Автономная некоммерческая организация высшего образования
«Русский университет метатехнологий»
Кафедра программных систем

Направление подготовки: 09.03.04 Программная инженерия
Профиль подготовки: Проектирование и разработка программных систем

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА К КУРСОВОЙ РАБОТЕ
на тему:

«Оптимальное исследование графа с доступом к незакреплённой карте»

Выполнил:
студент группы ПС–21
очной формы обучения
Кугелев Михаил Григорьевич
_____ / Кугелев М.Г.
подпись расшифровка
Руководитель:
доцент, канд. физ-мат. наук
_____ / Козлов А.И.
подпись расшифровка

Йошкар-Ола
2025

Оглавление

Введение

Развитие робототехники и её широкое использование во всём мире стало причиной решения множества проблем, связанных с безопасностью производств, затратами на рабочую силу, эффективностью выполнения той или иной работы и других. Но вместе с этим появилось ещё больше задач, которые направлены на оптимизацию и повышение эффективности работы. Одной из популярных задач для роботов является исследование неизвестной местности.

В данной работе рассматривается проблема локализации автономного робота на заранее известной, но незакреплённой карте местности [1]. Представлением неизвестной местности в нашей задаче будет являться неориентированный взвешенный связный граф $G = (V, E)$, где V – количество вершин, а E – количество ребер. В условиях отсутствия GPS или других систем абсолютного позиционирования, робот не знает своего точного положения на предоставленной карте. Его цель – определить свою позицию, используя исключительно собственные наблюдения о пройденном пути. Находясь в вершине, роботу известны только доступные пути и их веса, а также степени вершин, в которых он находится и в которые он переходит. Информация о ещё не пройденных рёбрах и их стоимости ему неизвестна.

Таким образом, центральной задачей является разработка и реализация алгоритма локализации робота на незакреплённом графе. Успешная локализация позволит роботу определить своё начальное положение на карте, фактически закрепив её. Это, в свою очередь, открывает путь для последующего применения различных алгоритмов обхода графа, например, для оптимального исследования всей местности, минимизируя повторные обходы рёбер. Однако в рамках данной работы акцент делается именно на этапе локализации как на фундаментальной предпосылке для дальнейшей автономной деятельности.

Целью данной работы является нахождение оптимального алгоритма для локализации робота на неизвестном графе при наличии незакреплённой карты, а также реализация соответствующего программного обеспечения. В качестве основы для решения поставленной задачи рассматривается подход, основанный на последовательном сопоставлении наблюдаемых роботом признаков пути (степеней вершин и весов рёбер) с гипотетическими последовательностями на известной карте.

В задачи работы входит:

1. Обзор существующих алгоритмов для исследования графа.
2. Определение оптимального алгоритма или алгоритмов для исследования графа в условиях доступности незакреплённой карты. Доказательство корректности и сложности алгоритма или алгоритмов.
3. Определить наиболее удобную среду программирования и написать программу для реализации алгоритма.
4. Тестирование алгоритма, в том числе на реальных данных.
5. Разработка интерфейса для удобного использования программы.

1. Аналитическая часть

1.1. Обзор существующих алгоритмов

1.1.1. Обход в глубину.

Разберём проблему исследования с картой и без карты [2]. Для начала введём понятия. A – алгоритм исследования связного неориентированного графа. M – класс алгоритмов исследования графов с картой и чувством направления. M' – класс алгоритмов исследования графов без карты. M^* - класс алгоритмов исследования графов с картой, но без чувствительности к направлению. Под этим мы подразумеваем, что робот, находясь в вершине v, не знает других концов свободных инцидентных рёбер. Например, $A \in M'$ значит, что мы используем алгоритм A для решения задачи для класса графов M'.

