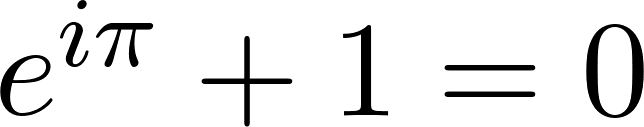
CheatSheet EAPC 2017 - Team [](https://www.codecogs.com/eqnedit.php?latex=e%5E%7Bi%5Cpi%7D%20%2B%201%20%3D%200).

[**Basics**](#_q45bjjnjh1fj) **2**

[Sorting](#_wrms4xieqin) 2

[Using Comparator](#_tgz9mluoalh2) 2

[Binary Search](#_vps0gxpzu6qg) 3

[Bitwise operations](#_9yhenyrprurr) 3

[**Number Theory**](#_87b1c8uv5rka) **4**

[Greatest Common Divisor, Least Common Multiplier](#_7865s3gj2hqe) 4

[Sieve of Eratosthenes](#_4mgqf633hlc) 4

[**Maths**](#_esuu36pqsgdm) **5**

[Permutations](#_9btd79yxc804) 5

[General maths](#_qsisscujozaq) 5

[Geometry Implementations](#_y6mj0a92gevn) 6

[Distance between point and line:](#_2joa9ze6et7a) 6

[Point in Polygon](#_2r0qpft0nsza) 7

[**Graphs**](#_f8rz1zmy3mrv) **7**

[Unweighted graph Algorithms](#_pdalea3n3d6c) 7

[DFS](#_mxh7uoq2muse) 7

[BFS](#_vpn42pnc2chx) 7

[Tarjan’s Algorithm](#_qu4czpcky8kc) 8

[Topological Sorting (cycle detection)](#_lew7ril0dzfg) 8

[Weighted graph Algorithms](#_z3n3p7bstria) 9

[Dijkstra](#_74wax93few1t) 9

[Bellman-Ford](#_glusqcko6s5h) 9

[Floyd-Warshall (shortest path with negative weights)](#_g8pnni1b9yo9) 11

[Minimum Spanning Tree (using Prim’s)](#_y06as94aov91) 12

[**Datastructures**](#_m26xt2b5711l) **13**

[Stack / Priority Queue / HashSet / HashMap](#_mazoq95k53g9) 13

[Pair](#_a84aeu94dmp3) 14

[UnionFind](#_gt7rxdsvk5ru) 15

[**Strings & Sequences**](#_8zc0bhvfejzv) **15**

[Check for duplicate characters in a string and print them.](#_h4wb8nl7qh81) 16

[Check if a string is an anagram](#_6oq1fi3s3vg5) 16

[Print all permutations of a string to the error output.](#_obkgppyprjd3) 17

[Check if a string is a palindrome](#_bupvjzya84lg) 17

[Knuth-Morris-Pratt](#_2qgc0nxsqi91) 17

[**Dynamic Programming**](#_52kkkuzdjw44) **18**

[Longest Common Subsequence](#_5kg22ach3qb2) 18

[Venus Rover (we solved this together)](#_ka2niewniixx) 19

[Knapsack 0/1](#_lqeacauzml7y) 20

[**Golden Rules and Quick Fixes**](#_75l9tl1f31w7) **21**

[**Eclipse Setup**](#_d357ja540nic) **21**

# Basics

## Sorting

Never write your own sorting algorithms! Use either built in sorting or comparators.

| In Arrays | In Lists |
| --- | --- |
| // Create an array to hold all fruits, populate it.  String[] fruits = new String[] {"BFruit", "AFruit"};  // Sort using the following  Arrays.sort(fruits);  // Results will be: "AFruit, BFruit". | // Create a list to hold fruits  List<String> fruits = new ArrayList<String>();  // add items to the list  fruits.add("Pineapple"); fruits.add("Apple");  // sort using the following  Collections.sort(fruits); |

### Using Comparator

| How to use the comparator in the main | A Comparator of type student, sorts by name |
| --- | --- |
| public class Main {  public static void main(String[] args) {  ArrayList<Student> ar = new ArrayList<>();  // arbitrarily add students to the list here  Collections.sort(ar, new Sortbyroll());  Collections.sort(ar, new Sortbyname());  }  } | class Sortbyname implements Comparator<Student> {  // Used for sorting in ascending order of roll name  @Override  public int compare(Student a, Student b) {  return a.name.compareTo(b.name);  }  } |
| Datastructure for a student | A Comparator of type Student, sorts by rollnumber |
| class Student {  String name, address;  int rollno;  // Constructor  public Student(int rollno, String name, String address) {  this.rollno = rollno;  this.name = name;  this.address = address;  }  } | class Sortbyroll implements Comparator<Student> {  //Used for sorting in ascending order of roll number  @Override  public int compare(Student a, Student b) {  return a.rollno - b.rollno;  }  } |

