Содержание

Введение

Самолёты крепко обосновались в нашей жизни. Уже в середине прошлого века самолёт не был чем-то необычным. А время шло. И вот уже миллионы людей летают ежемесячно по всему миру на самых разнообразных воздушных лайнерах.

Но самолёты не автомобили – они намного массивнее и много менее поворотливы. А самое главное отличие состоит в том, что во время полёта они не могут остановиться в случае опасности, когда для автомобиля это один из самых безопасных способов разрешить сложную ситуацию. Это всё не доставляло особых проблем, пока самолёты были явлением редким, экспериментальным. Но в связи с постоянно растущим авиапарком у компаний возникла потребность в регулировке движения этих мастодонтов. Поэтому оказались чрезвычайно важными грамотная и своевременная дача указаний к движению воздушных бортов. У человечества к тому времени был обширный опыт регулирования наземного и водного транспорта, поэтому для разработки новых правил движения широко использовался эти наработки. Но всё равно воздух это другая стихия. И помимо стандартных решений нужны были новые.

Если на трассе между городами всё было более-менее понятно и решения имеющихся проблем нашлись довольно быстро. А вот вблизи больших городов, когда в аэропорт прилетают самолёты отовсюду, надо было принимать серьёзные меры.

Самым первым, и использующимся по сей день, стало расписание полётов. Но расписание это разграничения в рамках часов, а когда к аэропорту подлетают самолёты для посадки, нужны решения с точностью до минут, а лучше – десятков секунд. Диспетчер – человек, ответственный за такое распределение. В помощь ему создаются компьютерные программы, которые рассчитывают примерное время прибытия воздушных судов, и диспетчер, используя эти данные, может лучше регулировать входящий поток.

Постановка задачи

Разработать численную процедуру, которая будет рассчитывать возможные интервалы прибытия для каждой контрольной точки на пути следования самолёта.

Условности и входные данные

Будем считать, что самолёт это материальная точка, двигающаяся равнопеременно. В начальной точке потока временной интервал нулевой. Расстояние — Евклидово. Необходимы три входных файла, соответствующих формату:

1. Файл с данными контрольных точек
2. Файл с данными о схемах
3. Файл с данными о потоках

Работа программы

Сначала программа считывает и обрабатывает информацию из входных файлов. Затем строит граф всей воздушной зоны, на основании схем и потоков. После чего для каждого потока строит подграф, который топологически сортирует и вычисляет возможные временные интервалы прибытия в каждую точку, считая, что в начальной точке потока временной интервал – нулевой. Результат выводится в консоль.

1 Основная часть

1.1 Формат

Определение 1. *Спрямление - возможность прервать выполнение текущей схемы и уйти на заданную(ые) точки.*

Имеет следующий вид:

Str(Точка1 Точка2 ... ТочкаK) Точка1 Точка2 ... ТочкаK /Str

Str(Точка1 Точка2 ... ТочкаK) — указание на какие точки возможно спрямление, а пункты, заключённые между Str() и /Str - с каких точек возможно спрямление

Точки разделены пробелами.

Определение 2. *Схема - последовательность точек, удовлетворяющая следующим свойствам:*

1. *В одной схеме может быть только одно спрямление.*
2. *Схемы пересекаются только по начальной и конечным точкам*

Определение 3. *Начальная точка схемы - первая точка этой схемы.*

Определение 4. *Конечная(ые) точка схемы - конечными точками схемы считаются ее последняя точка, а также не принадлежащие ей точки, на которые возможно спрямление.*

Определение 5. *Стандартная схема - схема, описывающаяся тремя точками и имеющая определённый вид. Допустимый диапазон скоростей для всей схемы определяется допустимым диапазоном первой точки. Стандартную схему можно проходить не более заданного числа раз. Имеет другое название — тромбон.*

