Федеральное государственное бюджетное образовательное учреждение

высшего образования

«Сибирский государственный университет телекоммуникаций и

информатики»

(СибГУТИ)

02.03.02 Фундаментальная информатика и информационные технологии

Профиль: Системное программное обеспечение

(очная форма обучения)

ОТЧЕТ ПО учебной ПРАКТИКЕ

в/на

(наименование профильной организации/структурного подразделения СибГУТИ)

ТЕМА ИНДИВИДУАЛЬНОГО ЗАДАНИЯ

Выполнил:

студент института ИВТ

гр. ИС-142 / /

«27» мая 2023 г. (подпись)

Проверил:

Руководитель от СибГУТИ / /

«27» мая 2023 г. (подпись)

Новосибирск 2023

**План-график проведения**  учебной\_\_\_\_\_\_ **практики**

Вид практики

Григорьев Юрий Вадимович

Фамилия Имя Отчество студента

института Информатика и вычислительная техника , 2 курса, гр. ИС-142

Направление: 02.03.02 Фундаментальная информатика и информационные технологии

Код – Наименование направления (специальности)

Профиль: Системное программное обеспечение

Место прохождения практики

Объем практики: **108/3** часов/ЗЕ

Вид практики  ***учебная***

Тип практики ***научно-исследовательская работа (получение первичных навыков научно-исследовательской работы)***

Срок практики с "30" января 2023 г.

по "27" мая 2023 г.

Содержание практики\*:

| Наименование видов деятельности | Дата (начало – окончание) |
| --- | --- |
| 1. Общее ознакомление со структурным подразделением предприятия, вводный инструктаж по технике безопасности | 30.01.2023–01.02.2023 |
| 2. Выдача задания на практику, деление студентов на группы (если необходимо), определение конкретной индивидуальной темы, формирование плана работ | 02.02.2023–04.02.2023 |
| 3. Работа с библиотечными фондами структурного подразделения или предприятия, сбор и анализ материалов по теме практики | 06.02.2023–11.02.2023 |
| 4. Выполнение работ в соответствии с составленным планом:   1. Разработка заголовочного файла с прототипами методов работы со структурой данных 2. Разработка Makefile для автоматизации компиляции программы с библиотекой структуры данных 3. Разработка структуры данных в отдельном файле 4. Разработка программы для тестирования полученной библиотеки для работы со структурой данных 5. Отладка и форматирование кода программы | 13.02.2023 – 20.05.2023 |
| 5. Анализ полученных результатов и произведенной работы.  Составление отчета по практике, защита отчета | 22.05.2023–27.05.2023 |

\*В соответствии с программой практики

Руководитель от СибГУТИ / /

«28» \_\_\_\_01\_\_\_\_ 2023г. (подпись)

**ЗАДАНИЕ НА ПРАКТИКУ**

Реализовать программно, исследовать эффективность и описать структуру данных “R-tree”.

**ВВЕДЕНИЕ**

R-дерево (R-Tree) - это индексная структура для доступа к пространственным данным, предложенная Антонином Гуттманом (Калифорнийский университет, Беркли) в 1984 году. R-дерево допускает произвольное выполнение операций добавления, удаления и поиска данных без периодической переиндексации. При этом дерево получается сбалансированным, что является одним из важных свойств любой иерархической структуры данных.

**СТРУКТУРА R-ДЕРЕВА**

R-дерево – это сбалансированное по высоте дерево, сходное с B+-деревом, листовые узлы которого содержат ссылки на конечные объекты. Если индексная структура находится на жестком диске, то каждый узел соответствует дисковой странице. Структура разработана так, чтобы для пространственного поиска требовалось посещение как можно меньшего числа узлов. Индексная структура полностью динамическая – добавление и удаление может выполняться одновременно с поиском, и никакой периодической реорганизации структуры производить не нужно. Для организации такой индексной структуры используют пространственную базу данных, состоящую из набора записей, каждой из которых соответствует некоторый уникальный идентификатор. Этот идентификатор используют как средство ссылки на запись из индекса. В качестве идентификатора может выступать некоторое уникальное число или номер записи в файле (второй вариант предпочтительнее, так как работает быстрее, однако для него присущи некоторые недостатки, связанные с удалением записей из файла).

