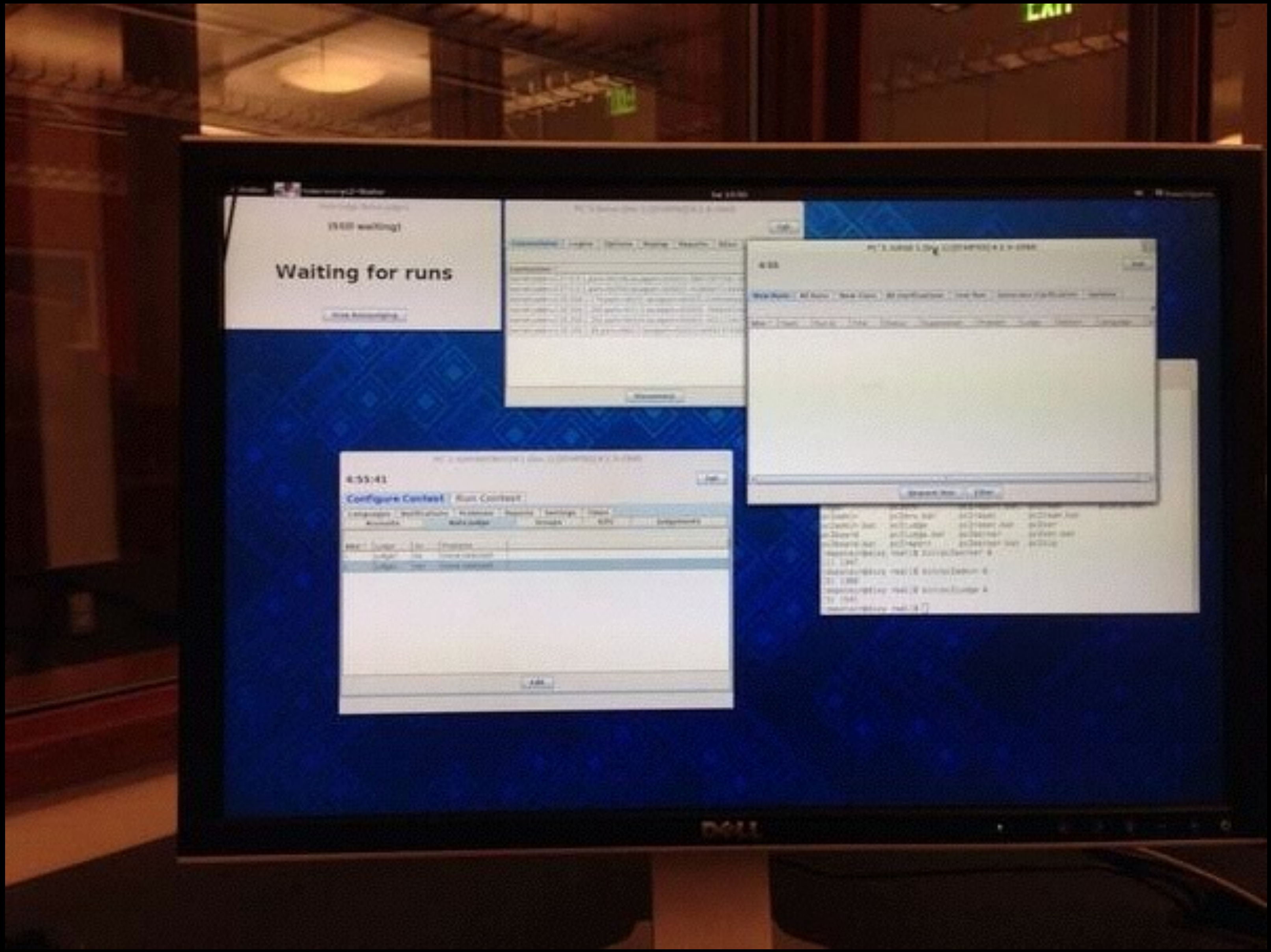
ACM-ICPC and you

Daniel Epstein, CSE Grad Student

Shameless plug #1

I’m organizing this year’s qualifying contest



(Sorry, this was the only picture I took…
will do better this year)

Shameless plug #1

Saturday, October 11, 10am-4pm, CSE Labs 002 & 006

(tell your friends)

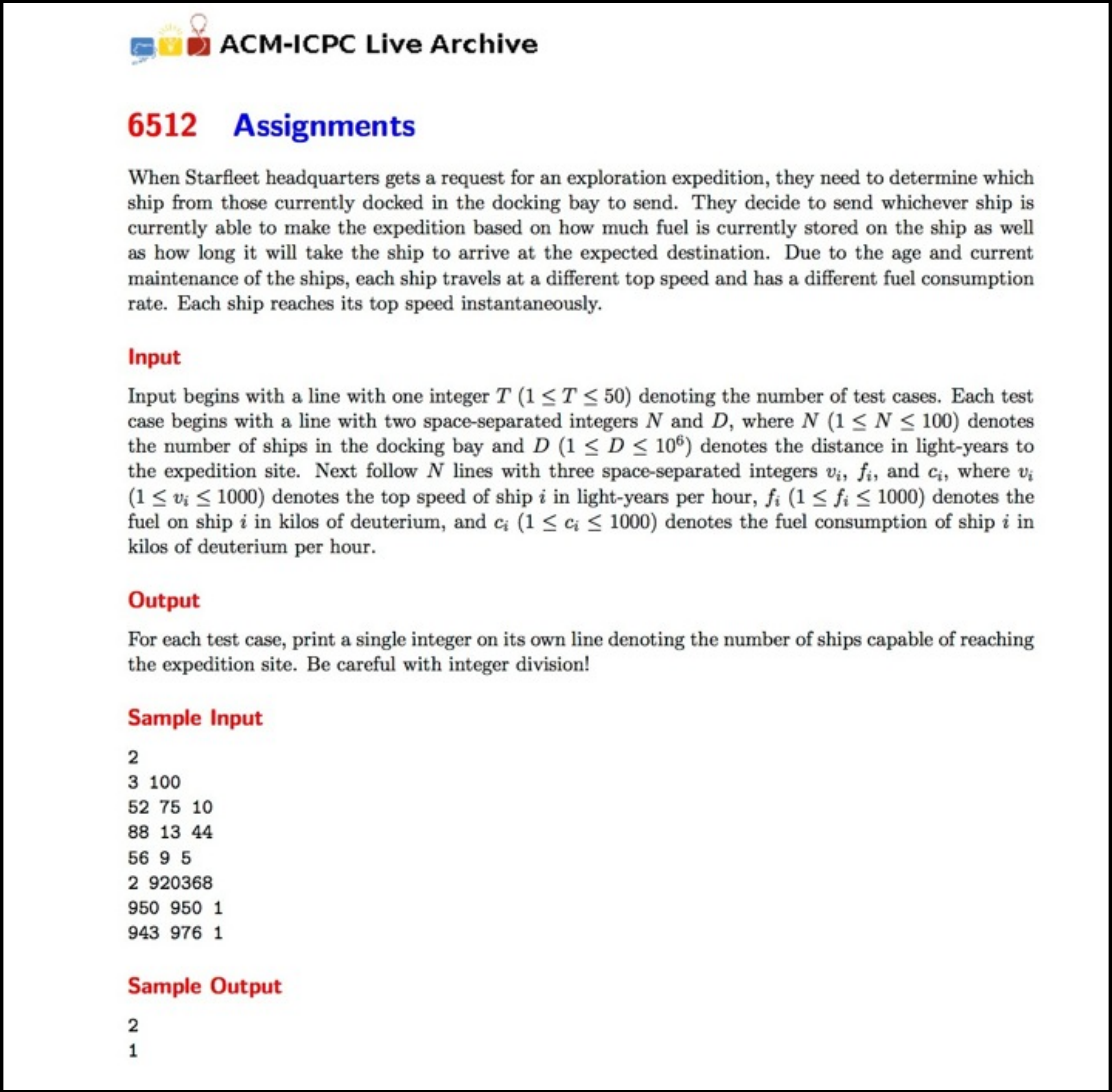
What are programming competitions?

And why should I care?

Shameless Plug #2

https://github.com/depstein/programming-competitions

World Finals
Stockholm
Sweden
acm
IBM
2009 World Finals
Stockholm
acm
IBM



6512 Assignments

When Starfleet headquarters gets a request for an exploration expedition, they need to determine which ship from those currently docked in the docking bay to send. They decide to send whichever ship is currently able to make the expedition based on how much fuel is currently stored on the ship as well as how long it will take the ship to arrive at the expected destination. Due to the age and current maintenance of the ships, each ship travels at a different top speed and has a different fuel consumption rate. Each ship reaches its top speed instantaneously.

Input

Input begins with a line with one integer T ($1 \leq T \leq 50$) denoting the number of test cases. Each test case begins with a line with two space-separated integers N and D, where N ($1 \leq N \leq 100$) denotes the number of ships in the docking bay and D ($1 \leq D \leq 10^6$) denotes the distance in light-years to the expedition site. Next follow N lines with three space-separated integers v_i, f_i, and c_i, where v_i ($1 \leq v_i \leq 1000$) denotes the top speed of ship i in light-years per hour, f_i ($1 \leq f_i \leq 1000$) denotes the fuel on ship i in kilos of deuterium, and c_i ($1 \leq c_i \leq 1000$) denotes the fuel consumption of ship i in kilos of deuterium per hour.

