1. **БИОАЛГОРИТМИКА**

**12.08 Занятие 4.**

**1. Повторение**

**2. Деревья отрезков как деревья поиска.**

* 1. Полное дерево отрезков.

Определение. *Полное дерево отрезков высоты k* – это полное бинарное дерево (см. рис.1), узлам которого соответствуют отрезки, причем

1. корень дерева помечен отрезком [1, 2k];
2. если узел помечен отрезком [a+1, a+2r], где r>0, то у узла есть два сына, причем левый сын помечен отрезком [a+1, a+2r-1], а правый сын – отрезком [a+2r-1 +1, a+2r]
3. если узел помечен отрезком [a, a], то этот узел не имеет сыновей (является *листом)*

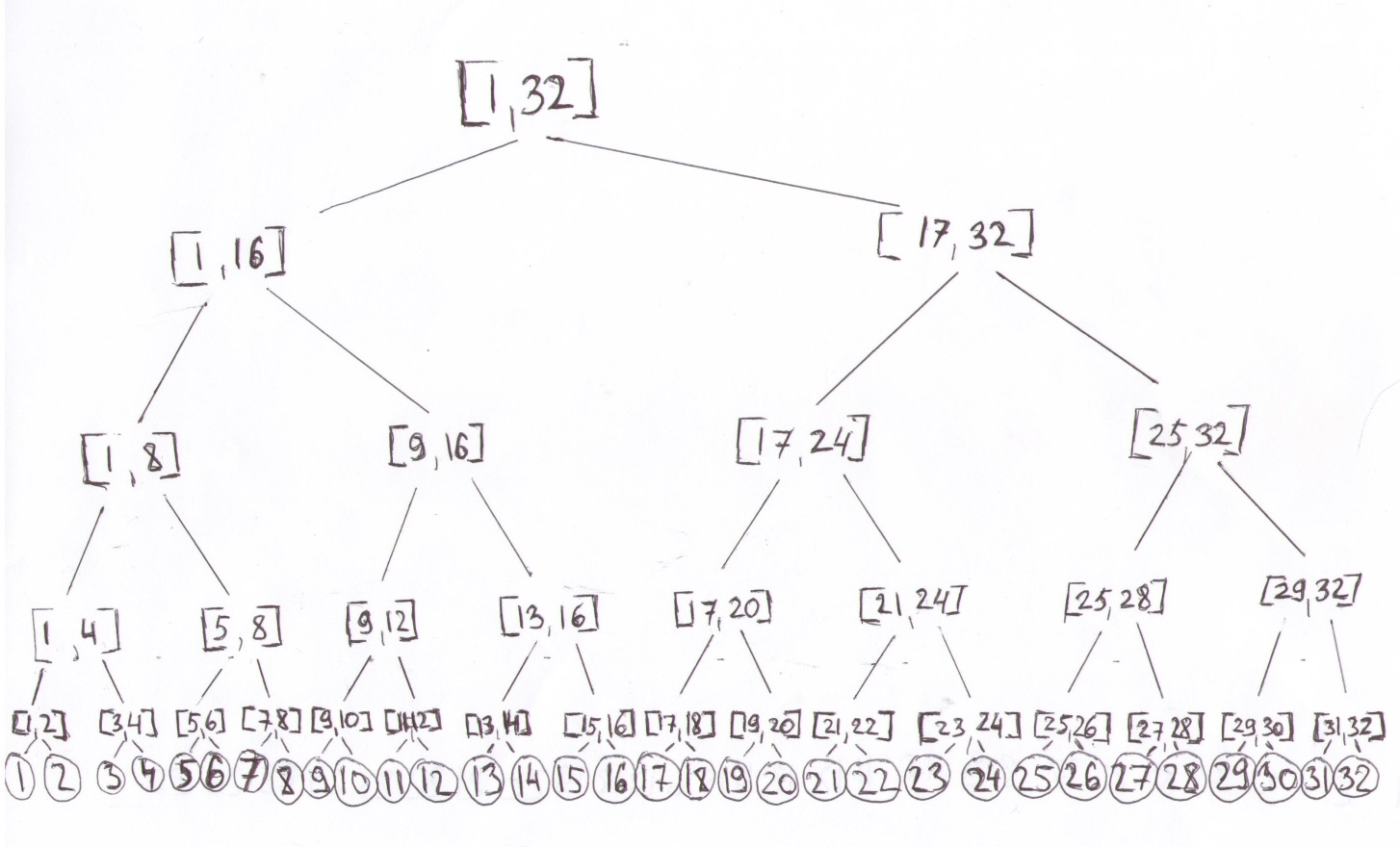


Рис.1. Полное дерево отрезков высоты 5.

Отрезки, которыми помечены узлы полного дерева отрезков, будем называть *бинарными* отрезками.

* 1. Дерево отрезков данного множества.

Пусть M – множество натуральных чисел, причем все элементы M не превосходят 2k (k – некоторое натуральное число).

