```
/** Modelo C/C++, macro de DEBUG e EOF ........................................ */
/** Truque de leitura com scanf() ............................................. */

/** Vector .................................................................... */
/** Algoritmos da STL ......................................................... */
/** Manipulacao de bits ....................................................... */
/** Stack/Queue da STL ........................................................ */
/** Map/Set da STL ............................................................ */
/** Priority Queue da STL ..................................................... */
/** Grafos (implementacao) .................................................... */
/** Union-Find Disjoint Sets .................................................. */
/** Arvore de Segmentos ....................................................... */
/** Fenwick Tree (Binary Indexed Tree, ou BIT) ................................ */

/** Backtracking (exemplo) .................................................... */
/** Programacao Dinamica (ex. 1: Top-Down) .................................... */
/** Programacao Dinamica (ex. 2: Bottom-Up) ................................... */
/** Max 1D Range Sum .......................................................... */
/** Maximum Sum ............................................................... */
/** Longest Increasing Subsequence (LIS) ...................................... */
/** Algoritmo da Mochila 0-1 .................................................. */
/** Coin Change (Problema do Troco) ........................................... */
/** Problema do Caixeiro Viajante ............................................. */
/** Qtd. de formas de obter um numero N somando K numeros ..................... */
/** Cutting Sticks ............................................................ */

/** DFS (Busca em Profundidade) ............................................... */
/** Flood Fill / grafo implicito em matriz .................................... */
/** Kruskal e Prim (Arvore Geradora Minima) ................................... */
/** BFS (Busca em Largura/Amplitude) .......................................... */
/** Dijkstra .................................................................. */
/** Bellman-Ford .............................................................. */
/** Floyd-Warshall ............................................................ */
/** Edmonds-Karp .............................................................. */
/** Emparelhamento Maximo em Grafos Bipartidos ................................ */

/** BigInt (implementacao em C/C++) ........................................... */
/** Crivo de Eratostenes (descobre n's primos) ................................ */
/** Floyd's Cycle-Finding Algorithm ........................................... */

/** Strings (algoritmos basicos) .............................................. */
/** Knuth-Morris-Pratt (string matching) ...................................... */
/** Alinhamento de Strings (Needleman-Wunsch) ................................. */
/** Array de Sufixos .......................................................... */

/** Pontos e Linhas ........................................................... */
/** Circulos .................................................................. */
/** Triangulos ................................................................ */
/** Poligonos ................................................................. */

/** 15-Puzzle Problem with IDA* ............................................... */
/** Prog. Dinamica com Bitmask ................................................ */
/** Prog. Dinamica (outro exemplo) ............................................ */
/** Outras tecnicas ........................................................... */

/** Eliminacao Gaussiana ...................................................... */
/** Lowest Common Ancestor (LCA) .............................................. */
/** Pollard Rho (fatoracao) ................................................... */
/** Range Minimum Query (RMQ) ................................................. */
/** Fibonacci Modular ......................................................... */
/** Shortest Path Faster Algorithm ............................................ */
/** Algarismos Romanos ........................................................ */
/** Distancia entre pontos em esfera + dist. euclidiana ....................... */
/** Componentes Fortemente Conectadas ......................................... */

```

```
/******************************************************************************/
/** Modelo C/C++, macro de DEBUG e EOF ........................................ */
/******************************************************************************/

#include<stdio.h>  // se preferir scanf() e printf()
#include<iostream> // se preferir cin e cout
#define MAX 1123   // alocacao estatica; se usar, ALTERE ISSO CONFORME O PROBLEMA
//#define DEBP     // DEBug Prints: descomente para debugar o codigo
using namespace std;

/* exemplo DEBP */
void funcao_com_problema(int N){
  for(int i = 0; i < N; i++){
    if(i = 0){ faca_alguma_coisa(); } // um erro bem idiota :V
    // o trecho abaixo so sera compilado e executado se DEBP estiver definida
    #ifdef DEBP
      printf("%d\n", i); // imprima variaveis pra tentar achar a causa do problema
    #endif
  }
}
/* fim do exemplo */

int main(void){
  /* EOF em C++ */
  string str; // funciona tambem com int, double e outros tipos
  while(cin >> str){ // para simular "fim-de-arquivo" no terminal, digite Ctrl-D
    // sua logica aqui
  }

  /* EOF em C puro */
  char s[MAX];
  // int N;
  // while(scanf("%d", &N) != EOF) {
  while(fgets(s, MAX, stdin) != NULL){
    // sua logica aqui
  }

  return 0; // SEMPRE retorne 0 para o sistema
  // caso contrario, vc pode tomar RUNTIME ERROR
}

```

```cpp
/*******************************************************************************/
/** Truque de leitura com scanf() ............................................ */
/*******************************************************************************/

#include <cstdio>
using namespace std;

int N;          // using global variables in contests can be a good strategy
char x[110];  // make it a habit to set array size a bit larger than needed

int main() {
  scanf("%d\n", &N);
  while (N--) {                    // we simply loop from N, N-1, N-2, ..., 0
    scanf("0.%[0-9]...\n", &x);    // '&' is optional when x is a char array
                         // note: if you are surprised with the trick above,
                      // please check scanf details in www.cppreference.com
    printf("the digits are 0.%s\n", x);
    return 0;
  }
}
```

```cpp
/**************************************************************************/
/** Vector .............................................................. */
/**************************************************************************/

#include <cstdio>
#include <vector>
using namespace std;

int main() {
  int arr[5] = {7,7,7};    // initial size (5) and initial value {7,7,7,0,0}
  vector<int> v(5, 5);     // initial size (5) and initial value {5,5,5,5,5}

  printf("arr[2] = %d and v[2] = %d\n", arr[2], v[2]);           // 7 and 5

  for (int i = 0; i < 5; i++) {
    arr[i] = i;
    v[i] = i;
  }

  printf("arr[2] = %d and v[2] = %d\n", arr[2], v[2]);           // 2 and 2

  // arr[5] = 5;      // static array will generate index out of bound error
  // uncomment the line above to see the error

  v.push_back(5);                               // but vector will resize itself
  printf("v[5] = %d\n", v[5]);                                          // 5

  return 0;
}
```

```cpp
/*****************************************************************************/
/** Algoritmos da STL ..................................................... */
/*****************************************************************************/

#include <algorithm>
#include <cstdio>
#include <string>
#include <vector>
using namespace std;

typedef struct {
  int id;
  int solved;
  int penalty;
} team;

bool icpc_cmp(team a, team b) {
  if (a.solved != b.solved) // can use this primary field to decide sorted order
    return a.solved > b.solved;   // ICPC rule: sort by number of problem solved
  else if (a.penalty != b.penalty)       // a.solved == b.solved, but we can use
                                       // secondary field to decide sorted order
    return a.penalty < b.penalty;       // ICPC rule: sort by descending penalty
  else                           // a.solved == b.solved AND a.penalty == b.penalty
    return a.id < b.id;                       // sort based on increasing team ID
}

int main() {
  int *pos, arr[] = {10, 7, 2, 15, 4};
  vector<int> v(arr, arr + 5);          // another way to initialize vector
  vector<int>::iterator j;

  // sort descending with vector
  sort(v.rbegin(), v.rend());         // example of using 'reverse iterator'
  for (vector<int>::iterator it = v.begin(); it != v.end(); it++)
    printf("%d ", *it);                           // access the value of iterator
  printf("\n");
  printf("==================\n");

  // sort descending with integer array
  sort(arr, arr + 5);                                           // ascending
  reverse(arr, arr + 5);                                      // then reverse
  for (int i = 0; i < 5; i++)
    printf("%d ", arr[i]);
  printf("\n");
  printf("==================\n");

  random_shuffle(v.begin(), v.end());          // shuffle the content again
  for (vector<int>::iterator it = v.begin(); it != v.end(); it++)
    printf("%d ", *it);
  printf("\n");
  printf("==================\n");
  partial_sort(v.begin(), v.begin() + 2, v.end());      // partial_sort demo
  for (vector<int>::iterator it = v.begin(); it != v.end(); it++)
    printf("%d ", *it);
  printf("\n");
  printf("==================\n");

  // sort ascending
  sort(arr, arr + 5);                           // arr should be sorted now
  for (int i = 0; i < 5; i++)                             // 2, 4, 7, 10, 15
    printf("%d ", arr[i]);
  printf("\n");
  sort(v.begin(), v.end());                   // sorting a vector, same output
  for (vector<int>::iterator it = v.begin(); it != v.end(); it++)
    printf("%d ", *it);
  printf("\n");
  printf("==================\n");

  // multi-field sorting example, suppose we have 4 ICPC teams
  team nus[4] = { {1, 1, 10},
```

```
                  {2, 3, 60},
                  {3, 1, 20},
                  {4, 3, 60} };

  // without sorting, they will be ranked like this:
  for (int i = 0; i < 4; i++)
    printf("id: %d, solved: %d, penalty: %d\n",
           nus[i].id, nus[i].solved, nus[i].penalty);

  sort(nus, nus + 4, icpc_cmp);         // sort using a comparison function
  printf("==================\n");
  // after sorting using ICPC rule, they will be ranked like this:
  for (int i = 0; i < 4; i++)
    printf("id: %d, solved: %d, penalty: %d\n",
           nus[i].id, nus[i].solved, nus[i].penalty);
  printf("==================\n");

  // there is a trick for multi-field sorting if the sort order is "standard"
  // use "chained" pair class in C++ and put the highest priority in front
  typedef pair < int, pair < string, string > > state;
  state a = make_pair(10, make_pair("steven", "grace"));
  state b = make_pair(7, make_pair("steven", "halim"));
  state c = make_pair(7, make_pair("steven", "felix"));
  state d = make_pair(9, make_pair("a", "b"));
  vector<state> test;
  test.push_back(a);
  test.push_back(b);
  test.push_back(c);
  test.push_back(d);
  for (int i = 0; i < 4; i++)
    printf("value: %d, name1 = %s, name2 = %s\n", test[i].first,
      ((string)test[i].second.first).c_str(),
      ((string)test[i].second.second).c_str());
  printf("==================\n");
  sort(test.begin(), test.end());   // no need to use a comparison function
  // sorted ascending based on value, then based on name1,
  // then based on name2, in that order!
  for (int i = 0; i < 4; i++)
    printf("value: %d, name1 = %s, name2 = %s\n", test[i].first,
      ((string)test[i].second.first).c_str(),
      ((string)test[i].second.second).c_str());
  printf("==================\n");

  // binary search using lower bound
  pos = lower_bound(arr, arr + 5, 7);                              // found
  printf("%d\n", *pos);
  j = lower_bound(v.begin(), v.end(), 7);
  printf("%d\n", *j);

  pos = lower_bound(arr, arr + 5, 77);                         // not found
  if (pos - arr == 5) // arr is of size 5 ->
                      //    arr[0], arr[1], arr[2], arr[3], arr[4]
                      // if lower_bound cannot find the required value,
                      //   it will set return arr index +1 of arr size, i.e.
                      //   the 'non existent' arr[5]
                      // thus, testing whether pos - arr == 5 blocks
                      //   can detect this "not found" issue
    printf("77 not found\n");
  j = lower_bound(v.begin(), v.end(), 77);
  if (j == v.end()) // with vector, lower_bound will do the same:
                    //   return vector index +1 of vector size
                    // but this is exactly the position of vector.end()
                    //   so we can test "not found" this way
    printf("77 not found\n");
  printf("==================\n");

  // useful if you want to generate permutations of set
  next_permutation(arr, arr + 5); // 2, 4, 7, 10, 15 -> 2, 4, 7, 15, 10
  next_permutation(arr, arr + 5); // 2, 4, 7, 15, 10 -> 2, 4, 10, 7, 15
  for (int i = 0; i < 5; i++)
```

```
    printf("%d ", arr[i]);
  printf("\n");

  next_permutation(v.begin(), v.end());
  next_permutation(v.begin(), v.end());
  for (vector<int>::iterator it = v.begin(); it != v.end(); it++)
    printf("%d ", *it);
  printf("\n");
  printf("==================\n");

  // sometimes these two useful simple macros are used
  printf("min(10, 7) = %d\n", min(10, 7));
  printf("max(10, 7) = %d\n", max(10, 7));

  return 0;
}
```

```cpp
/*******************************************************************************/
/** Manipulacao de bits ....................................................... */
/*******************************************************************************/

// note: for example usage of bitset, see ch5_06_primes.cpp

#include <cmath>
#include <cstdio>
#include <stack>
using namespace std;

#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, (int)((log((double)S) / log(2)) + 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))

void printSet(int vS) {                         // in binary representation
  printf("S = %2d = ", vS);
  stack<int> st;
  while (vS)
    st.push(vS % 2), vS /= 2;
  while (!st.empty())                         // to reverse the print order
    printf("%d", st.top()), st.pop();
  printf("\n");
}

int main() {
  int S, T;

  printf("1. Representation (all indexing are 0-based and counted from right)\n");
  S = 34; printSet(S);
  printf("\n");

  printf("2. Multiply S by 2, then divide S by 4 (2x2), then by 2\n");
  S = 34; printSet(S);
  S = S << 1; printSet(S);
  S = S >> 2; printSet(S);
  S = S >> 1; printSet(S);
  printf("\n");

  printf("3. Set/turn on the 3-th item of the set\n");
  S = 34; printSet(S);
  setBit(S, 3); printSet(S);
  printf("\n");

  printf("4. Check if the 3-th and then 2-nd item of the set is on?\n");
  S = 42; printSet(S);
  T = isOn(S, 3); printf("T = %d, %s\n", T, T ? "ON" : "OFF");
  T = isOn(S, 2); printf("T = %d, %s\n", T, T ? "ON" : "OFF");
  printf("\n");

  printf("5. Clear/turn off the 1-st item of the set\n");
  S = 42; printSet(S);
  clearBit(S, 1); printSet(S);
  printf("\n");

  printf("6. Toggle the 2-nd item and then 3-rd item of the set\n");
  S = 40; printSet(S);
  toggleBit(S, 2); printSet(S);
  toggleBit(S, 3); printSet(S);
```

```
  printf("\n");

  printf("7. Check the first bit from right that is on\n");
  S = 40; printSet(S);
  T = lowBit(S); printf("T = %d (this is always a power of 2)\n", T);
  S = 52; printSet(S);
  T = lowBit(S); printf("T = %d (this is always a power of 2)\n", T);
  printf("\n");

  printf("8. Turn on all bits in a set of size n = 6\n");
  setAll(S, 6); printSet(S);
  printf("\n");

  printf("9. Other tricks (not shown in the book)\n");
  printf("8 %c 4 = %d\n", '%', modulo(8, 4));
  printf("7 %c 4 = %d\n", '%', modulo(7, 4));
  printf("6 %c 4 = %d\n", '%', modulo(6, 4));
  printf("5 %c 4 = %d\n", '%', modulo(5, 4));
  printf("is %d power of two? %d\n", 9, isPowerOfTwo(9));
  printf("is %d power of two? %d\n", 8, isPowerOfTwo(8));
  printf("is %d power of two? %d\n", 7, isPowerOfTwo(7));
  for (int i = 0; i <= 16; i++)
    printf("Nearest power of two of %d is %d\n", i, nearestPowerOfTwo(i));
  printf("S = %d, turn off last bit in S, S = %d\n", 40, turnOffLastBit(40));
  printf("S = %d, turn on last zero in S, S = %d\n", 41, turnOnLastZero(41));
  printf("S = %d, turn off last consectuve bits in S, S = %d\n", 39,
    turnOffLastConsecutiveBits(39));
  printf("S = %d, turn on last consecutive zeroes in S, S = %d\n", 36,
    turnOnLastConsecutiveZeroes(36));

  return 0;
}
```

```cpp
/**********************************************************************************/
/** Stack/Queue da STL .......................................................... */
/**********************************************************************************/

#include <cstdio>
#include <stack>
#include <queue>
using namespace std;

int main() {
  stack<char> s;
  queue<char> q;
  deque<char> d;

  printf("%d\n", s.empty());                     // currently s is empty, true (1)
  printf("==================\n");
  s.push('a');
  s.push('b');
  s.push('c');
  // stack is LIFO, thus the content of s is currently like this:
  // c <- top
  // b
  // a
  printf("%c\n", s.top());                                      // output 'c'
  s.pop();                                                     // pop topmost
  printf("%c\n", s.top());                                      // output 'b'
  printf("%d\n", s.empty());           // currently s is not empty, false (0)
  printf("==================\n");

  printf("%d\n", q.empty());                // currently q is empty, true (1)
  printf("==================\n");
  while (!s.empty()) {                      // stack s still has 2 more items
    q.push(s.top());                             // enqueue 'b', and then 'a'
    s.pop();
  }
  q.push('z');                                            // add one more item
  printf("%c\n", q.front());                                     // prints 'b'
  printf("%c\n", q.back());                                      // prints 'z'

  // output 'b', 'a', then 'z' (until queue is empty),
  // according to the insertion order above
  printf("==================\n");
  while (!q.empty()) {
    printf("%c\n", q.front());                       // take the front first
    q.pop();                                 // before popping (dequeue-ing) it
  }

  printf("==================\n");
  d.push_back('a');
  d.push_back('b');
  d.push_back('c');
  printf("%c - %c\n", d.front(), d.back());                  // prints 'a - c'
  d.push_front('d');
  printf("%c - %c\n", d.front(), d.back());                  // prints 'd - c'
  d.pop_back();
  printf("%c - %c\n", d.front(), d.back());                  // prints 'd - b'
  d.pop_front();
  printf("%c - %c\n", d.front(), d.back());                  // prints 'a - b'

  return 0;
}
```

```cpp
/****************************************************************************/
/** Map/Set da STL ....................................................... */
/****************************************************************************/

#include <cstdio>
#include <map>
#include <set>
#include <string>
using namespace std;

int main() {
  char name[20];
  int value;
  // note: there are many clever usages of this set/map
  // that you can learn by looking at top coder's codes
  // note, we don't have to use .clear() if we have just initialized the set/map
  set<int> used_values; // used_values.clear();
  map<string, int> mapper; // mapper.clear();

  // suppose we enter these 7 name-score pairs below
  /*
  john 78
  billy 69
  andy 80
  steven 77
  felix 82
  grace 75
  martin 81
  */
  mapper["john"] = 78;   used_values.insert(78);
  mapper["billy"] = 69;  used_values.insert(69);
  mapper["andy"] = 80;   used_values.insert(80);
  mapper["steven"] = 77; used_values.insert(77);
  mapper["felix"] = 82;  used_values.insert(82);
  mapper["grace"] = 75;  used_values.insert(75);
  mapper["martin"] = 81; used_values.insert(81);

  // then the internal content of mapper MAY be something like this:
  // re-read balanced BST concept if you do not understand this diagram
  // the keys are names (string)!
  //                          (grace,75)
  //            (billy,69)                (martin,81)
  //      (andy,80)   (felix,82)    (john,78)   (steven,77)

  // iterating through the content of mapper will give a sorted output
  // based on keys (names)
  for (map<string, int>::iterator it = mapper.begin(); it != mapper.end(); it++)
    printf("%s %d\n", ((string)it->first).c_str(), it->second);

  // map can also be used like this
  printf("steven's score is %d, grace's score is %d\n",
    mapper["steven"], mapper["grace"]);
  printf("==================\n");

  // interesting usage of lower_bound and upper_bound
  // display data between ["f".."m") ('felix' is included, martin' is excluded)
  for (map<string, int>::iterator it = mapper.lower_bound("f"); it != mapper.upper_bound("m"); it++)
    printf("%s %d\n", ((string)it->first).c_str(), it->second);

  // the internal content of used_values MAY be something like this
  // the keys are values (integers)!
  //                 (78)
  //         (75)            (81)
  //     (69)    (77)    (80)    (82)

  // O(log n) search, found
  printf("%d\n", *used_values.find(77));
  // returns [69, 75]
  // (these two are before 77 in the inorder traversal of this BST)
```

```
  for (set<int>::iterator it = used_values.begin(); it != used_values.lower_bound(77); it++)
    printf("%d,", *it);
  printf("\n");
  // returns [77, 78, 80, 81, 82]
  // (these five are equal or after 77 in the inorder traversal of this BST)
  for (set<int>::iterator it = used_values.lower_bound(77); it != used_values.end(); it++)
    printf("%d,", *it);
  printf("\n");
  // O(log n) search, not found
  if (used_values.find(79) == used_values.end())
    printf("79 not found\n");

  return 0;
}
```

```cpp
/*************************************************************************************/
/** Priority Queue da STL ....................................................... */
/*************************************************************************************/

#include <cstdio>
#include <iostream>
#include <string>
#include <queue>
using namespace std;

int main() {
  int money;
  char name[20];
  priority_queue< pair<int, string> > pq;             // introducing 'pair'
  pair<int, string> result;

  // suppose we enter these 7 money-name pairs below
  /*
  100 john
  10 billy
  20 andy
  100 steven
  70 felix
  2000 grace
  70 martin
  */
  pq.push(make_pair(100, "john"));          // inserting a pair in O(log n)
  pq.push(make_pair(10, "billy"));
  pq.push(make_pair(20, "andy"));
  pq.push(make_pair(100, "steven"));
  pq.push(make_pair(70, "felix"));
  pq.push(make_pair(2000, "grace"));
  pq.push(make_pair(70, "martin"));
  // priority queue will arrange items in 'heap' based
  // on the first key in pair, which is money (integer), largest first
  // if first keys tie, use second key, which is name, largest first

  // the internal content of pq heap MAY be something like this:
  // re-read (max) heap concept if you do not understand this diagram
  // the primary keys are money (integer), secondary keys are names (string)!