Output

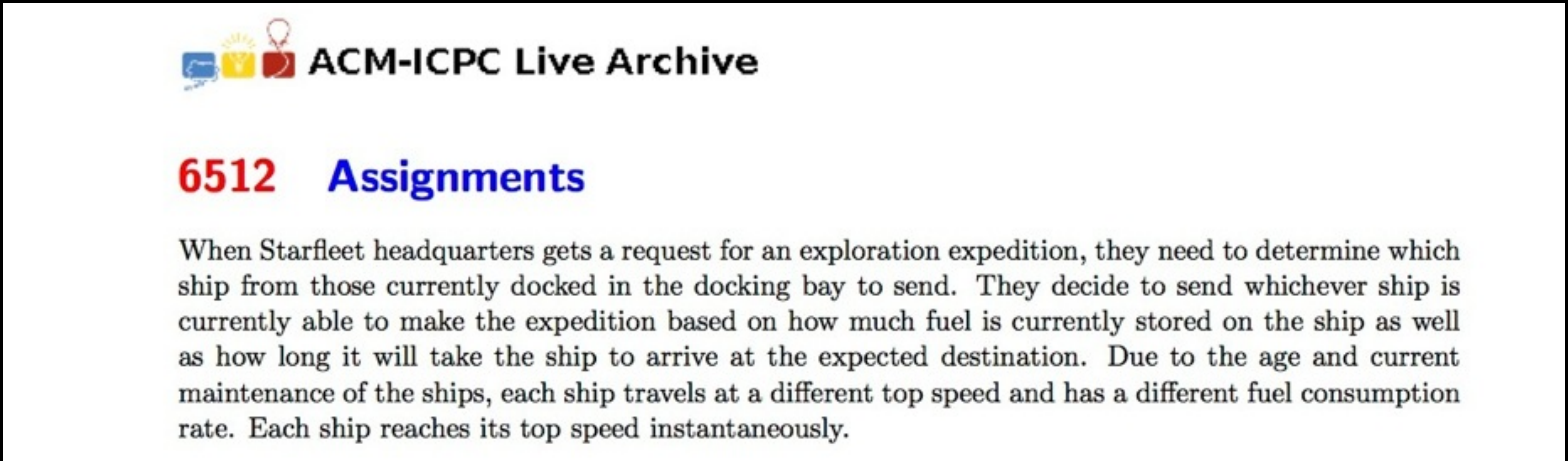
For each test case, print a single integer on its own line denoting the number of ships capable of reaching the expedition site. Be careful with integer division!

Sample Input

```
2
3 100
52 75 10
88 13 44
56 9 5
2 920368
950 950 1
943 976 1
```

Sample Output

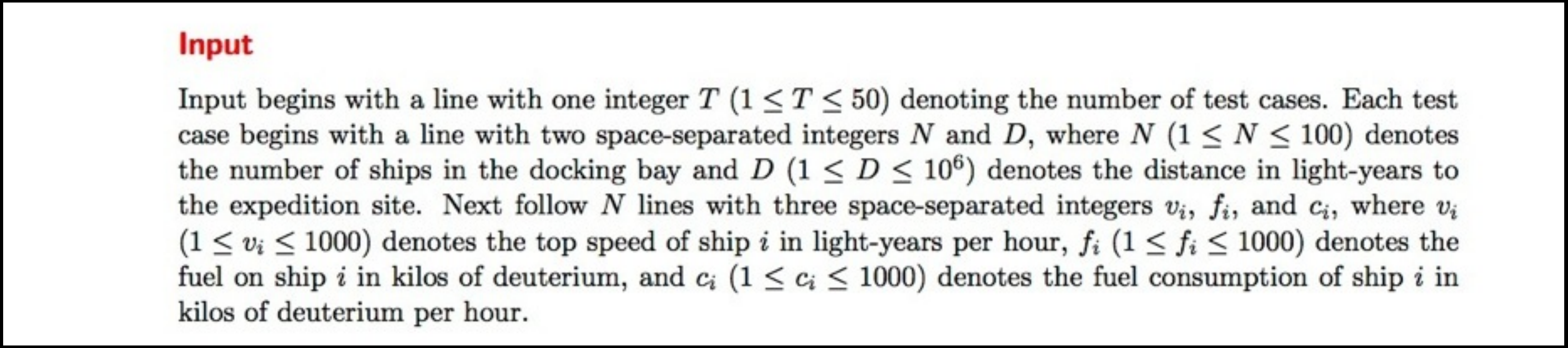
```
2
1
```



ACM-ICPC Live Archive

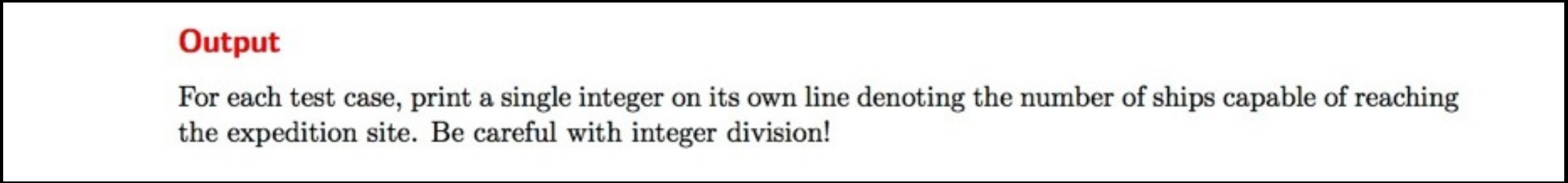
6512 Assignments

When Starfleet headquarters gets a request for an exploration expedition, they need to determine which ship from those currently docked in the docking bay to send. They decide to send whichever ship is currently able to make the expedition based on how much fuel is currently stored on the ship as well as how long it will take the ship to arrive at the expected destination. Due to the age and current maintenance of the ships, each ship travels at a different top speed and has a different fuel consumption rate. Each ship reaches its top speed instantaneously.



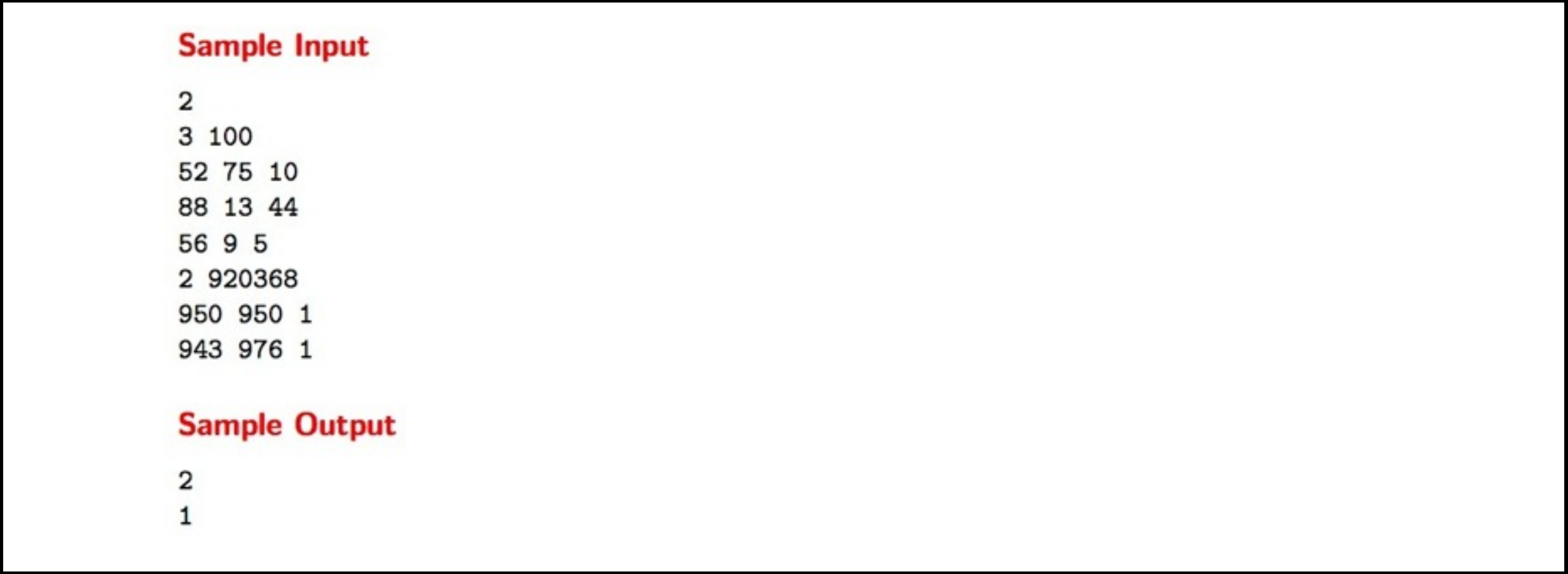
Input

Input begins with a line with one integer T ($1 \leq T \leq 50$) denoting the number of test cases. Each test case begins with a line with two space-separated integers N and D, where N ($1 \leq N \leq 100$) denotes the number of ships in the docking bay and D ($1 \leq D \leq 10^6$) denotes the distance in light-years to the expedition site. Next follow N lines with three space-separated integers v_i, f_i, and c_i, where v_i ($1 \leq v_i \leq 1000$) denotes the top speed of ship i in light-years per hour, f_i ($1 \leq f_i \leq 1000$) denotes the fuel on ship i in kilos of deuterium, and c_i ($1 \leq c_i \leq 1000$) denotes the fuel consumption of ship i in kilos of deuterium per hour.



Output

For each test case, print a single integer on its own line denoting the number of ships capable of reaching the expedition site. Be careful with integer division!



Sample Input

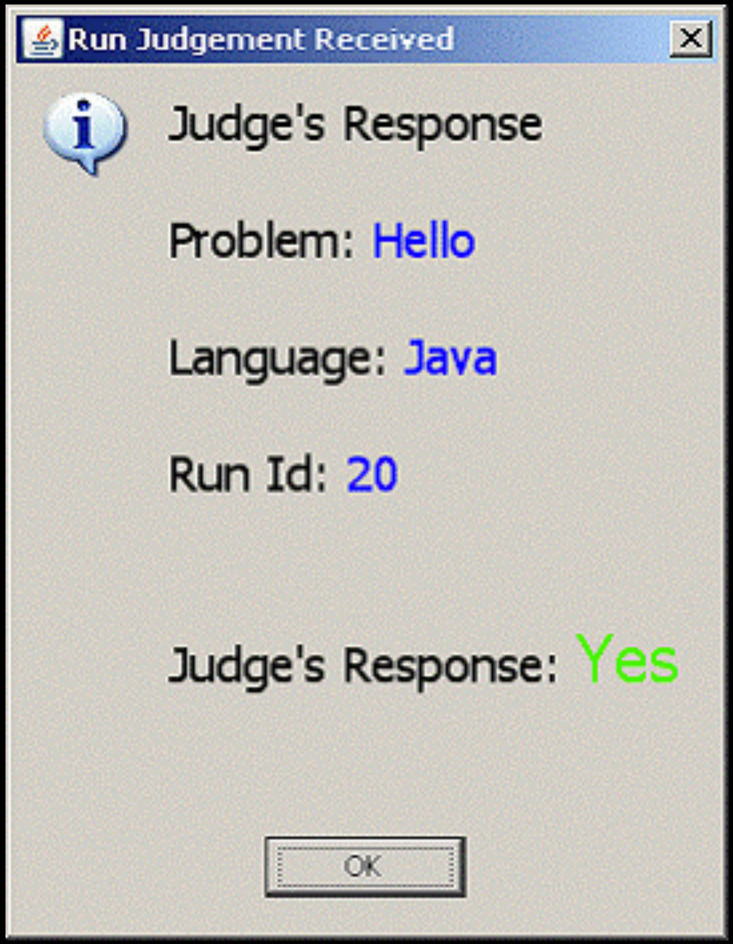
```
2
3 100
52 75 10
88 13 44
56 9 5
2 920368
950 950 1
943 976 1
```

