ZAV-UT

Készítette Doxygen 1.12.0

1. fejezet

zav-utazastervezo

Zedországi Államvasutak utazástervező rendszere

Házifeladat projekt

Menetrendek alapján leggyorsabb útvonalat keres két állomás között

2. fejezet

Hierarchikus mutató

2.1. Osztályhierarchia

Majdnem (de nem teljesen) betűrendbe szedett leszármazási lista:

3. fejezet

Osztálymutató

3.1. Osztálylista

Az összes osztály, struktúra, unió és interfész listája rövid leírásokkal:

4. fejezet

Fájlmutató

4.1. Fájllista

Az összes dokumentált fájl listája rövid leírásokkal:

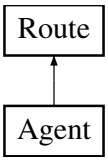
5. fejezet

Osztályok dokumentációja

5.1. Agent osztályreferencia

`#include <agent.h>`

Az Agent osztály származási diagramja:



Publikus tagfüggvények

- Agent (Edge &edge, const Node &start, const Node &target, Time startTime)
- Agent (const Agent &, Edge *firstEdge=NULL)
- AgentState step ()
- const Node * head () const

Publikus tagfüggvények a(z) Route osztályból származnak

- Route (Array< Edge * > edges, Time startTime)
- Array< Edge * > getEdges () const
- Time getStartTime () const
- size_t getTotalWeight () const

További örökölt tagok

Védett attribútumok a(z) Route osztályból származnak

- Array< Edge * > edges

5.1.1. Részletes leírás

Egy grafban elore halado agent

5.1.2. Konstruktorok és destruktorok dokumentációja

5.1.2.1. Agent() [1/2]

```
Agent::Agent (
            Edge & edge,
            const Node & start,
            const Node & target,
            Time startTime)
```

Letrehoz egy agentet az adott elen

Paraméterek

edge	az el amin eloszor vegig megy az agent
start	a kezdo node
target	a cel node
startTime	az indulas ideje

5.1.2.2. Agent() [2/2]

```
Agent::Agent (
            const Agent & other,
            Edge * firstEdge = NULL)
```

Letrehoz egy agentet egy masik agentbol

Paraméterek

firstEdge	ha megadva lecsereli az utolso elet erre az elre

5.1.3. Tagfüggvények dokumentációja

5.1.3.1. head()

```
const Node * Agent::head () const
```

Visszatérési érték

A legutobbi el altal mutatott csucs, az agent feje ha egy kigyokent nezzuk

5.1.3.2. step()

```
AgentState Agent::step ()
```

Egyszer lepteti az agentet

Visszatérési érték

az agent uj allapota

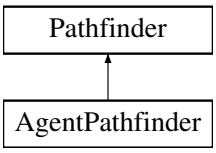
Ez a dokumentáció az osztályról a következő fájlok alapján készült:

- agent.h
- agent.cpp

5.2. AgentPathfinder osztályreferencia

```
#include <agent.h>
```

Az AgentPathfinder osztály származási diagramja:



Publikus tagfüggvények

- AgentPathfinder (Graph graph, size_t numRoutes=3)
- SortedList< Route * > getRoutes (const Node &from, const Node &to, Time starTime)

Publikus tagfüggvények a(z) Pathfinder osztályból származnak

- Pathfinder (Graph graph, size_t numRoutes=3)

További örökölt tagok

Védett attribútumok a(z) Pathfinder osztályból származnak

- Graph graph
- size_t numRoutes

5.2.1. Részletes leírás

Agenteket hasznalo utvonaltervezo

5.2.2. Konstruktorok és destruktorok dokumentációja

5.2.2.1. AgentPathfinder()

```
AgentPathfinder::AgentPathfinder (
            Graph graph,
            size_t numRoutes = 3)
```

see Pathfinder::Pathfinder()

5.2.3. Tagfüggvények dokumentációja

5.2.3.1. getRoutes()

```
SortedList< Route * > AgentPathfinder::getRoutes (
            const Node & from,
            const Node & to,
            Time starTime)  [virtual]
```

/see Pathfinder::getRoutes()

Visszatérési érték

a visszaadott listaban levo utvonalak dinamikusan taroltak es a hivo kezeli

Kivételek

NotFound	ha to vagy from nincs a grafban

Megvalósítja a következőket: Pathfinder.

Ez a dokumentáció az osztályról a következő fájlok alapján készült:

- agent.h
- agent.cpp

5.3. alwaysFirst< T > osztálysablon-referencia

Publikus tagfüggvények

- bool **operator()** (T t1, T t2)

Ez a dokumentáció az osztályról a következő fájl alapján készült:

- sorted_list.hpp

5.4. Array< T > osztálysablon-referencia

```
#include <array.hpp>
```

Publikus tagfüggvények

- Array (size_t length, const T *arr=nullptr)
- Array (const Array &other, size_t cap=0)
- size_t getLength () const
- T last () const
- Array & operator= (const Array &other)
- T operator[ ] (size_t i) const
- T * operator+ (size_t i) const
- void operator+= (T element)

5.4.1. Részletes leírás

template<typename T>
class Array< T >

Egy dinamikusan tarolt tomb

Sablon paraméterek

T	A tipus amit tarol

5.4.2. Konstruktorok és destruktorok dokumentációja

5.4.2.1. Array() [1/2]

```
template<typename T >
Array< T >::Array (
            size_t length,
            const T * arr = nullptr)  [inline]
```

Letrehoz egy Array-t egy tombbol amit lemasol

Paraméterek

length	a tomb hossza
arr	a tombre mutato pointer

5.4.2.2. Array() [2/2]

```
template<typename T >
Array< T >::Array (
            const Array< T > & other,
            size_t cap = 0)  [inline]
```

Letrehoz egy Array-t egy masikbol, lemasolja az elemeit

Paraméterek

other	a masik Array
cap	a dinamikusan foglalando tomb merete, ha nincs megadva a masik Array hossza

