**Структуры данных и алгоритмы**

**Курсовая работа студента Гридасова А. Ю.**

**Новосибирский государственный технический университет**

**Кафедра прикладной математики**

**Новосибирск**

**1998**

**Условие задачи**

Имеется некоторое конечное число городов, которые связаны транспортной сетью, состоящей из авиа, железнодорожных, автомобильных и водных рейсов произвольного направления и включающих произвольное число городов. Стоимость проезда различна по классам. Рейсы отправляются по недельному расписанию. При пересадки между рейсами должно быть не менее 2-х часов. По заданным начальному и конечному городам, дате желаемого отправления, максимальному времени пути и максимальной стоимости и максимальному числу пересадок выдать все возможные маршруты, так, чтобы маршруты с меньшей датой и временем прибытия отображались раньше, чем с большим.

**Анализ задачи**

Транспортная схема представляет собой направленный взвешенный мультиграф. Каждая дуга характеризуется принадлежностью к рейсу, временем пути, ценой каждого из классов, временем отправления.

**Входными данными является:**

Транспортная система. (города и все рейсы)

Начальный, конечный город, ориентировочная дата и время отправления, максимальное время пути максимальная цена, максимальное количество пересадок.

Причем данные первой группы изменяются крайне редко и задаются разработчиком транспортной системы, а данные второй группы изменяются от задачи к задачи и задаются каждым пользователем.

Результатом работы программы является конечное множество маршрутов. Два маршрута мы будем считать различными, если они отличаются хотя бы одним городом следования или хотя бы одним рейсом. После того, как найдены все маршруты они сортируются по времени прибытия.

Метод решения – метод последовательных испытаний. Поиск решений будет осуществляться рекурсивно, причем максимальная глубина рекурсии будет равна максимальному количеству пересадок. Так как мы имеем ограничения по некоторым параметрам то мы можем отсечь заведомо ошибочную ветвь поиска решений, сделав проверку на превышение параметров. Это позволит выиграть дополнительное время. (о реализации более подробно п.4)

**Выбор и обоснование форм представления данных.**

Так как транспортная система включает в себя достаточно большой объем информации, в целях доступа к большему объему памяти, также в целях более рационального использования памяти и по причине недопустимости использования статических объектов в некоторых случаях, в программе для внутреннего представления широко используются динамические объекты.

Для объединения большого количества данных в одном объекте, а также для реализации динамических объектов используется комбинированный тип (запись).

Для внутреннего хранения информации о рейсах используется цепь (однонаправленный список) PFlight с 7-ю информационными полями различных типов:

Для хранения названия компании-перевозчика используется тип string[20] так по понятным причинам.

Для хранения номера рейса используется тип string[10] т.к. в номерах рейса часто используются различные не цифровые шифры, индексы, коды.

Общее количество городов – интервальный тип. Автоматическая проверка границ этого типа повышает надежность программы.

Таблица отправления представляется своим динамическим типом. Этот динамический тип представляет совой цепь с одним информационным полем , содержащим время отправления в минутах от начала недели ( например Вт 17:42 будет записан числом 1\*24\*60+17\*60+42). Такая форма хранения времени сочетает с себе компактность и легкость пересчета (пересчет требуется только при вводе и выводе, а в программе в большинстве случаев пересчет не нужен. Динамический тип использован по причине большого разброса в частоте отправления рейсов (могут быть рейсы, отправляющиеся каждый день через час, а могут быть рейсы отправляющиеся раз в неделю).

Маршрут рейса также представляется своим динамическим типом – однонаправленным динамическим списком. Причина использования списка аналогична полю отправления -разброс. Так например самолеты обычно не имеют более 4 посадок, а поезда наоборот делают много остановок. Информационное поле содержит информацию не об одной а о 4-х станциях, т.е. представляет собой массив из 4 элементов. Это сделано для экономии памяти на избыточных указателях. При этом усложнение кода программы незначительно.

Тип транспорта кодируется числом 1..4. По понятным причинам. Перечислимый тип не был использован для упрощения ввода данных из внешнего файла.

Классы, которые предоставляет рейс, представляется в виде массива индексом является класс, а типом элемента – булевский.