## 

## 

## Binary Search

| // note : Array has to be sorted  int recursiveBinarySearch(int[] sortedArray, int start, int end, int key) {  if (start < end) { // check if you're still searching  int mid = start + (end - start) / 2; // the middle of the array  if (key < sortedArray[mid]) { // key < value in the middle  return recursiveBinarySearch(sortedArray, start, mid, key);  } else if (key > sortedArray[mid]) { // key > value in the middle  return recursiveBinarySearch(sortedArray, mid+1, end , key);  } else { // key = value in the middle  return mid;  }  } // 'start' not less than 'end', so nothing found.  return -(start + 1); // this was in the example, so I kept it for consistency  } |
| --- |

## Bitwise operations

| void main() {  int i = 37; // 00100101 (37 dec)  i = (i << 2); // 10010100 (148 dec) SHIFT 2 places (useful for trees)  i = 37 & i; // 00000100 (4 dec) -- AND --  i = 3 | i; // 00000111 (7 dec) -- OR --  i = ~i; // 11...1000 -- COMPLEMENT --  i = 4 ^ 15; // 00001011 (11 dec) -- XOR --  } |
| --- |

# 

# 

# Number Theory

## Greatest Common Divisor, Least Common Multiplier

The extended Euclidian algorithm is used in two cases, as described above the algorithms

|  | ONLY IF |
| --- | --- |
| static long euclidesX, euclidesY, gcd;  static void extendedEuclides(long a, long b) {  long x = 0, y = 1, lastX = 1, lastY = 0, temp;  while(b != 0) {  long q = a / b;  long r = a % b;  a = b;  b = r;  temp = x;  x = lastX - q \* x;  lastX = temp;  temp = y;  y = lastY - q \* y;  lastY = temp;  }  gcd = a;  euclidesX = lastX;  euclidesY = lastY;  } | static long euclidesX, euclidesY, gcd;  static void extendedEuclides(long a, long b, long c) {  extendedEuclides(a, b);  euclidesX \*= c;  euclidesY \*= c;  long ag = a/gcd;  long bg = b/gcd;  if (euclidesX < 0) {  long steps = (-euclidesX-1)/bg+1;  euclidesX += bg\*steps;  euclidesY -= ag\*steps;  }  if (euclidesY < 0) {  long steps = (-euclidesY-1)/ag+1;  euclidesX -= bg\*steps;  euclidesY += ag\*steps;  }  } |

## Sieve of Eratosthenes

The sieve of Eratosthenes finds all prime numbers up to some value n. Use it when one needs to check if a number is prime. **Runningtime: O(n(logn)(loglogn))**

| final int n = 1000; // find it up to n  boolean[] A = new boolean[n]; // boolean array  for (int i = 0; i < n; i++) {  A[i] = true;  }  // set all duplicates to false  for (int i = 2; i < Math.sqrt(n); i++) {  if (A[i] == true) {  for (int multiple = i \* i; multiple < n; multiple += i) {  A[multiple] = false;  }  }  } // if A[i] then i is a prime number |
| --- |

# Maths

## Permutations

Heap’s algorithm takes an array A, and calls the method *output* for every permutation of A. The method *output* can be any operation you wish to run on every permutation of A

| static void generate(int[] A) {  int n = A.length;  int[] c = new int[n];  output(A); // gives a permutation of A  int i = 1;  while (i < n) {  if (c[i] < i) {  if (i % 2 == 0) {  swap(A, 0, i);  } else {  swap(A, c[i], i);  }  output(A); // gives more permutations of A  c[i]++;  i = 1;  } else {  c[i] = 0;  i++;  }  }  } | static void swap(int[] A, int i , int j) {  // swaps elements at i and j in array A  int t = A[i];  A[i] = A[j];  A[j] = t;  } |
| --- | --- |