  //                        (2000,grace)
  //           (100,steven)               (70,martin)
  //     (100,john)   (10,billy)     (20,andy)  (70,felix)

  // let's print out the top 3 person with most money
  result = pq.top();                // O(1) to access the top / max element
  pq.pop();          // O(log n) to delete the top and repair the structure
  printf("%s has %d $\n", ((string)result.second).c_str(), result.first);
  result = pq.top(); pq.pop();
  printf("%s has %d $\n", ((string)result.second).c_str(), result.first);
  result = pq.top(); pq.pop();
  printf("%s has %d $\n", ((string)result.second).c_str(), result.first);

  return 0;
}
```

```cpp
/*********************************************************************************/
/** Grafos (implementacao) .................................................... */
/*********************************************************************************/

#include <cstdio>
#include <iostream>
#include <vector>
#include <queue>
using namespace std;

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

int main() {
  int V, E, total_neighbors, id, weight, a, b;
  int AdjMat[100][100];
  vector<vii> AdjList;
  priority_queue< pair<int, ii> > EdgeList;   // one way to store Edge List

  // Try this input for Adjacency Matrix/List/EdgeList
  // Adj Matrix
  //   for each line: |V| entries, 0 or the weight
  // Adj List
  //   for each line: num neighbors, list of neighbors + weight pairs
  // Edge List
  //   for each line: a-b of edge(a,b) and weight
  /*
  6
    0  10   0   0 100   0
   10   0   7   0   8   0
    0   7   0   9   0   0
    0   0   9   0  20   5
  100   8   0  20   0   0
    0   0   0   5   0   0
  6
  2 2 10 5 100
  3 1 10 3 7 5 8
  2 2 7 4 9
  3 3 9 5 20 6 5
  3 1 100 2 8 4 20
  1 4 5
  7
  1 2 10
  1 5 100
  2 3 7
  2 5 8
  3 4 9
  4 5 20
  4 6 5
  */
  freopen("in_07.txt", "r", stdin);

  scanf("%d", &V);                              // we must know this size first!
                             // remember that if V is > 100, try NOT to use AdjMat!
  for (int i = 0; i < V; i++)
    for (int j = 0; j < V; j++)
      scanf("%d", &AdjMat[i][j]);

  printf("Neighbors of vertex 0:\n");
  for (int j = 0; j < V; j++)                                           // O(|V|)
    if (AdjMat[0][j])
      printf("Edge 0-%d (weight = %d)\n", j, AdjMat[0][j]);

  scanf("%d", &V);
  AdjList.assign(V, vii());
  // quick way to initialize AdjList with V entries of vii
  for (int i = 0; i < V; i++) {
    scanf("%d", &total_neighbors);
    for (int j = 0; j < total_neighbors; j++) {
      scanf("%d %d", &id, &weight);
```

```
      AdjList[i].push_back(ii(id - 1, weight));    // some index adjustment
    }
  }

  printf("Neighbors of vertex 0:\n");
  for (vii::iterator j = AdjList[0].begin(); j != AdjList[0].end(); j++)
    // AdjList[0] contains the required information
    // O(k), where k is the number of neighbors
    printf("Edge 0-%d (weight = %d)\n", j->first, j->second);

  scanf("%d", &E);
  for (int i = 0; i < E; i++) {
    scanf("%d %d %d", &a, &b, &weight);
    EdgeList.push(make_pair(-weight, ii(a, b))); // trick to reverse sort order
  }

  // edges sorted by weight (smallest->largest)
  for (int i = 0; i < E; i++) {
    pair<int, ii> edge = EdgeList.top(); EdgeList.pop();
    // negate the weight again
    printf("weight: %d (%d-%d)\n", -edge.first, edge.second.first,
      edge.second.second);
  }

  return 0;
}
```

```cpp
/*****************************************************************************/
/** Union-Find Disjoint Sets ................................................ */
/*****************************************************************************/

#include <cstdio>
#include <vector>
using namespace std;

typedef vector<int> vi;

// Union-Find Disjoint Sets Library written in OOP manner,
// using both path compression and union by rank heuristics
class UnionFind {                                              // OOP style
private:
  vi p, rank, setSize;                       // remember: vi is vector<int>
  int numSets;
public:
  UnionFind(int N) {
    setSize.assign(N, 1); numSets = N; 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) { return findSet(i) == findSet(j); }
  void unionSet(int i, int j) {
    if (!isSameSet(i, j)) { numSets--;
    int x = findSet(i), y = findSet(j);
    // rank is used to keep the tree short
    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 numDisjointSets() { return numSets; }
  int sizeOfSet(int i) { return setSize[findSet(i)]; }
};

int main() {
  printf("Assume that there are 5 disjoint sets initially\n");
  UnionFind UF(5); // create 5 disjoint sets
  printf("%d\n", UF.numDisjointSets()); // 5
  UF.unionSet(0, 1);
  printf("%d\n", UF.numDisjointSets()); // 4
  UF.unionSet(2, 3);
  printf("%d\n", UF.numDisjointSets()); // 3
  UF.unionSet(4, 3);
  printf("%d\n", UF.numDisjointSets()); // 2
  printf("isSameSet(0, 3) = %d\n", UF.isSameSet(0, 3)); // will return 0 (false)
  printf("isSameSet(4, 3) = %d\n", UF.isSameSet(4, 3)); // will return 1 (true)
  int i;
  for (i = 0; i < 5; i++) // findSet will return 1 for {0, 1} and 3 for {2, 3, 4}
    printf("findSet(%d) = %d, sizeOfSet(%d) = %d\n", i, UF.findSet(i), i,
      UF.sizeOfSet(i));
  UF.unionSet(0, 3);
  printf("%d\n", UF.numDisjointSets()); // 1
  for (i = 0; i < 5; i++) // findSet will return 3 for {0, 1, 2, 3, 4}
    printf("findSet(%d) = %d, sizeOfSet(%d) = %d\n", i, UF.findSet(i), i,
      UF.sizeOfSet(i));
  return 0;
}
```

```
/********************************************************************************/
/** Arvore de Segmentos ........................................................ */
/********************************************************************************/

#include <cmath>
#include <cstdio>
#include <vector>
using namespace std;

typedef vector<int> vi;

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); }
};

int main() {
  int arr[] = { 18, 17, 13, 19, 15, 11, 20 };          // the original array
  vi A(arr, arr + 7);                          // copy the contents to a vector
  SegmentTree st(A);

  printf("              idx    0, 1, 2, 3, 4,  5, 6\n");
  printf("              A is {18,17,13,19,15, 11,20}\n");
  printf("RMQ(1, 3) = %d\n", st.rmq(1, 3));             // answer = index 2
  printf("RMQ(4, 6) = %d\n", st.rmq(4, 6));             // answer = index 5
  printf("RMQ(3, 4) = %d\n", st.rmq(3, 4));             // answer = index 4
  printf("RMQ(0, 0) = %d\n", st.rmq(0, 0));             // answer = index 0
  printf("RMQ(0, 1) = %d\n", st.rmq(0, 1));             // answer = index 1
  printf("RMQ(0, 6) = %d\n", st.rmq(0, 6));             // answer = index 5

  printf("              idx    0, 1, 2, 3, 4,  5, 6\n");
  printf("Now, modify A into {18,17,13,19,15,100,20}\n");
  st.update_point(5, 100);                     // update A[5] from 11 to 100
  printf("These values do not change\n");
  printf("RMQ(1, 3) = %d\n", st.rmq(1, 3));                            // 2
  printf("RMQ(3, 4) = %d\n", st.rmq(3, 4));                            // 4
  printf("RMQ(0, 0) = %d\n", st.rmq(0, 0));                            // 0
  printf("RMQ(0, 1) = %d\n", st.rmq(0, 1));                            // 1
  printf("These values change\n");
  printf("RMQ(0, 6) = %d\n", st.rmq(0, 6));                         // 5->2
  printf("RMQ(4, 6) = %d\n", st.rmq(4, 6));                         // 5->4
  printf("RMQ(4, 5) = %d\n", st.rmq(4, 5));                         // 5->4

  return 0;
}
```

```
/*************************************************************************************/
/** Fenwick Tree (Binary Indexed Tree, ou BIT) ................................ */
/*************************************************************************************/

#include <cstdio>
#include <vector>
using namespace std;

typedef vector<int> vi;
#define LSOne(S) (S & (-S))

class FenwickTree {
private:
  vi ft;

public:
  FenwickTree() {}
  // initialization: n + 1 zeroes, ignore index 0
  FenwickTree(int n) { ft.assign(n + 1, 0); }

  int rsq(int b) {                                     // returns RSQ(1, b)
    int sum = 0; for (; b; b -= LSOne(b)) sum += ft[b];
    return sum; }

  int rsq(int a, int b) {                              // returns RSQ(a, b)
    return rsq(b) - (a == 1 ? 0 : rsq(a - 1)); }

  // adjusts value of the k-th element by v (v can be +ve/inc or -ve/dec)
  void adjust(int k, int v) {                      // note: n = ft.size() - 1
    for (; k < (int)ft.size(); k += LSOne(k)) ft[k] += v; }
};

int main() {                 // idx   0 1 2 3 4 5 6 7  8 9 10, no index 0!
  FenwickTree ft(10);        // ft = {-,0,0,0,0,0,0,0, 0,0,0}
  ft.adjust(2, 1);           // ft = {-,0,1,0,1,0,0,0, 1,0,0}, idx 2,4,8 => +1
  ft.adjust(4, 1);           // ft = {-,0,1,0,2,0,0,0, 2,0,0}, idx 4,8 => +1
  ft.adjust(5, 2);           // ft = {-,0,1,0,2,2,2,0, 4,0,0}, idx 5,6,8 => +2
  ft.adjust(6, 3);           // ft = {-,0,1,0,2,2,5,0, 7,0,0}, idx 6,8 => +3
  ft.adjust(7, 2);           // ft = {-,0,1,0,2,2,5,2, 9,0,0}, idx 7,8 => +2
  ft.adjust(8, 1);           // ft = {-,0,1,0,2,2,5,2,10,0,0}, idx 8 => +1
  ft.adjust(9, 1);           // ft = {-,0,1,0,2,2,5,2,10,1,1}, idx 9,10 => +1
  printf("%d\n", ft.rsq(1, 1));  // 0 => ft[1] = 0
  printf("%d\n", ft.rsq(1, 2));  // 1 => ft[2] = 1
  printf("%d\n", ft.rsq(1, 6));  // 7 => ft[6] + ft[4] = 5 + 2 = 7
  printf("%d\n", ft.rsq(1, 10)); // 11 => ft[10] + ft[8] = 1 + 10 = 11
  printf("%d\n", ft.rsq(3, 6));  // 6 => rsq(1, 6) - rsq(1, 2) = 7 - 1

  ft.adjust(5, 2); // update demo
  printf("%d\n", ft.rsq(1, 10)); // now 13
} // return 0;
```

```cpp
/********************************************************************************/
/** Backtracking (exemplo) ..................................................... */
/********************************************************************************/

/* 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;
```

```cpp
/*******************************************************************************/
/** Programacao Dinamica (ex. 1: Top-Down) ..................................... */
/*******************************************************************************/

/* UVa 11450 - Wedding Shopping - Top Down */
// this code is similar to recursive backtracking code
// parts of the code specific to top-down DP are commented with: 'TOP-DOWN'
// if these lines are commented, this top-down DP will become backtracking!
#include <algorithm>
#include <cstdio>
#include <cstring>
using namespace std;

int M, C, price[25][25];                           // price[g (<= 20)][model (<= 20)]
int memo[210][25];      // TOP-DOWN: dp table memo[money (<= 200)][g (<= 20)]
int shop(int money, int g) {
  if (money < 0) return -1000000000;       // fail, return a large -ve number
  if (g == C) return M - money;          // we have bought last garment, done
  if (memo[money][g] != -1) return memo[money][g]; // TOP-DOWN: memoization
  int ans = -1;    // start with a -ve number as all prices are non negative
  for (int model = 1; model <= price[g][0]; model++)       // try all models
    ans = max(ans, shop(money - price[g][model], g + 1));
  return memo[money][g] = ans; // TOP-DOWN: assign ans to table + return it
}

int main() {               // easy to code if you are already familiar with it
  int i, j, TC, score;

  scanf("%d", &TC);
  while (TC--) {
    scanf("%d %d", &M, &C);
    for (i = 0; i < C; i++) {
      scanf("%d", &price[i][0]);                    // store K in price[i][0]
      for (j = 1; j <= price[i][0]; j++) scanf("%d", &price[i][j]);
    }
    memset(memo, -1, sizeof memo);     // TOP-DOWN: initialize DP memo table
    score = shop(M, 0);                               // start the top-down DP
    if (score < 0) printf("no solution\n");
    else           printf("%d\n", score);
} } // return 0;
```

```cpp
/**********************************************************************************/
/** Programacao Dinamica (ex. 2: Bottom-Up) ..................................... */
/**********************************************************************************/

/* UVa 11450 - Wedding Shopping - Bottom Up */
#include <cstdio>
#include <cstring>
using namespace std;

int main() {
  int i, j, k, TC, M, C;
  int price[25][25];                      // price[g (<= 20)][model (<= 20)]
  bool reachable[25][210];     // reachable table[g (<= 20)][money (<= 200)]
  scanf("%d", &TC);
  while (TC--) {
    scanf("%d %d", &M, &C);
    for (i = 0; i < C; i++) {
      scanf("%d", &price[i][0]);                // we store K in price[i][0]
      for (j = 1; j <= price[i][0]; j++) scanf("%d", &price[i][j]);
    }

    memset(reachable, false, sizeof reachable);         // clear everything
    for (i = 1; i <= price[0][0]; i++)         // initial values (base cases)
      if (M - price[0][i] >= 0)        // to prevent array index out of bound
        reachable[0][M - price[0][i]] = true;  // using first garment g = 0

    for (i = 1; i < C; i++)                     // for each remaining garment
      for (j = 0; j < M; j++) if (reachable[i - 1][j]) // a reachable state
        for (k = 1; k <= price[i][0]; k++) if (j - price[i][k] >= 0)
          reachable[i][j - price[i][k]] = true;   // also a reachable state

    for (j = 0; j <= M && !reachable[C - 1][j]; j++); // the answer in here

    if (j == M + 1) printf("no solution\n");          // last row has on bit
    else            printf("%d\n", M - j);
} } // return 0;
```

```cpp
/*******************************************************************************/
/** Max 1D Range Sum ......................................................... */
/*******************************************************************************/

#include <algorithm>
#include <cstdio>
using namespace std;

int main() {
  int n = 9, A[] = { 4, -5, 4, -3, 4, 4, -4, 4, -5 };   // a sample array A
  int running_sum = 0, ans = 0;
  for (int i = 0; i < n; i++)                                          // O(n)
    if (running_sum + A[i] >= 0) {  // the overall running sum is still +ve
      running_sum += A[i];
      ans = max(ans, running_sum);          // keep the largest RSQ overall
    }
    else        // the overall running sum is -ve, we greedily restart here
      running_sum = 0;      // because starting from 0 is better for future
                           // iterations than starting from -ve running sum
  printf("Max 1D Range Sum = %d\n", ans);                      // should be 9
} // return 0;
```

```cpp
/*******************************************************************************/
/** Maximum Sum ................................................................ */
/*******************************************************************************/

#include <algorithm>
#include <cstdio>
using namespace std;

int n, A[101][101], maxSubRect, subRect;

int main() {      // O(n^3) 1D DP + greedy (Kadane's) solution, 0.008 s in UVa
  scanf("%d", &n);                          // the dimension of input square matrix
  for (int i = 0; i < n; i++) for (int j = 0; j < n; j++) {
    scanf("%d", &A[i][j]);
    if (j > 0) A[i][j] += A[i][j - 1];        // only add columns of this row i
  }

  maxSubRect = -127*100*100;     // the lowest possible value for this problem
  for (int l = 0; l < n; l++) for (int r = l; r < n; r++) {
    subRect = 0;
    for (int row = 0; row < n; row++) {
      // Max 1D Range Sum on columns of this row i
      if (l > 0) subRect += A[row][r] - A[row][l - 1];
      else       subRect += A[row][r];

      // Kadane's algorithm on rows
      if (subRect < 0) subRect = 0;      // greedy, restart if running sum < 0
      maxSubRect = max(maxSubRect, subRect);
  } }

  printf("%d\n", maxSubRect);
  return 0;
}

/*******************************************************************************/
/** Maximum Sum (alternate version) ********************************************/
/*******************************************************************************/

#include <algorithm>
#include <cstdio>
using namespace std;

int n, A[101][101], maxSubRect, subRect;

int main() {                               // O(n^4) DP solution, ~0.076s in UVa
  scanf("%d", &n);                          // the dimension of input square matrix
  for (int i = 0; i < n; i++) for (int j = 0; j < n; j++) {
    scanf("%d", &A[i][j]);
    if (i > 0) A[i][j] += A[i - 1][j];              // if possible, add from top
    if (j > 0) A[i][j] += A[i][j - 1];             // if possible, add from left
    if (i > 0 && j > 0) A[i][j] -= A[i - 1][j - 1];      // avoid double count
  }                                            // inclusion-exclusion principle

  maxSubRect = -127*100*100;     // the lowest possible value for this problem
  for (int i = 0; i < n; i++) for (int j = 0; j < n; j++) // start coordinate
    for (int k = i; k < n; k++) for (int l = j; l < n; l++) {     // end coord
      subRect = A[k][l];      // sum of all items from (0, 0) to (k, l): O(1)
      if (i > 0) subRect -= A[i - 1][l];                                 // O(1)
      if (j > 0) subRect -= A[k][j - 1];                                 // O(1)
      if (i > 0 && j > 0) subRect += A[i - 1][j - 1];                    // O(1)
      maxSubRect = max(maxSubRect, subRect); }            // the answer is here

  printf("%d\n", maxSubRect);
  return 0;
}
```

```cpp
/*****************************************************************************/
/** Longest Increasing Subsequence (LIS) .................................. */
/*****************************************************************************/

#include <algorithm>
#include <cstdio>
#include <stack>
using namespace std;

#define MAX_N 100000

void print_array(const char *s, int a[], int n) {
  for (int i = 0; i < n; ++i) {
    if (i) printf(", ");
    else printf("%s: [", s);
    printf("%d", a[i]);
  }
  printf("]\n");
}

void reconstruct_print(int end, int a[], int p[]) {
  int x = end;
  stack<int> s;
  for (; p[x] >= 0; x = p[x]) s.push(a[x]);
  printf("[%d", a[x]);
  for (; !s.empty(); s.pop()) printf(", %d", s.top());
  printf("]\n");
}

int main() {
  int n = 11, A[] = {-7, 10, 9, 2, 3, 8, 8, 1, 2, 3, 4};
  int L[MAX_N], L_id[MAX_N], P[MAX_N];

  int lis = 0, lis_end = 0;
  for (int i = 0; i < n; ++i) {
    int pos = lower_bound(L, L + lis, A[i]) - L;
    L[pos] = A[i];
    L_id[pos] = i;
    P[i] = pos ? L_id[pos - 1] : -1;
    if (pos + 1 > lis) {
      lis = pos + 1;
      lis_end = i;
    }

    printf("Considering element A[%d] = %d\n", i, A[i]);
    printf("LIS ending at A[%d] is of length %d: ", i, pos + 1);
    reconstruct_print(i, A, P);
    print_array("L is now", L, lis);
    printf("\n");
  }

  printf("Final LIS is of length %d: ", lis);
  reconstruct_print(lis_end, A, P);
  return 0;
}
```

```
/*****************************************************************************/
/** Algoritmo da Mochila 0-1 .................................................. */
/*****************************************************************************/

// 0-1 Knapsack DP (Top-Down) - faster as not all states are visited

#include <algorithm>
#include <cstdio>
#include <cstring>
using namespace std;

#define MAX_N 1010
#define MAX_W 40

int i, T, G, ans, N, MW, V[MAX_N], W[MAX_N], memo[MAX_N][MAX_W];

int value(int id, int w) {
  if (id == N || w == 0) return 0;
  if (memo[id][w] != -1) return memo[id][w];
  if (W[id] > w)          return memo[id][w] = value(id + 1, w);
  return memo[id][w] = max(value(id + 1, w), V[id] + value(id + 1, w - W[id]));
}

int main() {
  scanf("%d", &T);
  while (T--) {
    memset(memo, -1, sizeof memo);

    scanf("%d", &N);
    for (i = 0; i < N; i++)
      scanf("%d %d", &V[i], &W[i]);

    ans = 0;
    scanf("%d", &G);
    while (G--) {
      scanf("%d", &MW);
      ans += value(0, MW);
    }

    printf("%d\n", ans);
  }

  return 0;
}

/*****************************************************************************/
/** Algoritmo da Mochila 0-1 (Bottom-Up) *****************************************/
/*****************************************************************************/

#include <algorithm>
#include <cstdio>
using namespace std;

#define MAX_N 1010
#define MAX_W 40

int i, w, T, N, G, MW, V[MAX_N], W[MAX_N], C[MAX_N][MAX_W], ans;

int main() {
  scanf("%d", &T);
  while (T--) {
    scanf("%d", &N);
    for (i = 1; i<= N; i++)
      scanf("%d %d", &V[i], &W[i]);

    ans = 0;
    scanf("%d", &G);
    while (G--) {
      scanf("%d", &MW);

```

```
      for (i = 0; i <= N;  i++) C[i][0] = 0;
      for (w = 0; w <= MW; w++) C[0][w] = 0;

      for (i = 1; i <= N; i++)
        for (w = 1; w <= MW; w++) {
          if (W[i] > w) C[i][w] = C[i - 1][w];
          else          C[i][w] = max(C[i - 1][w], V[i] + C[i - 1][w - W[i]]);
        }

      ans += C[N][MW];
    }

    printf("%d\n", ans);
  }

  return 0;
}
```

```cpp