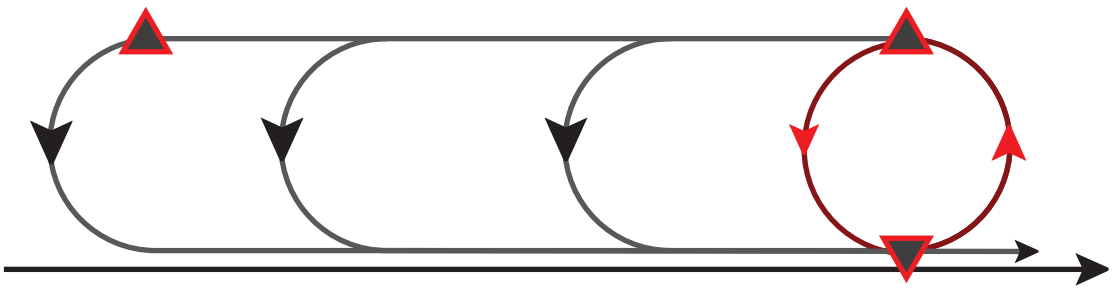

Рис. 1: Стандартная схема

Определение 6. *Поток - последовательность схем без циклов, оканчивающаяся схемой, которая содержит точку с флагом 'LAND'. Задаётся одной точкой — начальной точкой какой-либо схемы.*

1.1.1 Файл с данными точек

В первой строке указывается количество точек - число из $\mathbb{N}$

Далее перечисляются контрольные точки в соответствии следующему формату:

Имя начинается с буквы. Каждая точка в новой строке. Между элементами - пробел.

Название x y z V_{min} V_{max} флаг посадочной полосы

x, y, $z \in \mathbb{R}$, где x и y - координаты в плоскости земли, а z - вертикальная составляющая. Допускается ввод в метрах, километрах и морских милях.

V_{min}, $V_{max} \in \mathbb{R}^+$, где V_{min} - минимально допустимая скорость, а V_{max} – максимально допустимая скорость на данном контрольном пункте. Допускается ввод в метрах в секунду, километрах в час и морских милях в час.

флаг посадочной полосы - 'LAND' иначе пусто или ноль

Примеры:

- SS025 0 0 900 70 90
- RW08L -21.5 18431.5 10 65 85 LAND

1.1.2 Файл с данными об обычных и стандартных схемах

В первой строке указывается количество схем - число из $\mathbb{N}$

Во второй строке указывается количество стандартных схем - число из $\mathbb{N}$

Далее перечисляются все схемы в формате:

Название схемы - любой набор букв и цифр. Точки отделяются пробелами. После двоеточия начинать с начальной точки.

Название схемы (точка начала схемы)(точки конца схемы) : название точек

[спрямление] название точек

Формат для стандартных схем:

Название стандартной схемы (точка начала стандартной схемы)(количество повторений стандартной схемы) : точка начала точка конца разворота точка конца обратного плеча

Некоторый "синтаксический сахар":

1. В стандартной схеме можно опустить '(точка начала стандартной схемы)' и написать: Имя (количество повторений): T1 T2 T3
2. Если схема состоит только из двух точек: 'NameTwoP (A)(B): A B', то можно опустить точки после двоеточия: 'NameTwoP (A)(B):'

Примеры:

- NameA (DIPOP)(EE500 KOLOS): Str(KOLOS) DIPOP EE500 /Str
- NameB (KOLOS)(BEKAR) : KOLOS VALET Str(RODEL) EE020 EE021 EE022 EE023 EE024 EE025 /Str RODEL BEKAR
- NameC (RODEL)(RW25R): RODEL BEKAR KVOTA EE252 EE253 RW25R
- NameStScheme (GALEB)(5): GALEB N1 N2

1.1.3 Файл с данными о потоках

В первой строчке количество потоков - число из $\mathbb{N}$

Далее каждый поток в новой строке, согласно следующему правилу:

Имя потока Имя первой точки этого потока

Пример: 2

Flow1 DIPOP

Flow2 AKERA

1.2 Используемые структуры данных

Из стандартной библиотеки шаблонов – STL были взяты в качестве массивов — вектор и ассоциативный контейнер, также в алгоритмах используется стек. Для работы со строками используется класс string.

Реализованы следующие классы: координата, скорость и время с приватным конструктором и методами, позволяющими создавать экземпляры класса и возвращать значение в нужном виде. Перегрузка операторов $'+', '-', '*', '/', '\ll'$ в нужных случаях. Возведение в целую степень и извлечение квадратного корня для координаты.

Основная информация хранится в структуре *Zone*:

1. vector<vector<int≫ graph_of_descendants;
2. vector<CheckPoint> checkPoints;
3. vector<Scheme> schemes;
4. vector<Flow> flows;
5. vector<StandardScheme> standardSchemes;

Первое поле — массив идентификаторов точек $\mapsto$ идентификаторы точек, которые можно достичь из данной. Второе массив структур контрольных точек. Третье — массив структур схем. Четвёртое — массив структур потоков. Пятое — массив структур стандартных схем. Также Zone имеет различные методы для вывода информации на экран.