5.4.3. Tagfüggvények dokumentációja

5.4.3.1. getLength()

```
template<typename T >
size_t Array< T >::getLength () const  [inline]
```

Visszatérési érték

A tomb elemeinek szama

5.4.3.2. last()

```
template<typename T >
T Array< T >::last () const  [inline]
```

Visszatérési érték

A tomb utolso eleme

Kivételek

std::out_of_range	ha ures a tomb

5.4.3.3. operator+()

```
template<typename T >
T * Array< T >::operator+ (
            size_t i) const  [inline]
```

Hasznos ha egy Array<char>-bol c-stringet szeretnel csinalni

Paraméterek

i	index

Visszatérési érték

a tomb i-edik elemere mutato pointer

Kivételek

std::out_of_range	ha i nagyobb vagy egyenlo mint a tomb elemeinek szama

5.4.3.4. operator+=()

```
template<typename T >
void Array< T >::operator+= (
            T element)  [inline]
```

Hozzafuz a tomb vegere egy elemet ujra foglal ha nem ferne el

Paraméterek

element	a hozzadando elem

5.4.3.5. operator=()

```
template<typename T >
Array & Array< T >::operator= (
            const Array< T > & other)  [inline]
```

Lemasolja a jobbertekkent adott Array-t

Paraméterek

other	a masolando Array

Visszatérési érték

ez az Array

5.4.3.6. operator[]()

```
template<typename T >
T Array< T >::operator[] (
            size_t i) const  [inline]
```

Paraméterek

i	index

Visszatérési érték

a tomb i-edik eleme

Kivételek

std::out_of_range	ha i nagyobb vagy egyenlo mint a tomb elemeinek szama

Ez a dokumentáció az osztályról a következő fájl alapján készült:

- array.hpp

5.5. ascending< T > osztálysablon-referencia

Publikus tagfüggvények

- bool **operator()** (T t1, T t2)

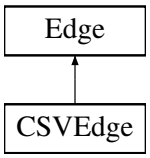
Ez a dokumentáció az osztályról a következő fájl alapján készült:

- sorted_list.hpp

5.6. CSVEdge osztályreferencia

`#include <csvgraph.h>`

A CSVEdge osztály származási diagramja:



Publikus tagfüggvények

- **CSVEdge** (int weight, Array< Time > startTimes, const char *name)
- void operator+= (CSVNode &node)

Publikus tagfüggvények a(z) Edge osztályból származnak

- Edge (Node *from, Node *to, int weight, Array< Time > startTimes, const char *name)
- const char * getName () const
- size_t getWeight (Time currentTime) const
- size_t getWeight () const
- Time getFirstStartTimeAfter (Time time) const
- Node * getToNode () const
- Edge & operator= (const Edge &other)

További örökölt tagok

Védett attribútumok a(z) Edge osztályból származnak

- Node * from
- Node * to
- size_t weight
- Array< Time > startTimes
- char * name

5.6.1. Részletes leírás

.csv fajlbol letrehozott Edge /see Edge

5.6.2. Tagfüggvények dokumentációja

5.6.2.1. operator+=()

```
void CSVEdge::operator+= (
            CSVNode & node)
```

Hozzafuz az elhez egy csucsot

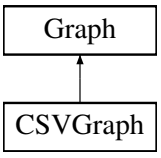
Ez a dokumentáció az osztályról a következő fájlok alapján készült:

- csvgraph.h
- csvgraph.cpp

5.7. CSVGraph osztályreferencia

```
#include <csvgraph.h>
```

A CSVGraph osztály származási diagramja:



Publikus tagfüggvények

- CSVGraph (CSVParser &csv)
- void operator+= (CSVNode &node)

Publikus tagfüggvények a(z) Graph osztályból származnak

- Graph ()
- Graph (Array< Node * > nodes)
- Array< Node * > getNodes () const
- Node * getNode (const char *name, bool exactMatch=false) const

További örökölt tagok

Védett attribútumok a(z) Graph osztályból származnak

- Array< Node * > nodes

5.7.1. Részletes leírás

.csv fajlbol letrehozott Graph /see Graph

5.7.2. Konstruktorok és destruktorok dokumentációja

5.7.2.1. CSVGraph()

```
CSVGraph::CSVGraph (
            CSVParser & csv)
```

Letrehoz egy grafot egy vagy tobb .csv fajlbol

Paraméterek

csv	a .csv fajl(ok)at kezelo parser

5.7.3. Tagfüggvények dokumentációja

5.7.3.1. operator+=()

```
void CSVGraph::operator+= (
            CSVNode & node)
```

Hozzafuz a grafhoz egy csucsot

Ez a dokumentáció az osztályról a következő fájlok alapján készült:

- csvgraph.h
- csvgraph.cpp

5.8. CSVLine osztályreferencia

`#include <csvparser.h>`

Publikus tagfüggvények

- CSVLine ()
- CSVLine (const char line[ ], char separator=',')
- Array< Array< char > > getColumns () const
- bool isEmpty () const
- CSVLine & **operator=** (const CSVLine &)
- void operator+= (const char *str)
- void operator+= (size_t num)

5.8.1. Részletes leírás

Egy .csv fajlban levo sor

5.8.2. Konstruktorok és destruktorok dokumentációja

5.8.2.1. CSVLine() [1/2]

```
CSVLine::CSVLine ()
```

Letrehoz egy ures sort

5.8.2.2. CSVLine() [2/2]

```
CSVLine::CSVLine (
            const char line[],
            char separator = ',')
```

Egy c-stringbol letrehoz egy sort

Paraméterek

line	c-string
separator	az oszlop elvalasztasara szolgalo karakter

5.8.3. Tagfüggvények dokumentációja

5.8.3.1. getColumns()

```
Array< Array< char > > CSVLine::getColumns () const
```

