Arwed J. Kirmse

BWINF Zum Winde Verweht

Aufgabe: Schreibe ein Programm, das die Auswahl der Hotels übernimmt Dein Programm soll die Angaben zu den Hotels, nämlich ihre Positionen entlang der Route und ihre Bewertungen, einlesen und eine Auswahl nach den Wünschen von Lara und Paul ausgeben.

Lösungsidee: Zuerst wird für den gerade geloopten Tag einen Maximal Reichweite und eine Minimal Reichweite festgelegt die er reisen muss. Danach wird damit und dem Derzeitigen Standpunkt eine Maximale und Minimale Range mit dem Startpunkt als Basis errechnet. Somit kann dann zwischen diesen beiden Rangen die Hotels angeschaut werden und das mit der besten Bewertung ausgewählt werden. Sollte der Fall auftreten, dass es kein Hotel in dieser Range gibt, dann wird ein Schritt zurück gegangen und ein anderes Hotel probiert.

Umsetzung: Zuerst werden die Hotel Daten in eine Hotel Klassen eingelesen. Wobei die Bewertung als double gespeichert wird und der Rest als int.

Danach erstellen wir eine Liste die Tage speichert. Dieser geben wir dann einen Dummy Tag der als ausgangs Punkt geht. Ein Tag wird wie folgt initialisiert: Noch zu fahren, Tag, Standort, Startpunkt, Hotel Pointer Liste.

Danach wird eine For-Loop gestartet, mit der Limitierung von 4, weil die Familie nur maximal 5 Tage fahren soll. 4 ist die Limitierung da 0 als Tag gilt. Außerdem gibt es noch ein Statement, das wenn die Reiseminuten unter 360 sind (also wie lange sie pro Tag fahren können), dann wird abgebrochen, da sie dann an ihrem Reiseziel angekommen sind. Danach wird ein Tag initialisiert mit den Werten des vorherigen Tages.
Noch zu Fahren → Vorherige Tag Noch zu Fahren
Tag → i + 1
Standort → Vorherige Tag Hotel Standorten
Startpunkt → Gesamtfahrzeit
Hotel Liste → Hotel Liste

Die Berechnung in der Tag Initialisierung sieht wie folgt aus:
1) Max Fahren → 360 , weil die Familie nur 6 Stunden fahren darf (= 360 Autominuten)
2) Min Fahren → Noch Zu Fahren – (5 – Tag) * 360 , hier wird die Noch zu Fahrende Distanz minus die maximal Anzahl von Tagen (5) Minus der Tag gerade * 360 (die Auto Minuten) gerechnet. Hier wird die Distanz berechnet die dieses Auto fahren kann wenn alle anderen Autos die maximale Anzahl von stunden am Tag möglich fahren.
3) Max Range → Standort + Max Fahren , der Derzeitige Standort addiert mit der maximalen Fahr Distanz ergibt die höchst Grenze in der sich das Hotel für diesen Tag befinden darf
4) Min Range → Standort + Min Fahren , der Derzeitige Standort addiert mit der minimalen Fahr Distanz ergibt die niedrigste Grenze in der sich das Hotel für den Tag befinden darf
5) hotel → finden(HotelListe) , durch die Methode [finden()] wird später erklärt
6) NochZuFahren → Starpunkt – Hotel Entfernung , wenn der Starpunkt mit der Hotel Entfernung subtrahiert wird, dann kommt man auf die noch zu Fahrende strecke.

Die Methode finden() mit der wir das beste Hotel in der nähe finden, funktioniert so: Zuerst wird eine Variable mit Namens best Hotel mit einem Placeholder erstellt mit der die Nummer -1 hat (wichtig für später). Danach loopen wir durch alle Hotels die es gibt und picken nur die raus die in zwischen der Minimalen und Maximalen Range liegen. Diese werden dann in eine Liste gespeichert wo alle Hotels in Range drin sind. Sollte dieses Hotel eine besserer Bewertung haben als das derzeitige best Hotel, dann wird es als neues Bestes Hotel abgespeichert und die Liste Hotels in Range gecleared. Am ende wird das Beste Hotel dann zurückgegeben.

