Soluciones OIFem 2020

Clasificatorio

Primos gemelos

Teoría

- Condicionales (if)
- Bucles (for)

Solución

Hay dos formas de resolver este problema: encontrar los primos entre 2 y 1000 y guardarlos en un conjunto o construir una función que mira si un número es primo. Pondremos este segundo método en las soluciones, pero os animamos a resolverlo usando un conjunto desordenado para practicar (buscad *Criba de Eratóstenes* en Google para aprender el algoritmo si no lo conocéis).

Usaremos un for para mirar todos los números impares hasta la raíz cuadrada de n. El motivo de esto es que en un if anterior miraremos si es par y, en caso contrario, sabremos que n no es múltiplo de ningún otro múltiplo de 2.

Código

C++

```cpp
#include <cmath>
#include <cstdio>
#include <vector>
#include <iostream>
#include <algorithm>
using namespace std;

bool esPrimo(int n) {
    if (n % 2 == 0)
        return false;
    int i = 3; // los impares cuya raíz cuadrada es menor que 3 y que son mayores que 2 (3, 5 y 7) son todos primos, por lo que podemos empezar por este valor de i
    while(i*i <= n) {
        if (n % i == 0) return false;
        i += 2;
    }
```

```
    return true;
}

int main() {
    int t, x;
    cin >> t;
    while(t--) {
        cin >> x;
        if (esPrimo(x) && esPrimo(x+2)) cout << "tiene gemelo mayor\n";
        else cout << "no tiene gemelo mayor\n";
    }
    return 0;
}
```

Java

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

public class Solution {
    public static boolean esPrimo(int n) {
        if (n % 2 == 0)
            return false;
        int i = 3; // los impares cuya raíz cuadrada es menor que 3 y que son mayores que 2 (3, 5 y 7) son todos primos, por lo que podemos empezar por este valor de i
        while(i*i <= n) {
            if (n % i == 0) return false;
            i += 2;
        }
        return true;
    }

    public static void main(String[] args) {
        int t, n;
        Scanner s = new Scanner(System.in);
        t = s.nextInt();
        for (int i = 0; i < t; i++) {
            n = s.nextInt();
            if (esPrimo(n) && esPrimo(n+2))
                System.out.println("tiene gemelo mayor");
            else
                System.out.println("no tiene gemelo mayor");
        }
        s.close();
    }
}
```

Python

```
def esPrimo(n):
    if n % 2 == 0:
        return False
```

```
    i = 3 # los impares cuya raíz cuadrada es menor que 3 y que son mayores que 2
    ↪  (3, 5 y 7) son todos primos, por lo que podemos empezar por este valor de
    ↪  i
    while(i*i <= n):
        if (n % i == 0):
            return False
        i += 2
    return True

t = int(input())
for k in range(t):
    x = int(input())
    if (esPrimo(x) and esPrimo(x+2)):
        print("tiene gemelo mayor")
    else:
        print("no tiene gemelo mayor")
```

Amañando las notas

Teoría

- Búsqueda binaria (opcional)

Solución

La solución era una ecuación cuadrática, que formaba una curva alrededor de los suspensos, aunque podía aproximarse con ecuaciones lineales.

La idea era que probarais ecuaciones lineales y cuadráticas y que encontrarais a través de búsqueda binaria (con ensayo y error) a mano los coeficientes.

Código

C++

```cpp
#include <cmath>
#include <cstdio>
#include <vector>
#include <iostream>
#include <algorithm>
using namespace std;


int main() {
    double x, y;
    cin >> x >> y;
    cout << (y >= -0.2*x*x + 0.72*x + 4.952);
    return 0;
}
```

Java

```java
import java.io.*;
import java.util.*;
import java.text.*;
import java.math.*;
import java.util.regex.*;

public class Solution {

    public static void main(String[] args) {
        Scanner s = new Scanner(System.in);
        double x, y;
        x = s.nextDouble();
        y = s.nextDouble();
        if(y >= -0.2*x*x + 0.72*x + 4.952)
            System.out.println(1);
        else
            System.out.println(0);
        s.close();
```

```
    }
}
```

Python

```
x, y = map(float, input().split())
print(int(y >= -0.2*x*x + 0.72*x + 4.952))
```

Potencias enormes

Teoría

- Condicionales
- Potencias

Solución

La solución se basaba en una observación: $1.64^2 = 2.6896$. Sabiendo esto, el resto se facilitaba. $1.64 < 2 < 2.6896$, por lo que necesitábamos $2j \geq i - 1$. La solución consiste en comprobar esto.

Código

C++