Чтобы изолировать влияние заранее имеющейся информации о карте графа на стоимость обхода, мы вводим для любого алгоритма обхода A, работающего без знания карты, и любого графа G индекс, называемый чувствительностью к отсутствию карты (map ignorance sensitivity) $mis(A, G)$. Он определяется как отношение стоимости обхода, выполняемого алгоритмом A на графе G, к минимальной стоимости обхода графа G среди всех алгоритмов, работающих с картой, но без чувства направления. Нижняя граница значений $mis(A, G)$ по всем графам показывает, насколько данная процедура уязвима к отсутствию именно этой информации.

$$mis(A, G) = \frac{C(A \in M')}{C(A \in M^*)}.$$

Аналогично, чтобы оценить влияние чувства направления на эффективность обхода, вводится индекс – чувствительность к отсутствию чувства направления (direction ignorance sensitivity) $dis(A, G)$ для алгоритма A, работающего с картой, но без чувства направления. $dis(A, G)$ - отношение

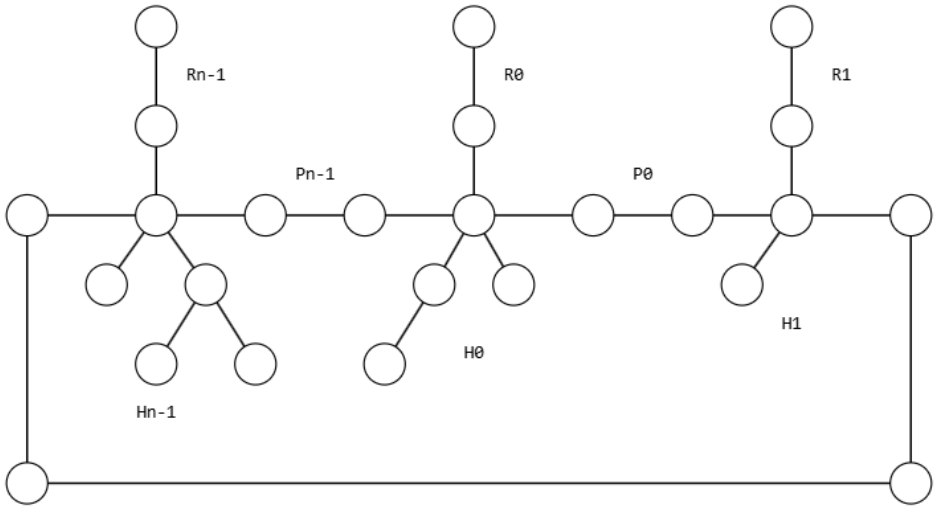
стоимости обхода алгоритма A на графе G к минимальной стоимости обхода графа G среди всех алгоритмов, имеющих и карту, и чувство направления.

$$dis(A,G) = \frac{C(A \in M^*)}{C(A \in M)}$$

Теорема 1.1. Для любого алгоритма $A \in M'$, $mis(A) \geq 2$.

Теперь докажем, что для любого $A \in M'$ стоимость будет составлять не менее 2. В худшем случае обход графа займёт как минимум 2 обхода каждого ребра при обходе с картой и в два раза больше при обходе без карты.

Для доказательства нижней границы $mis(A) \geq 2$ строится семейство графов S – n-лучевых солнц, сложного случая для обхода. Данный граф состоит из: *пересечений* – дерево высоты 2 с центром в точке v_i, где v_i имеет 4 ребра; *кодов* – структура H_i, исходящая из каждой вершины v_i, которая кодирует позицию v_i, например, через бинарное дерево; *отрезок* – линия P_i длины n, соединяющая пересечения; *лучи* – линии R_i произвольной длины, соединённые с пересечениями. Граф содержит единственный цикл, образованный отрезками P_i. Наш граф S будет именно таким.



n-лучевое солнце.

Докажем, что обход графа с картой, но без чувства направления будет стоить $2m'(S) + n^2 + O(nlogn)$, где $m'(S)$ – сумма длин всех лучей графа S. Работа алгоритма делится на фазы, в нулевой фазе алгоритм сначала должен найти пересечение, то есть вершину со степенью 4 и не имеющую соседей с вершиной степени 1. Стоимость поиска этой вершины займёт $n + O(1)$, если начальная точка находится на луче R, то стоимость будет $|R| + O(1)$, а при движении обратно уже $2|R| + O(1)$. После достижения пересечения v_0 робот начинает исследование всех смежных структур: кодов со стоимостью $O(logn)$, отрезки со стоимостью n и лучи со стоимостью $2|R_i|$. Таким образом общая стоимость исследования всего графа будет составлять $2m'(S) + n^2 + nlogn$. Мы получили n^2 и $nlogn$, так как количество структур кодов и отрезков n следовательно, умножили на n. $m'(S)$ уже содержит эту поправку.

