https://drive.google.com/open?id=14WHucJ46WSOFV4F_SAtYYhtQECCdy_GM

https://www.1point3acres.com/bbs/forum.php?mod=viewthread&tid=212885&extra=&page=1

滑雪/10 wizard/ flight

https://www.1point3acres.com/bbs/forum.php?mod=viewthread&tid=444503&extra=&page=1

Dirktra: 不能为负

Bell-Man ford:https://www.youtube.com/watch?v=obWXjtg0L64

Topological sort: 找入度为0

BFS:

滑雪： 如有有环，如果有负数

Time complexity: 因方法而异

131333331

<<IP to CIPD>>

```
package Airbnb;

import java.util.*;
/* Ex1
11111111 00000000 00000000 00001111 "255.0.0.7/32", curMask = (32 - 0) = 32, rangeMask = (32 - log(10, 2)) = 29
11111111 00000000 00000000 00001000 "255.0.0.8/29", curMask = (32 - 3) = 29, rangeMask = (32 - log(9, 2)) = 29
```

```
11111111 00000000 00000000 00001001
11111111 00000000 00000000 00001010
11111111 00000000 00000000 00001011
11111111 00000000 00000000 00001100
11111111 00000000 00000000 00001101
11111111 00000000 00000000 00001110
11111111 00000000 00000000 00001111
11111111 00000000 00000000 00010000 "255.0.0.16/32", curMask = (32 - 4) = 28, rangeMask
= (32 - log(1, 2)) = 32

 */

public class Q11_IP_CIDR {
    public static void main(String[] args) {
            // Java uses two's complement for negative numbers and
            // the basic rule is to take the positive, invert all bits then add one.
            int start=0b10100; //0b101;
            int y = -start;
            int z = start & (-start); // the lowest bit
            System.out.println("start: " + start + "  (-start):" + y);
            System.out.println("start: " + Integer.toBinaryString(start) + "  (-start): " +
Integer.toBinaryString(y));
            String s = Integer.toBinaryString(z);
            System.out.println("start & (-start): " + s);



            Solution sol = new Solution();
            //List<String> res = sol.ipRange2Cidr("255.0.0.7", 10);
            List<String> res = sol.ipRange2Cidr("0.0.1.7", 10);
            System.out.println("Ex1 res: " + res);

            /*
           assertEquals("255.0.0.7/32", res.get(0));
           assertEquals("255.0.0.8/29", res.get(1));
           assertEquals("255.0.0.16/32", res.get(2)); */

           //res = sol.ipRange2Cidr("1.1.1.0", 4);
           //System.out.println("Ex2 res: " + res);
           // assertEquals("1.1.1.0/30", res.get(0));

           //res = sol.ipRange2Cidr("1.1.1.1", 4);
           //System.out.println("Ex3 res: " + res);
           //assertEquals("1.1.1.1/32", res.get(0));
           //assertEquals("1.1.1.2/31", res.get(1));
```

```
        //assertEquals("1.1.1.4/32", res.get(2));
    }
}

    /*
     * IP range to CIDR https://en.wikipedia.org/wiki/Classless_Inter-Domain_Routing
     * http://www.ipaddressguide.com/cidr
     * https://stackoverflow.com/questions/33443914/how-to-convert-ip-address-range-
     * to-cidr-in-java AirBnB Interview Question
     */
class Solution {
    public List<String> ipRange2Cidr(String startIp, int range) {
        // check parameters
        String a = "";
        long start = ipToLong(startIp);
        long end = start + range - 1;
        List<String> res = new ArrayList<>();
        while (start <= end) {
            System.out.println("strat: " + start + " " + Long.toBinaryString(start));
            // identify the location of first 1's from lower bit to higher bit of start IP
            // e.g. 00000001.00000001.00000001.01101100, return 4 (100)
            long locOfFirstOne = start & (-start);
            int curMask = 32 - (int) (Math.log(locOfFirstOne)/ Math.log(2));


            // calculate how many IP addresses between the start and end
            // e.g. between 1.1.1.111 and 1.1.1.120, there are 10 IP address
            // 3 bits to represent 8 IPs, from 1.1.1.112 to 1.1.1.119 (119 - 112 + 1 = 8)
            double currRange = Math.log(end - start + 1) / Math.log(2);
            int currRangeMask = 32 - (int) Math.floor(currRange);
            System.out.println("locOfFirstOne: " + locOfFirstOne + " curMask: " + curMask +
"  currRangeMask: " + currRangeMask);

            // why max?
            // if the currRangeMask is larger than curMask
            // which means the numbers of IPs from start to end is smaller than mask range
            // so we can't use as many as bits we want to mask the start IP to avoid exceed
            // the end IP
            // Otherwise, if currRangeMask is smaller than curMask, which means number of
            // IPs is larger than mask range
            // in this case we can use curMask to mask as many as IPs from start we want.
            curMask = Math.max(currRangeMask, curMask);

            // Add to results
```

```
                String ip = longToIP(start);
                res.add(ip + "/" + curMask);
                // We have already included 2^(32 - curMask) numbers of IP into result
                // So the next roundUp start must insert that number
                start += Math.pow(2, (32 - curMask));
                System.out.println("res: " + res + " start: " + start + " curMask: " + curMask);
        }
        return res;
    }

    private long ipToLong(String strIP) {

        long[] ip = new long[4];
        String[] ipSec = strIP.split("\\."); // "." represent single char in regex
        System.out.println("\nipSec: " + Arrays.toString(ipSec));
        for (int k = 0; k < 4; k++) {
                ip[k] = Long.valueOf(ipSec[k]);
        }
        long res = (ip[0] << 24) + (ip[1] << 16) + (ip[2] << 8) + ip[3];
        System.out.println("ipToLong() res: " + res);
        return res;
    }

    private String longToIP(long longIP) {
        StringBuffer sb = new StringBuffer("");
        sb.append(String.valueOf(longIP >>> 24));
        sb.append(".");
        sb.append(String.valueOf((longIP & 0x00FFFFFF) >>> 16));
        sb.append(".");
        sb.append(String.valueOf((longIP & 0x0000FFFF) >>> 8));
        sb.append(".");
        sb.append(String.valueOf(longIP & 0x000000FF));

        return sb.toString();
    }


}
```

