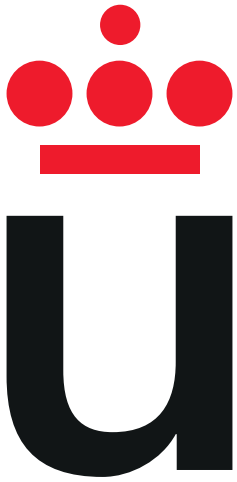

Universidad Rey Juan Carlos

Abelian Group

Alejandro García Carretero, Laura Medina Henche, Yihui Xia

AdaByron 2020

February 29, 2020

Contest (1)

content.txt

10 lines

```
Contest (1)
Data structures (1)
Graph (2)
Strings (5)
Mathematics (5)
Geometry (6)
Numerical (8)
Number theory (8)
Combinatorial (9)
Various (11)
```

Data structures (2)

SegmentTree.java

Description: Segment Tree structure. Used to store information about intervals (Sum of sub-intervals for example)

63 lines

```
class SegmentTree{
    int st[]; // The array that stores segment tree
        nodes

    SegmentTree(int arr[], int n){
        int x = (int) (Math.ceil(Math.log(n) / Math.log
            (2)));
        int max_size = 2 * (int) Math.pow(2, x) - 1;
        st = new int[max_size];
        constructSTUtil(arr, 0, n - 1, 0);
    }
    //Get the middle index from corner indexes.
    int getMid(int s, int e) {
        return s + (e - s) / 2;
    }

    int getSumUtil(int ss, int se, int qs, int qe, int
        si){
        if (qs <= ss && qe >= se)
            return st[si];
        if (se < qs || ss > qe)
            return 0;
        int mid = getMid(ss, se);
        return getSumUtil(ss, mid, qs, qe, 2 * si + 1)
            +
                getSumUtil(mid + 1, se, qs, qe, 2 * si
                    + 2);
    }
    void updateValueUtil(int ss, int se, int i, int
        diff, int si){
        if (i < ss || i > se)
            return;
        st[si] = st[si] + diff;
        if (se != ss) {
            int mid = getMid(ss, se);
            updateValueUtil(ss, mid, i, diff, 2 * si +
                1);
            updateValueUtil(mid + 1, se, i, diff, 2 *
                si + 2);
        }
    }

    // Updates a value in input array and segment tree.
    void updateValue(int arr[], int n, int i, int
        new_val){
        if (i < 0 || i > n - 1) {
            return;
        }
        int diff = new_val - arr[i];
        arr[i] = new_val;
        updateValueUtil(0, n - 1, i, diff, 0);
    }

    int getSum(int n, int qs, int qe){
        if (qs < 0 || qe > n - 1 || qs > qe) {
            return -1;
        }
        return getSumUtil(0, n - 1, qs, qe, 0);
    }

    // Constructs Segment Tree for array[ss..se].
    int constructSTUtil(int arr[], int ss, int se, int
        si){
        if (ss == se) {
            st[si] = arr[ss];
            return arr[ss];
        }
        int mid = getMid(ss, se);
        st[si] = constructSTUtil(arr, ss, mid, si * 2 +
            1) +
                constructSTUtil(arr, mid + 1, se, si *
                    2 + 2);
        return st[si];
    }
}
```

Searching and sorting (3)

ArrayInversion.java

Description: Indicates how far (or close) the array is from being sorted. If array is already sorted then inversion count is 0. If array is sorted in reverse order that inversion count is the maximum.

Time: $\mathcal{O}(NlogN)$

42 lines

```
import java.util.Arrays;
public class ArrayInversion {
    // Function to count the number of inversions
    // during the merge process
    private static int mergeAndCount(int[] arr, int l,
        int m, int r){
        // Left subarray
        int[] left = Arrays.copyOfRange(arr, l, m + 1);
        // Right subarray
        int[] right = Arrays.copyOfRange(arr, m + 1, r
            + 1);
        int i = 0, j = 0, k = l, swaps = 0;
        while (i < left.length && j < right.length) {
            if (left[i] <= right[j])
                arr[k++] = left[i++];
            else {
                arr[k++] = right[j++];
                swaps += (m + 1) - (l + i);
            }
        }
        // Fill from the rest of the left subarray
        while (i < left.length)
            arr[k++] = left[i++];
        // Fill from the rest of the right subarray
        while (j < right.length)
            arr[k++] = right[j++];
        return swaps;
    }
    // Merge sort function
    private static int mergeSortAndCount(int[] arr, int
         l, int r){
        int count = 0;
        if (l < r) {
            int m = (l + r) / 2;
            count += mergeSortAndCount(arr, l, m);
            count += mergeSortAndCount(arr, m + 1, r);
            count += mergeAndCount(arr, l, m, r);
        }
        return count;
    }
/*To run:
    int[] arr = { 1, 20, 6, 4, 5 }; (The data)
    System.out.println(mergeSortAndCount(arr, 0, arr.
        length - 1));
 */
}
```

BinarySearch.java

Description: Finds an element in an array(sorted)

Usage: `BinarySearch ob = new BinarySearch();`
`ob.binarySearch(arr, key);`

Time: $\mathcal{O}(log3N)$

15 lines

```
public class BinarySearch {
    int binarySearch(int arr[], int x){
        int l = 0, r = arr.length - 1;
        while (l <= r) {
            int m = l + (r - l) / 2;
            if (arr[m] == x)
                return m;
            if (arr[m] < x)
                l = m + 1;
            else
                r = m - 1;
        }
        return -1;
    }
```

```
}
```

TernarySearch.java
Description: Finds an element in an array(sorted)
Usage: `l=starting index, r=length of arr`
`ternarySearch(l, r, key, arr);`
Time: $\mathcal{O}(log3N)$

