Aufgabe 3: Tobis Turnier

Team-ID: 01144

Team: Legends

Bearbeiter/-innen dieser Aufgabe:
Christoph Waffler

23. November 2020

Lösungsidee

Ziel dieser Aufgabe ist es, Tobi eine Tuniervariante aufgrund der ermittelten Ergebnisse zu geben. Tobi möchte jene Turniervariante spielen, bei der der insgesamt beste Spieler am ehesten gefunden wird.

Dabei gibt es im Grundlegenden 3 verschiedene Möglichkeiten, das Turnier zu gestalten:

Beim Liga-System spielt jeder Spieler genau einmal gegen den anderen Spieler. Gewinner der Liga ist derjenige Spieler mit den besten Siegen bzw. bei Gleichstand mit der geringen Nummer.

Beim einfachen K.O.-System sind die einzelnen Startgruppen/Spielerpaarungen am Anfang des Turniers variabel. Das bedeutet, dass zu Beginn des K.O.-Turniers Spieler 1 gegen Spieler 2 oder gegen Spieler 8 spielen könnte. Der Turnierplan des K.O.-Systems ist abhängig von den Spielerpaarungen zu Beginn des Systems.
Sieger des K.O.-Systems ist der Spieler, der alle Paarungen gewinnt.
Um ein besonders aussagekräftiges Ergebnis zu bekommen, beinhaltet die Lösungsidee drei verschiedene Versionen, wie sich die Startgruppen des K.O.-Systems zusammensetzen:

1. Typ: sortierte Ausgangssituation: Spieler 1 spielt gegen Spieler 2; 3 gegen 4; ...; n-1 gegen n;
2. Typ: sortierte Ausgangssituation: Spieler 1 spielt gegen Spieler n; 2 gegen n-1; ...
3. Typ: randomisierte Ausgangssituation: die Spielerpaarungen zu Beginn der K.O.-Gruppe werden ausgelost: z.B. Spieler 7 gegen Spieler 4

Daneben gibt es die Möglichkeit anstatt eines einfachen K.O.-Systems das K.O.x5-System zu spielen. Der Unterschied zum vorherigen System ist, dass jede Partie 5-mal statt einmal gespielt wird.

Auswertung der Beispiele:

- Im Beispiel 1 überzeugt der 2. Typ des K.O.x5-Systems mit 63% durchschnittlichen Sieg des besten Spielers.
- Im Beispiel 2 sind der 1. und 2. Typ des K.O.x5-Systems die beste Wahl mit 37% durchschnittlichen Sieg für den stärksten Spieler.
- In Beispiel 3 hingegen ist der 3. Typ des K.O.x5-Systems eine gute Wahl mit durchschnittlich 28 % (1. und 2. Typ mit 25%), jedoch liefert die Liga mit 30% durchschnittlichen Sieg für den Stärksten besser Ergebnisse.
- Im Beispiel 4 ist der Typ 1 des K.O.-Systems die beste Entscheidung mit 53% bzw. 54%. Typ 2 des K.O.-Systems liefert ebenfalls einen guten Durchschnitt mit 52% bzw. 50%.
- In Beispiel 4 ist das Liga-System mit 11 % das beste System.

Als Fazit kann gesagt werden, dass der Typ 2 des K.O.x5-Systems zusammen mit dem Liga-System weitestgehend den besten Durchschnitt liefert und somit im Vergleich zu den anderen Systemen die beste Wahl ist.
1. und 3. Typ des K.O.x5-System liefern ebenfalls einen guten Durchschnitt.

Das K.O.-System sollte eher nicht verwendet werden, da dieses einen relativ geringen durchschnittlichen Sieg des stärksten Spielers ermittelt hat.
Das liegt vor allem daran, dass ein Spieler im K.O.x5-System nicht sofort nach einem verlorenen Spiel ausgeschieden ist, was immer mal wieder vorkommen kann und wird.

Umsetzung

Die Lösungsidee wird in Python umgesetzt.
In der Methode *get_winner(i, j, strength_i, strength_j)* wird sozusagen ein Spiel der Spieler *i* und *j* simuliert. Parameter sind ebenfalls die Spielerstärken *strength_i* und *strength_j* beider Spieler.

