Aufgabe 2: Schwimmbad

Team: “throw new Exception();”  
Einsendenummer: 00025

26. November 2017

Inhaltsverzeichnis

[Lösungsidee 1](#_Toc492762679)

[Umsetzung 2](#_Toc492762680)

[Beispiele 5](#_Toc492762681)

[Quellcode 10](#_Toc492762682)

# Lösungsidee

Das Problem wirkte zunächst, als wäre eine Bruteforce Methode angemessen, aber diese ist natürlich mit Laufzeitproblemen behaftet.

Aus diesem Grund habe ich versucht, einen mehr oder weniger laufzeiteffizienten Algorithmus zu finden, indem ich einige Abkürzungen einbaue, während ich dennoch mit Backtracking arbeite.

Generell wird immer unterschieden, ob es sich um Woche oder Wochenende handelt, da diese Datumsmerkmale einen großen Einfluss auf die Preise haben. Ferien bzw. Schulzeit ist nur bei Verrechnung der Gutscheine relevant und wird somit auf einer entfernteren Ebene beachtet. Nämlich genau da, wo außerdem verglichen wird, welche Preisvorteile bestimmte Datumsmerkmal haben.

Für Woche und Wochenende gibt es jedoch einzelne Funktionen, die sehr ähnlich funktionieren. Die Hauptfunktionen zum Berechnen der gesamten und „großen“ Altersverteilung / Kombination unterscheiden sich lediglich darin, welche andere Funktion sie zum Berechnen elementarer Fälle aufrufen und wo sie ihre Ergebnisse speichern.

Damit habe ich auch die wesentlichen Unterschiede zum Bruteforce genannt:

Ich habe alle Kombinationen mit maximal sechs zahlungspflichtigen Besuchern im Kopf berechnet und lasse diese Werte in einer Methode zurückgeben. Außerdem wird im Laufe der Berechnungen eine Liste mit bereits ermittelten Kombinationen gefüllt, sodass diese nicht mehrmals durchlaufen werden müssen.

Das funktioniert bei beiden Datumseigenschaften gleichermaßen.

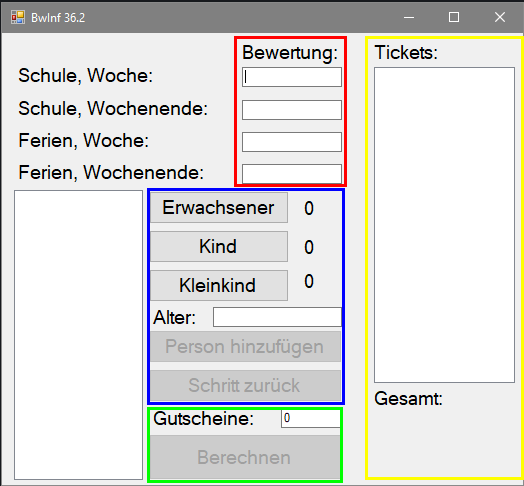
Da aus der Aufgabenstellung nicht klar hervorging, wie die Eingabe erfolgen soll, habe ich mir überlegt die Anzahl der Personen je Altersklasse, die Anzahl der Gutscheine und eine Bewertung jedes der vier Datumstypen einfließen zu lassen. Diese Bewertung stellt lediglich einen Wert dar, der zu dem Preis des besten Angebotes für eines Datumstyp addiert werden muss, bevor dieser mit anderen Preisen verglichen werden kann.

Um solch einen Vergleich vorzunehmen, muss natürlich für die gegebene Kombination von Altersstufen die Lösung zu jedem möglichen Datumstyp ermittelt werden. Das beste gefundene Gesamtergebnis wird ausgegeben.

# Umsetzung

Der eigentliche Ansatz war, es eine Konsolenanwendung zu schreibenb, die Eingaben in bestimmtem Format annimmt. Aus praktischen Gründen habe ich mich anschließend jedoch dafür entschieden, stattdessen ein GUI in Windows Forms zu verwenden. Die Einstellungen dieser Oberfläche sorgen jedoch auch nur dafür, dass ein String mit passender Formatierung erstellt wird und dieser als Ausgangspunkt für die Berechnungen gilt.