24 lines

```
public class TernarySearch {
    static int ternarySearch(int l, int r, int key, int
         ar[]){
        if (r >= l) {
            int mid1 = l + (r - l) / 3;
            int mid2 = r - (r - l) / 3;
            if (ar[mid1] == key) {
                return mid1;
            }
            if (ar[mid2] == key) {
                return mid2;
            }
            if (key < ar[mid1]) {
                return ternarySearch(l, mid1 - 1, key,
                    ar);
            }
            else if (key > ar[mid2]) {
                return ternarySearch(mid2 + 1, r, key,
                    ar);
            }
            else {
                return ternarySearch(mid1 + 1, mid2 -
                    1, key, ar);
            }
        }
        return -1;
    }
}
```

Graph (4)

DFS.java
Description: DFS

18 lines

```
public class DFS {
    public static void DFS(ArrayList<Arista>[] grafo,
        int N){
        ArrayDeque<State> pila = new ArrayDeque<>();
        boolean visitado[] = new boolean[N];
        pila.push(new State(0,0));
        visitado[0] = true;
        while(!pila.isEmpty()){
            State aux = pila.pop();
            for(Arista ady: grafo[aux.nodo]){
                int destino = ady.to;
                if(!visitado[destino]){
                    pila.push(new State(destino,0));
                    visitado[destino] = true;
                }
            }
        }
    }
}
```

DFSt.java
Description: Traverse DFS

19 lines

```
public class DFSt {
    public static void DFSt(int[][] grafo,int N,int
        inicio,boolean visitado[]){
        ArrayDeque<Integer> pila = new ArrayDeque<>();
        pila.push(inicio);
        visitado[inicio] = true;
        while(!pila.isEmpty()){
            int aux = pila.pop();
            for(int ady=0;ady<N;ady++){
                if(grafo[ady][aux]>0){
                    int destino = ady;
                    if(!visitado[destino]){
                        pila.push(destino);
                        visitado[destino] = true;
                    }
                }
            }
        }
    }
}
```

DijkstraList.java
Description: Shortest path from i to j in O(VLg(E)) with list implementation of graphs.
Time: $\mathcal{O}(E + Vlog(V))$

27 lines

```
public static int dijsktra(ArrayList<Arista>[] graph,
    int from, int to){
    PriorityQueue<State> queue = new PriorityQueue<>();
    int[] distancia = new int[graph.length];
    for (int i = 0; i < graph.length; i++) {
        distancia[i] = INF;
    }
    queue.offer(new State(from, 0));
    distancia[from] = 0;

    while(!queue.isEmpty()){
        State actual = queue.poll();

        //si solo queremos ir de un lado a otro
        if(actual.node == to)
            return actual.dist;

        for(Arista ady: graph[actual.node]){
            // if(!visitado[actual.nodo]) y sacar el
                offer del 2o if
            if (distancia[ady.to] >= ady.distancia +
                actual.dist) {
                distancia[ady.to] = ady.distancia +
                    actual.dist;
                queue.offer(new State(ady.to, distancia
                    [ady.to]));
            }
        }
        //visitado[actual.nodo] = true;
    }
    return -1;
}
```

TopoSort.java
Description: Returns a sorted list of nodes using topological sort. Useful when ordering certain tasks (breakfast)
Time: $\mathcal{O}(E + V)$

25 lines

```
public class TopoSort {
    public static void topoUtil(int[][] grafo, int
        actual, boolean[] visitado, ArrayDeque<Integer>
         pila){
        visitado[actual] = true;
        for (int i = 0; i < grafo.length; i++) {
            if(grafo[actual][i]>0 && !visitado[i]){
                topoUtil(grafo,i,visitado,pila);
            }
        }
        pila.push(actual);
    }
    public static ArrayDeque<Integer> topoSort(int[][]
        grafo){
        ArrayDeque<Integer> pila = new ArrayDeque<>();
        ArrayDeque<Integer> output = new ArrayDeque<>()
            ;
        boolean visitado[] = new boolean[grafo.length];
        for (int i = 0; i < grafo.length; i++) {
            if(!visitado[i]){
                topoUtil(grafo,i,visitado,pila);
            }
            while(!pila.isEmpty()){
                output.offer(pila.pop());
            }
        }
        return output;
    }
}
```

UnionFind.java
Description: Union find structure

86 lines

```
public class UnionFind {
    /*STRUCTURE*/
    int N;

    int parent[];
    int size[];
    public UnionFind(int N){
        this.N = N;
        this.parent = new int[N];
        this.size = new int[N];
    }
```

```
    public void init(){
        for (int i = 0; i < N; i++) {
            parent[i] = i;
            size[i] =1;
        }
    }
    public int find(int nodo){
        while(nodo!=parent[nodo]){
            nodo = parent[nodo];
        }
        return nodo;
    }
    public int getSize(int nodo){
        return this.size[nodo];
    }
    public void join(int A,int B){
        A = find(A);
        B = find(B);
        if(size[A]<size[B]){
            parent[A] = B;
            size[B]+=size[A];
        }else{
            parent[B] = A;
            size[A]+=size[B];
        }
    }
    public boolean isConnected(int A, int B){
        return (find(A) == find(B));
    }
    public int connect(int A, int B){//0 ya estaban
        conectados, 1 conectado correctamente
        if(isConnected(A,B)){
            return 0;
        }else{
            join(A,B);
            return 1;
        }
    }
}
class Arista implements Comparable<Arista>{
    int from,to,peso;
    public Arista(int from, int to, int peso) {
        this.from = from;
        this.to = to;
        this.peso = peso;
    }
    @Override
    public int compareTo(Arista o) {
        return this.peso-o.peso;
    }
    @Override
    public String toString(){
        return "[" + this.from + "->" + this.to +"||"+
            this.peso + "]";
    }
}
public class UF { /*Main class*/
    public static void main(String args[]){
        Scanner scn = new Scanner(System.in);
        int N = scn.nextInt();
        int M = scn.nextInt();
        ArrayList<Arista>[] grafo = new ArrayList[N];
        PriorityQueue<Arista> grafoK = new
            PriorityQueue<>();
        for (int i = 0; i < N; i++) {
            grafo[i] = new ArrayList<>();
        }
        for (int i = 0;i<M;i++){
            int from = scn.nextInt();
            int to = scn.nextInt();
            int peso = scn.nextInt();
            grafo[from].add(new Arista(from,to,peso));
            grafo[to].add(new Arista(to,from,peso));
            grafoK.offer(new Arista(from,to,peso));
        }
        System.out.println(KuruskalRM(grafoK,N));
    }
}
```