Для хранения контрольных точек используется следующая структура — *CheckPoint*:

1. string name;
2. Coordinate x = Coordinate::createMs(0);
3. Coordinate y = Coordinate::createMs(0);
4. Coordinate z = Coordinate::createMs(0);
5. Velocity Vmin = Velocity::createVkm_h(0);
6. Velocity Vmax = Velocity::createVkm_h(0);

7. bool landing_flag = false;

Где первое поле — это имя точки, второе, третье и четвёртые — это координаты точки. Пятое и шестое — скорости точки. И седьмое — это флаг посадочной полосы.

Для хранения схем — *Scheme*:

1. string name;
2. int start;
3. vector<int> end;
4. vector<int> path;
5. vector<int> straighteningFrom;
6. vector<int> straighteningWhere

Где первое поле — это имя схемы, второе — идентификатор точки, с которой начинается схема, третье — конечные точки схемы. Четвёртое — точки, которые образуют путь схемы, скелет. Пятое — точки, с которых можно спрямляться и шестое — точки, на которые допустимо спрямление.

Для хранения потоков – *Flow*:

1. string name;
2. int start_point;
3. map<int, vector<int≫ graph_of_descendants;
4. map<int, vector<int≫ graph_of_ancestors;
5. vector<int> keys;
6. map<int, vector<pair<Time, Time⋙ times;

Где первое поле — имя потока, второе — идентификатор точки начала потока. Третье — отображение идентификатор точки $\mapsto$ идентификаторы точек, которые можно достичь из данной. Четвёртое — отображение идентификатор точки $\mapsto$ идентификаторы точек, из которых можно достичь данной. Пятое — массив ключей значений идентификаторов точек после топологической сортировки потока. И шестое — идентификатор

точки $\mapsto$ массив пар времён, где первый элемент пары — минимально возможное время, за которое можно добраться до данной точки, а второй — максимально возможное.

Для хранения стандартных схем—*StandardScheme*:

1. string name;
2. int start;
3. int second;
4. int third;
5. int repeat;
6. Time Tmin = Time::createTsec(0);
7. Time Tmax = Time::createTsec(0);

Первое поле — имя стандартной схемы. Второе — идентификатор точки начала и конца стандартной схемы, третье — идентификатор точки начала обратного плеча стандартной схемы, четвёртое — идентификатор точки конца обратного плеча стандартной схемы. Пятое — количество повторений стандартной схемы. Шестое и седьмое поля хранят минимальное и максимальное время прохождения данной схемы соответственно.

Используемые отображения — *Maps*:

1. map<string, int> pointNameToID;
2. map<int, int> startPointIDtoStSchemeID;

Первое отображение — название точки $\mapsto$ идентификатор точки. Второе — отображение идентификатор точки начала стандартной схемы $\mapsto$ идентификатор стандартной схемы.

1.3 Описание алгоритмов работы программы

1.3.1 Чтение данных — основная идея

Чтение происходит в три этапа. Сначала считываются точки, потом схемы, затем потоки. Каждая читается с помощью регулярных выражений в соответствии с форматом.

Основная идея одинакова для всех трёх файлов. Процедура, соответствующая файлу, получает на вход путь к нему и массив структур, который требуется заполнить. Далее объявляется счётчик, который отвечает за движение по массиву. Затем делается попытка открыть файл. В случае успеха выводится сообщение об открытии файла, иначе выводится сообщение об ошибке, и программа останавливает свою работу. Если файл был открыт, то первым делом происходит считывание длины массива и проверка на не отрицательность этого числа. В случае, если полученная длина массива не положительна, то происходит остановка программы. Иначе длина массива становится равной полученному числу. Далее объявляется регулярное выражение для работы с данными из файла.

