Семестровая работа по алгоритмам и структурам данных №2.

Запустив среду программирования, я создал новый Java Project назвал его "Tree(Ds)". Далее я создал новый пакет "tree" и в нем три класса "List", "Tree", "Main". Итак разберемся с классами. Прочитав задание и разобравшись с суффиксным деревом, стало понятно, что для создания такого дерева нам понадобится параметризованный список, который мы создаем сами, вручную. Параметризованный список у нас будет реализован в классе "List". Итак что такое параметризованный список? Параметризованный список -это такой список, в котором могут храниться переменные с выбранным нами параметром. В нашем случае элементами данного списка будут переменные типа Node, то есть новые узлы. Это значит что каждый узел нашего будет иметь много потомков, а точнее целый список потомков. Итак что же в классе " List"? Все как в обычном списке, который был реализованный нами ранее, просто ячейку, где хранилось какое-то значение, мы заменяем параметром. У нас имеется функция добавления нового узла "add()", функция проверки существования заданного узла "check()" и функция удаления узла "delete()". Ничего лишнего, обычный список. С этим классом все понятно, теперь переходим к классу "Tree". Здесь привычный нам внутренний класс "Node", который имеет поля letter(символ) с типом переменных char, parent (ссылка на родителя) с типом переменных Node, параметризованный список children(узлы-потомки), с параметром Node и наконец positionEnd(показывает позицию буквы) с типом переменных boolean. У нас имеется переменные save и root типа Node. Save помогает нам запоминать узел, в котором мы побывали в последний раз, а root - это корень дерева. Итак переходим к функциям. В первую очередь нам надо создать корень дерева, который не будет иметь символа, то есть он будет пустым, так как мы всегда будем начитать с него. Функция, которая создает корень дерева называется "creationRoot()". Тут просто создается новый узел и указатель на корень дерева root начинает ссылается на него. Далее идет функция "addWord()", которая будет добавлять слово в дерево. Как же происходит добавление в дерево? Сначала мы создаем новую переменную a с типом переменных char, переменную temp с типом переменных Node, которая изначально ссылается на корень дерева. Далее мы заводим цикл который будет расщеплять с нашего слова по символу с помощью стандартного метода "charAt()". Как только мы отщепляем символ мы сразу же должны проверить, его наличие в нашем дереве. Для проверки реализован отдельный метод "check()". Как работает этот метод? В параметры передается узел у которого необходимо проверить потомков. Вообще этот метод пробегается по списку, то есть по потомкам. Заводится переменная check с типом переменных boolean , которая сначала равна false. Далее проверяется наличие детей у данного узла(проверяется наличие списка, потому что он мог ранее и не создаваться и ,если он был создан, наличие начала списка( содержимое списка)). Далее мы создаем переменную temp, с типом переменных List<Node>.Node( переменная с типом переменных Node в нашем списке). Она будет пробегаться по списку детей. Далее заводится цикл который продолжается пока мы не достигли конца списка или не нашли нужный элемент. Если мы находим узел с нужным символом, то сохраняем этот узел в переменой save и возвращаем true и false в противном случае. Все мы проверили наличие символа теперь возвращаемся обратно в функцию "addWord()" . Если мы обнаружили узел с нашим символом, то мы переменной temp присваиваем save(сдвигаемся по нему), в противном случае мы проверяем был ли у нас создан список, если нет , то создаем его , потом добавляем символ в дерево и сдвигаемся по нему. Функция добавления символа реализована отдельно, она называется "addChar()" . Мы передаем в этот метод символ и далее создается новый узел. Соответствующей ячейке мы присваиваем символ, присваиваем переменной save новый узел(сохраняем, где мы были в последний раз), добавляем в список детей последнего узла наш новый узел, указываем ссылку на родителя. После того как мы добавили символ, мы проверяем, а не является ли добавляемый символ последним, если да то у последнего посещенного узла меняем positionEnd на true. Далее идет функция "wordSearch()", которая проверяет наличие этого слова в нашем дереве. Как он работает? В параметры передается слово, которое мы хотим найти. Заводится переменная a с типом переменных char, переменная check с типом boolean, которая в начальный момент времени равно false, переменная temp с типом Node, которая изначально ссылается на корень дерева. Итак мы заводим цикл, который будет расщеплять из нашего слова символы. По мере расщепления символов мы проверяем их наличие, если мы находим его в нашем дереве, то мы сдвигаемся к нему и идем дальше при этом мы проверяем , не является ли символ который мы ищем последним и positionEnd у текущего узла равна ли true(является ли символ в текущем узле последним) , если оба условия истинны, то мы нашли наше слово, а если мы не находим символ в нашем дереве, то мы выходим из цикла и такого слова следовательно нет. Далее разберем метод "addText()", который из исходного текста, который состоит из слов, разделенных пробелами, строит суффиксное дерево. Что же у нас тут происходит? Мы создаем новую строку с типом переменных String, в которую скопируется наша исходная строка без лишних пробелов в начале и в конце. Добиваемся такого результата с помощью стандартного метода "trim()". Далее мы создаем вспомогательную переменную j с типом переменных int, которая изначально равна 0. Далее мы создаем цикл, который будет брать слова из нашего текста. Цикл начинается с 0 и продолжается пока мы не достигнем конца слова. Здесь с помощью метода "substring()" мы берем слова i до j . Переменная i меняется следующим образом: так как эта переменная показывает начальный индекс, с которого мы должны начинать брать слова, сначала она равна 0, потом она изменяется на индекс первого вхождения пробела начиная с i и +1 (i сначала 0 ,потом мы изменяем каждый раз, и получается что каждый раз мы сдвигаемся на начало нового слова ). Переменная j изменяется так: если есть пробел начиная с i , то она равна первому вхождению пробела начиная с i , если же пробела нет, а это означает что мы подошли к концу слова , он равняется длине этой строки. Каждый раз мы берем слово и вызываем метод добавления слова "addWord()". И наконец последний метод удаления слова из дерева, который называется "wordDelete()". В параметры передается слово которое мы хотим удалить. Естественно мы сначала проверим, а есть ли слово которое мы хотим удалить в нашем дереве? Если да, то мы переходим к удалению. После проверки наша вспомогательная переменная save запомнила последний узел, в котором мы побывали, то есть конец слова. Мы будем двигаться с конца, поэтому создаем переменную temp с типом переменных Node , которая в начальный момент времени равна save. В самом начале мы проверяем а есть ли у конечного узла дети ? Если их нет, заводим цикл, который будет работать пока мы не достигли конца дерева или пока у нашего текущего узла не появятся дети. Так как мы в начале проверили, что у нашего узла нет детей, мы смело удаляем этот узел(используем метод который был у нас в классе "List", он удалит этот узел), потом мы сдвигаемся к отцу удаленного элемента. Сразу же проверяем : не является ли следующий узел концом другого слова, если да то мы выходим из цикла и перестаем удалять. Если же мы натолкнулись на такой узел, который имеет потомков, мы просто меняем positionEnd на false. Вот и собственно все функции. Чтобы использовать нашу структуру данных мы просто заходим в класс "Main" и вызываем нужные функции.