Sample Output

```
2
1
```

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

public class Main {
  public static void main(String[] args) {
    Scanner in = new Scanner(System.in);
    int cases = in.nextInt();
    for(int i=0;i<cases;i++) {
      int n = in.nextInt();
      int d = in.nextInt();
      int count = 0;
      for(int j=0;j<n;j++) {
        int v = in.nextInt();
        int f = in.nextInt();
        int c = in.nextInt();
        if((v*f)/c >= d) {
          count++;
        }
      }
      System.out.println(count);
    }
  }
}
```



Run Judgement Received

Judge's Response

Problem: Hello

Language: Java

Run Id: 20

Judge's Response: Yes

No - Compilation Error

No - Runtime Exception

No - Time Limit Exceeded

No - Wrong Answer

No - See Contest Staff

No - Compilation Error

No - Runtime Exception

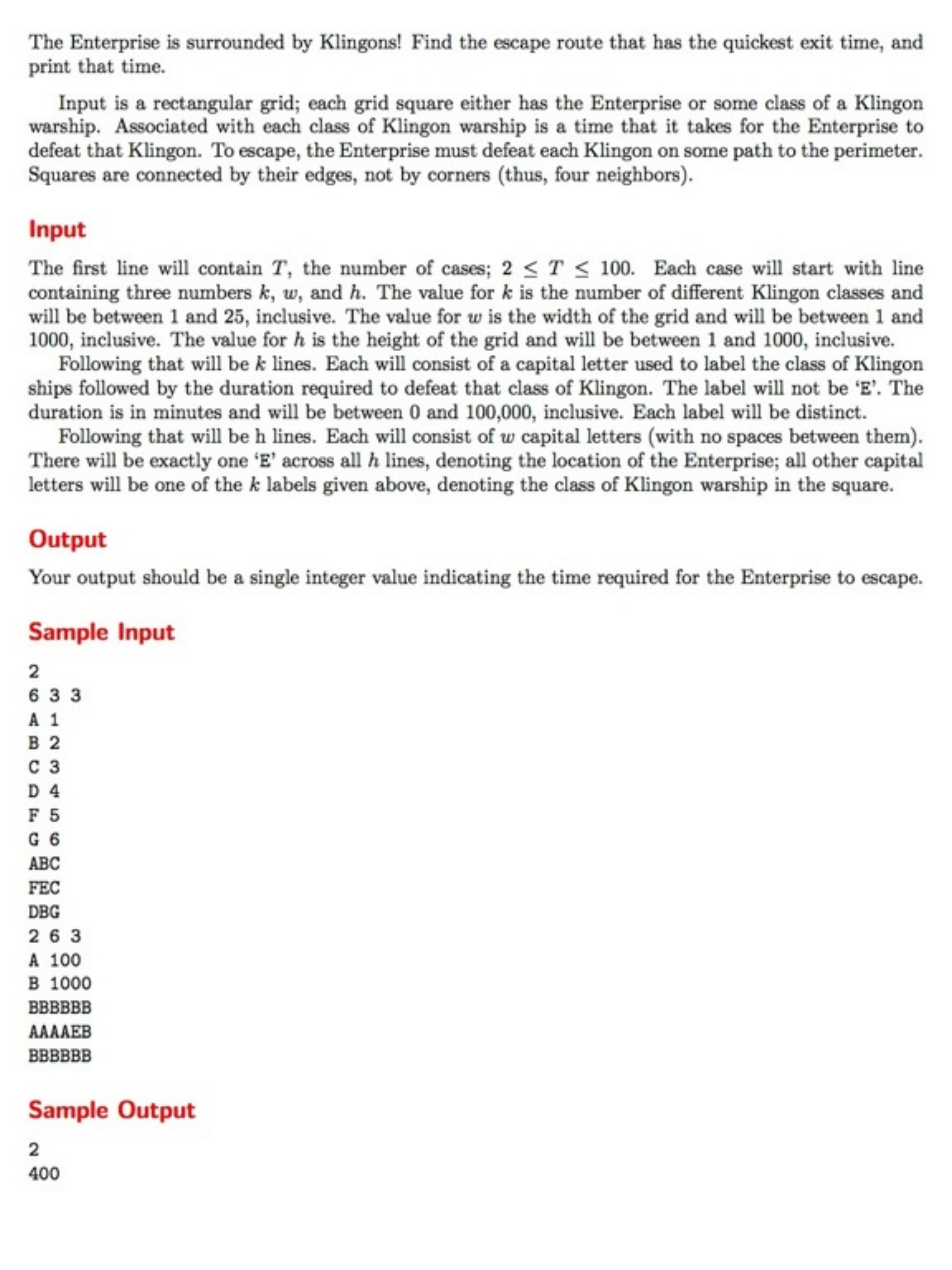
No - Time Limit Exceeded

~10,000,000 operations

No - Wrong Answer

No - See Contest Staff

Let’s try a harder problem



The Enterprise is surrounded by Klingons! Find the escape route that has the quickest exit time, and print that time.

Input is a rectangular grid; each grid square either has the Enterprise or some class of a Klingon warship. Associated with each class of Klingon warship is a time that it takes for the Enterprise to defeat that Klingon. To escape, the Enterprise must defeat each Klingon on some path to the perimeter. Squares are connected by their edges, not by corners (thus, four neighbors).

Input

The first line will contain T, the number of cases; $2 \leq T \leq 100$. Each case will start with line containing three numbers k, w, and h. The value for k is the number of different Klingon classes and will be between 1 and 25, inclusive. The value for w is the width of the grid and will be between 1 and 1000, inclusive. The value for h is the height of the grid and will be between 1 and 1000, inclusive.

Following that will be k lines. Each will consist of a capital letter used to label the class of Klingon ships followed by the duration required to defeat that class of Klingon. The label will not be 'E'. The duration is in minutes and will be between 0 and 100,000, inclusive. Each label will be distinct.

Following that will be h lines. Each will consist of w capital letters (with no spaces between them). There will be exactly one 'E' across all h lines, denoting the location of the Enterprise; all other capital letters will be one of the k labels given above, denoting the class of Klingon warship in the square.

Output

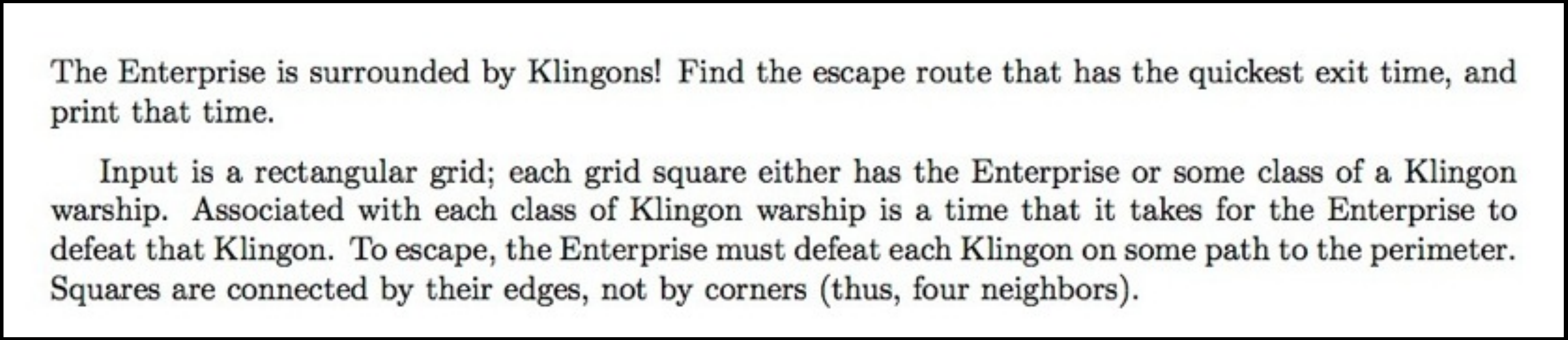
Your output should be a single integer value indicating the time required for the Enterprise to escape.

Sample Input

```
2
6 3 3
A 1
B 2
C 3
D 4
F 5
G 6
ABC
FEC
DBG
2 6 3
A 100
B 1000
BBBBBB
AAAAEB
BBBBBB
```

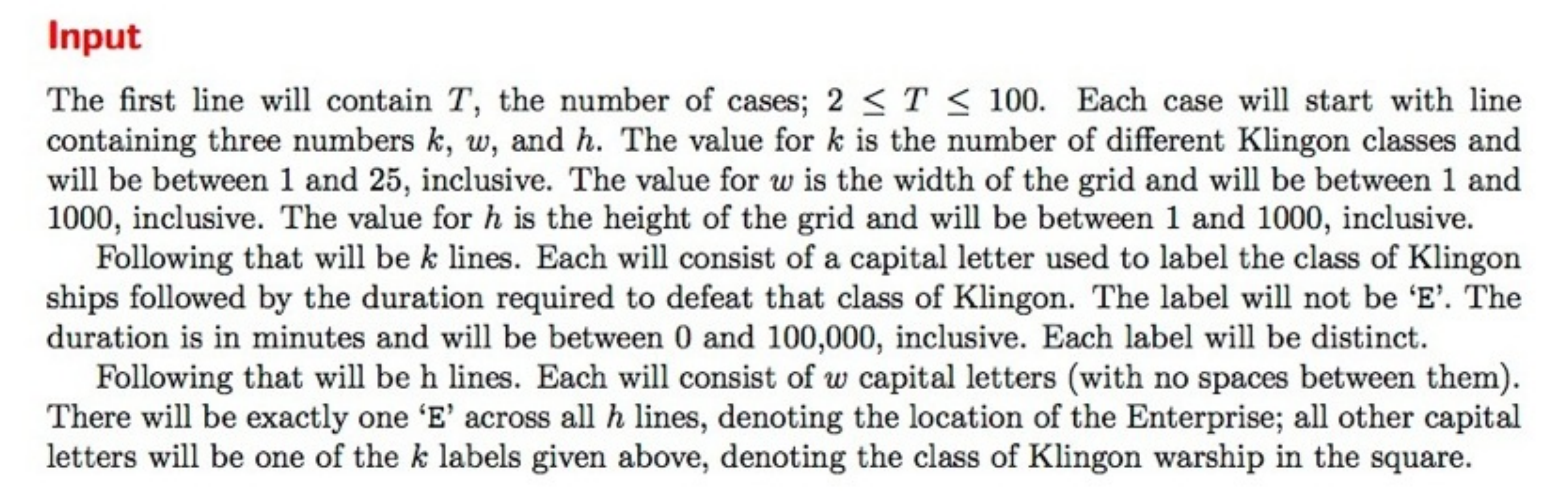
Sample Output