## General maths

| double e = Math.E;  double pi = Math.PI;  double r = Math.random(); // 0.0 <= r < 1.0  double absoluteVal = Math.abs(numberOne); // absolute value  double logBaseE = Math.log(numberOne); // returns e^a  double logBase10 = Math.log10(numberOne);  double eToThePowerX = Math.exp(numberOne); // equal to Math.pow(e, numberOne)  double root = Math.sqrt(numberOne);  int ceilingOfX = (int) Math.ceil(numberOne);  int floorOfX = (int) Math.floor(numberOne);  int roundANumber = (int) Math.round(numberOne);  int max = (int) Math.max(numberOne, numberTwo);  int min = (int) Math.min(numberOne, numberTwo);  int pow = (int) Math.pow(numberOne, numberTwo);  double AngleToDegrees = Math.toDegrees(radianAngle);  double AngleToRadian = Math.toRadians(degreeAngle);  double sinusOfAngle = Math.sin(numberOne); // a in radians  double cosineOfAngle = Math.cos(numberOne); // a in radians  double tangentOfAngle = Math.tan(numberOne); // a in radians |
| --- |

**Rounding to n decimal digits:**

| DecimalFormat fourDigitsRound = new DecimalFormat("#.0000");  double x = 21341241.154951345;  System.out.println(fourDigitsRound.format(x)); // returns 21341241.1550 |
| --- |

## Geometry Implementations

| Point | Line / Circle |
| --- | --- |
| class Point {  double x, y;  Point(double nx, double ny) {  this.x = nx; this.y = ny;  }  Point() {}  // implement methods as seen fit  Point sub(Point p, Point q) {  return new Point (p.x - q.x, p.y - q.y);  }  double inp(Point p, Point q, Point O) {  return (p.x - O.x) \* (q.x - O.x) + (p.y - O.x) \* (q.y - O.x);  }  double inp(Point p, Point q) {  return Point.inp(p, q, new Point(0, 0));  }  double hat(Point p, Point q){  return p.x \* q.y - p.y \* q.x;  }  double cross(Point A, Point B, Point O) {  return (A.x - O.x) \* (B.y - O.y) - (A.y - O.y) \* (B.x - O.x);  }  double cross(Point A, Point B) {  return cross(A, B, new Point(0, 0));  }  } | class Line {  Point p1, p2;  Line(double p1x, double p1y, double p2x, double p2y)  {  p1 = new Point(p1x, p1y);  p2 = new Point(p2x, p2y);  }  Line() {  p1 = new Point(0, 0);  p2 = new Point(0, 0);  }  }  class Circle {  Point center;  double radius;  Circle(double cx, double cy, double r)  {  center = new Point(cx, cy);  radius = r;  }  Circle() {  center = new Point(0, 0);  radius = 1;  }  } |
|

## Distance between point and line:

| double dist(Point p, Line l, boolean line) {  Point q1, q2;  q1 = l.p1;  q2 = l.p2;  if (line) {  if (Point.inp(Point.sub(p, q1), Point.sub(q2, q1)) < 0) {  return Math.sqrt(Point.inp(Point.sub(p, q1), Point.sub(p, q1)));  }  if (Point.inp(Point.sub(p, q2), Point.sub(q1, q2)) < 0) {  return Math.sqrt(Point.inp(Point.sub(p, q2), Point.sub(p, q2)));  }  }  return Math.abs(Point.hat(Point.sub(p, q1), Point.sub(q2, q1)))  / Math.sqrt(Point.inp(Point.sub(q2, q1), Point.sub(q2, q1)));  }  double dist(Point p, Line l) {  return dist(p, l, false);  } |
| --- |

## Point in Polygon

| int inPoly(Point P, ArrayList<Point> V) {  int i, j = V.size() - 1, c = 0;  for (i = 0; i < V.size(); j = i++) {  if ((V.get(j).y <= P.y) && (P.y < V.get(i).y) && (Point.cross(P, V.get(j), V.get(i)) > 0)) {  ++c;  }  if ((V.get(i).y <= P.y) && (P.y < V.get(j).y) && (Point.cross(P, V.get(j), V.get(i)) > 0)) {  --c;  }  }  return c;  } |
| --- |

# Graphs

## Unweighted graph Algorithms

### DFS

| void DFS(Graph graph , int start) {  Stack <Integer > nextStack = new Stack <Integer >();  Stack <Integer > traversed = new Stack <Integer >();  // Enqueue root  nextStack.push(start);  while (!nextStack.isEmpty()) {  // Dequeue next node for comparison and add it 2 list of traversed nodes  int node = nextStack.pop();  System.out.println(node); // do something with node  traversed.push(node);  // Enqueue new neighbors  for (int i = 0; i < graph.vertices.get(node).con.size(); i++) {  int neighbor = graph.vertices.get(node).con.get(i).first;  if (!traversed.contains(neighbor) && !nextStack.contains(neighbor)) {  nextStack.push(neighbor);  }  }  }  } |
| --- |

