Epsilon

Team Reference Document

12/09/2014

Contents

1.1. **C++ Template.** A C++ template.

```
#include <cctype>--------------------------------------------------------------//
#include <string>--------------------------------------------------------------//
#include <cstring>-------------------------------------------------------------//
#include <climits>-------------------------------------------------------------//
#include <cmath>---------------------------------------------------------------//
#include <cstdio>--------------------------------------------------------------//
#include <cstdlib>-------------------------------------------------------------//
#include <iostream>------------------------------------------------------------//
#include <sstream>-------------------------------------------------------------//
#include <fstream>-------------------------------------------------------------//
#include <iomanip>-------------------------------------------------------------//
#include <vector>--------------------------------------------------------------//
#include <list>---------------------------------------------------------------//
#include <set>----------------------------------------------------------------//
#include <map>----------------------------------------------------------------//
#include <stack>--------------------------------------------------------------//
#include <queue>--------------------------------------------------------------//
#include <algorithm>-----------------------------------------------------------//
#include <utility>-------------------------------------------------------------//
#include <bitset>--------------------------------------------------------------//
#include <limits>--------------------------------------------------------------//
-----------------------------------------------------------------------------//
using namespace std;---------------------------------------------------------//
-----------------------------------------------------------------------------//
#define PB(x)----   push_back(x)-----------------------------------------------//
#define MP(x,y)---- make_pair(x,y)---------------------------------------------//
#define ALL(x)----  x.begin(), x.end()-----------------------------------------//
#define RALL(x)---- x.rbegin(), x.rend()---------------------------------------//
#define READ(x)---- freopen(x, "r", stdin)-------------------------------------//
#define WRITE(x)----freopen(x, "w", stdout)------------------------------------//
#define SORT(x)---- sort(ALL(x))-----------------------------------------------//
#define DREP(x)---- sort(ALL(x)); x.erase(unique(ALL(x)),x.end())-------------//
#define CLEAR(c)----memset(c, 0, sizeof(c))------------------------------------//
#define P(x)--------">>> " << #x << " : " << x << endl-------------------------//
#define C(x)--------cout << P(x)-----------------------------------------------//
#define CC(x)----   clog << P(x)-----------------------------------------------//
-----------------------------------------------------------------------------//
struct _ { ios_base::Init i; _() { cin.sync_with_stdio(0); cin.tie(0); } } _;--//
-----------------------------------------------------------------------------//
typedef long long ll;--------------------------------------------------------//
typedef vector<int> vi;------------------------------------------------------//
typedef vector<ll> vl;-------------------------------------------------------//
typedef pair<int, int> ii;---------------------------------------------------//
typedef vector<ii> vii;------------------------------------------------------//
-----------------------------------------------------------------------------//
#define EPS 1e-14--------------------------------------------------------------//
#define INF 0x3f3f3f3f;--------------------------------------------------------//
#define PI atan(1)*4;----------------------------------------------------------//
-----------------------------------------------------------------------------//
```

```
int main() {
}
```

2. Data Structures

2.1. Union-Find. An implementation of the Union-Find disjoint sets data structure.

```
#include <cstdio>
#include <vector>

using namespace std;

typedef vector<int> vi;

class UnionFind {
  private:
    vi p, rank, setSize;
    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() {
  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)); // false
  printf("isSameSet(4, 3) = %d\n", UF.isSameSet(4, 3)); // true

  // findSet will return 1 for {0, 1} and 3 for {2, 3, 4}
  for (int i = 0; i < 5; i++) {
    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

  // findSet will return 3 for {0, 1, 2, 3, 4}
  for (int i = 0; i < 5; i++) {
    printf("findSet(%d) = %d, sizeOfSet(%d) = %d\n", i, UF.findSet(i),
                                                     i, UF.sizeOfSet(i));
  }
}
```

2.2. Segment Tree. An implementation of a Segment Tree.

```
#include <cmath>
#include <cstdio>
#include <vector>

using namespace std;

typedef vector<int> vi;

class SegmentTree {
  private: vi st, A;
    int n;
```

```
----int left (int p) { ------------------------------------------------------------//
----  return p << 1; ------------------------------------------------------------//
----}------------------------------------------------------------------------------//
-------------------------------------------------------------------------------//
----int right(int p) { -----------------------------------------------------------//
----  return (p << 1) + 1; -------------------------------------------------------//
----}------------------------------------------------------------------------------//
-------------------------------------------------------------------------------//
----void build(int p, int L, int R) {--------  // O(n log n)--------------------//
----  if (L == R) {--------------------------------------------------------------//
--------st[p] = L;-----------------------------------------------------------------//
----  } else {-------------------------------------------------------------------//
--------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;   // MIN---------------------------//
----  } --------------------------------------------------------------------------//
----}------------------------------------------------------------------------------//
-------------------------------------------------------------------------------//
  int rmq(int p, int L, int R, int i, int j) {  // O(log n)--------------------//
----if (i >  R || j <  L) return -1;----------------------------------------//
----if (L >= i && R <= j) return st[p];-------------------------------------//
-------------------------------------------------------------------------------//
----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 (p2 == -1) return p1;------------------------------------------------//
----return (A[p1] <= A[p2]) ? p1 : p2; }  // MIN-------------------------------//
-------------------------------------------------------------------------------//
  int update_point(int p, int L, int R, int idx, int new_value) {-------------//
----int i = idx, j = idx;---------------------------------------------------//
-------------------------------------------------------------------------------//
----if (i > R || j < L) return st[p];---------------------------------------//
-------------------------------------------------------------------------------//
----if (L == i && R == j) {-------------------------------------------------//
----  A[i] = new_value;--------------------------------------------------//
----  return st[p] = L;--------------------------------------------------//
----}------------------------------------------------------------------------//
-------------------------------------------------------------------------------//
----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 st[p] = (A[p1] <= A[p2]) ? p1 : p2;  // MIN-----------------------//
  }--------------------------------------------------------------------------//
-------------------------------------------------------------------------------//
  public:--------------------------------------------------------------------//
----SegmentTree(const vi &_A) {---------------------------------------------//
----  A = _A; n = (int)A.size();-----------------------------------------------//
----  st.assign(4 * n, 0);-----------------------------------------------------//
----  build(1, 0, n - 1);------------------------------------------------------//
----}------------------------------------------------------------------------//
-------------------------------------------------------------------------------//
----int rmq(int i, int j) { -------------------------------------------------//
----  return rmq(1, 0, n - 1, i, j); -------------------------------------//
----}------------------------------------------------------------------------//
-------------------------------------------------------------------------------//
----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 };--------------------------------//
  vi A(arr, arr + 7);--------------------------------------------------------//
  SegmentTree st(A);---------------------------------------------------------//
-------------------------------------------  //  0,  1,  2,  3,  4,  5, 6 ----//
-------------------------------------------  // 18, 17, 13, 19, 15, 11, 20----//
  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;------------------------------------------------------------------//
}--------------------------------------------------------------------------------//
```