Bevor ich aber zur eigentlichen Problemlösung komme, werde ich das GUI erklären:



Dabei gibt es vier Bereiche, welche ich im Screenshot (s. oben) hervorgehoben habe. Im roten Sektor gilt es die Bewertungen der links dargestellten Datumstypen niederzuschreiben. Sollte kein Eintrag vorhanden sein, wird der jeweilige Fall als nicht gewünscht behandelt, das bedeutet die Wertung ist unendlich groß.

Im blauen Bereich finden sich alle Bedienelemente zum Hinzufügen von Personen. Sollte man sich sicher sein, so kann mit den drei Buttons oben links direkt eingestellt werden, wie viele Personen der jeweiligen Altersgruppe vorhanden sein. Besteht Unsicherheit, so kann auch über das Textfeld im unteren Bereich ein Alter eingetragen werden, um dann mit dem darunterliegenden Button eine Person, die automatisch einer Altersgruppe zugeordnet wird, einen weiteren Eintrag vorzunehmen.

Der grüne Bereich wird für den letzten Eingabeschritt benötigt, denn hier wird die Anzahl der Gutscheine abgefragt und eine Option zum Anfordern der günstigsten Eintrittskartenkombination gegeben.

Wenn Eingabe und Berechnung abgeschlossen sind, werden in der ListBox im gelben Sektor untereinander Eintrittskarten, die es zu kaufen gilt, angezeigt. Außerdem wird darunter der ermittelte Gesamtpreis ausgegeben und ein Hinweis auf die zu wählenden Datumseigenschaften gegeben.

Den Code für das GUI werde ich nicht weiter thematisieren, da sowohl der betriebene Aufwand, als auch der informatische Anspruch vernachlässigbar gering sind. Bei Interesse ist jedoch (wie immer) das Projekt anbei zu finden.

Wie bereits erwähnt, erstellt das GUI nur einen string als Basis für alle folgenden Berechnungen.

Das Format dieses strings sieht wiefolgt aus: a b c d e f,g,h,...

Dabei sind a bis d die Bewertungen der Datumsmerkmale, wie sie auf dem Screenshot gelistet sind, e ist die Anzahl der Gutscheine und alle weiteren, durch Kommata getrennten, Argumente sind die Alter der Teilnehmer. Dabei sind diese Alterswerte keinesfalls als die tatsächlichen Werte zu verstehen, sondern stellen lediglich ein Alter für die Representation der Altersgruppe dar. (0, 10 und 18)

Sollte ein Datum gar nicht erwünscht sein, so wird an dieser Stelle der maximale Integerwert (int.MaxValue) eingetragen.

Mit eingeben dieser Zeichenfolge beginnt diese Lösung des eigentlichen, abstrakten Problems.

Dazu muss zunächst der string wieder in die eigentlichen Infromationen zerlegt werden. Diese Prozedur ist zwar umständlich, ist aber wichtig um den Code eventuell in eine andere Anwendunge zu integrieren, da eine eindeutige und simple Schnittstelle existiert.

Um die Information zu erhalten wird der string an Leerzeichen geteilt und jedes Argument einzeln behandelt. Wiederum jedes dieser Argumente wird nochmals an jedem Komma geteilt.

Die ersten vier Werte werden in ein Array geschrieben und stellen die Bewertung der Datumsmerkmale dar.

Das fünfte wird als “couponCount” mit selbiger Bedeutung abgespeichert und die Einzelwerte des sechsten Argumentes werden in einer Liste abgespeichert.

Die Liste mit den Alterswerten wird dann einer weiteren Methode übergeben, die je nach Wert die Anzahl der Personen je Altersgruppe erhöht und dann zurückgibt. Anschließend können die minimalen Preise jedes Datums ermittelt werden. Dazu durchläuft eine Funktion, welche als Argumente die Anzahl der Gutscheine und jeder Altersgruppe hat, die vier Indizes der Datumsmerkmale. (Schule, Woche: 0; Schule, Wochenende: 1; Ferien, Woche: 2; Ferien, Wochenende: 3)

