**Санкт-Петербургский Государственный Политехнический**

**Университет**

Факультет Технической Кибернетики

Кафедра компьютерных систем и программных технологий

**ОТЧЕТ**

**о курсовой работе**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ICFPC CONTEST 2012\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

(тема работы)

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_программирование\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

(наименование дисциплины)

**Работу выполнили студенты**

2081/4 Зорин Е.А, Серебряков Е.Д

Дирочка Д.А.

группа Ф.И.О

**Преподаватель**

**\_\_\_\_\_\_\_\_\_** Глухих М.И

подпись Ф.И.О

Санкт-Петербург

2012

1. Описание проекта.

Основные правила:

* Робот пропускает любые неизвестные ему символы (а известны ему только L, R, U, D, W, A.
* Если Робот не может выполнить команду, он этот ход просто ничего не делает.
* После хода робота обновляется карта по правилам, описанным ниже.
* После обновления карты проверяются условия завершения работы.

Описание карты

Карта — это сетка m×n клеток, координата (1,1) расположена **внизу** и слева.  
Каждая клетка содержит один из следующих символов:

1. **R** — робот
2. **\*** — камень
3. **L** — закрытый выход
4. **.** — земля
5. **#** — стена
6. **\** — λ
7. **O** — открытый выход
8. "" — пробел, пустая клетка

Ничто не может проникнуть сквозь стену. Ничто не может выйти за пределы поля m×n. Робот может копать землю, собирать λ и двигать камни. Камни падают. Камни могут убить робота.  
  
Пример начального состояния:

#######

#..\*\*\* #

#..\\\ #

# ..\*\* #

#.\*.\*\ #

LR.... #

#######

Команды робота

Если координаты робота (x, y):

1. L — налево, в (x-1, y)
2. R — направо, в (x+1, y)
3. U — вверх, в (x, y+1)
4. D — вниз, в (x, y-1)
5. W — ждать, ничего не делать
6. A — прервать исследование шахты

Двигаться можно в клетку с открытым выходом, в пустую, в клетку с землёй и клетку с λ. Ещё можно двигаться (только налево и направо) в клетку с камнем, если за камнем пустое место. После ухода из клетки робот всегда оставляет в клетку пустоту.

Обновление карты (симуляция)

После движения робота проходит шаг симуляции. Во время этого этапа всегда читается старое состояние и пишется в новое. В следующем порядке по координатам: (1, 1), (2, 1), ..., (n, 1), (1, 2)… (n, m).  
  
Правила проверяются по порядку, сверху вниз. Если одно сработало, то следующее уже не сработает.

1. Под камнем пустота — камень падает на 1 клетку вниз.
2. Под камнем — камень, справа пусто и справа внизу пусто — камень падает по диагонали вправо.
3. Под камнем — камень, слева пусто и слева внизу пусто — камень падает по диагонали влево.
4. Под камнем — λ, справа пусто и справа внизу пусто — камень падает по диагонали вправо.
5. Остальные клетки не трогаются.

Если на карте больше нету λ, все закрытые выходы открываются.  
Если под только что упавшим камнем стоял робот — робот ломается, это — поражение.  
За один шаг симуляции камни падают на 1 клетку вниз.  
Иногда камни с разных сторон могут упасть в одну клетку. В этом случае в этой клетке теперь лежит 1 камень.

Условия завершения

* Робот вошёл в открытый лифт — WIN
* Робот выполнил команду A — WIN
* Робота раздавило камнем — LOSE

Ввод/вывод

Ввод с stdin, вывод в stdout  
  
Если некоторые строки короче максимальной по длине строки, то они считаются дополненными пробелами справа.  
  
На каждой карте в начале нет открытых выходов. Есть ровно 1 робот и ровно 1 закрытый выход.

##### Начисление очков

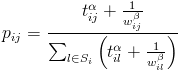
За каждый шаг -1  
За собранную λ +25  
За собранную λ в момент выполнения команды A +25  
За собранную λ в момент достижения выхода +50

1. Описание алгоритма.

Для нахождения пути между двумя узлами(лямбдами) используется алгоритм A\*. Пошагово просматриваются все пути, ведущие от начальной вершины в конечную, пока не найдётся минимальный. В начале работы просматриваются узлы, смежные с начальным; выбирается тот из них, который имеет минимальное значение *f(x)*, после чего этот узел раскрывается. На каждом этапе алгоритм оперирует с множеством путей из начальной точки до всех ещё не раскрытых (листовых) вершин графа («множеством частных решений»), которое размещается в очереди с приоритетом. Приоритет пути определяется по значению *f(x) = g(x) + h(x)*. Алгоритм продолжает свою работу до тех пор, пока значение *f(x)* целевой вершины не окажется меньшим, чем любое значение в очереди (либо пока всё дерево не будет просмотрено). Из множественных решений выбирается решение с наименьшей стоимостью. В нашей реализации при нахождении смежных вершин для узла генерируется путь робота до него, и, если путь существует, робот перемещается в проверяемую клетку, откуда выполняет проверку движения вверх, вниз, влево или вправо. Это нужно в связи с тем, что после каждого хода робота состояние карты изменяется. Также этот момент существенно упрощает просчет толкания камней при движении влево и вправо – не нужно хранить список перемещенных камней.