*Дерево отрезков высоты k для множества M –* это полное дерево высоты *k.* Из которого удалены все узлы, соответствующие отрезкам, в которых нет элементов множества *M* (см. рис.2).

Замечание. Так как число *s* такое, что 1≤ s ≤ 2k, принадлежит k+1 бинарным отрезкам длины не более 2k, то дерево отрезков высоты k для множества M содержит не более |M|\*k узлов (|M| - количество узлов в множестве M).

* 1. Добавление и удаление узлов в дерево отрезков.

Утверждение. Пусть T - дерево отрезков высоты k для множества M; TIns - дерево отрезков высоты k для множества M+{x} (x не лежит в М); TDel - дерево отрезков высоты k для множества M-{z} (z лежит в М). Тогда как дерево Tins, так и дерево TDel могут быть получены из дерева T за время ~ k.

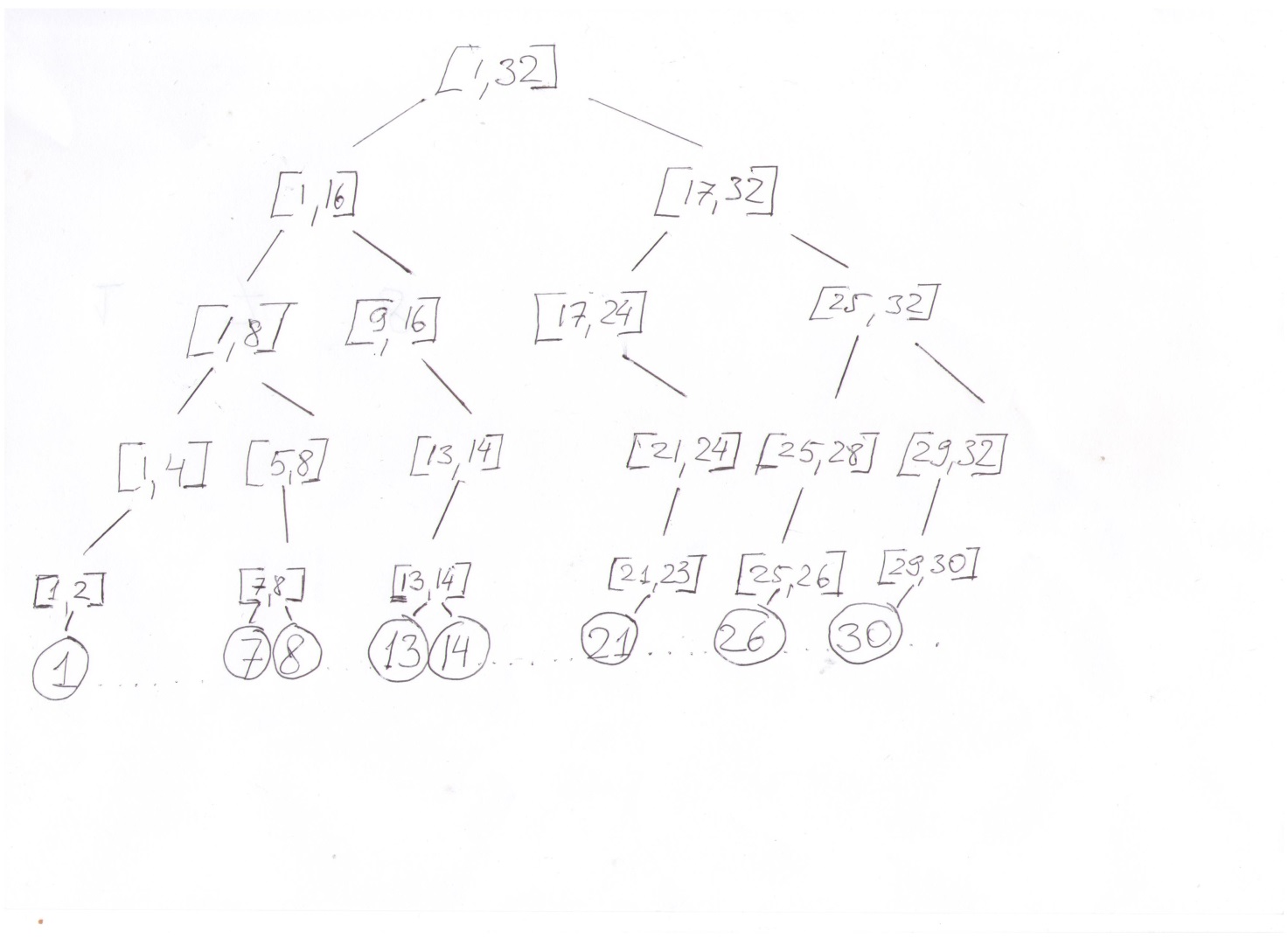


Рис.2. Дерево отрезков высоты 5 для множества {1, 7, 8, 13, 14, 21, 26, 30}

Доказательство (набросок).