Kuruskal.java

Description: Finds minimum spanning tree
Time: $\mathcal{O}(E * log(E))$

25 lines

```
public class Kuruskal {
    public static int KuruskalCC(PriorityQueue<Arista>
        grafo,int N){//Cuenta componentes Conexas
        UnionFind uf = new UnionFind(N);
        uf.init();
        while (N>1 && !grafo.isEmpty()){
            Arista aux = grafo.poll();
            N -= uf.connect(aux.from,aux.to);
        }
        return N;
    }
    public static int KuruskalRM(PriorityQueue<Arista>
        grafo, int N){//Version camino minimo
        UnionFind uf = new UnionFind(N);
        uf.init();
        int pesos = 0;
        while (N>1 && !grafo.isEmpty()){
            Arista aux = grafo.poll();
            if(uf.connect(aux.from,aux.to) == 1){
                pesos+=aux.peso;
                N--;
            }
        }

        return pesos;
    }
}
```

FloydWarshall.java

Description: Shortest path from i to j in a graph.
Time: $\mathcal{O}(N^3)$!

9 lines

```
public class FloydWarshall {
    for(int k = 0; k<N; k++){
        for(int i = 0; i<N;i++){
            for(int j = 0; j<N;j++){
                grafo[i][j] = Math.min(grafo[i][j],
                    grafo[i][k] + grafo[k][j]);
            }
        }
    }
}
```

SCC.java

Description: Returns the number of SCC in a graph (Every node can travel to any other in the component)

14 lines

```
public class SCC {
    public static int scc(int[][] grafo,int N) {
        boolean visitado[] = new boolean[N];
        ArrayDeque<Integer> cola = topoSort(grafo);
        int cuenta = 0;
        for (int i : cola) {
            if (!visitado[i]) {
                DFSt(grafo, N, i, visitado);
                cuenta++;
            }
        }
        return cuenta;
    }
}
```

CompConexas.java

Description: Returns how many CC there are on a graph

14 lines

```
public class CompConexas {
    //Modify dfs, passing visitado[] and the initial
        node
    public static int cc(ArrayList<Arista>[] grafo,int
        N){
        boolean visitado[] = new boolean[N];
        int cuenta = 0;
        for (int i = 0; i < N; i++) {
            if(!visitado[i]){
                DFS(grafo,N,i,visitado);
                cuenta++;
            }
        }
        return cuenta;
    }
}
```

Prim.java

Description: Finds minimum spanning tree using heaps

34 lines

```
public class Prim {
    //Previo
    class State{ int nodo, distancia; }
    class Arista{ int from,to,peso; }
    ArrayList<Arista>[] grafo = new ArrayList[N];
    grafo[ini].add(new Arista(ini,fin,coste));
```

```
    grafo[fin].add(new Arista(fin,ini,coste));
    //Algoritmo
    PriorityQueue<State> cola = new PriorityQueue<>();
    boolean visitado[] = new boolean[N];
    int suma = 0;
    cola.offer(new State(0,0));
    while(!cola.isEmpty()){
        State aux = cola.poll();
        if(!visitado[aux.nodo]){
            suma+=aux.distancia;
            visitado[aux.nodo] = true;
            for (int i = 0; i < grafo[aux.nodo].size();
                 i++) {
                int destino = grafo[aux.nodo].get(i).to
                    ;
                int peso = grafo[aux.nodo].get(i).peso;
                if(!visitado[destino]){
                    cola.offer(new State(destino,peso))
                        ;
                }
            }
        }
    }
    int f = 0;
    for (int i = 0; i < N; i++) {
        if(!visitado[i]){
            f = 1;
            break;
        }
    }
}
```

Hopcroft.java

Description: Hopcroft algorithm for maxflow

Time: $\mathcal{O}(EV^3)$

68 lines

```
public class Hopcroft {
    boolean bfs(int rGraph[][], int s, int t, int
        parent[]){
        // Create a visited array and mark all vertices
             as not
        // visited
        boolean visited[] = new boolean[V];
        for(int i=0; i<V; ++i)
            visited[i]=false;
        // Create a queue, enqueue source vertex and
            mark
        // source vertex as visited
        LinkedList<Integer> queue = new LinkedList<
            Integer>();
        queue.add(s);
        visited[s] = true;
        parent[s]=-1;
        // Standard BFS Loop
        while (queue.size()!=0){
            int u = queue.poll();
            for (int v=0; v<V; v++){
                if (visited[v]==false && rGraph[u][v] >
                     0){
                    queue.add(v);
                    parent[v] = u;
                    visited[v] = true;
                }
            }
        }
        // If we reached sink in BFS starting from
            source, then
        // return true, else false
        return (visited[t] == true);
    }
    // Returns tne maximum flow from s to t in the
        given graph
    int fordFulkerson(int graph[][], int s, int t){
        int u, v;
        // Create a residual graph and fill the
            residual graph
        // with given capacities in the original graph
            as
        // residual capacities in residual graph
        // Residual graph where rGraph[i][j] indicates
        // residual capacity of edge from i to j (if
            there
        // is an edge. If rGraph[i][j] is 0, then there
             is
        // not)
        int rGraph[][] = new int[V][V];
        for (u = 0; u < V; u++)
            for (v = 0; v < V; v++)
                rGraph[u][v] = graph[u][v];
        // This array is filled by BFS and to store
            path
        int parent[] = new int[V];
        int max_flow = 0;  // There is no flow
            initially
        // Augment the flow while tere is path from
            source
        // to sink
        while (bfs(rGraph, s, t, parent)){
            // Find minimum residual capacity of the
                edges
            // along the path filled by BFS. Or we can
                say
            // find the maximum flow through the path
                found.
            int path_flow = Integer.MAX_VALUE;
            for (v=t; v!=s; v=parent[v]){
                u = parent[v];
                path_flow = Math.min(path_flow, rGraph[
                    u][v]);
            }
            // update residual capacities of the edges
                and
            // reverse edges along the path
            for (v=t; v != s; v=parent[v]){
                u = parent[v];
                rGraph[u][v] -= path_flow;
                rGraph[v][u] += path_flow;
            }
            max_flow += path_flow;
        }
        return max_flow;
    }
}
```

4.0.1 Eulerian and Hamiltonian

- **Eulerian path:** only if C(V) is even or C(V) even with two C(V) odd. It means if we can go through all the graph's edges only one time.

