**Algoritmo Complejidad**

QuickSort O(N log N)

Binary Search O(log N)

Stack, Queue y ArrayList O(1)

Tree, Priority Queue O(log N)

DFS O(V+E)

BFS O(V+E)

MST O(E log V)

Dijkstra O((V+E) log V)

Belmann Ford O(V^3)

Floyd Warshall O(V^3)

DP O(k M)

Backtracking O(k^M)

Max Flow O(V E^2)

KMP O(N)

MCBM O(V E)

#########################################################################################

class ch2\_02\_**algorithm\_collections** {

public static void main(String[] args) {

Vector<Integer> v = new Vector<Integer>();

v.add(10); v.add(7); v.add(2); v.add(15); v.add(4);

// sort descending with vector

Collections.sort(v);

// if we want to modify comparison function, use the overloaded method: Collections.sort(List list, Comparator c);

Collections.reverse(v);

System.out.println(v);

// shuffle the content again

Collections.shuffle(v);

System.out.println(v);

// sort ascending

Collections.sort(v);

System.out.println(v);

int pos = Collections.binarySearch(v, 7);

System.out.println("Trying to search for 7 in v, found at index = " + pos);

pos = Collections.binarySearch(v, 77);

System.out.println("Trying to search for 77 in v, found at index = " + pos); // output is -5 (explanation below)

/\*

binarySearch will returns:

index of the search key, if it is contained in the list;

otherwise, (-(insertion point) - 1).

The insertion point is defined as the point at which the key would be inserted into the list:

the index of the first element greater than the key,

or list.size(), if all elements in the list are less than the specified key.

Note that this guarantees that the return value will be >= 0 if and only if the key is found.

\*/

System.out.printf("min(10, 7) = %d\n", Math.min(10, 7));

System.out.printf("max(10, 7) = %d\n", Math.max(10, 7));

}

}

#########################################################################################

class ch2\_03\_**bit\_manipulation** {

private static int setBit(int S, int j) { return S | (1 << j); }

private static int isOn(int S, int j) { return S & (1 << j); }

private static int clearBit(int S, int j) { return S & ~(1 << j); }

private static int toggleBit(int S, int j) { return S ^ (1 << j); }

private static int lowBit(int S) { return S & (-S); }

private static int setAll(int n) { return (1 << n) - 1; }

private static int modulo(int S, int N) { return ((S) & (N - 1)); } // returns S % N, where N is a power of 2

private static int isPowerOfTwo(int S) { return (S & (S - 1)) == 0 ? 1 : 0; }

private static int nearestPowerOfTwo(int S) { return ((int)Math.pow(2.0, (int)((Math.log((double)S) / Math.log(2.0)) + 0.5))); }

private static int turnOffLastBit(int S) { return ((S) & (S - 1)); }

private static int turnOnLastZero(int S) { return ((S) | (S + 1)); }

private static int turnOffLastConsecutiveBits(int S) { return ((S) & (S + 1)); }

private static int turnOnLastConsecutiveZeroes(int S) { return ((S) | (S - 1)); }

}

#########################################################################################

class ch2\_05\_**map\_set** {

public static void main(String[] args) {

// note: there are many clever usages of this set/map

TreeSet<Integer> used\_values = new TreeSet<Integer>(); // must use TreeSet as Set is an abstract class

TreeMap<String, Integer> mapper = new TreeMap<String, Integer>(); // must use TreeMap as Map is an abstract class

mapper.put("john", 78); used\_values.add(78);

mapper.put("billy", 69); used\_values.add(69);

mapper.put("andy", 80); used\_values.add(80);

mapper.put("steven", 77); used\_values.add(77);

mapper.put("felix", 82); used\_values.add(82);

mapper.put("grace", 75); used\_values.add(75);

mapper.put("martin", 81); used\_values.add(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)

System.out.println(mapper.keySet());

System.out.println(mapper.values());

// map can also be used like this

System.out.println("steven's score is " + mapper.get("steven") + ", grace's score is " + mapper.get("grace"));

System.out.println("==================");

// interesting usage of SubMap

// display data between ["f".."m") ('felix' is included, martin' is excluded)

SortedMap<String, Integer> res = mapper.subMap("f", "m");

System.out.println(res.keySet());

System.out.println(res.values());

// 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

System.out.println(used\_values.contains(77)); // returns true

System.out.println(used\_values.headSet(77)); // returns [69, 75] (these two are before 77 in the inorder traversal of this BST)

System.out.println(used\_values.tailSet(77)); // returns [77, 78, 80, 81, 82] (these five are equal or after 77 in the inorder traversal of this BST)

if (!used\_values.contains(79)) // O(log n) search, not found

System.out.println("79 not found");

}

}

#########################################################################################

class pair < X, Y > { // utilizing Java "Generics"

X \_first;

Y \_second;

public pair(X f, Y s) { \_first = f; \_second = s; }

X first() { return \_first; }

Y second() { return \_second; }

void setFirst(X f) { \_first = f; }

void setSecond(Y s) { \_second = s; }

}

class **ch2\_06\_priority\_queue** {

public static void main(String[] args) {

// introducing 'pair'

PriorityQueue < pair < Integer, String > > pq = new PriorityQueue < pair < Integer, String > >(1,

new Comparator< pair < Integer, String > >() { // overriding the compare method

public int compare(pair < Integer, String > i, pair < Integer, String > j) {

return j.first() - i.first(); // currently max heap, reverse these two to try produce min-heap

}

}

);

// suppose we enter these 7 money-name pairs below

pq.offer(new pair < Integer, String > (100, "john")); // inserting a pair in O(log n)

pq.offer(new pair < Integer, String > (10, "billy"));

pq.offer(new pair < Integer, String > (20, "andy"));

pq.offer(new pair < Integer, String > (100, "steven"));

pq.offer(new pair < Integer, String > (70, "felix"));

pq.offer(new pair < Integer, String > (2000, "grace"));

pq.offer(new pair < Integer, String > (70, "martin"));

// this is how we use Java PriorityQueue

// priority queue will arrange items in 'heap' based

// on the first key in pair, which is money (integer), largest first

// if first keys tied, 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

pair<Integer, String> result = pq.poll(); // O(1) to access the top / max element + O(log n) removal of the top and repair the structure

