Arwed J. Kirmse

BWINF Treffsicherheit

Aufgabe: Hilf Ada und schreibe ein Programm, das eine Präferenztabelle einliest und anschließend berechnet und ausgibt, wie viele Einträge wenigstens verändert werden müssten, damit ein allseits beliebter Termin entsteht. Lass auch diesen Termin ausgeben.

Lösungsidee: Es wird geschaut, wie viele Veränderungen pro Tag sein müssen um damit er zu einem „allseits beliebten Termin“ wird. Die Tage mit den niedrigsten Werten werden dann ausgegeben (möglicherweise mehr als 1).

Umsetzung: Man fängt an indem man die Zeilen in eine Member class einließt (dabei wird der beliebteste Tag mit herausgefunden). Danach fahren wir fort mit einer loop die durch alle Termine durch iteriert. In dieser wird ein Temporäre Tag class erstellt die später (wenn sie gut ist) in einen vektor gespeichert wird.

Danach iterieren wir durch alle Member und geben den tagen ein ranking. Wenn der beliebteste Tag größer oder gleich wie der Termin am Tag ist dann wird dem Tag ein Beliebtheitspunkt gegeben. Andernfalls wird berechnet wie viele Termine man verändern braucht bis man einen „beliebten Termin“ hat.

Diese Rechnung funktioniert so: Man nimmt den Termin Status vom gerade benutzten Tag. Danach looped man durch alle Termine dieses Members, außer den gerade benutzten Termin. Wenn der Termin Status von dem Tag größer ist als der gerade geloopete Termin (das heißt dieser Termin muss geändert werden) dann wird ein Zähler erhöht. Diese Zahl wird dann zu einer Variable in der TempTag class addiert.

Nach dem alle Member gelooped wurden, werden die veränderten Einträge des Tages überprüft. Wenn dieser genauso groß wie die (bis jetzt) beste Veränderung ist dann wird er einem Vektor hinzugefügt. Sollte er kleiner sein (=besser) dann wird der Vektor gereinigt (clear()), dieser Tag hinzugefügt und die beste Veränderung auf den Wert dieses Tages gesetzt.

Am ende werden alle Inhalte des Vektors ausgegeben (falls es mehrere Lösungen gibt).

Beispiel: In dem Falle benutzen wir das Programm mit 2 Beispiel datein.

```
Argv Path: C:\Users\Viper\source\repos\Bwinf40\Aufgaben\praeferenzen0.txt
File is Valid

Moeglicher Tag: Tag 6 mit 2 veraenderten eintraegen und einer beliebtheit von 4.
```

```
Argv Path: C:\Users\Viper\source\repos\Bwinf40\Aufgaben\praeferenzen3.txt
File is Valid

Moeglicher Tag: Tag 11 mit 41 veraenderten eintraegen und einer beliebtheit von 4.
Moeglicher Tag: Tag 16 mit 41 veraenderten eintraegen und einer beliebtheit von 4.
```

Die 4. Beispieldatei hat 2 mögliche Tage.

Quelltext:

```
20    for (int i = 0; i < Mitglieder; i++) //iterate throught Mitglieder
21    {
22        fmanager.ExtractInfo(content, false); //Add to content
23        Member temp{};
24        temp.Tage = content[i];
25        temp.BeliebtesterTag();
26        member.push_back(temp);
27    }
28
29    int bestveränderung{-1};
30
31    for (int i = 1; i < Tage + 1; i++) //loop Tage
32    {
33        //create Temptag
34        Tag tempTag;
35        tempTag.index = i;
36
37        for (Member mem : member) //lopp members
38        {
39            if (mem.beliebtestes >= mem.Tage[i - 1]) tempTag.beliebtheit++; //if beliebteste
40            else
41            {
42                tempTag.veränderteEinträge += mem.VeränderteTage(i-1); //veränderungs calculation
43            }
44        }
45
46        //ranking
47        if (tempTag.veränderteEinträge == bestveränderung) //gleich
48        {
49            besttage.push_back(tempTag);
50        }
51        else if (tempTag.veränderteEinträge < bestveränderung || bestveränderung == -1) //besser / niedriger
52        {
53            besttage.clear();
54            besttage.push_back(tempTag);
55            bestveränderung = tempTag.veränderteEinträge;
56        }
57    }
```

Textform:

```
for (int i = 0; i < Mitglieder; i++) //iterate throught Mitglieder
    {
        fmanager.ExtractInfo(content, false); //Add to content
        Member temp{};
        temp.Tage = content[i];
        temp.BeliebtesterTag();
        member.push_back(temp);
    }

    int bestveränderung{-1};

    for (int i = 1; i < Tage + 1; i++) //loop Tage
    {
        //create Temptag
        Tag tempTag;
        tempTag.index = i;

        for (Member mem : member) //lopp members
        {
            if (mem.beliebtestes >= mem.Tage[i - 1]) tempTag.beliebtheit++; //if beliebteste
            else
            {
                tempTag.veränderteEinträge += mem.VeränderteTage(i-1); //veränderungs
calculation
            }
        }

        //ranking
        if (tempTag.veränderteEinträge == bestveränderung) //gleich
        {
            besttage.push_back(tempTag);
        }
        else if (tempTag.veränderteEinträge < bestveränderung || bestveränderung == -1)
//besser / niedriger
        {
            besttage.clear();
            besttage.push_back(tempTag);
            bestveränderung = tempTag.veränderteEinträge;
        }
    }
```

```
3     struct Member
4     {
5         int beliebtestes{2};
6         std::vector<int> Tage{};
7
8         void BeliebtesterTag()
9         {
10            for (int Tag : Tage)
11            {
12                if (Tag < beliebtestes) beliebtestes = Tag;
13                if (beliebtestes == 0) break; //beliebtestes ist max
14            }
15        }
16
17        int VeränderteTage(int index) //Wie viele Tage verändert werden müssen um es beliebt zu machen
18        {
19            int output{0};
20            int TagBeliebtheit = this->Tage[index];
21
22            for (int i = 0; i < this->Tage.size(); i++)
23            {
24                if (i == index) continue;
25                else if (TagBeliebtheit > this->Tage[i]) output++;
26
27            }
28
29            return output;
30        }
31    };
32
33    struct Tag
34    {
35        int index{};
36        int beliebtheit{};
37        int veränderteEinträge{0};
38    };
```

Textform:

```
struct Member
{
	int beliebtestes{2};
	std::vector<int> Tage{};

	void BeliebtesterTag()
	{
		for (int Tag : Tage)
		{
			if (Tag < beliebtestes) beliebtestes = Tag;
			if (beliebtestes == 0) break; //beliebtestes ist max
		}
	}

	int VeränderteTage(int index) //Wie viele Tage verändert werden müssen um es beliebt
zu machen
	{
		int output{0};
		int TagBeliebtheit = this->Tage[index];

		for (int i = 0; i < this->Tage.size(); i++)
		{
			if (i == index) continue;
			else if (TagBeliebtheit > this->Tage[i]) output++;

		}

		return output;
	}
};

struct Tag
{
	int index{};
	int beliebtheit{};
	int veränderteEinträge{0};
};
```

Anmerkungen: Ich hatte noch ein System bedacht welches Tage nach der Beliebtheit ausschließt. Es wäre schneller gewesen, jedoch habe ich mich dagegen entschieden da es (in selten fällen) zu nur 1/3 Lösungen oder zu einer nicht korrekten Lösung führen kann (nur theoretisch und selten).

Außerdem werden alle Dateien durch meinen fileManager geregelt und man kann die Datei sowohl mit args öffnen als auch einen pfad eingeben.