/********************************************************************************/
/** Coin Change (Problema do Troco) ........................................... */
/********************************************************************************/

// O(NV) DP solution

#include <cstdio>
#include <cstring>
using namespace std;

int N = 5, V, coinValue[5] = {1, 5, 10, 25, 50}, memo[6][7500];
// N and coinValue are fixed for this problem, max V is 7489

int ways(int type, int value) {
  if (value == 0)              return 1;
  if (value < 0 || type == N)  return 0;
  if (memo[type][value] != -1) return memo[type][value];
  return memo[type][value] = ways(type + 1, value) +
    ways(type, value - coinValue[type]);
}

int main() {
  memset(memo, -1, sizeof memo); // we only need to initialize this once
  while (scanf("%d", &V) != EOF)
    printf("%d\n", ways(0, V));

  return 0;
}
```

```cpp
/*****************************************************************************/
/** Problema do Caixeiro Viajante ........................................ */
/*****************************************************************************/

// Collecting Beepers
// DP TSP

#include <algorithm>
#include <cmath>
#include <cstdio>
#include <cstring>
using namespace std;
                                                     // Karel + max 10 beepers
int i, j, TC, xsize, ysize, n, x[11], y[11], dist[11][11], memo[11][1 << 11];

int tsp(int pos, int bitmask) { // bitmask stores the visited coordinates
  if (bitmask == (1 << (n + 1)) - 1)
    return dist[pos][0]; // return trip to close the loop
  if (memo[pos][bitmask] != -1)
    return memo[pos][bitmask];

  int ans = 2000000000;
  for (int nxt = 0; nxt <= n; nxt++) // O(n) here
    if (nxt != pos && !(bitmask & (1 << nxt))) // if coord. nxt is not visited yet
      ans = min(ans, dist[pos][nxt] + tsp(nxt, bitmask | (1 << nxt)));
  return memo[pos][bitmask] = ans;
}

int main() {
  scanf("%d", &TC);
  while (TC--) {
    scanf("%d %d", &xsize, &ysize); // these two values are not used
    scanf("%d %d", &x[0], &y[0]);
    scanf("%d", &n);
    for (i = 1; i <= n; i++) // karel's position is at index 0
      scanf("%d %d", &x[i], &y[i]);

    for (i = 0; i <= n; i++) // build distance table
      for (j = 0; j <= n; j++)
        dist[i][j] = abs(x[i] - x[j]) + abs(y[i] - y[j]); // Manhattan distance

    memset(memo, -1, sizeof memo);
    printf("The shortest path has length %d\n", tsp(0, 1)); // DP-TSP
  }

  return 0;
}
```

```cpp
/******************************************************************************/
/** Qtd. de formas de obter um numero N somando K numeros ..................... */
/******************************************************************************/

// top-down

/* */
#include <cstdio>
#include <cstring>
using namespace std;

int N, K, memo[110][110];

int ways(int N, int K) {
  if (K == 1) // only can use 1 number to add up to N
    return 1; // the answer is definitely 1, that number itself
  else if (memo[N][K] != -1)
    return memo[N][K];

  // if K > 1, we can choose one number from [0..N] to be one of the number and
  // recursively compute the rest
  int total_ways = 0;
  for (int split = 0; split <= N; split++)                        // we just need
    total_ways = (total_ways + ways(N - split, K - 1)) % 1000000; // the modulo 1M
  return memo[N][K] = total_ways; // memoize them
}

int main() {
  memset(memo, -1, sizeof memo);
  while (scanf("%d %d", &N, &K), (N || K)) // some recursion formula + top down DP
    printf("%d\n", ways(N, K));
  return 0;
}
/* */



// bottom-up

#include <cstdio>
#include <cstring>
using namespace std;

int main() {
  int i, j, split, dp[110][110], N, K;

  memset(dp, 0, sizeof dp);

  for (i = 0; i <= 100; i++) // these are the base cases
    dp[i][1] = 1;

  for (j = 1; j < 100; j++) // these three nested loops form the correct
    for (i = 0; i <= 100; i++)                        // topological order
      for (split = 0; split <= 100 - i; split++) {
        dp[i + split][j + 1] += dp[i][j];
        dp[i + split][j + 1] %= 1000000;
      }

  while (scanf("%d %d", &N, &K), (N || K))
    printf("%d\n", dp[N][K]);

  return 0;
}
```

```cpp
/*****************************************************************************/
/** Cutting Sticks ........................................................ */
/*****************************************************************************/
// Top-Down DP

#include <algorithm>
#include <cstdio>
#include <cstring>
using namespace std;

int l, n, A[55], memo[55][55];

int cut(int left, int right) {
  if (left + 1 == right)       return 0;
  if (memo[left][right] != -1) return memo[left][right];

  int ans = 2000000000;
  for (int i = left + 1; i < right; i++)
    ans = min(ans, cut(left, i) + cut(i, right) + (A[right]-A[left]));
  return memo[left][right] = ans;
}

int main() {
  while (scanf("%d", &l), l) {
    A[0] = 0;
    scanf("%d", &n);
    for (int i = 1; i <= n; i++) scanf("%d", &A[i]);
    A[n + 1] = l;

    memset(memo, -1, sizeof memo);                       // start with left = 0
    printf("The minimum cutting is %d.\n", cut(0, n + 1)); // and right = n + 1
  }

  return 0;
}
```

```cpp
/*******************************************************************************/
/** DFS (Busca em Profundidade) ............................................ */
/*******************************************************************************/

#include <algorithm>
#include <cstdio>
#include <vector>
using namespace std;

typedef pair<int, int> ii;      // In this chapter, we will frequently use these
typedef vector<ii> vii;      // three data type shortcuts. They may look cryptic
typedef vector<int> vi;   // but shortcuts are useful in competitive programming

// normal DFS, do not change this with other values (other than 0), because
#define DFS_WHITE -1 // we usually use memset in conjunction with DFS_WHITE
#define DFS_BLACK 1

vector<vii> AdjList;

void printThis(char* message) {
  printf("==================================\n");
  printf("%s\n", message);
  printf("==================================\n");
}

vi dfs_num;     // this variable has to be global, we cannot put it in recursion
int numCC;

void dfs(int u) {          // DFS for normal usage: as graph traversal algorithm
  printf(" %d", u);                                    // this vertex is visited
  dfs_num[u] = DFS_BLACK;      // important step: we mark this vertex as visited
  for (int j = 0; j < (int)AdjList[u].size(); j++) {
    ii v = AdjList[u][j];                          // v is a (neighbor, weight) pair
    if (dfs_num[v.first] == DFS_WHITE)             // important check to avoid cycle
      dfs(v.first);      // recursively visits unvisited neighbors v of vertex u
} }

// note: this is not the version on implicit graph
void floodfill(int u, int color) {
  dfs_num[u] = color;                                  // not just a generic DFS_BLACK
  for (int j = 0; j < (int)AdjList[u].size(); j++) {
    ii v = AdjList[u][j];
    if (dfs_num[v.first] == DFS_WHITE)
      floodfill(v.first, color);
} }

vi topoSort;             // global vector to store the toposort in reverse order

void dfs2(int u) {    // change function name to differentiate with original dfs
  dfs_num[u] = DFS_BLACK;
  for (int j = 0; j < (int)AdjList[u].size(); j++) {
    ii v = AdjList[u][j];
    if (dfs_num[v.first] == DFS_WHITE)
      dfs2(v.first);
  }
  topoSort.push_back(u); }                          // that is, this is the only change

#define DFS_GRAY 2               // one more color for graph edges property check
vi dfs_parent;      // to differentiate real back edge versus bidirectional edge

void graphCheck(int u) {                    // DFS for checking graph edge properties
  dfs_num[u] = DFS_GRAY;   // color this as DFS_GRAY (temp) instead of DFS_BLACK
  for (int j = 0; j < (int)AdjList[u].size(); j++) {
    ii v = AdjList[u][j];
    if (dfs_num[v.first] == DFS_WHITE) {      // Tree Edge, DFS_GRAY to DFS_WHITE
      dfs_parent[v.first] = u;                      // parent of this children is me
      graphCheck(v.first);
    }
    else if (dfs_num[v.first] == DFS_GRAY) {                // DFS_GRAY to DFS_GRAY
      if (v.first == dfs_parent[u])          // to differentiate these two cases
```

```cpp
        printf(" Bidirectional (%d, %d) - (%d, %d)\n", u, v.first, v.first, u);
      else  // the most frequent application: check if the given graph is cyclic
        printf(" Back Edge (%d, %d) (Cycle)\n", u, v.first);
    }
    else if (dfs_num[v.first] == DFS_BLACK)              // DFS_GRAY to DFS_BLACK
      printf(" Forward/Cross Edge (%d, %d)\n", u, v.first);
  }
  dfs_num[u] = DFS_BLACK;      // after recursion, color this as DFS_BLACK (DONE)
}

vi dfs_low;       // additional information for articulation points/bridges/SCCs
vi articulation_vertex;
int dfsNumberCounter, dfsRoot, rootChildren;

void articulationPointAndBridge(int u) {
  dfs_low[u] = dfs_num[u] = dfsNumberCounter++;      // dfs_low[u] <= dfs_num[u]
  for (int j = 0; j < (int)AdjList[u].size(); j++) {
    ii v = AdjList[u][j];
    if (dfs_num[v.first] == DFS_WHITE) {                          // a tree edge
      dfs_parent[v.first] = u;
      if (u == dfsRoot) rootChildren++;  // special case, count children of root

      articulationPointAndBridge(v.first);

      if (dfs_low[v.first] >= dfs_num[u])              // for articulation point
        articulation_vertex[u] = true;          // store this information first
      if (dfs_low[v.first] > dfs_num[u])                           // for bridge
        printf(" Edge (%d, %d) is a bridge\n", u, v.first);
      dfs_low[u] = min(dfs_low[u], dfs_low[v.first]);       // update dfs_low[u]
    }
    else if (v.first != dfs_parent[u])       // a back edge and not direct cycle
      dfs_low[u] = min(dfs_low[u], dfs_num[v.first]);       // update dfs_low[u]
} }

vi S, visited;                                    // additional global variables
int numSCC;

void tarjanSCC(int u) {
  dfs_low[u] = dfs_num[u] = dfsNumberCounter++;      // dfs_low[u] <= dfs_num[u]
  S.push_back(u);           // stores u in a vector based on order of visitation
  visited[u] = 1;
  for (int j = 0; j < (int)AdjList[u].size(); j++) {
    ii v = AdjList[u][j];
    if (dfs_num[v.first] == DFS_WHITE)
      tarjanSCC(v.first);
    if (visited[v.first])                                // condition for update
      dfs_low[u] = min(dfs_low[u], dfs_low[v.first]);
  }

  if (dfs_low[u] == dfs_num[u]) {         // if this is a root (start) of an SCC
    printf("SCC %d:", ++numSCC);            // this part is done after recursion
    while (1) {
      int v = S.back(); S.pop_back(); visited[v] = 0;
      printf(" %d", v);
      if (u == v) break;
    }
    printf("\n");
} }

int main() {
  int V, total_neighbors, id, weight;

  freopen("in_01.txt", "r", stdin);

  scanf("%d", &V);
  AdjList.assign(V, vii()); // assign blank vectors of pair<int, int>s to AdjList
  for (int i = 0; i < V; i++) {
    scanf("%d", &total_neighbors);
    for (int j = 0; j < total_neighbors; j++) {
      scanf("%d %d", &id, &weight);
```

```
        AdjList[i].push_back(ii(id, weight));
      }
    }

    printThis("Standard DFS Demo (the input graph must be UNDIRECTED)");
    numCC = 0;
    dfs_num.assign(V, DFS_WHITE);     // this sets all vertices' state to DFS_WHITE
    for (int i = 0; i < V; i++)                      // for each vertex i in [0..V-1]
      if (dfs_num[i] == DFS_WHITE)                // if that vertex is not visited yet
        printf("Component %d:", ++numCC), dfs(i), printf("\n");    // 3 lines here!
    printf("There are %d connected components\n", numCC);

    printThis("Flood Fill Demo (the input graph must be UNDIRECTED)");
    numCC = 0;
    dfs_num.assign(V, DFS_WHITE);
    for (int i = 0; i < V; i++)
      if (dfs_num[i] == DFS_WHITE)
        floodfill(i, ++numCC);
    for (int i = 0; i < V; i++)
      printf("Vertex %d has color %d\n", i, dfs_num[i]);

    // make sure that the given graph is DAG
    printThis("Topological Sort (the input graph must be DAG)");
    topoSort.clear();
    dfs_num.assign(V, DFS_WHITE);
    for (int i = 0; i < V; i++)             // this part is the same as finding CCs
      if (dfs_num[i] == DFS_WHITE)
        dfs2(i);
    reverse(topoSort.begin(), topoSort.end());                   // reverse topoSort
    for (int i = 0; i < (int)topoSort.size(); i++)         // or you can simply read
      printf(" %d", topoSort[i]);             // the content of 'topoSort' backwards
    printf("\n");

    printThis("Graph Edges Property Check");
    numCC = 0;
    dfs_num.assign(V, DFS_WHITE); dfs_parent.assign(V, -1);
    for (int i = 0; i < V; i++)
      if (dfs_num[i] == DFS_WHITE)
        printf("Component %d:\n", ++numCC), graphCheck(i);        // 2 lines in one

    printThis("Articulation Points & Bridges (the input graph must be UNDIRECTED)");
    dfsNumberCounter = 0; dfs_num.assign(V, DFS_WHITE); dfs_low.assign(V, 0);
    dfs_parent.assign(V, -1); articulation_vertex.assign(V, 0);
    printf("Bridges:\n");
    for (int i = 0; i < V; i++)
      if (dfs_num[i] == DFS_WHITE) {
        dfsRoot = i; rootChildren = 0;
        articulationPointAndBridge(i);
        articulation_vertex[dfsRoot] = (rootChildren > 1); }        // special case
    printf("Articulation Points:\n");
    for (int i = 0; i < V; i++)
      if (articulation_vertex[i])
        printf(" Vertex %d\n", i);

    printThis("Strongly Connected Components (the input graph must be DIRECTED)");
    dfs_num.assign(V, DFS_WHITE); dfs_low.assign(V, 0); visited.assign(V, 0);
    dfsNumberCounter = numSCC = 0;
    for (int i = 0; i < V; i++)
      if (dfs_num[i] == DFS_WHITE)
        tarjanSCC(i);

    return 0;
  }
```

```cpp
/*******************************************************************************/
/** Flood Fill / grafo implicito em matriz ................................... */
/*******************************************************************************/

/* Wetlands of Florida */

// classic DFS flood fill

#include <cstdio>
#include <cstring>
using namespace std;

#define REP(i, a, b) \
  for (int i = int(a); i <= int(b); i++)

char line[150], grid[150][150];
int TC, R, C, row, col;

int dr[] = {1,1,0,-1,-1,-1, 0, 1}; // S,SE,E,NE,N,NW,W,SW
int dc[] = {0,1,1, 1, 0,-1,-1,-1}; // neighbors
int floodfill(int r, int c, char c1, char c2) {
  if (r<0 || r>=R || c<0 || c>=C) return 0; // outside
  if (grid[r][c] != c1) return 0; // we want only c1
  grid[r][c] = c2; // important step to avoid cycling!
  int ans = 1; // coloring c1 -> c2, add 1 to answer
  REP (d, 0, 7) // recurse to neighbors
    ans += floodfill(r + dr[d], c + dc[d], c1, c2);
  return ans;
}

// inside the int main() of the solution for UVa 469 - Wetlands of Florida
int main() {
  // read the implicit graph as global 2D array 'grid'/R/C and
  sscanf(gets(line), "%d", &TC);                    // (row, col) query coordinate
  gets(line); // remove dummy line

  while (TC--) {
    R = 0;
    while (1) {
      gets(grid[R]);
      if (grid[R][0] != 'L' && grid[R][0] != 'W') // start of query
        break;
      R++;
    }
    C = (int)strlen(grid[0]);

    strcpy(line, grid[R]);
    while (1) {
      sscanf(line, "%d %d", &row, &col); row--; col--; // index starts from 0!
      printf("%d\n", floodfill(row, col, 'W', '.'));
      // change water 'W' to '.'; count size of this lake
      floodfill(row, col, '.', 'W'); // restore for next query
      gets(line);
      if(strcmp(line, "") == 0 || feof(stdin)) // next test case or last test case
        break;
    }

    if (TC)
      printf("\n");
  }

  return 0;
}
```

```cpp
/********************************************************************************/
/** Kruskal e Prim (Arvore Geradora Minima) ................................... */
/********************************************************************************/

#include <algorithm>
#include <cstdio>
#include <vector>
#include <queue>
using namespace std;

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

// Union-Find Disjoint Sets Library written in OOP manner,
// using both path compression and union by rank heuristics
class UnionFind {                                              // OOP style
private:
  vi p, rank, setSize;                       // remember: vi is vector<int>
  int numSets;
public:
  UnionFind(int N) {
    setSize.assign(N, 1); numSets = N; 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) { return findSet(i) == findSet(j); }
  void unionSet(int i, int j) {
    if (!isSameSet(i, j)) { numSets--;
    int x = findSet(i), y = findSet(j);
    // rank is used to keep the tree short
    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 numDisjointSets() { return numSets; }
  int sizeOfSet(int i) { return setSize[findSet(i)]; }
};

vector<vii> AdjList;
vi taken;                                  // global boolean flag to avoid cycle
priority_queue<ii> pq;            // priority queue to help choose shorter edges

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

int main() {
  int V, E, u, v, w;

  /*
  // Graph in Figure 4.10 left, format: list of weighted edges
  // This example shows another form of reading graph input
  5 7
  0 1 4
  0 2 4
  0 3 6
  0 4 6
  1 2 2
  2 3 8
  3 4 9
  */

  freopen("in_03.txt", "r", stdin);

  scanf("%d %d", &V, &E);
  // Kruskal's algorithm merged with Prim's algorithm
  AdjList.assign(V, vii());
  vector< pair<int, ii> > EdgeList;   // (weight, two vertices) of the edge
```

```
  for (int i = 0; i < E; i++) {
    scanf("%d %d %d", &u, &v, &w);           // read the triple: (u, v, w)
    EdgeList.push_back(make_pair(w, ii(u, v)));                // (w, u, v)
    AdjList[u].push_back(ii(v, w));
    AdjList[v].push_back(ii(u, w));
  }
  sort(EdgeList.begin(), EdgeList.end()); // sort by edge weight O(E log E)
                      // note: pair object has built-in comparison function

  int mst_cost = 0;
  UnionFind UF(V);                     // all V are disjoint sets initially
  for (int i = 0; i < E; i++) {                      // for each edge, O(E)
    pair<int, ii> front = EdgeList[i];
    if (!UF.isSameSet(front.second.first, front.second.second)) {  // check
      mst_cost += front.first;                // add the weight of e to MST
      UF.unionSet(front.second.first, front.second.second);    // link them
  } }                            // note: the runtime cost of UFDS is very light

  // note: the number of disjoint sets must eventually be 1 for a valid MST
  printf("MST cost = %d (Kruskal's)\n", mst_cost);



// inside int main() --- assume the graph is stored in AdjList, pq is empty
  taken.assign(V, 0);                  // no vertex is taken at the beginning
  process(0);    // take vertex 0 and process all edges incident to vertex 0
  mst_cost = 0;
  while (!pq.empty()) {  // repeat until V vertices (E=V-1 edges) are taken
    ii front = pq.top(); pq.pop();
    u = -front.second, w = -front.first;  // negate the id and weight again
    if (!taken[u])                     // we have not connected this vertex yet
      mst_cost += w, process(u); // take u, process all edges incident to u
  }                                         // each edge is in pq only once!
  printf("MST cost = %d (Prim's)\n", mst_cost);

  return 0;
}
```

```cpp
/*******************************************************************************/
/** BFS (Busca em Largura/Amplitude) ........................................ */
/*******************************************************************************/

#include <algorithm>
#include <cstdio>
#include <vector>
#include <queue>
using namespace std;

typedef pair<int, int> ii;      // In this chapter, we will frequently use these
typedef vector<ii> vii;      // three data type shortcuts. They may look cryptic
typedef vector<int> vi;   // but shortcuts are useful in competitive programming

int V, E, a, b, s;
vector<vii> AdjList;
vi p;                                 // addition: the predecessor/parent vector

void printPath(int u) {    // simple function to extract information from 'vi p'
  if (u == s) { printf("%d", u); return; }
  printPath(p[u]);   // recursive call: to make the output format: s -> ... -> t
  printf(" %d", u); }

int main() {
  /*
  // Graph in Figure 4.3, format: list of unweighted edges
  // This example shows another form of reading graph input
  13 16
  0 1    1 2    2  3    0  4    1  5    2  6    3  7    5  6
  4 8    8 9    5 10    6 11    7 12    9 10    10 11   11 12
  */

  freopen("in_04.txt", "r", stdin);

  scanf("%d %d", &V, &E);

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

  // as an example, we start from this source, see Figure 4.3
  s = 5;

  // BFS routine
  // inside int main() -- we do not use recursion,
  // thus we do not need to create separate function!