System.out.println(result.second() + " has " + result.first() + " $");

result = pq.poll();

System.out.println(result.second() + " has " + result.first() + " $");

result = pq.poll();

System.out.println(result.second() + " has " + result.first() + " $");

}

}

#########################################################################################

class **UnionFind** { // OOP style

private Vector<Integer> p, rank, setSize;

private int numSets;

public UnionFind(int N) {

p = new Vector<Integer>(N);

rank = new Vector<Integer>(N);

setSize = new Vector<Integer>(N);

numSets = N;

for (int i = 0; i < N; i++) {

p.add(i);

rank.add(0);

setSize.add(1);

}

}

public int findSet(int i) {

if (p.get(i) == i) return i;

else {

int ret = findSet(p.get(i)); p.set(i, ret);

return ret; } }

public Boolean isSameSet(int i, int j) { return findSet(i) == findSet(j); }

public 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.get(x) > rank.get(y)) { p.set(y, x); setSize.set(x, setSize.get(x) + setSize.get(y)); }

else { p.set(x, y); setSize.set(y, setSize.get(y) + setSize.get(x));

if (rank.get(x) == rank.get(y)) rank.set(y, rank.get(y) + 1); } } }

public int numDisjointSets() { return numSets; }

public int sizeOfSet(int i) { return setSize.get(findSet(i)); }

}

class ch2\_08\_unionfind\_ds {

public static void main(String[] args) {

System.out.printf("Assume that there are 5 disjoint sets initially\n");

UnionFind UF = new UnionFind(5); // create 5 disjoint sets

System.out.printf("%d\n", UF.numDisjointSets()); // 5

UF.unionSet(0, 1);

System.out.printf("%d\n", UF.numDisjointSets()); // 4

UF.unionSet(2, 3);

System.out.printf("%d\n", UF.numDisjointSets()); // 3

UF.unionSet(4, 3);

System.out.printf("%d\n", UF.numDisjointSets()); // 2

System.out.printf("isSameSet(0, 3) = %b\n", UF.isSameSet(0, 3)); // will return false

System.out.printf("isSameSet(4, 3) = %b\n", UF.isSameSet(4, 3)); // will return true

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

System.out.printf("findSet(%d) = %d, sizeOfSet(%d) = %d\n", i, UF.findSet(i), i, UF.sizeOfSet(i));

UF.unionSet(0, 3);

System.out.printf("%d\n", UF.numDisjointSets()); // 1

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

System.out.printf("findSet(%d) = %d, sizeOfSet(%d) = %d\n", i, UF.findSet(i), i, UF.sizeOfSet(i));

}

}

#########################################################################################

class **SegmentTree** { // the segment tree is stored like a heap array

private int[] st, A;

private int n;

private int left (int p) { return p << 1; } // same as binary heap operations

private int right(int p) { return (p << 1) + 1; }

private void build(int p, int L, int R) {

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;

} }

private 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

private 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 position 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 position where the overall minimum is

return st[p] = (A[p1] <= A[p2]) ? p1 : p2;

}

public SegmentTree(int[] \_A) {

A = \_A; n = A.length; // copy content for local usage

st = new int[4 \* n];

for (int i = 0; i < 4 \* n; i++) st[i] = 0; // create vector with length `len' and fill it with zeroes

build(1, 0, n - 1); // recursive build

}

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

public int update\_point(int idx, int new\_value) {

return update\_point(1, 0, n - 1, idx, new\_value); }

}

class ch2\_09\_segmenttree\_ds{

public static void main(String[] args) {

int[] A = new int[] { 18, 17, 13, 19, 15, 11, 20 }; // the original array

SegmentTree st = new SegmentTree(A);

System.out.printf(" idx 0, 1, 2, 3, 4, 5, 6\n");

System.out.printf(" A is {18,17,13,19,15, 11,20}\n");

System.out.printf("RMQ(1, 3) = %d\n", st.rmq(1, 3)); // answer = index 2

System.out.printf("RMQ(4, 6) = %d\n", st.rmq(4, 6)); // answer = index 5

System.out.printf("RMQ(3, 4) = %d\n", st.rmq(3, 4)); // answer = index 4

System.out.printf("RMQ(0, 0) = %d\n", st.rmq(0, 0)); // answer = index 0

System.out.printf("RMQ(0, 1) = %d\n", st.rmq(0, 1)); // answer = index 1

System.out.printf("RMQ(0, 6) = %d\n", st.rmq(0, 6)); // answer = index 5

System.out.printf(" idx 0, 1, 2, 3, 4, 5, 6\n");

System.out.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

System.out.printf("These values do not change\n");

System.out.printf("RMQ(1, 3) = %d\n", st.rmq(1, 3)); // 2

System.out.printf("RMQ(3, 4) = %d\n", st.rmq(3, 4)); // 4

System.out.printf("RMQ(0, 0) = %d\n", st.rmq(0, 0)); // 0

System.out.printf("RMQ(0, 1) = %d\n", st.rmq(0, 1)); // 1

System.out.printf("These values change\n");

System.out.printf("RMQ(0, 6) = %d\n", st.rmq(0, 6)); // 5->2

System.out.printf("RMQ(4, 6) = %d\n", st.rmq(4, 6)); // 5->4

System.out.printf("RMQ(4, 5) = %d\n", st.rmq(4, 5)); // 5->4

}

}

#########################################################################################

class **FenwickTree** {

private Vector<Integer> ft;

private int LSOne(int S) { return (S & (-S)); }

public FenwickTree() {}

// initialization: n + 1 zeroes, ignore index 0

public FenwickTree(int n) {

ft = new Vector<Integer>();

for (int i = 0; i <= n; i++) ft.add(0);

}

public int rsq(int b) { // returns RSQ(1, b)

int sum = 0; for (; b > 0; b -= LSOne(b)) sum += ft.get(b);

return sum; }

public 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.set(k, ft.get(k) + v); }

};

class ch2\_10\_fenwicktree\_ds {

public static void main(String[] args) {

// idx 0 1 2 3 4 5 6 7 8 9 10, no index 0!