Внутренне каждый город обозначается своим номером (элемент интервального типа), что уменьшает расходы памяти и упрощает вычисление. А для хранения названий городов и их координат для отображения на экране используется свой тип – массив, элементами которого являются записи с полями для названия города и координат. Статический массив используется для простого и быстрого доступа к этим данным.

Для хранения времени пути используется тип Integer. Отрицательные числа нужны для контроля за превышением времени пути.

Для хранения цены используется тип LongInt. Причины выбора этого типа очевидны.

Тип Pattern для хранения исходных параметров поиска представляет собой запись с полями: время отправления относительно понедельника в минутах, начальный и конечный город, допустимые типы транспорта, допустимые классы, максимальное количество пересадок, максимальное время пути, максимальная цена, допустимые классы. Выбор типов для всех полей кроме «допустимые типы транспорта» обсуждался выше. Для поля «допустимые типы транспорта» выбран массив где тип индекс – это тип транспорта, а тип элемента – булевский. Это сделано по причине того что маршрут может включать. Поездки на разных видах транспорта (тех где в значение true). Запись использована чтоб передавать все данные единым объектом в процедуру поиска маршрута.

Тип Link предназначен для хранения информации о части маршрута между двумя городами, соединенными одним рейсом. Кроме ссылки на предыдущую такую часть он содержит ссылку на рейс, коды начального и конечного города, общую цену участка , время отправления, относительно заданного пользователем время отправления, общее время пути по участку. Типы полей и обоснования их выбора обсуждались выше. В совокупности цепочка таких элементов задает один маршрут.

Тип AnswerList предназначен для ответа - множества всех допустимых маршрутов. Представляет из себя однонаправленный список, в каждом элементе которого кроме ссылки на следующий имеется поле типа маршрут (Link), общее время пути, общая максимальная и минимальная цена, количество пересадок. Типы полей и обоснование обсуждались выше.

**Внешнее представление:**

Транспортная система хранится во внешнем текстовом файле. Файл может быть создан любым текстовым редактором. В файле указывается следующее:

Количество городов. Со следующей строки начинается информация о городах: название города, на следующей строчке координаты. После всех городов начинается информация о рейсах: компания, номер, тип, классы, количество станций; номер города, время пути, время стоянки цена по классам, для каждого города; время отправления от начальной станции.

Так как эта информация редактируется крайне редко, причем разработчиком сети, то такой способ является наиболее приемлемым.

Название городов вводятся как строки, дата – в любом формате (дд-мм-гг, дд-мм-гггг, дд-мес-гг и т.п.) время чч:мм. По умолчанию полагается дата – текущий день, время 0:00.

Максимальное время пути, максимальное число пересадок, максимальная цена – вводятся как числа.

**Алгоритм**

Begin

{Загрузка транспортной схемы};

{Ввод исходных данных и заполнение шаблона};

{Вызов процедуры поиска с введенным шаблоном, построенная часть маршрута - пустая};

{Вывод полученного множества маршрутов}

End

{Процедура поиска маршрута с данным шаблоном и уже построенной частью маршрута}

Begin

While {просмотрены не все рейсы} do begin

If {соответствует тип транспорта} and {Текущий рейс не равен предыдущему}then

Begin

If {город отправления присутствует в рейсе, причем раньше конечной станции} then begin

{Рассчитать время отправления ближайшего следующего рейса}

Repeat

{Перейти к следующему городу};

{Рассчитать время дороги с учетом нового участка}

If {текущий город еще не проезжали} and {время пути не превышает максимального}

and {количество пересадок не превышает максимального} and {не приехали[[1]](#footnote-1)}

then {Добавить к маршруту проеханный участок. Вызвать процедуру поиска маршрута

от текущего города до конечного с новыми значениями времени}

Until {текущий город проезжали} or {время исчерпано} or {приехали} or {конец рейса};

If {приехали} and {время не превышено} and {минимальная цена рейса не выше

допустимой} then {Добавить построенный маршрут в мно-во ответов на нужное место}

end;

end;

{Перейти к следующему рейсу}

end;

end

Текст программы на языке Pascal

uses Crt, Date, Graph;

Const MaxCity=100; MClass=6;