Solange die Begleitung ausreichend ist, wird für jeden dieser Werte dann das beste Angebot an diesem Datum ohne Berücksichtigung der Gutscheine bestimmt. Sollte es sich um einen Tag während der Schulzeit handeln, so werden anschließend noch alle Möglichen Kombinationen bei Verwendung aller möglicher Kombinationen von Gutscheinanzahlen auf Jugendliche und Erwachsene untersucht. Außerdem wird zwischen Gruppen- und Einzelgutscheinen unterschieden. Das bedeutet, dass maximal ein Gutschein auf die gesamte Gruppe verwendet werden kann und auch diese Option berücksichtigt werden muss. Das gestaltet sich jedoch recht simpel, da immer dann ein Gruppengutschein angewendet wird, wenn nicht bereits alle verwendet wurden. Somit wird der zuvor berechnete Preis mit Einzelgutscheinen mit 0.9 multipliziert.

Der somit berechnete Preis wird bei jeder Gutscheinverteilung mit dem zuvor ermittelten besten Wert (zu Beginn ist das der jenige ohne Verwendung von Gutscheinen) verglichen. Sollte der neue Wert besser sein, wird dieser gespeichert und der alte verworfen.

Nun habe ich ausführlich den Lösungsalgortihmus beschrieben, jedoch muss noch die Methode zum berechnen des besten Wertes bei gegebenen Datumsmerkmalen und gegebener Personenverteilung erklärt werden.

Sollte eine unzureichende Begleitung vorhanden sein, wird eine Leere Ticket-Liste und ein negativer Preis zurückgegeben.

Sollte die Gesamtpersonenzahl maximal sechs sein, so wird je nach Datumseigenschaft (Woche / Wochenende) die jeweilige Methode zum Ermitteln der festen, im Voraus berechneten Preise als bester Wert genutzt. Die Berechnung dieser Elementarwerte erfolgte durch Fallunterscheidungen mit Zettel und Stift, die tatsächlichen Werte finden Sie im Abschnitt „Quellcode“ bzw. in der Projektdatei.

Sollte die Anzahl größer als sechs sein, werden auch die zum Datum passenden Methoden aufgerufen, in diesem Fall jedoch für eine dynamische Berechnung.

In diesen Methoden wird zunächst geprüft, ob für die gegebenen Alterswerte bereits (seit Start der Anwendung) ein Angebot berechnet wurde, wenn ja wird selbiges zurückgegeben. Hierbei wird eine „.Clone()“-Methode verwendet, damit eine Kopie des eigentlichen Obejektes verwendet wird. Anderenfalls wäre es möglich die bereits berechneten Werte nachträglich zu verändern.

Sollte kein solches Angebot gefunden werden, fährt das Programm mit der eigentlichen Berechnung fort. Nun wird genannte Altersdistribution so geteilt, dass eine entsteht, die bereits mit Hilfe der Elementarwerte berechnet werden kann, und eine, welche der selben Methode nocheinmal übergeben wird. Dies geschieht für alle möglichen Anzahlen für Jugendliche und Erwachsene, wobei die Gesamtanzahl nie sechs überschreiten darf. (Diese Anzahlen sind natürlich für den Elementarabschnitt.)

Die beiden Angebote, das elementare und der Rest, werden dann basierend auf dem Gesamtpreis mit dem bisher besten Gesamtangebot verglichen. Der bessere Wert wird als neuer bester gespeichert. Durch dieses Verfahren wird am Ende der Funktion mit Sicherheit das beste Angebot gefunden worden sein.

Eine Kopie dieses Angebotes wird dann in der Liste der berechneten idealen Angebote gespeichert und das eigentliche Objekt zurückgegeben.

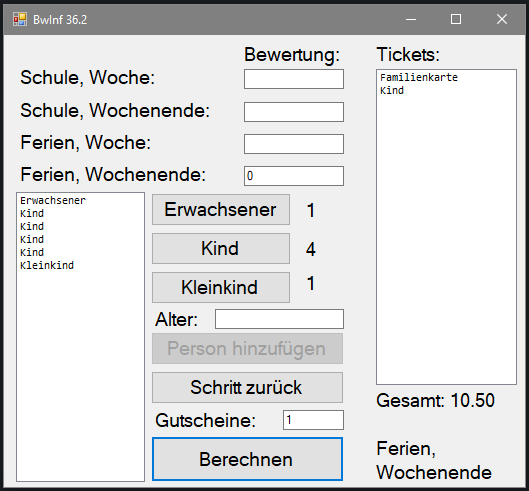
Somit habe ich die Umsetzung vollständig erklärt.