### BFS

| void BFS(Graph graph , int begin) {  ArrayList <Integer > Q = new ArrayList <Integer >();  boolean visited[] = new boolean[NUM];  visited[start] = true;  Q.add(start);  while (!Q.isEmpty()) {  int nu = Q.get(0);  Q.remove(0);  for (int i = 0; i < graph.vertices.get(nu).con.size(); i++) {  int to = graph.vertices.get(nu).con.get(i).first;  if (!visited[to]) {  visited[to] = true;  Q.add(to);  System.out.println(to);  }  }  }  } |
| --- |

### Tarjan’s Algorithm

| //algorithm for searching all strongly connected components in a graph  int index = 0;  ArrayList <Node> stack = new ArrayList <Node >();  ArrayList <ArrayList <Node>> SCC = new ArrayList <ArrayList <Node >>();  ArrayList <ArrayList <Node>> tarjan(Node v, AdjacencyList list){  v.index = index;  v.lowlink = index;  index++;  stack.add(0, v);  for(Edge e : list.getAdjacent(v)){  Node n = e.to;  if(n.index == -1){  tarjan(n, list);  v.lowlink = Math.min(v.lowlink , n.lowlink);  } else if(stack.contains(n)){  v.lowlink = Math.min(v.lowlink , n.index);  }  }  if(v.lowlink == v.index){  Node n;  ArrayList <Node> component = new ArrayList <Node >();  do {  n = stack.remove(0);  component.add(n);  } while(n != v);  SCC.add(component);  }  return SCC;  } |
| --- |

### Topological Sorting (cycle detection)

| static int N; // input: number of nodes  static IntegerList[] edges; // input: edge nodes from a are in edges[a]  static List<Integer> L; // output: ordered list of nodes  static int[] marked; // used by topological sort  static boolean topologicalSort() { // returns false if graph has cycle  L = new ArrayList<>(N);  marked = new int[N];  for (int i = 0; i < N; i++) {  if (marked[i] == 0) {  if (!visit(i)) {  return false;  }  }  }  return true;  }  static boolean visit(int n) {  if (marked[n] == 1) {  return false;  }  if (marked[n] == 0) {  marked[n] = 1;  for (int m : edges[n]) {  if (!visit(m)) {  return false;  }  }  marked[n] = 2;  L.add(n);  }  return true;  } |
| --- |

## Weighted graph Algorithms

### Dijkstra (Single source shortest path)

| static class Edge {  int from, to, length;  Edge(int from, int to, int length) {this.from = from; this.to = to this.length = length }  }  static class EdgeList extends ArrayList<Edge> {}  static class Path {  static Comparator<Path> C = (p1, p2) ->  {  int dCompare = Integer.compare(p1.dist, p2.dist);  if (dCompare != 0) {  return dCompare;  }  return Integer.compare(p1.node, p2.node);  };  int node, dist;  Path(int node, int dist) {  this.node = node;  this.dist = dist;  }  }  // N: number of nodes, nodes reachable (edge.to) from a: edges[a]  static void dijkstra(int N, EdgeList[] edges, int initialNode, int goal) {  int[] best = new int[N];  Arrays.fill(best, Integer.MAX\_VALUE);  Queue<Path> Q = new PriorityQueue<>(Path.C);  best[initialNode] = 0;  Q.add(new Path(initialNode, 0));  while (Q.size() > 0) {  Path p = Q.poll();  if (p.dist >= best[goal]) {  break;  }  if (p.dist > best[p.node]) {  continue;  }  for (Edge e : edges[p.node]) {  int nd = p.dist + e.length;  if (nd < best[e.to]) {  best[e.to] = nd;  Q.add(new Path(e.to, nd));  }  }  }  // now use best[goal] for the distance to target  } | |
| --- | --- |