Обход графа без карты из-за динамического построения будет иметь стоимость $4m'(S) + 2n^2 - O(n)$. Такая стоимость складывается из-за того, что робот не имеет карты, а соответственно он не знает заранее, куда вести лучший путь и здесь можно ввести противника - абстракцию, моделирующую наихудший сценарий для алгоритма. Например, оказавшись в точке ветвления противник может отправить робота исследовать один и тот же луч дважды, поэтому стоимость, по сравнению с алгоритмом, имеющим доступ к карте, возрастает как минимум в 2 раза. Отсюда

$$mis(A) = \frac{4m'(S) + 2n^2 - O(n)}{2m'(S) + n^2 + O(nlogn)} \geq 2.$$

Теорема 1.2. Для любого алгоритма $A \in M^*,\ dis(A) \geq 2$.

Для доказательства нижней границы $dis(A) \geq 2$ введём ещё одно семейство графов – толстая линия. Толстая линия – это линия L длиной n с вершинами $V(L) = \{v_0, v_1, \dots, v_n,\ x_1, \dots, x_n,\ y_1, \dots, y_n\}$ и рёбрами $E(L) = \{[x_i, v_{i-1}], [x_i, v_i], [y_i, v_{i-1}], [y_i, v_i] : 1 \leq i \leq n\}$. Заметим, что толстая линия длины n – это Эйлеров граф с $4n$ рёбрами, то есть граф, в котором существует

путь, который проходит по всем рёбрам только один раз. То есть при работе алгоритма $A \in M$, у которого есть карта и чувство направления оптимальным путём будет являться как раз этот Эйлеров путь, то есть стоимость такого алгоритма будет равняться $4n$.

Теперь рассмотрим алгоритм без чувства направления. В этом случае нам опять необходимо ввести противника, который будет моделировать худший случай для исследования графа, ведь без чувства направления робот, находясь в вершине v_0, не может определить куда ведут инцидентные рёбра. В таком случае при попытке продвинуться из v_0 в v_1 противник построит такой путь

$$v_0 \to x_1 \to v_0 \to y_1 \to v_0 \to x_1 \to v_1.$$

То есть наш робот совершит как минимум 6 шагов до того, как достигнет вершины v_1. Если мы возьмём толстую линию длины n, начнём путь с середины линии, то для каждого из внутренних $n-3$ циклов мы потратим как минимум 6 шагов. Мы не взяли два крайних цикла в линии, так как для них не гарантируется полный обход, робот по ним пройдёт как минимум 2 шага. В таком случае стоимость обхода составит

$$6(n-3) + 2 \cdot 2 = 6n - 18 + 4 = 6n - 14.$$

Но это стоимость только для обхода линии в одну сторону, с другой стороны у нас могут остаться непройденные рёбра, поэтому мы потратим ещё $2n+2$ шага на возвращение ($2n$ шагов) и обход последнего цикла (2 шага).

$$(6n - 14) + (2n + 2) = 8n - 12.$$

Итого мы получили, что алгоритм без чувства направления, из-за повторных обходов рёбер, потратит минимум $8n - 12$ шагов на обход всего графа.

$$dis(A) = \frac{8n - 12}{4n} \geq 2.$$

Лемма 1.1. Если $A \in M'$, то $mis(A, G) \leq dis(A, G)$ для любого графа G.

Так как для каждого графа G стоимость его обхода будет составлять $2E(G)$, мы можем определить следующую лемму:

Лемма 1.2. $dis(DFS) \leq 2$.

В соответствие с теоремами 1.1, 1.2 и леммами 1.1, 1.2 мы можем утверждать, что алгоритм обхода в глубину является самым эффективным алгоритмом обхода графа для сценария отсутствия карты, и самым эффективным для сценария наличия карты, но отсутствия чувствительности к направлению.

1.1.2. Случайное блуждание

Рассмотрим ещё один метод исследования неориентированных связных графов – случайное блуждание, и установим его сложность. Этот алгоритм не использует карту.

Пусть в нашем графе n – количество вершин и m – количество рёбер. Робот, реализующий случайное блуждание, в каждой вершине v равновероятно выбирает одно инцидентных рёбер и переходит по нему к соседу.