FenwickTree ft = new FenwickTree(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

System.out.printf("%d\n", ft.rsq(1, 1)); // 0 => ft[1] = 0

System.out.printf("%d\n", ft.rsq(1, 2)); // 1 => ft[2] = 1

System.out.printf("%d\n", ft.rsq(1, 6)); // 7 => ft[6] + ft[4] = 5 + 2 = 7

System.out.printf("%d\n", ft.rsq(1, 10)); // 11 => ft[10] + ft[8] = 1 + 10 = 11

System.out.printf("%d\n", ft.rsq(3, 6)); // 6 => rsq(1, 6) - rsq(1, 2) = 7 - 1

ft.adjust(5, 2); // update demo

System.out.printf("%d\n", ft.rsq(1, 10)); // now 13

}

}

#########################################################################################

public class ch4\_08\_**edmonds\_karp** {

private static final int MAX\_V = 40; // enough for sample graph in Figure 4.24/4.25/4.26

private static final int INF = 1000000000;

// we need these global variables

private static int[][] res = new int[MAX\_V][]; // define MAX\_V appropriately

private static int mf, f, s, t;

private static Vector < Integer > p = new Vector < Integer > ();

private static void augment(int v, int minEdge) { // traverse the BFS spanning tree as in print\_path (section 4.3)

if (v == s) { f = minEdge; return; } // reach the source, record minEdge in a global variable `f'

else if (p.get(v) != -1) { augment(p.get(v), Math.min(minEdge, res[p.get(v)][v])); // recursive call

res[p.get(v)][v] -= f; res[v][p.get(v)] += f; } // alter residual capacities

}

public static void main(String[] args) throws Exception {

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

File ff = new File("in\_08.txt");

Scanner sc = new Scanner(ff);

V = sc.nextInt();

s = sc.nextInt();

t = sc.nextInt();

for (int i = 0; i < V; i++) {

res[i] = new int[MAX\_V];

k = sc.nextInt();

for (int j = 0; j < k; j++) {

vertex = sc.nextInt();

weight = sc.nextInt();

res[i][vertex] = weight;

}

}

mf = 0;

while (true) { // run O(VE^2) Edmonds Karp to solve the Max Flow problem

f = 0;

// run BFS, please examine parts of the BFS code that is different than in Section 4.3

Queue < Integer > q = new LinkedList < Integer > ();

Vector < Integer > dist = new Vector < Integer > ();

dist.addAll(Collections.nCopies(V, INF)); // #define INF 2000000000

q.offer(s);

dist.set(s, 0);

p.clear();

p.addAll(Collections.nCopies(V, -1)); // (we have to record the BFS spanning tree)

while (!q.isEmpty()) { // (we need the shortest path from s to t!)

int u = q.poll();

if (u == t) break; // immediately stop BFS if we already reach sink t

for (int v = 0; v < MAX\_V; v++) // note: enumerating neighbors with AdjMatrix is `slow'

if (res[u][v] > 0 && dist.get(v) == INF) { // res[u][v] can change!

dist.set(v, dist.get(u) + 1);

q.offer(v);

p.set(v, u); // parent of vertex v is vertex u

}

}

augment(t, INF); // find the min edge weight `f' along this path, if any

if (f == 0) break; // if we cannot send any more flow (`f' = 0), terminate the loop

mf += f; // we can still send a flow, increase the max flow!

}

System.out.printf("%d\n", mf); // this is the max flow value of this flow graph

}

}

#########################################################################################

class ch4\_09\_**mcbm** {

private static Vector < Vector < Integer > > AdjList =

new Vector < Vector < Integer > >();

private static Vector < Integer > match, visited; // global variables

private static int Aug(int l) {

if (visited.get(l) == 1) return 0;

visited.set(l, 1);

Iterator it = AdjList.get(l).iterator();

while (it.hasNext()) { // either greedy assignment or recurse

Integer right = (Integer)it.next();

if (match.get(right) == -1 || Aug(match.get(right)) == 1) {

match.set(right, l);

return 1; // we found one matching

}

}

return 0; // no matching

}

public static void main(String[] args) {

int i, j;

/\*

// Graph in Figure 4.40 can be built on the fly

// we know there are 6 vertices in this bipartite graph, l side are numbered 0,1,2, right side 3,4,5

//int V = 6, V\_l = 3;

//int set1[] = new int[] {1,7,11}, set2[] = new int[] {4,10,12};

// Graph in Figure 4.41 can be built on the fly

// we know there are 5 vertices in this bipartite graph, l side are numbered 0,1, right side 3,4,5

int V = 5, V\_l = 2;

int set1[] = new int[] {1,7}, set2[] = new int[] {4,10,12};

// build the bipartite graph, only directed edge from l to right is needed

AdjList.clear();

for (i = 0; i < V; i++) {

Vector<Integer> Neighbor = new Vector<Integer>();

AdjList.add(Neighbor); // store blank vector first

}

for (i = 0; i < V\_l; i++)

for (j = 0; j < 3; j++)

if (isprime(set1[i] + set2[j]))

AdjList.get(i).add(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, V\_l = 3;

AdjList.clear();

for (i = 0; i < V; i++) {

Vector<Integer> Neighbor = new Vector<Integer>();

AdjList.add(Neighbor); // store blank vector first

}

AdjList.get(1).add(3);

AdjList.get(1).add(4);

AdjList.get(2).add(3);

int MCBM = 0;

match = new Vector < Integer > ();

match.addAll(Collections.nCopies(V, -1));

for (int l = 0; l < V\_l; l++) {

visited = new Vector < Integer > ();

visited.addAll(Collections.nCopies(V\_l, 0));

MCBM += Aug(l);

}

System.out.printf("Found %d matchings\n", MCBM);

}

}

#########################################################################################

class **LCA** {

public static Vector< Vector < Integer > > children = new Vector < Vector < Integer > > ();

public static int[] L = new int[2000], E = new int[2000], H = new int[2000];

public static int idx;

public static void dfs(int cur, int depth) {

H[cur] = idx;

E[idx] = cur;

L[idx++] = depth;

for (int i = 0; i < children.get(cur).size(); i++) {

dfs(children.get(cur).get(i), depth+1);

E[idx] = cur; // backtrack to current node

L[idx++] = depth;

}

}

public static void buildRMQ() {

idx = 0;

for (int i = 0; i < 2000; i++)