- **Eulerian cycle:** only if C(V) is even. It means we can go through all the graph's edges only one time and finish where we started.

HamiltonianWay.java

Description: Returns if there is a hamiltonian way

31 lines

```
public class HamiltonianWay {
    public static boolean hamiltonianWay(ArrayList<
        Arista>[] grafo, int N, int init_node){
        boolean isHamiltonian = false;
        int mask = (2^N)-1;
        int aux;
        //BFS
        ArrayDeque<State> cola = new ArrayDeque<>();
        boolean[] visitado = new boolean[N];
        cola.offer(new State(init_node,0));
        visitado[0] = true;
        while(!cola.isEmpty()){
            State actual = cola.poll();
            aux = 1<<actual.nodo;
            if((mask & aux) > 0){
                mask ^= aux;
            }else{
                isHamiltonian = true;
                break;
            }
            for(Arista ady: grafo[actual.nodo]){
                int destino = ady.to;
                if(!visitado[destino]){
                    cola.offer(new State(destino, 0));
                    visitado[destino] = true;
                }
            }
        }
        // Para ver si es ciclo simplemente ver que el
            nodo actual es el mismo que el init_node
        return isHamiltonian;
    }
}
```

Strings (5)

Z.java

Description: Finds all occurrences of a pattern in a text in linear time.

38 lines

```
public class Z {
    public static void search(String text, String
        pattern) {
        String concat = pattern + "$" + text;
        int l = concat.length();
        int Z[] = new int[l];
        // Construct Z array
        getZarr(concat, Z);
        for (int i = 0; i < l; ++i) {
            if (Z[i] == pattern.length()) {
                System.out.println("Pattern found at
                    index "+ (i - pattern.length() - 1)
                    );
            }
        }
    }
    private static void getZarr(String str, int[] Z) {
        int n = str.length();
        int L = 0, R = 0;
        for (int i = 1; i < n; ++i) {
            if (i > R) {
                L = R = i;
                while (R < n && str.charAt(R - L) ==
                    str.charAt(R))
                    R++;
                Z[i] = R - L;
                R--;
            } else {
                int k = i - L;
                if (Z[k] < R - i + 1)
                    Z[i] = Z[k];
                else {
                    L = i;
                    while (R < n && str.charAt(R - L)
                        == str.charAt(R))
                        R++;
                    Z[i] = R - L;
                    R--;
                }
            }
        }
    }
}
```

Mathematics (6)

6.1 Some formulas

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2} \ (triangular\ numbers)$$

$$\sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}$$

$$\sum_{i=1}^{n} i^3 = (\frac{n(n+1)}{6})^2$$

$$\sum_{i=1}^{n} i^4 = \frac{n(n+1)(2n+1)(3n^2+3n-1)}{30}$$

$$Pyramid = \frac{n(n+1)(n+2)}{6}$$

$$\binom{a}{b} = \frac{a!}{b!(a-b)!}$$

$$C_n = \frac{(2n)!}{(n+1)!n!} \ (Catalan\ numbers)$$

6.2 Geometry

6.2.1 Quadrilaterals

Sides: a, b, c, d
Angles: A, B, C, D
Perimeter: P, semiperimeter: s
Area: K
Diagonals: p = BD, q = AC
magic flux $F = b^2 + d^2 - a^2 - c^2$, then:

$$4A = 2pq \cdot \sin\theta = F \tan\theta = \sqrt{4p^2q^2 - F^2}$$

For cyclic quadrilaterals the sum of opposite angles is 180°, $ef = ac + bd$, and
$K = \sqrt{(p-a)(p-b)(p-c)(p-d)}$.

6.2.2 Volumes

	Sphere	Cube	Tetrahedron
Area	$4\pi r^2$	$6a^2$	$a^2\sqrt{3}$
Volume	$\frac{4}{3}\pi r^3$	a^3	$\frac{1}{12}a^3\sqrt{2}$

	Octahedron	Dodecahedron	Icosahedron
Area	$2\sqrt{3}a^2$	$3a^2\sqrt{25+10\sqrt{5}}$	$5\sqrt{3}a^2$
Volume	$\frac{1}{3}\sqrt{2}a^3$	$\frac{1}{4}(15+7\sqrt{5})a^3$	$\frac{5}{12}(3+\sqrt{5})a^3$

6.2.3 Triangles

Side lengths: a, b, c
Semiperimeter: $p = \frac{a+b+c}{2}$
Area: $A = \sqrt{p(p-a)(p-b)(p-c)}$
Circumradius: $R = \frac{abc}{4A}$
Inradius: $r = \frac{A}{p}$
Length of median (divides triangle into two equal-area triangles): $m_a = \frac{1}{2}\sqrt{2b^2+2c^2-a^2}$
Length of bisector (divides angles in two):
$s_a = \sqrt{bc\left[1 - \left(\frac{a}{b+c}\right)^2\right]}$
Law of sines: $\frac{\sin\alpha}{a} = \frac{\sin\beta}{b} = \frac{\sin\gamma}{c} = \frac{1}{2R}$
Law of cosines: $a^2 = b^2 + c^2 - 2bc\cos\alpha$
Law of tangents: $\frac{a+b}{a-b} = \frac{\tan\frac{\alpha+\beta}{2}}{\tan\frac{\alpha-\beta}{2}}$

6.3 Probability theory

Let X be a discrete random variable with probability $p_X(x)$ of assuming the value x. It will then have an expected value (mean) $\mu = \mathbb{E}(X) = \sum_x xp_X(x)$ and variance
$\sigma^2 = V(X) = \mathbb{E}(X^2) - (\mathbb{E}(X))^2 = \sum_x (x - \mathbb{E}(X))^2 p_X(x)$
where σ is the standard deviation. If X is instead continuous it will have a probability density function $f_X(x)$ and the sums above will instead be integrals with $p_X(x)$ replaced by $f_X(x)$.