Введём ключевую метрику сложности блуждания. Пусть $C_v(G)$ – это ожидаемое число шагов, которое требуется случайному блужданию, стартующему из вершины v, чтобы впервые посетить все вершины графа. Тогда полное время покрытия характеризует «худший» стартовый сценарий:

$$C(G) = \max_{v \in V} C_v(G).$$

Например, для простого пути P_n, где P_n – линия, можно решить рекуррентное соотношение для ожидаемого времени достижения концов пути. В таком случае робот с вероятностью $\frac{1}{2}$ будет двигаться либо влево, либо вправо. Обозначим через $h(i)$ ожидаемое число шагов до достижения одним

из конечных узлов (вершин 1 или n) при старте из внутренней вершины i ($1 < i < n$).

$$h(i) = 1 + \frac{1}{2}h(i-1) + \frac{1}{2}h(i+1). [3, \text{с.}51]$$

$$h(i) = 1 + \frac{h(i-1) + h(i+1)}{2}, h(1) = h(n) = 0.$$

Решение этого уравнения даёт:

$$h(i) = (i-1)(n-i).$$

В таком случае роботу необходимо сначала дойти до одного из концов графа, а потом пройти к другому, так мы гарантированно обойдём граф и число необходимых шагов будет равна в среднем $(n-1)(n-1) = (n-1)^2$. Именно поэтому для пути P_n мы получаем:

$$C(P_n) = (n-1)^2.$$

Для n-узлового графа K_n задача сводится к классической задаче «сбора купонов» [4, с.4]: при уже собранных i разных вершинах вероятность встретить новую равна $\frac{(n-i)}{(n-1)}$, что даёт:

$$C(K_n) = \sum_{i=1}^{n-1} \frac{n-1}{n-i} = n\big(\ln n + O(1)\big).$$

Базовая оценка, полученная Алелиунасом [5], гласит, что для любого связного графа стоимость обхода будет равна

$$C(G) \leq 2m(n-1).$$

1.1.3. Локализация робота на графе.

Следующий алгоритм обоснован тем, что, имея предварительно известную, но незакреплённую карту графа, робот способен локализовать себя на ней. Это позволяет ему получить полную информацию о своей позиции в графе и о структуре окружения ещё до того, как он полностью исследует всю карту. Такой подход является критически важным для применения в условиях,

где, например, использование GPS или других абсолютных систем позиционирования невозможно или нежелательно, например, в помещениях или под землей, как это описано в работе [6].

В основе данного метода локализации лежит сравнение локально наблюдаемой роботом последовательности признаков с предварительно сгенерированными гипотетическими последовательностями, соответствующими каждой возможной начальной позиции на карте. Для локализации роботу необходимо построить последовательность Q, состоящую из весов пройденных рёбер $w_{q,k}$ и степеней вершин $d_{q,k}$, в которых он находится. Длина этой наблюдаемой последовательности n соответствует числу шагов, которые совершит робот до полной локализации.

$$Q = \{(w_{q,k}\ , d_{q,k})|k = 1, \dots, n\}.$$

Параллельно с этим, для каждой потенциальной вершины на предварительно известной карте формируются такие же гипотетические последовательности M. Каждая из этих вершин на незакреплённой карте представляет собой гипотезу о текущем местоположении робота. В начале процесса локализации робот равновероятно предполагает, что он может находиться в любой из вершин графа, которые соответствуют начальной наблюдаемой степени.

$$M_j = \{(w_k\ , d_k)|k = 1, \dots, n\}$$

По мере продвижения робота и формирования наблюдаемой последовательности Q, алгоритм итеративно сравнивает Q с каждой из гипотетических последовательностей M_j. На каждом шаге происходит фильтрация гипотез: отбрасываются те, чья сформированная последовательность не совпадает с наблюдаемой последовательностью Q по весам рёбер и степеням вершин. Этот процесс продолжается до тех пор, пока не останется одна единственная уникальная гипотетическая

последовательность. Вершина, соответствующая этой последовательности, объявляется местоположением робота.

$$H_k = \left\{ \{ v_i, V_{M,i}, p_i \} | i = 1, \dots, n_{H_k} \right\}$$

H_k – множество гипотетических вершин, в которых может находится наш робот;

v_i – гипотетическая вершина;

$V_{M,i}$ – множество вершин пути;

p_i – текущая вероятность соответствия гипотетической вершины с реальной.

1.2. Название выбранного алгоритма

1.2.1. Доказательство корректности алгоритма.