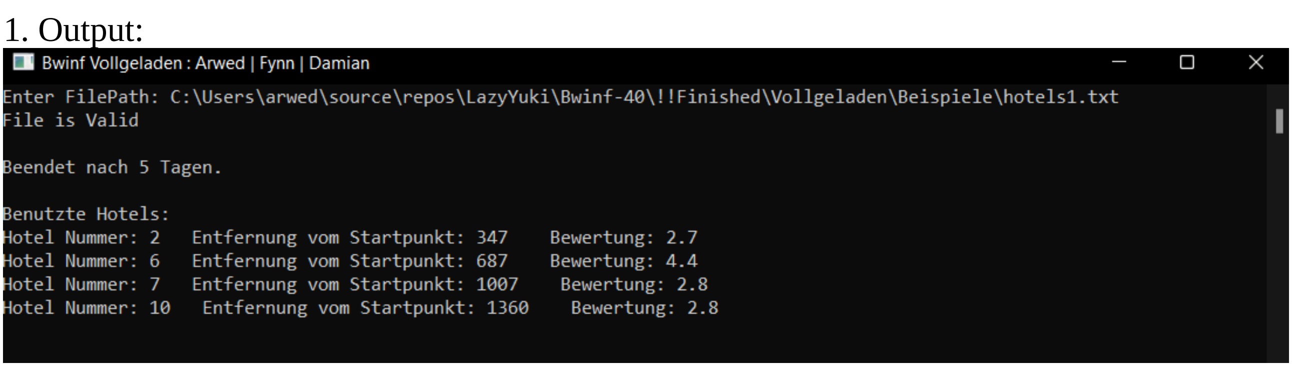
Weitergeht es in der Tages Loop. Dort wird (nachdem der Tag iniziert wurde) überprüft, ob der Nummer nicht gleich -1 ist. Das sagt aus das ein best Hotel gefunden wurde. Und dann wird dieser Tag in die Tage Liste gegeben.

Sollte das nicht so sein wird von dem Tag davor das erste Element aus der Liste Hotels in Range gelöscht (welches wegen der Finden Methode somit auch das beste ist) und eine neue Finden Methode gestartet. Dies arbeitet nur mit den Hotels in Range Liste. Das ist so weil wenn ein Hotel nicht erreichbar ist muss man vom Tag davor ein anderes Hotel nehmen, damit man eine größere Reichweite hat. Grundsätzlich ist die 2. Finden Methode nicht anders als die erste. Das einzige was anders ist, ist das wenn ein neues Best Hotel gefunden wurde nur alles was davor war in der Liste erased wird und nicht die ganze Liste. Nachdem finden wird der Iterator i – 1 gerechnet, damit sich der Tag sozusagen wiederholt.

Diese Tages loope findet so lange statt bis alle Tag gefunden hat.

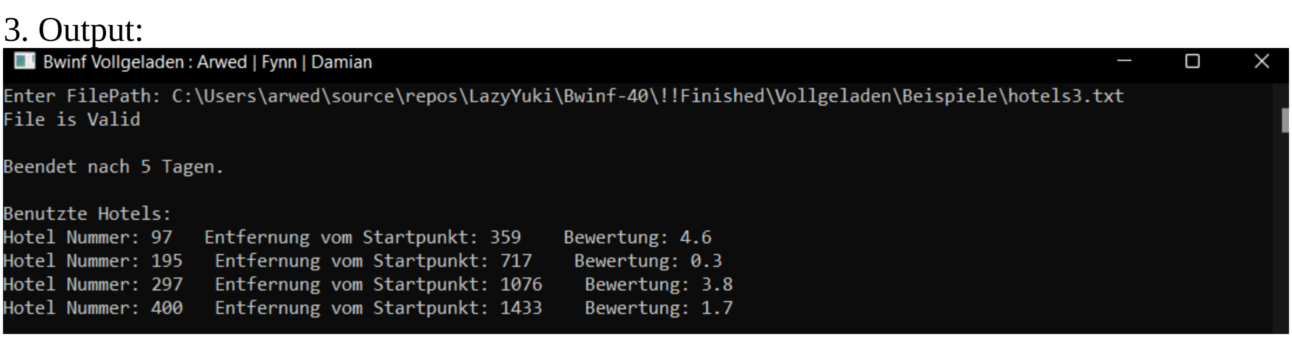
Beispiele:

1. Output:



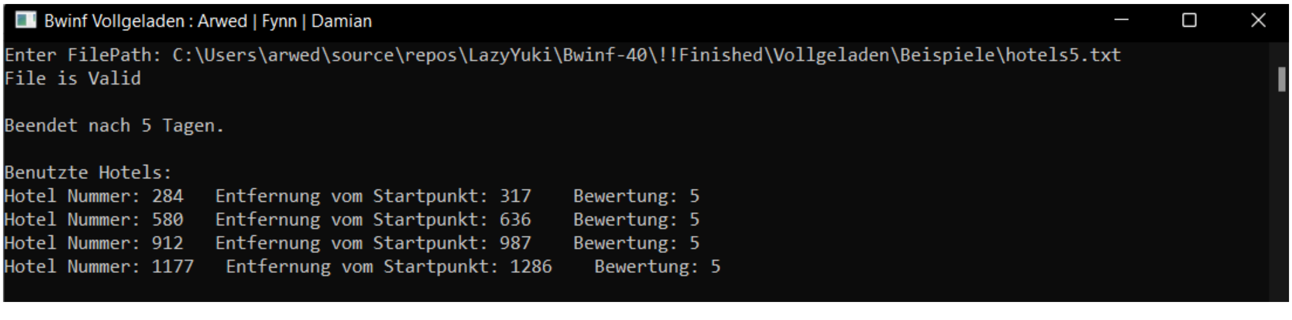
```
Enter FilePath: C:\Users\arwed\source\repos\LazyYuki\Bwinf-40\!!Finished\Vollgeladen\Beispiele\hotels1.txt
File is Valid

Beendet nach 5 Tagen.

Benutzte Hotels:
Hotel Nummer: 2   Entfernung vom Startpunkt: 347    Bewertung: 2.7
Hotel Nummer: 6   Entfernung vom Startpunkt: 687    Bewertung: 4.4
Hotel Nummer: 7   Entfernung vom Startpunkt: 1007    Bewertung: 2.8
Hotel Nummer: 10   Entfernung vom Startpunkt: 1360    Bewertung: 2.8
```

3. Output:



```
Enter FilePath: C:\Users\arwed\source\repos\LazyYuki\Bwinf-40\!!Finished\Vollgeladen\Beispiele\hotels3.txt
File is Valid

Beendet nach 5 Tagen.

Benutzte Hotels:
Hotel Nummer: 97   Entfernung vom Startpunkt: 359    Bewertung: 4.6
Hotel Nummer: 195   Entfernung vom Startpunkt: 717    Bewertung: 0.3
Hotel Nummer: 297   Entfernung vom Startpunkt: 1076    Bewertung: 3.8
Hotel Nummer: 400   Entfernung vom Startpunkt: 1433    Bewertung: 1.7
```

5. Output:



```
Enter FilePath: C:\Users\arwed\source\repos\LazyYuki\Bwinf-40\!!Finished\Vollgeladen\Beispiele\hotels5.txt
File is Valid

Beendet nach 5 Tagen.

Benutzte Hotels:
Hotel Nummer: 284   Entfernung vom Startpunkt: 317    Bewertung: 5
Hotel Nummer: 580   Entfernung vom Startpunkt: 636    Bewertung: 5
Hotel Nummer: 912   Entfernung vom Startpunkt: 987    Bewertung: 5
Hotel Nummer: 1177   Entfernung vom Startpunkt: 1286    Bewertung: 5
```

Die Restlichen Outputs sind in Textform im Ordner Beispiele.

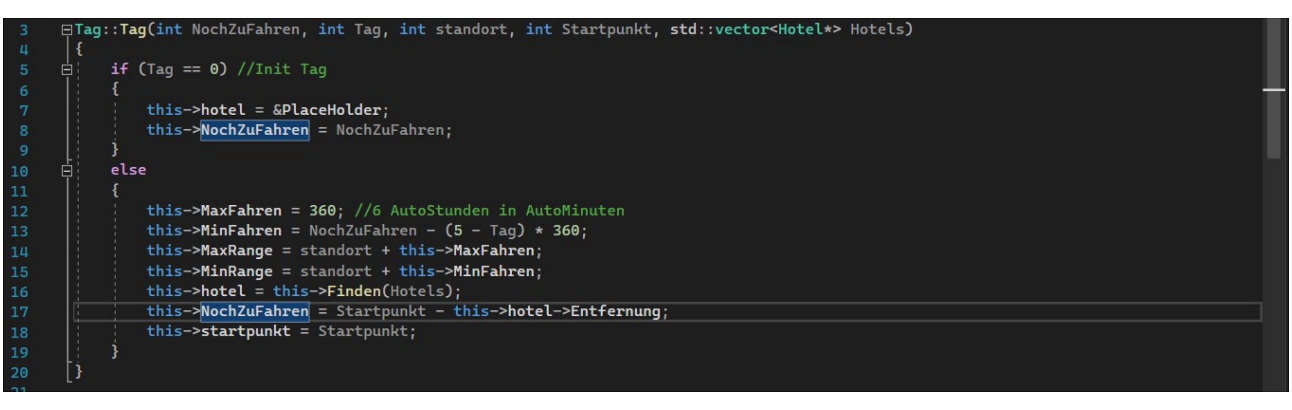
Quelltext:

Tage Loop:

```
    std::vector<Tag> Tage{Tag(Gesamtfahrzeit, 0, 0, Gesamtfahrzeit, HotelPointerList)};

    for (i = 0; i < 4; i++)
    {
        if (Tage[i].NochZuFahren <= 360) break;

        Tag tag = Tag(Tage[i].NochZuFahren, i + 1, Tage[i].hotel->Entfernung, Gesamtfahrzeit, HotelPointerList);

        if(MitSchritten) tag.PrintTag();

        if (tag.hotel->Nummer != -1) Tage.push_back(tag);
        else
        {
            Tage[i].HotelsInRange.erase(Tage[i].HotelsInRange.begin()); //erase best hotel (is always first hotel in list)
            Tage[i].Finden();
            if (MitSchritten) Tage[i].PrintTag();
            i--;
        }
    }
```

```
std::vector<Tag> Tage{Tag(Gesamtfahrzeit, 0, 0, Gesamtfahrzeit, HotelPointerList)};

    for (i = 0; i < 4; i++)
    {
        if (Tage[i].NochZuFahren <= 360) break;

        Tag tag = Tag(Tage[i].NochZuFahren, i + 1, Tage[i].hotel->Entfernung, 
Gesamtfahrzeit, HotelPointerList);

        if(MitSchritten) tag.PrintTag();

        if (tag.hotel->Nummer != -1) Tage.push_back(tag);
        else
        {
            Tage[i].HotelsInRange.erase(Tage[i].HotelsInRange.begin()); //erase best 
hotel (is always first hotel in list)
            Tage[i].Finden();
            if (MitSchritten) Tage[i].PrintTag();
            i--;
        }
    }
```

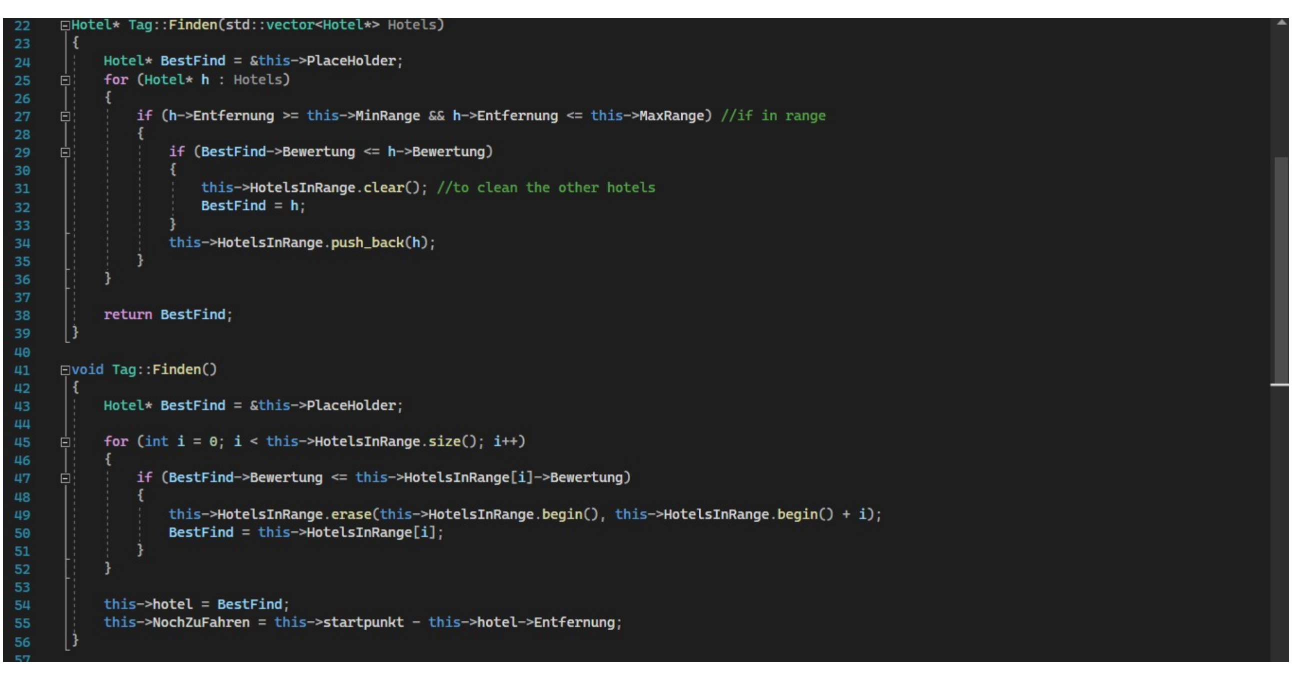
Tag initialisierung:



```
Tag::Tag(int NochZuFahren, int Tag, int standort, int Startpunkt, std::vector<Hotel*> Hotels)
{
    if (Tag == 0) //Init Tag
    {
        this->hotel = &PlaceHolder;
        this->NochZuFahren = NochZuFahren;
    }
    else
    {
        this->MaxFahren = 360; //6 AutoStunden in AutoMinuten
        this->MinFahren = NochZuFahren - (5 - Tag) * 360;
        this->MaxRange = standort + this->MaxFahren;
        this->MinRange = standort + this->MinFahren;
        this->hotel = this->Finden(Hotels);
        this->NochZuFahren = Startpunkt - this->hotel->Entfernung;
        this->startpunkt = Startpunkt;
    }
}
```

```
Tag::Tag(int NochZuFahren, int Tag, int standort, int Startpunkt, std::vector<Hotel*> 
Hotels)
{
	if (Tag == 0) //Init Tag
	{
		this->hotel = &PlaceHolder;
		this->NochZuFahren = NochZuFahren;
	}
	else
	{
		this->MaxFahren = 360; //6 AutoStunden in AutoMinuten
		this->MinFahren = NochZuFahren - (5 - Tag) * 360;
		this->MaxRange = standort + this->MaxFahren;
		this->MinRange = standort + this->MinFahren;
		this->hotel = this->Finden(Hotels);
		this->NochZuFahren = Startpunkt - this->hotel->Entfernung;
		this->startpunkt = Startpunkt;
	}
}
```