# Beispiele

Beispiel 0, aus der Aufgabenstellung:

Der Input-String lautet: 2147483647 2147483647 2147483647 0 1 10,10,10,10,18,0

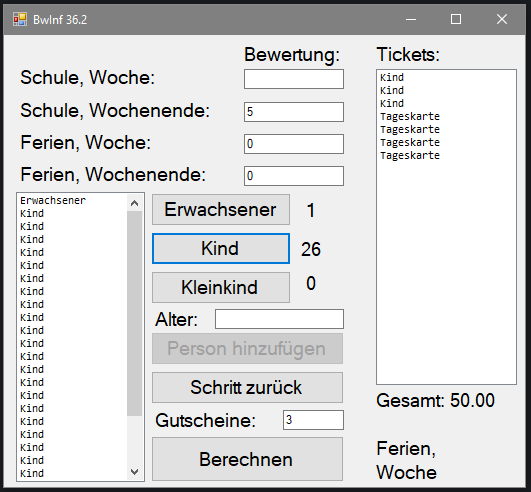
Der zu zahlende Preis ist EURO 10.50 und angemessen wäre eine Familienkarte und eine Einzelkarte für ein Kind.



Beispiel 1, aus den Materialien:

Der Input-String lautet: 2147483647 5 0 0 3 18,10[,10]\*25

Der beste Preis ist EURO 50.00 und angemessen wären 4 Tageskarten und 3 Einzelkarten für Kinder. Als Datum wäre dazu ein Wochentag in den Ferien geeignet.

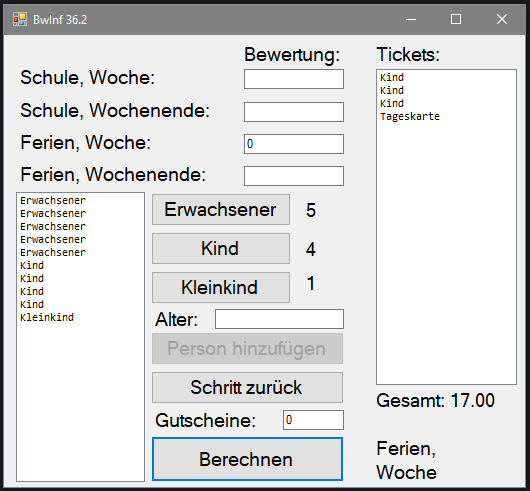


Beispiel 2, aus den Materialien:

Für Wochentage in den Ferien:

Input-String: 2147483647 2147483647 0 2147483647 0 18,18,18,18,00,18,10,10,10,10

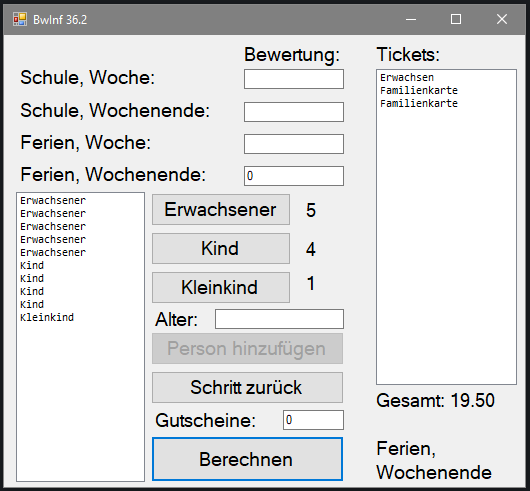
Damit ergibt sich ein Preis von EURO 17.00 mit einer Tageskarte und drei Einzelkarten für Kinder.



Für Wochenenden in den Ferien:

Input-String: 2147483647 2147483647 2147483647 0 0 18,18,18,18,00,18,10,10,10,10

Damit ergibt sich ein Preis von EURO 19.50 mit zwei Familienkarten und einer Einzelkarte für Erwachsene.



