority queue comparator
inner_product(all(v1), v2.begin(), 0); //scalar product [a, b, c].[d, e, f] = a*d + b*e + c*f
for_each(vec.begin(), vec.end(), func); //calls func(i) for every element in [begin, end)
nth_element (vec.begin(), vec.begin()+n, vec.end(), myfunction);
~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
#include <ctype.h>
isalpha(char c), isupper(char c) ,islower(char c), isdigit(char c), ispunct(char c), toupper(char c), tolower(char c)
~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
In a bipartite graph, the size of the maximum independent set (or dominating set) = V-MCBM
In a bipartite graph, the size of the min vertex cover = MCBM
```

```
~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
char str[] = "abc.␣sdfksm␣sgfda␣afdex..␣NJK-␣,,␣␣␣.␣␣hb564567....";
char * token = strtok(str, ".␣");
while (token != NULL)
{
    printf ("%s\n",token);
    token = strtok (NULL, ".␣");
}
~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
There are n^(n-2) spanning trees in a complete graph with n vertices
A dearangement is a permutation of a set where all elements are in a different position than their original position
der(n) = (n-1)*(der(n-1)+der(n-2)), der(0) = 1, der(1) = 0
a finite sequence of natural numbers can be a degree sequence of a graph iff the sum is even and sum from i=1 to k of d_i < k*(k-1) for 1<=k<=n
E - V - 2 = F where F is the number of faces in a planar graph
The number of pieces in which a circle is divided if n points on its circumference are joined by chords with no three internally concurrent:
g(n) = nCat4 + nCat2 + 1
A = i+b/2-1 where A is the area of a polygon, i is the number of integer points on the polygon and b is the number of integer points on the boundary
the number of spanning trees in complete a bipartite graph K(n, m) is m^(n-1) * n^(m-1)
~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
//splitting by spaces
istringstream iss(line);
vector<string> tokens;
copy(istream_iterator<string>(iss), istream_iterator<string>(), back_inserter<vector<string> >(tokens));
~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
Grundy numbers
At each turn, each player chooses a game and makes a move.
You lose if there is no possible move.
For each game, we compute its Grundy number
The first player wins iff the XOR of all grundy numbers is nonzero
Computing the grundy numbers:
Let S be a state, and T1, T2, ... Tm be states reachable from S using a single move.
The Grundy number of a losing state is 0
The Grundy number g(S) of S is the smallest nonnegative integer that does not appear in { g(T1 ), g(T2 ), ..., g(Tm ) }
~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
Given a 4x4 grid of unique numbers between 1 and 15 and one empty cell.
The position of a number can be exchanged with the position of the empty cell if it is adjacent.
Is it possible to arrange the numbers in order starting at the top left, increasing first to the right then down, and having the last cell empty?
Solution:
Let N be the number of inversions in the permutation.
Let K be the line number (starting at zero) of the empty cell.
```

```
A solution exists iff N+K is even
~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
Number of Paths of Length K for every Pair of Nodes
Problem:
Given an undirected, unweighed Graph G, find the number of paths of length k between every pair of vertices.
Solution:
D_k = G^k
~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
Build the set of all fractions
Problem:
Build the set of all non-negative fractions.
Solution:
Start with the fractions:
(0/1, 1/0)
For every pair of adjacent fractions, create a new fraction between them where the numerator is the sum of their numerators and the denominator is the sum of their denominators. Repeat infinitely.
~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
Problem:
Finding maximum area rectangle in histogram.
Solution:
L(i): number of adjacent bars to the left with height >= H(i)
R(i): number of adjacent bars to the right with height >= H(i)
A(i) = H(i) * (L(i)+R(i)+1)
O(n) solution:
Keep a stack with the indexes whose heights are smaller than the
height being currently considered. Those that are contiguous before
and greater can be assumed as already considered.
Code for L(i) (repeat for R):
for (int i=0; i<n; i++) {
  while (!st.empty() && h[st.top()] <= h[i])  st.pop();
  L[i] = i - (st.empty() ? -1 : st.top()) - 1;
  st.push (i);}
*Maximum zero submatrix:
For each cell, keep track of the last 1 on the same column.
For each row, apply above algorithm.
~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
Inverse SSSP
Problem:
Given an undirected weighted graph of m edges , n vertices, and a vector P of weights and a source vertex S, find new values for the weights of all edges such that the P[i] is now the length of the shortest path from S to vertex i.
Solution:
Linear, keep a vector cost_ch of changes to each edge, a vector of nodes decrease_id (stores the neighbors that must be decreased for each node), of and a vector of decreases decrease (the smallest decrease that must be made to any neighbor for each vertex).
const int INF = 1000*1000*1000;
```