Finden Methoden:



```
Hotel* Tag::Finden(std::vector<Hotel*> Hotels)
{
    Hotel* BestFind = &this->PlaceHolder;
    for (Hotel* h : Hotels)
    {
        if (h->Entfernung >= this->MinRange && h->Entfernung <= this->MaxRange) //if in range
        {
            if (BestFind->Bewertung <= h->Bewertung)
            {
                this->HotelsInRange.clear(); //to clean the other hotels
                BestFind = h;
            }
            this->HotelsInRange.push_back(h);
        }
    }

    return BestFind;
}

void Tag::Finden()
{
    Hotel* BestFind = &this->PlaceHolder;

    for (int i = 0; i < this->HotelsInRange.size(); i++)
    {
        if (BestFind->Bewertung <= this->HotelsInRange[i]->Bewertung)
        {
            this->HotelsInRange.erase(this->HotelsInRange.begin(), this->HotelsInRange.begin() + i);
            BestFind = this->HotelsInRange[i];
        }
    }

    this->hotel = BestFind;
    this->NochZuFahren = this->startpunkt - this->hotel->Entfernung;
}
```

```
Hotel* Tag::Finden(std::vector<Hotel*> Hotels)
{
	Hotel* BestFind = &this->PlaceHolder;
	for (Hotel* h : Hotels)
	{
		if (h->Entfernung >= this->MinRange && h->Entfernung <= this->MaxRange) 
//if in range
		{
			if (BestFind->Bewertung <= h->Bewertung)
			{
				this->HotelsInRange.clear(); //to clean the other hotels
				BestFind = h;
			}
			this->HotelsInRange.push_back(h);
		}
	}

	return BestFind;
}

void Tag::Finden()
{
	Hotel* BestFind = &this->PlaceHolder;

	for (int i = 0; i < this->HotelsInRange.size(); i++)
	{
		if (BestFind->Bewertung <= this->HotelsInRange[i]->Bewertung)
		{
			this->HotelsInRange.erase(this->HotelsInRange.begin(), this-
>HotelsInRange.begin() + i);
			BestFind = this->HotelsInRange[i];
		}
	}

	this->hotel = BestFind;
	this->NochZuFahren = this->startpunkt - this->hotel->Entfernung;
}
```

Anmerkungen: Alle Dateien durch meinen fileManager geregelt und man kann die Datei sowohl mit args öffnen als auch einen pfad eingeben.