```cpp
#include <cmath>
#include <cstdio>
#include <vector>
#include <iostream>
#include <algorithm>
using namespace std;


int main() {
    int n;
    long long int i, j;
    cin >> n;
    while(n--) {
        cin >> i >> j;
        if (2*j >= i-1)
            cout << "Verdadero\n";
        else
            cout << "Falso\n";
    }
    return 0;
}
```

Java

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

public class Solution {

    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        Double i, j;
        int n = input.nextInt();
        for (int k = 0; k < n; k++) {
            i = input.nextDouble();
            j = input.nextDouble();
            if (2*j >= i-1)
```

```
                System.out.println("Verdadero");
            else
                System.out.println("Falso");
        }
        input.close();
    }
}
```

Python

```
n = int(input())
for k in range(n):
    i, j = map(int, input().split())
    if 2*j >= i-1:
        print("Verdadero")
    else:
        print("Falso")
```

Recargar con manjar

Teoría

- Recursión
- Backtracking
- Grafos implícitos

Solución

La solución consistía en usar backtracking para explorar todos los caminos posibles y quedarnos con el mejor.

Código

C++

```cpp
#include <iostream>
#include <vector>

using namespace std;

int busca(int of, int oc, int f, int c, int contadorTramos, vector<vector<int> > &mapa) { // f,c es la posición en la que estamos, of,oc es la posición de la que venimos para evitar volver hacia atrás
    int m = mapa.size();
    int n = mapa[0].size();

    // Casos base
    if (f < 0 || f >= m || c < 0 || c >= n) return -1; // Posición inválida
    if (mapa[f][c] == -1 || mapa[f][c] == -2) return -1; // Este camino no lleva a Chicago
    if (contadorTramos > 5) return -1; // El vehículo se ha descargado
    if (f == 0 && c == n-1) return 0; // Hemos llegado a Chicago

    int v = 0;
    if (mapa[f][c] != 0) { // Estamos en una estación de recarga
        contadorTramos = 0; // Reiniciar el contador de tramos
        v++; // Sumar uno al contador de estaciones de servicio
    } else { // Estamos en un tramo de carretera
        contadorTramos++; // Sumar 1 al contador de tramos de carretera si no estamos en una estación
    }

    // Casos recursivos. Comprobamos que no volvamos hacia atrás, al no haber ciclos este es nuestro único movimiento válido prohibido
    int max_s = -1;
    if (!(f == of && c-1 == oc)) {
        int s = busca(f, c, f, c-1, contadorTramos, mapa); // Arriba
        if (s > max_s) max_s = s;
    }
    if (!(f == of && c+1 == oc)) {
        int s = busca(f, c, f, c+1, contadorTramos, mapa); // Abajo
```

```
        if (s > max_s) max_s = s;
    }
    if (!(f+1 == of && c == oc)) {
        int s = busca(f, c, f+1, c, contadorTramos, mapa); // Derecha
        if (s > max_s) max_s = s;
    }
    if (!(f-1 == of && c == oc)) {
        int s = busca(f, c, f-1, c, contadorTramos, mapa); // Izquierda
        if (s > max_s) max_s = s;
    }
    if (max_s < 0) return -1;
    else return v + max_s;
}

int main()
{
    int n, m;

    // Leemos el mapa
    cin >> m >> n;
    vector<vector<int> > vvi(m);
    for (int i = 0; i < (int) vvi.size(); i++) {
        vvi[i] = vector<int>(n);
        for (int j = 0; j < (int) vvi[i].size(); j++) {
            cin >> vvi[i][j];
        }
    }

    cout << busca(m-1, 0, m-1, 0, -1, vvi); // Ponemos el contador de tramos de carretera a -1 para que se incremente en la casilla inicial a 0
}
```