Начинается основной цикл, в котором происходит заполнение массива. Файл читается построчно и построчно обрабатывается регулярным выражением до тех пор, пока не закончится файл. Первым делом делается проверка на успешность сопоставления регулярного выражения и строки. В случае неуспеха выводится сообщение о несоответствии формату. Также проверяется уникальность имён входных данных. Далее делается проверка на возможность доступа к текущей координате массива. В случае невозможности осуществить это программа заканчивает работу и предупреждает о нехватке выделенного места под элементы массива. Затем идёт заполнение, подробности которого можно найти ниже. И, наконец, после успешного наполнения массива, делается проверка и правка, чтобы действительная длина массива совпадала с декларируемой, если это не так то производится коррекция, после чего файл закрывается.

1.3.2 Чтение точек

Напомним содержимое структуры *checkPoint*. Она состоит из семи полей:

1. Имя точки
2. Координата x

3. Координата y

4. Координата z

5. Минимальная скорость Vmin

6. Максимальная скорость Vmax

7. Флаг посадочной полосы

Согласно формату, в таком же порядке данные находятся в файле. Теперь рассмотрим регулярное выражение:

```
([a-z,A-Z]\w*)\s+([-+]?[0-9]*\.?[0-9]+)\s+([-+]?[0-9]*\.?[0-9]+)\s+
```

```
([-+]?[0-9]*\.?[0-9]+)\s+([0-9]*\.?[0-9]+)\s+([0-9]*\.?[0-9]+)\s*(LAND|0)?\s*
```

Первая группа захвата отвечает за имя точки. Согласно формату, имя начинается с буквы. Далее может быть любые комбинации букв и цифр. Вторая, третья и четвёртая группы захвата одинаковые и предназначены для распознания действительного числа. Пятая и шестая почти аналогичны предыдущим трём, но они захватывают не все действительные числа, а только неотрицательные. И, наконец, седьмая ищет совпадение с флагом посадки LAND, нулём либо же отсутствует вовсе. Однако, одна и только одна точка с таким флагом должна быть. За корректность этого отвечает счётчик «k».

Далее происходит заполнение структуры *CheckPoints* поэлементно, в зависимости от исходных данных. Поддерживается ввод в следующих единицах измерения:

- Координата в метрах, в километрах и в морских милях
- Скорость в метрах в секунду, километрах в час и морских милях в час

Флаг посадки по умолчанию $false$. $True$ присваивается, если седьмая группа захвата содержит LAND. И увеличивается счётчик «k». Если он превосходит единицу— программа останавливается и предупреждает об ошибке, что посадочных точек больше одной. Потом происходит заполнение отображения Имя точки $\mapsto$ идентификатор точки. Затем происходит проверка корректности введённых скоростей, чтобы максимальная скорость не была меньше минимальной. И увеличивается счётчик «i». Наконец, делается финальная проверка, что если «k» остался равен нулю, что означает отсутствие точки посадки, тогда программа завершает работу и предупреждает об ошибке.

1.3.3 Чтение схем

В файле со схемами лежит информация об обычных схемах и стандартных схемах. Начнём описание с обычных.

Напомним поля структуры *scheme*:

1. Имя схемы
2. Идентификатор точки начала схемы
3. Массив точек, которыми схема оканчивается
4. Массив точек, составляющие основной путь схемы
5. Массив точек, откуда можно спрямляться
6. Массив точек, куда можно спрямляться

Рассмотрим регулярное выражение и те части, которые относятся к обычной схеме:

```
(\w+)\s*(?:\((\w+)\))?\s*\(([\w\s]*|\d+)\)\s*:
```

```
(?:\s*([\w\s]*)\s+(?:Str\(([\w\s]+)\)\s*([\w\s]+)\/Str)?\s*([\w\s]*)?)?
```

Первая группа захвата — это имя схемы. Представляет из себя любой набор букв и цифр. Вторая группа — это имя точки, с которой начинается схема. Третья — это имена точек, которой оканчивается схема.

Далее после двоеточия следует основное тело схемы. Четвертая группа — это точки схемы, предшествующие спрямлению. Если оно отсутствует, то эта группа представляет из себя путь. Однако, в связи с тем, что после этой группы обязательно должен быть хотя бы один пробел, и если после последней точки схемы пробела не будет, то эта точка не попадёт в эту группу захвата. Её нужно искать в седьмой.

Пятая группа — это точки, на которые можно спрямляться с точек из шестой группы. Если спрямление в схеме отсутствует, то эти группы пустые. Седьмая группа — это точки, идущие после спрямления. Или, если нет спрямления в схеме и нет пробела после последней точки, тогда тут находится та самая последняя точка.