Type

CityCode=1..maxcity; {Внутрений код города}

Week=0..10079; {Тип время в минутак с 0:00 понедельника}

DayTable=^IDayTable; {Таблица отправлений от начальной станции}

IDayTable=record

Time:Week;

Next:DayTable;

end;

WayKind=1..4; {Тип пути (аэро, море, ж.д, авто)}

WayClass=1..MClass; {Класс или тип перевозки}

Cities=array[CityCode] of {Названия и координаты городов}

record

name:string[20];

x,y:word;

end;

mcost=array[wayclass] of longint; {Таблица стоимости по классам}

Way=record

City:Citycode;

Delay,Reboard:Word;

Cost:mcost;

end;

WayP=^way;

PWay=^Way1; {Информация о городах следования рейса}

Way1=record

Way:array [1..4] of way;

next:PWay;

end;

wclass=array [WayClass] of boolean;

PFlight=^Flight;

Flight=record {Информация о рейсе}

company:string[20];

number:string[10];

totalstation:CityCode;

table:DayTable;

path:PWay;

kind:WayKind;

class:WClass;

next:PFlight;

end;

Blank=record {Шаблон для поиска пути}

delay:Week;

BCity,ECity:CityCode;

Kind:array [WayKind] of boolean;

ReBoading:CityCode;

WayTime:Integer;

Cost:Longint;

Class:WClass;

end;

Link=^CityList; {Цепочка рейсов для проезда от начала до конца}

CityList=record {Информация о проезде между двумя пунктами одним рейсом}

DDelay:Word;

waytime:word;

cost:mcost;

Bcity,Target:CityCode;

Flight:PFlight;

Last:Link;

end;

AnswerList=^IAnswer; {Список всех возможных маршрутов следования}

IAnswer=record

path:link;

reboard:citycode;

mincost,maxcost:longint;

waytime:word;

next:AnswerList;

end;

var Lanswer:AnswerList; {глобальная переменная - начало списка маршрутов }

{Добавления нового найденного маршрута}

Procedure Answer(A:Link;cost:longint);

var P,Q:Link; d,s1,s2:word; W,PAnswer:answerlist; r:citycode;

function min(a:mcost):longint; {Минимальная стоимость по классам}

var i:integer; m:longint;

begin

m:=1000000000;

for i:=1 to Mclass do if (m>a[i]) and (a[i]>0) then m:=a[i];

min:=m

end;

function max(a:mcost):longint; {Максимальная стоимость по классам}

var i:integer; m:longint;

begin

m:=a[1];

for i:=2 to Mclass do if m<a[i] then m:=a[i];

max:=m

end;

begin

new(PAnswer);

Panswer^.path:=nil;

P:=A;

s1:=0; s2:=0; {верхняя и нижняя границы цены}

r:=1; {количество пересадок}

d:=0; {время пути}

Repeat

s1:=s1+min(P^.cost); {Подсчет суммы параметров по рейсам в маршруте}

s2:=s2+max(P^.cost);

d:=d+P^.ddelay+P^.waytime;

P:=P^.last; {Переход к следующему рейсу в маршруте}

inc(r);

Until P=nil;

if s1<=cost then begin {Если соответствует цена}

P:=A;

Repeat

new(Q); {Сборка цепочки рейсов маршрута}

Q^:=P^;

Q^.last:=Panswer^.path;

Panswer^.path:=Q;

P:=P^.last; {Переход к следующему рейсу в маршруте}

Until p=nil;

Panswer^.mincost:=s1; Panswer^.maxcost:=s2; {Сохранение сумарных цен и времени}

Panswer^.waytime:=d; Panswer^.reboard:=r; {и числа пересадок в элементе маршрута}

W:=LAnswer;

While (W^.next<>nil) and ((W^.next)^.waytime<d) do W:=W^.next; {Поиск места в соответствии времени пути}

While (W^.next<>nil) and ((W^.next)^.reboard<r) and ((W^.next)^.waytime=d) do W:=W^.next; {Поиск места по кол-ву пересадок}

Panswer^.next:=W^.next; {Добавление маршрута в найденное место}

W^.next:=Panswer;

end

end;

{Возвращает ссылку на информацию об I-ой станции следования}

Function CityInPath(A:Pway; I:citycode):WayP;

var P:Pway;