2.3. **Fenwick Tree.** A Fenwick Tree is a data structure that represents an array of n numbers. It supports adjusting the i-th element in $O(\log n)$ time, and computing the sum of numbers in the range $i..j$ in $O(\log n)$ time. It only needs $O(n)$ space.

```
#include <cstdio>------------------------------------------------------------//
#include <vector>------------------------------------------------------------//
-------------------------------------------------------------------------------//
using namespace std;-------------------------------------------------------//
-------------------------------------------------------------------------------//
typedef vector<int> vi;----------------------------------------------------//
#define LSOne(S) (S & (-S))--------------------------------------------------//
-------------------------------------------------------------------------------//
```

```cpp
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) {//
    int sum = 0; //
    for (; b; b -= LSOne(b)) sum += ft[b];//
    return sum; //
  }//
//
  int rsq(int a, int b) {//
    return rsq(b) - (a == 1 ? 0 : rsq(a - 1)); //
  }//
//
  void adjust(int k, int v) {//
    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//
}//
```

2.4. **BigInt.** A big integer class.

```cpp
#include <iostream>//
#include <sstream>//
#include <vector>//
#include <cmath>//
#include <cstdio>//
#include <cstdlib>//
//
#define MIN(x,y) ((x) < (y) ? (x) : (y))//
#define MAX(x,y) ((x) > (y) ? (x) : (y))//
//
using namespace std;//
//
typedef unsigned int uint32;//
typedef unsigned long long int uint64;//
//
class BigUInt {//
  vector<uint32> data;//
//
  friend istream& operator>>(istream& is, BigUInt& bi);//
  friend ostream& operator<<(ostream& os, BigUInt& bi);//
  friend uint32 operator%(BigUInt& bi, uint32 v);//
//
  public://
//
  BigUInt() {}//
//
  BigUInt(string s) {//
    fromstring(s);//
  }//
//
  void setzero() {//
    data.clear();//
  }//
//
  void fromstring(string s) {//
    setzero();//
    size_t n = s.size();//
    static const uint32 tens[] = {0, 10, 100, 1000, 10000, 100000,//
                                  1000000, 10000000, 100000000};//
    for (size_t i = 0; i < n; i += 8) {//
      size_t len = MIN(8, s.size() - i);//
      string part = s.substr(i, len);//
      this->multiply(tens[len]);//
      istringstream is(part);//
      int a;//
      is >> a;//
      this->add(a);//
    }//
  }//
//
  string tostring() {//
    ostringstream os;//
    os << doubleval();//
    return os.str();//
  }//
//
  void add(uint32 val) {//
    uint64 left = val;//
    for (size_t i = 0; left; i++) {//
```

```
------------if (i >= data.size()) {-----------------------------------------//
---------------data.push_back(0);-----------------------------------------//
------------}-----------------------------------------//
------------uint64 sum = data[i] + left;-----------------------------------------//
------------data[i] = ((uint32)sum & ~(uint32)0);-----------------------------//
------------left = sum >> (sizeof(uint32) * 8);-----------------------------//
--------}-----------------------------------------//
----}-----------------------------------------//
-----------------------------------------//
----void multiply(uint32 val) {-----------------------------------------//
--------uint64 mente = 0;-----------------------------------------//
--------for (size_t i = 0; i < data.size() || mente; i++) {-----------------//
------------if (i >= data.size()) {-----------------------------------------//
---------------data.push_back(0);-----------------------------------------//
------------}-----------------------------------------//
------------uint64 product = (uint64)data[i] * (uint64)val + mente;------------//
------------data[i] = ((uint32)product & ~(uint32)0);-----------------------//
------------mente = product >> (sizeof(uint32) * 8);-----------------------//
--------}-----------------------------------------//
----}-----------------------------------------//
-----------------------------------------//
----static uint32 mod(BigUInt& bi, uint32 val) {-----------------------------//
--------uint64 rest = 0;-----------------------------------------//
--------for (int i = bi.data.size() - 1; i >= 0; i--) {-----------------------//
------------rest = ((rest << (sizeof(uint32) * 8)) + bi.data[i]) % val;--------//
--------}-----------------------------------------//
--------return rest;-----------------------------------------//
----}-----------------------------------------//
-----------------------------------------//
----double doubleval() {-----------------------------------------//
--------double dval = 0;-----------------------------------------//
--------for (size_t i = 0; i < data.size(); i++) {-----------------------------//
------------dval += (double)data[i] * pow(2.0, (int)(sizeof(uint32) * 8 * i));-//
--------}-----------------------------------------//
--------return dval;-----------------------------------------//
----}-----------------------------------------//
};-----------------------------------------//
-----------------------------------------//
istream& operator>>(istream& is, BigUInt& bi) {-----------------------------//
----string s;-----------------------------------------//
----is >> s;-----------------------------------------//
----bi.fromstring(s);-----------------------------------------//
----return is;-----------------------------------------//
}-----------------------------------------//
-----------------------------------------//
ostream& operator<<(ostream& os, BigUInt& bi) {-----------------------------//
----os << bi.tostring();-----------------------------------------//
----return os;-----------------------------------------//
}-----------------------------------------//
-----------------------------------------//
uint32 operator%(BigUInt& bi, uint32 v) {-----------------------------------//
----return BigUInt::mod(bi, v);-----------------------------------------//
}-----------------------------------------//
```