Expectation is linear:

$$\mathbb{E}(aX + bY) = a\mathbb{E}(X) + b\mathbb{E}(Y)$$

For independent X and Y,

$$V(aX + bY) = a^2V(X) + b^2V(Y).$$

6.3.1 Discrete distributions

Binomial distribution

The number of successes in n independent yes/no experiments, each which yields success with probability p is $\text{Bin}(n, p)$, $n = 1, 2, \ldots, 0 \leq p \leq 1$.

$$p(k) = \binom{n}{k} p^k (1-p)^{n-k}$$

$$\mu = np,\ \sigma^2 = np(1-p)$$

$\text{Bin}(n, p)$ is approximately $\text{Po}(np)$ for small p.

6.4 Catalan Numbers

- Number of correct bracket sequence consisting of n opening and n closing brackets.
- The number of rooted full (0 or 2 children) binary trees with n+1 leaves.
- The number of triangulations of a convex polygon with n+2 sides.
- sub-diagonal monotone paths in an $n \times n$ grid.
- The number of ways to connect the 2n points on a circle to form n disjoint chords.
- ordered trees with $n + 1$ vertices.
- permutations of $[n]$ with no 3-term increasing subseq.
- The number of non-crossing partitions of a set of n elements.

Catalan.java

Description: Return the nth catalan number

10 lines

```
public class Catalan {
    int catalan(int n) {
        int res = 0;
        if (n <= 1) { return 1; }
        for (int i = 0; i < n; i++) {
            res += catalan(i) * catalan(n - i - 1);
        }
        return res;
    }
}
```

6.5 Prime Numbers

Erathostenes.java

Description: Generating all primes up to a certain limit. limit of 100.000.000 elements.

13 lines

```
public class Erathostenes {
    public static void sieveOfEratosthenes(int n) {
        // Create a boolean array "prime[0..n]" and
            initialize
        // if prime[i] is false then it's prime
        boolean prime[] = new boolean[n + 1];
        for (int p = 2; p * p <= n; p++) {
            if (prime[p] == false) {
                for (int i = p * p; i <= n; i += p)
                    prime[i] = true;
            }
        }
    }
}
```

PrimeFactors.java

Description: Stores all prime factors of a given number n

21 lines

```
import java.lang.Math;
import java.util.ArrayList;

public class PrimeFactors {
    public static void primeFactors(int n){
        ArrayList<Integer> factors = new ArrayList<>();
        // Print the number of 2s that divide n
        while (n%2==0){
            factors.add(2);
            n /= 2;
        }
        for (int i = 3; i <= Math.sqrt(n); i+= 2){
            while (n%i == 0){
                factors.add(i);
                n /= i;
            }
        }
        if (n > 2)
            factors.add(n);
    }
}
```

Geometry (7)

7.1 Basic formulas

- Scalar: $(a_x \cdot b_x + a_y \cdot b_y)$
- Vectorial: $(a_x \cdot b_y - a_y \cdot b_x)$
- Angle: $atan2(a_y, a_x)$
- Perpend: $(-a_y, a_x)$

7.2 Geometric primitives

LineIntersec.java

Description: Returns if two lines intersect (p1p2, q1q2)

36 lines

```
public class LineIntersec {
    static boolean onSegment(Point p, Point q, Point r)
        {
        if (q.x <= Math.max(p.x, r.x) && q.x >= Math.
            min(p.x, r.x) &&
                q.y <= Math.max(p.y, r.y) && q.y >=
                    Math.min(p.y, r.y))
            return true;
        return false;
    }
    static int orientation(Point p, Point q, Point r)
        {
        int val = (q.y - p.y)*(r.x - q.x)-(q.x - p.x)*(
            r.y - q.y);
        if (val == 0) return 0;
        return (val > 0)? 1: 2; // clock or
            counterclock wise
    }
    static boolean doIntersect(Point p1, Point q1,
        Point p2, Point q2){
        int o1 = orientation(p1, q1, p2);
        int o2 = orientation(p1, q1, q2);
        int o3 = orientation(p2, q2, p1);
        int o4 = orientation(p2, q2, q1);

        if (o1 != o2 && o3 != o4)
            return true;

        // p1, q1 and p2 are colinear and p2 lies on
            segment p1q1
        if (o1 == 0 && onSegment(p1, p2, q1)) return
            true;

        // p1, q1 and q2 are colinear and q2 lies on
            segment p1q1
        if (o2 == 0 && onSegment(p1, q2, q1)) return
            true;

        // p2, q2 and p1 are colinear and p1 lies on
            segment p2q2
        if (o3 == 0 && onSegment(p2, p1, q2)) return
            true;

        // p2, q2 and q1 are colinear and q1 lies on
            segment p2q2
        if (o4 == 0 && onSegment(p2, q1, q2)) return
            true;

        return false; // Doesn't fall in any of the
            above cases
    }
}
```

PointIntersect.java

Description: Returns the intersection point of 2 lines(AB and CD)