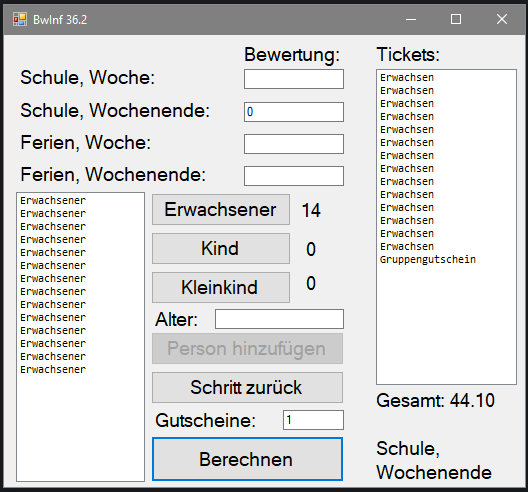
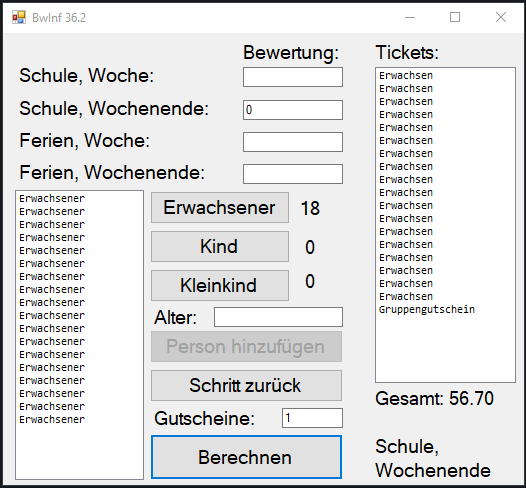
Beispiel 3, aus Materialien:

Getrennt:

Input-Strings: 2147483647 0 2147483647 2147483647 1 18[,18]\*13

2147483647 0 2147483647 2147483647 1 18[,18]\*17

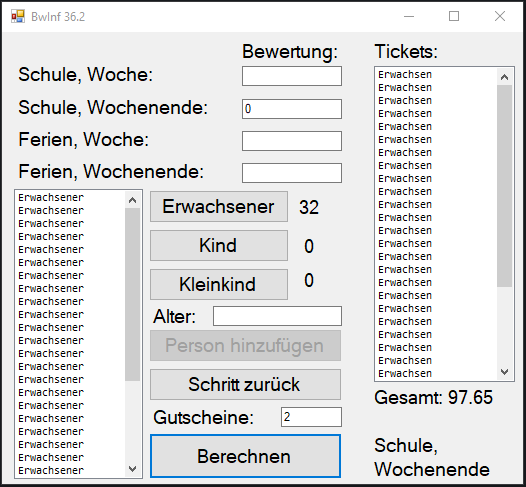
Zusammen kostet das EURO 100.80 (EURO 56.70 + EURO 44.10) mit zwei Gruppengutscheinen und 32 Einzelkarten.



Zusammen:

Input-String: 2147483647 0 2147483647 2147483647 2 18[,18]\*31

Getrennt kostet es EURO 97.65. Es ist also EURO 3.15 günstiger als die getrennte Variante.



# Quellcode

private void calculate(string input)

{

(List<int> ages, double[] differenceNeeded, int couponCount) data

= getData(input);

(int children, int adults, int smallChildren) ageDistribution

= getAgeDistribution(data.ages);

Offer[] minimalPrices = getOffers

(data.couponCount, ageDistribution.adults, ageDistribution.children,

ageDistribution.smallChildren);

int best = 0;

for (int i = 0; i < 4; i++)

{

if (minimalPrices[i].Price

+ Convert.ToDouble(data.differenceNeeded[i])   
 < minimalPrices[best].Price

+ Convert.ToDouble(data.differenceNeeded[best]))

{

best = i;

}

}

returnResult(minimalPrices[best], best);

}

private (List<int> ages, double[] differenceNeeded, int couponCount)

getData(string input)

{

string[] args = input.Split(' ');