3. Graphs

3.1. Single-Source Shortest Paths.

3.1.1. *Dijkstra's algorithm.* An implementation of Dijkstra's 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, u, v, w;-----------------------------------------//
  vector<vii> AdjList;-----------------------------------------//
-----------------------------------------//
-----------------------------------------//
  freopen("in_05.txt", "r", stdin);-----------------------------------------//
-----------------------------------------//
  scanf("%d %d %d", &V, &E, &s);-----------------------------------------//
-----------------------------------------//
  AdjList.assign(V, vii());-----------------------------------------//
  for (int i = 0; i < E; i++) {-----------------------------------------//
----scanf("%d %d %d", &u, &v, &w);-----------------------------------------//
----AdjList[u].push_back(ii(v, w));-----------------------------------------//
  }-----------------------------------------//
-----------------------------------------//
-----------------------------------------//
  vi dist(V, INF); dist[s] = 0;-----------------------------------------//
  priority_queue< ii, vector<ii>, greater<ii> > pq; pq.push(ii(0, s));---------//
-----------------------------------------//
  while (!pq.empty()) {-----------------------------------------//
----ii front = pq.top(); pq.pop();-----------------------------------------//
----int d = front.first, u = front.second;-----------------------------------//
----if (d > dist[u]) continue;-----------------------------------------//
----for (int j = 0; j < (int)AdjList[u].size(); j++) {-----------------------//
----  ii v = AdjList[u][j];-----------------------------------------//
----  if (dist[u] + v.second < dist[v.first]) {-----------------------------//
--------dist[v.first] = dist[u] + v.second;-----------------------------------//
--------pq.push(ii(dist[v.first], v.first));-----------------------------------//
  } } }-----------------------------------------//
-----------------------------------------//
  for (int i = 0; i < V; i++)-----------------------------------------//
----printf("SSSP(%d, %d) = %d\n", s, i, dist[i]);-----------------------------//
-----------------------------------------//
```

```
  return 0;-------------------------------------------------------------------//
}---------------------------------------------------------------------------//
```

3.1.2. *Bellman-Ford algorithm.* The Bellman-Ford algorithm solves the single-source shortest paths problem in $O(|V||E|)$ time. It is slower than Dijkstra's algorithm, but it works on graphs with negative edges and has the ability to detect negative cycles, neither of which Dijkstra's algorithm can do.

```
#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;--------------------------------------------------------//
----------------------------------------------------------------------------//
  scanf("%d %d %d", &V, &E, &s);----------------------------------------------//
----------------------------------------------------------------------------//
  AdjList.assign(V, vii());---------------------------------------------------//
  for (int i = 0; i < E; i++) {-----------------------------------------------//
----scanf("%d %d %d", &a, &b, &w);-------------------------------------------//
----AdjList[a].push_back(ii(b, w));------------------------------------------//
  }---------------------------------------------------------------------------//
----------------------------------------------------------------------------//
  vi dist(V, INF); dist[s] = 0;-----------------------------------------------//
  for (int i = 0; i < V - 1; i++)---------------------------------------------//
----for (int u = 0; u < V; u++)-----------------------------------------------//
----  for (int j = 0; j < (int)AdjList[u].size(); j++) {-----------------------//
--------ii v = AdjList[u][j];-------------------------------------------------//
--------dist[v.first] = min(dist[v.first], dist[u] + v.second);---------------//
----  }---------------------------------------------------------------------//
----------------------------------------------------------------------------//
  bool hasNegativeCycle = false;----------------------------------------------//
  for (int u = 0; u < V; u++)-------------------------------------------------//
----for (int j = 0; j < (int)AdjList[u].size(); j++) {-----------------------//
----  ii v = AdjList[u][j];-------------------------------------------------//
----  if (dist[v.first] > dist[u] + v.second)-------------------------------//
--------hasNegativeCycle = true;----------------------------------------------//
----}-----------------------------------------------------------------------//
----------------------------------------------------------------------------//
  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]);---------------------------//
}---------------------------------------------------------------------------//
```

3.2. All-Pairs Shortest Paths.

3.2.1. *Floyd-Warshall algorithm.* The Floyd-Warshall algorithm solves the all-pairs shortest paths problem in $O(|V|^3)$ time.

```
#include <algorithm>----------------------------------------------------------//
#include <cstdio>-------------------------------------------------------------//
using namespace std;--------------------------------------------------------//
----------------------------------------------------------------------------//
#define INF 1000000000--------------------------------------------------------//
----------------------------------------------------------------------------//
int main() {----------------------------------------------------------------//
  int V, E, u, v, w, AdjMatrix[200][200];-------------------------------------//
----------------------------------------------------------------------------//
  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;-----------------------------------------------------//
  }---------------------------------------------------------------------------//
----------------------------------------------------------------------------//
  for (int k = 0; k < V; k++)-------------------------------------------------//
----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;-------------------------------------------------------------------//
}---------------------------------------------------------------------------//
```

3.3. Minimum Spanning Tree.

3.3.1. *Kruskal's algorithm.*