H[i] = -1;

dfs(0, 0); // we assume that the root is at index 0

}

public static void main(String[] args) {

for (int i = 0; i < 10; i++)

children.add(new Vector < Integer > ());

children.get(0).add(1); children.get(0).add(7);

children.get(1).add(2); children.get(1).add(3); children.get(1).add(6);

children.get(3).add(4); children.get(3).add(5);

children.get(7).add(8); children.get(7).add(9);

buildRMQ();

for (int i = 0; i < 2\*10-1; i++) System.out.printf("%d ", H[i]);

System.out.printf("\n");

for (int i = 0; i < 2\*10-1; i++) System.out.printf("%d ", E[i]);

System.out.printf("\n");

for (int i = 0; i < 2\*10-1; i++) System.out.printf("%d ", L[i]);

System.out.printf("\n");

}

}

#########################################################################################

class ch6\_02\_**kmp** {

char[] T, P; // T = text, P = pattern

int n, m; // n = length of T, m = length of P

int [] b; // b = back table

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

System.out.printf("P is found at index %d in T\n", i - j);

j = b[j]; // prepare j for the next possible match

} } }

void run() {

String Tstr = "I DO NOT LIKE SEVENTY SEV BUT SEVENTY SEVENTY SEVEN";

String Pstr = "SEVENTY SEVEN";

T = new String(Tstr).toCharArray();

P = new String(Pstr).toCharArray();

n = T.length;

m = P.length;

System.out.println(T);

System.out.println(P);

System.out.println();

System.out.printf("KMP\n");

b = new int[100010];

kmpPreprocess();

kmpSearch();

System.out.println();

System.out.printf("String Library\n");

int pos = Tstr.indexOf(Pstr);

while (pos != -1) {

System.out.printf("P is found at index %d in T\n", pos);

pos = Tstr.indexOf(Pstr, pos + 1);

}

System.out.println();

}

}

#########################################################################################

class ch6\_03\_**str\_align** {

public static void main(String[] args){

char[] A = "ACAATCC".toCharArray(), B = "AGCATGC".toCharArray();

int[][] table = new int[20][20]; // Needleman Wunsnch's algorithm

int i, j, n = A.length, m = B.length;

// 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); // cost for match or mismatches

// insert/delete = -1 point

table[i][j] = Math.max(table[i][j], table[i - 1][j] - 1); // delete

table[i][j] = Math.max(table[i][j], table[i][j - 1] - 1); // insert

}

System.out.printf("DP table:\n");

for (i = 0; i <= n; i++) {

for (j = 0; j <= m; j++)

System.out.printf("%3d", table[i][j]);

System.out.printf("\n");

}

System.out.printf("Maximum Alignment Score: %d\n", table[n][m]);

}

}

#########################################################################################

class ch6\_01\_**basic\_string** {

static int isvowel(char ch) { // make sure ch is in lowercase

String vowel = "aeiou";

for (int j = 0; j < 5; j++)

if (vowel.charAt(j) == ch)

return 1;

return 0;

}

public static void main(String[] args) throws Exception {

int i, pos, digits, alphas, vowels, consonants;

Boolean first, prev\_dash, this\_dash;

String str = "";

first = true; // technique to differentiate first line with the other lines

prev\_dash = this\_dash = false; // to differentiate whether the previous line contains a dash or not

File f = new File("ch6.txt");

Scanner sc = new Scanner(f);

while (sc.hasNext()) {

String line = sc.nextLine();

if (line.equals(".......")) break;

if (line.charAt(line.length() - 1) == '-') {

line = line.substring(0, line.length() - 1); // if the last character is '-', delete it

this\_dash = true;

}

else

this\_dash = false;

if (!first && !prev\_dash)

str = str + " "; // only append " " if this line is the second one onwards

first = false;

str = str + line;

prev\_dash = this\_dash;

}

char[] temp = str.toCharArray();

for (i = digits = alphas = vowels = consonants = 0; i < str.length(); i++) { // we can use str[i] as terminating condition as string in C++ is also terminated with NULL (0)

temp[i] = Character.toLowerCase(temp[i]); // make each character lower case

digits += Character.isDigit(temp[i]) ? 1 : 0;

alphas += Character.isLetter(temp[i]) ? 1 : 0;

vowels += isvowel(temp[i]); // already returns 1 or 0

}

consonants = alphas - vowels;

str = new String(temp);

System.out.println(str);

System.out.printf("%d %d %d\n", digits, vowels, consonants);

int hascs3233 = (str.indexOf("cs3233") != -1) ? 1 : 0;

Vector<String> tokens = new Vector<String>();

TreeMap<String, Integer> freq = new TreeMap<String, Integer>();

StringTokenizer st = new StringTokenizer(str, " .");

while (st.hasMoreTokens()) {

String p = st.nextToken();

tokens.add(p);

if (!freq.containsKey(p)) freq.put(p, 1);

else freq.put(p, freq.get(p) + 1);

}

Collections.sort(tokens);

System.out.println(tokens.get(0) + " " + tokens.get(tokens.size() - 1));

System.out.printf("%d\n", hascs3233);

int ans\_s = 0, ans\_h = 0, ans\_7 = 0;

String lastline = sc.nextLine();

for (i = 0; i < lastline.length(); i++) {

char ch = lastline.charAt(i);

if (ch == 's') ans\_s++;

else if (ch == 'h') ans\_h++;

else if (ch == '7') ans\_7++;

}

System.out.printf("%d %d %d\n", ans\_s, ans\_h, ans\_7);

}

}

#########################################################################################

public class ch5\_06\_**primes** {

int \_sieve\_size;

boolean[] bs; // 10^7 should be enough for most cases

List<Integer> primes = new ArrayList<Integer>(); // compact list of primes in form of vector<int>

// first part

void sieve(int upperbound) { // create list of primes in [0..upperbound]

\_sieve\_size = upperbound + 1; // add 1 to include upperbound

bs = new boolean[\_sieve\_size];

Arrays.fill(bs,true); // set all bits to 1

bs[0] = bs[1] = false; // except index 0 and 1

for (long i = 2; i < \_sieve\_size; i++) if (bs[(int)i]) {

// cross out multiples of i starting from i \* i!

for (long j = i \* i; j < \_sieve\_size; j += i) bs[(int)j] = false;

primes.add((int)i); // also add this vector containing list of primes

} } // call this method in main method

boolean isPrime(long N) { // a good enough deterministic prime tester

if (N < \_sieve\_size) return bs[(int)N]; // O(1) for small primes

for (int i = 0; i < primes.size(); i++)

if (N % primes.get(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

List<Integer> primeFactors(long N) { // remember: vi is vector of integers, long is long long

List<Integer> factors = new ArrayList<Integer>(); // vi `primes' (generated by sieve) is optional

int PF\_idx = 0;

long PF = primes.get(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.add((int)PF); } // remove this PF

PF = primes.get(++PF\_idx); // only consider primes!