Корректность алгоритма локализации основывается на предположении, что последовательность наблюдаемых роботом признаков (степени вершин и веса рёбер) является уникальной для его истинного пути на известной карте. Это ключевое условие, которое позволяет роботу однозначно идентифицировать своё положение.

Если граф карты является сильно асимметричным (т.е. нет двух различных путей, начинающихся в разных вершинах, которые генерируют абсолютно идентичные последовательности степеней вершин и весов рёбер на протяжении всей необходимой длины), и робот успешно перемещается по графу, тогда алгоритм корректно локализует робота.

В начале алгоритма множество гипотез H_0 включает все вершины графа, чья степень совпадает со степенью начальной вершины робота. Это гарантирует, что истинное начальное положение робота всегда включено в начальный набор гипотез, если робот не совершает ошибку в первом наблюдении своей степени.

На каждом шаге перемещения робота (и соответствующего обновления гипотез) поддерживается следующий инвариант: истинное текущее местоположение робота всегда остаётся активной гипотезой, если только наблюдаемая последовательность не является аномальной, например, из-за ошибки датчиков.

Когда робот совершает переход, он наблюдает эту тройку признаков $\{d_{from}, w, d_{to}\}$. Для каждой активной гипотезы $\{v_i, V_{M,i}, p_i\}$, алгоритм ищет такого соседа $d_{neighbor}$ вершины d_{from}, что переход $\{d'_{from}, w', d'_{neighbor}\}$ на гипотетической карте точно соответствует наблюдаемой тройке, то есть $w = w'$ и $d_{to} = d'_{neighbor}$. Только те гипотезы, которые могут точно воспроизвести наблюдаемый переход на карте, сохраняются или порождают новые активные

гипотезы. Все остальные гипотезы отбрасываются. Поскольку истинный путь робота всегда точно соответствует структуре графа, его истинная гипотеза всегда будет успешно проходить эту фильтрацию.

По мере того, как робот продолжает двигаться и накапливать наблюдения, последовательность Q становится длиннее. Если граф является достаточно асимметричным, то постепенно всё больше и больше ложных гипотез будет отбрасываться, поскольку их гипотетические пути не смогут точно соответствовать всей наблюдаемой последовательности Q.

Алгоритм завершается успешно, когда остаётся ровно одна активная гипотеза. В силу инварианта, эта единственная оставшаяся гипотеза должна быть истинным местоположением робота. Если граф асимметричный, то рано или поздно все ложные гипотезы будут отфильтрованы, оставляя только истинную.

Однако, если в графе существуют топологически или структурно идентичные подграфы, которые генерируют одинаковые последовательности степеней вершин и весов рёбер, то локализация может быть неоднозначной, и алгоритм может не сойтись к единственной гипотезе, либо сойтись к неверной.

Таким образом, алгоритм является корректным при условии асимметричности графа и точности наблюдений, что позволяет ему эффективно фильтровать неверные гипотезы, как это следует из принципов, описанных в работе [6] для графо-ориентированной проприоцептивной локализации.

1.2.2. Трудоёмкость алгоритма.

Для оценки эффективности предложенного алгоритма локализации мы проанализируем его вычислительную сложность. Мы будем использовать нотацию "О-большое" для описания верхней границы роста времени выполнения алгоритма в зависимости от размера входных данных.

На начальном этапе работы алгоритма происходит формирование первичного набора гипотез. В этой фазе алгоритм итерирует по всем вершинам графа доступной карты, чтобы отобрать те, которые соответствуют начальным наблюдаемым характеристикам, в нашем случае степени текущей вершины робота. Если общее количество вершин в графе равно N, то операция просмотра всех вершин занимает время, пропорциональное N. Создание начальных объектов гипотез для каждой подходящей вершины также пропорционально их количеству, которое в худшем случае может достигать N. Таким образом, вычислительная сложность фазы инициализации составляет $O(N)$.

Наиболее критичной для производительности алгоритма является фаза обновления гипотез, которая выполняется после каждого нового наблюдения (шага робота). Пусть K обозначает текущее количество активных гипотез, а D_{max} - максимальная степень вершины в графе. L - текущая длина последовательности наблюдений, то есть количество шагов, пройденных роботом.