```
#include <algorithm>----------------------------------------------------------//
#include <cstdio>-------------------------------------------------------------//
#include <vector>-------------------------------------------------------------//
#include <queue>--------------------------------------------------------------//
using namespace std;--------------------------------------------------------//
----------------------------------------------------------------------------//
typedef pair<int, int> ii;--------------------------------------------------//
typedef vector<int> vi;-----------------------------------------------------//
typedef vector<ii> vii;-----------------------------------------------------//
----------------------------------------------------------------------------//
class UnionFind {-----------------------------------------------------------//
```

```
private:------------------------------------------------------------------------//
  vi p, rank, setSize;----------------------------------------------------------//
  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);----------------------------------------//
----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;---------------------------------------------------------------------//
priority_queue<ii> pq;--------------------------------------------------------//
//------------------------------------------------------------------------------//
void process(int vtx) {-------------------------------------------------------//
  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));---------------------//
} }---------------------------------------------------------------------------//
//------------------------------------------------------------------------------//
int main() {------------------------------------------------------------------//
  int V, E, u, v, w;------------------------------------------------------------//
//------------------------------------------------------------------------------//
  scanf("%d %d", &V, &E);-------------------------------------------------------//
  AdjList.assign(V, vii());-----------------------------------------------------//
  vector< pair<int, ii> > EdgeList;---------------------------------------------//
  for (int i = 0; i < E; i++) {-------------------------------------------------//
----scanf("%d %d %d", &u, &v, &w);---------------------------------------------//
----EdgeList.push_back(make_pair(w, ii(u, v)));-------------------------------//
----AdjList[u].push_back(ii(v, w));-------------------------------------------//
----AdjList[v].push_back(ii(u, w));-------------------------------------------//
  }-----------------------------------------------------------------------------//
  sort(EdgeList.begin(), EdgeList.end());---------------------------------------//
//------------------------------------------------------------------------------//
  int mst_cost = 0;-------------------------------------------------------------//
  UnionFind UF(V);--------------------------------------------------------------//
  for (int i = 0; i < E; i++) {-------------------------------------------------//
----pair<int, ii> front = EdgeList[i];----------------------------------------//
----if (!UF.isSameSet(front.second.first, front.second.second)) {-------------//
----  mst_cost += front.first;-----------------------------------------------//
----  UF.unionSet(front.second.first, front.second.second);------------------//
  } }---------------------------------------------------------------------------//
//------------------------------------------------------------------------------//
  printf("MST cost = %d (Kruskal's)\n", mst_cost);--------------------------//
//------------------------------------------------------------------------------//
  taken.assign(V, 0);-----------------------------------------------------------//
  process(0);-------------------------------------------------------------------//
  mst_cost = 0;-----------------------------------------------------------------//
  while (!pq.empty()) {---------------------------------------------------------//
----ii front = pq.top(); pq.pop();--------------------------------------------//
----u = -front.second, w = -front.first;--------------------------------------//
----if (!taken[u])------------------------------------------------------------//
----  mst_cost += w, process(u);---------------------------------------------//
  }-----------------------------------------------------------------------------//
  printf("MST cost = %d (Prim's)\n", mst_cost);-----------------------------//
}-----------------------------------------------------------------------------//
```

3.4. Maximum Flow.

3.4.1. *Edmonds Karp's algorithm.* An implementation of Edmonds Karp's algorithm that runs in $O(|V||E|^2)$. It computes the maximum flow of a flow network.

```
#include <algorithm>------------------------------------------------------------//
#include <cstdio>---------------------------------------------------------------//
#include <vector>---------------------------------------------------------------//
#include <queue>----------------------------------------------------------------//
using namespace std;----------------------------------------------------------//
//------------------------------------------------------------------------------//
typedef vector<int> vi;-------------------------------------------------------//
//------------------------------------------------------------------------------//
#define MAX_V 40----------------------------------------------------------------//
#define INF 1000000000----------------------------------------------------------//
//------------------------------------------------------------------------------//
int res[MAX_V][MAX_V], mf, f, s, t;-------------------------------------------//
vi p;-------------------------------------------------------------------------//
//------------------------------------------------------------------------------//
void augment(int v, int minEdge) {--------------------------------------------//
  if (v == s) { f = minEdge; return; }------------------------------------------//
  else if (p[v] != -1) { augment(p[v], min(minEdge, res[p[v]][v]));-------------//
------------------------ res[p[v]][v] -= f; res[v][p[v]] += f; }----------------//
}-----------------------------------------------------------------------------//
//------------------------------------------------------------------------------//
int main() {------------------------------------------------------------------//
  int V, k, vertex, weight;-----------------------------------------------------//
//------------------------------------------------------------------------------//
  /*----------------------------------------------------------------------------//
  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---------------------------------------------------------------//
//------------------------------------------------------------------------------//
  4 0 3-------------------------------------------------------------------------//
  2 1 100 3 100-----------------------------------------------------------------//
```

```
  2 2 1 3 100
  1 3 100
  0
-----
  5 1 0
  0
  2 2 100 3 50
  3 3 50 4 50 0 50
  1 4 100
  1 0 125
-----
  5 1 0
  0
  2 2 100 3 50
  3 3 50 4 50 0 50
  1 4 100
  1 0 75
-----
  5 1 0
  0
  2 2 100 3 50
  2 4 5 0 5
  1 4 100
  1 0 125
  */
-----
  scanf("%d %d %d", &V, &s, &t);
-----
  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;
  while (1) {
----f = 0;
----vi dist(MAX_V, INF); dist[s] = 0; 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 v = 0; v < MAX_V; v++)
--------if (res[u][v] > 0 && dist[v] == INF)
--------  dist[v] = dist[u] + 1, q.push(v), p[v] = u;
----}
----augment(t, INF);
----if (f == 0) break;
----mf += f;
  }
-----
  printf("%d\n", mf);
}
```

3.5. Bipartite.

3.5.1. *Breadth-first search with bipartite checking.* Breadth-first search and bipartite graph check