(1) Удаление. Удаляем лист, соответствующий числу *z.* Далее двигаемся вверх по дереву T и удаляем узлы, соответствующие отрезкам в которых нет элементов множества M, отличных от z.

(2) Вставка. Идем от корня вниз. Обнаружив уровень, на котором нет узла, помеченного отрезком, содержащим *x,* добавляем эту вершину. Далее добавляем вершину для числа *x* на каждом следующем уровне.

Конец доказательства.

Замечание. Если появилось число за пределами диапазона [1. 2k], то можно надстроить дерево. Время надстройки логартфмически зависит от того, насколько добавляемый элемент превосходит 2k.

* 1. Обогащенные деревья отрезков.

Пусть M – множество натуральных чисел. Будем считать, что все элементы M принадлежат отрезку [1, 2k ], где k – некоторое натуральное число.

*2.4.1. Задача о сокровищах на дороге (сумма).* Для каждого из элементов множества М задано целое число («вес мешка сокровищ под столбом»). Нужно уметь для каждого допустимого отрезка, лежащего внутри отрезка [1, 2k ], назвать сумму весов сокровищ на этом отрезке.

Замечание. Количество возможных запросов ~ (2k)2.

Идея решения – использовать дерево отрезков высоты k для множества M с дополнительными пометками (*обогащенное* дерево отрезков высоты k для множества M). Это дерево будем обозначать *TSegmSum(M, k).* Исходное дерево отрезков высоты k для множества M будем обозначать *TSegm(M, k).*

Дерево *TSegmSum(M, k)* получается из дерева *TSegm(M, k)* дописыванием в каждом узле сумму весов сокровищ на соответствующем отрезке.

Замечание. Дерево *TSegmSum(M, k)* строится из дерева *TSegm(M, k)* движением от листьев к корню за время ~ количества узлов в дереве *TSegm(M, k).*

*2.4.2. Решение задачи о сокровищах на дороге (сумма).*

Пусть нужно узнать сумму сокровищ для отрезка [a, b]. Двигаясь по дереву *TSegmSum(M, k)* от корня к листьям строим разбиение [a, b] на непересекающиеся бинарные отрезки так, чтобы разбиение было минимальным, то есть так, что никакие два бинарных отрезка, входящих в разбиение, нельзя объединить в бОльший бинарный отрезок.

Сопоставим исходный тестовый отрезок [a, b] корню дерева *TSegmSum(M, k)*. Далее будем переходить от узла дерева к его сыновьям. При этом тестовый отрезок будет «дробиться» на более мелкие отрезки – по одному на каждому сына узла. А именно, тестовый отрезок узла-сына – это пересечение тестового отрезка узла-отца и бинарного отрезка узла-сына.

Если оказалось, что тестовый отрезок узла-сына совпадает с бинарным отрезком этого узла, то бинарный отрезок включается в искомое разбиение. В противном случае к узлу сыну и его тестовому отрезку применяется та же процедура.

Утверждение. Если узел не является корнем, то не более одного из его сыновей получает тестовый отрезок, не совпадающий с бинарным отрезком этого узла.

Доказательство - разберем на примере отрезка [5. 19] и обогащенного дерева, изображенного на рис. 3.