```
2
400
```



The Enterprise is surrounded by Klingons! Find the escape route that has the quickest exit time, and print that time.

Input is a rectangular grid; each grid square either has the Enterprise or some class of a Klingon warship. Associated with each class of Klingon warship is a time that it takes for the Enterprise to defeat that Klingon. To escape, the Enterprise must defeat each Klingon on some path to the perimeter. Squares are connected by their edges, not by corners (thus, four neighbors).

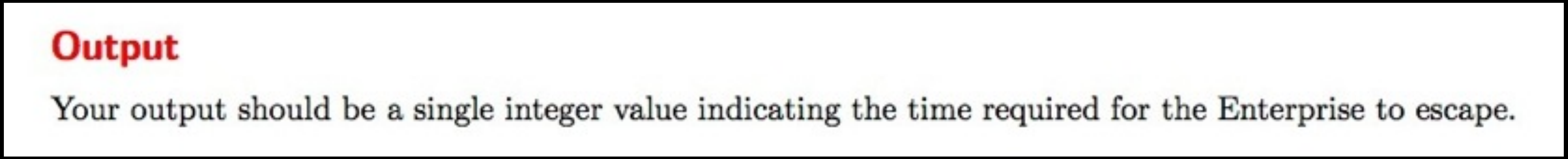


Input

The first line will contain T, the number of cases; $2 \leq T \leq 100$. Each case will start with line containing three numbers k, w, and h. The value for k is the number of different Klingon classes and will be between 1 and 25, inclusive. The value for w is the width of the grid and will be between 1 and 1000, inclusive. The value for h is the height of the grid and will be between 1 and 1000, inclusive.

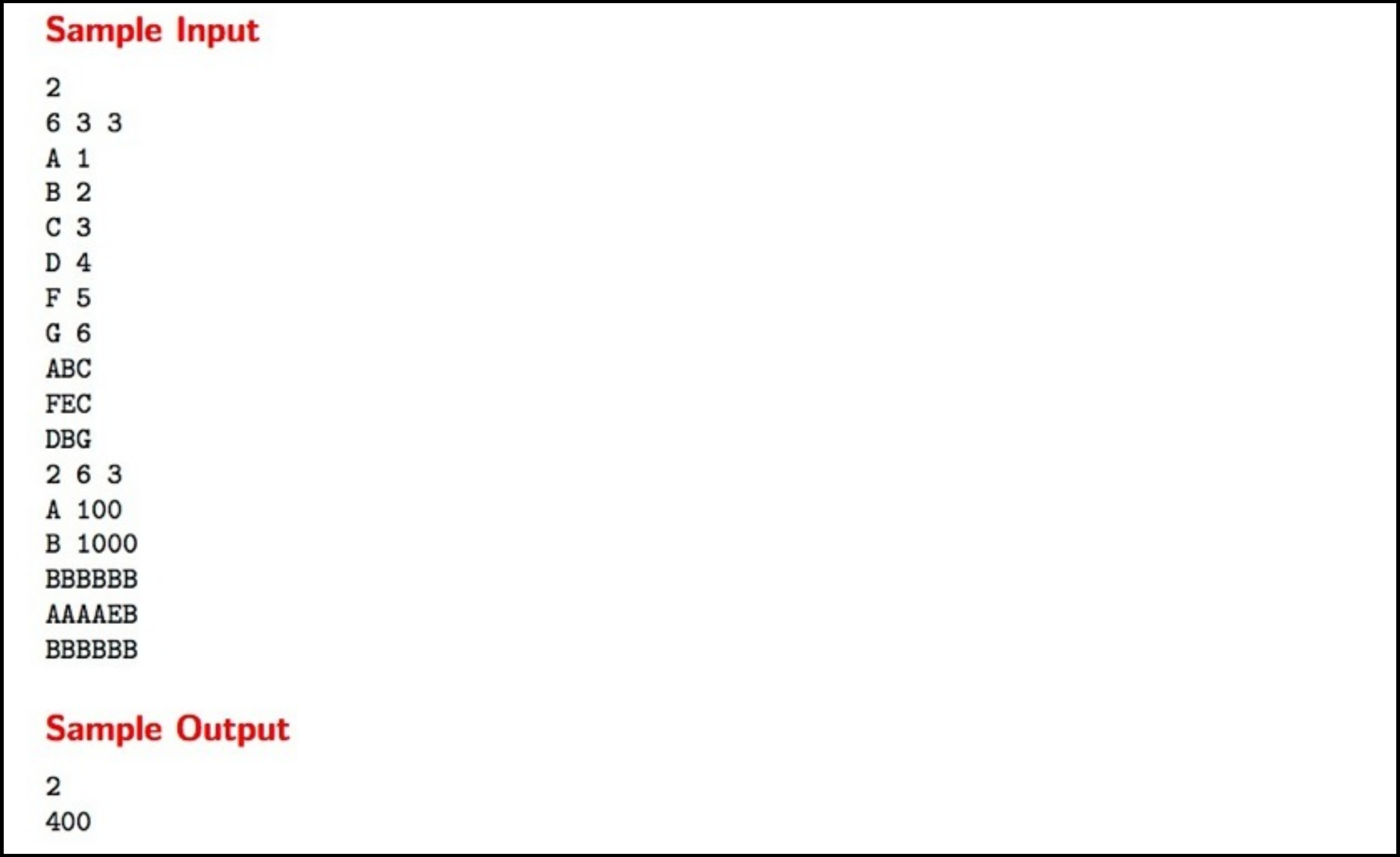
Following that will be k lines. Each will consist of a capital letter used to label the class of Klingon ships followed by the duration required to defeat that class of Klingon. The label will not be '`E`'. The duration is in minutes and will be between 0 and 100,000, inclusive. Each label will be distinct.

Following that will be h lines. Each will consist of w capital letters (with no spaces between them). There will be exactly one '`E`' across all h lines, denoting the location of the Enterprise; all other capital letters will be one of the k labels given above, denoting the class of Klingon warship in the square.



Output

Your output should be a single integer value indicating the time required for the Enterprise to escape.



Sample Input

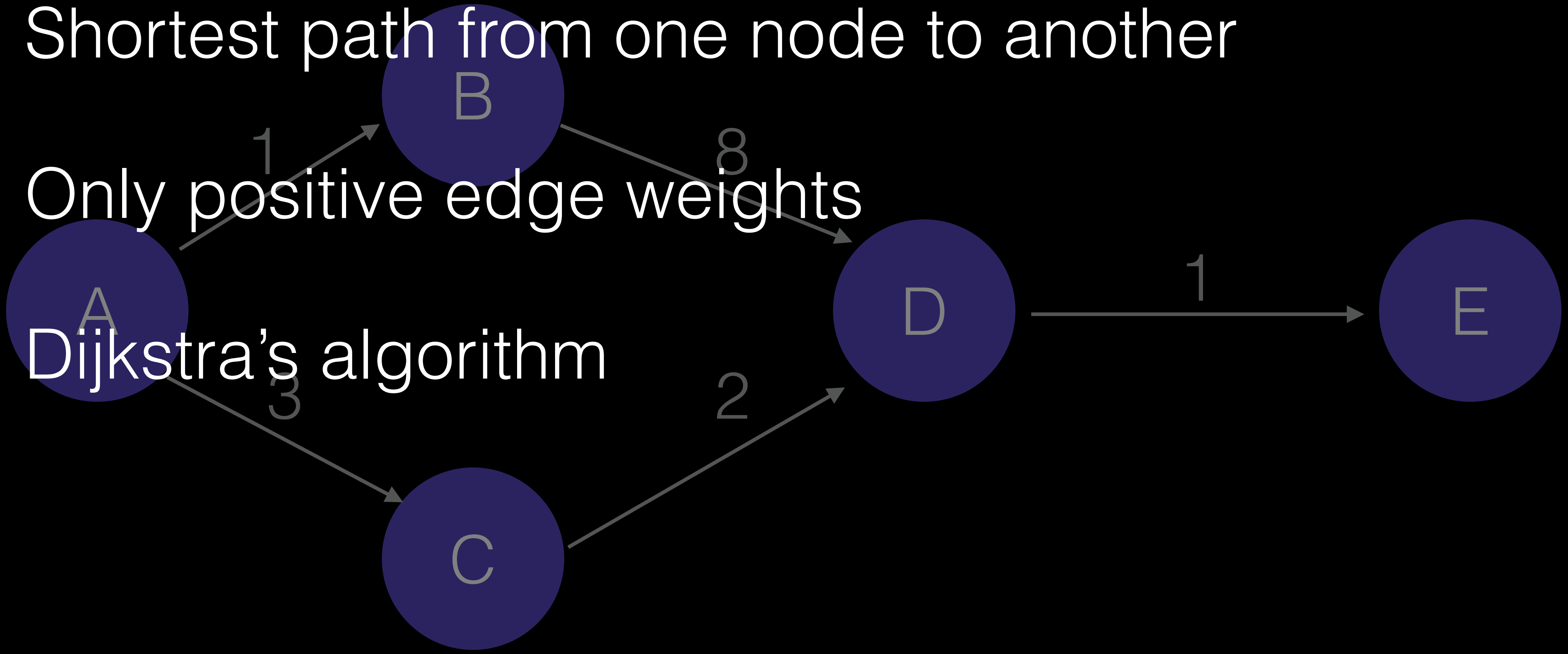
```
2
6 3 3
A 1
B 2
C 3
D 4
F 5
G 6
ABC
FEC
DBG
2 6 3
A 100
B 1000
BBBBBB
AAAAEB
BBBBBB
```

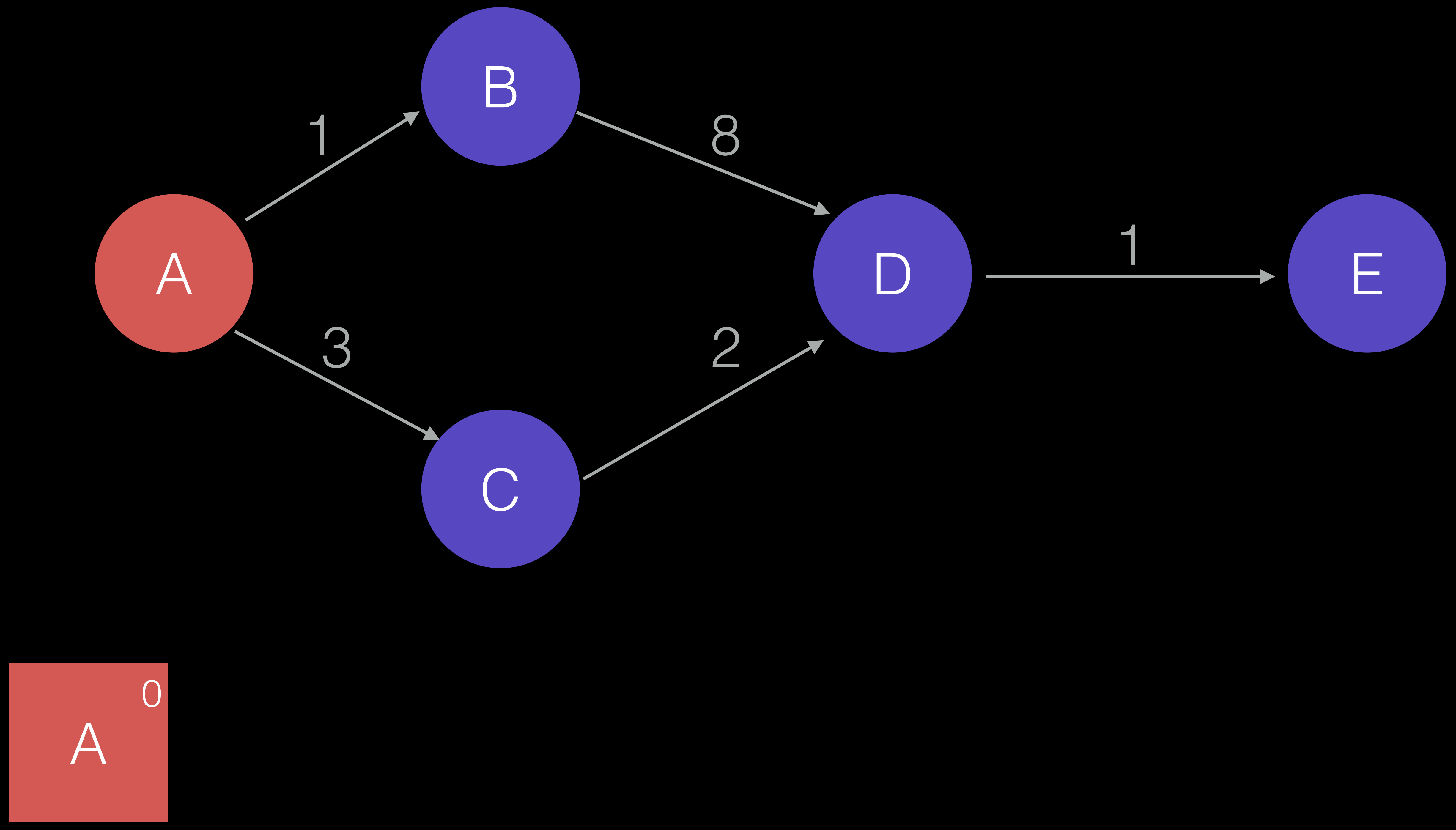
Sample Output

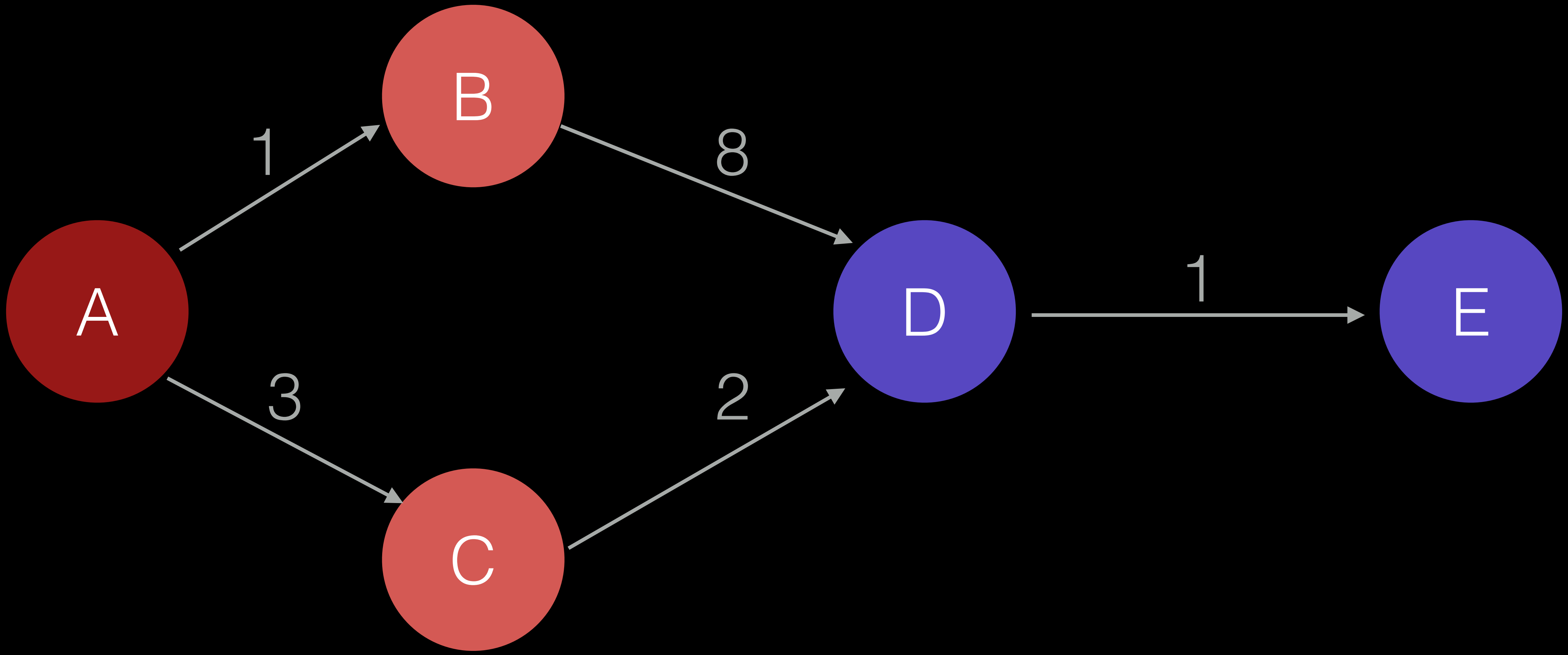
```
2
400
```

Weighted Shortest Path

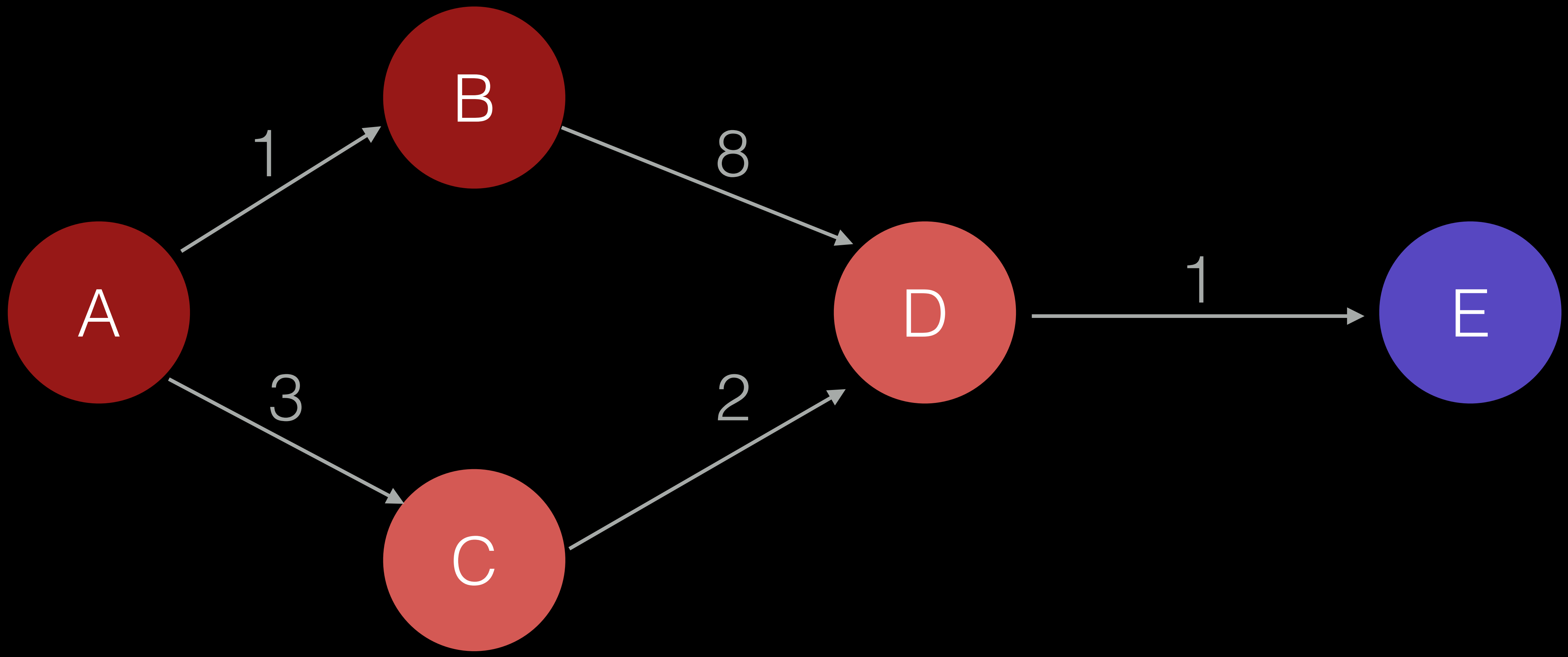
- Shortest path from one node to another
- Only positive edge weights
- Dijkstra’s algorithm



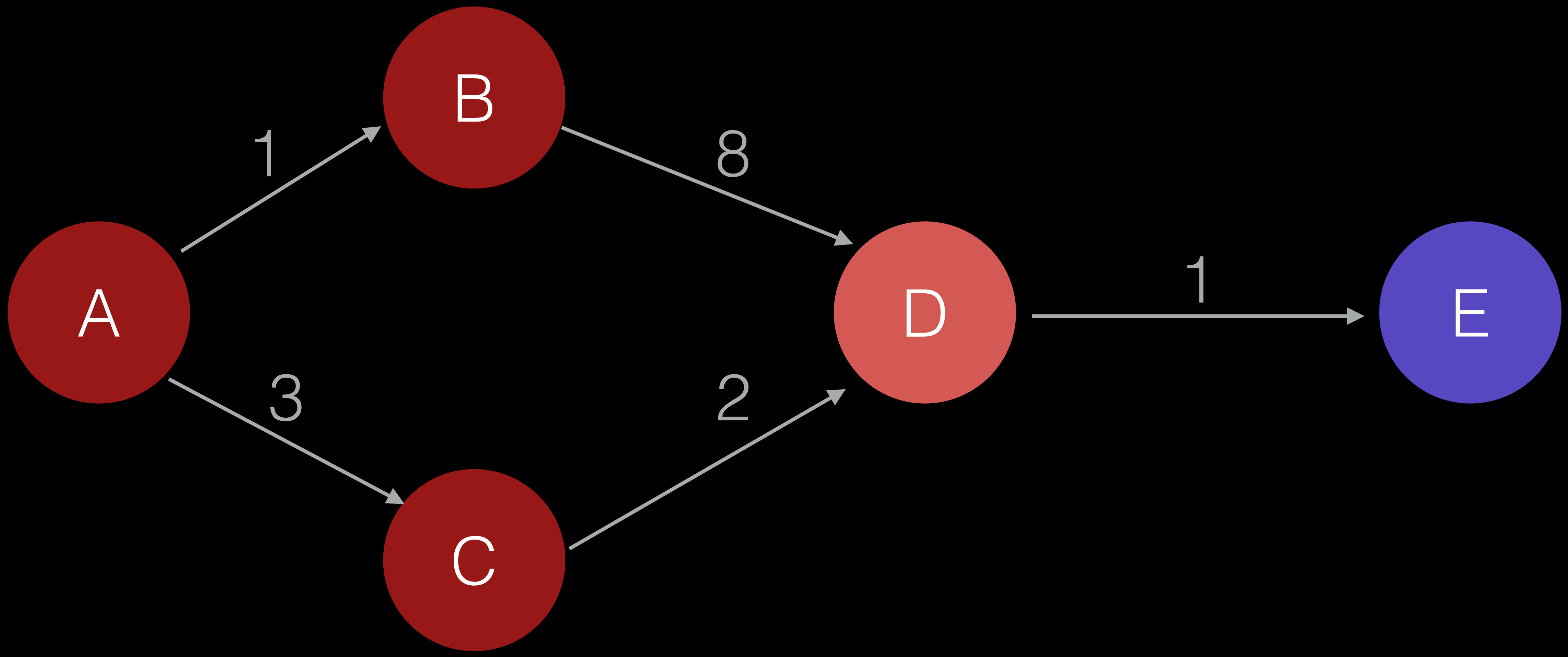

A
B
C
D
E
1
8
3
2
1
A
0



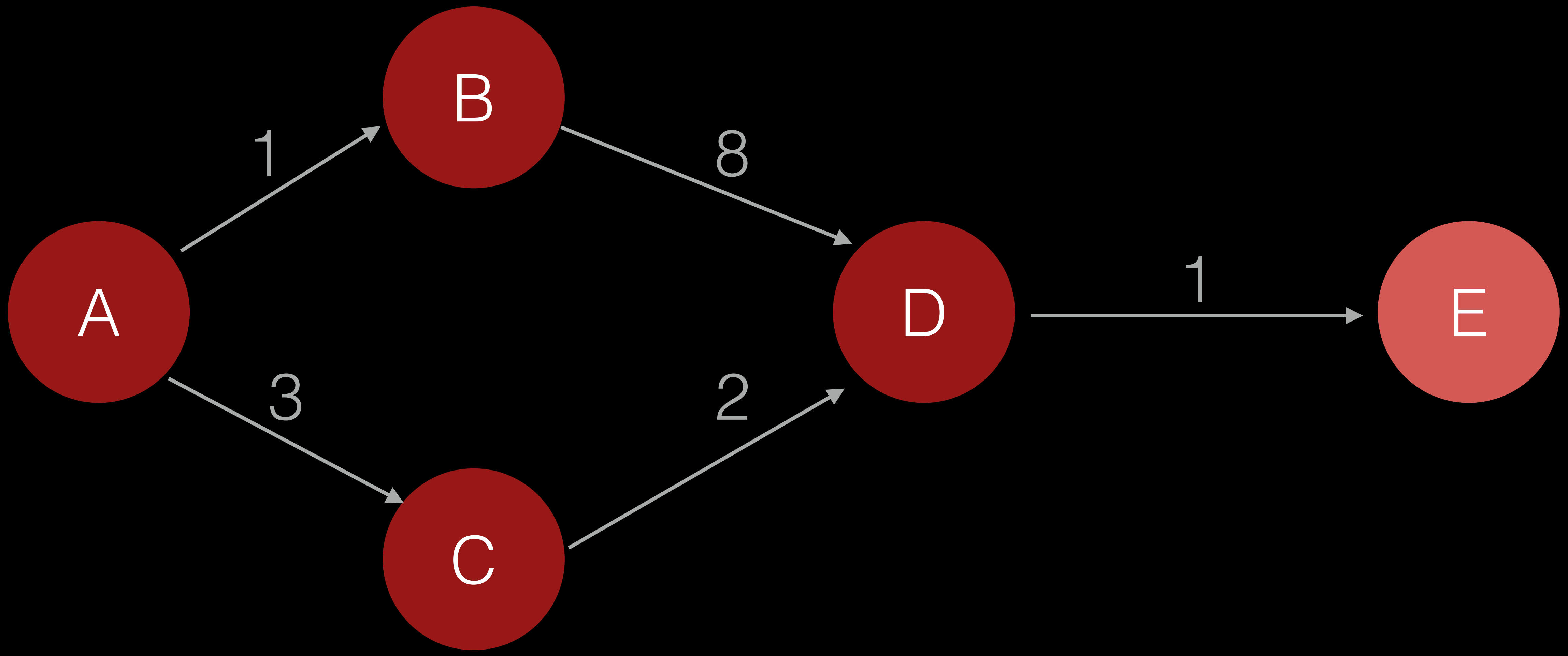
A	B	C
0	1	3



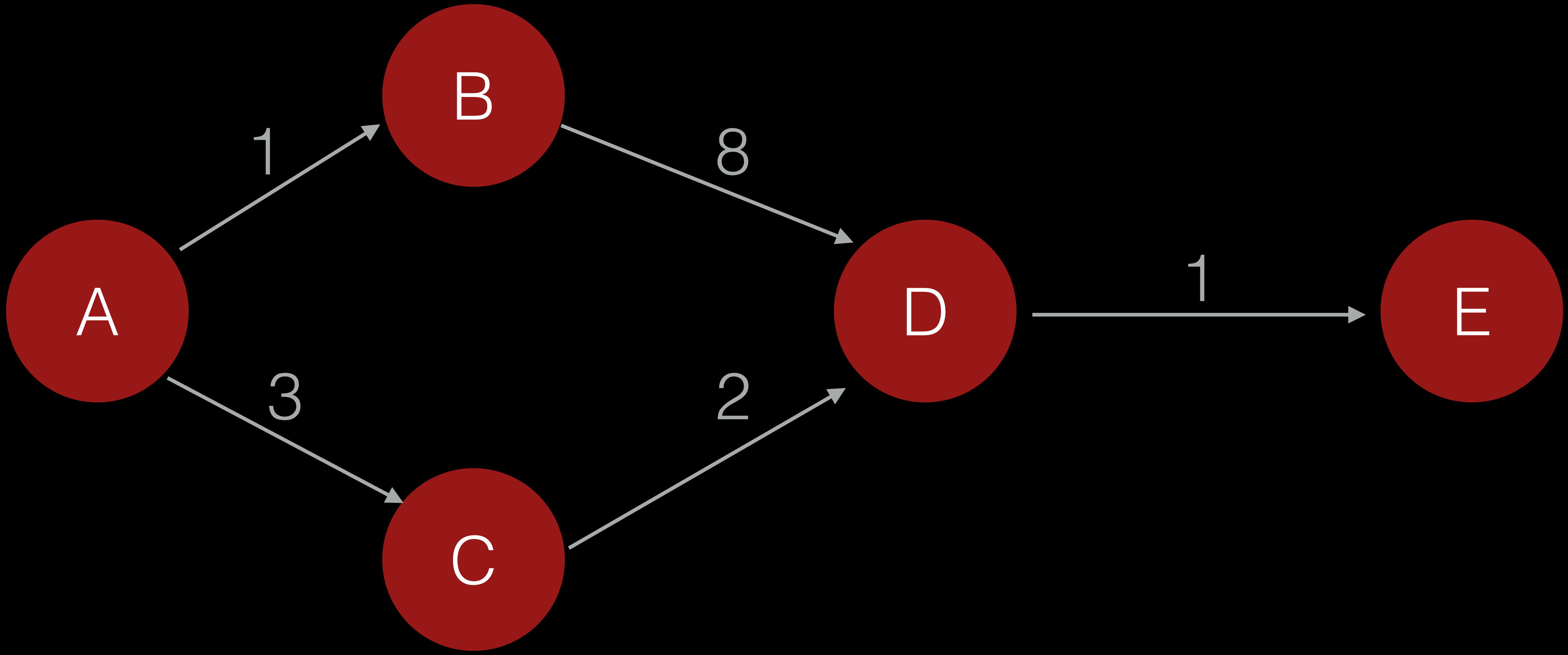
A 0	B 1	C 3	D 9



A	B	C	D
0	1	3	5



A	B	C	D	E
0	1	3	5	6



A	B	C	D	E
0	1	3	5	6

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

public class dijkstra {