Также для удобства записи схем состоящих из двух точек, было введено упрощение записи. Можно вводить только имя схемы, точку начала, точку конца и двоеточие. Что регулируется большой не захватывающей скобкой после двоеточия с вопросительным знаком на окончании.

Если спрямление присутствует, то путь состоит из четвёртой, шестой и седьмой групп захвата.

Заполнение. В общем случае первая группа сразу же присваивается в первое поле структуры. Второе поле – это результат отображения Имя точки $\mapsto$ идентификатор точки, где в качестве имени выступает вторая группа захвата.

Далее необходимо заполнить массив конечных точек. Для этого используется вспомогательная функция, которая получает строку на вход и массив, который требуется заполнить. Строка читается поэлементно, и с помощью отображения Имя точки $\mapsto$ идентификатор точки заполняет данный массив. В случае, если точки не оказалось среди *checkPoints*, тогда выдаётся ошибка и программа заканчивает свою работу с предупреждением о том, что в такой-то строке такая-та точка не найдена среди контрольных точек.

Затем абсолютно аналогично заполняются поля пять и шесть. Теперь необходимо собрать путь схемы. Для этого используя ту же вспомогательную функцию с четвёртой группой и полем путь, мы собираем точки до спрямления. Затем записываем в путь все элементы из поля пять, и, наконец, используя вспомогательную функцию с седьмой группой и полем путь, мы собираем точки после спрямления.

Если четвёртая и пятая группы пустые, то это означает, что мы имеем дело со схемой, состоящей из двух точек. В этом случае, имя и точка начала схемы заполняются как в общем случае, но также нам известно, что массив конечных точек состоит из одного элемента, поэтому мы можем его сразу же записывать в массив конечных точек. И тогда путь, также легко определяется, как точка начала схемы плюс первый, он же и единственный, элемент массива конечных точек.

После чего увеличивается счётчик «i».

Теперь опишем работу со ***стандартными схемами***.

Напомним, что стандартной схемой называется схема типа «тромбон». Она определяется тремя точками и количеством повторений, что и отражено в полях структуры *StandardScheme*:

1. Имя схемы
2. Идентификатор точки начала и конца стандартной схемы
3. Идентификатор точки начала обратного плеча стандартной схемы
4. Идентификатор точки конца обратного плеча стандартной схемы

5. Количество повторений стандартной схемы

6. Минимальное время, за которое можно пройти схему

7. Максимальное время, за которое можно пройти схему

Поля шесть, семь вычисляются в дальнейшем.

Рассмотрим регулярное выражение и те части, которые относятся к стандартной схеме:

```
(\w+)\s*(?:\((\w+)\))?\s*\(([\w\s]*|\d+)\)\s*:
```

```
(?:\s*([\w\s]*)\s+(?:Str\(([\w\s]+)\)\s*([\w\s]+)\/Str)?\s*([\w\s]*)?)?
```

Первая группа захвата — это имя стандартной схемы. Так как начало и конец совпадают, то вторая группа захвата это второе поле структуры. Третья группа — это количество повторений схемы.

В стандартной схеме нет спрямлений, поэтому пятая и шестая группы всегда пустые. Четвёртая и седьмая вместе дают все три точки. Поэтому с помощью вспомогательной функции из обычных схем, заполняем временный массив *path*, аналогичный одноименному полю из обычных схем. И, затем, заполняем второе, третье и четвёртые поля структуры стандартной схемы как — первый, второй и третий элемент массива *path*. Если же длина этого массива получилась больше трёх, то программа завершает работу и выдаёт ошибку.

Далее заполняется отображение идентификатор точки начала стандартной схемы $\mapsto$ идентификатор стандартной схемы.

После чего увеличивается счётчик «k».

1.3.4 Построение графа зоны

Сначала заполняем поля времён у стандартных схем. Минимальное время — пролёт по окружности с максимальной скоростью, а максимальное — пролёт по всей схеме с минимальной скоростью.