}

if (N != 1) factors.add((int)N); // special case if N is actually a prime

return factors; // if pf exceeds 32-bit integer, you have to change vi

}

// third part

long numPF(long N) {

int PF\_idx = 0;

long PF = primes.get(PF\_idx), ans = 0;

while (N != 1 && (PF \* PF <= N)) {

while (N % PF == 0) { N /= PF; ans++; }

PF = primes.get(++PF\_idx);

}

if (N != 1) ans++;

return ans;

}

long numDiffPF(long N) {

int PF\_idx = 0;

long PF = primes.get(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.get(++PF\_idx);

}

if (N != 1) ans++;

return ans;

}

long sumPF(long N) {

int PF\_idx = 0;

long PF = primes.get(PF\_idx), ans = 0;

while (N != 1 && (PF \* PF <= N)) {

while (N % PF == 0) { N /= PF; ans += PF; }

PF = primes.get(++PF\_idx);

}

if (N != 1) ans += N;

return ans;

}

long numDiv(long N) {

int PF\_idx = 0;

long PF = primes.get(PF\_idx), ans = 1; // start from ans = 1

while (N != 1 && (PF \* PF <= N)) {

long power = 0; // count the power

while (N % PF == 0) { N /= PF; power++; }

ans \*= (power + 1); // according to the formula

PF = primes.get(++PF\_idx);

}

if (N != 1) ans \*= 2; // (last factor has pow = 1, we add 1 to it)

return ans;

}

long sumDiv(long N) {

int PF\_idx = 0;

long PF = primes.get(PF\_idx), ans = 1; // start from ans = 1

while (N != 1 && (PF \* PF <= N)) {

long power = 0;

while (N % PF == 0) { N /= PF; power++; }

ans \*= ((long)Math.pow((double)PF, power + 1.0) - 1) / (PF - 1); // formula

PF = primes.get(++PF\_idx);

}

if (N != 1) ans \*= ((long)Math.pow((double)N, 2.0) - 1) / (N - 1); // last one

return ans;

}

long EulerPhi(long N) {

int PF\_idx = 0;

long PF = primes.get(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.get(++PF\_idx);

}

if (N != 1) ans -= ans / N; // last factor

return ans;

}

void run(){

// first part: the Sieve of Eratosthenes

sieve(10000000); // can go up to 10^7 (need few seconds)

System.out.printf("%b\n", isPrime(2147483647)); // 10-digits prime

System.out.printf("%b\n", isPrime(136117223861L)); // not a prime, 104729\*1299709

// second part: prime factors

List<Integer> res = primeFactors(2147483647); // slowest, 2147483647 is a prime

for (int i : res) System.out.printf("> %d\n", i);

res = primeFactors(136117223861L); // slow, 2 large pfactors 104729\*1299709

for (int i : res) System.out.printf("# %d\n", i);

res = primeFactors(142391208960L); // faster, 2^10\*3^4\*5\*7^4\*11\*13

for (int i : res) System.out.printf("! %d\n", i);

//res = primeFactors((long)(1010189899 \* 1010189899)); // "error"

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

// third part: prime factors variants

System.out.printf("numPF(%d) = %d\n", 50, numPF(50)); // 2^1 \* 5^2 => 3

System.out.printf("numDiffPF(%d) = %d\n", 50, numDiffPF(50)); // 2^1 \* 5^2 => 2

System.out.printf("sumPF(%d) = %d\n", 50, sumPF(50)); // 2^1 \* 5^2 => 2 + 5 + 5 = 12

System.out.printf("numDiv(%d) = %d\n", 50, numDiv(50)); // 1, 2, 5, 10, 25, 50, 6 divisors

System.out.printf("sumDiv(%d) = %d\n", 50, sumDiv(50)); // 1 + 2 + 5 + 10 + 25 + 50 = 93

System.out.printf("EulerPhi(%d) = %d\n", 50, EulerPhi(50)); // 20 integers < 50 are relatively prime with 50

}

public static void main(String[] args){

new ch5\_06\_primes().run();

}

}

#########################################################################################

public class **mapcolouring** {

final int V = 4;

int color[];

/\* A utility function to check if the current

color assignment is safe for vertex v \*/

boolean isSafe(int v, int graph[][], int color[],

int c)

{

for (int i = 0; i < V; i++)

if (graph[v][i] == 1 && c == color[i])

return false;

return true;

}

/\* A recursive utility function to solve m

coloring problem \*/

boolean graphColoringUtil(int graph[][], int m,

int color[], int v)

{

/\* base case: If all vertices are assigned

a color then return true \*/

if (v == V)

return true;

/\* Consider this vertex v and try different

colors \*/

for (int c = 1; c <= m; c++)

{

/\* Check if assignment of color c to v

is fine\*/

if (isSafe(v, graph, color, c))

{

color[v] = c;

/\* recur to assign colors to rest

of the vertices \*/

if (graphColoringUtil(graph, m,

color, v + 1))

return true;

/\* If assigning color c doesn't lead

to a solution then remove it \*/

color[v] = 0;

}

}

/\* If no color can be assigned to this vertex

then return false \*/

return false;

}

/\* This function solves the m Coloring problem using

Backtracking. It mainly uses graphColoringUtil()

to solve the problem. It returns false if the m

colors cannot be assigned, otherwise return true

and prints assignments of colors to all vertices.