```
int main() {
  int visited[202];
  int n, l, x, y;
  vector<vi> v;
-----
  while (cin >> n, n != 0) {
----cin >> l;
----v.clear();
----v.assign(n+1, vi());
----memset(visited, -1, visited(c));
-----
----for (int i=0; i<l; i++) {
----  cin >> x >> y;
----  v[x].PB(y);
----  v[y].PB(x);
----}
-----
----queue<int> q;
----q.push(0);
-----
----bool isBipartite = true;
----visited[0] = 0;
----while(!q.empty() && isBipartite) {
----  int node = q.front();
----  q.pop();
-----
----  for (int i=0; i<v[node].size(); i++) {
--------int next = v[node][i];
--------if (visited[next] == -1) {
--------  visited[next] = visited[node] + 1;
--------  q.push(next);
--------} else if ((visited[node] % 2) == (visited[next] % 2)) {
--------  isBipartite = false;
--------}
----  }
----}
  }
}
```

4. Dynamic programming

4.1. **Longest increasing subsequence.** Find a subsequence of a given sequence in which the subsequence's elements are in sorted order, lowest to highest, and in which the subsequence is as long as possible

```cpp
#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;
}
```

5. Math

5.1. **Euclidean algorithm.** The Euclidean algorithm computes the greatest common divisor of two integers a, b.

```cpp
unsigned gcd (unsigned n1, unsigned n2) {
  return (n2 == 0) ? n1 : gcd (n2, n1 % n2);
}
```

5.2. **Primes.** Sieve, prime factors, etc.

```cpp
#include <bitset>
#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;
vi primes;

void sieve(ll upperbound) {
  _sieve_size = upperbound + 1;
  bs.set();
  bs[0] = bs[1] = 0;

  for (ll i = 2; i <= _sieve_size; i++) {
    if (bs[i]) {
      for (ll j = i * i; j <= _sieve_size; j += i) {
        bs[j] = 0;
      }
      primes.push_back((int)i);
    }
  }
}

bool isPrime(ll N) {
  if (N <= _sieve_size) {
    return bs[N];
  }

  for (int i = 0; i < (int)primes.size(); i++) {
    if (N % primes[i] == 0) {
      return false;
    }
  }

  return true;
}
```

```
//------------------------------------------------------------------------//
vi primeFactors(ll N) {-----------------------------------------------------//
  vi factors;---------------------------------------------------------------//
  ll PF_idx = 0, PF = primes[PF_idx];---------------------------------------//
//------------------------------------------------------------------------//
  while (N != 1 && (PF * PF <= N)) {----------------------------------------//
----while (N % PF == 0) {--------------------------------------------------//
----  N /= PF; factors.push_back(PF);----------------------------------------//
----}---------------------------------------------------------------------//
----PF = primes[++PF_idx];------------------------------------------------//
  }---------------------------------------------------------------------//
//------------------------------------------------------------------------//
  if (N != 1) {-------------------------------------------------------------//
----factors.push_back(N);-------------------------------------------------//
  }---------------------------------------------------------------------//
//------------------------------------------------------------------------//
  return factors;-----------------------------------------------------------//
}-------------------------------------------------------------------------//
//------------------------------------------------------------------------//
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------//
//------------------------------------------------------------------------//
  vi res = primeFactors(2147483647);----------------------------------------//
}-------------------------------------------------------------------------//
```

5.3. **Fraction.** A fraction (rational number) class. Note that numbers are stored in lowest common terms.

```
template <class T>--------------------------------------------------------//
class fraction {----------------------------------------------------------//
private:------------------------------------------------------------------//
----T gcd(T a, T b) { return b == T(0) ? a : gcd(b, a % b); }-------------//
public:-------------------------------------------------------------------//
----T n, d;---------------------------------------------------------------//
----fraction(T n_, T d_) {------------------------------------------------//
--------assert(d_ != 0);----------------------------------------------------//
--------n = n_, d = d_;----------------------------------------------------//
--------if (d < T(0)) n = -n, d = -d;--------------------------------------//
--------T g = gcd(abs(n), abs(d));------------------------------------------//
--------n /= g, d /= g; }---------------------------------------------------//
----fraction(T n_) : n(n_), d(1) { }--------------------------------------//
----fraction(const fraction<T>& other) : n(other.n), d(other.d) { }-------//
----fraction<T> operator +(const fraction<T>& other) const {-------------//
--------return fraction<T>(n * other.d + other.n * d, d * other.d);}-------//
----fraction<T> operator -(const fraction<T>& other) const {-------------//
--------return fraction<T>(n * other.d - other.n * d, d * other.d);}-------//
----fraction<T> operator *(const fraction<T>& other) const {-------------//
--------return fraction<T>(n * other.n, d * other.d); }--------------------//
----fraction<T> operator /(const fraction<T>& other) const {-------------//
--------return fraction<T>(n * other.d, d * other.n); }--------------------//
----bool operator <(const fraction<T>& other) const {--------------------//
--------return n * other.d < other.n * d; }--------------------------------//
----bool operator <=(const fraction<T>& other) const {-------------------//
--------return !(other < *this); }-----------------------------------------//
----bool operator >(const fraction<T>& other) const {--------------------//
--------return other < *this; }--------------------------------------------//
----bool operator >=(const fraction<T>& other) const {-------------------//
--------return !(*this < other); }-----------------------------------------//
----bool operator ==(const fraction<T>& other) const {-------------------//
--------return n == other.n && d == other.d; }-----------------------------//
----bool operator !=(const fraction<T>& other) const {-------------------//
--------return !(*this == other);}-----------------------------------------//
};------------------------------------------------------------------------//
```

6. Optimization

6.1. **Hungarian Method.** Combinatorial optimization algorithm that solves the assignment problem in polynomial time