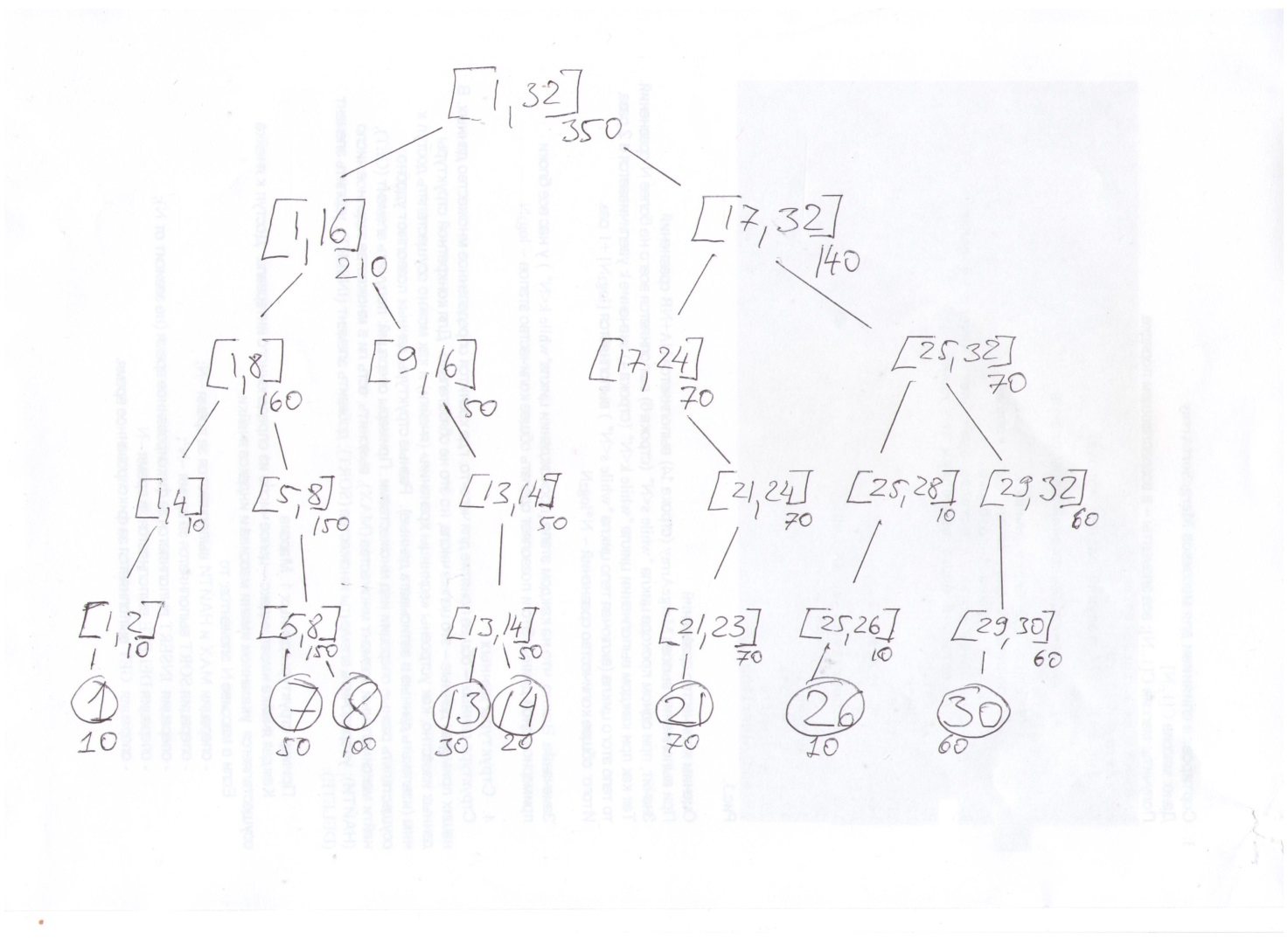


Рис.3. Обогащенное дерево для дерева отрезков, изображенного на рис.2.

Уровень 1. Узел: [1, 32]. Тестовый отрезок T(1, 32) = [5, 19]. Сыновья: [1, 16], [17. 32].

Новые тестовые отрезки: T(1, 16) = [5, 16]; T(17, 32) = [17, 19].

*Текущее разбиение:* [5, 16] + [17, 19]. Оба тестовые отрезка – не бинарные Пока искомых бинарных отрезков нет.

Текущая сумма для найденных бинарных отрезков S=0.

Уровень 2.

А. Узел: [1, 16]. Тестовый отрезок T(1, 16) = [5, 16]. Сыновья: [1, 8], [9. 16].

Новые тестовые отрезки: T(1, 16) = [5, 8]; **T(9, 16) = [9, 16]. Жирным выделен тестовый отрезок, совпадающий с бинарным.**

Б. Узел: [17, 32]. Тестовый отрезок T(17, 32) = [17, 19]. Сыновья: [17, 24], [25. 32].

Новые тестовые отрезки: T(17, 24) = [17, 19]; T(25, 32) – пустой.

*Текущее разбиение*: [5, 8] + ***[9, 16]*** + [17, 19]. Есть два небинарных отрезка (по краям) и один бинарный, входящий в искомое разбиение (выделен жирным).

Текущая сумма для найденных бинарных отрезков S= 0+ S(9, 16) = 0+50 = 50

Уровень 3.

А. Узел: [1, 8]. Тестовый отрезок T(1, 8) = [5, 8]. Сыновья: [1, 4], [5. 8].

Новые тестовые отрезки: T(1, 4) - пустой; **T(5, 8) = [5, 8].**

Б. Узел: [17, 24]. Тестовый отрезок T(17, 24) = [17, 19]. Сын: [21. 24] (на отрезке [17, 20] элементов множества M нет).

Новый тестовый отрезки: T(21, 24) – пустой.

*Текущее разбиение*: ***[5, 8]*** + **[9, 16]** . Небинарных отрезков нет – это и есть искомое разбиение.

Новый бинарный отрезок - ***[5, 8]*** (выделен курсивом).

Сумма для найденных бинарных отрезков S= 50+ S(5, 8) = 50+150 = 200

Ответ: 200.

Конец примера.

Следствие. В разбиении будет не более 2\*k бинарных отрезков, найти эти отрезки можно за время ~k. Поэтому общее время получения ответа будет ~k.

2.4.3. Задача о сокровищах на дороге (максимум). Для каждого из заданных чисел задано целое число («вес мешка сокровищ под столбом»). Нужно уметь для каждого допустимого сегмента (=отрезка) назвать максимальное сокровище на этом отрезке.