<<Wizard>>

```
import java.io.*;
import java.util.*;


class Solution {


        public static void main(String[] args) {
        System.out.println("Hello World");

        List<List<Integer>> wizards = new ArrayList<>();
        for (int i = 0; i < 5; i++) {
        List<Integer> list = new ArrayList<>();
        if (i == 0) {
                list.add(1);
                list.add(2);
        } else if (i == 1) {
                list.add(3);
        } else if (i == 2) {
                list.add(3);
                list.add(4);
        } else if (i == 3) {
                list.add(4);
        }
        wizards.add(list);

        }
        System.out.println("wizards: " + wizards);
        List<Integer> path = getMinCostBFS(wizards, 0, 4);
        System.out.println("path [0, 1, 3, 4]: " + path);
        path = getMinCostPq(wizards, 0, 4);
        System.out.println("path [0, 1, 3, 4]: " + path);


        wizards= new ArrayList<>();
        for (int i = 0; i < 10; i++) {
        List<Integer> list = new ArrayList<>();
```

```
        if (i == 0) {
                list.add(1);
                list.add(5);
                list.add(9);
        } else if (i == 1) {
                list.add(2);
                list.add(3);
                list.add(9);
        } else if (i == 2) {
                list.add(4);
        } else if (i == 5) {
                list.add(9);
        }
        wizards.add(list);
        }
        System.out.println("wizards: " + wizards);
        path = getMinCostBFS(wizards, 0, 9);
        System.out.println("path ([0, 5, 9]): " + path);
        path = getMinCostPq(wizards, 0, 9);
        System.out.println("path ([0, 5, 9]): " + path);

        }


        public static List<Integer> getMinCostBFS(List<List<Integer>> wizards, Integer source,
Integer target) {
        if (wizards == null || wizards.size() == 0) {
        return new ArrayList<>();
        }


        int n = wizards.size();
        int[] pre = new int[n];
        Map<Integer, Wizard> map = new HashMap<>();
        for (int i = 0; i < n; i++) {
        pre[i] = i;
        map.put(i,  new Wizard(i));
        }

        map.get(source).dist = 0;
        Queue<Wizard> queue = new LinkedList<>();
        queue.offer(map.get(source));
        while (!queue.isEmpty()) {
        Wizard cur = queue.poll();
```

```
        List<Integer> neighbors = wizards.get(cur.id);
        for (Integer neighbor : neighbors) {
                Wizard next = map.get(neighbor);
                int weight = (next.id - cur.id) * (next.id - cur.id);
                if (cur.dist + weight < next.dist) {
                pre[next.id] = cur.id;
                next.dist = cur.dist + weight;
                queue.offer(next);

                }
        }
        }

        List<Integer> list = new ArrayList<>();
        int cur = target;
        list.add(0, cur);
        while (pre[cur] != source) {
        cur = pre[cur];
        list.add(0, cur);
        }
        list.add(0, source);
        return list;
        }

        public static List<Integer> getMinCostPq(List<List<Integer>> wizards, Integer source,
Integer target) {
        if (wizards == null || wizards.size() == 0) {
        return new ArrayList<>();
        }


        int n = wizards.size();
        int[] pre = new int[n];
        Map<Integer, Wizard> map = new HashMap<>();
        for (int i = 0; i < n; i++) {
        pre[i] = i;
        map.put(i,  new Wizard(i));
        }

        map.get(source).dist = 0;
        PriorityQueue<Wizard> pq = new PriorityQueue<>();
        pq.offer(map.get(source));
        while (!pq.isEmpty()) {
        Wizard cur = pq.poll();
```

```
        List<Integer> neighbors = wizards.get(cur.id);
        for (Integer neighbor : neighbors) {
                Wizard next = map.get(neighbor);
                int weight = (next.id - cur.id) * (next.id - cur.id);
                if (cur.dist + weight < next.dist) {
                pre[next.id] = cur.id;
                next.dist = cur.dist + weight;
                if (next.id == target) {
                break;
                }
                pq.offer(next);

                }
        }
        }

        List<Integer> list = new ArrayList<>();
        int cur = target;
        list.add(0, cur);
        while (pre[cur] != source) {
        cur = pre[cur];
        list.add(0, cur);
        }
        list.add(0, source);
        return list;
        }



}


class Wizard implements Comparable<Wizard> {
        int id;
        int dist;

        public Wizard(int n) {
        id = n;
        dist = Integer.MAX_VALUE;
        }

        @Override
        public int compareTo(Wizard w) {
        return this.dist - w.dist;
```

```
        }
}
```