Als erstes werden bei Spielerstärken zu einem Wert s addiert.
Dann wird mit dem Modul *random()* eine Zufallszahl *r* zwischen 0 und 1 erzeugt.

Anschließend wird geprüft, welcher Spieler der schwächere Spieler ist, d.h. die Spielerstärke ist kleiner als die des anderen Spielers.

Die Nummer des schwächeren Spielers wird in *min_index* gespeichert und die des stärkeren in *max_index*. Ebenfalls wird die Stärke des schwächeren Spielers in *min_value* gespeichert.

Nun wird *min_value* durch *s* geteilt und der Variablen *quot* zugewiesen.

Falls $r < quot$ gilt, so hat der Spieler mit der kleineren Spielstärke gewonnen, andernfalls der Spieler mit der höheren Spielstärke.

Der Gewinner ist der Rückgabewert der Methode.

In der Methode *liga(n, arr)* wird das Liga-System simuliert und ein Gewinner ermittelt:
Zu Beginn wird ein leeres Array *num_siege* mit Größe *n* erzeugt, in dem die Anzahl der Siege der einzelnen Spieler gespeichert werden.
Mithilfe einer for-Schleife wird *i* von *0* bis *n* iteriert. In dieser for-Schleife befindet sich eine weitere for-Schleife mit der *j* von *i+1* bis *n* iteriert wird. In der 2. for-Schleife wird die Methode *get_winner(i, j, arr[i], arr[j])* aufgerufen, welche den Gewinner (entweder *i* oder *j*) zurückliefert. Die Anzahl der Siege des Gewiners wird in *num_siege* um *1* erhöht.

Sind beide for-Schleifen zu Ende, wird der Spieler (= Index) mit dem größten Wert in *num_siege* ermittelt, welcher der Gewinner der Liga ist.
Der Gewinner ist der Rückgabewert der Methode *liga*.

In der Methode *k_o(n, arr, type, num_spiele)* wird je nach Typ des K.O.-Systems eine Wartschlange unterschiedlich gebildet, wodurch der Turnierplan eine unterschiedliche Form annimmt. Eine leere Liste *q* dient als Warteschlange und beinhaltet jeweils 2 Spieler, die gegeneinander spielen.

Für *type==0* wird eine aufsteigende Reihenfolge benötigt, was mit einer for-Schleife von *0* bis *n* umgesetzt wird, welche jeweils den aktuellen Wert an *q* hinten angehängt.

Für *type==1* wird ein andere sortierte Reihenfolge benötigt: Mithilfe einer for-Schleife von *i=0* bis *n/2* wird an *q* der Wert *i* und der Wert *n-(i+1)* hinten angehängt.

Für *type==2* werden die Spieler zufällig miteinander kombiniert: Als erstes werden alle Spieler in der Liste *uebrige* gespeichert. Dann wird für mithilfe eine for-Schleife von *0* bis *n* iteriert, wobei jedes Mal über einen zufällig ausgewählten Index ein Element von *uebrige* ausgewählt und zu *q* hinzugefügt wird.

Nun ist die Warteschlange mit den Anfangspaarungen gefüllt und das Turnier kann beginnen.
Mithilfe einer while-Schleife werden folgende Bedingungen solange ausgeführt, bis *q* nur noch ein Element enthält.
Zwei Spieler *spieler1* und *spieler2* werden mithilfe von *q.pop(0)* vom vorderen Ende von der Warteschlange den beiden Variablen zugewiesen und von *q* entfernt.
Mithilfe der Methode *get_winner(spieler1, spieler2, arr[spieler1], arr[spieler2])* wird so oft ein Spiel gespielt, wie es der Wert des Parameters *num_spiele* angibt.
Die Anzahl der Gewinne wird ebenfalls gespeichert.
Nachdem beide Spieler *num_spiele*-mal gegeneinander gespielt haben wird der Gewinner von beiden wieder an das Ende der Warteschlange hinzugefügt.

Ist die while-Schleife fertig, so enthält die Warteschlange nur noch einen Spieler und zwar den Gewinner des Turniers.
Dieser ist Rückgabewert der Methode.