Если принять описанные условия, то каждый листовой узел дерева будет состоять из элементов, имеющих вид:

*[MBR, указатель\_на\_объект]*,

где *указатель\_на\_объект* ссылается на объект в памяти устройства, а *MBR* – это *n*-мерный прямоугольник, который является минимальным охватывающим прямоугольником для пространственного объекта, со сторонами параллельными осям координат. Обычно *MBR* задают в виде интервала размерности *n* с закрытыми концами *[a,b]*, где *n* - число размерностей (измерений). Внутренние узлы дерева содержат элементы, имеющие похожую структуру:

*[MBR, ссылка\_на\_потомка],*

где *ссылка\_на\_потомка* – это адрес узла низшего уровня в *R*-дереве (дочернего по отношению к данному), все записи внутри которого покрываются прямоугольником *MBR.*

И листовые узлы, и внутренние представляют собой набор из элементов описанной структуры, и даже в простейшей реализации таких элементов должно быть больше одного. Обозначим *M* как максимальное число элементов в любом узле, а *m* – минимальное. Для реализации основных алгоритмов необходимо выполнение условия *т ≤ M /* 2*.*

*R*-дерево должно удовлетворять следующим требованиям:

1. Каждый узел дерева содержит не меньше *m* и не больше *М* записей. Исключение может составлять только корень.
2. Корень, если он не является листом, содержит как минимум двух потомков. Максимальное количество элементов в корне также ограничивается значением *M*.
3. Для каждой индексной записи листового узла *MBR* является минимальным прямоугольником, который полностью вмещает в себя пространственный объект, на который ссылается запись.
4. Для каждой индексной записи внутреннего узла дерева *MBR* является минимальным прямоугольником, охватывающим все *MBR* дочерних узлов.
5. Все листовые узлы дерева расположены на одном уровне (дерево является сбалансированным).
6. Каждый объект упоминается в дереве ровно один раз.

На рис. 1 показан пример структуры *R*-дерева и проиллюстрированы отношения ограничения и перекрытия, которые могут существовать между его прямоугольниками.

Имея представление о свойствах R-дерева, можно оценить его высоту при числе элементов *N*. Из свойств, описанных выше, следует, что каждый узел дерева содержит как минимум *m* потомков. Поэтому наибольшая высота *R*-дерева, содержащего *N* индексных записей, будет не больше ⏐*logmN*⏐ - 1. При этом максимальное число узлов в таком дереве будет равно ⏐*N / m*⏐ + ⏐*N / m*2⏐ + … + 1.

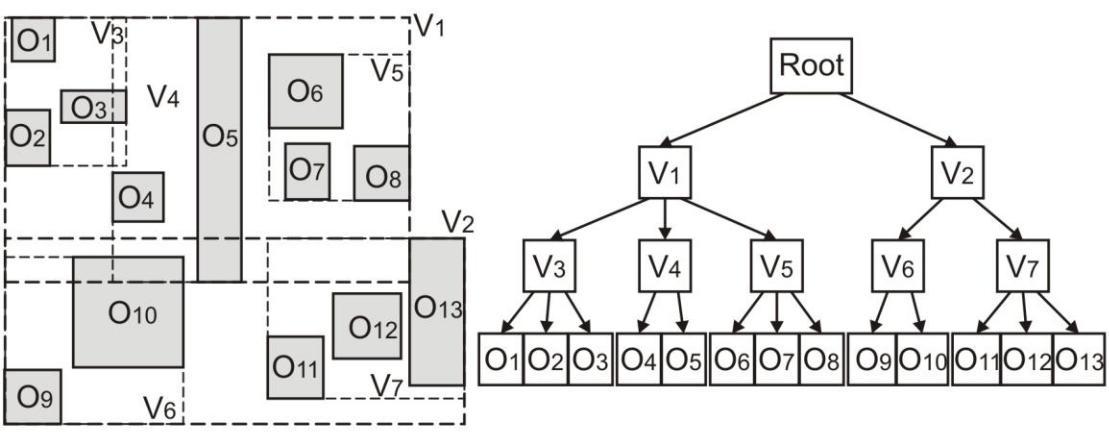


Рис. 1. Пример *R*-дерева

* худшем случае использование пространства памяти, в которой хранится индексная структура, будет *т / M*. Однако алгоритмы построения дерева разработаны таким образом, что структура будет стремиться содержать более *m* записей в узле. Это уменьшает высоту дерева и увеличивает полезное использование памяти.