Visszatérési érték

sor oszlopai

5.8.3.2. isEmpty()

```
bool CSVLine::isEmpty () const
```

Visszatérési érték

a sor ures-e

5.8.3.3. operator+=() [1/2]

```
void CSVLine::operator+= (
            const char * str)
```

hozzad egy c-stringet a sorhoz

Paraméterek

sor	c-string

5.8.3.4. operator+=() [2/2]

```
void CSVLine::operator+= (
            size_t num)
```

hozzad egy szamot a sorhoz

Paraméterek

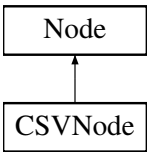
num	szam

Ez a dokumentáció az osztályról a következő fájlok alapján készült:

- csvparser.h
- csvparser.cpp

5.9. CSVNode osztályreferencia

A CSVNode osztály származási diagramja:



Publikus tagfüggvények

- **CSVNode** (const char *name)
- void operator+= (CSVEdge &edge)

Publikus tagfüggvények a(z) Node osztályból származnak

- Node (Array< Edge * > edges, const char *name)
- Node ()
- const char * getName () const
- Array< Edge * > getEdges () const
- Node & operator= (const Node &other)

További örökölt tagok

Védett attribútumok a(z) Node osztályból származnak

- Array< Edge * > edges
- char * name

5.9.1. Tagfüggvények dokumentációja

5.9.1.1. operator+=()

```
void CSVNode::operator+= (
            CSVEdge & edge)
```

Hozzafuz a csucshoz egy elt

Ez a dokumentáció az osztályról a következő fájlok alapján készült:

- csvgraph.h
- csvgraph.cpp

5.10. CSVParser osztályreferencia

```
#include <csvparser.h>
```

Publikus tagfüggvények

- CSVParser (const char *filePath)
- void write (CSVLine line)
- CSVLine read ()
- char * getFileName () const
- void operator+= (CSVParser &parser)

5.10.1. Részletes leírás

Egy vagy tobb .csv fajl beolvasasaert felelos osztaly a beolvasas nem letrehozaskor tortenik hanem folyamatosan ahogy az osztalybol kerik

5.10.2. Konstruktorok és destruktorok dokumentációja

5.10.2.1. CSVParser()

```
CSVParser::CSVParser (
            const char * filePath)
```

Letrehoz egy parsert egy fajl eleresi utvonalbol

Paraméterek

filePath	fajl eleresi utvonal c-stringként

5.10.3. Tagfüggvények dokumentációja

5.10.3.1. getFileName()

```
char * CSVParser::getFileName () const
```

Megadja a jelenleg nyitott fajl eleresi utjat

Visszatérési érték

5.10.3.2. operator+=()

```
void CSVParser::operator+= (
            CSVParser & parser)
```

Kombinal ket parsert a jobb oldali parsernek dinamikusan taroltnak kell lennie es ez utan a bal oldali parser fogja kezelni ezutan a jobb oldali parsert nem szabad hasznalni

Paraméterek

parser	

5.10.3.3. read()

```
CSVLine CSVParser::read ()
```

olvas a .csv fajlbol

Visszatérési érték

a sor amit olvas a fajlbol

5.10.3.4. write()

```
void CSVParser::write (
            CSVLine line)
```

Ir a .csv fajlba

Paraméterek

line	sor amit ir a fajlba

Ez a dokumentáció az osztályról a következő fájlok alapján készült:

- csvparser.h
- csvparser.cpp

5.11. descending< T > osztálysablon-referencia

Publikus tagfüggvények

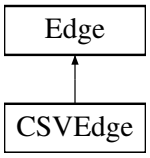
- bool **operator()** (T t1, T t2)

Ez a dokumentáció az osztályról a következő fájl alapján készült:

- sorted_list.hpp

5.12. Edge osztályreferencia

Az Edge osztály származási diagramja:



Publikus tagfüggvények

- Edge (Node *from, Node *to, int weight, Array< Time > startTimes, const char *name)
- const char * getName () const
- size_t getWeight (Time currentTime) const
- size_t getWeight () const
- Time getFirstStartTimeAfter (Time time) const
- Node * getToNode () const
- Edge & operator= (const Edge &other)

Védett attribútumok

- Node * from
- Node * to
- size_t weight
- Array< Time > startTimes
- char * name

5.12.1. Konstruktorok és destruktorok dokumentációja

5.12.1.1. Edge()

```
Edge::Edge (
            Node * from,
            Node * to,
            int weight,
            Array< Time > startTimes,
            const char * name)
```

Letrehoz egy elt

Paraméterek

from	az el ebbol a csucsbol indul
to	az el ebbe a csucsba megy
weight	az el sulya
startTimes	az el indulasi ideje
name	az el neve

5.12.2. Tagfüggvények dokumentációja

5.12.2.1. getFirstStartTimeAfter()

```
Time Edge::getFirstStartTimeAfter (
            Time time) const
```

Megadja az elso indulasi idot egy adott ido utan

Paraméterek

time	az ido ami utan keres

Visszatérési érték

a legkorabbi indulasi ido

5.12.2.2. getName()

```
const char * Edge::getName () const
```

Visszatérési érték

az el neve c-stringkent

5.12.2.3. getToNode()

```
Node * Edge::getToNode () const
```

Visszatérési érték

az csucs ahova az el megy

5.12.2.4. getWeight() [1/2]

```
size_t Edge::getWeight () const
```

Visszatérési érték

az el sulya

5.12.2.5. getWeight() [2/2]

```
size_t Edge::getWeight (
            Time currentTime) const
```

Megadja az el sulyat es az indulasaig vart idot

Paraméterek

currentTime	az ido amitol var

Visszatérési érték

a suly es vart ido osszege percben

5.12.2.6. operator=()

```
Edge & Edge::operator= (
            const Edge & other)
```