  public static void main(String[] args) {
    Node a = new Node();
    Node b = new Node();
    Node c = new Node();
    Node d = new Node();
    Node e = new Node();
    a.edges.put(b, 1);
    a.edges.put(c, 3);
    b.edges.put(d, 8);
    c.edges.put(d, 2);
    d.edges.put(e, 1);
    ArrayList<Node> allNodes = new ArrayList<Node>(Arrays.asList(new Node[]{e, d, c, b, a}));
    dijkstra(a, allNodes);
    System.out.printf("Distance to Node e is: %d\n", e.distance);
  }

  public static void dijkstra(Node root, ArrayList<Node> allNodes) {
    PriorityQueue<Node> q = new PriorityQueue<Node>();
    root.distance = 0;
    q.add(root);

    while(q.size() > 0) {
      Node u = q.poll();
      for(Node n : u.edges.keySet()) {
        if(n.distance == Integer.MAX_VALUE) { // Update the distance to node n
          q.remove(n);
        }
        n.distance = Math.min(n.distance, u.distance + u.edges.get(n));
        q.add(n);
      }
    }
  }
}

class Node implements Comparable<Node> {
  public HashMap<Node, Integer> edges = new HashMap<Node, Integer>();
  public int distance = Integer.MAX_VALUE;

  public int compareTo(Node o) {
    return (distance < o.distance) ? -1 : ((distance == o.distance) ? 0 : 1);
  }
}
```

```
public static void dijkstra(Node root, ArrayList<Node> allNodes) {
  PriorityQueue<Node> q = new PriorityQueue<Node>();
  root.distance = 0;
  q.add(root);

  while(q.size() > 0) {
    Node u = q.poll();
    for(Node n : u.edges.keySet()) {
      if(n.distance == Integer.MAX_VALUE) { // Update the distance to node n
        q.remove(n);
      }
      n.distance = Math.min(n.distance, u.distance + u.edges.get(n));
      q.add(n);
    }
  }
}
```

```
  public static void bfs(Node root, ArrayList<Node> allNodes) {
    Queue<Node> q = new LinkedList<Node>();
    root.distance = 0;
    q.add(root);

    while(q.size() > 0) {
      Node u = q.poll();
      for(Node n : u.edges) {
        if(n.distance == Integer.MAX_VALUE) { // Has not been visited yet
          n.distance = u.distance + 1;
          q.add(n);
        }
      }
    }
  }
```

```
public static void dijkstra(Node root, ArrayList<Node> allNodes) {
  PriorityQueue<Node> q = new PriorityQueue<Node>();
  root.distance = 0;
  q.add(root);

  while(q.size() > 0) {
    Node u = q.poll();
    for(Node n : u.edges.keySet()) {
      if(n.distance == Integer.MAX_VALUE) { // Update the distance to node n
        q.remove(n);
      }
      n.distance = Math.min(n.distance, u.distance + u.edges.get(n));
      q.add(n);
    }
  }
}
```