На каждом шаге обновления алгоритм выполняет следующие основные операции:

1. Алгоритм перебирает каждую из K текущих гипотез.
2. Для каждой гипотезы алгоритм рассматривает её соседей. В худшем случае, если гипотетическая вершина имеет максимальную степень D_{max}, то необходимо обработать до D_{max} соседних вершин.
3. При формировании новой гипотезы для каждого потенциального следующего состояния, алгоритм конструирует новый путь наблюдений, копируя предыдущий путь и добавляя к нему текущее наблюдение. Длина этого пути L растет с каждым шагом робота. Операция копирования пути длиной L занимает время, пропорциональное L.

Таким образом, на одном шаге обновления гипотез, для каждой из K гипотез, мы можем выполнить до D_{max} операций, каждая из которых включает копирование пути длиной L. Это приводит к вычислительной сложности $O(KD_{max}L)$ для одного цикла обновления. В худшем случае, количество гипотез K может быть равно общему количеству вершин N, а максимальная степень D_{max} также может быть близка к N для плотных графов. Следовательно, сложность одного шага обновления в худшем случае составляет $O(NNL) = O(N^2L)$.

Общая вычислительная сложность алгоритма зависит от количества шагов, которые совершает робот, обозначим их как M. На каждом из этих M шагов происходит вызов фазы обновления гипотез. Важно отметить, что длина последовательности наблюдений L увеличивается с каждым шагом робота, становясь равной номеру текущего шага. То есть, на первом шаге $L = 1$, на втором $L = 2$, и так до M-ого шага, где $L = M$. Следовательно, суммарная вычислительная сложность для M шагов будет представлять собой сумму сложностей каждого отдельного шага:

$$\sum_{i=1}^{M} O(N^2 i).$$

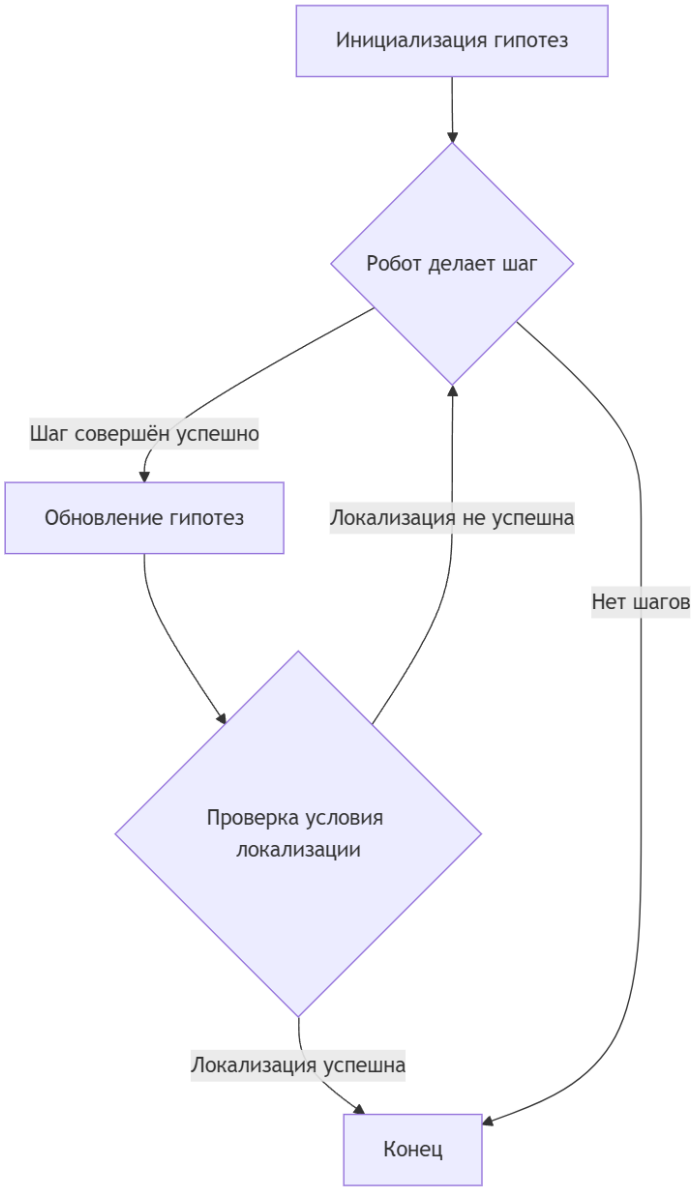
Вынося константу $2N$ за сумму, получаем:

$$O\left(N^2 \sum_{i=1}^{M} i\right).$$

Сумма первых M натуральных чисел известна как арифметическая прогрессия и равна $\frac{M(M+1)}{2}$. В рамках нотации "О-большое" мы отбрасываем константные множители и младшие члены, поскольку нас интересует асимптотическое поведение функции при больших значениях M. Таким образом, $\frac{M(M+1)}{2}$ асимптотически эквивалентно M^2.