Эта задача решается так же, как задача о сумме (см. п. 2.4.1-2.4.2) , но в узлах обогащенного дерева нужно писать не сумму весов сокровищ соответствующего бинарного отрезка, а вес максимального из этих сокровищ.

2.4.4. Обновление обогащенного дерева сегментов.

Утверждение. Пусть T = *TSegmSum(M, k) -* обогащенное дерево отрезков высоты k для множества M; TIns – обогащенное дерево отрезков высоты k для множества M+{x} (x не лежит в М); TDel – обогащенное дерево отрезков высоты k для множества M-{z} (z лежит в М). Тогда как дерево Tins, так и дерево TDel могут быть получены из дерева T за время ~ k.

Доказательство – аналогично доказательству утверждения из п. 2.3. Единственное отличие – нужно пересчитывать суммы в тех узлах, которые мы проходим.

с небольшой сложностью (допускаем пропуск уровней, если цепь – УТОЧНИТЬ)

2.4.5. Вывод

Обогащенные деревья отрезков позволяют для заданного множества M и натурального число k, такого, что все элементы M не превосходят 2k , решать задачи ДОБАВИТЬ, УДАЛИТЬ, СУММАпоОТРЕЗКУ, МАКСИМУМпоОТРЕЗКУ, НАЙТИ за время ~k.

Если элементы множества M «примерно равномерно» распределены на участке [1, 2k], то можно считать, что k ~ log(|M|).

Замечание. Если это не так, например, один из элементов множества M намного больше остальных, то времени ~ log(|M|) можно достичь за счет небольшого изменения деревьев отрезков (т.н. «сжатие цепей»).

**3. Поиск точных вхождений**

3.1. Поиск вхождений одного слова: постановка задачи

Дан длинный текст (геном) длины L и короткое слово (сайт) P длины k; предполагается, что k <<L (читается «k много меньше L»).

Требуется найти все вхождения сайта в геноме.

Замечание. Слово, которое мы ищем принято называть *паттерном* (от англ. *pattern –* образец).

Наивный метод сравнения — «прикладывать» паттерн к геному, начиная с каждой позиции. Приложив паттерн к геному, побуквенно проверяем, совпадают ли буквы в паттерне и соответствующем месте генома. Если нашли несовпадение, сдвигаем начальную позицию в геноме на 1 и повторяем сравнение.

Недостаток метода: если, скажем первые 5 букв просматриваемого участка генома совпадают с началом паттерна, а 6-я буква нет, то при сдвиге начала просмотра генома на 1 мы 4 буквы будем анализировать повторно. Это приводит к оценке времени работы алгоритма поиска ~L\*k.

Упражнение. Для произвольных L и k придумайте текст длины не менее L и паттерн длиной k так, чтобы при поиске паттерна в тексте с помощью описанного наивного алгоритма выполнялось L\*(k-1)/2 сравнений букв.

Хорошо бы избежать повторного просмотра уже просмотренных букв – то есть получить алгоритм со временем работы ~L.

3.2. Поиск вхождений одного слова: идея быстрого алгоритма

Длинный текст (геном) будем обозначать G, а паттерн – P. Букву в k-й позиции генома (паттерна) будем обозначать G[k] (соответственно, P[k]). Позиции будем нумеровать, начиная с 1.

Для примеров будем использовать паттерн P=ACAGACAT.

Пусть мы «прошли» s позиций генома и паттерн приложен, начиная с позиции s+1, где s ≥0.

То есть буква G[s+1] сравнивается с буквой P[1]= А,

Пример 1. Текущая буква генома сравнивается с 1-й буквой паттерна, т.е. никакой фрагмент генома, заканчивающийся в позиции s, не является началом паттерна. В этом случае быднм говорить «обнаружено пустое начало паттерна» или «обнаружен пустой *префикс* паттерна»

Если G[s+1] ≠ P[1]= А, то мы не смотрели лишних букв и можно просто сдвинуть начало анализируемого фрагмента генома. Следующую букву - букву G[s+2] - мы снова должны сравнивать с 1-й буквой паттерна P[1]= А. Если G[s+1] = P[1]= А, то букву G[s+2] мы должны сравнивать со 2-й буквой паттерна P[2]= С. Как видим, в этом случае повторного просмотра букв нет. Сказанное будем изображать так (см. рис.4).

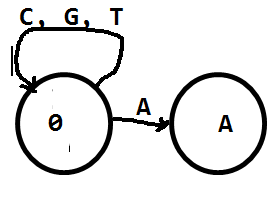


Рис.4

Каждый кружок обозначает, какой префикс в паттерне мы обнаружили (0 обозначает пустой префикс). Соответственно, следующая буква генома (в нашем случае) – буква G[s+2]) должна сравниваться с позицией в паттерне, которая следует за найденным префиксом.

Стрелки, выходящие из кружка соответствуют возможным буквам на *текущей* позиции генома. Конец стрелки указывает префикс, который будет обнаружен после сравнения.