25 lines

```
import java.awt.geom.Point2D;
public class PointIntersect {
```

```
    static Point2D.Double lineLineIntersection(Point A,
         Point B, Point C, Point D){
        // Line AB represented as a1x + b1y = c1
        double a1 = B.y - A.y;
        double b1 = A.x - B.x;
        double c1 = a1*(A.x) + b1*(A.y);
        // Line CD represented as a2x + b2y = c2
        double a2 = D.y - C.y;
        double b2 = C.x - D.x;
        double c2 = a2*(C.x)+ b2*(C.y);

        double determinant = a1*b2 - a2*b1;

        if (determinant == 0){
            // The lines are parallel
            return new Point2D.Double(Double.MAX_VALUE,
                 Double.MAX_VALUE);
        }
        else{
            double x = (b2*c1 - b1*c2)/determinant;
            double y = (a1*c2 - a2*c1)/determinant;
            return new Point2D.Double(x, y);
        }
    }
}
```

- Can form triang:
  $(a+b>c)AND(a+c>b)AND(b+c>a)$
- Can form quadr:
  $(a+b+c>d)AND(a+c+d>b)AND...$

hasIntersectQuadrangles.cpp

Description: Return true if two quadrangles intersect
Time: $\mathcal{O}(1)$

7 lines

```
int hasIntersectQuadrangles(int lx, int ly, int rx, int
     ry, int la, int lb, int ra, int rb) {
    lx = lxsol = max(lx, la);
    ly = lysol = max(ly, lb);
    rx = rxsol = min(rx, ra);
    ry = rysol = min(ry, rb);
    return lx < rx && ly < ry;
}
```

7.3 Polygons

basics.cpp

Description: Basic geometry functions

56 lines

```
#define EPS 1e-9
#define PI acos(-1.0)
typedef long long int ll;
struct point{
    double x,y;
    point(){x=y=0;}
    point(int _x,int _y){x=_x,y=_y;}
    bool operator <(point other) const{
        if (fabs(x-other.x)>EPS)return x<other.x;
        return y<other.y;
    }bool operator ==(point other) const{
        return (fabs(x-other.x)<EPS)&&
        (fabs(y-other.y)<EPS);
    }
    point operator +(point other) const{
        return point(x+other.x,y+other.y);
    }point operator -(point other) const{
        return point(x-other.x,y-other.y);
    }
};
struct vec {
  double x, y;
  vec(double _x, double _y): x(_x), y(_y) {}
};
vec toVec(point a, point b) { // convert 2 points to
    vec
  return vec(b.x - a.x, b.y - a.y);
}
double cross(vec a, vec b) {
  return a.x * b.y - a.y * b.x;
}
double cross(point o, point a, point b) {
    return (a.x-o.x)*(b.y-o.y) - (a.y-o.y)*(b.x-o.x);
}
// note: to accept collinear points, we have to change
    the > 0
// returns true if point r is on the left side of line
    pq
bool ccw(point p, point q, point r) {
  return cross(toVec(p, q), toVec(p, r)) > 0;
}
// returns true if point r is on the same line as the
    line pq
bool collinear(point p, point q, point r) {
  return fabs(cross(toVec(p, q), toVec(p, r))) < EPS;
}
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;
}
vec scale(vec v, double s) {
    return vec(v.x * s, v.y * s);
}

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

insidePolygon.h

Description: inPolygon
Time: $\mathcal{O}(n)$

15 lines

```
bool inPolygon(point P, vector < point > poly) {
  int n = poly.size();
  bool in = 0;
  for (int i = 0, j = n - 1; i < n; j = i++) {
    double dx = poly[j].x - poly[i].x;
    double dy = poly[j].y - poly[i].y;
    if ((poly[i].y <= P.y + EPS && P.y < poly[j].y) ||
         (poly[j].y <= P.y + EPS && P.y < poly[i].y))
      if (P.x - EPS < dx * (P.y - poly[i].y) / dy +
          poly[i].x)
        in ^= 1;
  }
  for (int i = 0; i < n - 1; i++)
    if (collinear(poly[i], poly[i + 1], P)) return
        false;
  if (collinear(poly[0], poly[n - 1], P)) return false;
  return in;
}
```

PolygonArea.h

Description: Return polygon area.
Time: $\mathcal{O}(N)$

6 lines

```
double calc_area(vector<point> Pa) {
  double ans = 0;
  for (int i = 0; i < (int) Pa.size() - 1; i++)
    ans += Pa[i].x * Pa[i + 1].y - Pa[i].y * Pa[i + 1].
        x;
  return fabs(ans) / 2.0;
}
```

PolygonCenterOfMass.h

Description: Return center of mass
Time: $\mathcal{O}(N)$

15 lines

```
point centroid(vector <point> g) //center of mass
{
  double cx = 0.0, cy = 0.0;
  for (unsigned int i = 0; i < g.size() - 1; i++) {
    double x1 = g[i].x, y1 = g[i].y;
    double x2 = g[i + 1].x, y2 = g[i + 1].y;
    double f = x1 * y2 - x2 * y1;
    cx += (x1 + x2) * f;
    cy += (y1 + y2) * f;
  }
  double res = calc_area(g); //remove abs
  cx /= 6.0 * res;
  cy /= 6.0 * res;
  return point(cx, cy);
}
```

PolygonConvex.cpp

Description: Returns if the polygon it is convex
Time: $\mathcal{O}(N)$

9 lines

```
bool isConvex(const vector < point > & P) {
  int sz = (int) P.size();
  if (sz <= 3) return false;
```

```
  bool isLeft = ccw(P[0], P[1], P[2]);
  for (int i = 1; i < sz - 1; i++)
    if (ccw(P[i], P[i + 1], P[(i + 2) == sz ? 1 : i +
        2]) != isLeft)
      return false;
  return true;
}
```

GrahamScan.java

Description:
Returns a vector of indices of the convex hull in counter-clockwise order.

Time: $\mathcal{O}(NlogN)$

36 lines