List<List<string>> splitArgs = new List<List<string>>();

foreach (string s in args)

{

string[] splitS = s.Split(',');

List<string> splitArg = new List<string>();

foreach (string split in splitS)

{

splitArg.Add(split);

}

splitArgs.Add(splitArg);

}

double[] differenceNeeded = new double[4];

for (int i = 0; i < 4; i++)

{

differenceNeeded[i] = Convert.ToDouble(args[i]);

}

int couponCount = Convert.ToInt32(args[4]);

List<int> ages = new List<int>();

foreach (string s in splitArgs[5])

{

ages.Add(Convert.ToInt32(s));

}

return (ages, differenceNeeded, couponCount);

}

private (int children, int adults, int smallChildren)

getAgeDistribution(List<int> ages)

{

int children = 0;

int adults = 0;

int smallChildren = 0;

foreach (int age in ages)

{

if (age > 16)

{

adults++;

}

else if (age < 4)

{

smallChildren++;

}

else

{

children++;

}

}

return (children, adults, smallChildren);

}

private Offer[] getOffers

(int couponCount, int adults, int children, int smallChildren)

{

if (adults == 0 && smallChildren > 0) { return BadOffers; }

Offer[] minimalPrices = new Offer[4];

int ageCount = children + adults + smallChildren;

for (int i = 0; i < 4; i++)

{

Offer currentPrice

= getPrice(smallChildren, children, adults, i);

if (i < 2)

{

if (couponCount >= ageCount)

{ currentPrice = OnlyCoupons(ageCount); }

else

{

for (int childrenCoupons = 0;

childrenCoupons <= couponCount

&& childrenCoupons <= children;

childrenCoupons++)

{

for (int adultCoupons = 0;

adultCoupons <= couponCount – childrenCoupons

&& adultCoupons <= adults;

adultCoupons++)

{

Offer tempPrice = getPrice(0, children

- childrenCoupons, adults - adultCoupons, i);

for (int j = 0; j < childrenCoupons

+ adultCoupons; j++)

{ tempPrice.Tickets.Add("Einzelgutschein"); }

if (childrenCoupons + adultCoupons < couponCount)

{

tempPrice.Price \*= 0.9;

tempPrice.Tickets.Add("Gruppengutschein");

}

if (tempPrice.Price < currentPrice.Price)

{

currentPrice = tempPrice.Clone();

tempPrice = null;

}

}

}

}

}

minimalPrices[i] = currentPrice;

currentPrice = null;

}

return minimalPrices;

}

private Offer OnlyCoupons(int ageCount)

{

List<string> coupons = new List<string>();

for (int j = 0; j < ageCount; j++)

{

coupons.Add("Einzelgutschein");

}

return new Offer(coupons, 0);

}

public static Offer getPrice

(int smallChildren, int children, int adults, int day)

{

if (smallChildren == 0 || adults > 0)

{

Offer best = OnlySinglePrices(children, adults);

if (adults + children <= 6)

{

if (day % 2 == 0)

{ best = FixedPricesDuringWeek(children, adults); }

else { best = FixedPricesDuringWeekEnd(children, adults); }

}

else

{

Offer contestant = new Offer

(new List<string>(), double.MaxValue);

if (day % 2 == 0)

{ contestant = CombinedPricesDuringWeek(children, adults); }

else { contestant

= CombinedPricesDuringWeekEnd(children, adults); }

if (best.Price > contestant.Price) { best = contestant; }

}

return best;

}

else { return new Offer(new List<string>(), -1); }

}

private static Offer OnlySinglePrices(int children, int adults)

{

List<string> total = new List<string>();

for (int i = 0; i < adults; i++)

{

total.Add("Erwachsen");

}

for (int i = 0; i < children; i++)

{

total.Add("Kind");

}

return new Offer(total, children \* 2.50 + adults \* 3.50);

}

private static Offer CombinedPricesDuringWeekEnd

(int children, int adults)

{

foreach (var v in calculatedPricesWE)

{

if (v.input.adults == adults && v.input.children == children)

{ return v.output.Clone(); }

}

Offer best = OnlySinglePrices(children, adults);

if (children + adults > 6)