В нашем примере после сравнению с буквами C.G,T у нас по-прежнему будет пустой паттерн, а после сравнения с буквой A мы обнаружим префикс длины 1.

Пример 2. Найден префикс длины 1 P[1:1] = A. Текущая буква генома сравнивается с 2-й буквой паттерна. Например, G[s+1] = P[1]= А и мы выполняем сравнение очередной буквы генома (это буква G[s+2]) с P[2].

Если G[s+2] = P[2]= С, то найденный префикс продлевается (теперь это префикс AC) и на следующем шаге мы продолжим сравнение с паттерном, т.е. будем сравнивать букву G[s+3] c буквой P[3]. Повторного анализа позиции G[s+2] нет.

Если G[s+2] = G или T, то ясно, что с позиции s+2 паттерн начинаться не может (паттерн начинается с A, а G[s+2] ≠ P[1]= А). Поэтому новое сравнение с 1-й позицией паттерна мы начнем с позиции s+3, при этом мы располагаем только пустым префиксом. Отметим, что мы снова избежали повторного анализа позиции s+2.

Пусть, наконец, G[s+2] = A≠ P[2]= C. В этом случае совпадение с паттерном нарушено и нам нужно сдвинуть начало исследуемого фрагмента генома в позицию s+2. Но мы уже знаем, что G[s+2]=A=P[1]! Поэтому можно сразу перейти к сравнению G[s+3] со ВТОРОЙ буквой паттерна P[2] – мы располагаем префиксом длины 1!. Схема сравнений изображена на рис. 5.

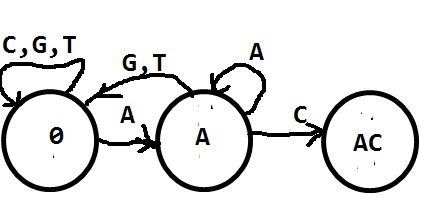


Рис.5

Как видим, во всех случаях, мы перешли к анализу позиции G[s+3], избежав повторного анализа позиции G[s+2]. При этом, c какой буквой паттерна сравнивается буква G[s+3], зависит от буквы G[s+2] (напомним, что мы рассматриваем случай G[s+1] =P[1] = A).

Если G[s+2] = G или T, то сравниваем букву G[s+3] с P[1]; если G[s+2] = A, то сравниваем G[s+3] с P[2]; если G[s+2] = C, то сравниваем G[s+3] с P[3].

Подобным образом можно рассмотреть случаи, когда текущая буква генома сравнивается с каждой из букв паттерна. Формальное правило выглядит так.

*Пусть текущая буква генома G[t] = x сравнивается с (r+1)-й буквой паттерна. Это значит, что предшествующие r букв генома совпадают с r первыми буквами паттерна (у нас есть префиксP[1..r]). То есть в позиции t генома заканчивается слово v = P[1]…P[r]x. Пусть n – такое наибольшее число, что конец слова v длины n является началом (префиксом) паттерна (но не совпадает с паттерном целиком!). Это и будет новый текущий найденный префикс паттерна. Следующую букву генома – букву G[t+1] нужно сравнивать с (n+1)-й буквой паттерна – следующей буквой после найденного префикса.*

Рассмотрим еще 2 примера.

Пример 3. Текущая буква генома G[t] = x сравнивается с 4-й буквой паттерна P[4] = G, то есть, найденный префикс – это префикс ACA (см. рис.6).

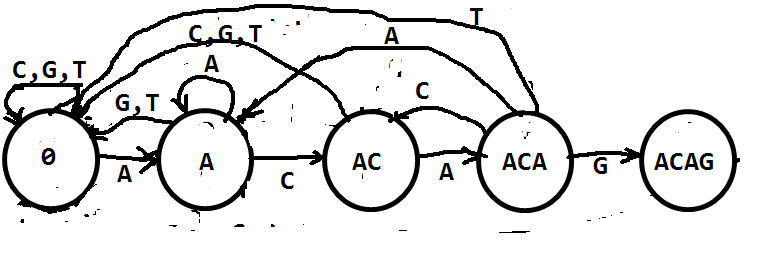


Рис.6

Если x = P[4] = G ( в геноме найдено АСАG), то следующая буква генома G[t+1] сравнивается с P[5] (*n=4,* найденный префикс – ACAG, см. определение *n* в *формальном описании*)*.*

Если x = P[4] = A ( в геноме найдено АСАA), то *n=1,* найденный префикс – Aи следующая буква генома G[t+1] сравнивается с P[2] =C.

Если x = P[4] = C ( в геноме найдено АСАC), то *n=2, найденный префикс –* AСи следующая буква генома G[t+1] сравнивается с P[3] = A.

Если x = P[4] = T ( в геноме найдено АСАT), то *n=0*, найденный префикс - пустойи следующая буква генома G[t+1] сравнивается с P[1] = A.