  vi dist(V, 1000000000); dist[s] = 0;      // distance to source is 0 (default)
  queue<int> q; q.push(s);                                 // start from source
  p.assign(V, -1); // to store parent information (p must be a global variable!)
  int layer = -1;                              // for our output printing purpose
  bool isBipartite = true;       // addition of one boolean flag, initially true

  while (!q.empty()) {
    int u = q.front(); q.pop();                         // queue: layer by layer!
    if (dist[u] != layer) printf("\nLayer %d: ", dist[u]);
    layer = dist[u];
    printf("visit %d, ", u);
    for (int j = 0; j < (int)AdjList[u].size(); j++) {
      ii v = AdjList[u][j];                           // for each neighbors of u
      if (dist[v.first] == 1000000000) {
        dist[v.first] = dist[u] + 1;                  // v unvisited + reachable
        p[v.first] = u;          // addition: the parent of vertex v->first is u
        q.push(v.first);                              // enqueue v for next step
      }
      else if ((dist[v.first] % 2) == (dist[u] % 2))              // same parity
        isBipartite = false;
  } }
```

```

  printf("\nShortest path: ");
  printPath(7), printf("\n");
  printf("isBipartite? %d\n", isBipartite);

  return 0;
}
```

```cpp
/*********************************************************************************/
/** Dijkstra .................................................................. */
/*********************************************************************************/

#include <cstdio>
#include <vector>
#include <queue>
using namespace std;

typedef pair<int, int> ii;
typedef vector<int> vi;
typedef vector<ii> vii;
#define INF 1000000000

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

  /*
  // Graph in Figure 4.17
  5 7 2
  2 1 2
  2 3 7
  2 0 6
  1 3 3
  1 4 6
  3 4 5
  0 4 1
  */

  freopen("in_05.txt", "r", stdin);

  scanf("%d %d %d", &V, &E, &s);

  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;
}
```

```cpp
/*********************************************************************************/
/** Bellman-Ford ............................................................. */
/*********************************************************************************/

#include <algorithm>
#include <cstdio>
#include <vector>
#include <queue>
using namespace std;

typedef pair<int, int> ii;
typedef vector<int> vi;
typedef vector<ii> vii;
#define INF 1000000000

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

  /*
  // Graph in Figure 4.18, has negative weight, but no negative cycle
  5 5 0
  0 1 1
  0 2 10
  1 3 2
  2 3 -10
  3 4 3

  // Graph in Figure 4.19, negative cycle exists
  3 3 0
  0 1 1000
  1 2 15
  2 1 -42
  */

  freopen("in_06.txt", "r", stdin);

  scanf("%d %d %d", &V, &E, &s);

  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;
}
```

```
/******************************************************************************/
/** Floyd-Warshall ......................................................... */
/******************************************************************************/

#include <algorithm>
#include <cstdio>
using namespace std;

#define INF 1000000000

int main() {
  int V, E, u, v, w, AdjMatrix[200][200];

  /*
  // Graph in Figure 4.30
  5 9
  0 1 2
  0 2 1
  0 4 3
  1 3 4
  2 1 1
  2 4 1
  3 0 1
  3 2 3
  3 4 5
  */

  freopen("in_07.txt", "r", stdin);

  scanf("%d %d", &V, &E);
  for (int i = 0; i < V; i++) {
    for (int j = 0; j < V; j++)
      AdjMatrix[i][j] = INF;
    AdjMatrix[i][i] = 0;
  }

  for (int i = 0; i < E; i++) {
    scanf("%d %d %d", &u, &v, &w);
    AdjMatrix[u][v] = w; // directed graph
  }

  for (int k = 0; k < V; k++) // common error: remember that loop order is k->i->j
    for (int i = 0; i < V; i++)
      for (int j = 0; j < V; j++)
        AdjMatrix[i][j] = min(AdjMatrix[i][j], AdjMatrix[i][k] + AdjMatrix[k][j]);

  for (int i = 0; i < V; i++)
    for (int j = 0; j < V; j++)
      printf("APSP(%d, %d) = %d\n", i, j, AdjMatrix[i][j]);

  return 0;
}
```

```
/**********************************************************************************/
/** Edmonds-Karp ............................................................... */
/**********************************************************************************/

#include <algorithm>
#include <cstdio>
#include <vector>
#include <queue>
using namespace std;

typedef vector<int> vi;

#define MAX_V 40 // enough for sample graph in Figure 4.24/4.25/4.26/UVa 259
#define INF 1000000000

int res[MAX_V][MAX_V], mf, f, s, t;                          // global variables
vi p;

void augment(int v, int minEdge) {     // traverse BFS spanning tree from s to t
  if (v == s) { f = minEdge; return; }  // record minEdge in a global variable f
  else if (p[v] != -1) { augment(p[v], min(minEdge, res[p[v]][v])); // recursive
                         res[p[v]][v] -= f; res[v][p[v]] += f; }       // update
}

int main() {
  int V, k, vertex, weight;

  /*
  // Graph in Figure 4.24
  4 0 1
  2 2 70 3 30
  2 2 25 3 70
  3 0 70 3 5 1 25
  3 0 30 2 5 1 70

  // Graph in Figure 4.25
  4 0 3
  2 1 100 3 100
  2 2 1 3 100
  1 3 100
  0

  // Graph in Figure 4.26.A
  5 1 0
  0
  2 2 100 3 50
  3 3 50 4 50 0 50
  1 4 100
  1 0 125

  // Graph in Figure 4.26.B
  5 1 0
  0
  2 2 100 3 50
  3 3 50 4 50 0 50
  1 4 100
  1 0 75

  // Graph in Figure 4.26.C
  5 1 0
  0
  2 2 100 3 50
  2 4 5 0 5
  1 4 100
  1 0 125
  */

  freopen("in_08.txt", "r", stdin);

  scanf("%d %d %d", &V, &s, &t);
```

```cpp

  memset(res, 0, sizeof res);
  for (int i = 0; i < V; i++) {
    scanf("%d", &k);
    for (int j = 0; j < k; j++) {
      scanf("%d %d", &vertex, &weight);
      res[i][vertex] = weight;
    }
  }

  mf = 0;                                                    // mf stands for max_flow
  while (1) {                // O(VE^2) (actually O(V^3E) Edmonds Karp's algorithm
    f = 0;
    // run BFS, compare with the original BFS shown in Section 4.2.2
    vi dist(MAX_V, INF); dist[s] = 0; queue<int> q; q.push(s);
    p.assign(MAX_V, -1);            // record the BFS spanning tree, from s to t!
    while (!q.empty()) {
      int u = q.front(); q.pop();
      if (u == t) break;      // immediately stop BFS if we already reach sink t
      for (int v = 0; v < MAX_V; v++)                  // note: this part is slow
        if (res[u][v] > 0 && dist[v] == INF)
          dist[v] = dist[u] + 1, q.push(v), p[v] = u;
    }
    augment(t, INF);      // find the min edge weight 'f' along this path, if any
    if (f == 0) break;       // we cannot send any more flow ('f' = 0), terminate
    mf += f;                   // we can still send a flow, increase the max flow!
  }

  printf("%d\n", mf);                                  // this is the max flow value

  return 0;
}



/* */

#include <algorithm>
#include <bitset>
#include <cstdio>
#include <vector>
#include <queue>
using namespace std;

typedef vector<int> vi;

#define MAX_V 40 // enough for sample graph in Figure 4.24/4.25/4.26/UVa 259
#define INF 1000000000

int res[MAX_V][MAX_V], mf, f, s, t;                            // global variables
vi p;
vector<vi> AdjList;

void augment(int v, int minEdge) {      // traverse BFS spanning tree from s to t
  if (v == s) { f = minEdge; return; }  // record minEdge in a global variable f
  else if (p[v] != -1) { augment(p[v], min(minEdge, res[p[v]][v])); // recursive
                         res[p[v]][v] -= f; res[v][p[v]] += f; }        // update
}

int main() {
  int V, k, vertex, weight;

  scanf("%d %d %d", &V, &s, &t);

  memset(res, 0, sizeof res);
  AdjList.assign(V, vi());
  for (int i = 0; i < V; i++) {
    scanf("%d", &k);
    for (int j = 0; j < k; j++) {
      scanf("%d %d", &vertex, &weight);
```

```
      res[i][vertex] = weight;
      AdjList[i].push_back(vertex);
    }
  }

  mf = 0;
  while (1) {                         // now a true O(VE^2) Edmonds Karp's algorithm
    f = 0;
    bitset<MAX_V> vis; vis[s] = true;            // we change vi dist to bitset!
    queue<int> q; q.push(s);
    p.assign(MAX_V, -1);
    while (!q.empty()) {
      int u = q.front(); q.pop();
      if (u == t) break;
      for (int j = 0; j < (int)AdjList[u].size(); j++) {  // we use AdjList here!
        int v = AdjList[u][j];
        if (res[u][v] > 0 && !vis[v])
          vis[v] = true, q.push(v), p[v] = u;
      }
    }
    augment(t, INF);
    if (f == 0) break;
    mf += f;
  }

  printf("%d\n", mf);                              // this is the max flow value

  return 0;
}

/* */
```

```
/*******************************************************************************/
/** Emparelhamento Maximo em Grafos Bipartidos ................................ */
/*******************************************************************************/

#include <cstdio>
#include <iostream>
#include <vector>
using namespace std;

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

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
}

bool isprime(int v) {
  int primes[10] = {2,3,5,7,11,13,17,19,23,29};
  for (int i = 0; i < 10; i++)
    if (primes[i] == v)
      return true;
  return false;
}

int main() {
// inside int main()
  // build bipartite graph with directed edge from left to right set

/*
  // Graph in Figure 4.40 can be built on the fly
  // we know there are 6 vertices in this bipartite graph,
  // left side are numbered 0,1,2, right side 3,4,5
  int V = 6, Vleft = 3, set1[3] = {1,7,11}, set2[3] = {4,10,12};

  // Graph in Figure 4.41 can be built on the fly
  // we know there are 5 vertices in this bipartite graph,
  // left side are numbered 0,1, right side 3,4,5
  //int V = 5, Vleft = 2, set1[2] = {1,7}, set2[3] = {4,10,12};

  // build the bipartite graph, only directed edge from left to right is needed
  AdjList.assign(V, vi());
  for (int i = 0; i < Vleft; i++)
    for (int j = 0; j < 3; j++)
      if (isprime(set1[i] + set2[j]))
        AdjList[i].push_back(3 + j);
*/

  // For bipartite graph in Figure 4.44, V = 5, Vleft = 3 (vertex 0 unused)
  // AdjList[0] = {} // dummy vertex, but you can choose to use this vertex
  // AdjList[1] = {3, 4}
  // AdjList[2] = {3}
  // AdjList[3] = {}   // we use directed edges from left to right set only
  // AdjList[4] = {}

  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;
}
```

```cpp
/******************************************************************************/
/** BigInt (implementacao em C/C++) ....................................... */
/******************************************************************************/

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

#define TAMAX 12345 /* precisa ser alterado a cada problema */
#define BASE 100000000 /* 10^8 */

typedef struct bigint {
  unsigned long long V[TAMAX];
  int tam;
} bigint;

/* le o bigint apontado por I
   nao tem problema se em *I ja existe algum valor ou nao
   devolve 1 se consegue ler ou 0 c.c. <== util para pÃ´r num while
   CUIDADO: considera que o fim do bigint eh marcado por \n
            se nao for o caso, NAO use fgets
            por ex: se for um espaÃ§o que marca o fim do bigint,
                    use getchar() dentro dum for */
int leia_bigint(bigint *I) {
  char S[8*TAMAX];
  int i, j, pot;
  fgets(S, 8*TAMAX, stdin);
  i = strlen(S) - 2;
  if (i == -1) return 0;
  I->tam = 0;
  while (i >= 0) {
    pot = 1;
    I->V[I->tam] = 0;
    for (j = 1; j <= 8 && i >= 0; j++, i--) {
      I->V[I->tam] = I->V[I->tam] + pot*(S[i] - '0');
      pot *= 10;
    }
    I->tam++;
  }
  return 1;
}

/* imprime um bigint
   eh necessario que *I seja um bigint valido */
void imprima_bigint(bigint *I) {
  int i;
  printf("%llu", I->V[I->tam - 1]);
  for (i = I->tam - 2; i >= 0; i--)
    printf("%08llu", I->V[i]);
  printf("\n");
}

/* retorna 1 se *I1 < *I2 ou 0 caso contrario
   necessario que ambos sejam bigint's validos */
int menor_bigint(bigint *I1, bigint *I2) {
  int i;
  if (I1->tam < I2->tam) return 1;
  if (I1->tam > I2->tam) return 0;
  for (i = I1->tam - 1; i >= 0 && I1->V[i] == I2->V[i]; i--);
  if (i == -1) return 0;
  if (I1->V[i] < I2->V[i]) return 1;
  return 0;
}

/* retorna 1 se *I1 = *I2 ou 0 caso contrario
   necessario que ambos sejam bigint's validos */
int igual_bigint(bigint *I1, bigint *I2) {
  int i;
  if (I1->tam < I2->tam) return 0;
  if (I1->tam > I2->tam) return 0;
  for (i = I1->tam - 1; i >= 0 && I1->V[i] == I2->V[i]; i--);
```

```
  if (i == -1) return 1;
  return 0;
}

/* copia *I1 para *I2
   necessario que *I1 seja um bigint valido
   nao tem problema se em *I2 ja existe algum valor ou nao */
void copie_bigint(bigint *I1, bigint *I2) {
  int i;
  I2->tam = I1->tam;
  for (i = 0; i < I1->tam; i++)
    I2->V[i] = I1->V[i];
}

/* remove eventuais zeros Ã  esquerda de *I
   necessario que *I seja um bigint valido,
   apenas podendo ter zeros Ã  esquerda */
void conserte_zeros(bigint *I) {
  int i;
  for (i = I->tam - 1; i >= 0 && I->V[i] == 0; i--);
  if (i == -1) I->tam = 1;
  else I->tam = i + 1;
}

/* efetua *I1 + *I2 e armazena o resultado em *I3
   necessario que *I1 e *I2 sejam bigint's validos
   necessario que max(I1->tam, I2->tam) < TAMAX
   nao tem problema se em *I3 ja existe algum valor ou nao
   funciona mesmo se o ponteiro I3 = I1 ou I3 = I2 */
void some_bigint(bigint *I1, bigint *I2, bigint *I3) {
  int i;
  unsigned long long carry=0, valor1, valor2;
  for (i = 0; i < I1->tam || i < I2->tam; i++) {
    valor1 = i < I1->tam ? I1->V[i] : 0;
    valor2 = i < I2->tam ? I2->V[i] : 0;
    I3->V[i] = valor1 + valor2 + carry;
    carry = I3->V[i] / BASE;
    I3->V[i] = I3->V[i] % BASE;
  }
  I3->V[i] = carry;
  I3->tam = i + 1;
  conserte_zeros(I3);
}

/* efetua *I1 + *I2 e armazena o resultado em *I3
   necessario que *I1 e *I2 sejam bigint's validos
   nao tem problema se em *I3 ja existe algum valor ou nao
   funciona mesmo se o ponteiro I3 = I1 ou I3 = I2
   funciona apenas se garantidamente *I1 >= *I2 */
void subtraia_bigint(bigint *I1, bigint *I2, bigint *I3) {
  int i;
  unsigned long long carry=0, valor2;
  for (i = 0; i < I1->tam || i < I2->tam; i++) {
    valor2 = i < I2->tam ? I2->V[i] : 0;
    if (I1->V[i] < valor2 + carry) {
      I3->V[i] = BASE + I1->V[i] - valor2 - carry;
      carry = 1;
    } else {
      I3->V[i] = I1->V[i] - valor2 - carry;
      carry = 0;
    }
  }
  I3->tam = I1->tam;
  conserte_zeros(I3);
}

/* efetua *I1 * a e armazena o resultado em *I2
   necessario que *I1 seja um bigint valido e 0 <= a < BASE
   necessario que I1->tam < TAMAX
   nao tem problema se em *I2 ja existe algum valor ou nao
```

```
   funciona mesmo se o ponteiro I2 = I1 */
void mult_escalar(bigint *I1, unsigned long long a, bigint *I2) {
  int i;
  unsigned long long carry=0;
  for (i = 0; i < I1->tam; i++) {
    I2->V[i] = I1->V[i] * a + carry;
    carry = I2->V[i] / BASE;
    I2->V[i] = I2->V[i] % BASE;
  }
  I2->V[i] = carry;
  I2->tam = i + 1;
  conserte_zeros(I2);
}

/* desloca *I i dÃ-gitos (ref. base BASE, nao base 10) para a esquerda
   necessario que *I seja um bigint valido e que i >= 0
   necessario que I->tam + i <= TAMAX */
void lshift_bigint(bigint *I, int i) {
  int j;
  for (j = I->tam - 1 + i; j >= i; j--)
    I->V[j] = I->V[j-i];
  for (; j >= 0; j--) I->V[j] = 0;
  I->tam = I->tam + i;
  conserte_zeros(I);
}

/* efetua *I1 * *I2 e armazena o resultado em *I3
   necessario que *I1 e *I2 sejam bigint's validos
   necessario que I1->tam + I2->tam < TAMAX
   nao tem problema se em *I3 ja existe algum valor ou nao
   funciona mesmo se o ponteiro I3 = I1 ou I3 = I2 */
void mult_bigint(bigint *I1, bigint *I2, bigint *I3) {
  bigint tmp, soma;
  int i;
  soma.tam = 1; soma.V[0] = 0;
  for (i = 0; i < I1->tam; i++) {
    mult_escalar(I2, I1->V[i], &tmp);
    lshift_bigint(&tmp, i);
    some_bigint(&soma, &tmp, &soma);
  }
  copie_bigint(&soma, I3);
}

/* efetua a divisao inteira de *I1 por *I2 e armazena o quociente em
   *quoc e o resto em *mod
   necessario que *I1 e *I2 sejam bigint's validos
   nao tem problema se em *quoc ou *mod ja existem valores ou nao
   funciona mesmo se ha coincidencias entre os ponteiros */
void div_bigint(bigint *I1, bigint *I2, bigint *quoc, bigint *mod) {
        unsigned long long d;
        int diftam;
        bigint tmp, dtmp;
        copie_bigint(I1, mod);
        quoc->V[0] = 0;
        quoc->tam = 1;
        while (menor_bigint(I2, mod) || igual_bigint(I2, mod)) {
                if (mod->V[mod->tam-1] >= I2->V[I2->tam-1])
                        d = mod->V[mod->tam-1] / I2->V[I2->tam-1];
                else
                        d = (mod->V[mod->tam-1]*BASE + mod->V[mod->tam-2])
                                  / I2->V[I2->tam-1];
                dtmp.V[0] = d;
                dtmp.tam = 1;
                while (1) {
                        mult_escalar(I2, d, &tmp);
                        diftam = mod->tam - tmp.tam;
                        lshift_bigint(&tmp, diftam);
                        lshift_bigint(&dtmp, diftam);
                        if (menor_bigint(mod, &tmp)) {
              d--;
```

```
                dtmp.V[0] = d;
                dtmp.tam = 1;
              }
                         else break;
              }
              subtraia_bigint(mod, &tmp, mod);
              some_bigint(quoc, &dtmp, quoc);
        }
}

/* exemplo de uso */
int main(void) {
  bigint I1, I2, quoc, mod;
  leia_bigint(&I1);
  leia_bigint(&I2);
  div_bigint(&I1, &I2, &quoc, &mod);
  imprima_bigint(&quoc);
  imprima_bigint(&mod);
  return 0;
}
```

```cpp
/*****************************************************************************/
/** Crivo de Eratostenes (descobre n's primos) ................................ */
/*****************************************************************************/

#include <bitset>   // compact STL for Sieve, more efficient than vector<bool>!