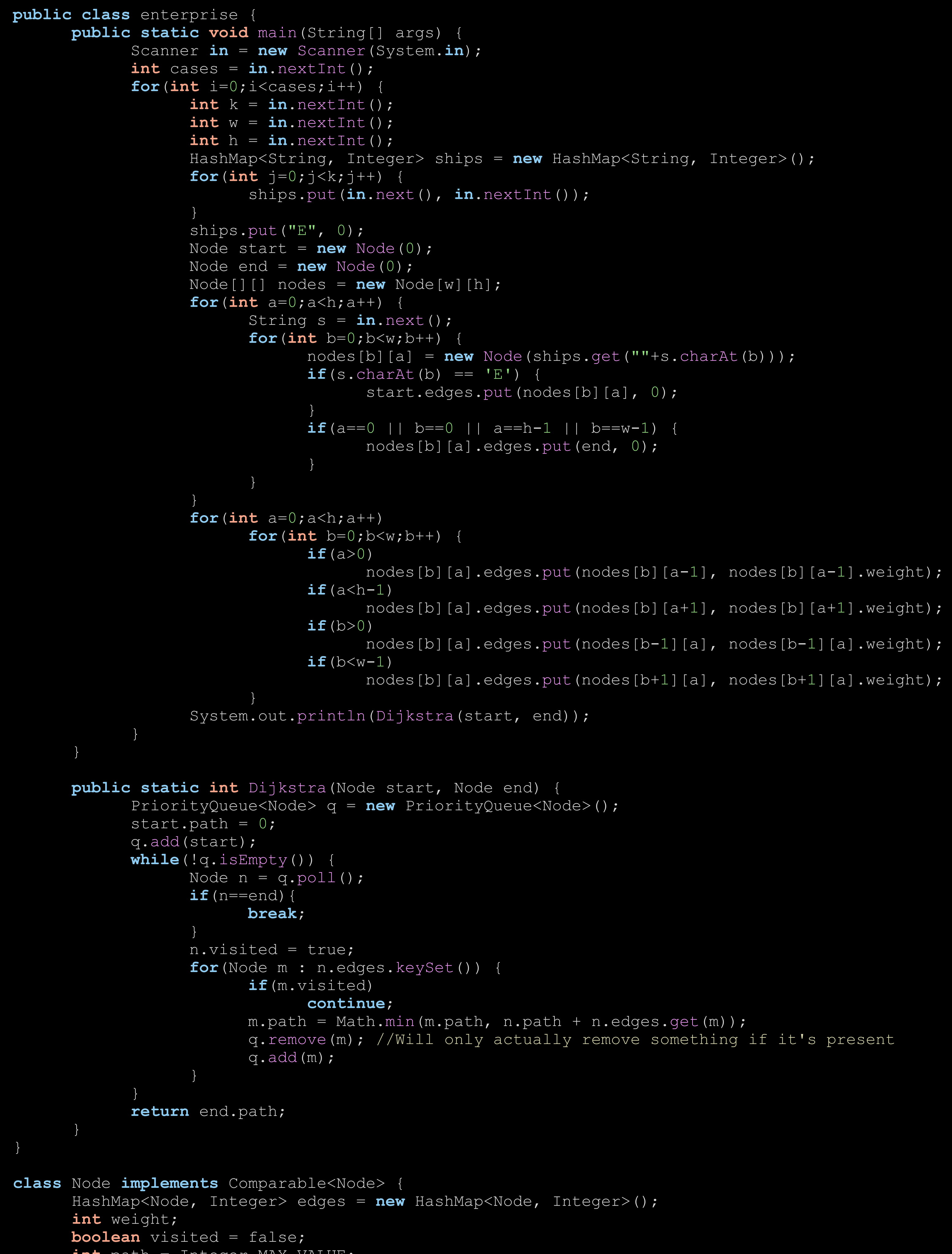
```
public static void bfs(Node root, ArrayList<Node> allNodes) {
  Queue<Node> q = new LinkedList<Node>();
  root.distance = 0;
  q.add(root);

  while(q.size() > 0) {
    Node u = q.poll();
    for(Node n : u.edges) {
      if(n.distance == Integer.MAX_VALUE) { // Has not been visited yet
        n.distance = u.distance + 1;
        q.add(n);
      }
    }
  }
}
```

```
public static void dijkstra(Node root, ArrayList<Node> allNodes) {
  PriorityQueue<Node> q = new PriorityQueue<Node>();
  root.distance = 0;
  q.add(root);

  while(q.size() > 0) {
    Node u = q.poll();
    for(Node n : u.edges.keySet()) {
      if(n.distance == Integer.MAX_VALUE) { // Update the distance to node n
        q.remove(n);
      }
      n.distance = Math.min(n.distance, u.distance + u.edges.get(n));
      q.add(n);
    }
  }
}
```