Пример 4. Текущая буква генома G[t] = x сравнивается с последней буквой паттерна P[8] = T, см. рис.7. Предшествующие 7 букв генома совпадают с соответствующими буквами паттерна, то есть найденный префикс - это АСАGACA.

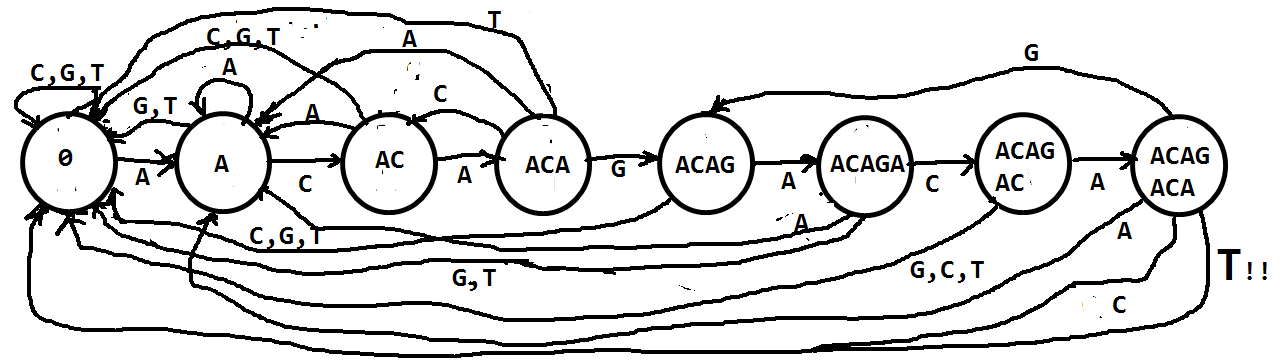


Рис.7

Если x = P[4] = T ( в геноме найдено АСАGACAT), то найдено вхождение паттерна. Это обозначено восклицательными знаками на соответствующей стрелке, такую стрелку назовем *допускающей*. Следующая буква генома G[t+1] будет сравниваться с P[1] (*n=0,* см. определение *n* в *формальном описании*)*.* Мы продолжаем поиск новых вхождений паттерна.

Если x = P[4] = A ( в геноме найдено АСАGACAA), то *n=1* и следующая буква генома G[t+1] сравнивается с P[2] =C.

Если x = P[4] = C ( в геноме найдено АСАGACAC), то *n=2* и следующая буква генома G[t+1] сравнивается с P[3] = A.

Если x = P[4] = G ( в геноме найдено АСАG), то *n=4* и следующая буква генома G[t+1] сравнивается с P[5] = A.

Замечание. Рассмотрим паттерн ACAGACAA (он отличается от нашего паттерна тем, что последняя буква не T, а A). Для такого паттерна возможны перекрывающиеся вхождения. Например, в тексте F = ACAGACAACAGACAA есть 2 вхождения: F[1..8] и F[8..15]. Допускающая стрелка из 8-го узла с буквой A ведет в этом случае во второй узел (*n= 1,* см. определение *n* в *формальном описании*).

Замечание. Построенная нами конструкции известна в теории алгоритмов под названием *конечного автомата* (классическое определение немного отличается от нашего). Конечный автомат имеет конечный набор *состояний* (они у нас обозначены кружками). Состояния нашего автомата соответствуют началам (по-научному - *префиксам*) паттерна.

На вход автомата по очереди подаются буквы определенного текста (у нас это буквы генома). Получив букву, автомат переходит в новое состояние. Находясь в определенном состоянии и получив на вход определенную букву, автомат может выдать сигнал о том, что обнаружено «допустимое слово». Конечные автоматы используются во многих алгоритмах поиска паттернов

3.3. Паттерны, включающие более одного слова.

Часто сайты связывания в геноме бывают *вырожденными,* то есть связывание происходит в месте, где в геноме находится одна последовательность нуклеотидов (слово) из некоторого набора. Такой набор слов тоже называется *паттерном* (паттерны, которые рассматривались в пп. 3.1, 3.2 – это *простые* или *однословные* паттерны).

Описанная идея (построение конечного автомата, которому на вход по очереди подаются буквы генома) применима и для паттернов, которые содержат несколько слов.

В этом случае состояния соответствуют префиксам всех слов паттерна (если у нескольких слов паттерна есть общее начало, ему соответствует одно состояние).

Формальное правило построения автомата теперь выглядит так:

*Рассмотрим слово w, которое* *является началом (префиксом) одного или нескольких слов паттерна. Пусть x – буква (в случае геномного алфавита – одна из букв A, C, G, T). Пусть текущая буква генома G[t] = x сравнивается с (r+1)-й буквой паттерна. Пусть n – такое наибольшее число, что конец слова wx длины n является началом одного из слов паттерна (но не совпадает с этим словом целиком!). Пусть z и есть это самое начало.*

*Тогда стрелка из состояния w, помеченная буквой x ведет в состояние z. Если при этом слово wx принадлежит паттерну, то стрелка является допускающей.*

На рис.8 изображен автомат для поиска вхождений паттерна, состоящего из двух слов ACAGACAT и ACAACACA. Все стрелки, которые явно не показаны на рисунке, ведут в начальное состояние, которое соответствует пустому префиксу.