Begin

P:=A;

While I>4 do begin I:=I-4; P:=P^.next end; {Поиск четверки в которой данная станция}

CityInPath:=@P^.Way[I]; {Результат}

end;

const ReBoadingDelay=120; {Минимальное время пересадки}

{Возвращает время до следещего после указанного времени time отоправление от станции}

{номер N рейса A}

Function DepartureDelay(A:PFlight; N:CityCode; time:week ):word;

var S:word; I:1..4; P:PWay; Q:DayTable;

begin

P:=A^.path;

S:=0;

While N>4 do begin

N:=N-4;

For I:=1 to 4 do S:=S+P^.Way[I].delay+P^.Way[I].reboard; {подсчет времени пути по полным четверкам}

P:=P^.next;

end;

For I:=1 to N do S:=S+P^.Way[I].delay+P^.Way[I].reboard; {Подсчет по неполной четверке}

time:=(10080+time-(S mod 10080)) mod 10080; {Время отправления этого рейса от начальной станции}

Q:=A^.Table;

while (Q<>nil) and (Q^.time<time+ReboadingDelay) do Q:=Q^.next; {Поиск ближайшего времени на текущей неделе}

If Q<>nil then Departuredelay:=Q^.time-time else {Если на текущей неделе не найден}

DepartureDelay:=10080-time+(A^.Table)^.time; {Поиск ближайщего времени на следующей неделе}

end;

{Поиск всех возможных маршрутов, удовлетворяющих Pattern}

Procedure Search (FlightList:Pflight; const Pattern:Blank; Path:Link);

Var P:Pflight; I,J:CityCode; D,DDelay:Word; K:WayClass; B1,B2:Boolean;

NPattern:Blank; NPath:Link; c:Longint;

{Проверка допустимости маршрута (проверка дублирования города)}

Function Posible (P:Link; L:CityCode):Boolean;

Var b:boolean; i:citycode; Q:pway;

Begin

b:=true;

While (P<>nil) and b do begin {Просмотр всех предидущих пересадок}

Q:=P^.flight^.path;

i:=1;

while Q^.way[i].city<>P^.bcity do begin {Поиск города отправления}

i:=(i mod 4)+1; if i=1 then Q:=Q^.next;

end;

repeat

b:=Q^.way[i].city<>L; {Проверка города на дублирование}

i:=(i mod 4)+1; if i=1 then Q:=Q^.next

until (Q^.way[i].city=P^.target) or not b; {переход к следующему пока не город назначения}

p:=p^.last

end;

Posible:=b;

End;

begin

New(NPath);

NPath^.last:=Path;

P:=FlightList;

While P<>nil do begin {Просмотр всех рейсов}

if ((Path=nil) or (P<>Path^.Flight)) and Pattern.Kind[P^.Kind] then {не повторяется рейс и сответствует тип перевозки}

begin

I:=1; {Поиск среди городов следования начальный пункт}

While (I<P^.TotalStation-1) and (CityInPath(P^.path, I)^.city<>Pattern.BCity) do inc (I);

If CityInPath(P^.path, I)^.city=Pattern.BCity then begin {Если начальный найден}

NPattern:=Pattern; {Подготовка нового шаблона и новой пересадки}

if Npattern.reboading>1 then dec(Npattern.reboading);

Npath^.flight:=P;

For K:=1 to Mclass do Npath^.cost[k]:=0;

Npath^.bcity:=pattern.bcity;

Npath^.Ddelay:=DepartureDelay(P,I,Pattern.delay);

Npath^.waytime:=0;

J:=I;

Repeat {просмотр следующих городов}

Inc(J);

{Внесение исправлений в шаблон и элемент маршрута о цене и времени}

For K:=1 to MClass do If Pattern.Class[K] and P^.class[K] then

Npath^.cost[k]:=Npath^.cost[k]+CityInPath(P^.path,J)^.Cost[K];

Npath^.waytime:=Npath^.waytime+CityInPath(P^.path,J)^.delay;

Npath^.target:=CityInPath(P^.path,J)^.City;

NPattern.Bcity:=CityInPath(P^.path,J)^.City;

Npattern.WayTime:=Pattern.WayTime-Npath^.ddelay-Npath^.waytime;