Java

```
import java.io.*;
import java.util.*;
import java.text.*;
import java.math.*;
import java.util.regex.*;

public class Solution {
    public static int busca(int of, int oc, int f, int c, int contadorTramos, int[][] mapa) {
        // f,c es la posición en la que estamos, of,oc es la posición de la que venimos para evitar volver hacia atrás
        int m = mapa.length;
        int n = mapa[0].length;

        // Casos base
        if (f < 0 || f >= m || c < 0 || c >= n) return -1; // Posición inválida
        if (mapa[f][c] == -1 || mapa[f][c] == -2) return -1; // Este camino no lleva a Chicago
        if (contadorTramos > 5) return -1; // El vehículo se ha descargado
        if (f == 0 && c == n-1) return 0; // Hemos llegado a Chicago

```

```
        int v = 0;
        if (mapa[f][c] != 0) { // Estamos en una estación de recarga
            contadorTramos = 0; // Reiniciar el contador de tramos
            v++; // Sumar uno al contador de estaciones de servicio
        } else { // Estamos en un tramo de carretera
            contadorTramos++; // Sumar 1 al contador de tramos de carretera si no
            ↪  estamos en una estación
        }

        // Casos recursivos. Comprobamos que no volvamos hacia atrás, al no haber
        ↪  ciclos este es nuestro único movimiento válido prohibido
        int max_s = -1;
        if (!(f == of && c-1 == oc)) {
            int s = busca(f, c, f, c-1, contadorTramos, mapa); // Arriba
            if (s > max_s) max_s = s;
        }
        if (!(f == of && c+1 == oc)) {
            int s = busca(f, c, f, c+1, contadorTramos, mapa); // Abajo
            if (s > max_s) max_s = s;
        }
        if (!(f+1 == of && c == oc)) {
            int s = busca(f, c, f+1, c, contadorTramos, mapa); // Derecha
            if (s > max_s) max_s = s;
        }
        if (!(f-1 == of && c == oc)) {
            int s = busca(f, c, f-1, c, contadorTramos, mapa); // Izquierda
            if (s > max_s) max_s = s;
        }
        if (max_s < 0) return -1;
        else return v + max_s;
    }

    public static void main(String[] args) {
        Scanner s = new Scanner(System.in);
        int n, m;

        // Leemos el mapa
        m = s.nextInt();
        n = s.nextInt();
        int [][] vvi = new int[m][n];
        for (int i = 0; i < m; i++) {
            for (int j = 0; j < n; j++) {
                vvi[i][j] = s.nextInt();
            }
        }

        System.out.println(busca(m-1, 0, m-1, 0, -1, vvi)); // Ponemos el contador
        ↪  de tramos de carretera a -1 para que se incremente en la casilla
        ↪  inicial a 0
        s.close();
    }
}
```

Python

```python
mapa = []
def busca(of, oc, f, c, contadorTramos):
    global mapa
    # f,c es la posición en la que estamos, of,oc es la posición de la que venimos para evitar volver hacia atrás
    m = len(mapa)
    n = len(mapa[0])

    # Casos base
    if (f < 0 or f >= m or c < 0 or c >= n):
        return -1 # Posición inválida
    if (mapa[f][c] == -1 or mapa[f][c] == -2):
        return -1 # Este camino no lleva a Chicago
    if (contadorTramos > 5):
        return -1 # El vehículo se ha descargado
    if (f == 0 and c == n-1):
        return 0 # Hemos llegado a Chicago

    v = 0
    if (mapa[f][c] != 0): # Estamos en una estación de recarga
        contadorTramos = 0 # Reiniciar el contador de tramos
        v += 1 # Sumar uno al contador de estaciones de servicio
    else: # Estamos en un tramo de carretera
        contadorTramos += 1 # Sumar 1 al contador de tramos de carretera si no estamos en una estación

    #Casos recursivos. Comprobamos que no volvamos hacia atrás, al no haber ciclos este es nuestro único movimiento válido prohibido
    max_s = -1
    if (not(f == of and c-1 == oc)):
        s = busca(f, c, f, c-1, contadorTramos) # Arriba
        if (s > max_s):
            max_s = s
    if (not(f == of and c+1 == oc)):
        s = busca(f, c, f, c+1, contadorTramos) # Abajo
        if (s > max_s):
            max_s = s
    if (not(f+1 == of and c == oc)):
        s = busca(f, c, f+1, c, contadorTramos) # Derecha
        if (s > max_s):
            max_s = s
    if (not(f-1 == of and c == oc)):
        s = busca(f, c, f-1, c, contadorTramos) # Izquierda
        if (s > max_s):
            max_s = s
    if (max_s < 0):
        return -1
    else:
        return v + max_s

m, n = map(int, input().split())
for i in range(m):
    mapa.append([])
    mapa[i] = list(map(int, input().split()))
```

```
52 print(busca(m-1, 0, m-1, 0, -1)) # Ponemos el contador de tramos de carretera a -1
   ↪  para que se incremente en la casilla inicial a 0
```