{

for (int i = 1; i <= 6; i++)

{

for (int j = 0; j <= i && j <= children; j++)

{

if (adults - (i - j) >= 0)

{

Offer part1 = FixedPricesDuringWeekEnd(j, i - j);

Offer part2 = CombinedPricesDuringWeekEnd

(children - j, adults - (i - j));

if (part1.Price + part2.Price < best.Price)

{

part1.Tickets.AddRange(part2.Tickets);

best = new Offer(part1.Tickets, part1.Price

+ part2.Price);

}

}

}

}

}

else { best = FixedPricesDuringWeekEnd(children, adults); }

calculatedPricesWE.Add(((children, adults), best.Clone()));

return best;

}

private static Offer CombinedPricesDuringWeek(int children, int adults)

{

foreach (var v in calculatedPricesW)

{

if (v.input.adults == adults && v.input.children == children)

{ return v.output.Clone(); }

}

Offer best = OnlySinglePrices(children, adults);

if (children + adults > 6)

{

for (int i = 1; i <= 6 && i <= children + adults; i++)

{

for (int j = 0; j <= i && j <= children; j++)

{

if (adults - (i - j) >= 0)

{

Offer part1 = FixedPricesDuringWeek(j, i - j);

Offer part2 = CombinedPricesDuringWeek

(children - j, adults - (i - j));

if (part1.Price + part2.Price < best.Price)

{

part1.Tickets.AddRange(part2.Tickets);

best = new Offer(part1.Tickets, part1.Price

+ part2.Price);

}

}

}

}

}

else { best = FixedPricesDuringWeek(children, adults); }

calculatedPricesW.Add(((children, adults),

new Offer(best.Tickets, best.Price)));

return best;

}

Zuletzt folgen die beiden Methoden mit den. bereits im Voraus bestimmten. Minimalpreisen:

private static Offer FixedPricesDuringWeekEnd(int children, int adults)

{

if ((adults >= 2 && children >= 2) || (adults >= 1 && children >= 3))

{

if (adults == 4 && children == 2)

{

return new Offer(new List<string>() { "Familie", "Erwachsen", "Erwachsen" }, 15);

}

else if (adults == 3 && children == 3)

{

return new Offer(new List<string>() { "Familie", "Erwachsen", "Kind" }, 14);

}

else if (adults == 3 && children == 2)

{

return new Offer(new List<string>() { "Familie", "Erwachsen" }, 11.50);

}

else if (adults == 2 && children == 4)

{

return new Offer(new List<string>() { "Familie", "Kind", "Kind" }, 13);

}

else if (adults == 2 && children == 3)

{

return new Offer(new List<string>() { "Familie", "Kind" }, 10.50);

}

else if (adults == 2 && children == 2)

{

return new Offer(new List<string>() { "Familie" }, 8);

}

else if (adults == 1 && children == 5)

{

return new Offer(new List<string>() { "Familie", "Kind", "Kind" }, 13);

}

else if (adults == 1 && children == 4)

{

return new Offer(new List<string>() { "Familie", "Kind" }, 10.50);

}

else if (adults == 1 && children == 3)

{

return new Offer(new List<string>() { "Familie" }, 8);

}

else

{

throw new Exception();

}

}

else

{

return OnlySinglePrices(children, adults);

}

}

private static Offer FixedPricesDuringWeek(int children, int adults)

{

Offer total = OnlySinglePrices(children, adults);

total.Price \*= 0.8;

if (adults + children == 4)

{

if (children == 0 || children == 1)

{

total = new Offer(new List<string>() { "Tag" }, 11);

}

else if (children == 2 || children == 3)

{

total = new Offer(new List<string>() { "Familie" }, 8);

}

else

{

total = new Offer(new List<string>() { "Kind", "Kind", "Kind", "Kind" }, 8);

}

}

else if (adults + children == 5)

{

if (children >= 3 && adults >= 1)

{

total = new Offer(new List<string>() { "Familie", "Kind" }, 10);

}

else

{

total = new Offer(new List<string>() { "Tag" }, 11);

}

}

else if (adults + children == 6)

{

total = new Offer(new List<string>() { "Tag" }, 11);

}

return total;

}