Letrehoz egy elt egy masik masolatakent

Paraméterek

other	a jobb ertek

Visszatérési érték

ez az el

5.12.3. Adattagok dokumentációja

5.12.3.1. from

```
Node* Edge::from  [protected]
```

A csucs amibol indul az el

5.12.3.2. name

```
char* Edge::name  [protected]
```

Az el neve

5.12.3.3. startTimes

```
Array<Time> Edge::startTimes  [protected]
```

Az el indulasi idejei Tobb van mert egy el tobszor is indulhat egy nap

5.12.3.4. to

```
Node* Edge::to  [protected]
```

A csucs amibe megy az el

5.12.3.5. weight

```
size_t Edge::weight  [protected]
```

A csucs sulya

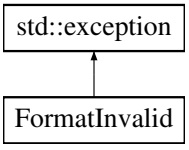
Ez a dokumentáció az osztályról a következő fájlok alapján készült:

- graph.h
- graph.cpp

5.13. FormatInvalid osztályreferencia

```
#include <csvparser.h>
```

A FormatInvalid osztály származási diagramja:



Publikus tagfüggvények

- FormatInvalid (const char file[ ]=NULL, size_t line=0, size_t character=0)
- virtual const char ∗ **what** () const throw ()

5.13.1. Részletes leírás

Hibas formatum kivetel

5.13.2. Konstruktorok és destruktorok dokumentációja

5.13.2.1. FormatInvalid()

```
FormatInvalid::FormatInvalid (
            const char file[] = NULL,
            size_t line = 0,
            size_t character = 0)
```

Paraméterek

file	a fajl amiben a hiba van
line	a sor ahol a hiba van
character	a karakter ami a hibat okozta

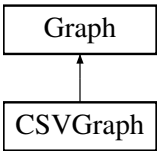
Ez a dokumentáció az osztályról a következő fájlok alapján készült:

- csvparser.h
- csvparser.cpp

5.14. Graph osztályreferencia

```
#include <graph.h>
```

A Graph osztály származási diagramja:



Publikus tagfüggvények

- Graph ()
- Graph (Array< Node ∗ > nodes)
- Array< Node ∗ > getNodes () const
- Node ∗ getNode (const char ∗name, bool exactMatch=false) const

Védett attribútumok

- Array< Node * > nodes

5.14.1. Részletes leírás

Utvonaltervezesben hasznalt graf

5.14.2. Konstruktorok és destruktorok dokumentációja

5.14.2.1. Graph() [1/2]

```
Graph::Graph ()
```

letrehoz egy ures grafot

5.14.2.2. Graph() [2/2]

```
Graph::Graph (
            Array< Node * > nodes)
```

Letrehoz egy grafot

Paraméterek

nodes	a graf csucsai

5.14.3. Tagfüggvények dokumentációja

5.14.3.1. getNode()

```
Node * Graph::getNode (
            const char * name,
            bool exactMatch = false) const
```

Megkeresi a graf egyik csucsat nev alapjan alapbol nem teljes egyezes kell, de a megadott stringnek benne kell lennie a nevben

Paraméterek

name	a keresett nev c-stringkent
exactMatch	ha igaz akkor teljes egyezes kell

Visszatérési érték

a csucs aminek ez a neve

Kivételek

NotFound	ha nem talalja a csucsot

5.14.3.2. getNodes()

```
Array< Node * > Graph::getNodes () const
```

Visszatérési érték

A graf csucsai

5.14.4. Adattagok dokumentációja

5.14.4.1. nodes

```
Array<Node*> Graph::nodes  [protected]
```

a graf elei

Ez a dokumentáció az osztályról a következő fájlok alapján készült:

- graph.h
- graph.cpp

5.15. SortedList< T, compare >::Iterator osztályreferencia

```
#include <sorted_list.hpp>
```

Publikus tagfüggvények

- **Iterator** (ListMember *member)
- Iterator & **operator++** (int)
- Iterator & **operator--** (int)
- T **operator*** () const
- bool **operator==** (const Iterator &iter) const
- bool **operator!=** (const Iterator &iter) const

5.15.1. Részletes leírás

template<typename T, typename compare = ascending<T>>
class SortedList< T, compare >::Iterator

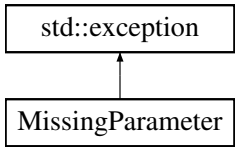
A lancolt lista iteratora Implementalja az iteratortol elvart muveleteket

Ez a dokumentáció az osztályról a következő fájl alapján készült:

- sorted_list.hpp

5.16. MissingParameter osztályreferencia

A MissingParameter osztály származási diagramja:



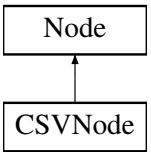
Ez a dokumentáció az osztályról a következő fájl alapján készült:

- main.cpp

5.17. Node osztályreferencia

`#include <graph.h>`

A Node osztály származási diagramja:



Publikus tagfüggvények

- Node (Array< Edge * > edges, const char *name)
- Node ()
- const char * getName () const
- Array< Edge * > getEdges () const
- Node & operator= (const Node &other)

Védett attribútumok

- Array< Edge * > edges
- char * name

5.17.1. Részletes leírás

Egy graf csucsa

5.17.2. Konstruktorok és destruktorok dokumentációja

5.17.2.1. Node() [1/2]

```
Node::Node (
            Array< Edge * > edges,
            const char * name)
```

Letrehoz egy csucsot

Paraméterek

edges	a csucs elei
name	a csucs neve

5.17.2.2. Node() [2/2]

```
Node::Node ()
```

Letrehoz egy izolalt pontot

5.17.3. Tagfüggvények dokumentációja

5.17.3.1. getEdges()

```
Array< Edge * > Node::getEdges () const
```

Visszatérési érték

A graf eleit tartalmazo Arra

5.17.3.2. getName()

```
const char * Node::getName () const
```

Visszatérési érték

A graf neve c-stringkent

5.17.3.3. operator=()

```
Node & Node::operator= (
            const Node & other)
```