More about the contest



```java
public class enterprise {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        int cases = in.nextInt();
        for(int i=0;i<cases;i++) {
            int k = in.nextInt();
            int w = in.nextInt();
            int h = in.nextInt();
            HashMap<String, Integer> ships = new HashMap<String, Integer>();
            for(int j=0;j<k;j++) {
                ships.put(in.next(), in.nextInt());
            }
            ships.put("E", 0);
            Node start = new Node(0);
            Node end = new Node(0);
            Node[][] nodes = new Node[w][h];
            for(int a=0;a<h;a++) {
                String s = in.next();
                for(int b=0;b<w;b++) {
                    nodes[b][a] = new Node(ships.get(""+s.charAt(b)));
                    if(s.charAt(b) == 'E') {
                        start.edges.put(nodes[b][a], 0);
                    }
                    if(a==0 || b==0 || a==h-1 || b==w-1) {
                        nodes[b][a].edges.put(end, 0);
                    }
                }
            }
            for(int a=0;a<h;a++)
                for(int b=0;b<w;b++) {
                    if(a>0)
                        nodes[b][a].edges.put(nodes[b][a-1], nodes[b][a-1].weight);
                    if(a<h-1)
                        nodes[b][a].edges.put(nodes[b][a+1], nodes[b][a+1].weight);
                    if(b>0)
                        nodes[b][a].edges.put(nodes[b-1][a], nodes[b-1][a].weight);
                    if(b<w-1)
                        nodes[b][a].edges.put(nodes[b+1][a], nodes[b+1][a].weight);
                }
            System.out.println(Dijkstra(start, end));
        }
    }

    public static int Dijkstra(Node start, Node end) {
        PriorityQueue<Node> q = new PriorityQueue<Node>();
        start.path = 0;
        q.add(start);
        while(!q.isEmpty()) {
            Node n = q.poll();
            if(n==end){
                break;
            }
            n.visited = true;
            for(Node m : n.edges.keySet()) {
                if(m.visited)
                    continue;
                m.path = Math.min(m.path, n.path + n.edges.get(m));
                q.remove(m); //Will only actually remove something if it's present
                q.add(m);
            }
        }
        return end.path;
    }
}

class Node implements Comparable<Node> {
    HashMap<Node, Integer> edges = new HashMap<Node, Integer>();
    int weight;
    boolean visited = false;
    int path = Integer.MAX_VALUE;
```