In der Main-Methode werden alle möglichen Varianten insgesamt 1000-mal gespielt und die Anzahl der Siege des stärksten Spielers gezählt. Danach wird die durchschnittliche Häufigkeit seines Siegs berechnet und ausgegeben.

Beispiele

spielstaerken1.txt

0
10
20
30
40
50
60
100
bester Spieler: Nr. 8 mit 100

Alle Variationen werden 1000 Mal durchgeführt
Ausgabe, wie oft der spielstärkste Spieler im Durchschnitt gewonnen hat (mit Angabe in Prozent)

Spiel mit Liga-System:
>> 324 Siege bei 1000 Spielen --> 32.4 %

Verschiedene Varianten des einfachen K.O.-Systems:

> sortiertes Ausgangs-Matching 1 vs 2, 3 vs 4, ...
>> 371 Siege bei 1000 Spielen --> 37.1 %

> sortiertes Ausgangs-Matching 1 vs n, 2 vs n-1
>> 454 Siege bei 1000 Spielen --> 45.4 %

> randomisiertes Ausgangsmatching
>> 383 Siege bei 1000 Spielen --> 38.3 %

Verschiedene Varianten des K.0.x5-Systems:

> sortiertes Ausgangs-Matching 1 vs 2, 3 vs 4, ...
>> 545 Siege bei 1000 Spielen --> 54.5 %

> sortiertes Ausgangs-Matching 1 vs n, 2 vs n-1, ...
>> 625 Siege bei 1000 Spielen --> 62.5 %

> randomisiertes Ausgangs-Matching
>> 585 Siege bei 1000 Spielen --> 58.5 %

spielstaerken2.txt

8
10
10
10
10
80
80
80
100
bester Spieler: Nr. 8 mit 100

Alle Variationen werden 1000 Mal durchgeführt
Ausgabe, wie oft der spielstärkste Spieler im Durchschnitt gewonnen hat (mit Angabe in Prozent)

Spiel mit Liga-System:
>> 200 Siege bei 1000 Spielen --> 20.0 %

Verschiedene Varianten des einfachen K.O.-Systems:

> sortiertes Ausgangs-Matching 1 vs 2, 3 vs 4, ...
>> 294 Siege bei 1000 Spielen --> 29.4 %

> sortiertes Ausgangs-Matching 1 vs n, 2 vs n-1
>> 286 Siege bei 1000 Spielen --> 28.6 %

> randomisiertes Ausgangsmatching
>> 293 Siege bei 1000 Spielen --> 29.3 %

Verschiedene Varianten des K.0.x5-Systems:

> sortiertes Ausgangs-Matching 1 vs 2, 3 vs 4, ...
>> 368 Siege bei 1000 Spielen --> 36.8 %

> sortiertes Ausgangs-Matching 1 vs n, 2 vs n-1, ...
>> 365 Siege bei 1000 Spielen --> 36.5 %

> randomisiertes Ausgangs-Matching
>> 325 Siege bei 1000 Spielen --> 32.5 %

spielstaerken3.txt

16
22
38
66
93
51
51
58
67
51
57
57
60
73
13
41
42
bester Spieler: Nr. 4 mit 93

Alle Variationen werden 1000 Mal durchgeführt
Ausgabe, wie oft der spielstärkste Spieler im Durchschnitt gewonnen hat (mit Angabe in Prozent)

Spiel mit Liga-System:
>> 309 Siege bei 1000 Spielen --> 30.9 %

Verschiedene Varianten des einfachen K.O.-Systems:

> sortiertes Ausgangs-Matching 1 vs 2, 3 vs 4, ...
>> 169 Siege bei 1000 Spielen --> 16.9 %

> sortiertes Ausgangs-Matching 1 vs n, 2 vs n-1
>> 150 Siege bei 1000 Spielen --> 15.0 %

> randomisiertes Ausgangsmatching
>> 159 Siege bei 1000 Spielen --> 15.9 %

Verschiedene Varianten des K.0.x5-Systems:

> sortiertes Ausgangs-Matching 1 vs 2, 3 vs 4, ...
>> 250 Siege bei 1000 Spielen --> 25.0 %

> sortiertes Ausgangs-Matching 1 vs n, 2 vs n-1, ...
>> 249 Siege bei 1000 Spielen --> 24.9 %

> randomisiertes Ausgangs-Matching
>> 283 Siege bei 1000 Spielen --> 28.3 %

spielstaerken4.txt

16
100
95
95
95
95
95
95
95
95
95
95
95
95
95
95
95
bester Spieler: Nr. 1 mit 100

Alle Variationen werden 1000 Mal durchgeführt
Ausgabe, wie oft der spielstärkste Spieler im Durchschnitt gewonnen hat (mit Angabe in Prozent)

Spiel mit Liga-System:
>> 109 Siege bei 1000 Spielen --> 10.9 %

Verschiedene Varianten des einfachen K.O.-Systems:

> sortiertes Ausgangs-Matching 1 vs 2, 3 vs 4, ...
>> 50 Siege bei 1000 Spielen --> 5.0 %

> sortiertes Ausgangs-Matching 1 vs n, 2 vs n-1
>> 82 Siege bei 1000 Spielen --> 8.2 %

> randomisiertes Ausgangsmatching
>> 71 Siege bei 1000 Spielen --> 7.1 %

Verschiedene Varianten des K.0.x5-Systems:

> sortiertes Ausgangs-Matching 1 vs 2, 3 vs 4, ...
>> 67 Siege bei 1000 Spielen --> 6.7 %

> sortiertes Ausgangs-Matching 1 vs n, 2 vs n-1, ...
>> 89 Siege bei 1000 Spielen --> 8.9 %