Letrehoz egy csucsot egy masik masolatakent

Paraméterek

other	a jobb ertek

Visszatérési érték

ez a csucs

5.17.4. Adattagok dokumentációja

5.17.4.1. edges

```
Array<Edge*> Node::edges  [protected]
```

A csucs elei

5.17.4.2. name

```
char* Node::name  [protected]
```

A csucs neve

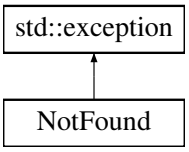
Ez a dokumentáció az osztályról a következő fájlok alapján készült:

- graph.h
- graph.cpp

5.18. NotFound osztályreferencia

```
#include <graph.h>
```

A NotFound osztály származási diagramja:



5.18.1. Részletes leírás

Kivetel arra, ha valami nem szerepel valamiben

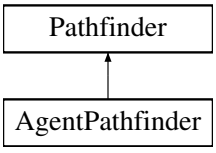
Ez a dokumentáció az osztályról a következő fájl alapján készült:

- graph.h

5.19. Pathfinder osztályreferencia

```
#include <pathfinder.h>
```

A Pathfinder osztály származási diagramja:



Publikus tagfüggvények

- Pathfinder (Graph graph, size_t numRoutes=3)
- virtual SortedList< Route * > getRoutes (const Node &from, const Node &to, Time starTime)=0

Védett attribútumok

- Graph graph
- size_t numRoutes

5.19.1. Részletes leírás

Virtualis utvonalkereso egy grafban oroklesevel lehet utvonalkereso algoritmusokat implementalni

5.19.2. Konstruktorok és destruktorok dokumentációja

5.19.2.1. Pathfinder()

```
Pathfinder::Pathfinder (
            Graph graph,
            size_t numRoutes = 3)
```

Letrehoz egy utvonalkeresot

Paraméterek

graph	a graf amiben az utvonalakat keressuk
numRoutes	a keresett utak szama

5.19.3. Tagfüggvények dokumentációja

5.19.3.1. getRoutes()

```
virtual SortedList< Route * > Pathfinder::getRoutes (
            const Node & from,
            const Node & to,
            Time starTime)  [pure virtual]
```

Utvonalakat keres a megadott parameterekkel

Paraméterek

from	a csucs ahonnan indulunk
to	a csucs ahova erkezunk
starTime	az indulas ideje

Visszatérési érték

Az utvonalakat tartalmazo idohossz szerint novekvo sorrendben rendezett lista

Megvalósítják a következők: AgentPathfinder.

5.19.4. Adattagok dokumentációja

5.19.4.1. graph

```
Graph Pathfinder::graph  [protected]
```

A graf amiben az utvonalat keressuk

5.19.4.2. numRoutes

```
size_t Pathfinder::numRoutes  [protected]
```

A keresett utak szama

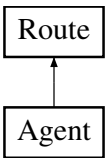
Ez a dokumentáció az osztályról a következő fájlok alapján készült:

- pathfinder.h
- pathfinder.cpp

5.20. Route osztályreferencia

```
#include <graph.h>
```

A Route osztály származási diagramja:



Publikus tagfüggvények

- Route (Array< Edge * > edges, Time startTime)
- Array< Edge * > getEdges () const
- Time getStartTime () const
- size_t getTotalWeight () const

Védett attribútumok

- Array< Edge * > edges

5.20.1. Részletes leírás

Egy ut egy grafban

5.20.2. Konstruktorok és destruktorok dokumentációja

5.20.2.1. Route()

```
Route::Route (
            Array< Edge * > edges,
            Time startTime)
```

Letrehoz egy utat

Paraméterek

edges	az ut elei
startTime	a kezdesi ido

5.20.3. Tagfüggvények dokumentációja

5.20.3.1. getEdges()

```
Array< Edge * > Route::getEdges () const
```

Visszatérési érték

Az ut elei

5.20.3.2. getStartTime()

```
Time Route::getStartTime () const
```

Visszatérési érték

Az ut kezdesi ideje

5.20.3.3. getTotalWeight()

```
size_t Route::getTotalWeight () const
```

Az ut osszes elen vegig menve enny ido lenne varakozasokkal eggyutt

Visszatérési érték

az osszsuly percben

5.20.4. Adattagok dokumentációja

5.20.4.1. edges

```
Array<Edge*> Route::edges  [protected]
```

Az ut altal erintett elek sorrendben

Ez a dokumentáció az osztályról a következő fájlok alapján készült:

- graph.h
- graph.cpp

5.21. SortedList< T, compare > osztálysablon-referencia

`#include <sorted_list.hpp>`

Osztályok

- class Iterator

Publikus tagfüggvények

- SortedList ()
- size_t getLength () const
- void remove (T element)
- void removeAt (size_t at)
- Iterator begin ()
- Iterator end ()
- void operator+= (T element)

5.21.1. Részletes leírás

template<typename T, typename compare = ascending<T>>
class SortedList< T, compare >

Egy dinamikusan tarolt rendezett lancolt lista

Sablon paraméterek

T	az elemek tipusa == operatort implementalni kell
compare	a rendezeshez hasznalt functor

5.21.2. Konstruktorok és destruktorok dokumentációja

5.21.2.1. SortedList()

```
template<typename T , typename compare = ascending<T>>
SortedList< T, compare >::SortedList ()  [inline]
```

Letrehoz egy ures listat

5.21.3. Tagfüggvények dokumentációja

5.21.3.1. begin()

```
template<typename T , typename compare = ascending<T>>
Iterator SortedList< T, compare >::begin ()  [inline]
```

Visszatérési érték

Lista elejen levo iterator

5.21.3.2. end()

```
template<typename T , typename compare = ascending<T>>
Iterator SortedList< T, compare >::end ()  [inline]
```

Visszatérési érték

Lista utolso eleme utan mutato iterator

5.21.3.3. getLength()

```
template<typename T , typename compare = ascending<T>>
size_t SortedList< T, compare >::getLength () const  [inline]
```

Visszatérési érték

A lista hossza

5.21.3.4. operator+=()

```
template<typename T , typename compare = ascending<T>>
void SortedList< T, compare >::operator+= (
            T element)  [inline]
```