**АЛГОРИТМ РАБОТЫ**

**Поиск элемента**

Алгоритм поиска в *R*-дереве похож на алгоритм поиска по *B*-дереву: он начинается в корне и опускается по нему клистовому узлу, выбирая в зависимости от заданных параметров поиска то или иное поддерево. Главное отличие состоит в том, что возможен вариант, при котором более одного поддерева текущего узла участвует в поиске: такая ситуация связана с применением метода размещения многомерных объектов, разрешающего пересекаться ограничивающим областям разных элементов. Это может привести к многократному уменьшению скорости поиска, однако алгоритмы построения и изменения дерева стараются поддерживать дерево в наиболее оптимальном виде. Пример рекурсивной процедуры поиска объектов, имеющих хотя бы одну общую точку с областью поиска *S,* в псевдокоде:

**rtree\_search** (V, S, Res) // V - текущ. вершина, S - область поиска, Res - результаты поиска

if V.kind != LEAF then

for each V’ in V // цикл по всем записям V’ в узле V

if V’.MBR \* S != 0 then // MBR записи V’ пересекается с S

rtree\_search (V’, S, Res)

end if

end if

if V.kind == LEAF then

for each O in V

if O.MBR \* S != 0 then // MBR записи O пересекается с S

Res += O

end if

end for

end if

end

Рассмотрим описанный алгоритм на примере, показанном на рис. 2. Область поиска соответствует заданному прямоугольнику *ABCD*. Первоначально процедура поиска вызывается для корня. Так как корень является внутренней вершиной, то для него выполняется первая ветка алгоритма поиска. Она проверяет узлы *V*1 и *V*2 на пересечение с заданной областью. Как нетрудно заметить, оба этих узла имеют общие точки с областью поиска, и поэтому для обоих из этих узлов рекурсивно вызывается процедура **rtree\_search**.

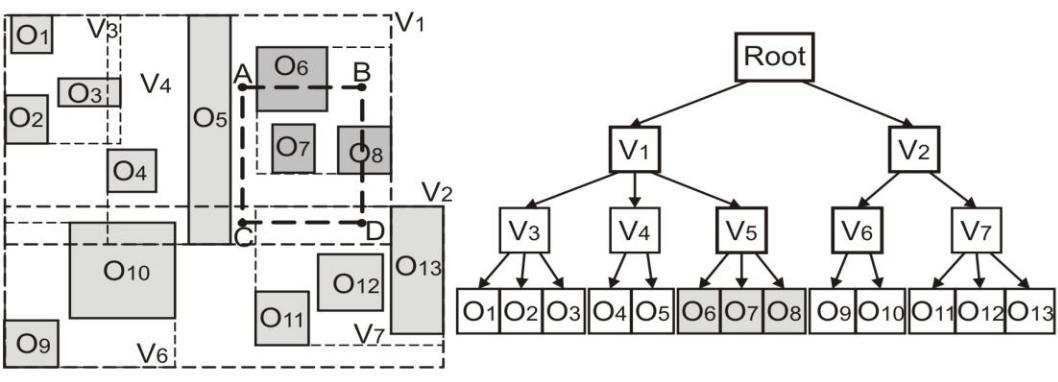


Рис. 2. Пример поиска в *R*-дереве

**rtree\_search** для вершины *V*1 перебирает элементы *V*3,*V*4, *V*5, причем только *V*5 имеет пресечение с прямоугольником *ABCD*. Поэтому вершины *V*3 и *V*4 пропускаются и далее не рассматриваются. Дальнейший вызов процедуры для вершины *V*5 выдаст в качестве результата три элемента – *O*6, *O*7, *O*8, которые и будут добавлены в множество результата *Res*. Аналогичным образом будет просмотрена ветка *V*2. Из ее потомков только *V*7 имеет общие точки с *ABCD*. Однако не один из элементов *V*7 не пересекается с областью поиска. Данная ветка поиска оказалась ложной. В результате поиска получаем список элементов, удовлетворяющих заданному запросу:

*Res = {O*6*, O*7*, O*8*}.*

**Вставка элемента**

Добавление нового объекта в *R*-дерево похоже на процедуру вставки в *B*+-дерево. Новая индексная запись добавляется в листовой узел, если узел переполняется, то происходит его деление, в результате которого у предка появляется еще один потомок. Если предок также оказывается переполненным, то и он делится дальше и так далее. Таким образом, вставка одного объекта может повлиять на структуру дерева в целом. Процедура вставки объекта представлена в следующем псевдокоде:

**rtree\_insert** (O) // O - вставляемый объект

L = choose\_leaf (O)

if L.count < M then

L += O

L’’ = NULL

else

L’’ = node\_split (L, O)

end if

rtree\_correct (L, L’’)

end

Во-первых, процедура ищет листовой узел, в который необходимо поместить данный объект (шаг 1). Процедура поиска такого листа является важным шагом, так как неправильно выбранная позиция может привести к неэффективности структуры в целом. После того, как узел для вставки выбран, в нем размещается вставляемый объект (шаг 2). Если в листовом узле *L* есть место для новой записи, объект *O* помещается в него и процедура заканчивает свою работу. В противном случае, если узел *L* уже содержит максимально возможное число записей, то происходит деление узла на два новых *L* и *L″*, которые содержат старые записи узла *L* и добавляемый объект *O*. После вставки объекта в дерево и возможного расщепления узла необходимо корректировать дерево (шаг 3). Эта процедура включает расширение границ *MBR* для текущего узла и всех его предков. Также эта процедура распространяется вверх по дереву при необходимости.

Рассмотрим алгоритмы упомянутых процедур подробнее в псевдокоде:

**choose\_leaf** (O) // O - вставляемый объект

V = root

for each V’ in V

if V.kind == LEAF then

return V

end if

Vnew = потомок, для которого MBR(V’,O)-MBR(V’) - минимальный

V = Vnew

end for

end

**rtree\_correct** (L, L”) // L - корректируемая вершина, L’’ - вершина от деления L

V=L, V”=L”

* 1. Если V является корнем, то

if V” ≠ NULL then

root = tree->root

root.data[] = V, V”

return

end if

P = V.parent

PV = запись в узле P о потомке V

rtree\_correct (PV)

if V” ≠ NULL then

PV” = новая запись о узле V”

Если число элементов в P меньше M, то

P += PV”

Иначе

P” = node\_split (P, PV”)

1. V = P V”=P”

Перейти к шагу 2

end

Как было отмечено, процедура корректировки изменяет *MBR* всех вершин дерева, которые расположены выше листа с вставленным объектом. Второй и не менее важной функций процедуры корректировки является распространение деления вершин вверх по дереву, в случае, если будет происходить переполнение на внутренних узлах дерева. В качестве параметров в процедуру передаются два новых узла, которые получились при вставке объекта в дерево. Если разбиение не произошло, то первым параметром передается старый узел, а второй параметр приравнивается в *NULL*. На первом шаге процедура заносит переданные параметры в переменные *V* и *V″*. Эти переменные будут отвечать за текущие вершины в дереве, которые необходимо исправить. После этого происходит сравнение вершины *V* корня дерева. Если данная вершина является корнем, то это означает, что изменения уже распространились до верха дерева и необходимо просто завершить процедуру корректировки. Однако стоит учитывать один момент: если после предыдущих манипуляций произошло расщепление корня на два узла (переменная *V″≠NULL*), то необходимо создать новый корень дерева, узлами-потомками которого будут *V* и *V″*. Если предыдущий пункт не выполнен, то происходит корректировка. Для этого определяется предок узла *V*, а также запись в нем об этом узле. После этого *MBR* найденной записи изменяется таким образом, чтобы включать в себя все *MBR* дочерних элементов узла *V*, но при этом не содержать лишних областей. Четвертый шаг алгоритма выполняется только в том случае, если предыдущие действия вызвали деление узла. В этом случае у нас в переменной *V″* будет находиться вершина с элементами, которые пока еще не помещены в дерево. Для этой вершины необходимо создать запись *PV″*, которая будет содержать минимальный описывающий прямоугольник для данной вершины и ссылку на саму вершину. Эту запись и нужно разместить в предке узла *V*. Однако при помещении в узел *P* записи *PV″* необходимо помнить, что данная операция может привести к переполнению и тогда придется разбивать узел *P* на два новых. После всех описанных операций в переменные *V* и *V″* заносятся новые значения *P* и *P″* соответственно, и алгоритм повторяется заново с шага 2.