#include <cmath>
#include <cstdio>
#include <map>
#include <vector>
using namespace std;

typedef long long ll;
typedef vector<int> vi;
typedef map<int, int> mii;

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((int)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;
}

int main() {
  // first part: the Sieve of Eratosthenes
  sieve(10000000);                          // can go up to 10^7 (need few seconds)
  printf("%d\n", isPrime(2147483647));                        // 10-digits prime
  printf("%d\n", isPrime(136117223861LL));        // not a prime, 104729*1299709


  // second part: prime factors
  vi res = primeFactors(2147483647);   // slowest, 2147483647 is a prime
  for (vi::iterator i = res.begin(); i != res.end(); i++) printf("> %d\n", *i);

  res = primeFactors(136117223861LL);   // slow, 2 large pfactors 104729*1299709
  for (vi::iterator i = res.begin(); i != res.end(); i++) printf("# %d\n", *i);

  res = primeFactors(142391208960LL);   // faster, 2^10*3^4*5*7^4*11*13
  for (vi::iterator i = res.begin(); i != res.end(); i++) printf("! %d\n", *i);

  //res = primeFactors((ll)(1010189899 * 1010189899)); // "error"
```

```
  //for (vi::iterator i = res.begin(); i != res.end(); i++) printf("^ %d\n", *i);


  // third part: prime factors variants
  printf("numPF(%d) = %lld\n", 50, numPF(50)); // 2^1 * 5^2 => 3
  printf("numDiffPF(%d) = %lld\n", 50, numDiffPF(50)); // 2^1 * 5^2 => 2
  printf("sumPF(%d) = %lld\n", 50, sumPF(50)); // 2^1 * 5^2 => 2 + 5 + 5 = 12
  printf("numDiv(%d) = %lld\n", 50, numDiv(50));// 1, 2, 5, 10, 25, 50, 6 divisors
  printf("sumDiv(%d) = %lld\n", 50, sumDiv(50)); // 1 + 2 + 5 + 10 + 25 + 50 = 93
  printf("EulerPhi(%d) = %lld\n", 50, EulerPhi(50));
                                // 20 integers < 50 are relatively prime with 50
  return 0;
}
```

```cpp
/********************************************************************************/
/** Floyd's Cycle-Finding Algorithm ........................................... */
/********************************************************************************/

#include <cstdio>
#include <iostream>
using namespace std;

typedef pair<int, int> ii;

int caseNo = 1, Z, I, M, L;

int f(int x) { return (Z * x + I) % M; }

ii floydCycleFinding(int x0) {  // function int f(int x) is defined earlier
  // 1st part: finding k*mu, hare's speed is 2x tortoise's
  int tortoise = f(x0), hare = f(f(x0));     // f(x0) is the node next to x0
  while (tortoise != hare) { tortoise = f(tortoise); hare = f(f(hare)); }
  // 2nd part: finding mu, hare and tortoise move at the same speed
  int mu = 0; hare = x0;
  while (tortoise != hare) { tortoise = f(tortoise); hare = f(hare); mu++; }
  // 3rd part: finding lambda, hare moves, tortoise stays
  int lambda = 1; hare = f(tortoise);
  while (tortoise != hare) { hare = f(hare); lambda++; }
  return ii(mu, lambda);
}

int main() {
  while (scanf("%d %d %d %d", &Z, &I, &M, &L), (Z || I || M || L)) {
    ii result = floydCycleFinding(L);
    printf("Case %d: %d\n", caseNo++, result.second);
  }
  return 0;
}
```

```cpp
/*******************************************************************************/
/** Strings (algoritmos basicos) ............................................ */
/*******************************************************************************/

#include <algorithm>
#include <ctype.h> // no equivalent C++ version: note that C++ can use C features
#include <iostream>
#include <map>
#include <fstream>
#include <sstream>
#include <string> // string class
#include <string.h>
#include <vector>
using namespace std;

int isvowel(char ch) { // make sure ch is in lowercase
  char vowel[6] = "aeiou";
  for (int j = 0; vowel[j]; j++)
    if (vowel[j] == ch)
      return 1;
  return 0;
}

int main() {
  int i, pos, digits, alphas, vowels, consonants;
  bool first = true, prev_dash, this_dash;
  char str[10010], line[110], *p;

  freopen("ch6.txt", "r", stdin);

  strcpy(str, "");
  first = true; // technique to differentiate first line with the other lines
  prev_dash = this_dash = false; // to differentiate whether the previous line
  while (1) {                                       // contains a dash or not
    fgets(line, 100, stdin);
    line[(int)strlen(line) - 2] = 0; // delete dummy char
    if (strncmp(line, ".......", 7) == 0) break;
    if (line[(int)strlen(line) - 1] == '-') {
      // if the last character is '-', delete it by moving the NULL (0)
      line[(int)strlen(line) - 1] = 0;         // one character forward
      this_dash = true;
    }
    else
      this_dash = false;
    if (!first && !prev_dash)
      strcat(str, " "); // only append " " if this line is the second one onwards
    first = false;
    strcat(str, line);
    prev_dash = this_dash;
  }
  //we can use str[i] as terminating condition as string in C++ is also terminated
  for(i = digits = alphas = vowels = consonants = 0; str[i]; i++) { // w/ NULL (0)
    str[i] = tolower(str[i]); // make each character lower case
    digits += isdigit(str[i]) ? 1 : 0;
    alphas += isalpha(str[i]) ? 1 : 0;
    vowels += isvowel(str[i]); // already returns 1 or 0
  }
  consonants = alphas - vowels;
  printf("%s\n", str);
  printf("%d %d %d\n", digits, vowels, consonants);
  int hascs3233 = (strstr(str, "cs3233") != NULL);

  vector<string> tokens;
  map<string, int> freq;
  for (p = strtok(str, " ."); p; p = strtok(NULL, " .")) {
    tokens.push_back(p); // casting from C string to C++ string is automatic
    freq[p]++;
  }

  sort(tokens.begin(), tokens.end());
```

```
  // to cast C++ string to C string, we need to use c_str()
  printf("%s %s\n", tokens[0].c_str(), tokens[(int)tokens.size() - 1].c_str());
  printf("%d\n", hascs3233);

  int ans_s = 0, ans_h = 0, ans_7 = 0;
  char ch;
  while (scanf("%c", &ch), ch != '\n') {
         if (ch == 's') ans_s++;
    else if (ch == 'h') ans_h++;
    else if (ch == '7') ans_7++;
  }
  printf("%d %d %d\n", ans_s, ans_h, ans_7);

  return 0;
}
```

```
/*****************************************************************************/
/** Knuth-Morris-Pratt (string matching) ..................................... */
/*****************************************************************************/

#include <cstdio>
#include <cstring>
#include <time.h>
using namespace std;

#define MAX_N 100010

char T[MAX_N], P[MAX_N]; // T = text, P = pattern
int b[MAX_N], n, m; // b = back table, n = length of T, m = length of P

void naiveMatching() {
  for (int i = 0; i < n; i++) { // try all potential starting indices
    bool found = true;
    for (int j = 0; j < m && found; j++) // use boolean flag 'found'
      if (i + j >= n || P[j] != T[i + j]) // if mismatch found
        found = false; // abort this, shift starting index i by +1
    if (found) // if P[0 .. m - 1] == T[i .. i + m - 1]
      printf("P is found at index %d in T\n", i);
} }

void kmpPreprocess() { // call this before calling kmpSearch()
  int i = 0, j = -1; b[0] = -1; // starting values
  while (i < m) { // pre-process the pattern string P
    while (j >= 0 && P[i] != P[j]) j = b[j]; // if different, reset j using b
    i++; j++; // if same, advance both pointers
    b[i] = j; // observe i = 8, 9, 10, 11, 12 with j = 0, 1, 2, 3, 4
} }           // in the example of P = "SEVENTY SEVEN" above

void kmpSearch() { // this is similar as kmpPreprocess(), but on string T
  int i = 0, j = 0; // starting values
  while (i < n) { // search through string T
    while (j >= 0 && T[i] != P[j]) j = b[j]; // if different, reset j using b
    i++; j++; // if same, advance both pointers
    if (j == m) { // a match found when j == m
      printf("P is found at index %d in T\n", i - j);
      j = b[j]; // prepare j for the next possible match
} } }

int main() {
  strcpy(T, "I DO NOT LIKE SEVENTY SEV BUT SEVENTY SEVENTY SEVEN");
  strcpy(P, "SEVENTY SEVEN");
  n = (int)strlen(T);
  m = (int)strlen(P);

  //if the end of line character is read too, uncomment the line below
  //T[n-1] = 0; n--; P[m-1] = 0; m--;

  printf("T = '%s'\n", T);
  printf("P = '%s'\n", P);
  printf("\n");

  clock_t t0 = clock();
  printf("Naive Matching\n");
  naiveMatching();
  clock_t t1 = clock();
  printf("Runtime = %.10lf s\n\n", (t1 - t0) / (double) CLOCKS_PER_SEC);

  printf("KMP\n");
  kmpPreprocess();
  kmpSearch();
  clock_t t2 = clock();
  printf("Runtime = %.10lf s\n\n", (t2 - t1) / (double) CLOCKS_PER_SEC);

  printf("String Library\n");
  char *pos = strstr(T, P);
  while (pos != NULL) {
```

```
    printf("P is found at index %d in T\n", pos - T);
    pos = strstr(pos + 1, P);
  }
  clock_t t3 = clock();
  printf("Runtime = %.10lf s\n\n", (t3 - t2) / (double) CLOCKS_PER_SEC);

  return 0;
}
```

```cpp
/*******************************************************************************/
/** Alinhamento de Strings (Needleman-Wunsch) ................................ */
/*******************************************************************************/

#include <algorithm>
#include <cstdio>
#include <cstring>
using namespace std;

int main() {
  char A[20] = "ACAATCC", B[20] = "AGCATGC";
  int n = (int)strlen(A), m = (int)strlen(B);
  int i, j, table[20][20]; // Needleman Wunsch's algorithm

  memset(table, 0, sizeof table);
  // insert/delete = -1 point
  for (i = 1; i <= n; i++)
    table[i][0] = i * -1;
  for (j = 1; j <= m; j++)
    table[0][j] = j * -1;

  for (i = 1; i <= n; i++)
    for (j = 1; j <= m; j++) {
      // match = 2 points, mismatch = -1 point
      table[i][j] = table[i - 1][j - 1] + (A[i - 1] == B[j - 1] ? 2 : -1);
      // insert/delete = -1 point
      table[i][j] = max(table[i][j], table[i - 1][j] - 1); // delete
      table[i][j] = max(table[i][j], table[i][j - 1] - 1); // insert
    }

  printf("DP table:\n");
  for (i = 0; i <= n; i++) {
    for (j = 0; j <= m; j++)
      printf("%3d", table[i][j]);
    printf("\n");
  }
  printf("Maximum Alignment Score: %d\n", table[n][m]);

  return 0;
}
```

```cpp
/*****************************************************************************/
/** Array de Sufixos ....................................................... */
/*****************************************************************************/

#include <algorithm>
#include <cstdio>
#include <cstring>
using namespace std;

typedef pair<int, int> ii;

#define MAX_N 100010                          // second approach: O(n log n)
char T[MAX_N];                    // the input string, up to 100K characters
int n;                                         // the length of input string
int RA[MAX_N], tempRA[MAX_N];         // rank array and temporary rank array
int SA[MAX_N], tempSA[MAX_N];     // suffix array and temporary suffix array
int c[MAX_N];                                     // for counting/radix sort

char P[MAX_N];                   // the pattern string (for string matching)
int m;                                       // the length of pattern string

int Phi[MAX_N];                       // for computing longest common prefix
int PLCP[MAX_N];
int LCP[MAX_N];  // LCP[i] stores the LCP between previous suffix T+SA[i-1]
                                               // and current suffix T+SA[i]

bool cmp(int a, int b) { return strcmp(T + a, T + b) < 0; }       // compare

void constructSA_slow() {                   // cannot go beyond 1000 characters
  for (int i = 0; i < n; i++) SA[i] = i; // initial SA: {0, 1, 2, ..., n-1}
  sort(SA, SA + n, cmp); // sort: O(n log n) * compare: O(n) = O(n^2 log n)
}

void countingSort(int k) {                                          // O(n)
  int i, sum, maxi = max(300, n);   // up to 255 ASCII chars or length of n
  memset(c, 0, sizeof c);                          // clear frequency table
  for (i = 0; i < n; i++)        // count the frequency of each integer rank
    c[i + k < n ? RA[i + k] : 0]++;
  for (i = sum = 0; i < maxi; i++) {
    int t = c[i]; c[i] = sum; sum += t;
  }
  for (i = 0; i < n; i++)           // shuffle the suffix array if necessary
    tempSA[c[SA[i]+k < n ? RA[SA[i]+k] : 0]++] = SA[i];
  for (i = 0; i < n; i++)                      // update the suffix array SA
    SA[i] = tempSA[i];
}

void constructSA() {          // this version can go up to 100000 characters
  int i, k, r;
  for (i = 0; i < n; i++) RA[i] = T[i];                 // initial rankings
  for (i = 0; i < n; i++) SA[i] = i;     // initial SA: {0, 1, 2, ..., n-1}
  for (k = 1; k < n; k <<= 1) {        // repeat sorting process log n times
    countingSort(k);  // actually radix sort: sort based on the second item
    countingSort(0);          // then (stable) sort based on the first item
    tempRA[SA[0]] = r = 0;             // re-ranking; start from rank r = 0
    for (i = 1; i < n; i++)                    // compare adjacent suffixes
      tempRA[SA[i]] = // if same pair => same rank r; otherwise, increase r
      (RA[SA[i]] == RA[SA[i-1]] && RA[SA[i]+k] == RA[SA[i-1]+k]) ? r : ++r;
    for (i = 0; i < n; i++)                     // update the rank array RA
      RA[i] = tempRA[i];
    if (RA[SA[n-1]] == n-1) break;               // nice optimization trick
} }

void computeLCP_slow() {
  LCP[0] = 0;                                              // default value
  for (int i = 1; i < n; i++) {                // compute LCP by definition
    int L = 0;                                        // always reset L to 0
    while (T[SA[i] + L] == T[SA[i-1] + L]) L++;       // same L-th char, L++
    LCP[i] = L;
} }
```

```cpp

void computeLCP() {
  int i, L;
  Phi[SA[0]] = -1;                                         // default value
  for (i = 1; i < n; i++)                           // compute Phi in O(n)
    Phi[SA[i]] = SA[i-1];    // remember which suffix is behind this suffix
  for (i = L = 0; i < n; i++) {            // compute Permuted LCP in O(n)
    if (Phi[i] == -1) { PLCP[i] = 0; continue; }            // special case
    while (T[i + L] == T[Phi[i] + L]) L++;       // L increased max n times
    PLCP[i] = L;
    L = max(L-1, 0);                             // L decreased max n times
  }
  for (i = 0; i < n; i++)                           // compute LCP in O(n)
    LCP[i] = PLCP[SA[i]];   // put the permuted LCP to the correct position
}

ii stringMatching() {                      // string matching in O(m log n)
  int lo = 0, hi = n-1, mid = lo;              // valid matching = [0..n-1]
  while (lo < hi) {                                     // find lower bound
    mid = (lo + hi) / 2;                              // this is round down
    int res = strncmp(T + SA[mid], P, m);  // try to find P in suffix 'mid'
    if (res >= 0) hi = mid;        // prune upper half (notice the >= sign)
    else          lo = mid + 1;           // prune lower half including mid
  }                                      // observe '=' in "res >= 0" above
  if (strncmp(T + SA[lo], P, m) != 0) return ii(-1, -1);    // if not found
  ii ans; ans.first = lo;
  lo = 0; hi = n - 1; mid = lo;
  while (lo < hi) {            // if lower bound is found, find upper bound
    mid = (lo + hi) / 2;
    int res = strncmp(T + SA[mid], P, m);
    if (res > 0) hi = mid;                              // prune upper half
    else         lo = mid + 1;            // prune lower half including mid
  }                           // (notice the selected branch when res == 0)
  if (strncmp(T + SA[hi], P, m) != 0) hi--;                 // special case
  ans.second = hi;
  return ans;
} // return lower/upperbound as first/second item of the pair, respectively

ii LRS() {                   // returns a pair (the LRS length and its index)
  int i, idx = 0, maxLCP = -1;
  for (i = 1; i < n; i++)                         // O(n), start from i = 1
    if (LCP[i] > maxLCP)
      maxLCP = LCP[i], idx = i;
  return ii(maxLCP, idx);
}

int owner(int idx) { return (idx < n-m-1) ? 1 : 2; }

ii LCS() {                   // returns a pair (the LCS length and its index)
  int i, idx = 0, maxLCP = -1;
  for (i = 1; i < n; i++)                         // O(n), start from i = 1
    if (owner(SA[i]) != owner(SA[i-1]) && LCP[i] > maxLCP)
      maxLCP = LCP[i], idx = i;
  return ii(maxLCP, idx);
}

int main() {
  //printf("Enter a string T below, we will compute its Suffix Array:\n");
  strcpy(T, "GATAGACA");
  n = (int)strlen(T);
  T[n++] = '$';
  // if '\n' is read, uncomment the next line
  //T[n-1] = '$'; T[n] = 0;

  constructSA_slow();                                       // O(n^2 log n)
  printf("The Suffix Array of string T = '%s' is shown below (O(n^2 log n) version):\n", T);
  printf("i\tSA[i]\tSuffix\n");
  for (int i = 0; i < n; i++) printf("%2d\t%2d\t%s\n", i, SA[i], T + SA[i]);

```

```cpp
  constructSA();                                                   // O(n log n)
  printf("\nThe Suffix Array of string T = '%s' is shown below (O(n log n) version):\n", T);
  printf("i\tSA[i]\tSuffix\n");
  for (int i = 0; i < n; i++) printf("%2d\t%2d\t%s\n", i, SA[i], T + SA[i]);

  computeLCP();                                                          // O(n)

  // LRS demo
  ii ans = LRS();                   // find the LRS of the first input string
  char lrsans[MAX_N];
  strncpy(lrsans, T + SA[ans.second], ans.first);
  printf("\nThe LRS is '%s' with length = %d\n\n", lrsans, ans.first);

  // stringMatching demo
  //printf("\nNow, enter a string P below, we will try to find P in T:\n");
  strcpy(P, "A");
  m = (int)strlen(P);
  // if '\n' is read, uncomment the next line
  //P[m-1] = 0; m--;
  ii pos = stringMatching();
  if (pos.first != -1 && pos.second != -1) {
    printf("%s is found SA[%d..%d] of %s\n", P, pos.first, pos.second, T);
    printf("They are:\n");
    for (int i = pos.first; i <= pos.second; i++)
      printf("  %s\n", T + SA[i]);
  } else printf("%s is not found in %s\n", P, T);

  // LCS demo
  //printf("\nRemember, T = '%s'\nNow, enter another string P:\n", T);
  // T already has '$' at the back
  strcpy(P, "CATA");
  m = (int)strlen(P);
  // if '\n' is read, uncomment the next line
  //P[m-1] = 0; m--;
  strcat(T, P);                                                    // append P
  strcat(T, "#");                                       // add '$' at the back
  n = (int)strlen(T);                                              // update n

  // reconstruct SA of the combined strings
  constructSA();                                                   // O(n log n)
  computeLCP();                                                          // O(n)
  printf("\nThe LCP information of 'T+P' = '%s':\n", T);
  printf("i\tSA[i]\tLCP[i]\tOwner\tSuffix\n");
  for (int i = 0; i < n; i++)
    printf("%2d\t%2d\t%2d\t%2d\t%s\n", i, SA[i], LCP[i], owner(SA[i]), T + SA[i]);

  ans = LCS();         // find the longest common substring between T and P
  char lcsans[MAX_N];
  strncpy(lcsans, T + SA[ans.second], ans.first);
  printf("\nThe LCS is '%s' with length = %d\n", lcsans, ans.first);

  return 0;
}
```

```cpp
/***************************************************************************/
