Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Российский химико-технологический университет имени Д.И. Менделеева»
Кафедра информационных компьютерных технологий

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ № 11

Выполнил студент группы.....КС-33.....(Вагенлейтнер Никита Сергеевич)

Ссылка на репозиторий:.....(https://github.com/MUCTR-IKT-CPP/VagenlejtnerNS_33_alg.git)

Приняли:..... Пысин Максим Дмитриевич
.....Краснов Дмитрий Олегович
.....Лобанов Алексей Владимирович
.....Крашенинников Роман Сергеевич

Дата сдачи: ..... (26.02.2025)

Оглавление

Описание задачи.

ℜ □◊⎍ ◊J ⌊◊⟨⌊□◊⎍⌊□ⵖ ⎍Γ □◊⟨⎍Γ⌋ ⎍□Σ⟨ ⎍Σ⎍⟩□ ⟩⎍ ™◊⎍ ©Σⵖ Σ□◊⎍⌊□ ⌊◊⟨⌊□⍀ⵖ⎍◊ N ⵖ◊ N. ℘Σⵖ Σ□◊⎍⌊□ ™⎍⎍Σ ⵖ ⟩⎍ ™◊®◊⎍ ⌊◊⟨⌊□⍀ⵖ⎍ ⎍ ⟩⌊Σ™ ⎍J ⎍⌊⎍ ⎍□◊®⎍⌊◊⎍:

≠◊⟨⌊□⍀ⵖ⎍ ⌊⟨□⎍ ◊ⵖ ⌊⎍Σ⎍ ⎍⌊⌊⟩⎍ ⌊ ⌊™⍀ⵖ ®⌋J ⌊™.
≠◊⟨⌊□⍀ⵖ⎍ ⌊⟨□⎍ ◊ⵖ ⌊⎍Σ⎍ ⎍⌊⌊⟩⎍ ⌊ ⌊™⍀ⵖ ®⌋J ⌊™.
≠◊⟨⌊□⍀ⵖ⎍ ⌊⟨□⎍ ◊ⵖ ⌊⎍Σ⎍ ⎍ ⎍⎍⎍⌊.
∉□◊®⎍⌊⟩ⵖ⌋ ⎍ ⎍⎍⎍ ⵖΣ ⌊⟨□⎍ ◊⎍Σ⌊⟩ⵖ⌊ ™⎍⎍⎍Σⵖ ⟨⌋⎍ Σ™⍀ⵖ⟩⎍®Σⵖⵖ⌋⎍.
∉□◊®⎍⌊⟩ⵖ⌋ ⎍ ⎍⎍⎍ ⵖΣ ™⎍⎍⎍Σⵖ ⟨⌋⎍ ⍀ ⎍⎍⍀□⌊®◊ⵖⵖ⌋⎍, ⟩ⵖΣ©⌊ ⎍⍀ⵖ⌊ ⌊⎍⎍⎍, ®ⵖΣ©⌊ ⎍⍀ⵖ⌊ ®⎍⎍⎍.
ℜ⌊◊⟨⌊□⍀ⵖ⎍Σ ⵖΣ ⎍⍀⎍Σ⎍ ⟩ ⎍⎍Σ⟩⎍®⌊®◊⎍ ⌊⟨⌋◊⟩⎍Σ⎍ ® ⌊ ⎍⎍⌊□⌋Σ ⵖΣ⌋⌊⌊□ ⌊ ⌊⌊◊⟩⎍ ⎍⎍ ®J ⌊™◊.
⊂⍀ⵖ⍀⎍⌊⟩ⵖ◊□ ⅄□⍀ⵖ◊ ⌊□⌊J ⌊™◊ 1 ⌊ ®◊™□◊⎍.
≠◊⟨⌊□⍀ⵖ⎍ ⟩⟩⎍⎍⎍⎍ ⌊ ⌊®◊™□◊⎍⌊®, ⌊Σ□Σ™®⌊©◊⎍ ⟩□ ⌊⌊ ⌊®◊™□◊⎍◊⎍ ⍀⍀ⵖ⌊ ⎍⎍⌊⌊⟩⎍ ⌊ ⌊⌊ ⟩⌊Σ™ ⎍J ⎍Σ⍀ ⍀ ⌊◊⎍Σ□ⵖ ⍀:

0	0	0	0	0
0	1	1	1	0
0	1	Ξ	1	0
0	1	1	1	0
0	0	0	0	0

℘Σⵖ Σ□◊⎍⌊□ ™⎍⎍Σⵖ ⌊□ⵖ⍀⍀◊⎍ ⟩⌊Σ™ ⎍J ⎍Σ⌊ ⌊◊□◊⍀Σ⎍□⌋:

∠◊⌊ ⍀Σ□ ⌊◊⟨⌊□⍀ⵖ⎍◊.
⊂◊⌊ ⟩⍀⍀⌊⟩ⵖ◊□ ⅄□⍀ⵖ◊ ⌊□⌊J ⌊™◊.
⊆◊⌊⍀⌊Σ ⌊□□⍀⌋J ©◊⌊Σ□Σ⌋ ⍀◊⌊ ⟩⍀⍀⌊⟩ⵖ ⎍Γ ⅄□⍀ⵖ⌋.
⊆◊⌊⍀⌊Σ ⌊®◊™□◊⎍⌊® ⌊ⵖ⟩Σⵖ⎍⎍□◊⟩⟩⟩ (⌊ ®◊™□◊⎍ ⟩⌊ ⟩⎍⌊□⍀ⵖ ⎍Γ = ⍀◊⌊ ⟩⍀⍀⌊⟩ⵖ ⎍Γ ⅄□⍀ⵖ⌋ ⌊□⌊J ⌊™◊, ⵖ⌊ ⎍◊⌊ , ⌊⎍⌊ ⟨⌋ ® ⌊™ⵖ ⍀ ⟩⎍⌊□⍀ⵖ ⍀ ⍀⌊©⌋⌊ ⌊⌊⍀Σ⟩⎍⟩□ 2 ⌊□⌊J ⌊™◊ ⍀⍀ⵖ⍀⍀⌊⟩ⵖ ⎍Γ ⅄□⍀ⵖ⌋)
∉⌊⟩⌋Σ⌊◊®Σ□⁄Σⵖ⌊□ ©Σⵖ Σ□◊⎍⌊□◊ ⎍□Σ⟨ ⍀Σ⎍⟩□ ⵖ◊⌊⟩◊⎍ ◊⌋©⌊□⍁⍀⌊ ⍁⎍⌊□⌋ ⟩⌋Σ⎍ ®⌋J ⌊™⍁ ⌊◊⟨⌊□⍀ⵖ⎍◊.

∗®◊⌊ ⍀Σ ⌊□⌊©□◊⍀⍀◊ ™⎍⎍⍀ⵖ◊ ®⌋®⌊™⍁ ⌊◊□◊⍀Σ⎍□⌋⌊◊⟨⌊□⍀ⵖ⎍◊ ® ⌊◊⌋⌊ ⎍◊⌊ , ⌊⎍⌊ ⟨⌋ ⍀ⵖ ⍀⌊© ⟨⌋⎍ ⟩©Σⵖ Σ□⍁□ ⌊®◊ⵖ ⌊⌊®⎍⌊□ⵖ⌊, ⎍.Σ. ⌊◊⟨⌊□⍀ⵖ⎍Σ ⟩⌊J □◊ⵖ □Σ⎍⟩□ ⟩◊⍀ ⌊⌊ ⟩Σ⟨Σ ® ⌊◊⌋⌊ ◊ ⌊Σ□Σ©Σⵖ Σ□⍁□ ⍀Σ⎍⟩□ ⵖ◊ ⌊ⵖ⌊®Σ ⌊⌊□Σ™Σ⌋Σⵖ⎍⌋J ™◊ⵖ⎍⌋J (⌒◊ ®⌋⌊⍀ⵖ⍀Σⵖ⍁□ ⵖ⎍⌊©⌊⌊◊™◊ⵖ⍁□ ⌊⍀⌊◊©◊Σ⎍⟩□ ™⌊⌊⍀ⵖ⍀Σ⌋⎍⌋Σ 5 ⌊Σ□®⍁⌊⌋J ⟨◊⌋⌊®.)

Описание метода/модели.

Поиск в глубину

В коде реализован поиск в глубину (DFS - Depth-First Search) с помощью стека (stack). Этот

алгоритм используется для нахождения пути от входа (1,1) к выходу (n-2, n-2).

- Используется стек (stack) для хранения текущей позиции (x, y) и пути до нее.
- Используется множество посещенных клеток (visited), чтобы не заходить в одну и ту же клетку дважды.
- Пока стек не пуст, извлекаем последний добавленный элемент (это делает алгоритм жадным — сначала идем вглубь).
- path хранит последовательность клеток, которые привели нас в (x, y).
- Если текущая клетка совпадает с выходом (n-2, n-2), возвращаем найденный путь.
- Если клетка уже была посещена, мы ее пропускаем.
- Если нет — добавляем ее в visited, чтобы не заходить в нее снова.
- Перебираем четыре направления движения: вправо, вниз, влево, вверх.
- Если соседняя клетка (nx, ny) является проходом (0) и не выходит за границы лабиринта, она добавляется в стек.
- Важно: Путь path + [(nx, ny)] создаёт новую копию пути для каждого направления.
- Если все возможные пути исчерпаны и стек пуст, значит, выхода нет.