	Problem A	Problem B	Problem C	Problem D	Total
Team 1	—	—	—	—	0 solved 0 mins
Team 2	—	—	—	—	0 solved 0 mins
Team 3	—	—	—	—	0 solved 0 mins

	Problem A	Problem B	Problem C	Problem D	Total
Team 3	—	—	Solved! 1 try 5 mins	—	1 solved 5 mins
Team 1	—	—	—	—	0 solved 0 mins
Team 2	—	—	—	—	0 solved 0 mins

	Problem A	Problem B	Problem C	Problem D	Total
Team 3	—	—	Solved! 1 try 5 mins	—	1 solved 5 mins
Team 2	Solved! 1 try 8 mins	—	—	—	1 solved 8 mins
Team 1	—	—	—	—	0 solved 0 mins

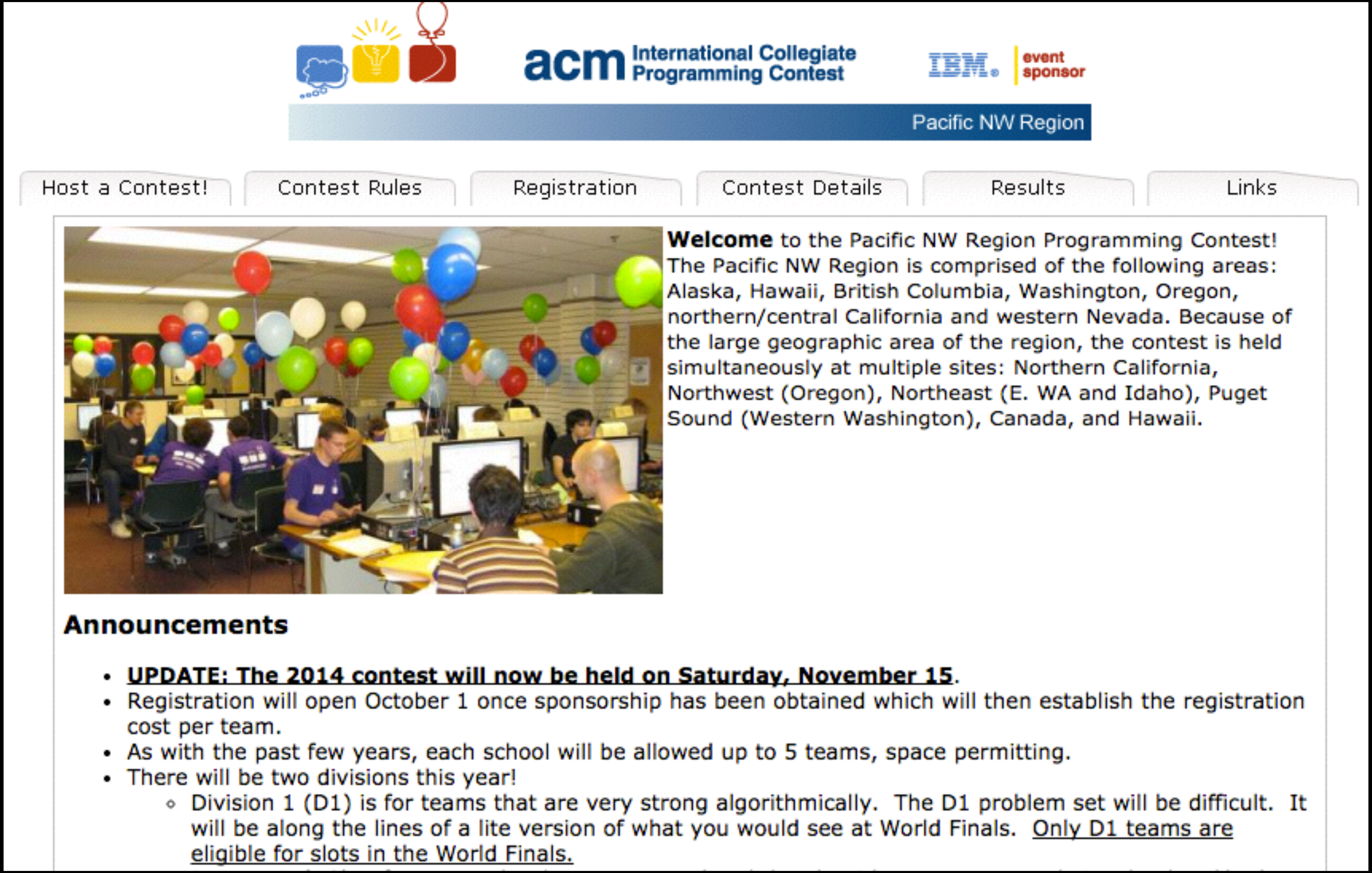
	Problem A	Problem B	Problem C	Problem D	Total
Team 3	—	—	Solved! 1 try 5 mins	—	1 solved 5 mins
Team 2	Solved! 1 try 8 mins	—	—	—	1 solved 8 mins
Team 1	—	—	Unsolved 1 try	—	0 solved 0 mins

	Problem A	Problem B	Problem C	Problem D	Total
Team 3	—	—	Solved! 1 try 5 mins	—	1 solved 5 mins
Team 2	Solved! 1 try 8 mins	—	—	—	1 solved 8 mins
Team 1	Solved! 1 try 14 mins	—	Unsolved 1 try	—	1 solved 14 mins

	Problem A	Problem B	Problem C	Problem D	Total
Team 1	Solved! 1 try 14 mins	—	Solved! 2 tries 18 mins	—	2 solved 14+18+20= 52 mins
Team 3	—	—	Solved! 1 try 5 mins	—	1 solved 5 mins
Team 2	Solved! 1 try 8 mins	—	—	—	1 solved 8 mins

Solving problems is good!

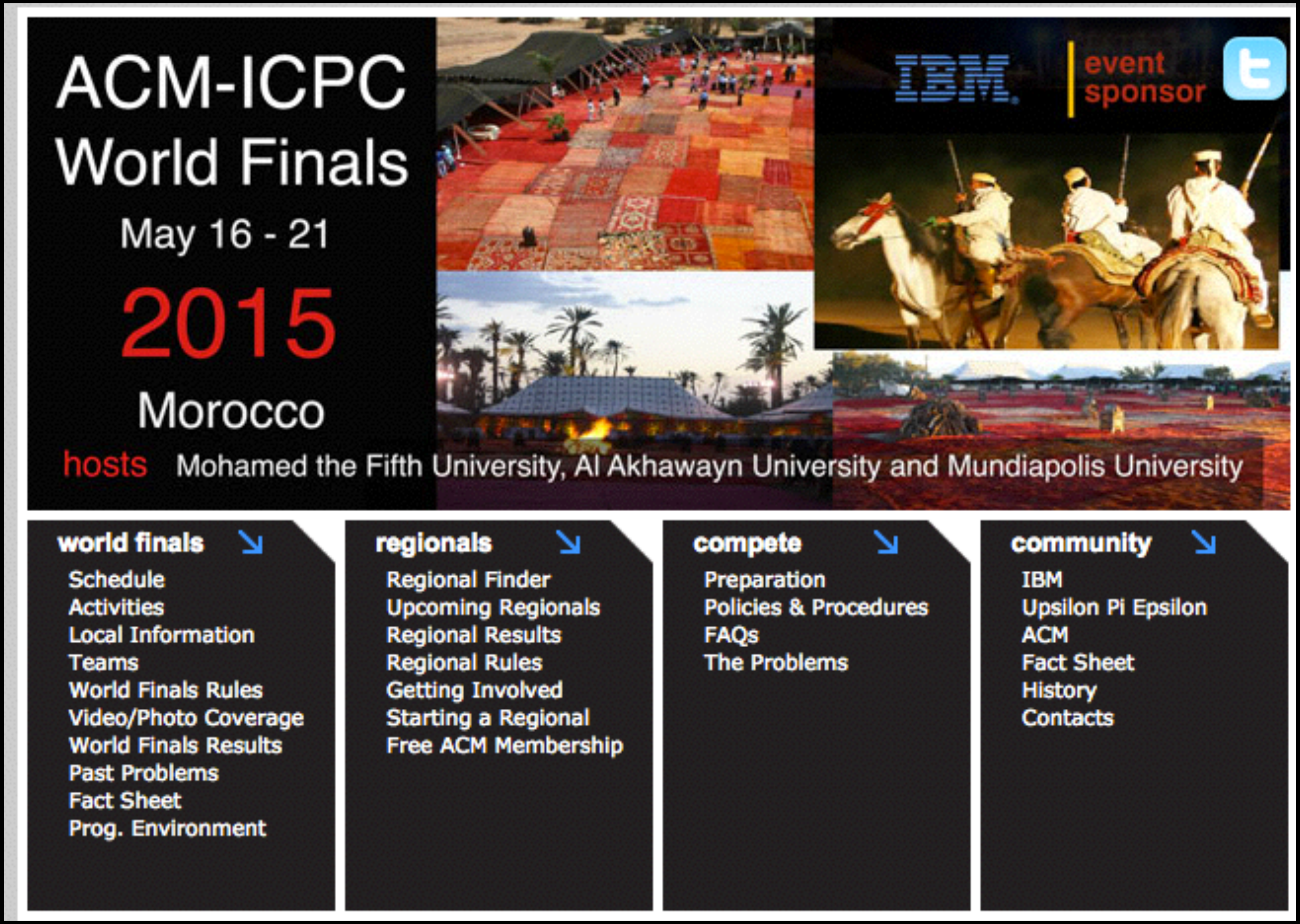
Guessing is bad, but only if you eventually figure out the solution



Welcome to the Pacific NW Region Programming Contest! The Pacific NW Region is comprised of the following areas: Alaska, Hawaii, British Columbia, Washington, Oregon, northern/central California and western Nevada. Because of the large geographic area of the region, the contest is held simultaneously at multiple sites: Northern California, Northwest (Oregon), Northeast (E. WA and Idaho), Puget Sound (Western Washington), Canada, and Hawaii.

Announcements

- **UPDATE: The 2014 contest will now be held on Saturday, November 15**.
- Registration will open October 1 once sponsorship has been obtained which will then establish the registration cost per team.
- As with the past few years, each school will be allowed up to 5 teams, space permitting.
- There will be two divisions this year!
    - Division 1 (D1) is for teams that are very strong algorithmically. The D1 problem set will be difficult. It will be along the lines of a lite version of what you would see at World Finals. Only D1 teams are eligible for slots in the World Finals.



ACM-ICPC World Finals

May 16 - 21

2015

Morocco

hosts Mohamed the Fifth University, Al Akhawayn University and Mundiapolis University

Shameless Plug #3

Organized practices

Organized Practices

- Saturday, October 4, 10am-4pm, CSE 006
- Saturday, October 11, 10am-4pm, CSE 002 & 006
- Sunday, October 19, 10am-4pm, CSE 006
- Other practices?
- Saturday, November 15

Questions?

Daniel Epstein

depstein@cs.washington.edu

https://github.com/depstein/programming-competitions