### Bellman-Ford (Single source shortest path)

| public class BellmanFord{  LinkedList<Edge> edges;  int d[], p[];  int n, e, s;  final int INFINITY = Integer.MAX\_VALUE;  private static class Edge {  int u, v, w;  public Edge(int a, int b, int c) {  u = a;  v = b;  w = c;  }  }  BellmanFord() throws IOException {  int item;  edges = new LinkedList<Edge>();  BufferedReader inp = new BufferedReader(new InputStreamReader(System.in));  System.out.print("Enter number of vertices ");  n = Integer.parseInt(inp.readLine());  System.out.println("Cost Matrix");  for (int i = 0; i < n; i++) {  for (int j = 0; j < n; j++) {  item = Integer.parseInt(inp.readLine());  if (item != 0)  edges.add(new Edge(i, j, item));  }  }  e = edges.size();  d = new int[n];  p = new int[n];  System.out.print("Enter the source vertex ");  s = Integer.parseInt(inp.readLine());  }  void relax() {  int i, j;  for (i = 0; i < n; ++i) {  d[i] = INFINITY;  p[i] = -1;  }  d[s] = 0;  for (i = 0; i < n - 1; ++i) {  for (j = 0; j < e; ++j) { // here i am calculating the shortest path  if (d[edges.get(j).u] + edges.get(j).w < d[edges.get(j).v]) {  d[edges.get(j).v] = d[edges.get(j).u] + edges.get(j).w;  p[edges.get(j).v] = edges.get(j).u;  }  }  }  }  boolean cycle() {  int j;  for (j = 0; j < e; ++j)  if (d[edges.get(j).u] + edges.get(j).w < d[edges.get(j).v])  return false;  return true;  }  void print() {  for (int i = 0; i < n; i++) {  System.out.println("Vertex " + i + " has predecessor " + p[i]);  }  }  public static void main(String args[]) throws IOException {  BellmanFord r = new BellmanFord();  r.relax();  if (r.cycle()) {  for (int i = 0; i < r.n; i++)  System.out.println(r.s + " ==> " + r.d[i]);  } else {  System.out.println(" There is a negative edge cycle ");  }  r.print();  }  } |
| --- |

### 

### Floyd-Warshall (all-pairs shortest path)

| void FloydWarshall(Graph graph) {  for (int i = 0; i < graph.numVertices; i++) {  for (int j = 0; j < graph.numVertices; j++) {  if (i == j) {  graph.path[i][j] = 0;  } else {  if (graph.path[i][j] == 0) {  graph.path[i][j] = Graph.LARGE;  }  if (graph.path[j][i] == 0) {  graph.path[j][i] = Graph.LARGE;  }  }  }  }  for (int k = 0; k < graph.numVertices; k++) {  for (int i = 0; i < graph.numVertices; i++) {  for (int j = 0; j < graph.numVertices; j++) {  System.out.println("k = " + k + ", i = " + i + ", j = " + j);  System.out.println("path[i][j] = " + graph.path[i][j]);  System.out.println("path[i][k] + path[k][j] = " + graph.path[i][k] + " + " + graph.path[k][j]);  graph.path[i][j] = Math.min(graph.path[i][j], graph.path[i][k] + graph.path[k][j]);  }  }  }  // graph.path contains the shortest path from each node to each node  // if value equal to LARGE, impossible to reach  } | class Graph {  final static int MAX\_VERTS = 1000;  final static int LARGE = Integer.MAX\_VALUE - 3000;  int path[][];  int numVertices;  Graph() {  path = new int[MAX\_VERTS][];  for (int i = 0; i < MAX\_VERTS; i++) {  path[i] = new int[MAX\_VERTS];  }  }  void AddEdge(int a, int b, int capacity) {  this.path[a][b] = capacity;  }  }  class Edge {  int s, t, r;  } |
| --- | --- |

### 

### 

### Minimum Spanning Tree (using Prim’s)

| public class Prim {  // Prim-Jarník's algorithm to find MST rooted at s  public static int[] prim(WeightedGraph G, int s) {  final int[] dist = new int[G.size()]; // shortest known distance to MST  final int[] pred = new int[G.size()]; // preceding node in tree  final boolean[] visited = new boolean[G.size()]; // all false initially  for (int i = 0; i < dist.length; i++) {  dist[i] = Integer.MAX\_VALUE;  }  dist[s] = 0;  for (int i = 0; i < dist.length; i++) {  final int next = minVertex(dist, visited);  visited[next] = true;  // The edge from pred[next] to next is in the MST (if next!=s)  final int[] n = G.neighbors(next);  for (int j = 0; j < n.length; j++) {  final int v = n[j];  final int d = G.getWeight(next, v);  if (dist[v] > d) {  dist[v] = d;  pred[v] = next;  }  }  }  return pred; // (ignore pred[s]==0!)  }  private static int minVertex(int[] dist, boolean[] v) {  int x = Integer.MAX\_VALUE;  int y = -1; // graph not connected, or no unvisited vertices  for (int i = 0; i < dist.length; i++) {  if (!v[i] && dist[i] < x) {  y = i;  x = dist[i];  }  }  return y;  }  } |
| --- |