Tragaperras

Teoría

- Programación dinámica

Solución

La solución consistía en usar programación dinámica para probar todas las posibilidades. Para cada número de intentos de 1 a 1000, íbamos desde la máquina 0 hasta la $n-1$ viendo si había alguna posibilidad. Si en algún momento esta existía, retornábamos el número actual de intentos, ya que los números menores ya se habían descartado y nos interesaba el mínimo.

Por lo tanto, la estructura de nuestra DP es `DP[maquina][intentos]`, donde `DP[maquina][intentos] = max(DP[maquina-1][ap = 0..intentos] + apostar(intentos - ap)`. Es decir, que la fila con índice `maquina` guarda el máximo alcanzable en las primeras `maquina+1` máquinas y, para calcular una nueva fila de la columna actual, probamos a apostar todas las veces entre 0 y el número de intentos actual en la máquina actual, apostando el resto en las máquinas anteriores. Además, guardamos el total máximo con x apuestas en la i-ésima máquina y el mínimo alcanzado en el camino para llegar a este máximo, ya que si en algún momento tenemos menos de c monedas, no podremos seguir apostando.

Código

C++

```cpp
#include <iostream>
#include <vector>
#include <climits>
using namespace std;

int calcular(vector<vector<int>> & secuencias, int objetivo, int coste) {
    if (coste >= objetivo)
        return 0; // ya tenemos la cantidad buscada
    int N = (int) secuencias.size();
    // hacemos la DP
    vector<vector<int>> DP(N, vector<int>(1001)), totales(N, vector<int>(1001)),
    ↪  minimos(N, vector<int>(1001));
    for (int i = 0; i < N; i++) {
        DP[i][0] = coste;
        totales[i][0] = 0;
        minimos[i][0] = 0;
    }
    DP[0][0] = totales[0][0] = coste;
    int m;
    for (int intentos = 1; intentos <= 1000; intentos++) {
        m = (int) secuencias[0].size();
        totales[0][intentos] = totales[0][intentos-1] +
        ↪  secuencias[0][(intentos-1+m)%m] - coste;
        minimos[0][intentos] = min(minimos[0][intentos-1], totales[0][intentos]);
        DP[0][intentos] = totales[0][intentos];
        if (DP[0][intentos] >= objetivo)
            return intentos;
        if (minimos[0][intentos] < 0)
            DP[0][intentos] = -1e9;
```

```
        for (int maq = 1; maq < N; maq++) {
            m = (int) secuencias[maq].size();
            totales[maq][intentos] = totales[maq][intentos-1] +
            ↪ secuencias[maq][(intentos-1+m)%m] - coste;
            minimos[maq][intentos] = min(minimos[maq][intentos-1],
            ↪ totales[maq][intentos]);
            for (int apuestas = 0; apuestas <= intentos; apuestas++) {
                if (DP[maq-1][intentos - apuestas] < coste)
                    continue; // no podemos apostar
                if (DP[maq-1][intentos - apuestas] + minimos[maq][apuestas] <
                ↪ coste)
                    continue; // no llegaremos con posibilidades de seguir
                DP[maq][intentos] = max(DP[maq][intentos], DP[maq-1][intentos -
                ↪ apuestas] + totales[maq][apuestas]);
            }
            if (DP[maq][intentos] >= objetivo)
                return intentos; // hemos encontrado un mínimo de intentos
        }
    }
    return -1;
}

int main() {
    ios::sync_with_stdio(false);
    cin.tie(NULL);
    int t, n, objetivo, c, m;
    vector<vector<int>> secuencias;
    cin >> t;
    while(t--) {
        cin >> n >> objetivo >> c;
        secuencias = vector<vector<int>>(n);
        for (int i = 0; i < n; i++) {
            cin >> m;
            secuencias[i] = vector<int>(m);
            for (int j = 0; j < m; j++)
                cin >> secuencias[i][j];
        }
        cout << calcular(secuencias, objetivo, c) << '\n';
    }
    return 0;
}
```