Следовательно, общая вычислительная сложность алгоритма локализации в худшем случае составляет $O(N^2M^2)$, что свидетельствует о том, что алгоритм локализации обладает полиномиальной вычислительной сложностью.

1.2.3. Блок-схема алгоритма



2. Конструкторская часть

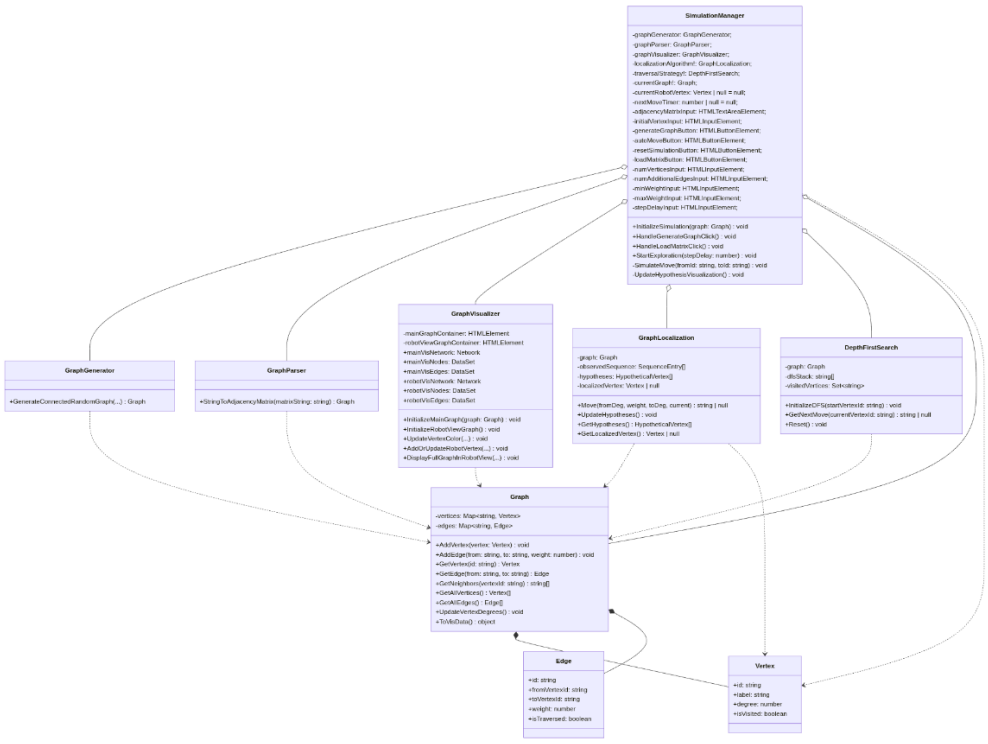
2.1. Выбор языка программирования

При выборе языка программирования я руководствовался следующими соображениями: язык должен поддерживать объектно-ориентированную парадигму программирования, для него должны существовать визуальные библиотеки для отображения графов, пользователю должно быть удобно использовать конечный продукт, а также на нём должно быть удобно написание интерфейсов, которые необходимы для взаимодействия с пользователем.

Таким образом было принято решение использовать язык программирования TypeScript, так как он соответствует требованием, высказанным выше.

2.2. Структура приложения

Приложение разработано с использованием объектно-ориентированного подхода, что позволяет эффективно управлять взаимосвязями между сущностями графа и алгоритмами, работающими с ним.



Приложение состоит из нескольких основных классов, каждый из которых реализует специализированную логику. Центральными сущностями являются Graph, Vertex и Edge, которые формируют базовую модель графа. Класс Graph управляет коллекциями вершин и рёбер, предоставляя методы для их добавления, удаления, получения соседей и так далее. Каждый Vertex хранит свой уникальный идентификатор и степень. Edge фиксирует начальную и конечную вершины, а также вес ребра, что позволяет работать с взвешенными графами.