# 

# 

# Datastructures

## Stack / Priority Queue / HashSet / HashMap

| Stack |
| --- |
| Stack <Integer > S = new Stack <Integer >();  S.push(num); // put 'num' on stack  S.peek(); // return top elem  S.pop(); // return and remove top elem |
| HashSet |
| HashSet <Integer > set;  set = new HashSet<Integer >();  Iterator <Integer > iter = set.iterator();  int i;  while (iter.hasNext()) {  i = iter.next();  System.out.println(i + " => " + set.contains(i));  } // prints 1 => true, 2 => true, .... |
| Priority Queue (0 = highest priority) |
| PriorityQueue <Pair<Integer , String >> Q;  Q = new PriorityQueue <Pair<Integer , String >>();  Q.add(new Pair<Integer , String >(valOfStr, "StringHere"));  Q.remove(); // return and remove highest priority |
| HashMap, map integer to list of integers |
| HashMap <Integer , ArrayList <Integer >> map;  map = new HashMap <Integer , ArrayList <Integer >>();  for (int i = 0; i < n; i++) {  map.put(i, new ArrayList <Integer >()); // at i, put new list  map.get(i); // get value at i,  } |

## 

## 

## 

## Pair

| // implements pairing support in Java of types E and F (e.g. string, integer, list, ..)  class Pair<E, F> implements Comparable<Pair<E, F>> {  E first;  F second;  Pair(E first, F second) {  super();  this.first = first;  this.second = second;  }  public boolean equals(Object other) {  if (other instanceof Pair) {  Pair otherPair = (Pair) other;  return ((this.first == otherPair.first  || (this.first != null && otherPair.first != null && this.first.equals(otherPair.first)))  && (this.second == otherPair.second || (this.second != null && otherPair.second != null  && this.second.equals(otherPair.second))));  }  return false;  }  public int compareTo(Pair<E, F> otherP) {  if (first instanceof Comparable) {  final int k = ((Comparable<E>) first).compareTo(otherP.first);  if (k > 0) {  return 1;  }  if (k < 0) {  return -1;  }  }  if (second instanceof Comparable) {  final int k = ((Comparable<F>) second).compareTo(otherP.second);  if (k > 0) {  return 1;  }  if (k < 0) {  return -1;  }  }  return 0;  }  } |
| --- |

## 

## 

## UnionFind

| public class UF {  static void unionFindInit(int N) {  root = new int[N];  rank = new int[N];  for (int i = 1; i < N; i++) {  root[i] = i;  } // now use find(x) and merge(x, y)  }  static int[] root, rank; // find the group x belongs to  static int find(int x) {  if (root[x] != x) {  root[x] = find(root[x]);  }  return root[x];  } // merge the groups xand y belong to  static void merge(int x, int y) {  x = find(x);  y = find(y);  if (x == y) {  return;  }  if (rank[x] < rank[y]) {  root[x] = y;  } else if (rank[x] >= rank[y]) {  root[y] = x;  if (rank[x] == rank[y]) {  rank[x]++;  }  }  }  } |
| --- |

# Strings & Sequences

| Stringbuilder >>> Using a normal string and adding manually (speed) |
| --- |
| // Create a new StringBuilder  StringBuilder sb = new StringBuilder();  // Add stuff to the string  sb.append("strings here, always include a newline when needed \n");  System.out.println(sb.toString()); |

### 

### 

## Check for duplicate characters in a string and print them.

| public class FindDuplicateCharacters {  public static void main(String args[]) {  printDuplicateCharacters("Programming");  printDuplicateCharacters("Combination");  printDuplicateCharacters("Java");  }  /\*  \* Find all duplicate characters in a String and print each of them.  \*/  public static void printDuplicateCharacters(String word) {  char[] characters = word.toCharArray();  // build HashMap with character and number of times they appear in  // String  Map<Character, Integer> charMap = new HashMap<Character, Integer>();  for (Character ch : characters) {  if (charMap.containsKey(ch)) {  charMap.put(ch, charMap.get(ch) + 1);  } else {  charMap.put(ch, 1);  }  }  // Iterate through HashMap to print all duplicate characters of String  Set<Map.Entry<Character, Integer>> entrySet = charMap.entrySet();  System.out.printf("List of duplicate characters in String '%s' %n", word);  for (Map.Entry<Character, Integer> entry : entrySet) {  if (entry.getValue() > 1) {  System.out.printf("%s : %d %n", entry.getKey(), entry.getValue());  }  }  }  } |
| --- |