Выполнение задачи.

Для реализации и выполнения данного алгоритма был задействован язык Python. Были реализованы следующие программные блоки: алгоритмический блок и тестовый блок работы алгоритма.

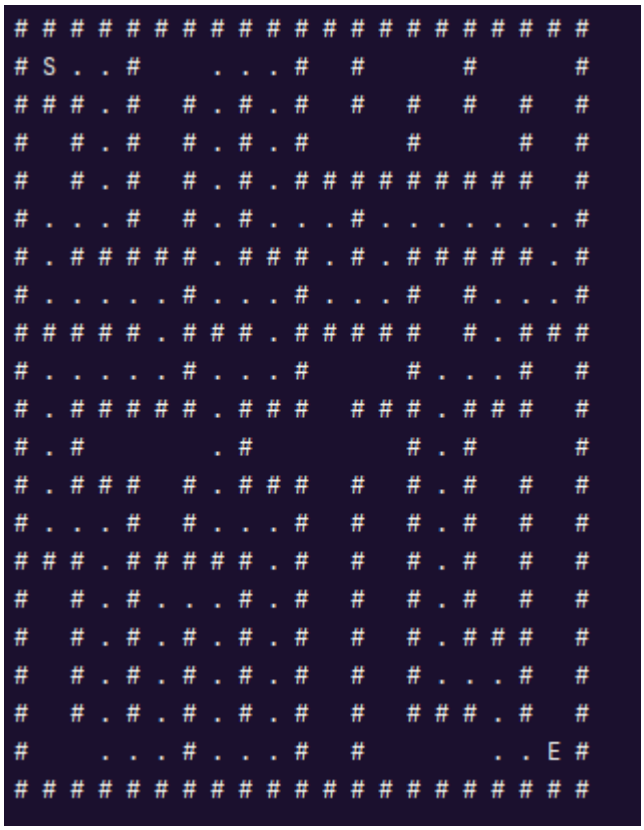
Блок кода отвечающий за реализацию лабиринта:

```
def generate_maze(n, max_path_width, straight_galleries, concentration_squares):
    maze = np.ones((n, n), dtype=int)
    def carve_passages(x, y):
        directions = [(0, 2), (0, -2), (2, 0), (-2, 0)]
        random.shuffle(directions)
        for dx, dy in directions:
            nx, ny = x + dx, y + dy
            if 1 <= nx < n - 1 and 1 <= ny < n - 1 and maze[nx, ny] == 1:
                maze[nx - dx // 2, ny - dy // 2] = 0
                maze[nx, ny] = 0
                carve_passages(nx, ny)
    start_x, start_y = 1, 1
    exit_x, exit_y = n - 2, n - 2
    maze[start_x, start_y] = 0
    carve_passages(start_x, start_y)
    maze[exit_x, exit_y] = 0
    return maze
```

Блоки кода отвечающие за тестирование:

```
size = 21
maze = generate_maze(size, 2, True, True)
solution = solve_maze(maze)
for i in range(size):
    for j in range(size):
        if (i, j) == (1, 1):
            print("S", end=" ")
        elif (i, j) == (size - 2, size - 2):
            print("E", end=" ")
        elif (i, j) in solution:
            print(".", end=" ")
        else:
            print("#" if maze[i, j] == 1 else " ", end=" ")
    print()
```

Результаты работы программы выводится в виде графика зависимости и приведены ниже. Так же результаты тестирования в виде строк данных хранятся в data.txt и приведены ниже.



Заключение.

В рамках лабораторной работы был разработан генератор лабиринта размером N × N с учетом заданных условий: наличие единственного выхода, тупиков, минимальной ширины прохода и доступности всех областей.

Для нахождения выхода из лабиринта был реализован алгоритм поиска в глубину (DFS), который эффективно прокладывает путь от входа к выходу, но не всегда гарантирует кратчайшее расстояние.

При необходимости алгоритм может быть заменен на поиск в ширину (BFS), который обеспечит нахождение кратчайшего пути.

Если важна скорость извлечения минимума — бинарная куча.

Если нужно часто объединять кучи — биномиальная куча предпочтительнее.

Так же в данном случае используется язык C в качестве основного для реализации куч и их тестирования, так как на языке Python перегружается стек и работа программы происходит в разы дольше, чем на C.

Ещё по графикам видно, что скорость работы функций Search и Remove происходят довольно быстро (доли микросекунды), поэтому они выводятся как равные 0, что само собой не так.