Npattern.Delay:=(pattern.Delay+Npath^.Ddelay+Npath^.wayTime) mod 10080;

B1:=Posible(Path,CityInPath(P^.path,J)^.City) and (NPattern.WayTime>=0);

{Проверка: не превышены лимиты времени и стоимости и нет повтора пути}

B2:=CityInPath(P^.path,J)^.city=Pattern.ECity; {приехали?}

{Если не приехали и лимиты не превышены то делаем рассмотроим маршруты от текущего до конечного городов}

if B1 and (not B2) and (Pattern.reboading>1) then Search(FlightList,Npattern,Npath);

Npath^.waytime:=Npath^.waytime+CityInPath(P^.path,J)^.reboard;

Until (not B1) or B2 or (J>=P^.totalStation); {Выходим, если есть нарушения или рейс закончился или прехали}

If B2 and B1 then Answer(Npath,pattern.cost); {Если приехали, добавить маршрут в список}

end {найден начальный город}

end; {маршрут подходит по типу}

P:=P^.next; {переход к следущему циклу}

end;

Dispose(NPath)

end;

{Загрузка исходных данных из файла}

Function Load (A:PFlight; FName:String;var City:cities):PFlight;

Var

Source:Text; P:Pflight; I:WayClass; J,MC:CityCode; K:byte;

C:char; Q:Pway; G,L:DayTable; D:string[8];

Begin

Assign(Source,FName);

Reset(Source);

readln(Source,MC); {Количество городов}

{Считывание название городов и координат на карте }

For J:=1 to MC do begin ReadLn(source,City[j].name); readln(source,city[j].x,city[j].y) end;

While Not EOF(Source) do begin

New(P);

P^.Next:=A;

A:=P;

{Общая информация о рейсе}

ReadLn(Source, P^.company);

ReadLn(Source, P^.number);

ReadLn(Source, P^.kind);

{Стоимость каждого из классов}

For I:=1 to MClass do begin Read(Source,C); P^.class[i]:=C='X' end;

ReadLn(Source, P^.TotalStation);

New(P^.path);

Q:=P^.path;

{информация о городах следования времени пути, стоянках}

For J:=1 to P^.TotalSTation do begin

K:=((J-1) mod 4)+1;

Read(Source,Q^.Way[K].City,Q^.Way[K].Delay,Q^.Way[K].Reboard);

For I:=1 to MClass do If P^.class[I] then Read(Source,Q^.Way[K].cost[I])

else Q^.Way[K].cost[I]:=0;

If (J mod 4)=0 then begin

If (J<>P^.TotalStation) then begin New(Q^.Next); Q:=Q^.next end

else Q^.next:=nil;

end;

ReadLn(Source);

end;

New(P^.Table);

G:=P^.Table;

L:=G;

{Информация о отправлении из начального пункта}

While Not EOLn(Source) do begin

Read(Source,D);

G^.Time:=(ord(D[1])-ord('0')-1)\*1440+((ord(D[3])-ord('0'))\*10+ord(D[4])-ord('0'))\*60

+(ord(D[6])-ord('0'))\*10+ord(D[7])-ord('0');

if L^.time>G^.time then write('Wrong data');

If not EOLn(Source) then begin New(G^.next); G:=G^.next end else G^.next:=nil;

end;

ReadLn(Source);

end;

Load:=A;

end;

const line='--------------------------------------------------------------------------------';

procedure graphout(const city:cities);

var

grDriver: Integer;

grMode: Integer;

p:citycode;

begin

grDriver := Detect;

InitGraph(grDriver, grMode,'');

setcolor(12);

outtextxy(200,0,'Карта транспортной схемы');

p:=1;

while (p<maxcity) and (city[p].name<>'') do begin

setcolor(5);

fillellipse(4\*city[p].x,380-3\*city[p].y,2,2);

setcolor(11);

outtextxy(4\*city[p].x+5,376-3\*city[p].y,city[p].name);

inc(p)

end;

end;

var List:PFLight; pattern:blank; st:string; p:answerlist;

city:cities; a:dat;

Procedure Input(var Pattern:blank; var a:dat);

var i:citycode; st:string; b:dat; w:real;

begin

with pattern do begin

GotoXY(30,1);