## Check if a string is an anagram

| boolean isAnagram(String word, String anagram) {  char[] charFromWord = word.toLowerCase().toCharArray();  char[] charFromAnagram = anagram.toLowerCase().toCharArray();  Arrays.sort(charFromWord);  Arrays.sort(charFromAnagram);  return Arrays.equals(charFromWord, charFromAnagram);  } |
| --- |

### 

### 

## Print all permutations of a **string** to the error output.

| public class test {  public static void main(String args[]) {  permutation("XYZ");  }  public static void permutation(String input) {  permutation("", input);  }  private static void permutation(String perm, String word) {  if (word.isEmpty()) {  System.err.println(perm + word);  } else {  for (int i = 0; i < word.length(); i++) {  permutation(perm + word.charAt(i), word.substring(0, i) + word.substring(i + 1, word.length()));  }  }  }  } |
| --- |

## Check if a string is a palindrome

| boolean isPalindromString(String text) {  String reverse = reverse(text);  if (text.equals(reverse)) {  return true;  }  return false;  }  static String reverse(String input) {  if (input == null || input.isEmpty()) {  return input;  }  return input.charAt(input.length() - 1) + reverse(input.substring(0, input.length() - 1));  } |
| --- |

## Knuth-Morris-Pratt (=Given a string S, find all occurrences of S in a big string)

| public class Main {  public int[] preProcessPattern(char[] ptrn) {  int i = 0, j = -1;  int ptrnLen = ptrn.length;  int[] b = new int[ptrnLen + 1];  b[i] = j;  while (i < ptrnLen) {  while (j >= 0 && ptrn[i] != ptrn[j]) {  // if there is mismatch consider the next widest border  // The borders to be examined are obtained in decreasing order  // from the values b[i], b[b[i]] etc.  j = b[j];  }  i++;  j++;  b[i] = j;  }  return b;  }  } |
| --- |

# 

# Dynamic Programming

## Longest Common Subsequence

| /\* Dynamic Programming Java implementation of LCS problem \*/  public class LCS {  /\* Returns length of LCS for X[0..m-1], Y[0..n-1] \*/  int lcs(char[] X, char[] Y, int m, int n) {  int L[][] = new int[m + 1][n + 1];  /\*  \* Following steps build L[m+1][n+1] in bottom up fashion. Note that  \* L[i][j] contains length of LCS of X[0..i-1] and Y[0..j-1]  \*/  for (int i = 0; i <= m; i++) {  for (int j = 0; j <= n; j++) {  if (i == 0 || j == 0)  L[i][j] = 0;  else if (X[i - 1] == Y[j - 1])  L[i][j] = L[i - 1][j - 1] + 1;  else  L[i][j] = max(L[i - 1][j], L[i][j - 1]);  }  }  return L[m][n];  }  /\* Utility function to get max of 2 integers \*/  int max(int a, int b) {  return (a > b) ? a : b;  }  public static void main(String[] args) {  LCS lcs = new LCS();  String s1 = "AGGTAB";  String s2 = "GXTXAYB";  char[] X = s1.toCharArray();  char[] Y = s2.toCharArray();  int m = X.length;  int n = Y.length;  System.out.println("Length of LCS is" + " " + lcs.lcs(X, Y, m, n));  }  } |
| --- |

## Venus Rover (we solved this together)

| /\*\*  \* @author Daniel, Nihal, Storm; Problem: EAPC 2005 H - Venus Rover  \*/  public class venusrover {  // main method, run the program  public static void main(String[] args) throws IOException {  (new venusrover()).run();  }  void run() throws IOException {  // A buffered reader to read and writer to write.  BufferedReader r = new BufferedReader(new InputStreamReader(System.in));  BufferedWriter w = new BufferedWriter(new OutputStreamWriter(System.out));  int cases = Integer.parseInt(r.readLine());  while (cases-- > 0) {  StringTokenizer s = new StringTokenizer(r.readLine());  int numStones = Integer.parseInt(s.nextToken());  int timeLeft = Integer.parseInt(s.nextToken());  int maxCapacity = Integer.parseInt(s.nextToken());  // Store the time, mass and value of all stones in their own array  int[] stoneTime = new int[numStones];  int[] stoneMass = new int[numStones];  int[] stoneVal = new int[numStones];  for (int i = 0; i < numStones; i++) {  s = new StringTokenizer(r.readLine());  stoneTime[i] = Integer.parseInt(s.nextToken());  stoneMass[i] = Integer.parseInt(s.nextToken());  stoneVal[i] = Integer.parseInt(s.nextToken());  }  /\*\*  \* F(N,M,T) denotes the optimal profit for taking N stones, with  \* weight M and time to return T  \*/  int[][][] table = new int[numStones + 10][maxCapacity + 10][timeLeft + 10];  // Fill the table  for (int i = 0; i <= (numStones); i++) {  for (int m = 0; m <= (maxCapacity); m++) {  for (int t = 0; t <= (timeLeft); t++) {  if ((i == 0) || (m == 0) || (t == 0)) {  table[i][m][t] = 0;  } else if ((stoneTime[i - 1] <= t) && (stoneMass[i - 1] <= m)) {  table[i][m][t] = Math.max(table[i - 1][m][t],  table[i - 1][m - stoneMass[i - 1]][t - stoneTime[i - 1]] + stoneVal[i - 1]);  } else {  table[i][m][t] = table[i - 1][m][t];  }  }  }  }  w.write(table[numStones][maxCapacity][timeLeft] + "\n");  }  w.flush();  }  } |
| --- |