/** Pontos e Linhas ...................................................... */
/***************************************************************************/

#include <algorithm>
#include <cstdio>
#include <cmath>
#include <vector>
using namespace std;

#define INF 1e9
#define EPS 1e-9
#define PI acos(-1.0) // important constant;
// alternative #define PI (2.0 * acos(0.0))

double DEG_to_RAD(double d) { return d * PI / 180.0; }

double RAD_to_DEG(double r) { return r * 180.0 / PI; }

// struct point_i { int x, y; };     // basic raw form, minimalist mode
struct point_i { int x, y;      // whenever possible, work with point_i
  point_i() { x = y = 0; }                        // default constructor
  point_i(int _x, int _y) : x(_x), y(_y) {} };           // user-defined

struct point { double x, y;   // only used if more precision is needed
  point() { x = y = 0.0; }                        // default constructor
  point(double _x, double _y) : x(_x), y(_y) {}          // user-defined
  bool operator < (point other) const { // override less than operator
    if (fabs(x - other.x) > EPS)                  // useful for sorting
      return x < other.x;          // first criteria , by x-coordinate
    return y < other.y; }          // second criteria, by y-coordinate
  // use EPS (1e-9) when testing equality of two floating points
  bool operator == (point other) const {
   return (fabs(x - other.x) < EPS && (fabs(y - other.y) < EPS)); } };

double dist(point p1, point p2) {                  // Euclidean distance
                      // hypot(dx, dy) returns sqrt(dx * dx + dy * dy)
  return hypot(p1.x - p2.x, p1.y - p2.y); }             // return double

// rotate p by theta degrees CCW w.r.t origin (0, 0)
point rotate(point p, double theta) {
  double rad = DEG_to_RAD(theta);    // multiply theta with PI / 180.0
  return point(p.x * cos(rad) - p.y * sin(rad),
               p.x * sin(rad) + p.y * cos(rad)); }

struct line { double a, b, c; };          // a way to represent a line

// the answer is stored in the third parameter (pass by reference)
void pointsToLine(point p1, point p2, line &l) {
  if (fabs(p1.x - p2.x) < EPS) {              // vertical line is fine
    l.a = 1.0;   l.b = 0.0;   l.c = -p1.x;           // default values
  } else {
    l.a = -(double)(p1.y - p2.y) / (p1.x - p2.x);
    l.b = 1.0;              // IMPORTANT: we fix the value of b to 1.0
    l.c = -(double)(l.a * p1.x) - p1.y;
} }

// not needed since we will use the more robust form: ax + by + c = 0 (see above)
struct line2 { double m, c; };       // another way to represent a line

int pointsToLine2(point p1, point p2, line2 &l) {
 if (abs(p1.x - p2.x) < EPS) {            // special case: vertical line
   l.m = INF;                      // l contains m = INF and c = x_value
   l.c = p1.x;                    // to denote vertical line x = x_value
   return 0;    // we need this return variable to differentiate result
 }
 else {
   l.m = (double)(p1.y - p2.y) / (p1.x - p2.x);
   l.c = p1.y - l.m * p1.x;
   return 1;     // l contains m and c of the line equation y = mx + c
```

```
} }

bool areParallel(line l1, line l2) {       // check coefficients a & b
  return (fabs(l1.a-l2.a) < EPS) && (fabs(l1.b-l2.b) < EPS); }

bool areSame(line l1, line l2) {           // also check coefficient c
  return areParallel(l1 ,l2) && (fabs(l1.c - l2.c) < EPS); }

// returns true (+ intersection point) if two lines are intersect
bool areIntersect(line l1, line l2, point &p) {
  if (areParallel(l1, l2)) return false;            // no intersection
  // solve system of 2 linear algebraic equations with 2 unknowns
  p.x = (l2.b * l1.c - l1.b * l2.c) / (l2.a * l1.b - l1.a * l2.b);
  // special case: test for vertical line to avoid division by zero
  if (fabs(l1.b) > EPS) p.y = -(l1.a * p.x + l1.c);
  else                  p.y = -(l2.a * p.x + l2.c);
  return true; }

struct vec { double x, y;  // name: 'vec' is different from STL vector
  vec(double _x, double _y) : x(_x), y(_y) {} };

vec toVec(point a, point b) {       // convert 2 points to vector a->b
  return vec(b.x - a.x, b.y - a.y); }

vec scale(vec v, double s) {        // nonnegative s = [<1 .. 1 .. >1]
  return vec(v.x * s, v.y * s); }               // shorter.same.longer

point translate(point p, vec v) {        // translate p according to v
  return point(p.x + v.x , p.y + v.y); }

// convert point and gradient/slope to line
void pointSlopeToLine(point p, double m, line &l) {
  l.a = -m;                                                // always -m
  l.b = 1;                                                  // always 1
  l.c = -((l.a * p.x) + (l.b * p.y)); }                 // compute this

void closestPoint(line l, point p, point &ans) {
  line perpendicular;         // perpendicular to l and pass through p
  if (fabs(l.b) < EPS) {              // special case 1: vertical line
    ans.x = -(l.c);   ans.y = p.y;      return; }

  if (fabs(l.a) < EPS) {           // special case 2: horizontal line
    ans.x = p.x;      ans.y = -(l.c);   return; }

  pointSlopeToLine(p, 1 / l.a, perpendicular);          // normal line
  // intersect line l with this perpendicular line
  // the intersection point is the closest point
  areIntersect(l, perpendicular, ans); }

// returns the reflection of point on a line
void reflectionPoint(line l, point p, point &ans) {
  point b;
  closestPoint(l, p, b);                     // similar to distToLine
  vec v = toVec(p, b);                             // create a vector
  ans = translate(translate(p, v), v); }         // translate p twice

double dot(vec a, vec b) { return (a.x * b.x + a.y * b.y); }

double norm_sq(vec v) { return v.x * v.x + v.y * v.y; }

// returns the distance from p to the line defined by
// two points a and b (a and b must be different)
// the closest point is stored in the 4th parameter (byref)
double distToLine(point p, point a, point b, point &c) {
  // formula: c = a + u * ab
  vec ap = toVec(a, p), ab = toVec(a, b);
  double u = dot(ap, ab) / norm_sq(ab);
  c = translate(a, scale(ab, u));                   // translate a to c
  return dist(p, c); }           // Euclidean distance between p and c

```

```cpp
// returns the distance from p to the line segment ab defined by
// two points a and b (still OK if a == b)
// the closest point is stored in the 4th parameter (byref)
double distToLineSegment(point p, point a, point b, point &c) {
  vec ap = toVec(a, p), ab = toVec(a, b);
  double u = dot(ap, ab) / norm_sq(ab);
  if (u < 0.0) { c = point(a.x, a.y);                       // closer to a
    return dist(p, a); }          // Euclidean distance between p and a
  if (u > 1.0) { c = point(b.x, b.y);                       // closer to b
    return dist(p, b); }          // Euclidean distance between p and b
  return distToLine(p, a, b, c); }           // run distToLine as above

double angle(point a, point o, point b) {  // returns angle aob in rad
  vec oa = toVec(o, a), ob = toVec(o, b);
  return acos(dot(oa, ob) / sqrt(norm_sq(oa) * norm_sq(ob))); }

double cross(vec a, vec b) { return a.x * b.y - a.y * b.x; }

//// another variant: returns 'twice' the area of this triangle A-B-c
//int area2(point p, point q, point r) {
//  return p.x * q.y - p.y * q.x +
//         q.x * r.y - q.y * r.x +
//         r.x * p.y - r.y * p.x;
//}

// note: to accept collinear points, we have to change the '> 0'
// returns true if point r is on the left side of line pq
bool ccw(point p, point q, point r) {
  return cross(toVec(p, q), toVec(p, r)) > 0; }

// returns true if point r is on the same line as the line pq
bool collinear(point p, point q, point r) {
  return fabs(cross(toVec(p, q), toVec(p, r))) < EPS; }

int main() {
  point P1, P2, P3(0, 1); // note that both P1 and P2 are (0.00, 0.00)
  printf("%d\n", P1 == P2);                                     // true
  printf("%d\n", P1 == P3);                                    // false

  vector<point> P;
  P.push_back(point(2, 2));
  P.push_back(point(4, 3));
  P.push_back(point(2, 4));
  P.push_back(point(6, 6));
  P.push_back(point(2, 6));
  P.push_back(point(6, 5));

  // sorting points demo
  sort(P.begin(), P.end());
  for (int i = 0; i < (int)P.size(); i++)
    printf("(%.2lf, %.2lf)\n", P[i].x, P[i].y);

  // rearrange the points as shown in the diagram below
  P.clear();
  P.push_back(point(2, 2));
  P.push_back(point(4, 3));
  P.push_back(point(2, 4));
  P.push_back(point(6, 6));
  P.push_back(point(2, 6));
  P.push_back(point(6, 5));
  P.push_back(point(8, 6));

  /*
  // the positions of these 7 points (0-based indexing)
  6   P4      P3  P6
  5           P5
  4   P2
  3       P1
  2   P0
  1
```

```
  0 1 2 3 4 5 6 7 8
  */

  double d = dist(P[0], P[5]);
  printf("Euclidean distance between P[0] and P[5] = %.2lf\n", d);
  // should be 5.000

  // line equations
  line l1, l2, l3, l4;
  pointsToLine(P[0], P[1], l1);
  printf("%.2lf * x + %.2lf * y + %.2lf = 0.00\n", l1.a, l1.b, l1.c);
  // should be -0.50 * x + 1.00 * y - 1.00 = 0.00

  pointsToLine(P[0], P[2], l2);
  // a vertical line, not a problem in "ax + by + c = 0" representation
  printf("%.2lf * x + %.2lf * y + %.2lf = 0.00\n", l2.a, l2.b, l2.c);
  // should be 1.00 * x + 0.00 * y - 2.00 = 0.00

  // parallel, same, and line intersection tests
  pointsToLine(P[2], P[3], l3);
  printf("l1 & l2 are parallel? %d\n", areParallel(l1, l2)); // no
  printf("l1 & l3 are parallel? %d\n", areParallel(l1, l3)); // yes,
                                   // l1 (P[0]-P[1]) and l3 (P[2]-P[3]) are parallel
  pointsToLine(P[2], P[4], l4);
  printf("l1 & l2 are the same? %d\n", areSame(l1, l2)); // no
  printf("l2 & l4 are the same? %d\n", areSame(l2, l4)); // yes,
                              // l2 (P[0]-P[2]) and l4 (P[2]-P[4]) are the same line
                       // (note, they are two different line segments, but same line)
  point p12;
  bool res = areIntersect(l1, l2, p12);
  // yes, l1 (P[0]-P[1]) and l2 (P[0]-P[2]) are intersect at (2.0, 2.0)
  printf("l1 & l2 are intersect? %d, at (%.2lf, %.2lf)\n", res, p12.x, p12.y);

  // other distances
  point ans;
  d = distToLine(P[0], P[2], P[3], ans);
  printf("Closest point from P[0] to line         (P[2]-P[3]): (%.2lf, %.2lf), dist = %.2lf\n", ans.x, ans.y, d);
  closestPoint(l3, P[0], ans);
  printf("Closest point from P[0] to line V2      (P[2]-P[3]): (%.2lf, %.2lf), dist = %.2lf\n", ans.x, ans.y, dist(P[0], ans));

  d = distToLineSegment(P[0], P[2], P[3], ans);
  printf("Closest point from P[0] to line SEGMENT (P[2]-P[3]): (%.2lf, %.2lf), dist = %.2lf\n", ans.x, ans.y, d); // closer to A (or P[2]) = (2.00, 4.00)
  d = distToLineSegment(P[1], P[2], P[3], ans);
  printf("Closest point from P[1] to line SEGMENT (P[2]-P[3]): (%.2lf, %.2lf), dist = %.2lf\n", ans.x, ans.y, d); // closer to midway between AB = (3.20, 4.60)
  d = distToLineSegment(P[6], P[2], P[3], ans);
  printf("Closest point from P[6] to line SEGMENT (P[2]-P[3]): (%.2lf, %.2lf), dist = %.2lf\n", ans.x, ans.y, d); // closer to B (or P[3]) = (6.00, 6.00)

  reflectionPoint(l4, P[1], ans);
  printf("Reflection point from P[1] to line      (P[2]-P[4]): (%.2lf, %.2lf)\n", ans.x, ans.y); // should be (0.00, 3.00)

  printf("Angle P[0]-P[4]-P[3] = %.2lf\n", RAD_to_DEG(angle(P[0], P[4], P[3]))); // 90 degrees
  printf("Angle P[0]-P[2]-P[1] = %.2lf\n", RAD_to_DEG(angle(P[0], P[2], P[1]))); // 63.43 degrees
  printf("Angle P[4]-P[3]-P[6] = %.2lf\n", RAD_to_DEG(angle(P[4], P[3], P[6]))); // 180 degrees

  printf("P[0], P[2], P[3] form A left turn? %d\n", ccw(P[0], P[2], P[3])); // no
  printf("P[0], P[3], P[2] form A left turn? %d\n", ccw(P[0], P[3], P[2])); // yes

  printf("P[0], P[2], P[3] are collinear? %d\n", collinear(P[0], P[2], P[3])); // no
  printf("P[0], P[2], P[4] are collinear? %d\n", collinear(P[0], P[2], P[4])); //
```

```
yes

  point p(3, 7), q(11, 13), r(35, 30); // collinear if r(35, 31)
  printf("r is on the %s of line p-r\n", ccw(p, q, r) ? "left" : "right"); //right

  /*
  // the positions of these 6 points
     E<--  4
           3      B D<--
           2   A C
           1
  -4-3-2-1 0 1 2 3 4 5 6
          -1
          -2
   F<--   -3
  */

  // translation
  point A(2.0, 2.0);
  point B(4.0, 3.0);
  vec v = toVec(A, B); // imagine there is an arrow from A to B
  point C(3.0, 2.0);                 // (see the diagram above)
  point D = translate(C, v);
  // D will be located in coordinate (3.0 + 2.0, 2.0 + 1.0) = (5.0, 3.0)
  printf("D = (%.2lf, %.2lf)\n", D.x, D.y);
  point E = translate(C, scale(v, 0.5));
  // E will be located in coordinate (3.0 + 1/2 * 2.0, 2.0 + 1/2 * 1.0) =
  printf("E = (%.2lf, %.2lf)\n", E.x, E.y);                 // (4.0, 2.5)

  // rotation
  printf("B = (%.2lf, %.2lf)\n", B.x, B.y); // B = (4.0, 3.0)
  point F = rotate(B, 90); // rotate B by 90 degrees COUNTER clockwise,
  printf("F = (%.2lf, %.2lf)\n", F.x, F.y);          // F = (-3.0, 4.0)
  point G = rotate(B, 180); // rotate B by 180 degrees COUNTER clockwise,
  printf("G = (%.2lf, %.2lf)\n", G.x, G.y);          // G = (-4.0, -3.0)

  return 0;
}
```

```cpp
/**********************************************************************************/
/** Circulos ..................................................................... */
/**********************************************************************************/

#include <cstdio>
#include <cmath>
using namespace std;

#define INF 1e9
#define EPS 1e-9
#define PI acos(-1.0)

double DEG_to_RAD(double d) { return d * PI / 180.0; }

double RAD_to_DEG(double r) { return r * 180.0 / PI; }

struct point_i { int x, y;      // whenever possible, work with point_i
  point_i() { x = y = 0; }                         // default constructor
  point_i(int _x, int _y) : x(_x), y(_y) {} };             // constructor

struct point { double x, y;   // only used if more precision is needed
  point() { x = y = 0.0; }                         // default constructor
  point(double _x, double _y) : x(_x), y(_y) {} };         // constructor

int insideCircle(point_i p, point_i c, int r) { // all integer version
  int dx = p.x - c.x, dy = p.y - c.y;
  int Euc = dx * dx + dy * dy, rSq = r * r;              // all integer
  return Euc < rSq ? 0 : Euc == rSq ? 1 : 2; } //inside/border/outside

bool circle2PtsRad(point p1, point p2, double r, point &c) {
  double d2 = (p1.x - p2.x) * (p1.x - p2.x) +
              (p1.y - p2.y) * (p1.y - p2.y);
  double det = r * r / d2 - 0.25;
  if (det < 0.0) return false;
  double h = sqrt(det);
  c.x = (p1.x + p2.x) * 0.5 + (p1.y - p2.y) * h;
  c.y = (p1.y + p2.y) * 0.5 + (p2.x - p1.x) * h;
  return true; }          // to get the other center, reverse p1 and p2

int main() {
  // circle equation, inside, border, outside
  point_i pt(2, 2);
  int r = 7;
  point_i inside(8, 2);
  printf("%d\n", insideCircle(inside, pt, r));             // 0-inside
  point_i border(9, 2);
  printf("%d\n", insideCircle(border, pt, r));           // 1-at border
  point_i outside(10, 2);
  printf("%d\n", insideCircle(outside, pt, r));           // 2-outside

  double d = 2 * r;
  printf("Diameter = %.2lf\n", d);
  double c = PI * d;
  printf("Circumference (Perimeter) = %.2lf\n", c);
  double A = PI * r * r;
  printf("Area of circle = %.2lf\n", A);

  printf("Length of arc   (central angle = 60 degrees) = %.2lf\n",
    60.0 / 360.0 * c);
  printf("Length of chord (central angle = 60 degrees) = %.2lf\n",
    sqrt((2 * r * r) * (1 - cos(DEG_to_RAD(60.0)))));
  printf("Area of sector  (central angle = 60 degrees) = %.2lf\n",
    60.0 / 360.0 * A);

  point p1;
  point p2(0.0, -1.0);
  point ans;
  circle2PtsRad(p1, p2, 2.0, ans);
  printf("One of the center is (%.2lf, %.2lf)\n", ans.x, ans.y);
  circle2PtsRad(p2, p1, 2.0, ans);     // we simply reverse p1 with p2
```

```
  printf("The other center  is (%.2lf, %.2lf)\n", ans.x, ans.y);

  return 0;
}
```

```
/*******************************************************************************/
/** Triangulos ............................................................. */
/*******************************************************************************/

#include <cstdio>
#include <cmath>
using namespace std;

#define EPS 1e-9
#define PI acos(-1.0)

double DEG_to_RAD(double d) { return d * PI / 180.0; }

double RAD_to_DEG(double r) { return r * 180.0 / PI; }

struct point_i { int x, y;      // whenever possible, work with point_i
  point_i() { x = y = 0; }                       // default constructor
  point_i(int _x, int _y) : x(_x), y(_y) {} };           // constructor

struct point { double x, y;   // only used if more precision is needed
  point() { x = y = 0.0; }                       // default constructor
  point(double _x, double _y) : x(_x), y(_y) {} };       // constructor

double dist(point p1, point p2) {
  return hypot(p1.x - p2.x, p1.y - p2.y); }

double perimeter(double ab, double bc, double ca) {
  return ab + bc + ca; }

double perimeter(point a, point b, point c) {
  return dist(a, b) + dist(b, c) + dist(c, a); }

double area(double ab, double bc, double ca) {
  // Heron's formula, split sqrt(a * b) into sqrt(a) * sqrt(b); in implementation
  double s = 0.5 * perimeter(ab, bc, ca);
  return sqrt(s) * sqrt(s - ab) * sqrt(s - bc) * sqrt(s - ca); }

double area(point a, point b, point c) {
  return area(dist(a, b), dist(b, c), dist(c, a)); }

//====================================================================
// from ch7_01_points_lines
struct line { double a, b, c; }; // a way to represent a line

// the answer is stored in the third parameter (pass by reference)
void pointsToLine(point p1, point p2, line &l) {
  if (fabs(p1.x - p2.x) < EPS) {                 // vertical line is fine
    l.a = 1.0;   l.b = 0.0;   l.c = -p1.x;              // default values
  } else {
    l.a = -(double)(p1.y - p2.y) / (p1.x - p2.x);
    l.b = 1.0;                 // IMPORTANT: we fix the value of b to 1.0
    l.c = -(double)(l.a * p1.x) - p1.y;
} }

bool areParallel(line l1, line l2) {          // check coefficient a + b
  return (fabs(l1.a-l2.a) < EPS) && (fabs(l1.b-l2.b) < EPS); }

// returns true (+ intersection point) if two lines are intersect
bool areIntersect(line l1, line l2, point &p) {
  if (areParallel(l1, l2)) return false;              // no intersection
  // solve system of 2 linear algebraic equations with 2 unknowns
  p.x = (l2.b * l1.c - l1.b * l2.c) / (l2.a * l1.b - l1.a * l2.b);
  // special case: test for vertical line to avoid division by zero
  if (fabs(l1.b) > EPS) p.y = -(l1.a * p.x + l1.c);
  else                  p.y = -(l2.a * p.x + l2.c);
  return true; }

struct vec { double x, y;  // name: 'vec' is different from STL vector
  vec(double _x, double _y) : x(_x), y(_y) {} };

```

```
vec toVec(point a, point b) {       // convert 2 points to vector a->b
  return vec(b.x - a.x, b.y - a.y); }

vec scale(vec v, double s) {        // nonnegative s = [<1 .. 1 .. >1]