```
import java.awt.geom.Point2D;
import java.util.ArrayList;
import java.util.Comparator;
import java.util.List;

public class GrahamScan {
    double cross(Point2D.Double o, Point2D.Double a,
        Point2D.Double b){
        return (a.getX()-o.getX())*(b.getY()-o.getY())-
                (a.getY()-o.getY())*(b.getX()-o.getX())
                    ;

    }

    public List<Point2D.Double> CH(ArrayList<Point2D.
        Double> Pa){
        ArrayList<Point2D.Double> res= new ArrayList
            <>();
        Pa.sort(Comparator.comparing(Point2D.Double::
            getX).thenComparing(Point2D.Double::getY));
        int n = Pa.size();
        int m = 0;
        for(int i = 0; i<n; i++){
            while(m>1 && (cross(res.get(m-2),res.get(m
                -1),Pa.get(i)))<=0){
                res.remove(res.size()-1);
                m--;
            }
            res.add(Pa.get(i));
            m++;
        }
        for(int i = n-1, t=m+1; i>=0; i--){
            while(m>=t && (cross(res.get(m-2),res.get(m
                -1),Pa.get(i)))<=0){
                res.remove(res.size()-1);
                m--;
            }
            res.add(Pa.get(i));
            m++;
        }
        return res;
    }
}
```

ConvexHull.java

Description: Given a set of points in the plane, the convex hull of the set is the smallest convex polygon that contains all the points of it.

Time: $\mathcal{O}(n^2)$

38 lines

```
import java.util.*;
class Point{
    int x, y;
    Point(int x, int y){
        this.x=x;
        this.y=y;
    }
}
public class ConvexHull{
    public static int orientation(Point p, Point q,
        Point r){
        int val = (q.y - p.y) * (r.x - q.x) -
                (q.x - p.x) * (r.y - q.y);
        if (val == 0) return 0;  // collinear
        return (val > 0)? 1: 2; // clock or
            counterclock wise
    }
    // Prints convex hull of a set of n points.
    public static void convexHull(Point points[], int n
        ){
        if (n < 3) return;
        Vector<Point> hull = new Vector<Point>();
        int l = 0;
        for (int i = 1; i < n; i++)
            if (points[i].x < points[l].x)
                l = i;
        int p = l, q;
        do{
            hull.add(points[p]);
            q = (p + 1) % n;
            for (int i = 0; i < n; i++){
                if (orientation(points[p], points[i],
                    points[q])== 2)
                    q = i;
            }
            p = q;
        } while (p != l);
        for (Point temp : hull)
            System.out.println("(" + temp.x + ", " +
                    temp.y + ")");
    }
}
```

PolygonPerimeter.cpp

Description: Returns the perimeter of a polygon

Time: $\mathcal{O}(N)$

5 lines

```
double perimeter(const vector<point> &Pa) {
    double result = 0.0;
    for (int i = 0; i < (int)Pa.size()-1; i++)
      result += dist(Pa[i], Pa[i+1]);
    return result; }
```

PolygonDiameter.h

Description: Rotate Caliper

Time: $\mathcal{O}(N)$

39 lines

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

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);
}
double distToLineSegment(point p, point a, point b,
    point &c) {
  vec ap = toVec(a, p), ab = toVec(a, b);
  double u = dot(ap, ab) / norm_sq(ab);
  if (u < 0.0) {
    c = point(a.x, a.y); // closer to a
    return dist(p, a);
  }
  if (u > 1.0) {
    c = point(b.x, b.y); // closer to b
    return dist(p, b);
  }
  return distToLine(p, a, b, c);
}

double polygonDiameter(int n, point pt[]) {
  if (n <= 2) return 0;
  double ret = 1e+60;
  point aux;
  pt[n]=pt[0];
  //int n =  pt.size()-1; //just end added
  for (int i = 0, j = 0; i <n; i++) {
      while (cross(pt[i], pt[i+1], pt[j+1]) >= cross(pt
          [i], pt[i+1], pt[j]))
        j = (j+1)%n;
      double dist = distToLineSegment(pt[j], pt[i], pt[
          i+1],aux);
      ret = min(ret, dist);
    }
  return ret;
}
```

Combinatorial (8)

8.1 Partitions and subsets

8.1.1 Partition function

Number of ways of writing n as a sum of positive integers, disregarding the order of the summands.

$$p(0) = 1,\ p(n) = \sum_{k \in \mathbb{Z} \setminus \{0\}} (-1)^{k+1} p(n - k(3k-1)/2)$$

$$p(n) \sim 0.145/n \cdot \exp(2.56\sqrt{n})$$

n	0	1	2	3	4	5	6	7	8	9	20	50	100
$p(n)$	1	1	2	3	5	7	11	15	22	30	627	∼2e5	∼2e8

8.1.2 Binomials

binomialModPrime.h

Description: Lucas' thm: Let n, m be non-negative integers and p a prime. Write $n = n_k p^k + ... + n_1 p + n_0$ and $m = m_k p^k + ... + m_1 p + m_0$. Then $\binom{n}{m} \equiv \prod_{i=0}^{k} \binom{n_i}{m_i} \pmod{p}$. fact and invfact must hold precomputed factorials / inverse factorials, e.g. from ModInverse.h.

Time: $\mathcal{O}(\log_p n)$

10 lines

```
ll chooseModP(ll n, ll m, int p, vi& fact, vi& invfact)
    {
  ll c = 1;
  while (n || m) {
    ll a = n % p, b = m % p;
    if (a < b) return 0;
    c = c * fact[a] % p * invfact[b] % p * invfact[a -
        b] % p;
    n /= p; m /= p;
  }
  return c;
}
```

Various (9)

9.1 Dynamic programming

DPKnapsack01.java

Description: 0/1 Knapsack Problem - Given items of certain weights/values and maximum allowed weight how to pick items from this set to maximize sum of value of items such that sum of weights is less than or equal to maximum allowed weight.

