Текстовод Уникальность https://textovod.com/unique

Отчет проверки уникальности текста

Дата проверки: 2023-06-13 19:40:37

Уникальность 49%

Удовлетворительно. Текст желательно доработать.

Текст

#include < algorithm>
#include < iomanip>
#include < iostream>
#include < unordered_map>
#include < vector>

#include "priority_queue.h"

template < typename Graph> class Dijkstra
{
private:
// Reconstruct a path from start to goal
static std: :vector< std: :pair< typename Graph: :location_t, typename Graph: :cost_t>
> reconstruct_path(
const typename Graph: :location_t & start,
const typename Graph: :location_t & goal,
std: :unordered_map< typename Graph: :location_t, typename Graph: :location_t> &
came_from,
std: :unordered_map< typename Graph: :location_t, typename Graph: :cost_t> &
cost_so_far
)
{
std: :vector< std: :pair< typename Graph: :location_t, typename Graph: :cost_t> >
path;
typename Graph: :location_t current = goal;

if (came_from.find(goal) == came_from.end())

```
{
return path; // no path can be found
}

while (current! = start)
{
path.push_back({current, cost_so_far[current]});
current = came_from[current];
}

path.push_back({start, (typename Graph: :cost_t)0});

// reconstructed path will start from end, reverse
std: :reverse(path.begin(), path.end());

return path;
}

public:
static void show(
std: :vector< std: :pair< typename Graph: :location_t, typename Graph: :cost_t> >
path,
const typename Graph: :location_t & start,
const typename Graph: :location_t & goal
)
{

std: :cout < < "\033[31m" < < std: :setw(2) < < start < < "\033[0m"
< < " -> "
< < "\033[36m" < < std: :setw(2) < < goal < < "\033[0m"
< < " | ";

if (path.size() == 0)
{
std: :cout < < "No path can be found from "
< < "\033[36m"
< < "'" < < start < < "'"
< < "\033[0m"
< < " to "
< < "\033[36m"
< < "'" < < goal < < "'"
```

```
< < "\033[0m"
< < "." < < std: :endl;

return;
}
else if (path.size() == 1)
{
std: :cout < < "Moving to the same edge. (cost: "
< < "\033[33m" < < std: :setw(2) < < path[0].second < < "\033[0m"
< < ")." < < std: :endl;

return;
}

std: :cout < < "\033[31m" < < std: :setw(2) < < start < < "\033[0m";

for (size_t i = 1; i < path.size(); i++)
{
// cost from the first edge to current is stored, so subtract cost of prev from current
std: :cout < < " -("
< < "\033[33m" < < std: :setw(2) < < path[i].second - path[i - 1].second < < "/" < <
std: :setw(3)
< < path[i].second < < "\033[0m"
< < ")-> ";

if (path[i].first == goal)
{
std: :cout < < "\033[32m";
}
else
{
std: :cout < < "\033[36m";
}

std: :cout < < std: :setw(2) < < path[i].first;

std: :cout < < "\033[0m";
}

std: :cout < < std: :endl;
}
```

```
static std: :vector< std: :pair< typename Graph: :location_t, typename Graph: :cost_t>
> search(
Graph & graph, const typename Graph: :location_t & start, const typename Graph:
:location_t & goal
)
{
std: :unordered_map< typename Graph: :location_t, typename Graph: :cost_t>
cost_so_far;
PriorityQueue< typename Graph: :location_t, typename Graph: :cost_t> frontier;
std: :unordered_map< typename Graph: :location_t, typename Graph: :location_t>
came_from;

frontier.push(start, typename Graph: :cost_t(0));
came_from[start] = start;
cost_so_far[start] = typename Graph: :cost_t(0);

while (! frontier.empty())
{
typename Graph: :location_t current = frontier.pop();

if (current == goal)
{
break;
}

for (typename Graph: :location_t next: graph.neighbors(current))
{
typename Graph: :cost_t new_cost = cost_so_far[current] + graph.cost(current, next);
if (cost_so_far.find(next) == cost_so_far.end() || new_cost < cost_so_far[next])
{
cost_so_far[next] = new_cost;
came_from[next] = current;
frontier.push(next, new_cost);
}
}
}

return Dijkstra< Graph> :: reconstruct_path(start, goal, came_from, cost_so_far);
}
};
```

```
#include < algorithm>
#include < iostream>
#include < map>
#include < stdexcept>
#include < string>
#include < vector>

class Graph
{
public:
// Types for graphs
typedef size_t location_t;
typedef int cost_t;

// Some constants
static const Graph: :location_t INF = 0;

private:
// Map of edges that stores every available edge from it
std: :map< location_t, std: :vector< std: :pair< Graph: :location_t, Graph: :cost_t> > >
edges;

public:
// Constructor
Graph(std: :vector< std: :vector< cost_t> > matrix)
{
// Convert matrix representation of a graph to map
for (location_t i = 0; i < matrix.size(); i++)
{
if (matrix.size()! = matrix[i].size())
{
throw std: :invalid_argument(
"Matrix representation of a graph must be a square. Matrix size: " + std:
:to_string(matrix.size()) + "; Matrix[i] size: " + std: :to_string(matrix[i].size()));
}

for (location_t j = 0; j < matrix[i].size(); j++)
{
if (matrix[i][j] > 0)
{
edges[i].push_back({j, matrix[i][j]});
```

```
}
}
}
}

// Get all neighbors
std: :vector< Graph: :location_t> neighbors(int id)
{
std: :vector< Graph: :location_t> result;

for (auto location: edges[id])
{
result.push_back(location.first);
}

return result;
}

// Print graph to a console
void show()
{
for (auto it: edges)
{
std: :cout < < "\033[36m" < < it.first < < "\033[0m" < < std: :endl;

for (location_t i = 0; i < it.second.size(); i++)
{
std: :cout < < " -("
< < "\033[33m" < < it.second[i].first < < "\033[0m"
< < ")-> "
< < "\033[36m" < < it.second[i].second < < "\033[0m" < < std: :endl;
}
}
}

// Get a cost of moving from one edge to another
Graph: :cost_t cost(Graph: :location_t first, Graph: :location_t second)
{
auto result = std: :find_if(
edges[first].begin(),
edges[first].end(),
```

```
[second](const std: :pair< Graph: :location_t, Graph: :cost_t> & element) { return
element.first == second; }
);

if (result == edges[first].end())
{
throw std: :invalid_argument(
"First edge '" + std: :to_string(first) + "' does not a have a path to edge '" + std:
:to_string(second)
+ "'."
);
}

return result-> second;
}
};
#include < queue>
#include < utility>
#include < vector>

// Modified version of std: :priority_queue. Use vector instead of heap and change
priority direction
template < typename T, typename priority_t> struct PriorityQueue
{
public:
typedef std: :pair< priority_t, T> PQElement;

std: :priority_queue< PQElement, std: :vector< PQElement> , std: :greater<
PQElement> > elements;

inline bool empty() const
{
return elements.empty();
}

inline void push(T item, priority_t priority)
{
elements.emplace(priority, item);
}

T pop()
```

```
{
T best_item = elements.top().second;
elements.pop();
return best_item;
}
};
```