Hozzaad a listahoz egy elemet A megadott rendezo functor alapjan jo helyre teszi

Paraméterek

element	A beszurando ertek

5.21.3.5. remove()

```
template<typename T , typename compare = ascending<T>>
void SortedList< T, compare >::remove (
            T element)  [inline]
```

Torol egy elemet a listabol, ha tartalmazza

Paraméterek

element	A torolendo ertek

5.21.3.6. removeAt()

```
template<typename T , typename compare = ascending<T>>
void SortedList< T, compare >::removeAt (
            size_t at)  [inline]
```

Torol egy elemet a listabol

Paraméterek

at	a torlendo elem indexe

Ez a dokumentáció az osztályról a következő fájl alapján készült:

- sorted_list.hpp

5.22. Time osztályreferencia

`#include <mytime.h>`

Publikus tagfüggvények

- Time ()
- Time (size_t hour, size_t minute)
- void operator+= (size_t minutes)
- void operator-= (size_t minutes)
- size_t operator- (const Time &other) const
- std::ostream & print (std::ostream &os) const

5.22.1. Részletes leírás

Egy nap ideje perc pontossaggal

5.22.2. Konstruktorok és destruktorok dokumentációja

5.22.2.1. Time() [1/2]

```
Time::Time ()
```

Letrehoz egy idot ejfelkor (00:00)

5.22.2.2. Time() [2/2]

```
Time::Time (
            size_t hour,
            size_t minute)
```

Letrehoz egy idot

Paraméterek

hour	az ido oraja
minute	az ora perce

5.22.3. Tagfüggvények dokumentációja

5.22.3.1. operator+=()

```
void Time::operator+= (
            size_t minutes)
```

Hozzaad az idohoz valahany percet

Paraméterek

minutes	

5.22.3.2. operator-()

```
size_t Time::operator- (
            const Time & other) const
```

Megadja hany percel kesobb lesz a bal oldali ido a jobb oldali hoz kepest Ha a jobb oldali operator kesobb van mint a bal oldali akkor a kovetkezo nap bal oldali idejehez szamol

Paraméterek

other	a jobb oldali ido

Visszatérési érték

A kulonbseg percben

5.22.3.3. operator-=()

```
void Time::operator-= (
            size_t minutes)
```

Kivon egy idobol valahany percet

Paraméterek

minutes	

5.22.3.4. print()

```
std::ostream & Time::print (
            std::ostream & os) const
```

Kiirja az idot HH:mm formatumban

Paraméterek

os	a stream amibe irja

Visszatérési érték

a stream amibe irta

Ez a dokumentáció az osztályról a következő fájlok alapján készült:

- mytime.h
- mytime.cpp

6. fejezet

Fájlok dokumentációja

6.1. agent.h

```
00001 #include "pathfinder.h"
00002 #include "graph.h"
00003 #include "array.hpp"
00004 #include "sorted_list.hpp"
00005
00006 #ifndef AGENT
00007 #define AGENT
00008
00012 enum AgentState
00013 {
00017     moved = 0,
00021     arrived = 1,
00025     terminated = 2
00026 };
00027
00031 class Agent : public Route {
00035     const Node* start;
00036
00040     const Node* target;
00041 public:
00049     Agent(Edge& edge, const Node& start, const Node& target, Time startTime);
00050
00055     Agent(const Agent&, Edge* firstEdge = NULL);
00056
00061     AgentState step();
00062
00066     const Node* head() const;
00067 };
00068
00072 class AgentPathfinder : public Pathfinder {
00077     SortedList<Agent*, alwaysFirst<Agent*>> agents;
00078
00083     Array<Agent*> agentsToDelete;
00084
00090     void splitAgent(const Agent&, Time startTime, size_t startEdge = 1);
00091
00097     void deleteAgent(Agent* agent);
00098
00099 public:
00103     AgentPathfinder(Graph graph, size_t numRoutes = 3);
00104
00110     SortedList<Route*> getRoutes(const Node& from, const Node& to, Time starTime);
00111
00112     ~AgentPathfinder();
00113 };
00114 #endif
```

6.2. array.hpp

```
00001 #include <stdexcept>
00002 #include "memtrace.h"
00003
00004
00005 #ifndef ARRAY
```

```
00006 #define ARRAY
00007
00012 template<typename T>
00013 class Array {
00017     T* array;
00018
00022     size_t length;
00023
00027     size_t capacity;
00028
00034     void fill(const T* arr, size_t len) {
00035         for (size_t i = 0; i < len; i++)
00036         {
00037             array[i] = arr[i];
00038         }
00039     }
00040 public:
00041     Array() : array(nullptr), length(0), capacity(0) {}
00042
00048     Array(size_t length, const T* arr = nullptr) : array(new T[length]), length(length),
      capacity(length) {
00049         if (arr != nullptr) {
00050             fill(arr, length);
00051         }
00052     }
00053
00059     Array(const Array& other, size_t cap = 0) : length(other.length), capacity((cap > other.length) ?
      cap : other.length) {
00060         array = new T[capacity];
00061         fill(other.array, length);
00062     }
00063
00067     size_t getLength() const {
00068         return length;
00069     }
00070
00075     T last() const {
00076         return operator[](length-1);
00077     }
00078
00084     Array& operator=(const Array& other) {
00085         if (&other == this) {
00086             return *this;
00087         }
00088         length = other.length;
00089         capacity = other.length;
00090         if (array != NULL) {
00091             delete[] array;
00092         }
00093         array = new T[length];
00094         fill(other.array, length);
00095         return *this;
00096     }
00097
00103     T operator[](size_t i) const {
00104         if (i >= length) {
00105             throw std::out_of_range("array overindexed");
00106         }
00107         return array[i];
00108     }
00109
00116     T* operator+(size_t i) const {
00117         if (i >= length) {
00118             throw std::out_of_range("array overindexed");
00119         }
00120         return array + i;
00121     }
00122
00128     void operator+=(T element) {
00129         if (length >= capacity) {
00130             capacity = 2 * length + 1;
00131             T* tmp = array;
00132             array = new T[capacity];
00133             if (tmp != NULL) {
00134                 fill(tmp, length);
00135             }
00136             delete[] tmp;
00137         }
00138         array[length] = element;
00139         length++;
00140     }
00141
00142     ~Array() {
00143         delete[] array;
00144     }
00145 };
00146 #endif // !ARRAY
```