Java

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

public class Solution {
    public static int calcular(ArrayList<ArrayList<Integer>> secuencias, int
    ↪ objetivo, int coste) {
        if (coste >= objetivo)
            return 0; // ya tenemos la cantidad buscada
        int N = (int) secuencias.size();
        // hacemos la DP
```

```
        int[][] DP = new int[N][1001], totales = new int[N][1001], minimos = new int[N][1001];
        for (int i = 0; i < N; i++) {
            DP[i][0] = coste;
            totales[i][0] = 0;
            minimos[i][0] = 0;
        }
        DP[0][0] = totales[0][0] = coste;
        int m;
        for (int intentos = 1; intentos <= 1000; intentos++) {
            m = (int) secuencias.get(0).size();
            totales[0][intentos] = totales[0][intentos-1] + secuencias.get(0).get((intentos-1+m)%m) - coste;
            minimos[0][intentos] = Math.min(minimos[0][intentos-1], totales[0][intentos]);
            DP[0][intentos] = totales[0][intentos];
            if (DP[0][intentos] >= objetivo)
                return intentos;
            if (minimos[0][intentos] < 0)
                DP[0][intentos] = -1000000000;
            for (int maq = 1; maq < N; maq++) {
                m = (int) secuencias.get(maq).size();
                totales[maq][intentos] = totales[maq][intentos-1] + secuencias.get(maq).get((intentos-1+m)%m) - coste;
                minimos[maq][intentos] = Math.min(minimos[maq][intentos-1], totales[maq][intentos]);
                for (int apuestas = 0; apuestas <= intentos; apuestas++) {
                    if (DP[maq-1][intentos - apuestas] < coste)
                        continue; // no podemos apostar
                    if (DP[maq-1][intentos - apuestas] + minimos[maq][apuestas] < coste)
                        continue; // no llegaremos con posibilidades de seguir
                    DP[maq][intentos] = Math.max(DP[maq][intentos], DP[maq-1][intentos - apuestas] + totales[maq][apuestas]);
                }
                if (DP[maq][intentos] >= objetivo)
                    return intentos; // hemos encontrado un mínimo de intentos que no es igual a 0
            }
        }
        return -1;
    }

    public static void main(String[] args) throws IOException {
        BufferedReader bufferedRead = new BufferedReader(new InputStreamReader(System.in));
        StringTokenizer stringTok = new StringTokenizer(bufferedRead.readLine());
        int t, n, objetivo, c, m;
        t = Integer.parseInt(stringTok.nextToken());
        ArrayList<ArrayList<Integer>> secuencias;
        for (int k = 0; k < t; k++) {
            stringTok = new StringTokenizer(bufferedRead.readLine());
            n = Integer.parseInt(stringTok.nextToken());
            objetivo = Integer.parseInt(stringTok.nextToken());
            c = Integer.parseInt(stringTok.nextToken());
```

```
            secuencias = new ArrayList<ArrayList<Integer>>();
            for (int i = 0; i < n; i++) {
                stringTok = new StringTokenizer(bufferedRead.readLine());
                m = Integer.parseInt(stringTok.nextToken());
                secuencias.add(new ArrayList<Integer>());
                for (int j = 0; j < m; j++) {
                    secuencias.get(i).add(Integer.parseInt(stringTok.
                    ↪  nextToken()));
                }
            }
            System.out.println(calcular(secuencias, objetivo, c));
        }
    }
}
```

Python

```
secuencias = []
objetivo = []
coste = 0

def calcular():
    if (coste >= objetivo):
        return 0 # ya tenemos la cantidad buscada
    N = len(secuencias)
    # hacemos la DP
    DP = [[0 for i in range(1001)] for j in range(1001)]
    totales = [[0 for i in range(1001)] for j in range(1001)]
    minimos = [[0 for i in range(1001)] for j in range(1001)]
    for i in range(n):
        DP[i][0] = coste
    DP[0][0] = totales[0][0] = coste;
    m = 0
    for intentos in range(1, 1001):
        m = len(secuencias[0])
        totales[0][intentos] = totales[0][intentos-1] +
        ↪  secuencias[0][(intentos-1+m)%m] - coste
        minimos[0][intentos] = min(minimos[0][intentos-1], totales[0][intentos])
        DP[0][intentos] = totales[0][intentos]
        if (DP[0][intentos] >= objetivo):
            return intentos
        if (minimos[0][intentos] < 0):
            DP[0][intentos] = -1e9
        for maq in range(1, N):
            m = len(secuencias[maq])
            totales[maq][intentos] = totales[maq][intentos-1] +
            ↪  secuencias[maq][(intentos-1+m)%m] - coste
            minimos[maq][intentos] = min(minimos[maq][intentos-1],
            ↪  totales[maq][intentos])
            for apuestas in range(intentos+1):
                if (DP[maq-1][intentos - apuestas] < coste):
                    continue # no podemos apostar
                if (DP[maq-1][intentos - apuestas] + minimos[maq][apuestas] <
                ↪  coste):
                    continue # no llegaremos con posibilidades de seguir
```