**Удаление элемента**

Для того, чтобы структуру можно было считать динамической, необходима поддержка удаления уже существующих в дереве элементов, которая также должна корректировать дерево для сохранения его свойств.

**rtree\_delete** (O) // O - удаляемый объект

* 1. V = корень дерева

L = ПОИСК\_ОБЪЕКТА(V,O)

Если L = NULL, то

Завершить процедуру удаления

* 1. Удалить объект O из L

V = L

Q = пустое множество

[3] Если узел V является корнем, то Перейти к шагу 7

1. P = Parent(V)

PV = запись в узле P о потомке V

1. Если число элементов в V меньше m, то Удалить PV из P

Переместить все элементы из V в множество Q Удалить V

Иначе

Скорректировать MBR(V)

1. V = P

Перейти к шагу 3

1. Если у корня всего один потомок, то
2. Удалить корневой узел

Сделать новым корнем этого потомка

1. Вставить узлы из множества Q обратно в дерево

end

Первое, что производит процедура удаления объекта *O* из *R*-дерева, это ищет листовой узел, в котором находится данный объект. Для этого используется процедура поиска rtree\_search. В качестве параметров ей передается вершина, с которой нужно начать поиск (в нашем случае это корень) и объект поиска. Если объект не найден, то будет возвращён *NULL*. При этом необходимо завершить и процедуру удаления. На втором шаге удаляется объект *O* из узла *L* и подготавливаются временные переменные для коррекции дерева. В переменную *V* (текущая вершина для коррекции) заносится листовой узел *L*, а в переменную *Q* – пустое множество (это множество вершин, которые необходимо потом вставить в дерево заново). Далее необходимо проверить, является ли вершина *V* корнем. Если *V* – корневая вершина, то шаги 4–6 нужно пропустить и перейти сразу к седьмому пункту алгоритма. Иначе – находим предка для вершины *V*, и в нем определяем запись, ссылающуюся на *V* (*PV*). Если в рассматриваемом узле число записей меньше минимально возможного (*m*), то необходимо удалить этот узел из дерева. При этом все элементы из *V* помещаются в множество *Q* (чтобы потом снова быть размещенными в дереве, но в других вершинах) и из вершины *P* удаляется элемент, ссылающийся на удаленную вершину (удаляется *PV*). Если же записей в вершине *V* больше, чем заданный параметр *m*, то удалять вершину не нужно. При этом необходимо просто скорректировать *MBR* узла таким образом, чтобы он охватывал все прямоугольники дочерних узлов, но при этом не включал лишнего пространства (после удаления узлов вполне вероятно можно будет сузить *MBR*, который хранится в записи *PV*). После проделанных операций необходимо распространить сделанные изменения вверх по дереву (скорректировать *MBR* узла предка или, возможно, даже удалить его, если он оказался не заполненным до предела *m*). Для этого в переменную *V* заносится предок текущей вершины и повторяется алгоритм с шага 3. После того, как все изменения дойдут до корня, алгоритм продолжится с шага 7. Исходя из свойств *R*-дерева, описанных в начале данного параграфа, корень должен иметь не меньше двух потомков. Поэтому необходимо просто проверить число дочерних узлов у корня и при нахождении там всего одного потомка сделать его новым корнем дерева. Последнее, что необходимо выполнить в процедуре удаления, это вставить временно удаленные узлы из множества *Q* обратно в дерево. Данная процедура выполняется полностью аналогично описанной ранее процедуре ВСТАВКА за одним лишь исключением: вершины из множества *Q* необходимо разместить на тех же уровнях, на которых они были до процедуры удаления. Этого требования необходимо придерживаться для того, чтобы не нарушить сбалансированность дерева (одно из свойств *R*-дерева заключается в том, что все листовые узлы находятся в нем на одном уровне).