6.3. csvgraph.h

```
00001 #include "csvparser.h"
00002 #include "graph.h"
00003
00004 #ifndef CSVGRAPH
00005 #define CSVGRAPH
00006
00011 class CSVNode;
00012
00017 class CSVEdge : public Edge {
00018 public:
00019     CSVEdge(int weight, Array<Time> startTimes, const char* name);
00020
00024     void operator+=(CSVNode& node);
00025 };
00026
00027 class CSVNode : public Node {
00028 public:
00029     CSVNode(const char* name);
00030
00034     void operator+=(CSVEdge& edge);
00035 };
00036
00041 class CSVGraph : public Graph {
00042
00043 public:
00048     CSVGraph(CSVParser& csv);
00049
00053     void operator+=(CSVNode& node);
00054
00055     ~CSVGraph();
00056 };
00057
00063 CSVLine writeRoute(Route& route);
00064
00071 Array<Time> parseTime(const char* timeString);
00072 #endif
```

6.4. csvparser.h

```
00001 #include <exception>
00002 #include <fstream>
00003 #include "array.hpp"
00004
00005 #ifndef CSVPARSER
00006 #define CSVPARSER
00007
00011 class FormatInvalid : public std::exception {
00015     char* whatStr;
00016 public:
00022     FormatInvalid(const char file[] = NULL, size_t line = 0, size_t character = 0);
00023
00024     virtual const char* what() const throw();
00025
00026     virtual ~FormatInvalid();
00027 };
00028
00032 class CSVLine {
00036     Array<Array<char>> columns;
00037
00043     Array<char> trim(const Array<char>& chars);
00044
00051     void createColumn(const char* start, size_t len);
00052
00059     const char* findNextSeparator(const char str[], char sep);
00060 public:
00064     CSVLine();
00065
00071     CSVLine(const char line[], char separator = ',');
00072
00076     Array<Array<char>> getColumns() const;
00077
00081     bool isEmpty() const;
00082
00083     CSVLine& operator=(const CSVLine&);
00084
00089     void operator+=(const char* str);
00090
00095     void operator+=(size_t num);
00096 };
00097
00104 std::ostream& operator<<(std::ostream& os, const CSVLine& line);
```

```
00105
00110 class CSVParser {
00114     char* path;
00115
00119     std::fstream file;
00120
00125     CSVParser* next;
00126
00130     bool beenOpened;
00131
00137     char* readLine();
00138
00144     void openFile();
00145 public:
00146
00151     CSVParser(const char* filePath);
00152
00157     void write(CSVLine line);
00158
00163     CSVLine read();
00164
00169     char* getFileName() const;
00170
00177     void operator+=(CSVParser& parser);
00178
00179     ~CSVParser();
00180 };
00181
00182 #endif // !CSVPARSER
```

6.5. graph.h

```
00001 #include "mytime.h"
00002 #include "array.hpp"
00003
00004 #ifndef GRAPH
00005 #define GRAPH
00006
00010 class NotFound : public std::exception {};
00011
00015 class Edge;
00016
00020 class Node {
00021 protected:
00025     Array<Edge*> edges;
00026
00030     char* name;
00031 public:
00037     Node(Array<Edge*> edges, const char* name);
00038
00042     Node();
00043
00047     const char* getName() const;
00048
00052     Array<Edge*> getEdges() const;
00053
00059     Node& operator=(const Node& other);
00060
00061     virtual ~Node();
00062 };
00063
00064 class Edge {
00065 protected:
00069     Node* from;
00070
00074     Node* to;
00075
00079     size_t weight;
00080
00085     Array<Time> startTimes;
00089     char* name;
00090
00091 public:
00100     Edge(Node* from, Node* to, int weight, Array<Time> startTimes, const char* name);
00101
00102     Edge();
00103
00107     const char* getName() const;
00108
00114     size_t getWeight(Time currentTime) const;
00115
00119     size_t getWeight() const;
00120
```

```
00126     Time getFirstStartTimeAfter(Time time) const;
00127
00131     Node* getToNode() const;
00132
00138     Edge& operator=(const Edge& other);
00139
00140     virtual ~Edge();
00141 };
00142
00146 class Graph {
00147 protected:
00151     Array<Node*> nodes;
00152 public:
00156     Graph();
00157
00162     Graph(Array<Node*> nodes);
00163
00168     Array<Node*> getNodes() const;
00169
00178     Node* getNode(const char* name, bool exactMatch = false) const;
00179 };
00180
00184 class Route {
00188     Time startTime;
00189 protected:
00193     Array<Edge*> edges;
00194 public:
00200     Route(Array<Edge*> edges, Time startTime);
00201
00206     Array<Edge*> getEdges() const;
00207
00212     Time getStartTime() const;
00213
00218     size_t getTotalWeight() const;
00219 };
00220
00221 #endif
```

6.6. log.hpp

```
00001 #include <sstream>
00002 #include <cstring>
00003 #include <string>
00004 #include <iostream>
00005
00006 #ifndef LOGGER
00007 #define LOGGER
00008
00009 //KIIRASOKHOZ SEGEDFUGGVENYEK
00010
00014 inline bool writeLog = false;
00015
00019 inline std::ostream& lstream = std::cout;
00020
00025 inline std::stringstream dispose;
00026
00031 inline std::ostream& l() {
00032     if(!writeLog ){
00033         dispose.clear();
00034         return dispose;
00035     }
00036         return lstream;
00037 }
00038
00045 template<typename T>
00046 std::string ID(T* ptr) {
00047
00048     std::stringstream sstream;
00049     sstream « ptr;
00050     std::string str = sstream.str();
00051     std::stringstream sstream2;
00052     sstream2 « '<' « typeid(T).name() « (char)str[strlen(str.c_str()) - 4] «
      (char)str[strlen(str.c_str()) - 3] « (char)str[strlen(str.c_str()) - 5] « '>';
00053     return sstream2.str();
00054 }
00055 #endif
```