## Knapsack 0/1

| static List<Integer> knapsack(int capacity, int n, int[] values, int[] weights) {  int[][] bestValue = new int[n + 1][capacity + 1];  for (int i = 1; i <= n; i++) {  int itemIndex = i - 1;  for (int j = 0; j <= capacity; j++) {  int weightIfNotIncluded = bestValue[i - 1][j];  if (j >= weights[itemIndex]) {  int weightIfIncluded = bestValue[i - 1][j - weights[itemIndex]] + values[itemIndex];  bestValue[i][j] = Main.max(weightIfIncluded, weightIfNotIncluded);  } else {  bestValue[i][j] = weightIfNotIncluded;  }  }  }  // you can check the bestValue[n][capacity] here for example  // backtrack the chosen items (optional)  List<Integer> chosen = new ArrayList<>();  int j = capacity;  for (int i = n; i > 0; i--) {  if (bestValue[i][j] != bestValue[i - 1][j]) {  int itemIndex = i - 1;  chosen.add(itemIndex);  j -= weights[itemIndex];  }  }  return chosen;  }  static int max(int a, int b) {  return (a > b) ? a : b;  } |
| --- |

**Common problems with DP implementation**

* The dimensions of the table in a DP should be 1 bigger than the actual values, since the 0th row and columns are initial values.
* The loopbounds when filling in the table should be "i <= X", not "i < X"
* We should account for indexing offset (in venusrover: in stoneTime, stoneMass and stoneVal) in the recursion.

# 

# 

# Golden Rules and Quick Fixes

1. ***Think before you code***
2. ***Always choose the simplest solution that is fast enough***
3. ***Code carefully rather than fast***

Wrong Answer?: Check:

* Loop bounds / array bounds / initialization / output correctness / nesting / precision / overflow / invalid expressions / index offset / rounding / reading the input / boundary cases / copying mistakes.

Runtime error? Check:

* Runtime error means that there was an error. Go through the code and see where the error may occur (e.g. NullPointer with array bounds / invalid expressions / out of memory (shouldn't happen))

Time-limit-exceeded? Check:

* Algortihm speed? / Infinite loop? Simple optimizations?

# Eclipse Setup

**Window > Preferences > Editor > Save-Actions > Auto-imports**

**Window > Preferences > Java > Editor > Templates > new > enter template there:**

The reason for following template: We do not need to work in the main, hence bothering making things static. Furthermore we can use auxiliary function easily. Secondly: Use your favourite reading and writing tools!

| Our template we will use while coding. |
| --- |
| import java.io.BufferedReader;  import java.io.BufferedWriter;  import java.io.IOException;  import java.io.InputStreamReader;  import java.io.OutputStreamWriter;  import java.util.Scanner;  public class Main {  public static void main(String[] args) throws IOException {  (new Main()).run();  }  void run() throws IOException {  BufferedReader r = new BufferedReader(new InputStreamReader(System.in));  BufferedWriter w = new BufferedWriter(new OutputStreamWriter(System.out));  Scanner sc = new Scanner(System.in);  StringBuilder sb = new StringBuilder();  sb.append("\n");  w.write(sb.toString());  // Note: A bufferedWriter only takes strings, so add "\n" to the end for a new line!!  w.flush(); // Needed for bufferedWriter  }  } |

**Important shortcuts:**

To use the above made template, type a 't' (the character t), then press **CTRL+SPACE.** Eclipse will automatically put down above template. Another very useful tool in Eclipse is multi-renaming. This is done by the combination **ALT+SHIFT+R**.Furthermore, you can automatically clean code with **CTRL+SHIFT+F**.