```
#include <cstdio>-----------------------------------------------------------//
#include <cstdlib>----------------------------------------------------------//
#include <iostream>---------------------------------------------------------//
#include <string>-----------------------------------------------------------//
#include <vector>-----------------------------------------------------------//
#include <algorithm>--------------------------------------------------------//
#include <cassert>----------------------------------------------------------//
#include <utility>----------------------------------------------------------//
//------------------------------------------------------------------------//
const int INF = 0x3f3f3f3f;-----------------------------------------------//
#define N 50----------------------------------------------------------------//
//------------------------------------------------------------------------//
using namespace std;------------------------------------------------------//
//------------------------------------------------------------------------//
class Hungarian {---------------------------------------------------------//
  private:------------------------------------------------------------------//
----int n;----------------------------------------------------------------//
----int max_match;--------------------------------------------------------//
----int cost[N*N];--------------------------------------------------------//
----int costMin[N*N];------------------------------------------------------//
//------------------------------------------------------------------------//
----vector<int> lx;-------------------------------------------------------//
----vector<int> ly;-------------------------------------------------------//
----vector<int> xy;-------------------------------------------------------//
----vector<int> yx;-------------------------------------------------------//
----vector<bool> S;-------------------------------------------------------//
----vector<bool> T;-------------------------------------------------------//
----int slack[N];---------------------------------------------------------//
----int slackx[N];--------------------------------------------------------//
----vector<int> prev;-----------------------------------------------------//
//------------------------------------------------------------------------//
----void update_labels() {------------------------------------------------//
----  int x, y, delta = INF;-------------------------------------------------//
----  for (y = 0; y < n; y++) {----------------------------------------------//
```

```
        if (!T[y]) {
          delta = min(delta, slack[y]);
        }
    }
    for (x = 0; x < n; x++) {
        if (S[x]) {
          lx[x] -= delta;
        }
    }
    for (y = 0; y < n; y++) {
        if (T[y]) {
          ly[y] += delta;
        }
    }
    for (y = 0; y < n; y++) {
        if (!T[y]) {
          slack[y] -= delta;
        }
    }
}

void init_labels() {
    lx.assign(n, 0);
    ly.assign(n, 0);
    for (int x = 0; x < n; x++) {
        for (int y = 0; y < n; y++) {
          int it = x * n + y;
          lx[x] = max(lx[x], cost[it]);
        }
    }
}

void add_to_tree(int x, int prevx) {
    S[x] = true;
    prev[x] = prevx;
    for (int y = 0; y < n; y++) {
        int it = x * n + y;
        if (lx[x] + ly[y] - cost[it] < slack[y]) {
          int it = x * n + y;
          slack[y] = lx[x] + ly[y] - cost[it];
          slackx[y] = x;
        }
    }
}

void augment() {
    if (max_match == n) {
        return;
    }
    int x, y, root;
    int q[N], wr = 0, rd = 0;

    S.assign(n, false);
    T.assign(n, false);
    prev.assign(n, -1);

    for (x = 0; x < n; x++) {
        if (xy[x] == -1) {
          q[wr++] = root = x;
          prev[x] = -2;
          S[x] = true;
          break;
        }
    }

    for (y = 0; y < n; y++) {
        int it = root * n + y;
        slack[y] = lx[root] + ly[y] - cost[it];
        slackx[y] = root;
    }

    while (true) {
        while (rd < wr) {
          x = q[rd++];

          for (y = 0; y < n; y++) {
            int it = x * n + y;

            if (cost[it] == lx[x] + ly[y] &&  !T[y]) {
              if (yx[y] == -1) {
                break;
              }

              T[y] = true;
              q[wr++] = yx[y];
              add_to_tree(yx[y], x);
            }
          }

          if (y < n) {
            break;
          }
        }

        if (y < n) {
          break;
```

```
        }
        update_labels();
        wr = rd = 0;

        for (y = 0; y < n; y++) {
          if (!T[y] &&  slack[y] == 0) {
            if (yx[y] == -1) {
              x = slackx[y];
              break;
            } else {
              T[y] = true;

              if (!S[yx[y]]) {
                q[wr++] = yx[y];
                add_to_tree(yx[y], slackx[y]);
              }
            }
          }
        }

        if (y < n) {
          break;
        }
      }

      if (y < n) {
        max_match++;

        for (int cx = x, cy = y, ty; cx != -2; cx = prev[cx], cy = ty) {
          ty = xy[cx];
          yx[cy] = cx;
          xy[cx] = cy;
        }

        augment();
      }
    }

    int getMax() {
      int ret = 0;
      max_match = 0;
      xy.assign(n, -1);
      yx.assign(n, -1);

      init_labels();
      augment();

      for (int x = 0; x < n; x++) {
        int it = x * n + xy[x];
        ret += cost[it];
      }

      return ret;
    }

    int getMin() {
      int ret = 0;
      max_match = 0;
      xy.assign(n, -1);
      yx.assign(n, -1);

      init_labels();
      augment();

      for (int x = 0; x < n; x++) {
        int it = x * n + xy[x];
        ret += costMin[it];
      }

      return ret;
    }

  public:
    int max_optimal;
    int min_optimal;

    Hungarian(int size, int array[]) {
      n = size;
      int max_value = 0;

      for (int row = 0; row < n; ++row) {
        for (int col = 0; col < n; ++col) {
          int it = row * n + col;
          cost[it] = array[it];
          costMin[it] = array[it];
          max_value = max(max_value, array[it]);
        }
      }

      max_optimal = getMax();

      for (int row = 0; row < n; ++row) {
        for (int col = 0; col < n; ++col) {
          int it = row * n + col;
          cost[it] = max_value - cost[it];
        }
      }