> randomisiertes Ausgangs-Matching
>> 86 Siege bei 1000 Spielen --> 8.6 %

Quellcode

(Es wurde der gesamte Quellcode miteingefügt.)

```
from random import random, randrange


def get_winner(i: int, j: int, strength_i: int, strength_j: int):
    """ Spieler mit den Nummer i und j spielen gegeneinander
    Parameter sind ebenfalls die Spielerstärken beider Spieler
    Rückgabewert ist der Gewinner der beiden Spieler (entweder i oder j)
    """
    # Summe der Gewichtungen der beiden Spieler
    s = strength_i + strength_j

    # Zufallszahl zwischen 0 und 1 wird erzeugt
    r = random()

    # der kleinere Wert von beiden wird ermittelt
    if strength_i < strength_j:
        min_index = i
        max_index = j
        min_value = strength_i
    else:
        min_index = j
        max_index = i
        min_value = strength_j

    quot = min_value/s

    if r < quot:
        # Spieler mit der kleineren Spielstärke gewinnt
        winner = min_index
```

```
    else:
        # anderer Spieler ist Gewinner (Spieler mit der größeren Spielstärke)
        winner = max_index

    return winner

def liga(n: int, arr: list):
    # Anzahl der Siege der einzelnen Spieler
    num_siege = []
    for i in range(n):
        num_siege.append(0)

    # Jeder Spieler spielt gegen jeden anderne genau ein Mal
    for i in range(n):
        for j in range(i+1, n):
            w = get_winner(i, j, arr[i], arr[j])
            # Anzahl der Siege des Gewinners wird erhöht
            num_siege[w] += 1

    max_element = max(num_siege)
    winner = num_siege.index(max_element)

    return winner

def k_o(n: int, arr: list, type: int, num_spiele: int):
    """ Je nach Typ wird die Wartschlange unterschiedlich gebildet
        und der Turnierplan nimmt eine unterschiedliche Form an
    """
    # Wartschlange q
    q = []
```

```
    # Typ 1: Sortierte Reihenfolge: 1 gegen 2, 3 gegen 4, etc.
    if type == 0:
        for i in range(n):
            q.append(i)

    # Typ 2: Spieler 1 gegen Spieler n, 2 gegen n-1, etc.
    elif type == 1:
        for i in range(int(n/2)):
            q.append(i)
            q.append((n-(i+1)))

    # Typ 3: Spieler werden zufällig zusammen kombiniert
    # der Index wird zufällig ausgewählt
    elif type == 2:
        uebrige = []
        for i in range(n):
            uebrige.append(i)

        for i in range(n):
            index = randrange(0, len(uebrige))
            q.append(uebrige[index])

            uebrige.remove(uebrige[index])



    # Solange die Warteschlange noch nicht leer ist
    while len(q) > 1:
        spieler1 = q.pop(0)
        spieler2 = q.pop(0)

        win_counter = []
```

```
        for _ in range(n):
            win_counter.append(0)
        # Es wird so oft gespielt, wie es der Methode übergeben wird
        # also kann das K.O.- und das K.O.x5-System darstellen
        for _ in range(num_spiele):
            # Anzahl der gewonnenen Spiele des Gewinners wird um 1 erhöht
            w = get_winner(spieler1, spieler2, arr[spieler1], arr[spieler2])
            win_counter[w] += 1

        # Spieler, der öfters gewonnen hat, wird zur Warteschlange wieder hinzugefügt
        max_value = max(win_counter)
        winner = win_counter.index(max_value)

        q.append(winner)

    # Gewinner ist der letzte verbleibende Spieler in der Warteschlange
    winner_player = q.pop()

    return winner_player

def ausgabe(games: int, wins: int):
    """ kleine Helper-Methode zum Vereinfachen der Ausgabe """
    perc = (wins/games)*100
    rounded = round(perc, 4)
    print(f">> {wins} Siege bei {games} Spielen --> {rounded} % ")

def main():
    n = int(input())
    arr = []
    for i in range(n):
        s = int(input())
```

```
        arr.append(s)

    best_value = 0
    best_player = -1
    for i in range(n):
        if arr[i] > best_value:
            best_value = arr[i]
            best_player = i
    print(f"bester Spieler: Nr. {best_player+1} mit {arr[best_player]}")

    games = 1000
    print(f"\nAlle Variationen werden {games} Mal durchgeführt")
    print("Ausgabe, wie oft der spielstärkste Spieler im Durchschnitt gewonnen hat (mit Angabe
in Prozent)\n")

    # Liga wird gespielt
    wins = 0
    print("Spiel mit Liga-System:")
    for _ in range(games):
        winner_liga = liga(n, arr)
        if best_player == winner_liga:
            wins += 1
    ausgabe(games, wins)



    # K.O. (1-fach) wird gespielt
    print("\n Verschiedene Varianten des einfachen K.O.-Systems:")
    print("\n> sortiertes Ausgangs-Matching 1 vs 2, 3 vs 4, ...")
    wins = 0
    for _ in range(games):
        winner = k_o(n, arr, 0, 1)
```

```
    if best_player == winner:
        wins += 1
ausgabe(games, wins)

print("\n> sortiertes Ausgangs-Matching 1 vs n, 2 vs n-1 ")
wins = 0
for _ in range(games):
    winner = k_o(n, arr, 1, 1)
    if best_player == winner:
        wins += 1
ausgabe(games, wins)

print("\n> randomisiertes Ausgangsmatching")
wins = 0
for _ in range(games):
    winner = k_o(n, arr, 2, 1)
    if best_player == winner:
        wins += 1
ausgabe(games, wins)


# K.O.x5
print("\n\nVerschiedene Varianten des K.0.x5-Systems:")
print("\n> sortiertes Ausgangs-Matching 1 vs 2, 3 vs 4, ... ")
wins = 0
for _ in range(games):
    winner = k_o(n, arr, 0, 5)
    if best_player == winner:
        wins += 1
ausgabe(games, wins)
```

```
    print("\n> sortiertes Ausgangs-Matching 1 vs n, 2 vs n-1, ...")
    wins = 0
    for _ in range(games):
        winner = k_o(n, arr, 1, 5)
        if best_player == winner:
            wins += 1
    ausgabe(games, wins)

    print("\n> randomisiertes Ausgangs-Matching")
    wins = 0
    for _ in range(games):
        winner = k_o(n, arr, 2, 5)
        if best_player == winner:
            wins += 1
    ausgabe(games, wins)



if __name__ == "__main__":
    main()
```