```
                DP[maq][intentos] = max(DP[maq][intentos], DP[maq-1][intentos -
                ↪  apuestas] + totales[maq][apuestas])
            if (DP[maq][intentos] >= objetivo):
                return intentos # hemos encontrado un mínimo de intentos
    return -1

t = int(input())
for k in range(t):
    n, objetivo, coste = map(int, input().split())
    secuencias = []
    for i in range(n):
        inp = list(map(int, input().split()))
        m = inp[0]
        secuencias.append([])
        for j in range(1, m+1):
            secuencias[i].append(inp[j])
    print(calcular())
```

Liga de caballos

Teoría

- Ordenamiento
- Programación dinámica
- Mapas

Solución

Creamos un mapa que pase del nombre del caballo a su puesto. Ordenamos las carreras por la hora a la que empiezan y creamos una DP con complejidad cuadrática (al haber solo 10 carreras no es problemático), donde para cada carrera su multiplicador sea el máximo multiplicador tras otra carrera que termine antes que esta empiece. Finalmente, imprimimos `DP[9]`.

Nota: aunque yo lo haya resuelto con structs, se puede resolver con parejas anidadas o similares, como hacíamos para guardar tríos.

Código

C++

```cpp
#include <iostream>
#include <vector>
#include <map>
#include <algorithm>
#include <iomanip>
using namespace std;

// Opción alternativa vista en clase: parejas anidadas
struct carrera {
    int pos1, pos2;
    string hora;
    double gana1, emp, gana2;
    carrera(int p1, int p2, string h, double g1, double e, double g2) {
        pos1 = p1;
        pos2 = p2;
        hora = h;
        gana1 = g1;
        emp = e;
        gana2 = g2;
    }
};

// Para poder ordenar las carreras (opción alternativa vista en clase: comparador custom)
bool operator < (const carrera &A, const carrera &B){
    return A.hora <= B.hora;
}

bool operator == (const carrera &A, const carrera &B){
    return A.hora == B.hora;
}
```

```
31
32  double multiplicador(carrera & P) {
33      if (abs(P.pos1 - P.pos2) <= 4)
34          return P.emp;
35      else if(P.pos1 > P.pos2)
36          return P.gana2;
37      else
38          return P.gana1;
39  }
40
41  string menos1h30(string hora) {
42      int hh = stoi(hora.substr(0,2)), mm = stoi(hora.substr(3, 5));
43      if (mm >= 30) {
44          mm -= 30;
45          hh--;
46      } else {
47          mm += 30;
48          hh -= 2;
49      }
50      string h = to_string(hh), m = to_string(mm);
51      if ((int) h.size() == 1)
52          h = "0" + h;
53      if ((int) m.size() == 1)
54          m = "0" + m;
55      return h + ":" + m;
56  }
57
58  int main() {
59      map<string, int> caballos;
60      string eq1, eq2, hora;
61      double g1, em, g2;
62      for (int i = 0; i < 20; i++) {
63          cin >> eq1;
64          caballos[eq1] = i;
65      }
66      vector<carrera> carreras(10, carrera(0, 0, "00:00", 1, 1, 1));
67      for (int i = 0; i < 10; i++) {
68          cin >> eq1 >> eq2 >> hora;
69          cin >> g1 >> em >> g2;
70          carreras[i] = carrera(caballos[eq1], caballos[eq2], hora, g1, em, g2);
71      }
72      sort(carreras.begin(), carreras.end());
73      vector<double> DP(10);
74      double mult;
75      for (int i = 0; i < 10; i++) {
76          DP[i] = mult = multiplicador(carreras[i]);
77          string maxHora = menos1h30(carreras[i].hora);
78          for (int j = 0; j < i; j++) {
79              if (carreras[j].hora > maxHora) {
80                  break;
81              }
82              if (mult * DP[j] > DP[i]) {
83                  DP[i] = mult*DP[j];
84              }
85          }
```