Please note that there may be more than one

solutions, this function prints one of the

feasible solutions.\*/

boolean graphColoring(int graph[][], int m)

{

// Initialize all color values as 0. This

// initialization is needed correct functioning

// of isSafe()

color = new int[V];

for (int i = 0; i < V; i++)

color[i] = 0;

// Call graphColoringUtil() for vertex 0

if (!graphColoringUtil(graph, m, color, 0))

{

System.out.println("Solution does not exist");

return false;

}

// Print the solution

printSolution(color);

return true;

}

/\* A utility function to print solution \*/

void printSolution(int color[])

{

System.out.println("Solution Exists: Following" +

" are the assigned colors");

for (int i = 0; i < V; i++)

System.out.print(" " + color[i] + " ");

System.out.println();

}

// driver program to test above function

public static void main(String args[])

{

mapcolouring Coloring = new mapcolouring();

/\* Create following graph and test whether it is

3 colorable

(3)---(2)

| / |

| / |

| / |

(0)---(1)

\*/

int graph[][] = {{0, 1, 1, 1},

{1, 0, 1, 0},

{1, 1, 0, 1},

{1, 0, 1, 0},

};

int m = 3; // Number of colors

Coloring.graphColoring(graph, m);

}

}

#########################################################################################

class Node

{

int key;

Node left, right;

public Node(int item)

{

key = item;

left = right = null;

}

}

class **BinaryTree**

{

// Root of Binary Tree

Node root;

BinaryTree()

{

root = null;

}

/\* Given a binary tree, print its nodes according to the

"bottom-up" postorder traversal. \*/

void printPostorder(Node node)

{

if (node == null)

return;

// first recur on left subtree

printPostorder(node.left);

// then recur on right subtree

printPostorder(node.right);

// now deal with the node

System.out.print(node.key + " ");

}

/\* Given a binary tree, print its nodes in inorder\*/

void printInorder(Node node)

{

if (node == null)

return;

/\* first recur on left child \*/

printInorder(node.left);

/\* then print the data of node \*/

System.out.print(node.key + " ");

/\* now recur on right child \*/

printInorder(node.right);

}

/\* Given a binary tree, print its nodes in preorder\*/

void printPreorder(Node node)

{

if (node == null)

return;

/\* first print data of node \*/

System.out.print(node.key + " ");

/\* then recur on left sutree \*/

printPreorder(node.left);

/\* now recur on right subtree \*/

printPreorder(node.right);

}

// Wrappers over above recursive functions

void printPostorder() { printPostorder(root); }

void printInorder() { printInorder(root); }

void printPreorder() { printPreorder(root); }

// Driver method

public static void main(String[] args)

{

BinaryTree tree = new BinaryTree();

tree.root = new Node(1);

tree.root.left = new Node(2);

tree.root.right = new Node(3);

tree.root.left.left = new Node(4);

tree.root.left.right = new Node(5);

System.out.println("Preorder traversal of binary tree is ");

tree.printPreorder();

System.out.println("\nInorder traversal of binary tree is ");

tree.printInorder();

System.out.println("\nPostorder traversal of binary tree is ");

tree.printPostorder();

}

}

#########################################################################################

public class **matching** {

static char[] seq, subseq;

static long[][] tbl;

public static long countMatches() {

tbl = new long[seq.length + 1][subseq.length + 1];

for (int row = 0; row < tbl.length; row++)

for (int col = 0; col < tbl[row].length; col++)

tbl[row][col] = countMatchesFor(row, col);

return tbl[seq.length][subseq.length];

}

private static long countMatchesFor(int seqDigitsLeft, int subseqDigitsLeft) {

if (subseqDigitsLeft == 0)

return 1;

if (seqDigitsLeft == 0)

return 0;

char currSeqDigit = seq[seq.length-seqDigitsLeft];

char currSubseqDigit = subseq[subseq.length-subseqDigitsLeft];

long result = 0;

if (currSeqDigit == currSubseqDigit)

result += tbl[seqDigitsLeft - 1][subseqDigitsLeft - 1];

result += tbl[seqDigitsLeft - 1][subseqDigitsLeft];

return result;

}

public static void main(String args[])

{

Scanner in = new Scanner (System.in);

in.nextInt ();

seq = in.next ().toCharArray ();

subseq = "COW".toCharArray ();

System.out.println (countMatches ());

}

}

#########################################################################################

class ch7\_01\_**points\_lines** {

final double INF = 1e9;

final double EPS = 1e-9;

// we will use constant Math.PI in Java

double DEG\_to\_RAD(double d) { return d \* Math.PI / 180.0; }

double RAD\_to\_DEG(double r) { return r \* 180.0 / Math.PI; }

//struct point\_i { int x, y; }; // basic raw form, minimalist mode

class 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

class point implements Comparable<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

// use EPS (1e-9) when testing equality of two floating points

public int compareTo(point other) { // override less than operator

if (Math.abs(x - other.x) > EPS) // useful for sorting

return (int)Math.ceil(x - other.x); // first: by x-coordinate

else if (Math.abs(y - other.y) > EPS)

return (int)Math.ceil(y - other.y); // second: by y-coordinate

else

return 0; } }; // they are equal

double dist(point p1, point p2) { // Euclidean distance

// Math.hypot(dx, dy) returns sqrt(dx \* dx + dy \* dy)

return Math.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 new point(p.x \* Math.cos(rad) - p.y \* Math.sin(rad),

p.x \* Math.sin(rad) + p.y \* Math.cos(rad)); }

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

// the answer is stored in the third parameter

void pointsToLine(point p1, point p2, line l) {

if (Math.abs(p1.x - p2.x) < EPS) { // vertical line is fine

l.a = 1.0; l.b = 0.0; l.c = -p1.x;