      min_optimal = getMin();
    }
};
```

```
//------------------------------------------------------------------------------//
int main() {------------------------------------------------------------------//
  int test_cost[] = { 250, 400, 250,----//  [ 250  400  250 ]------------------//
--------------------  400, 600, 250,----//  [ 400  600  250 ]------------------//
--------------------  200, 400, 250 };  //  [ 200  400  250 ]------------------//
  int n = 3;--------------------------------------------------------------------//
  Hungarian hungarian(n, test_cost);--------------------------------------------//
//------------------------------------------------------------------------------//
  assert(hungarian.max_optimal == 1100);----------------------------------------//
  //  [ (250)  400   250  ]-------------------------------------------------//
  //  [  400  (600)  250  ]   MAX COST = 1100-------------------------------//
  //  [  200   400  (250) ]-------------------------------------------------//
//------------------------------------------------------------------------------//
  assert(hungarian.min_optimal == 850);-----------------------------------------//
  //  [  250  (400)  250  ]-------------------------------------------------//
  //  [  400   600  (250) ]   MIN COST = 850--------------------------------//
  //  [ (200)  400   250  ]-------------------------------------------------//
}-----------------------------------------------------------------------------//
```

7. Strings

7.1. **Levenshtein Distance.** Distance between two words is the minimum number of single-character edits (i.e. insertions, deletions or substitutions) required to change one word into the other

```
template<class T> unsigned int levenshteinDistance(const T &s1, const T & s2) {//
  const size_t len1 = s1.size(), len2 = s2.size();------------------------------//
  vector<unsigned int> col(len2+1), prevCol(len2+1);----------------------------//
//------------------------------------------------------------------------------//
  for (unsigned int i = 0; i < prevCol.size(); i++) {---------------------------//
----prevCol[i] = i;------------------------------------------------------------//
  }-----------------------------------------------------------------------------//
//------------------------------------------------------------------------------//
  for (unsigned int i = 0; i < len1; i++) {-------------------------------------//
----col[0] = i+1;--------------------------------------------------------------//
----for (unsigned int j = 0; j < len2; j++) {----------------------------------//
----  col[j+1] = min(min(prevCol[1 + j] + 1, col[j] + 1),---------------------//
--------------------- prevCol[j] + (s1[i] == s2[j] ? 0 : 1));-----------------//
----}-------------------------------------------------------------------------//
----col.swap(prevCol);---------------------------------------------------------//
  }-----------------------------------------------------------------------------//
//------------------------------------------------------------------------------//
  return prevCol[len2];---------------------------------------------------------//
}-----------------------------------------------------------------------------//
```

7.2. **Damerau–Levenshtein distance.** Distance between two words is the minimum number of single-character edits (i.e. insertions, deletions, substitutions and transpositions) required to change one word into the other

```
template<class T> unsigned int damerauDistance(const T &s1, const T &s2) {-----//
  const size_t len1 = s1.size(), len2 = s2.size();------------------------------//
  unsigned int d[len1 + 1][len2 + 1], cost;-------------------------------------//
//------------------------------------------------------------------------------//
  for (int i = 0; i <= len1; i++) {---------------------------------------------//
----d[i][0] = i;---------------------------------------------------------------//
  }-----------------------------------------------------------------------------//
//------------------------------------------------------------------------------//
  for (int j = 0; j <= len2; j++) {---------------------------------------------//
----d[0][j] = j;---------------------------------------------------------------//
  }-----------------------------------------------------------------------------//
//------------------------------------------------------------------------------//
  for (int i = 1; i <= len1; i++) {---------------------------------------------//
----for (int j = 1; j <= len2; j++) {---------------------------------------------//
----  cost = s1[i - 1] == s2[j - 1] ? 0 : 1;----------------------------------//
//------------------------------------------------------------------------------//
----  d[i][j] = min(min(d[i - 1][j] + 1, d[i][j - 1] + 1),----------------------//
--------------------d[i - 1][j - 1] + cost);-------------------------------------//
//------------------------------------------------------------------------------//
----  if ((i > 1) && (j > 1) && (s1[i - 1] == s2[j - 2]) &&--------------------//
--------(s1[i - 2] == s2[j - 1])) {-------------------------------------------//
--------d[i][j] = min(d[i][j], d[i - 2][j - 2] + cost);------------------------//
----  }-----------------------------------------------------------------------//
----}---------------------------------------------------------------------------//
  }-----------------------------------------------------------------------------//
//------------------------------------------------------------------------------//
  return d[len1][len2];---------------------------------------------------------//
}-----------------------------------------------------------------------------//
```

8. Geometry

8.1. **Formulas.** Let $a = (a_x, a_y)$ and $b = (b_x, b_y)$ be two-dimensional vectors.

- $a \cdot b = |a||b| \cos\theta$, where θ is the angle between a and b.
- $a \times b = |a||b| \sin\theta$, where θ is the signed angle between a and b.
- $a \times b$ is equal to the area of the parallelogram with two of its sides formed by a and b. Half of that is the area of the triangle formed by a and b.

9. Other Algorithms

9.1. **Dates.** Functions to simplify date calculations.

```
int intToDay(int jd) {--------------------------------------------------------//
  return jd % 7;----------------------------------------------------------------//
}-----------------------------------------------------------------------------//
//------------------------------------------------------------------------------//
int dateToInt(int y, int m, int d) {------------------------------------------//
----return 1461 * (y + 4800 + (m - 14) / 12) / 4 +-----------------------------//
--------367 * (m - 2 - (m - 14) / 12 * 12) / 12 -----------------------------//
--------3 * ((y + 4900 + (m - 14) / 12) / 100) / 4 +---------------------------//
--------d - 32075;-----------------------------------------------------------//
}-----------------------------------------------------------------------------//
//------------------------------------------------------------------------------//
double dateToDouble(int y, int m, int d, int hour, int minute, int second) {---//
----int JDN = 1461 * (y + 4800 + (m - 14) / 12) / 4 +--------------------------//
--------367 * (m - 2 - (m - 14) / 12 * 12) / 12 -----------------------------//
--------3 * ((y + 4900 + (m - 14) / 12) / 100) / 4 +---------------------------//
--------d - 32075;-----------------------------------------------------------//
----return (double)JDN + (((double)hour - 12.0) / 24.0) +---------------------//
--------   ((double)minute / 1440.0) +-------------------------------------//
--------   ((double)second / 86400.0);-------------------------------------//
```