Time: $\mathcal{O}(W * totalitems)$

19 lines

```
public class Knapsack01 {
    public int bottomUpDP(int val[], int wt[], int W){
        int K[][] = new int[val.length+1][W+1];
        for(int i=0; i <= val.length; i++){
            for(int j=0; j <= W; j++){
                if(i == 0 || j == 0){
                    K[i][j] = 0;
                    continue;
                }
                if(j - wt[i-1] >= 0){
                    K[i][j] = Math.max(K[i-1][j], K[i
                        -1][j-wt[i-1]] + val[i-1]);
                }else{
                    K[i][j] = K[i-1][j];
                }
            }
        }
        return K[val.length][W];
    }
}
```

KnapsackChoice.java

Description: Descripción generica

20 lines

```
public class KnapsackChoice {
    public int f(i, weight) {
        if (weight > W) return -INF;
        if (weight == W || i >=N)
            return 0;
        if (memo[i, weight] !=-1)
            return memo[i,weight];

        T1 = f(i + 1, weight + W[i]) + V[i];
        T2 = f(i + 1, weight);
        if (T1 > T2) {
            memo[i, weight] =T1;
            choice[i, weight] =(i + 1, weight + W[i]);
        }else {
            memo[i, weight] =T2;
            choice[i, weight] =(i + 1, weight);
        }
        return memo[i,weight];
    }
}
```

EditDistance.java

Description: Given words A and B with only 3 operations (delete, insert, modify), returns the minimum number of operations to turn A into B (length<100)

16 lines

```
public class EditDistance {
    public int  F(int i, int j) {
        if (i >= S.length && j >= T.length)
            return 0;
        if (i >= S.length || j >=T.length)
            return INF;
        if (memo[i][j] != -1)
            return memo[i][j];
        if (S[i] == T[j])
            memo[i][j] = F(i + 1, j + 1);
        else
            memo[i][j] = min(F(i + 1, j) + 1, F(i, j +
                1) + 1, F(i + 1, j + 1) + 1);

        return memo[i][j];
    }
}
```

LongestCommonSubseq.java

Description: Given 2 sequences A and B, determine the longest existing subsequence of A in B

13 lines

```
public class LongestCommonSubseq {
    public int LCS(int i, int j) {
        if (i >= S.length || j >=T.length)
            return 0;
        if (memo[i][j] != -1)
            return memo[i][j];
        if (S[i] == T[j])
            memo[i][j] = F(i + 1, j + 1) + 1;
        else
            memo[i][j] = max(LCS(i + 1, j), LCS(i, j +
                1));
        return memo[i][j];
    }
}
```

LongestIncreasingSubSeq.java

Description: Determines the longest increasing subsequence on a list

14 lines

```
public class LongestIncreasingSubseq {
    public int F(i, j) {
        if (i >= A.length)
            return 0;
        if (memo[i][j] != -1)
            return memo[i][j];
        if (A[i] > A[j])
            memo[i][j] = F(i + 1, i) + 1;
        else
            memo[i][j] = max(F(i + 1, i), F(i + 1, j));

        return memo[i][j];
    }
}
```

CodifiedMessage.java

Description: Given N codes, no one repeated and without 0s and given a codified message, returns how many different messages we can obtain from that codified message. Note that 0s have to be treated as spaces so we split the incoming message by 0s on the message and multiply the result of the DP of each of the words to obtain the final result.

23 lines

```
import java.util.HashSet;
public class tia {
    public static int dp(String word, int[] memo, int i
        , int n, HashSet<String> set) {
        if (i > n - 1)
            return 1;

        if (memo[i] != -1)
            return memo[i];

        memo[i] = 0;
```

```
        if (set.contains("" + word.charAt(i)))
            memo[i] = (memo[i] + dp(word, memo, i + 1,
                n, set)) % MOD; //MOD is to present the
                 results

        if (i + 1 < n && set.contains("" + word.charAt(
            i) + word.charAt(i + 1)))
            memo[i] = (memo[i] + dp(word, memo, i + 2,
                n, set)) % MOD;

        if (i + 2 < n && set.contains("" + word.charAt(
            i) + word.charAt(i + 1) + word.charAt(i +
            2)))
            memo[i] = (memo[i] + dp(word, memo, i + 3,
                n, set)) % MOD;

        return memo[i];
    }
}
```

DP3dim

Description: Given N codes, no one repeated and without 0s and given a codified message, returns how many different messages we can obtain from that codified message. Note that 0s have to be treated as spaces so we split the incoming message by 0s on the message and multiply the result of the DP of each of the words to obtain the final result. 23 lines

```
import java.util.HashSet;
public class tia {
    public static int dp(String word, int[] memo, int i
        , int n, HashSet<String> set) {
        if (i > n - 1)
            return 1;

        if (memo[i] != -1)
            return memo[i];

        memo[i] = 0;

        if (set.contains("" + word.charAt(i)))
            memo[i] = (memo[i] + dp(word, memo, i + 1,
                n, set)) % MOD; //MOD is to present the
                 results

        if (i + 1 < n && set.contains("" + word.charAt(
            i) + word.charAt(i + 1)))
            memo[i] = (memo[i] + dp(word, memo, i + 2,
                n, set)) % MOD;

        if (i + 2 < n && set.contains("" + word.charAt(
            i) + word.charAt(i + 1) + word.charAt(i +
            2)))
            memo[i] = (memo[i] + dp(word, memo, i + 3,
                n, set)) % MOD;

        return memo[i];
    }
}
```

9.1.1 Maximum subarray sum (Kadane)

Let's take an array dp[] where each dp[i] denotes maximum subarray sum ending at index i (including i).

We can say that:

$dp[i] = max(dp[i-1], 0) + arr[i] \forall i \in [1, n-1]$

9.2 Area of polygon with circumradio

Area: $(r * r * n * sin(360/n))/2$

9.3 Complexity

Complejidad	Rangos
100000M	$O(1)$
1000M	$O(log(n))$
1M	$O(n)$
100K	$O(log(n))$
1000	$O(n^2)$
100	$O(n^3)$
50	$O(n^4)$
20	$O(2^n)$
13	$O(n!)$