} 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)

class line2 { double m, c; }; // another way to represent a line

int pointsToLine2(point p1, point p2, line2 l) {

if (Math.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

} }

boolean areParallel(line l1, line l2) { // check coefficients a & b

return (Math.abs(l1.a-l2.a) < EPS) && (Math.abs(l1.b-l2.b) < EPS); }

boolean areSame(line l1, line l2) { // also check coefficient c

return areParallel(l1 ,l2) && (Math.abs(l1.c - l2.c) < EPS); }

// returns true (+ intersection point) if two lines are intersect

boolean 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 (Math.abs(l1.b) > EPS) p.y = -(l1.a \* p.x + l1.c);

else p.y = -(l2.a \* p.x + l2.c);

return true; }

class vec { double x, y; // name: `vec' is different from Java Vector

vec(double \_x, double \_y) { x = \_x; y = \_y; } };

vec toVec(point a, point b) { // convert 2 points to vector

return new vec(b.x - a.x, b.y - a.y); }

vec scale(vec v, double s) { // nonnegative s = [<1 .. 1 .. >1]

return new vec(v.x \* s, v.y \* s); } // shorter.same.longer

point translate(point p, vec v) { // translate p according to v

return new 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 = new line(); // perpendicular to l and pass through p

if (Math.abs(l.b) < EPS) { // special case 1: vertical line

ans.x = -(l.c); ans.y = p.y; return; }

if (Math.abs(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 = new point();

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

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

// 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

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 = new point(a.x, a.y); // closer to a

return dist(p, a); } // Euclidean distance between p and a

if (u > 1.0) { c = new 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 Math.acos(dot(oa, ob) / Math.sqrt(norm\_sq(oa) \* norm\_sq(ob))); }

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

//// another variant

//int area2(point p, point q, point r) { // returns 'twice' the area of this triangle A-B-c

// 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

boolean 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

boolean collinear(point p, point q, point r) {

return Math.abs(cross(toVec(p, q), toVec(p, r))) < EPS; }

void run() {

point P1 = new point(), P2 = new point(), P3 = new point(0, 1); // note that both P1 and P2 are (0.00, 0.00)

System.out.println(P1.compareTo(P2) == 0); // true

System.out.println(P1.compareTo(P3) == 0); // false

point[] P = new point[6];

P[0] = new point(2, 2);

P[1] = new point(4, 3);

P[2] = new point(2, 4);

P[3] = new point(6, 6);

P[4] = new point(2, 6);

P[5] = new point(6, 5);

// sorting points demo

Arrays.sort(P);

for (int i = 0; i < P.length; i++)

System.out.printf("(%.2f, %.2f)\n", P[i].x, P[i].y);

// rearrange the points as shown in the diagram below

P = new point[7];

P[0] = new point(2, 2);

P[1] = new point(4, 3);

P[2] = new point(2, 4);

P[3] = new point(6, 6);

P[4] = new point(2, 6);

P[5] = new point(6, 5);

P[6] = new 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]);

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

// line equations

line l1 = new line(), l2 = new line(), l3 = new line(), l4 = new line();

pointsToLine(P[0], P[1], l1);

System.out.printf("%.2f \* x + %.2f \* y + %.2f = 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

System.out.printf("%.2f \* x + %.2f \* y + %.2f = 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);

System.out.printf("l1 & l2 are parallel? %b\n", areParallel(l1, l2)); // no

System.out.printf("l1 & l3 are parallel? %b\n", areParallel(l1, l3)); // yes, l1 (P[0]-P[1]) and l3 (P[2]-P[3]) are parallel

pointsToLine(P[2], P[4], l4);

System.out.printf("l1 & l2 are the same? %b\n", areSame(l1, l2)); // no

System.out.printf("l2 & l4 are the same? %b\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 = new point();

boolean res = areIntersect(l1, l2, p12); // yes, l1 (P[0]-P[1]) and l2 (P[0]-P[2]) are intersect at (2.0, 2.0)

System.out.printf("l1 & l2 are intersect? %b, at (%.2f, %.2f)\n", res, p12.x, p12.y);

// other distances

point ans = new point();

d = distToLine(P[0], P[2], P[3], ans);

System.out.printf("Closest point from P[0] to line (P[2]-P[3]): (%.2f, %.2f), dist = %.2f\n", ans.x, ans.y, d);

closestPoint(l3, P[0], ans);

System.out.printf("Closest point from P[0] to line V2 (P[2]-P[3]): (%.2f, %.2f), dist = %.2f\n", ans.x, ans.y, dist(P[0], ans));

d = distToLineSegment(P[0], P[2], P[3], ans);

System.out.printf("Closest point from P[0] to line SEGMENT (P[2]-P[3]): (%.2f, %.2f), dist = %.2f\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);

System.out.printf("Closest point from P[1] to line SEGMENT (P[2]-P[3]): (%.2f, %.2f), dist = %.2f\n", ans.x, ans.y, d); // closer to midway between AB = (3.20, 4.60)

d = distToLineSegment(P[6], P[2], P[3], ans);

System.out.printf("Closest point from P[6] to line SEGMENT (P[2]-P[3]): (%.2f, %.2f), dist = %.2f\n", ans.x, ans.y, d); // closer to B (or P[3]) = (6.00, 6.00)

reflectionPoint(l4, P[1], ans);

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

System.out.printf("Angle P[0]-P[4]-P[3] = %.2f\n", RAD\_to\_DEG(angle(P[0], P[4], P[3]))); // 90 degrees

System.out.printf("Angle P[0]-P[2]-P[1] = %.2f\n", RAD\_to\_DEG(angle(P[0], P[2], P[1]))); // 63.43 degrees

System.out.printf("Angle P[4]-P[3]-P[6] = %.2f\n", RAD\_to\_DEG(angle(P[4], P[3], P[6]))); // 180 degrees

System.out.printf("P[0], P[2], P[3] form A left turn? %b\n", ccw(P[0], P[2], P[3])); // no

System.out.printf("P[0], P[3], P[2] form A left turn? %b\n", ccw(P[0], P[3], P[2])); // yes

System.out.printf("P[0], P[2], P[3] are collinear? %b\n", collinear(P[0], P[2], P[3])); // no

System.out.printf("P[0], P[2], P[4] are collinear? %b\n", collinear(P[0], P[2], P[4])); // yes

point p = new point(3, 7), q = new point(11, 13), r = new point(35, 30); // collinear if r(35, 31)

System.out.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 = new point(2.0, 2.0);

point B = new point(4.0, 3.0);

vec v = toVec(A, B); // imagine there is an arrow from A to B (see the diagram above)

point C = new point(3.0, 2.0);

point D = translate(C, v); // D will be located in coordinate (3.0 + 2.0, 2.0 + 1.0) = (5.0, 3.0)