WriteLn('Ввод исходных данных');

write(line);

repeat

write('Начальный город ... ');

readln(st);

Bcity:=1; while (BCity<Maxcity) and (City[BCity].name<>st) do inc(BCity);

until BCity<>MaxCity;

repeat

write('Конечный город ... ');

readln(st);

Ecity:=1; while (ECity<Maxcity) and (City[ECity].name<>st) do inc(ECity);

until Ecity<>MaxCity;

repeat

gotoxy(1,5);

WriteLn('Дата отправление:');

DTInput(a);

delay:=a.Dweek\*1440+a.time;

Write('Максимальное время пути (сутки):');

readln(w);

waytime:=round(1440\*w);

until waytime>0;

write('Максимальная стоимость ... ');

ReadLn(cost);

write('Максимальное число пересадок ... ');

readln(reboading);

write('Тип перевозки (авиа,ж.д.,авто,водн.) ... ');

readln(st);

if st='' then for i:=1 to 4 do kind[i]:=true else

for i:=1 to 4 do kind[i]:=(st[i]='Y') or (st[i]='y') or (st[i]='X') or (st[i]='x');

write('Допустимые классы 123456 ... ');

readln(st);

if st='' then for i:=1 to 4 do class[i]:=true else

for i:=1 to 4 do class[i]:=(st[i]='Y') or (st[i]='y') or (st[i]='X') or (st[i]='x');

end;

end;

procedure outres(p:Answerlist; a:dat);

var k:word; q:link; b:dat; i:citycode; y:pway; c:byte;

begin

k:=0;

while P<>nil do begin

inc(k);

{ write(p^.path^.bcity);}

Q:=P^.path;

b:=a;

while Q<>nil do begin

write(city[q^.bcity].name);

Writeln(' <',q^.flight^.company,q^.Flight^.Number,'> ',city[Q^.Target].name);

newdat(b,Q^.ddelay,b);

write('Отправление: '); writedat(b);

newdat(b,Q^.waytime,b);

write(' Прибытие: '); writedat(b); writeln;

Q:=Q^.last;

end;

newdat(a,p^.waytime,b);

writeln (' цена: ',P^.mincost,' - ',p^.maxcost);

readln(st);

if st='p' then begin

graphout(city);

q:=p^.path;

c:=2;

while q<>nil do begin

i:=1;

y:=q^.flight^.path;

while y^.way[i].city<>q^.bcity do begin

i:=(i mod 4)+1; if i=1 then y:=y^.next;

end;

setcolor(c);

moveto(4\*city[q^.bcity].x,380-3\*city[q^.bcity].y);

repeat

i:=(i mod 4)+1; if i=1 then y:=y^.next;

lineto(4\*city[y^.way[i].city].x,380-3\*city[y^.way[i].city].y);

until (y^.way[i].city=q^.target);

Q:=Q^.last; inc(c);

end; repeat until keypressed; CloseGraph;

end;

P:=P^.next;

end;

if k=0 then write('При данных условиях добраться нельзя') else writeln('Всего ',k,' маршшрутов');

end;

Begin

List:=Load(nil,'trafic',city);

graphout(city);

repeat until keypressed;

closegraph;

Input(pattern,a);

new(lanswer);

lanswer^.next:=nil;

Search(List,pattern,nil);

outres(Lanswer^.next,a);

end.

**Выбор и обоснование набора тестов**

В качестве транспортной системы бала взята система, состоящая из 23 городов, соединенных 19 прямыми и таким же числом обратных рейсами. Название городов и перевозчиков вымышленные. Рейсы были разработаны так, что имеется несколько крупных транспортных развязок: Palace of Dream, Diamond World, Golden River, Seaside City; и несколько «удаленных» городов: Far Star City, Oil City, North Star City.

Разные рейсы отправляются от 3 до 18 раз в неделю.

1. Общий тест

Начальный город ... Tropic Port

Конечный город ... Beatiful

Дата отправление:

Дата ... 8.5.1998 Пт

Время ... 0:0

Максимальное время пути (сутки):3

Максимальная стоимость ... 200

Максимальное число пересадок ... 3

Тип перевозки (авиа,ж.д.,авто,водн.) ...