  return vec(v.x * s, v.y * s); }               // shorter.same.longer

point translate(point p, vec v) {        // translate p according to v
  return point(p.x + v.x , p.y + v.y); }
//====================================================================

double rInCircle(double ab, double bc, double ca) {
  return area(ab, bc, ca) / (0.5 * perimeter(ab, bc, ca)); }

double rInCircle(point a, point b, point c) {
  return rInCircle(dist(a, b), dist(b, c), dist(c, a)); }

// assumption: the required points/lines functions have been written
// returns 1 if there is an inCircle center, returns 0 otherwise
// if this function returns 1, ctr will be the inCircle center
// and r is the same as rInCircle
int inCircle(point p1, point p2, point p3, point &ctr, double &r) {
  r = rInCircle(p1, p2, p3);
  if (fabs(r) < EPS) return 0;                   // no inCircle center

  line l1, l2;                    // compute these two angle bisectors
  double ratio = dist(p1, p2) / dist(p1, p3);
  point p = translate(p2, scale(toVec(p2, p3), ratio / (1 + ratio)));
  pointsToLine(p1, p, l1);

  ratio = dist(p2, p1) / dist(p2, p3);
  p = translate(p1, scale(toVec(p1, p3), ratio / (1 + ratio)));
  pointsToLine(p2, p, l2);

  areIntersect(l1, l2, ctr);           // get their intersection point
  return 1; }

double rCircumCircle(double ab, double bc, double ca) {
  return ab * bc * ca / (4.0 * area(ab, bc, ca)); }

double rCircumCircle(point a, point b, point c) {
  return rCircumCircle(dist(a, b), dist(b, c), dist(c, a)); }

// assumption: the required points/lines functions have been written
// returns 1 if there is a circumCenter center, returns 0 otherwise
// if this function returns 1, ctr will be the circumCircle center
// and r is the same as rCircumCircle
int circumCircle(point p1, point p2, point p3, point &ctr, double &r){
  double a = p2.x - p1.x, b = p2.y - p1.y;
  double c = p3.x - p1.x, d = p3.y - p1.y;
  double e = a * (p1.x + p2.x) + b * (p1.y + p2.y);
  double f = c * (p1.x + p3.x) + d * (p1.y + p3.y);
  double g = 2.0 * (a * (p3.y - p2.y) - b * (p3.x - p2.x));
  if (fabs(g) < EPS) return 0;

  ctr.x = (d*e - b*f) / g;
  ctr.y = (a*f - c*e) / g;
  r = dist(p1, ctr);  // r = distance from center to 1 of the 3 points
  return 1; }

// returns true if point d is inside the circumCircle defined by a,b,c
int inCircumCircle(point a, point b, point c, point d) {
  return (a.x - d.x) * (b.y - d.y) * ((c.x - d.x) * (c.x - d.x) + (c.y - d.y) *
(c.y - d.y)) +
          (a.y - d.y) * ((b.x - d.x) * (b.x - d.x) + (b.y - d.y) * (b.y - d.y)) *
(c.x - d.x) +
          ((a.x - d.x) * (a.x - d.x) + (a.y - d.y) * (a.y - d.y)) * (b.x - d.x) *
(c.y - d.y) -
          ((a.x - d.x) * (a.x - d.x) + (a.y - d.y) * (a.y - d.y)) * (b.y - d.y) *
(c.x - d.x) -
          (a.y - d.y) * (b.x - d.x) * ((c.x - d.x) * (c.x - d.x) + (c.y - d.y) *
```

```
(c.y - d.y)) -
           (a.x - d.x) * ((b.x - d.x) * (b.x - d.x) + (b.y - d.y) * (b.y - d.y)) * (c.y - d.y) > 0 ? 1 : 0;
}

bool canFormTriangle(double a, double b, double c) {
  return (a + b > c) && (a + c > b) && (b + c > a); }

int main() {
  double base = 4.0, h = 3.0;
  double A = 0.5 * base * h;
  printf("Area = %.2lf\n", A);

  point a;                                          // a right triangle
  point b(4.0, 0.0);
  point c(4.0, 3.0);

  double p = perimeter(a, b, c);
  double s = 0.5 * p;
  A = area(a, b, c);
  printf("Area = %.2lf\n", A);             // must be the same as above

  double r = rInCircle(a, b, c);
  printf("R1 (radius of incircle) = %.2lf\n", r);            // 1.00
  point ctr;
  int res = inCircle(a, b, c, ctr, r);
  printf("R1 (radius of incircle) = %.2lf\n", r);        // same, 1.00
  printf("Center = (%.2lf, %.2lf)\n", ctr.x, ctr.y);   // (3.00, 1.00)

  printf("R2 (radius of circumcircle) = %.2lf\n", rCircumCircle(a, b, c)); // 2.50
  res = circumCircle(a, b, c, ctr, r);
  printf("R2 (radius of circumcircle) = %.2lf\n", r);   // same, 2.50
  printf("Center = (%.2lf, %.2lf)\n", ctr.x, ctr.y);   // (2.00, 1.50)

  point d(2.0, 1.0);                // inside triangle and circumCircle
  printf("d inside circumCircle (a, b, c) ? %d\n", inCircumCircle(a, b, c, d));
  point e(2.0, 3.9);    // outside the triangle but inside circumCircle
  printf("e inside circumCircle (a, b, c) ? %d\n", inCircumCircle(a, b, c, e));
  point f(2.0, -1.1);                               // slightly outside
  printf("f inside circumCircle (a, b, c) ? %d\n", inCircumCircle(a, b, c, f));

  // Law of Cosines
  double ab = dist(a, b);
  double bc = dist(b, c);
  double ca = dist(c, a);
  double alpha = RAD_to_DEG(acos((ca * ca + ab * ab - bc * bc) / (2.0 * ca * ab)));
  printf("alpha = %.2lf\n", alpha);
  double beta  = RAD_to_DEG(acos((ab * ab + bc * bc - ca * ca) / (2.0 * ab * bc)));
  printf("beta  = %.2lf\n", beta);
  double gamma = RAD_to_DEG(acos((bc * bc + ca * ca - ab * ab) / (2.0 * bc * ca)));
  printf("gamma = %.2lf\n", gamma);

  // Law of Sines
  printf("%.2lf == %.2lf == %.2lf\n", bc / sin(DEG_to_RAD(alpha)), ca / sin(DEG_to_RAD(beta)), ab / sin(DEG_to_RAD(gamma)));

  // Phytagorean Theorem
  printf("%.2lf^2 == %.2lf^2 + %.2lf^2\n", ca, ab, bc);

  // Triangle Inequality
  printf("(%d, %d, %d) => can form triangle? %d\n", 3, 4, 5, canFormTriangle(3, 4, 5)); // yes
  printf("(%d, %d, %d) => can form triangle? %d\n", 3, 4, 7, canFormTriangle(3, 4, 7)); // no, actually straight line
  printf("(%d, %d, %d) => can form triangle? %d\n", 3, 4, 8, canFormTriangle(3, 4, 8)); // no

```

```
    return 0;
}
```

```cpp
/*****************************************************************************/
/** Poligonos ............................................................. */
/*****************************************************************************/

#include <algorithm>
#include <cstdio>
#include <cmath>
#include <stack>
#include <vector>
using namespace std;

#define EPS 1e-9
#define PI acos(-1.0)

double DEG_to_RAD(double d) { return d * PI / 180.0; }

double RAD_to_DEG(double r) { return r * 180.0 / PI; }

struct point { double x, y;   // only used if more precision is needed
  point() { x = y = 0.0; }                      // default constructor
  point(double _x, double _y) : x(_x), y(_y) {}        // user-defined
  bool operator == (point other) const {
   return (fabs(x - other.x) < EPS && (fabs(y - other.y) < EPS)); } };

struct vec { double x, y;  // name: 'vec' is different from STL vector
  vec(double _x, double _y) : x(_x), y(_y) {} };

vec toVec(point a, point b) {       // convert 2 points to vector a->b
  return vec(b.x - a.x, b.y - a.y); }

double dist(point p1, point p2) {                // Euclidean distance
  return hypot(p1.x - p2.x, p1.y - p2.y); }               // return double

// returns the perimeter, which is the sum of Euclidian distances
// of consecutive line segments (polygon edges)
double perimeter(const vector<point> &P) {
  double result = 0.0;
  for (int i = 0; i < (int)P.size()-1; i++)  // remember that P[0] = P[n-1]
    result += dist(P[i], P[i+1]);
  return result; }

// returns the area, which is half the determinant
double area(const vector<point> &P) {
  double result = 0.0, x1, y1, x2, y2;
  for (int i = 0; i < (int)P.size()-1; i++) {
    x1 = P[i].x; x2 = P[i+1].x;
    y1 = P[i].y; y2 = P[i+1].y;
    result += (x1 * y2 - x2 * y1);
  }
  return fabs(result) / 2.0; }

double dot(vec a, vec b) { return (a.x * b.x + a.y * b.y); }

double norm_sq(vec v) { return v.x * v.x + v.y * v.y; }

double angle(point a, point o, point b) {  // returns angle aob in rad
  vec oa = toVec(o, a), ob = toVec(o, b);
  return acos(dot(oa, ob) / sqrt(norm_sq(oa) * norm_sq(ob))); }

double cross(vec a, vec b) { return a.x * b.y - a.y * b.x; }

// note: to accept collinear points, we have to change the '> 0'
// returns true if point r is on the left side of line pq
bool ccw(point p, point q, point r) {
  return cross(toVec(p, q), toVec(p, r)) > 0; }

// returns true if point r is on the same line as the line pq
bool collinear(point p, point q, point r) {
  return fabs(cross(toVec(p, q), toVec(p, r))) < EPS; }

```

```
// returns true if we always make the same turn while examining
// all the edges of the polygon one by one
bool isConvex(const vector<point> &P) {
  int sz = (int)P.size();
  if (sz <= 3) return false;   // a point/sz=2 or a line/sz=3 is not convex
  bool isLeft = ccw(P[0], P[1], P[2]);               // remember one result
  for (int i = 1; i < sz-1; i++)            // then compare with the others
    if (ccw(P[i], P[i+1], P[(i+2) == sz ? 1 : i+2]) != isLeft)
      return false;            // different sign -> this polygon is concave
  return true; }                                  // this polygon is convex

// returns true if point p is in either convex/concave polygon P
bool inPolygon(point pt, const vector<point> &P) {
  if ((int)P.size() == 0) return false;
  double sum = 0;    // assume the first vertex is equal to the last vertex
  for (int i = 0; i < (int)P.size()-1; i++) {
    if (ccw(pt, P[i], P[i+1]))
         sum += angle(P[i], pt, P[i+1]);                   // left turn/ccw
    else sum -= angle(P[i], pt, P[i+1]); }                 // right turn/cw
  return fabs(fabs(sum) - 2*PI) < EPS; }

// line segment p-q intersect with line A-B.
point lineIntersectSeg(point p, point q, point A, point B) {
  double a = B.y - A.y;
  double b = A.x - B.x;
  double c = B.x * A.y - A.x * B.y;
  double u = fabs(a * p.x + b * p.y + c);
  double v = fabs(a * q.x + b * q.y + c);
  return point((p.x * v + q.x * u) / (u+v), (p.y * v + q.y * u) / (u+v)); }

// cuts polygon Q along the line formed by point a -> point b
// (note: the last point must be the same as the first point)
vector<point> cutPolygon(point a, point b, const vector<point> &Q) {
  vector<point> P;
  for (int i = 0; i < (int)Q.size(); i++) {
    double left1 = cross(toVec(a, b), toVec(a, Q[i])), left2 = 0;
    if (i != (int)Q.size()-1) left2 = cross(toVec(a, b), toVec(a, Q[i+1]));
    if (left1 > -EPS) P.push_back(Q[i]);       // Q[i] is on the left of ab
    if (left1 * left2 < -EPS)         // edge (Q[i], Q[i+1]) crosses line ab
      P.push_back(lineIntersectSeg(Q[i], Q[i+1], a, b));
  }
  if (!P.empty() && !(P.back() == P.front()))
    P.push_back(P.front());          // make P's first point = P's last point
  return P; }

point pivot;
bool angleCmp(point a, point b) {                 // angle-sorting function
  if (collinear(pivot, a, b))                               // special case
    return dist(pivot, a) < dist(pivot, b);    // check which one is closer
  double d1x = a.x - pivot.x, d1y = a.y - pivot.y;
  double d2x = b.x - pivot.x, d2y = b.y - pivot.y;
  return (atan2(d1y, d1x) - atan2(d2y, d2x)) < 0; }   // compare two angles

/*****************************************************************************/
/** Fecho Convexo ************************************************************/
/*****************************************************************************/

vector<point> CH(vector<point> P) {   // the content of P may be reshuffled
  int i, j, n = (int)P.size();
  if (n <= 3) {
    if (!(P[0] == P[n-1])) P.push_back(P[0]); // safeguard from corner case
    return P;                           // special case, the CH is P itself
  }

  // first, find P0 = point with lowest Y and if tie: rightmost X
  int P0 = 0;
  for (i = 1; i < n; i++)
    if (P[i].y < P[P0].y || (P[i].y == P[P0].y && P[i].x > P[P0].x))
      P0 = i;

```

```
  point temp = P[0]; P[0] = P[P0]; P[P0] = temp;    // swap P[P0] with P[0]

  // second, sort points by angle w.r.t. pivot P0
  pivot = P[0];                      // use this global variable as reference
  sort(++P.begin(), P.end(), angleCmp);                 // we do not sort P[0]

  // third, the ccw tests
  vector<point> S;
  S.push_back(P[n-1]); S.push_back(P[0]); S.push_back(P[1]);   // initial S
  i = 2;                                              // then, we check the rest
  while (i < n) {            // note: N must be >= 3 for this method to work
    j = (int)S.size()-1;
    if (ccw(S[j-1], S[j], P[i])) S.push_back(P[i++]);  // left turn, accept
    else S.pop_back(); }   // or pop the top of S until we have a left turn
  return S; }                                             // return the result

int main() {
  // 6 points, entered in counter clockwise order, 0-based indexing
  vector<point> P;
  P.push_back(point(1, 1));
  P.push_back(point(3, 3));
  P.push_back(point(9, 1));
  P.push_back(point(12, 4));
  P.push_back(point(9, 7));
  P.push_back(point(1, 7));
  P.push_back(P[0]); // loop back

  printf("Perimeter of polygon = %.2lf\n", perimeter(P)); // 31.64
  printf("Area of polygon = %.2lf\n", area(P)); // 49.00
  printf("Is convex = %d\n", isConvex(P)); // false (P1 is the culprit)

  //// the positions of P6 and P7 w.r.t the polygon
  //7 P5--------------P4
  //6 |                  \
  //5 |                    \
  //4 |   P7                 P3
  //3 |   P1___            /
  //2 | / P6    \ ___     /
  //1 P0              P2
  //0 1 2 3 4 5 6 7 8 9 101112

  point P6(3, 2); // outside this (concave) polygon
  printf("Point P6 is inside this polygon = %d\n", inPolygon(P6, P)); // false
  point P7(3, 4); // inside this (concave) polygon
  printf("Point P7 is inside this polygon = %d\n", inPolygon(P7, P)); // true

  // cutting the original polygon based on line P[2] -> P[4] (get the left side)
  //7 P5--------------P4
  //6 |               |  \
  //5 |               |    \
  //4 |               |     P3
  //3 |   P1___       |    /
  //2 | /       \ ___ |  /
  //1 P0              P2
  //0 1 2 3 4 5 6 7 8 9 101112
  // new polygon (notice the index are different now):
  //7 P4--------------P3
  //6 |               |
  //5 |               |
  //4 |               |
  //3 |   P1___       |
  //2 | /       \ ___ |
  //1 P0              P2
  //0 1 2 3 4 5 6 7 8 9

  P = cutPolygon(P[2], P[4], P);
  printf("Perimeter of polygon = %.2lf\n", perimeter(P)); // smaller now 29.15
  printf("Area of polygon = %.2lf\n", area(P)); // 40.00

  // running convex hull of the resulting polygon (index changes again)
```

```
  //7 P3--------------P2
  //6 |               |
  //5 |               |
  //4 |   P7          |
  //3 |               |
  //2 |               |
  //1 P0--------------P1
  //0 1 2 3 4 5 6 7 8 9

  P = CH(P); // now this is a rectangle
  printf("Perimeter of polygon = %.2lf\n", perimeter(P)); // precisely 28.00
  printf("Area of polygon = %.2lf\n", area(P)); // precisely 48.00
  printf("Is convex = %d\n", isConvex(P)); // true
  printf("Point P6 is inside this polygon = %d\n", inPolygon(P6, P)); // true
  printf("Point P7 is inside this polygon = %d\n", inPolygon(P7, P)); // true

  return 0;
}
```

```cpp
/*****************************************************************************/
/** 15-Puzzle Problem with IDA* ............................................ */
/*****************************************************************************/

#include <algorithm>
#include <cstdio>
#include <map>
using namespace std;

#define INF 1000000000
#define ROW_SIZE 4 // ROW_SIZE is a matrix of 4 x 4
#define PUZZLE (ROW_SIZE*ROW_SIZE)
#define X 15

int p[PUZZLE];
int lim, nlim;
int dr[] = { 0,-1, 0, 1}; // E,N,W,S
int dc[] = { 1, 0,-1, 0}; // R,U,L,D
map<int, int> pred;
map<unsigned long long, int> vis;
char ans[] = "RULD";

inline int h1() { // heuristic: sum of Manhattan distances (compute all)
  int ans = 0;
  for (int i = 0; i < PUZZLE; i++) {
    int tgt_i = p[i] / 4, tgt_j = p[i] % 4;
    if (p[i] != X)
      ans += abs(i / 4 - tgt_i) + abs(i % 4 - tgt_j); // Manhattan distance
  }
  return ans;
}
// heuristic: sum of manhattan distances (compute delta)
inline int h2(int i1, int j1, int i2, int j2) {
  int tgt_i = p[i2 * 4 + j2] / 4, tgt_j = p[i2 * 4 + j2] % 4;
  return -(abs(i2 - tgt_i) + abs(j2 - tgt_j)) +
    (abs(i1 - tgt_i) + abs(j1 - tgt_j));
}

inline bool goal() {
  for (int i = 0; i < PUZZLE; i++)
    if (p[i] != X && p[i] != i)
      return false;
  return true;
}

inline bool valid(int r, int c) {
  return 0 <= r && r < 4 && 0 <= c && c < 4;
}

inline void swap(int i, int j, int new_i, int new_j) {
  int temp = p[i * 4 + j];
  p[i * 4 + j] = p[new_i * 4 + new_j];
  p[new_i * 4 + new_j] = temp;
}

bool DFS(int g, int h) {
  if (g + h > lim) {
    nlim = min(nlim, g + h);
    return false;
  }

  if (goal())
    return true;

  unsigned long long state = 0;
  // transform 16 numbers into 64 bits, exactly into ULL
  for (int i = 0; i < PUZZLE; i++) {
    state <<= 4; // move left 4 bits
    state += p[i]; // add this digit (max 15 or 1111)
  }
```

```

  // not pure backtracking... this is to prevent cycling
  if (vis.count(state) && vis[state] <= g)
    return false; // not good
  vis[state] = g; // mark this as visited

  int i, j, d, new_i, new_j;
  for (i = 0; i < PUZZLE; i++)
    if (p[i] == X)
      break;
  j = i % 4;
  i /= 4;

  for (d = 0; d < 4; d++) {
    new_i = i + dr[d]; new_j = j + dc[d];
    if (valid(new_i, new_j)) {
      int dh = h2(i, j, new_i, new_j);
      swap(i, j, new_i, new_j); // swap first
      pred[g + 1] = d;
      if (DFS(g + 1, h + dh)) // if ok, no need to restore, just go ahead
        return true;
      swap(i, j, new_i, new_j); // restore
    }
  }

  return false;
}

int IDA_Star() {
  lim = h1();
  while (true) {
    nlim = INF; // next limit
    pred.clear();
    vis.clear();
    if (DFS(0, h1()))
      return lim;
    if (nlim == INF)
      return -1;
    lim = nlim; // nlim > lim
    if (lim > 45) // pruning condition in the problem
      return -1;
  }
}

void output(int d) {
  if (d == 0)
    return;
  output(d - 1);
  printf("%c", ans[pred[d]]);
}

int main() {
#ifndef ONLINE_JUDGE
  freopen("in.txt", "r", stdin);
#endif

  int N;
  scanf("%d", &N);
  while (N--) {
    int i, j, blank = 0, sum = 0, ans = 0;
    for (i = 0; i < 4; i++)
      for (j = 0; j < 4; j++) {
        scanf("%d", &p[i * 4 + j]);
        if (p[i * 4 + j] == 0) {
          p[i * 4 + j] = X; // change to X (15)
          blank = i * 4 + j; // remember the index
        }
        else
          p[i * 4 + j]--; // use 0-based indexing
      }
```

```

    for (i = 0; i < PUZZLE; i++)
      for (j = 0; j < i; j++)
        if (p[i] != X && p[j] != X && p[j] > p[i])
          sum++;
    sum += blank / ROW_SIZE;

    if (sum % 2 != 0 && ((ans = IDA_Star()) != -1))
      output(ans), printf("\n");
    else
      printf("This puzzle is not solvable.\n");
  }

  return 0;
}
```

```cpp
/*******************************************************************************/
/** Prog. Dinamica com Bitmask ................................................ */
/*******************************************************************************/
// Forming Quiz Teams

#include <algorithm>            // if you have problems with this C++ code,
#include <cmath>            // consult your programming text books first...