```
int n, m;
vector<int> p (n);
bool ok = true;
vector<int> cost (m), cost_ch (m), decrease (n, INF), decrease_id (n, -1);
decrease[0] = 0;
for (int i=0; i<m; ++i) {
  int a, b, c;
  cost[i] = c;
  for (int j=0; j<=1; ++j) {
    int diff = p[b] - p[a] - c;
    if (diff > 0) {
      ok &= cost_ch[i] == 0 || cost_ch[i] == diff;
      cost_ch[i] = diff;
      decrease[b] = 0;
    } else if (-diff <= c && -diff < decrease[b]) {
      decrease[b] = -diff;
      decrease_id[b] = i; }
    swap (a, b);}}
for (int i=0; i<n; ++i) {
  ok &= decrease[i] != INF;
  int r_id = decrease_id[i];
  if (r_id != -1) {
    ok &= cost_ch[r_id] == 0 || cost_ch[r_id] == -decrease[i];
    cost_ch[r_id] = -decrease[i];}}
//cost_ch now holds the changes to each edge (increase or decrease) with minimum sum of absolute values, if ok is true
~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
K camino mas corto
const int INF = 1000*1000*1000; const int W = ...; // peso maximo
int n, s, t;
vector < vector < pair<int,int> > > g; vector<int> dist;
vector<char> used;
vector<int> curpath, kth path;
int kth_path_exists(int k, int maxlen, int v, int curlen = 0) { curpath.push back(v);
  if(v == t) {
    if(curlen == maxlen) kth path = curpath;
    curpath.pop back();
    return 1; }
  used[v] = true;
  int found = 0;
  for(size t i=0; i<g[v].size(); ++i) {
    int to = g[v][i].first, len = g[v][i].second;
    if(!used[to] && curlen + len + dist[to] <= maxlen) {
      found += kth_path_exists(k - found, maxlen, to, curlen + len);
      if(found == k) break; }}
  used[v] = false; curpath.pop back(); return found;}
int main() {
//... inicializar (n, k, g, s, t) ...
dist.assign(n, INF); dist[t] = 0; used.assign(n, false); for(;;) {
  int sel = -1;
  for(int i=0; i<n; ++i)
    if(!used[i] && dist[i] < INF && (sel == -1 || dist[i] < dist[sel])) sel = i;
  if(sel == -1) break;
  used[sel] = true;
  for(size t i=0; i<g[sel].size(); ++i) {
    int to = g[sel][i].first,  len = g[sel][i].second;
    dist[to] = min (dist[to], dist[sel] + len); }}
int minw = 0, maxw = W; while(minw < maxw) {
  int wlimit = (minw + maxw) >> 1; used.assign(n, false);
  if(kth_path_exists(k, wlimit, s) == k)
    maxw = wlimit;
  else
    minw = wlimit + 1; }
used.assign(n, false);
if(kth_path_exists(k, minw, s) < k)
  puts("NO␣SOLUTION");
else {
  cout << minw << ? ? << kth path.size() << endl; for(size t i=0; i<kth path.size(); ++i)
  cout << kth path[i]+1 << ? ?; }}
~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
Primes less than 1000:
2     3     5     7    11    13    17    19    23    29    31    37
41    43    47    53    59    61    67    71    73    79    83    89
97   101   103   107   109   113   127   131   137   139   149   151
157   163   167   173   179   181   191   193   197   199   211   223
227   229   233   239   241   251   257   263   269   271   277   281
283   293   307   311   313   317   331   337   347   349   353   359
367   373   379   383   389   397   401   409   419   421   431   433
439   443   449   457   461   463   467   479   487   491   499   503
509   521   523   541   547   557   563   569   571   577   587   593
599   601   607   613   617   619   631   641   643   647   653   659
661   673   677   683   691   701   709   719   727   733   739   743
751   757   761   769   773   787   797   809   811   821   823   827
829   839   853   857   859   863   877   881   883   887   907   911
919   929   937   941   947   953   967   971   977   983   991   997
```