Допустимые классы 123456 ...

Tropic Port <GoldenAirBridge004> Palace Of The Dream

Отправление: 14:29 8.5.1998 Пт Прибытие: 19:14 8.5.1998 Пт

Palace Of The Dream <GoldenAirBridge009> Diamond World

Отправление: 2:15 9.5.1998 Пт Прибытие: 5:15 9.5.1998 Пт

Diamond World <DiamondAirlines003> Beatiful

Отправление: 17:20 9.5.1998 Пт Прибытие: 19:20 9.5.1998 Пт

цена: 195 – 250

Tropic Port <GoldenAirBridge004> Lakes Land

Отправление: 14:29 8.5.1998 Пт Прибытие: 16:29 8.5.1998 Пт

Lakes Land <DiamondAirlines006> Diamond World

Отправление: 0:25 9.5.1998 Пт Прибытие: 3:25 9.5.1998 Пт

Diamond World <DiamondAirlines003> Beatiful

Отправление: 17:20 9.5.1998 Пт Прибытие: 19:20 9.5.1998 Пт

цена: 165 - 195

Tropic Port <DeepWater02> Oil City

Отправление: 12:0 8.5.1998 Пт Прибытие: 4:40 9.5.1998 Пт

Oil City <TransExpress002> Beatiful

Отправление: 12:0 9.5.1998 Пт Прибытие: 16:10 10.5.1998 Пт

цена: 75 – 105

2. Тест с «урезанием бюджета»

Начальный город ... Tropic Port

Конечный город ... Beatiful

Дата отправление:

Дата ... 8.5.1998 Пт

Время ... 0:0

Максимальное время пути (сутки):3

Максимальная стоимость ... 180

Максимальное число пересадок ... 3

Тип перевозки (авиа,ж.д.,авто,водн.) ...

Допустимые классы 123456 ...

Tropic Port <GoldenAirBridge004> Lakes Land

Отправление: 14:29 8.5.1998 Пт Прибытие: 16:29 8.5.1998 Пт

Lakes Land <DiamondAirlines006> Diamond World

Отправление: 0:25 9.5.1998 Пт Прибытие: 3:25 9.5.1998 Пт

Diamond World <DiamondAirlines003> Beatiful

Отправление: 17:20 9.5.1998 Пт Прибытие: 19:20 9.5.1998 Пт

цена: 165 - 195

Tropic Port <DeepWater02> Oil City

Отправление: 12:0 8.5.1998 Пт Прибытие: 4:40 9.5.1998 Пт

Oil City <TransExpress002> Beatiful

Отправление: 12:0 9.5.1998 Пт Прибытие: 16:10 10.5.1998 Пт

цена: 75 – 105

3. Уменьшение числа пересадок

Начальный город ... Tropic Port

Конечный город ... Beatiful

Дата отправление:

Дата ... 8.5.1998 Пт

Время ... 0:0

Максимальное время пути (сутки):3

Максимальная стоимость ... 200

Максимальное число пересадок ... 2

Тип перевозки (авиа,ж.д.,авто,водн.) ...

Допустимые классы 123456 ...

Tropic Port <DeepWater02> Oil City

Отправление: 12:0 8.5.1998 Пт Прибытие: 4:40 9.5.1998 Пт

Oil City <TransExpress002> Beatiful

Отправление: 12:0 9.5.1998 Пт Прибытие: 16:10 10.5.1998 Пт

цена: 75 – 105

4. Нереальные условия

Начальный город ... Tropic Port

Конечный город ... Beatiful

Дата отправление:

Дата ... 8.5.1998 Пт

Время ... 0:0

Максимальное время пути (сутки):3

Максимальная стоимость ... 200

Максимальное число пересадок ... 1

Тип перевозки (авиа,ж.д.,авто,водн.) ...

Допустимые классы 123456 ...

При данных условиях добраться нельзя

**Анализ результатов**

Время пути зависит от дня оправления.

По причине ожидания рейса можно с меньшим числом пересадок добраться позже, чем с большим

Дороже – не значит быстрее

Для нормальной транспортной системы нужно как можно больше больших транспортных узлов.

1. Текущий город – есть пункт назначения. [↑](#footnote-ref-1)