System.out.printf("D = (%.2f, %.2f)\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) = (4.0, 2.5)

System.out.printf("E = (%.2f, %.2f)\n", E.x, E.y);

// rotation

System.out.printf("B = (%.2f, %.2f)\n", B.x, B.y); // B = (4.0, 3.0)

point F = rotate(B, 90); // rotate B by 90 degrees COUNTER clockwise, F = (-3.0, 4.0)

System.out.printf("F = (%.2f, %.2f)\n", F.x, F.y);

point G = rotate(B, 180); // rotate B by 180 degrees COUNTER clockwise, G = (-4.0, -3.0)

System.out.printf("G = (%.2f, %.2f)\n", G.x, G.y);

}

}

##################################################################

// **DIVISORS**

#include <bits/stdc++.h>

// Function to print the divisors

void printDivisors(int n)

{

    // Note that this loop runs till square root

    for (int i=1; i<=sqrt(n)+1; i++)

    {

        if (n%i==0)

        {

            // If divisors are equal, print only one

            if (n/i == i)

                printf("%d ", i); // I is divisor

            else // Otherwise print both

                printf("%d %d ", i, n/i); // I and n/I are divisors

        }

    }

}

/\* Driver program to test above function \*/

int main()

{

    printf("The divisors of 100 are: \n");

    printDivisors(100);

    return 0;

}

##################################################################

public class **LongestCommonSubsequence** {

public static int find(char[] A, char[] B) {

int[][] LCS = new int[A.length + 1][B.length + 1];

String[][] solution = new String[A.length + 1][B.length + 1];

// if A is null then LCS of A, B =0

for (int i = 0; i <= B.length; i++) {

LCS[0][i] = 0;

solution[0][i] = "0";

}

// if B is null then LCS of A, B =0

for (int i = 0; i <= A.length; i++) {

LCS[i][0] = 0;

solution[i][0] = "0";

}

for (int i = 1; i <= A.length; i++) {

for (int j = 1; j <= B.length; j++) {

if (A[i - 1] == B[j - 1]) {

LCS[i][j] = LCS[i - 1][j - 1] + 1;

solution[i][j] = "diagonal";

} else {

LCS[i][j] = Math.max(LCS[i - 1][j], LCS[i][j - 1]);

if (LCS[i][j] == LCS[i - 1][j]) {

solution[i][j] = "top";

} else {

solution[i][j] = "left";

}

}

}

}

// below code is to just print the result

String x = solution[A.length][B.length];

String answer = "";

int a = A.length;

int b = B.length;

while (x != "0") {

if (solution[a][b] == "diagonal") {

answer = A[a - 1] + answer;

a--;

b--;

} else if (solution[a][b] == "left") {

b--;

} else if (solution[a][b] == "top") {

a--;

}

x = solution[a][b];

}

System.out.println(answer);

for (int i = 0; i <= A.length; i++) {

for (int j = 0; j <= B.length; j++) {

System.out.print(" " + LCS[i][j]);

}

System.out.println();

}

return LCS[A.length][B.length];

}

public static void main(String[] args) {

String A = "ACBDEA";

String B = "ABCDA";

System.out.println("LCS :" + find(A.toCharArray(), B.toCharArray()));

}

}

##################################################################

class **MaxBipartite**

{

// M is number of applicants and N is number of jobs

static final int M = 6;

static final int N = 6;

// A DFS based recursive function that returns true if a

// matching for vertex u is possible

boolean bpm(boolean bpGraph[][], int u, boolean seen[],

int matchR[])

{

// Try every job one by one

for (int v = 0; v < N; v++)

{

// If applicant u is interested in job v and v

// is not visited

if (bpGraph[u][v] && !seen[v])

{

seen[v] = true; // Mark v as visited

// If job 'v' is not assigned to an applicant OR

// previously assigned applicant for job v (which

// is matchR[v]) has an alternate job available.

// Since v is marked as visited in the above line,

// matchR[v] in the following recursive call will

// not get job 'v' again

if (matchR[v] < 0 || bpm(bpGraph, matchR[v],

seen, matchR))

{

matchR[v] = u;

return true;

}

}

}

return false;

}

// Returns maximum number of matching from M to N

int maxBPM(boolean bpGraph[][])

{

// An array to keep track of the applicants assigned to

// jobs. The value of matchR[i] is the applicant number

// assigned to job i, the value -1 indicates nobody is

// assigned.

int matchR[] = new int[N];

// Initially all jobs are available

for(int i=0; i<N; ++i)

matchR[i] = -1;

int result = 0; // Count of jobs assigned to applicants

for (int u = 0; u < M; u++)

{

// Mark all jobs as not seen for next applicant.

boolean seen[] =new boolean[N] ;

for(int i=0; i<N; ++i)

seen[i] = false;

// Find if the applicant 'u' can get a job

if (bpm(bpGraph, u, seen, matchR))

result++;

}

return result;

}

// Driver method

public static void main (String[] args) throws java.lang.Exception

{

// Let us create a bpGraph shown in the above example

boolean bpGraph[][] = new boolean[][]{

{false, true, true, false, false, false},

{true, false, false, true, false, false},

{false, false, true, false, false, false},

{false, false, true, true, false, false},

{false, false, false, false, false, false},

{false, false, false, false, false, true}

};

MaxBipartite m = new MaxBipartite();

System.out.println( "Maximum number of applicants that can"+

" get job is "+m.maxBPM(bpGraph));

}

}

##################################################################

FAST READER

/\*

PrintWriter printer = new PrintWriter(System.out);

printer.println("Hola mundo");

printer.close();

\*/

class FastReader

{

BufferedReader br;

StringTokenizer st;

public FastReader()

{ br = new BufferedReader(new InputStreamReader(System.in)); }

String next()

{

while (st == null || !st.hasMoreElements())

{

try

{ st = new StringTokenizer(br.readLine()); }

catch (IOException e)

{ e.printStackTrace(); }

}

return st.nextToken();

}

int nextInt()

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

long nextLong()

{ return Long.parseLong(next()); }

double nextDouble()

{ return Double.parseDouble(next()); }

String nextLine()

{

String str = "";

try

{ str = br.readLine(); }

catch (IOException e)

{ e.printStackTrace(); }

return str;

}

}