Для определения порядка обхода лямбд используется муравьиный алгоритм. Суть его заключается в том, что каждый раз проходя от муравейника до пищи и обратно, муравьи оставляют за собой дорожку феромонов. Другие муравьи, почувствовав такие следы на земле, будут инстинктивно устремляться к нему. Поскольку эти муравьи тоже оставляют за собой дорожки феромонов, то чем больше муравьев проходит по определенному пути, тем более привлекательным он становится для их сородичей. При этом, чем короче путь до источника пищи, тем меньше времени требуется муравьям на него – а следовательно, тем быстрее оставленные на нем следы становятся заметными.

Каждый муравей хранит в памяти список пройденных им узлов. Выбирая узел для следующего шага, муравей «помнит» об уже пройденных узлах и не рассматривает их в качестве возможных для перехода. На каждом шаге список запретов пополняется новым узлом, а перед новой итерацией алгоритма – то есть перед тем, как муравей вновь проходит путь – он опустошается.  
Кроме списка запретов, при выборе узла для перехода муравей руководствуется «привлекательностью» ребер, которые он может пройти. Она зависит, во-первых, от расстояния между узлами (то есть от веса ребра), а во-вторых, от следов феромонов, оставленных на ребре прошедшими по нему ранее муравьями. Естественно, что в отличие от весов ребер, которые являются константными, следы феромонов обновляются на каждой итерации алгоритма: как и в природе, со временем следы испаряются, а проходящие муравьи, напротив, усиливают их.  
Пусть муравей находится в узле http://latex.codecogs.com/gif.latex?i, а узел http://latex.codecogs.com/gif.latex?j– это один из узлов, доступных для перехода: http://latex.codecogs.com/gif.latex?j%20%5Cin%20S_i. Обозначим вес ребра, соединяющего узлы http://latex.codecogs.com/gif.latex?iи http://latex.codecogs.com/gif.latex?j, как http://latex.codecogs.com/gif.latex?w_%7Bij%7D, а интенсивность феромона на нем – как http://latex.codecogs.com/gif.latex?t_%7Bij%7D. Тогда вероятность перехода муравья из http://latex.codecogs.com/gif.latex?iв http://latex.codecogs.com/gif.latex?jбудет равна:



где http://latex.codecogs.com/gif.latex?%5Calphaи http://latex.codecogs.com/gif.latex?%5Cbeta– это регулируемые параметры, определяющие важность составляющих (веса ребра и уровня феромонов) при выборе пути. В нашей реализации http://latex.codecogs.com/gif.latex?%5Calpha = http://latex.codecogs.com/gif.latex?%5Cbeta =0.5

После того, как муравей успешно проходит маршрут, он оставляет на всех пройденных ребрах след, обратно пропорциональный длине пройденного пути:C:\Users\fix\Downloads\4462a7b26c1d2772a800ddef20a74398.png

где http://latex.codecogs.com/gif.latex?L– длина пути.

Кроме этого, следы феромона испаряются, то есть интенсивность феромона на всех ребрах уменьшается на каждой итерации алгоритма. Таким образом, в конце каждой итерации необходимо обновить значения интенсивностей:

C:\Users\fix\Downloads\4b503628e93faf66585fdd96055df1bc.png

Где p – скорость испарения феромонов. У нас p = 0.2.

Таким образом, реализация следующая:

В точке, в которой находится робот, создается муравей. Текущая точка добавляется в список пройденных муравьем. C помощью алгоритма локального поиска – A\* - создаются ребра, ведущие из текущей точки ко всем непройденным лямбдам и добавляются в список ребер. Если путь до какой-либо лямбды не найден, лямбда игнорируется на текущем шаге. Если при взятии какой-либо лямбды становится не возможным взять другую(робот попадает в ловушку), лямбда игнорируется. Вычисляются вероятности(зависящие от длины ребра и числа феромонов) прохода муравья по каждому ребру и в соответствии с ними выбирается ребро, по которому пойдет муравей. Муравей двигается по ребру и, следовательно, конечная точка этого ребра добавляется в список пройденных. Также ребро, добавляется в список пройденных ребер, чтобы в дальнейшем было возможно обновить количество феромонов. Вместе с муравьем двигается и робот. Это нужно для того чтобы искать возможные ребра, начала которых в текущей вершине муравья. Муравей и робот ходят до тех пора не пройдут все вершины (если какая-либо из вершин недоступна, она игнорируется). Вычисляется стоимость пути и сравнивается с предыдущей. Результатом движения одного муравья является последовательность граней. Далее происходит обновление феромонов на пройденных ребрах. На следующей итерации карта возвращается в изначальное состояние. Число итераций равно количеству муравьев, которые проходят карту. У нас это 100.

В результате работы муравьиного алгоритма получается последовательность точек. Робот генерирует маршруты и двигается по ним. Если оказывается, что какой то путь не найден, вызывается команда A.

1. Распределение обязанностей.

Зорин Е.А – написание глобального поиска, совмещение локального поиска с глобальным.

Серебряков Е.Д. – написание локального поиска, совмещение локального поиска с глобальным.

Дирочка Д.А – написание симулятора, тестирование.

1. Результаты работы программы:

Ввод:

######## #..R...# #..\*...# #..#...# #.\.\..L ####\*\*.# #\.....# #\..\* .#########

Вывод:



1. Выводы: реализованный нами робот далеко не оптимален. Главная проблема состоит в том, что робот не пытается обойти камень, столкнув который, он закроет себе путь к конечной точке. Мы не смогли решить данную проблему. Попыткой было просматривать существование пути из точки камня до конечной точки, при смещении камня роботом. Но мы столкнулись с ошибкой переполнения стека (Stack Overflow). Также не реализована команда ожидания.