Источники

- https://www.redblobgames.com/pathfinding/a-star/implementation.html (37%)
- https://www.programmersought.com/article/979310198986/ (36%)
- https://www.programmersought.com/article/771710702710/ (30%)
- https://habr.com/ru/articles/331220/ (29%)
- https://www.pvsm.ru/algoritmy/258470 (29%)
- https://www.redblobgames.com/pathfinding/a-star/implementation.cpp (26%)
- https://codeforum.org/threads/problems-in-creating-the-h-file-of-an-already-existing-code.5035/ (24%)
- https://gamedev.stackexchange.com/questions/135228/need-some-help-with-a-pathfinding (19%)
- https://gist.github.com/MORTAL2000/e26c072bf1e730bde93235da95308d77 (17%)
- https://simblob.blogspot.com/2018/01/updating-c-code-on-a-implementation-page.html (14%)
- https://simblob.blogspot.com/2018/01/ (14%)
- https://simblob.blogspot.com/2018/ (14%)
- https://github.com/johnnychhsu/Interview_notes/blob/master/md_files/algorithm.md (8%)
- https://github.com/melkir/A-Star-Python/blob/master/Algorithms.py (8%)
- https://www.redblobgames.com/pathfinding/a-star/introduction.html (8%)
- https://github.com/samdjstephens/pydstarlite/blob/master/pydstarlite/utility.py (7%)
- https://electshema.ru/elektrotehnika/algorithm-builder-urok-1-vvedenie.html (6%)
- https://habr.com/ru/articles/331192/ (6%)
- https://itnan.ru/post.php?c=1&p=331192 (6%)
- https://programmerall.com/article/67302134195/ (5%)
- https://programmerall.com/article/73631202070/ (5%)
- https://ppt-online.org/384661 (5%)
- https://en.ppt-online.org/384661 (5%)
- https://mypreza.com/informatika/algoritmy-poiska-puti-ot-poiska-v (4%)