Обозначим точки стандартной схемы $\nu_0 = (x_0, y_0, z_0),\ \nu_1 = (x_1, y_1, z_1),\ \nu_2 = (x_2, y_2, z_2)$ Вычислим радиус поворота, как половину от расстояния между ν_0 и ν_1:

$$R = \frac{1}{2} * \sqrt{(x_1 - x_0)^2 + (y_1 - y_0)^2 + (z_1 - z_0)^2}$$

Длину плеча найдём, как расстояние между ν_1 и ν_2:

$$S = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2}$$

Тогда минимальное и максимальное времена выражаются следующим образом:

$$T_{min} = \frac{2\pi R}{v_{max}}, \quad T_{max} = \frac{2(S + \pi R)}{v_{min}}$$

где v_{max} и v_{min} — это максимально и минимально возможные скорости первой точки «тромбона».

Граф зоны будем строить списками следующих. То есть соединяем ребрами данную вершину, со всеми вершинами, которые имеют связь с данной. Делать это будем следующим образом: идём по массиву схем, обрабатывая каждую схему в отдельности. На данном этапе нас интересует поле *path*. Так как в нём лежат вершины в порядке прохождения самолётом схемы, то мы будем соединять текущую точку со следующей. Далее нужно соединить вершины, с которых возможно спрямление со всеми вершинами на которые это спрямление доступно. После чего переходить к другой схеме. Но есть нюанс. Если из предпоследней точки схемы можно спрямиться на конечную, то возникает проблема с добавлением этого ребра дважды — первый раз как элемент пути, второй как элемент спрямления. Это необходимо учесть.

1.3.5 Построение потока

Начальной точке потока выставляем $T_{min} = T_{max} = 0$. Теперь построим граф потока, как подграф графа зоны, также списками потомков. Для этого воспользуемся поиском в глубину. Заведём стек, на который будем складывать не посещённые вершины и массив меток, в котором будем отмечать посещённые вершины. Будем работать до тех пор, пока стек не опустеет. Сначала складываем на него точку начала потока. Используя граф зоны, соединяем эту точку со всеми сыновьями, и если сын ещё не посещён, то добавляем его на стек и ставим метку, что его посетили. В результате получим граф потока. После собираем граф потока, заданный списками предшественников, путём перебора графа, построенного списками потомков.

1.3.6 Топологическая сортировка

Данный ориентированный граф не имеет циклов, поэтому его можно топологически отсортировать, то есть сделать так, чтобы из вершин с меньшими номерами дуги шли

в вершины с большими номерами. Для этой цели есть у потока поле *keys*. Это массив, хранящий идентификатор в топологически отсортированном порядке $\mapsto$ идентификатор в исходном порядке. Сортировка проводится методом полустепеней захода. Заведём переменную *number*, которая будет помогать отсортировывать вершины, инициализируем её нулём. Сначала сделаем размер массива ключей равным количеству вершин в потоке. Затем инициализируем отображение Точка $\mapsto$ полу степень захода следующим образом всем вершинам сопоставляется ноль. Заполняем его так: идя по спискам потомков, увеличиваем значение на единицу за каждого предка у вершины. Заводим стек и складываем на него все вершины с нулевой полустепенью захода. Пока стек не пуст, берём вершину со стека и на место number в массиве записываем идентификатор вершины. Увеличиваем *number* на единицу. И для всех потомков данной вершины уменьшаем на единицу полустепень захода, и, если она стала равна нулю — кладём этого потомка на стек. В результате получаем нужный массив.

1.3.7 Объединение времён

На вход дается массив пар действительных чисел. Необходимо объединить вложенные пары в одну, две пересекающиеся в одну. Будем делать с помощью отображения число$\mapsto$ действие. Для корректного сравнения действительных чисел реализованы два компаратора для проверки что одно число больше или меньше другого с точностью ϵ, заданной в классе времени (по умолчанию $\epsilon = 10^{-4}$). Сначала проверим, что пары заданы корректно, то есть первый элемент меньше либо равен второго, иначе программа завершает работу и выдаёт сообщение об ошибке. Затем инициализируем нулевыми значениями отображение и заполняем его, если элемент стоит на первом месте в паре, тогда его значение увеличиваем на единицу, если на втором, то уменьшаем на единицу. Очистим массив времён. Заведём переменную, в которой будем считать сумму, инициализировав нулём. Вместе с тем, запомним первый элемент и назовём его начальным. Идём до конца ассоциативного массива, суммируя с накоплением значения отображения. Если в какой-то момент сумма стала равна нулю, то добавляем пару, состоящую из первого элемента, который мы запомнили, и текущего элемента. Проверяем, если мы не дошли до конца ассоциативного массива, то следующий элемент объявляем начальным и продолжаем цикл. В результате мы объединили вложенные и пересекающиеся интервалы, остались лишь не пересекающиеся.