Замечание. Если убрать стрелки, ведущие назад, останется дерево. Это дерево называется *префиксным деревом*.

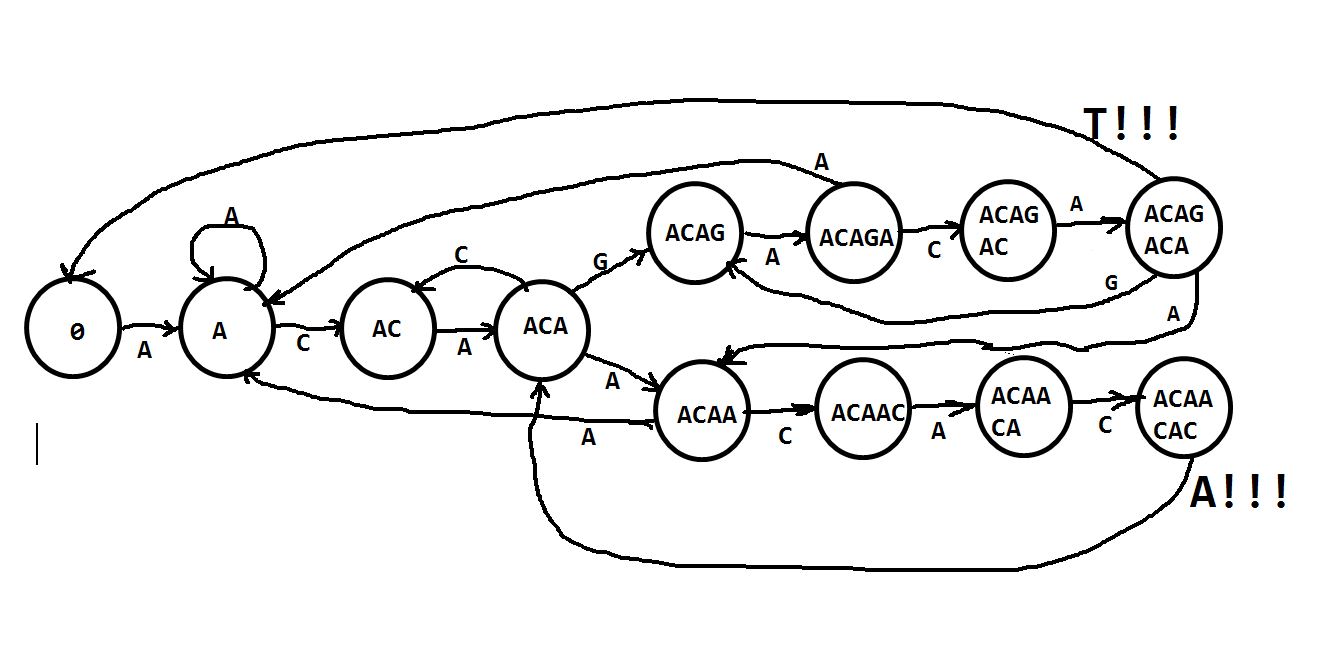


Рис.8. Автомат для поиска вхождений паттерна, состоящего из двух слов ACAGACAT и ACAACACA. Все стрелки, которые явно не показаны на рисунке, ведут в начальное состояние, которое соответствует пустому префиксу.

**4. Подведение итогов.**

За 4 занятия мы познакомились со следующими темами:

(Занятие 1)

- формальная запись алгоритма, время работы алгоритма,

- задача сортировки; наивный алгоритм сортировки; время его работы;

потребность в улучшении

(Занятие 2)

- задача слияния массивов; алгоритм слияния массивов и алгоритм MergeSort

с использованием массивов; количество сравнений в этих алгоритмах и общее время работы

(Занятие 3)

- списки; списки с доступом к концу; явное использование указателей в языках

программирования;

- понятие структуры данных, допустимые операции для каждой из рассмотренных структур

данных:

- алгоритмы сцепления списков и вставки списка в другой список;

(Занятие 4)

- задача поиска объекта (числа) в множестве; сбалансированные бинарные деревья поиска;

время поиска при их использовании

- идея балансировки; примеры ее использования

- деревья отрезков (различные виды); время решения с их помощью задач НАЙТИ,

ДОБАВИТЬ, УДАЛИТЬ, МАКСИМУМпоОТРЕЗКУ, СУММАпоОТРЕЗКУ;

- задача поиска всех вхождений паттерна в тексте (геноме) для случая простого паттерна (из

одного слова) и паттерна, содержащего несколько слов; конечный автомат на основе

префиксного дерева; решение задачи с помощью этого автомата.