```
}----------------------------------------------------------------------------//
----------------------------------------------------------------------------//
void intToDate(int jd, int &y, int &m, int &d) {-------------------------------//
----int x, n, i, j;-------------------------------------------------------------//
----x = jd + 68569;------------------------------------------------------------//
----n = 4 * x / 146097;--------------------------------------------------------//
----x -= (146097 * n + 3) / 4;-------------------------------------------------//
----i = (4000 * (x + 1)) / 1461001;--------------------------------------------//
----x -= 1461 * i / 4 - 31;----------------------------------------------------//
----j = 80 * x / 2447;---------------------------------------------------------//
----d = x - 2447 * j / 80;-----------------------------------------------------//
----x = j / 11;----------------------------------------------------------------//
----m = j + 2 - 12 * x;--------------------------------------------------------//
----y = 100 * (n - 49) + i + x;------------------------------------------------//
}----------------------------------------------------------------------------//
```

9.2. **itoa.** Converts an integer value to a null-terminated string using the specified base and stores the result in the array given by str parameter.

```
char* itoa(int value, char* result, int base) {--------------------------------//
----if (base < 2 || base > 36) { *result = '\0'; return result; }---------------//
----------------------------------------------------------------------------//
----char* ptr = result, *ptr1 = result, tmp_char;-------------------------------//
----int tmp_value;-------------------------------------------------------------//
----------------------------------------------------------------------------//
----do {-----------------------------------------------------------------------//
--------tmp_value = value;-----------------------------------------------------//
--------value /= base;---------------------------------------------------------//
--------//*ptr++ = "zyxwvutsrqponmlkjihgfedcba9876543210123456789abcd----------//
--------// efghijklmnopqrstuvwxyz" [35 + (tmp_value - value * base)];----------//
--------*ptr++ = "ZYXWVUTSRQPONMLKJIHGFEDCBA9876543210123456789ABCDEFGHI-------//
---------------  JKLMNOPQRSTUVWXYZ" [35 + (tmp_value - value * base)];--------//
----} while ( value );---------------------------------------------------------//
----------------------------------------------------------------------------//
----if (tmp_value < 0) *ptr++ = '-';-------------------------------------------//
----------------------------------------------------------------------------//
----*ptr-- = '\0';-------------------------------------------------------------//
----------------------------------------------------------------------------//
----while(ptr1 < ptr) {---------------------------------------------------------//
--------tmp_char = *ptr;-------------------------------------------------------//
--------*ptr--= *ptr1;-------------------------------------------------------//
--------*ptr1++ = tmp_char;-------------------------------------------------//
----}---------------------------------------------------------------------------//
----------------------------------------------------------------------------//
----return result;-------------------------------------------------------------//
}----------------------------------------------------------------------------//
```

9.3. **Bit Hacks.**

- `n & -n` returns the first set bit in n.
- `n & (n - 1)` is 0 only if n is a power of two.
- `snoob(x)` returns the next integer that has the same amount of bits set as `x`. Useful for iterating through subsets of some specified size.

```
int snoob (int x) {------------------------------------------------------------//
  int y = x + (x & -x);--------------------------------------------------------//
  x = x & ~y;------------------------------------------------------------------//
  while ((x & 1) == 0) x  = x >> 1;--------------------------------------------//
  x = x >> 1;------------------------------------------------------------------//
  return y | x;----------------------------------------------------------------//
}----------------------------------------------------------------------------//
```

10. Useful Information

10.1. **Tips & Tricks.**

- How fast does our algorithm have to be? Can we use brute-force?
- Does order matter?
- Is it better to look at the problem in another way? Maybe backwards?
- Are there subproblems that are recomputed? Can we cache them?
- Do we need to remember everything we compute, or just the last few iterations of computation?
- Does it help to sort the data?
- Can we speed up lookup by using a map (tree or hash) or an array?
- Can we binary search the answer?
- Can we add vertices/edges to the graph to make the problem easier? Can we turn the graph into some other kind of a graph (perhaps a DAG, or a flow network)?
- Make sure integers are not overflowing.
- Is it better to compute the answer modulo n? Perhaps we can compute the answer modulo $m_1, m_2, \ldots, m_k$, where $m_1, m_2, \ldots, m_k$ are pairwise coprime integers, and find the real answer using CRT?
- Are there any edge cases? When $n = 0, n = -1, n = 1, n = 2^{31} - 1$ or $n = -2^{31}$? When the list is empty, or contains a single element? When the graph is empty, or contains a single vertex? When the graph contains self-loops? When the polygon is concave or non-simple?
- Can we use exponentiation by squaring?

10.2. **128-bit Integer.** GCC has a 128-bit integer data type named `__int128`. Useful if doing multiplication of 64-bit integers, or something needing a little more than 64-bits to represent.

10.3. **Worst Time Complexity.**

n	Worst AC Algorithm	Comment
≤ 10	$O(n!), O(n^6)$	e.g. Enumerating a permutation
≤ 15	$O(2^n \times n^2)$	e.g. DP TSP
≤ 20	$O(2^n), O(n^5)$	e.g. DP + bitmask technique
≤ 50	$O(n^4)$	e.g. DP with 3 dimensions + $O(n)$ loop, choosing ${}_nC_k = 4$
$\leq 10^2$	$O(n^3)$	e.g. Floyd Warshall's
$\leq 10^3$	$O(n^2)$	e.g. Bubble/Selection/Insertion sort
$\leq 10^5$	$O(n \log_2 n)$	e.g. Merge sort, building a Segment tree
$\leq 10^6$	$O(n), O(\log_2 n), O(1)$	Usually, contest problems have $n \leq 10^6$ (e.g. to read input)