#include <cstdio>
#include <cstring>
using namespace std;
        /* Forming Quiz Teams, the solution for UVa 10911 above */
          // using global variables is a bad software engineering practice,
int N, target;                  // but it is OK for competitive programming
double dist[20][20], memo[1 << 16];  // 1 << 16 = 2^16, note that max N = 8

double matching(int bitmask) {                        // DP state = bitmask
                       // we initialize 'memo' with -1 in the main function
  if (memo[bitmask] > -0.5)          // this state has been computed before
    return memo[bitmask];                   // simply lookup the memo table
  if (bitmask == target)                // all students are already matched
    return memo[bitmask] = 0;                              // the cost is 0

  double ans = 2000000000.0;               // initialize with a large value
  int p1, p2;
  for (p1 = 0; p1 < 2 * N; p1++)
    if (!(bitmask & (1 << p1)))
      break;                              // find the first bit that is off
  for (p2 = p1 + 1; p2 < 2 * N; p2++)              // then, try to match p1
    if (!(bitmask & (1 << p2)))     // with another bit p2 that is also off
      ans = min(ans,                                    // pick the minimum
                dist[p1][p2] + matching(bitmask | (1 << p1) | (1 << p2)));

  return memo[bitmask] = ans;    // store result in a memo table and return
}

int main() {
  int i, j, caseNo = 1, x[20], y[20];
  // freopen("10911.txt", "r", stdin);      // redirect input file to stdin

  while (scanf("%d", &N), N) {                    // yes, we can do this :)
    for (i = 0; i < 2 * N; i++)
      scanf("%*s %d %d", &x[i], &y[i]);                // '%*s' skips names
    for (i = 0; i < 2 * N - 1; i++)        // build pairwise distance table
      for (j = i + 1; j < 2 * N; j++)      // have you used 'hypot' before?
        dist[i][j] = dist[j][i] = hypot(x[i] - x[j], y[i] - y[j]);

    // use DP to solve min weighted perfect matching on small general graph
    for (i = 0; i < (1 << 16); i++) memo[i] = -1.0;  // set -1 to all cells
    target = (1 << (2 * N)) - 1;
    printf("Case %d: %.2lf\n", caseNo++, matching(0));
} } // return 0;
```

```cpp
/*****************************************************************************/
/** Prog. Dinamica (outro exemplo) ...................................... */
/*****************************************************************************/
// ACORN, UVa 1231, LA 4106

#include <algorithm>
#include <cstdio>
#include <cstring>
using namespace std;

int main() {
  int i, j, c, t, h, f, a, n, acorn[2010][2010], dp[2010];

  scanf("%d", &c);
  while (c--) {
    scanf("%d %d %d", &t, &h, &f);
    memset(acorn, 0, sizeof acorn);
    for (i = 0; i < t; i++) {
      scanf("%d", &a);
      for (j = 0; j < a; j++) {
        scanf("%d", &n);
        acorn[i][n]++; // there is an acorn here
      }
    }

    for (int tree = 0; tree < t; tree++) // initialization
      dp[h] = max(dp[h], acorn[tree][h]);
    for (int height = h - 1; height >= 0; height--)
      for (int tree = 0; tree < t; tree++) {
        acorn[tree][height] +=
          max(acorn[tree][height + 1], // from this tree, +1 above
          ((height + f <= h) ? dp[height + f] : 0));
          // best from tree at height + f
        dp[height] = max(dp[height], acorn[tree][height]); // update this too
      }
    printf("%d\n", dp[0]); // solution will be here
  }
  // ignore the last number 0

  return 0;
}
```

```cpp
/*****************************************************************************/
/** Outras tecnicas ........................................................ */
/*****************************************************************************/
// World Finals Stockholm 2009, A - A Careful Approach, UVa 1079, LA 4445

#include <algorithm>
#include <cmath>
#include <cstdio>
using namespace std;

int i, n, caseNo = 1, order[8];
double a[8], b[8], L, maxL;

double greedyLanding() {  // with certain landing order, and certain L, try
         // landing those planes and see what is the gap to b[order[n - 1]]
  double lastLanding = a[order[0]];      // greedy, 1st aircraft lands ASAP
  for (i = 1; i < n; i++) {                      // for the other aircrafts
    double targetLandingTime = lastLanding + L;
    if (targetLandingTime <= b[order[i]])
       // can land: greedily choose max of a[order[i]] or targetLandingTime
      lastLanding = max(a[order[i]], targetLandingTime);
    else
      return 1;
  }
  // return +ve value to force binary search to reduce L
  // return -ve value to force binary search to increase L
  return lastLanding - b[order[n - 1]];
}

int main() {
  while (scanf("%d", &n), n) {                               // 2 <= n <= 8
    for (i = 0; i < n; i++) {    // plane i land safely at interval [ai, bi]
      scanf("%lf %lf", &a[i], &b[i]);
      a[i] *= 60; b[i] *= 60;  // originally in minutes, convert to seconds
      order[i] = i;
    }

    maxL = -1.0;                                 // variable to be searched for
    do {                               // permute plane landing order, up to 8!
      double lo = 0, hi = 86400;                 // min 0s, max 1 day = 86400s
      L = -1;                               // start with an infeasible solution
      while (fabs(lo - hi) >= 1e-3) {           // binary search L, EPS = 1e-3
        L = (lo + hi) / 2.0;    // we want the answer rounded to nearest int
        double retVal = greedyLanding();                   // round down first
        if (retVal <= 1e-2) lo = L;                         // must increase L
        else                hi = L;           // infeasible, must decrease L
      }
      maxL = max(maxL, L);                 // get the max over all permutations
    }
    while (next_permutation(order, order + n));      // try all permutations

    // other way for rounding is to use printf format string: %.0lf:%0.2lf
    maxL = (int)(maxL + 0.5);                         // round to nearest second
    printf("Case %d: %d:%0.2d\n", caseNo++, (int)(maxL/60), (int)maxL%60);
  }

  return 0;
}
```

```cpp
/*****************************************************************************/
/** Eliminacao Gaussiana ...................................................... */
/*****************************************************************************/

#include <cmath>
#include <cstdio>
using namespace std;

#define MAX_N 3                                 // adjust this value as needed
struct AugmentedMatrix { double mat[MAX_N][MAX_N + 1]; };
struct ColumnVector { double vec[MAX_N]; };

ColumnVector GaussianElimination(int N, AugmentedMatrix Aug) {
  // input: N, Augmented Matrix Aug, output: Column vector X, the answer
  int i, j, k, l; double t;

  for (i = 0; i < N - 1; i++) {              // the forward elimination phase
    l = i;
    for (j = i + 1; j < N; j++)          // which row has largest column value
      if (fabs(Aug.mat[j][i]) > fabs(Aug.mat[l][i]))
        l = j;                                       // remember this row l
    // swap this pivot row, reason: minimize floating point error
    for (k = i; k <= N; k++)             // t is a temporary double variable
      t = Aug.mat[i][k], Aug.mat[i][k] = Aug.mat[l][k], Aug.mat[l][k] = t;
    for (j = i + 1; j < N; j++)         // the actual forward elimination phase
      for (k = N; k >= i; k--)
        Aug.mat[j][k] -= Aug.mat[i][k] * Aug.mat[j][i] / Aug.mat[i][i];
  }

  ColumnVector Ans;                            // the back substitution phase
  for (j = N - 1; j >= 0; j--) {                           // start from back
    for (t = 0.0, k = j + 1; k < N; k++) t += Aug.mat[j][k] * Ans.vec[k];
    Ans.vec[j] = (Aug.mat[j][N] - t) / Aug.mat[j][j]; // the answer is here
  }
  return Ans;
}

int main() {
  AugmentedMatrix Aug;
  Aug.mat[0][0] = 1; Aug.mat[0][1] = 1; Aug.mat[0][2] = 2; Aug.mat[0][3] = 9;
  Aug.mat[1][0] = 2; Aug.mat[1][1] = 4; Aug.mat[1][2] = -3; Aug.mat[1][3] = 1;
  Aug.mat[2][0] = 3; Aug.mat[2][1] = 6; Aug.mat[2][2] = -5; Aug.mat[2][3] = 0;

  ColumnVector X = GaussianElimination(3, Aug);
  printf("X = %.1lf, Y = %.1lf, Z = %.1lf\n", X.vec[0], X.vec[1], X.vec[2]);

  return 0;
}
```

```cpp
/****************************************************************************/
/** Lowest Common Ancestor (LCA) ............................................. */
/****************************************************************************/

#include <cstdio>
#include <vector>
using namespace std;

#define MAX_N 1000

vector< vector<int> > children;

int L[2*MAX_N], E[2*MAX_N], H[MAX_N], idx;

void dfs(int cur, int depth) {
  H[cur] = idx;
  E[idx] = cur;
  L[idx++] = depth;
  for (int i = 0; i < children[cur].size(); i++) {
    dfs(children[cur][i], depth+1);
    E[idx] = cur;                              // backtrack to current node
    L[idx++] = depth;
  }
}

void buildRMQ() {
  idx = 0;
  memset(H, -1, sizeof H);
  dfs(0, 0);                        // we assume that the root is at index 0
}

int main() {
  children.assign(10, vector<int>());
  children[0].push_back(1); children[0].push_back(7);
  children[1].push_back(2); children[1].push_back(3); children[1].push_back(6);
  children[3].push_back(4); children[3].push_back(5);
  children[7].push_back(8); children[7].push_back(9);

  buildRMQ();
  for (int i = 0; i < 2*10-1; i++) printf("%d ", H[i]);
  printf("\n");
  for (int i = 0; i < 2*10-1; i++) printf("%d ", E[i]);
  printf("\n");
  for (int i = 0; i < 2*10-1; i++) printf("%d ", L[i]);
  printf("\n");

  return 0;
}
```

```cpp
/*******************************************************************************/
/** Pollard Rho (fatoracao) ................................................... */
/*******************************************************************************/

#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;
```

```cpp
/*****************************************************************************/
/** Range Minimum Query (RMQ) ............................................ */
/*****************************************************************************/
#include <algorithm>
#include <cmath>
#include <cstdio>
using namespace std;

#define MAX_N 1000                              // adjust this value as needed
#define LOG_TWO_N 10                  // 2^10 > 1000, adjust this value as needed

class RMQ {                                             // Range Minimum Query
private:
  int _A[MAX_N], SpT[MAX_N][LOG_TWO_N];
public:
  RMQ(int n, int A[]) {     // constructor as well as pre-processing routine
    for (int i = 0; i < n; i++) {
      _A[i] = A[i];
      SpT[i][0] = i; // RMQ of sub array starting at index i + length 2^0=1
    }
    // the two nested loops below have overall time complexity = O(n log n)
    for (int j = 1; (1<<j) <= n; j++) // for each j s.t. 2^j <= n, O(log n)
      for (int i = 0; i + (1<<j) - 1 < n; i++)     // for each valid i, O(n)
        if (_A[SpT[i][j-1]] < _A[SpT[i+(1<<(j-1))][j-1]])               // RMQ
          SpT[i][j] = SpT[i][j-1];     // start at index i of length 2^(j-1)
        else                  // start at index i+2^(j-1) of length 2^(j-1)
          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));     // 2^k <= (j-i+1)
    if (_A[SpT[i][k]] <= _A[SpT[j-(1<<k)+1][k]]) return SpT[i][k];
    else                                          return SpT[j-(1<<k)+1][k];
} };

int main() {
  // same example as in chapter 2: segment tree
  int n = 7, A[] = {18, 17, 13, 19, 15, 11, 20};
  RMQ rmq(n, A);
  for (int i = 0; i < n; i++)
    for (int j = i; j < n; j++)
      printf("RMQ(%d, %d) = %d\n", i, j, rmq.query(i, j));

  return 0;
}
```

```cpp
/*****************************************************************************/
/** Fibonacci Modular .......................................................*/
/*****************************************************************************/
// Modular Fibonacci

#include <cmath>
#include <cstdio>
#include <cstring>
using namespace std;

typedef long long ll;
ll MOD;

#define MAX_N 2                                    // increase this if needed
struct Matrix { ll mat[MAX_N][MAX_N]; };      // to let us return a 2D array

Matrix matMul(Matrix a, Matrix b) {            // O(n^3), but O(1) as n = 2
  Matrix ans; int i, j, k;
  for (i = 0; i < MAX_N; i++)
    for (j = 0; j < MAX_N; j++)
      for (ans.mat[i][j] = k = 0; k < MAX_N; k++) {
        ans.mat[i][j] += (a.mat[i][k] % MOD) * (b.mat[k][j] % MOD);
        ans.mat[i][j] %= MOD;                 // modulo arithmetic is used here
      }
  return ans;
}

Matrix matPow(Matrix base, int p) {  // O(n^3 log p), but O(log p) as n = 2
  Matrix ans; int i, j;
  for (i = 0; i < MAX_N; i++)
    for (j = 0; j < MAX_N; j++)
      ans.mat[i][j] = (i == j);                      // prepare identity matrix
  while (p) {         // iterative version of Divide & Conquer exponentiation
    if (p & 1)                          // check if p is odd (the last bit is on)
      ans = matMul(ans, base);                                     // update ans
    base = matMul(base, base);                                // square the base
    p >>= 1;                                                     // divide p by 2
  }
  return ans;
}

int main() {
  int i, n, m;

  while (scanf("%d %d", &n, &m) == 2) {
    Matrix ans;                                 // special matrix for Fibonaccci
    ans.mat[0][0] = 1;  ans.mat[0][1] = 1;
    ans.mat[1][0] = 1;  ans.mat[1][1] = 0;
    for (MOD = 1, i = 0; i < m; i++)                              // set MOD = 2^m
      MOD *= 2;
    ans = matPow(ans, n);                                             // O(log n)
    printf("%lld\n", ans.mat[0][1]);                             // this if fib(n)
  }

  return 0;
}
```

```
/*******************************************************************************/
/** Shortest Path Faster Algorithm .............................................. */
/*******************************************************************************/
// Sending email
// standard SSSP problem
// demo using Dijkstra's and SPFA

#include <cstdio>
#include <iostream>
#include <queue>
#include <vector>
using namespace std;

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

#define INF 2000000000

int i, j, t, n, m, S, T, a, b, w, caseNo = 1;
vector<vii> AdjList;

int main() {
#ifndef ONLINE_JUDGE
  freopen("in.txt", "r", stdin);
#endif

  scanf("%d", &t);
  while (t--) {
    scanf("%d %d %d %d", &n, &m, &S, &T);

    // build graph
    AdjList.assign(n, vii());
    while (m--) {
      scanf("%d %d %d", &a, &b, &w);
      AdjList[a].push_back(ii(b, w)); // bidirectional
      AdjList[b].push_back(ii(a, w));
    }

/*
    // Dijkstra from source S
    vi dist(n, INF); dist[S] = 0;
    priority_queue< ii, vii, greater<ii> > pq; pq.push(ii(0, S)); // sort based on
                                                                // increasing distance
    while (!pq.empty()) { // main loop
      ii top = pq.top(); pq.pop(); // greedy: pick shortest unvisited vertex
      int d = top.first, u = top.second;
      if (d != dist[u]) continue;
      for (j = 0; j < (int)AdjList[u].size(); j++) { // all outgoing edges from u
        int v = AdjList[u][j].first, weight_u_v = AdjList[u][j].second;
        if (dist[u] + weight_u_v < dist[v]) { // if can relax
          dist[v] = dist[u] + weight_u_v; // relax
          pq.push(ii(dist[v], v)); // enqueue this neighbor
        }                          // regardless it is already in pq or not
      }
    }
*/

    // SPFA from source S
    // initially, only S has dist = 0 and in the queue
    vi dist(n, INF); dist[S] = 0;
    queue<int> q; q.push(S);
    vi in_queue(n, 0); in_queue[S] = 1;

    while (!q.empty()) {
      int u = q.front(); q.pop(); in_queue[u] = 0;
      for (j = 0; j < (int)AdjList[u].size(); j++) { // all outgoing edges from u
        int v = AdjList[u][j].first, weight_u_v = AdjList[u][j].second;
        if (dist[u] + weight_u_v < dist[v]) { // if can relax
          dist[v] = dist[u] + weight_u_v; // relax
```

```
          if (!in_queue[v]) { // add to the queue only if it’s not in the queue
            q.push(v);
            in_queue[v] = 1;
          }
        }
      }
    }

    printf("Case #%d: ", caseNo++);
    if (dist[T] != INF) printf("%d\n", dist[T]);
    else                printf("unreachable\n");
  }

  return 0;
}
```

```cpp
/*****************************************************************************/
/** Algarismos Romanos .................................................... */
/*****************************************************************************/
// Roman Numerals

#include <cstdio>
#include <cstdlib>
#include <ctype.h>
#include <map>
#include <string>
using namespace std;

void AtoR(int A) {
  map<int, string> cvt;
  cvt[1000] = "M"; cvt[900] = "CM"; cvt[500] = "D"; cvt[400] = "CD";
  cvt[100]  = "C"; cvt[90]  = "XC"; cvt[50]  = "L"; cvt[40]  = "XL";
  cvt[10]   = "X"; cvt[9]   = "IX"; cvt[5]   = "V"; cvt[4]   = "IV";
  cvt[1]    = "I";
  // process from larger values to smaller values
  for (map<int, string>::reverse_iterator i = cvt.rbegin();
       i != cvt.rend(); i++)
    while (A >= i->first) {
      printf("%s", ((string)i->second).c_str());
      A -= i->first; }
  printf("\n");
}

void RtoA(char R[]) {
  map<char, int> RtoA;
  RtoA['I'] = 1;   RtoA['V'] = 5;   RtoA['X'] = 10;   RtoA['L'] = 50;
  RtoA['C'] = 100; RtoA['D'] = 500; RtoA['M'] = 1000;

  int value = 0;
  for (int i = 0; R[i]; i++)
    if (R[i+1] && RtoA[R[i]] < RtoA[R[i+1]]) {     // check next char first
      value += RtoA[R[i+1]] - RtoA[R[i]];                // by definition
      i++; }                                              // skip this char
    else value += RtoA[R[i]];
  printf("%d\n", value);
}

int main() {
#ifndef ONLINE_JUDGE
  freopen("in.txt", "r", stdin);
#endif

  char str[1000];

  while (gets(str) != NULL) {
    if (isdigit(str[0])) AtoR(atoi(str)); // Arabic to Roman Numerals
    else                 RtoA(str); // Roman to Arabic Numerals
  }

  return 0;
}
```

```cpp
/*******************************************************************************/
/** Distancia entre pontos em esfera + dist. euclidiana ....................... */
/*******************************************************************************/
// Tunnelling the Earth
// Great Circle distance + Euclidean distance

#include <cstdio>
#include <cmath>
using namespace std;

#define PI acos(-1.0)
#define EARTH_RAD (6371009) // in meters

double gcDistance(double pLat, double pLong,
                  double qLat, double qLong, double radius) {
  pLat *= PI / 180; pLong *= PI / 180;
  qLat *= PI / 180; qLong *= PI / 180;
  return radius * acos(cos(pLat)*cos(pLong)*cos(qLat)*cos(qLong) +
                       cos(pLat)*sin(pLong)*cos(qLat)*sin(qLong) +
                       sin(pLat)*sin(qLat));
}

double EucledianDistance(double pLat, double pLong, // 3D version
                         double qLat, double qLong, double radius) {
  double phi1 = (90 - pLat) * PI / 180;
  double theta1 = (360 - pLong) * PI / 180;
  double x1 = radius * sin(phi1) * cos(theta1);
  double y1 = radius * sin(phi1) * sin(theta1);
  double z1 = radius * cos(phi1);

  double phi2 = (90 - qLat) * PI / 180;
  double theta2 = (360 - qLong) * PI / 180;
  double x2 = radius * sin(phi2) * cos(theta2);
  double y2 = radius * sin(phi2) * sin(theta2);
  double z2 = radius * cos(phi2);

  double dx = x1 - x2, dy = y1 - y2, dz = z1 - z2;
  return sqrt(dx * dx + dy * dy + dz * dz);
}

int main() {
  int TC;
  double lat1, lon1, lat2, lon2;

  scanf("%d", &TC);
  while (TC--) {
    scanf("%lf %lf %lf %lf", &lat1, &lon1, &lat2, &lon2);
    printf("%.0lf\n", gcDistance(lat1, lon1, lat2, lon2, EARTH_RAD) -
                      EucledianDistance(lat1, lon1, lat2, lon2, EARTH_RAD));
  }

  return 0;
}
```

```cpp
/********************************************************************************/
/** Componentes Fortemente Conectadas ........................................ */
/********************************************************************************/
// Come and Go
// check if the graph is strongly connected,
// i.e. the SCC of the graph is the graph itself (only 1 SCC)

#include <algorithm>
#include <cstdio>
#include <iostream>
#include <vector>
using namespace std;

typedef pair<int, int> ii;
typedef vector<int> vi;
typedef vector<ii> vii;
#define DFS_WHITE -1

int i, j, N, M, V, W, P, dfsNumberCounter, numSCC;
vector<vii> AdjList, AdjListT;
vi dfs_num, dfs_low, S, S_copy, visited;                          // global variables

void tarjanSCC(int u) {
  dfs_low[u] = dfs_num[u] = dfsNumberCounter++;       // dfs_low[u] <= dfs_num[u]
  S.push_back(u);            // stores u in a vector based on order of visitation
  visited[u] = 1;
  for (int j = 0; j < (int)AdjList[u].size(); j++) {
    ii v = AdjList[u][j];
    if (dfs_num[v.first] == DFS_WHITE)
      tarjanSCC(v.first);
    if (visited[v.first])                                 // condition for update
      dfs_low[u] = min(dfs_low[u], dfs_low[v.first]);
  }

  if (dfs_low[u] == dfs_num[u]) {          // if this is a root (start) of an SCC
    ++numSCC;
    while (1) {
      int v = S.back(); S.pop_back(); visited[v] = 0;
      if (u == v) break;
    }
  }
}

void Kosaraju(int u, int pass) {      // pass = 1 (original), 2 (transpose)
  dfs_num[u] = 1;
  vii neighbor;
  if (pass == 1) neighbor = AdjList[u]; else neighbor = AdjListT[u];
  for (int j = 0; j < (int)neighbor.size(); j++) {
    ii v = neighbor[j];
    if (dfs_num[v.first] == DFS_WHITE)
      Kosaraju(v.first, pass);
  }
  S.push_back(u);        // as in finding topological order in Section 4.2.5
}

int main() {
#ifndef ONLINE_JUDGE
  freopen("in.txt", "r", stdin);
#endif

  while (scanf("%d %d", &N, &M), (N || M)) {
    AdjList.assign(N, vii());
    AdjListT.assign(N, vii()); // the transposed graph
    for (i = 0; i < M; i++) {
      scanf("%d %d %d", &V, &W, &P); V--; W--;
      AdjList[V].push_back(ii(W, 1)); // always
      AdjListT[W].push_back(ii(V, 1));
      if (P == 2) { // if this is two way, add the reverse direction
        AdjList[W].push_back(ii(V, 1));
        AdjListT[V].push_back(ii(W, 1));
```

```
      }
    }

    //// run Tarjan's SCC code here
    //dfs_num.assign(N, DFS_WHITE); dfs_low.assign(N, 0); visited.assign(N, 0);
    //dfsNumberCounter = numSCC = 0;
    //for (i = 0; i < N; i++)
    //  if (dfs_num[i] == DFS_WHITE)
    //    tarjanSCC(i);

    // run Kosaraju's SCC code here
    S.clear();  // first pass is to record the 'post-order' of original graph
    dfs_num.assign(N, DFS_WHITE);
    for (i = 0; i < N; i++)
      if (dfs_num[i] == DFS_WHITE)
        Kosaraju(i, 1);

    numSCC = 0;   // second pass: explore the SCCs based on first pass result
    dfs_num.assign(N, DFS_WHITE);
    for (i = N-1; i >= 0; i--)
      if (dfs_num[S[i]] == DFS_WHITE) {
        numSCC++;
        Kosaraju(S[i], 2);
      }

    // if SCC is only 1, print 1, otherwise, print 0
    printf("%d\n", numSCC == 1 ? 1 : 0);
  }

  return 0;
}
```