```
        if (i && DP[i-1] > DP[i]) {
            DP[i] = DP[i-1];
        }
    }
    cout << fixed << setprecision(1) << DP[9] << '\n';
    return 0;
}
```

Java

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

public class Solution {
    public static HashMap<String, Integer> caballos;
    public static double multiplicador(ArrayList<String> P) {
        int pos1 = caballos.get(P.get(0)), pos2 = caballos.get(P.get(1));
        if (Math.abs(pos1 - pos2) <= 4)
            return Double.parseDouble(P.get(4));
        else if(pos1 > pos2)
            return Double.parseDouble(P.get(5));
        else
            return Double.parseDouble(P.get(3));
    }

    public static String menos1h30(String hora) {
        int hh = Integer.parseInt(hora.substring(0,2)), mm =
        ↪  Integer.parseInt(hora.substring(3, 5));
        if (mm >= 30) {
            mm -= 30;
            hh--;
        } else {
            mm += 30;
            hh -= 2;
        }
        String h = String.valueOf(hh), m = String.valueOf(mm);
        if (h.length() == 1)
            h = "0" + h;
        if (m.length() == 1)
            m = "0" + m;
        return h + ":" + m;
    }

    public static void main(String[] args) {
        Scanner s = new Scanner(System.in);
        caballos = new HashMap<String, Integer>();
        String eq1, eq2, hora, g1, em, g2;
        for (int i = 0; i < 20; i++) {
            eq1 = s.nextLine();
            caballos.put(eq1, i);
        }
        ArrayList<ArrayList<String>> carreras = new
        ↪  ArrayList<ArrayList<String>>();
        for (int i = 0; i < 10; i++) {
```

```
            carreras.add(new ArrayList<String>());
            for (int j = 0; j < 6; j++) {
                carreras.get(i).add(s.next());
            }
        }
        carreras.sort((o1, o2) -> o1.get(2).compareTo(o2.get(2)));
        double[] DP = new double[10];
        double mult;
        for (int i = 0; i < 10; i++) {
            DP[i] = mult = multiplicador(carreras.get(i));
            String maxHora = menos1h30(carreras.get(i).get(2));
            for (int j = 0; j < i; j++) {
                if (carreras.get(j).get(2).compareTo(maxHora) > 0) {
                    break;
                }
                if (mult * DP[j] > DP[i]) {
                    DP[i] = mult*DP[j];
                }
            }
            if (i > 0 && DP[i-1] > DP[i]) {
                DP[i] = DP[i-1];
            }
        }
        System.out.printf("%.1f\n", DP[9]);
        s.close();
    }
}
```

Python

```
def multiplicador(P):
    if (abs(P[0] - P[1]) <= 4):
        return P[4]
    elif(P[0] > P[1]):
        return P[5]
    else:
        return P[3]

def menos1h30(hora):
    hh = int(hora[:2])
    mm = int(hora[3:])
    if (mm >= 30):
        mm -= 30
        hh -= 1
    else:
        mm += 30
        hh -= 2
    h = str(hh)
    m = str(mm)
    if (len(h) == 1):
        h = "0" + h
    if (len(m) == 1):
        m = "0" + m
    return h + ":" + m

```

```
def verHora(carrera):
    return carrera[2]

caballos = dict()
for i in range(20):
    eq1 = input()
    caballos[eq1] = i
carreras = []
for i in range(10):
    eq1, eq2, hora = input().split()
    g1, em, g2 = map(float, input().split())
    carreras.append([caballos[eq1], caballos[eq2], hora, g1, em, g2])
carreras.sort(key = verHora)
DP = []
for i in range(10):
    mult = multiplicador(carreras[i])
    DP.append(mult)
    maxHora = menos1h30(carreras[i][2])
    for j in range(i):
        if (carreras[j][2] > maxHora):
            break
        if (mult * DP[j] > DP[i]):
            DP[i] = mult*DP[j];
    if (i and DP[i-1] > DP[i]):
        DP[i] = DP[i-1]
print("{:.1f}".format(DP[9]))
```