Для решения специализированных задач на графе предусмотрены классы GraphLocalization и DepthFirstSearch. GraphLocalization реализует алгоритм локализации робота на известной, но незакреплённой карте. Он поддерживает множество гипотетических положений робота, обновляя их на основе наблюдаемой последовательности переходов. Класс принимает в конструкторе экземпляр графа и начальную вершину, из которой робот начинает наблюдения. Метод InitializeHypotheses формирует начальный набор возможных местоположений, фильтруя их по начальной степени вершины. Ключевая функция Move обрабатывает каждое новое наблюдение робота (степень исходной вершины, вес пройденного ребра, степень конечной вершины), вызывая внутренний метод UpdateHypotheses. Этот метод фильтрует и обновляет набор гипотез, отбрасывая те, которые не соответствуют наблюдаемым данным, и формируя новые на основе совпадающих соседей. Локализация считается успешной, когда остаётся ровно одна активная гипотеза.

DepthFirstSearch отвечает за реализацию основной стратегии обхода роботом графа. Его основная задача — реализовать алгоритм поиска в глубину, который начинается с указанной стартовой вершины и исследует каждую ветвь настолько глубоко, насколько это возможно, прежде чем вернуться и исследовать следующую. Этот класс непосредственно взаимодействует со структурой Graph для доступа к вершинам и их соседям, а

также для отслеживания уже посещенных вершин во избежание бесконечных циклов в циклических графах.

Присутствуют вспомогательные классы, такие как GraphParser, GraphGenerator, GraphVisualizer и SimulationManager. Они реализуют интерфейс приложения, чтобы пользователь мог удобно пользоваться приложением и алгоритмом. GraphParser позволяет преобразовывать матрицы смежности графов, для отображения и запуска алгоритма на графах, заданных пользователем. GraphGenerator предоставляет удобную функцию для генерации случайных графов, если пользователь не захочет создавать свой граф. GraphVisualizer работает с библиотекой для отображения графов, а SimulationManager управляет всей симуляцией алгоритма.

2.3. Использованные библиотеки

Для визуализации состояний графа во время работы алгоритма была выбрана библиотека vis.js, так как она имеет богатый и гибкий интерфейс настройки параметров графа, например, можно задавать вершинам текстовые описания, можно легко задавать цвета рёбрам и вершинам прямо во время выполнения программы, а также добавлять и удалять вершины, что очень важно для визуализации логики исследования роботом неизвестного пространства.

Заключение

В рамках данной курсовой работы были исследованы различные подходы к решению поставленной задачи, выделена наиболее оптимальная стратегия решения данной задачи и было разработано программное обеспечение, предназначенное для решения и визуализации задачи локализации робота на известной, но не закреплённой карте. Приложение демонстрирует практическое применение алгоритма, где локализация осуществляется на основе последовательностей степеней вершин и весов пройденных рёбер. Были спроектированы сложные тест-кейсы для тестирования алгоритма, которые были успешно им пройдены.

Список использованных источников

1. Dessmark A., Pelc A. Optimal graph exploration without good maps //Theoretical Computer Science. – 2004. – Т. 326. – №. 1-3. – С. 343-362.
2. Panaite P., Pelc A. Impact of topographic information on graph exploration efficiency // Networks: An International Journal. – 2000. – Т. 36. – №. 2. – С. 96-103.
3. Aldous D., Fill J. Reversible Markov chains and random walks on graphs, 2002 //Unfinished monograph, recompiled. – 2014. – Т. 2002.
4. Gąsieniec L., Radzik T. Memory efficient anonymous graph exploration // International Workshop on Graph-Theoretic Concepts in Computer Science. – Berlin, Heidelberg : Springer Berlin Heidelberg, 2008. – С. 14-29.
5. Aleliunas R., Lipton R. J., Lovasz L., Rackoff C. and Karp R. M. Random walks, universal traversal sequences, and the complexity of maze problems //20th Annual Symposium on Foundations of Computer Science (sfcs 1979). – IEEE Computer Society, 1979. – С. 218-223.
6. Cheng H. M., Song D. Graph-based proprioceptive localization using a discrete heading-length feature sequence matching approach //IEEE Transactions on Robotics. – 2021. – Т. 37. – №. 4. – С. 1268-1281.