**Разбиение узла**

Изменение данных прикладной задачи требует частого изменения индексной структуры. Для добавления новой записи в уже заполненный узел *R*-дерева, содержащий *М* записей, необходимо распределить *М +* 1 элемент между двумя узлами. Процедура разбиения узла может быть вызвана не только при добавлении новых элементов в индекс, но и при перестройке дерева, при удалении ненужной записи, при обновлении данных или даже при его корректировке. Алгоритм, выполняющий деление узла, особенно важен, так как плохое разбиение может сильно затруднить операции поиска по дереву. Разбиение узла без учета критериев оптимальности построения дерева приводит к увеличению времени работы процедуры поиска конкретного объекта, а следовательно, к ухудшению работы индексной структуры в целом. При плохом разбиении узлы дерева разрастаются вдоль осей координат и захватывают много пространства, не содержащего ни одного объекта. Такой пример показан на рис. 3.11. С одной стороны вариант (а) обеспечивает нулевое перекрытие двух узлов дерева. Однако суммарная площадь этих узлов будет значительно больше самих.

**ИССЛЕДОВАНИЕ ЭФФЕКТИВНОСТИ**

фыва

**ЗАКЛЮЧЕНИЕ**

В ходе проведения работы была изучена и смоделирована структура данных «R-дерево» и тестовая программа для работы с ней. Подводя итоги, можно сказать, что главные свойства R-дерева следующие:

1. Состоит из внутренних узлов, листовых узлов и единственного корня
2. Корень содержит указатель на самую большую область в пространстве
3. Родительские узлы содержат указатели на свои дочерние узлы, чья совокупная область (их сумма) покрывает область родительского узла
4. Листовые узлы содержат данные о MBR текущих объектов
5. MBR — параметр, обозначающий минимальную ограничивающую область (рамку/прямоугольник), окружающую рассматриваемую область/объект в пространстве

Сравнивая R-дерево с Quad-tree, структурой данных, используемой для похожих целей, можно заметить следующие отличия:

1. В Quad-tree требуется оптимизация уровня листов (перебалансировка дерева), тогда как R-дерево не требует такой оптимизации
2. Quad-tree может быть реализовано поверх существующего B-дерева, тогда как R-дерево имеет структуру, отличную от B-дерева
3. Создание пространственного индекса в Quad-tree происходит быстрее по сравнению с R-деревом
4. R-дерево быстрее, чем Quad-tree, для поиска ближайшего соседа, в то время как для оконных запросов Quad-tree быстрее, чем R-tree

**СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ**

1. *Томас Х. Кормен, Чарльз И. Лейзерсон, Рональд Л. Ривест, Клиффорд Штайн.* Алгоритмы: построение и анализ, 3-е издание = Introduction to Algorithms, Third Edition. — М.: [«Вильямс»](https://ru.wikipedia.org/w/index.php?title=%D0%92%D0%B8%D0%BB%D1%8C%D1%8F%D0%BC%D1%81_(%D0%B8%D0%B7%D0%B4%D0%B0%D1%82%D0%B5%D0%BB%D1%8C%D1%81%D1%82%D0%B2%D0%BE)&action=edit&redlink=1), 2013. — 1328 с. — [ISBN 978-5-8459-1794-2](https://ru.wikipedia.org/wiki/%D0%A1%D0%BB%D1%83%D0%B6%D0%B5%D0%B1%D0%BD%D0%B0%D1%8F:%D0%98%D1%81%D1%82%D0%BE%D1%87%D0%BD%D0%B8%D0%BA%D0%B8_%D0%BA%D0%BD%D0%B8%D0%B3/9785845917942)
2. *Курносов М.Г., Берлизов Д.М.* Алгоритмы и структуры обработки информации. – Новосибирск: Параллель, 2019. – 211 с. — ISBN 978-5-98901-230-5

**ПРИЛОЖЕНИЯ**

Исходный код программы

1. Файл main.c
2. Файл rtree.c
3. Файл rtree.h

**Отзыв о работе студента**

| Григорьев Юрий Вадимович |
| --- |
| (ФИО студента) |
|  |

Уровень освоения компетенций

|  | Григорьев Юрий Вадимович | |
| --- | --- | --- |
|  | (ФИО студента) | |
|  |  | |
| Компетенции | | Уровень сформированности  компетенций |
| *ОПК-1 - Способен применять фундаментальные знания, полученные в области математических и (или) естественных наук, и использовать их в профессиональной деятельности* | |  |

отметка о зачете \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Руководитель практики от СибГУТИ:

Должность руководителя подпись ФИО руководителя

"\_\_\_" \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 20\_\_\_\_ г.