6.7. mytime.h

```
00001 #include <iostream>
```

```
00002
00003 #ifndef TIME
00004 #define TIME
00005
00009 class Time {
00013     size_t day;
00014
00018     size_t hour;
00019
00023     size_t minute;
00024
00028     void validate();
00029 public:
00033     Time();
00034
00040     Time(size_t hour, size_t minute);
00041
00046     void operator+=(size_t minutes);
00047
00052     void operator-=(size_t minutes);
00053
00060     size_t operator-(const Time& other) const;
00061
00067     std::ostream& print(std::ostream& os) const;
00068 };
00069
00075 std::ostream& operator«(std::ostream& os, const Time& time);
00076 #endif
```

6.8. pathfinder.h

```
00001 #include "graph.h"
00002 #include "sorted_list.hpp"
00003
00004 #ifndef PATHFINDER
00005 #define PATHFINDER
00006
00011 class Pathfinder {
00012 protected:
00016     Graph graph;
00020     size_t numRoutes;
00021 public:
00027     Pathfinder(Graph graph, size_t numRoutes = 3);
00035     virtual SortedList<Route*> getRoutes(const Node& from,const Node& to, Time starTime) = 0;
00036
00037     virtual ~Pathfinder();
00038 };
00039 #endif
```

6.9. sorted_list.hpp

```
00001 #include <stdexcept>
00002 #include "memtrace.h"
00003
00004 #ifndef SORTEDLIST
00005 #define SORTEDLIST
00006
00007 //Functorok a listahoz
00008 template<typename T>
00009 class alwaysFirst {
00010 public:
00011     bool operator()(T t1, T t2) { return false; }
00012 };
00013
00014 template<typename T>
00015 class descending {
00016 public:
00017     bool operator()(T t1, T t2) { return t1 > t2; }
00018 };
00019
00020 template<typename T>
00021 class ascending {
00022 public:
00023     bool operator()(T t1, T t2) { return t1 < t2; }
00024 };
00025
00031 template<typename T, typename compare = ascending<T»
00032 class SortedList {
00036     class ListMember {
```

```
00037     public:
00041         T element;
00042
00046         ListMember* prev;
00047
00051         ListMember* next;
00052
00059         ListMember(T element, ListMember* prevP, ListMember* nextP) : element(element), prev(prevP),
      next(nextP) {
00060             if (prev != NULL) {
00061                 prev->next = this;
00062             }
00063             if (next != NULL) {
00064                 next->prev = this;
00065             }
00066         }
00067     };
00068
00069     compare pred;
00073     ListMember* list;
00074
00078     size_t length;
00079 public:
00084     class Iterator {
00085         ListMember* member;
00086     public:
00087         Iterator(ListMember* member) : member(member) {}
00088
00089         Iterator& operator++(int) {
00090             member = member->next;
00091             return *this;
00092         }
00093
00094         Iterator& operator--(int) {
00095             member = member->prev;
00096             return *this;
00097         }
00098
00099         T operator*() const {
00100             if (member == NULL) {
00101                 throw std::out_of_range("end of list");
00102             }
00103             return member->element;
00104         }
00105
00106         bool operator ==(const Iterator& iter) const {
00107             return member == iter.member;
00108         }
00109
00110         bool operator !=(const Iterator& iter) const {
00111             return !operator==(iter);
00112         }
00113     };
00114
00118     SortedList() : list(NULL), length(0) {}
00119
00123     size_t getLength() const {
00124         return length;
00125     }
00126
00131     void remove(T element) {
00132         ListMember* ptr = list;
00133         while (ptr != NULL) {
00134             if (ptr->element == element) {
00135                 if (ptr->next) {
00136                     ptr->next->prev = ptr->prev;
00137                 }
00138                 else {
00139                     ptr->prev = NULL;
00140                 }
00141                 if (ptr->prev) {
00142                     ptr->prev->next = ptr->next;
00143                 }
00144                 else {
00145                     list = ptr->next;
00146                 }
00147                 delete ptr;
00148                 length--;
00149                 return;
00150             }
00151             ptr = ptr->next;
00152         }
00153     }
00154
00159     void removeAt(size_t at) {
00160         ListMember* ptr = list;
00161         for (size_t i = 0; i < at; i++) {
```

```
00162             ptr = ptr->next;
00163         }
00164         if (ptr->prev) {
00165             ptr->prev->next = ptr->next;
00166         }
00167         else {
00168             list = ptr->next;
00169         }
00170         delete ptr;
00171         length--;
00172     }
00173 
00177     Iterator begin() {
00178         return Iterator(list);
00179     }
00180 
00184     Iterator end() {
00185         return Iterator(NULL);
00186     }
00187 
00193     void operator+=(T element) {
00194         ListMember* ptr = list;
00195         if (ptr != NULL) {
00196             while ((ptr->next != NULL)) {
00197                 if (!pred(ptr->next->element, element)) {
00198                     break;
00199                 }
00200                 ptr = ptr->next;
00201             }
00202         }
00203         ListMember* newPtr = new ListMember(element, ptr, ptr ? ptr->next : NULL);
00204         if (newPtr->prev == NULL) {
00205             list = newPtr;
00206         }
00207         length++;
00208     }
00209 
00210     ~SortedList() {
00211         ListMember* ptr = list;
00212         while (ptr != NULL) {
00213             ListMember* oldPtr = ptr;
00214             ptr = ptr->next;
00215             delete oldPtr;
00216         }
00217     }
00218 };
00219 #endif // !SORTEDLIST
```