1.3.8 Расчёт времён

Будем рассчитывать время для каждого потока по отдельности.

Так как поток топологически отсортирован, мы будем двигаться по возрастанию идентификатора вершин до тех пор, пока не дойдём до вершины с максимальным индексом, рассчитывая возможное время прибытия в следующие точки из данной. Сразу же выполним объединение времён, так как с прошлого шага мы получили набор интервалов, не обязательно не пересекающихся.

Требует отдельного рассмотрения стандартная схема. Если в данная точка является началом стандартной схемы, у которой остались повторения, то нужно к текущим временным интервалам данной точки добавить интервалы, полученные прибавлением T_{min} и T_{max} из полей стандартной схемы к каждому из имеющихся интервалов данной точки, и уменьшить количество повторений этой стандартной схемы на единицу.

Если же стандартной схемы нет или она исчерпала себя, то тогда для всех потомков время вычисляется как равнопеременное:

$$T_{min} = \frac{2S}{v^0_{max} + v^1_{max}}$$

где v^0_{max} — это максимально допустимая скорость на текущей контрольной точке, а v^1_{max} — это максимально допустимая скорость на данном потомке текущей контрольной точке

$$T_{max} = \frac{2S}{v^0_{min} + v^1_{min}}$$

где v^0_{min} — это минимально допустимая скорость на текущей контрольной точке, а v^1_{min} — это минимально допустимая скорость на данном потомке текущей контрольной точке

S — обычное Евклидово расстояние:

$$S = \sqrt{(x_1 - x_0)^2 + (y_1 - y_0)^2 + (z_1 - z_0)^2)}$$

После чего добавляем к каждому интервалу полученные T_{min} и T_{max}, и этот новый интервал записываем в массив времён потомка. И увеличиваем счётчик на единицу. После выполнения цикла получаем набор временных интервалов для каждой точки из потока.

Выполняя данную процедуру для каждого из потоков получим набор временных интервалов для каждой точки.

После чего выводим на экран.

Заключение

В результате работы была решена поставленная задача о вычислении временных интервалов прибытия воздушного судна на контрольные точки. Процесс работы состоял в изучении задачи, выборе структур данных и алгоритмов. Реализации на языке C++ 14-го стандарта. В качестве примера была рассмотрена схема аэропорта «Кольцово» в городе Екатеринбурге.

Список использованных источников и литературы

Список литературы

[1] Спиридонов А.А. Кумков С.С. Разработка и исследование формализаций задачи бесконфликтного слияния потоков воздушных судов. Отчёт о научно-экспериментальной работе «Алгоритмы и программное обеспечение обработки информации в АС УВД» по Договору №29-19У. (Этап 1). Том 3, июнь 2019 г., 25 стр.

[2] Проект структуры воздушного пространства МУДР. Стандартные маршруты прибытия на аэродром Шереметьево. Версия 03 ПСВП МУДР 8.5.1 (таблицы). ГосНИИГА, Москва, 2016.

[3] Проект структуры воздушного пространства МУДР. Схемы захода на посадку на аэродром Шереметьево. Версия 03 ПСПВ. Стандартные маршруты прибытия на аэродром Шереметьево. Версия 04 ПСВП МУДР 8.5.1 (схемы). ГосНИИГА, Москва, 2016.

[4] Point Merge Integration of Arrival Flows Enabling Extensive RNAV Application and Continuous Descent. Operation Services and Environment Definition. Report, July 2010. Eurocontrol Experimental Center, Bretigny-sur-Orge.
`http://www.eurocontrol.int/eec/gallery/content/public..`

[5] Air Traffic Management Technology Demonstration–1 (ATD–1). NASA Report FS-2011-10-01-ARC. 2011.

[6] Пятко С.Г., Красов А.И. и др. Автоматизированные системы управления воздушным движением. Новые информационные технологии в авиации. Санкт-Петербург; Изд. Политехника, 2004.

[7] Королев Е. Н. Технологии работы диспетчеров управления воздушным движением. -— Москва: Воздушный транспорт, 2000.

[8] ГОСТ 20058-80. Динамика летательных аппаратов в атмосфере. Термины, определения и обозначения. —Москва: Госстандарт, 1980.