План лекции

1. Задача нахождения оптимальных значений
2. Жадные алгоритмы. Задача об интервалах.
3. Применимость жадных алгоритмов.
4. Приближённое решение задачи о рюкзаке.
5. Алгоритм Хафмена.
6. Абстракция строка символов.
7. Динамические структуры данных.
8. Префиксное дерево.

Задачи нахождения оптимальных значений

Задачи:

- Путешественник желает посетить несколько городов и потратить минимальную сумму денег.

Задачи нахождения оптимальных значений

Задачи:

- Путешественник желает посетить несколько городов и потратить минимальную сумму денег.
- Управление дорожного движения хочет определить длительность фаз светофора, при котором будет обеспечен наибольший трафик.

Задачи нахождения оптимальных значений

Задачи:

- Путешественник желает посетить несколько городов и потратить минимальную сумму денег.
- Управление дорожного движения хочет определить длительность фаз светофора, при котором будет обеспечен наибольший трафик.
- Задача о рюкзаке.

Экстремальные задачи

Экстремальные задачи — задачи на нахождение оптимальных (максимальных или минимальных) значений.
Решение таких задач — *оптимизация*.
Некоторые экстремальные задачи мы можем решить точно, некоторые — приближённо.

Жадные алгоритмы

Жадные алгоритмы

- состоят из итераций
- принимают решение на каждом шаге, стараясь найти *локально* оптимальное решение.

Градиентный спуск

Пример типичного жадного алгоритма.
Имеется непрерывная функция n переменных $f(x_1, x_2, \ldots, x_n)$, принимающая действительные значения на области определения.
Она определяет поверхность в $n-$мерном пространстве.
Один из алгоритм минимизации:

1. выбираем начальную точку $(x_1, x_2, \ldots, x_n)$, она становится текущей точкой алгоритма.
2. обследуя точки вокруг текущей находим такую, в которой $f(x_1', x_2', \ldots, x_n')$ имеет минимальное значение.
3. если найденная точка отлична от текущей, то делаем его текущей и переходим к второму шагу алгоритма.
4. конец.

Пример градиентного спуска

$f(x, y) = (x-3)^2 + (y+2)^2$
Начальная точка $(x_0 = 0, y_0 = 0)$. Шаг поиска 0.1.
Осматриваем окрестности начальной точки.

$$
\begin{aligned}
&f(0,0) = 3^2 + 2^2 = 13 \\
&f(0+0.1,0) = 2.9^2 + 2^2 = 8.41 + 4 = 12.41 \\
&f(0-0.1,0) = 3.1^2 + 2^2 = 9.61 + 4 = 13.61 \\
&f(0,0+0.1) = 9 + 4.41 = 13.41 \\
&f(0,0-0.1) = 9 + 3.61 = 12.61
\end{aligned}
$$

Победила точка $(0.1, 0)$, она становится текущей точкой.

Программа для решения задачи

```
#include <stdio.h>
double f(double x, double y) {
   return (x-3)*(x-3) + (y+2)*(y+2);
}
int main() {
   double x0 = 0., y0 = 0., d = 0.1;
   double dx[] = {d, -d, 0, 0}, dy[] = {0, 0,  d, -d};
   double newx, newy;   bool bestfound = false;
   double maxf = f(x0, y0);
   while (!bestfound) {
      bestfound = true;
      for (int i = 0; i < 4; i++) {
        double newf = f(x0+dx[i],y0+dy[i]);
        if (newf < maxf) {
           maxf = newf;  bestfound = false;
           newx = x0+dx[i];  newy = y0+dy[i];
        }
      }
      if (!bestfound) {
         x0 = newx;  y0 = newy;
      }
   }
   printf("Best f(%.1f,%.1f)=%.2f\n", x0, y0, maxf);
}
```

Пример градиентного спуска

Следующий шаг. Текущая точка $(0.1, 0)$.
Мы выбираем из точек $(0.2, 0), (0, 0), (0.1, 0.1), (0.1, -0.1)$, что приводит нас к следующей точке $(0.2, 0)$
Далее маршрут проходит через точки от (0.3,0) до (1,0), затем попеременно до (3,2).

Решение правильное.

Градиентный спуск

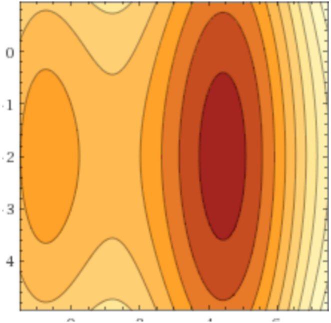
А что произойдёт с решением задачи для функции $(x-3)^2 + 10\sin x + (y+2)^2$?

Наш алгоритм выдаст, что лучшим решением будет точка (-0.7,2) с значением ≈ 7.24, хотя минимум этой функции достигается в точке $(\approx 4.4, -2)$ с значением ≈ -7.6.

Градиентный спуск

Функция $(x-3)^2 + 10\sin x + (y+2)^2$ имеет несколько локальных минимумов.

Её линии уровня:



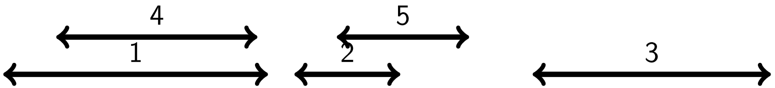
Градиентный спуск

Этот алгоритм, как и все жадные алгоритмы, склонен к нахождению локальных экстремумов.

Задача об интервалах

Задача об интервалах

Задача 1. На прямой дано множество отрезков. Необходимо найти максимальный размер множества непересекающихся отрезков.



Задача об интервалах

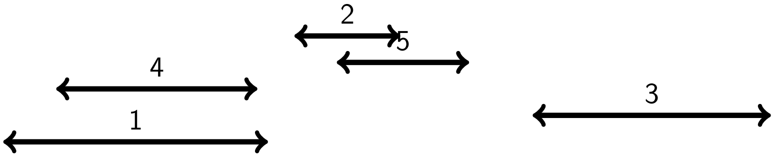
Предлагается рассмотреть следующий вариант жадной стратегии:

- упорядочить отрезки по какому-либо признаку.
- рассматриваем отрезки по-одному. Если он не перекрывается с каким-либо из уже внесённым в выходное множество, то добавляем её в это множество.

Жадность алгоритма здесь заключается в том, что каждый раз, когда мы видим подходящий вариант (рассматривая очередной отрезок), то сразу его хватаем.
Принципов упорядочивания можно выбрать несколько, но, как оказывается, не все из них одинаково полезны.

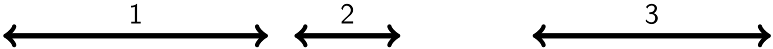
Задача об интервалах

По длительности. Сначала выберем самые короткие отрезки.



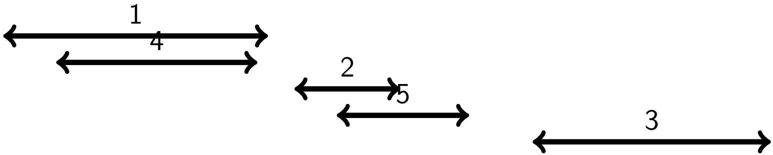
Задача об интервалах

По длительности: итоговая расстановка:



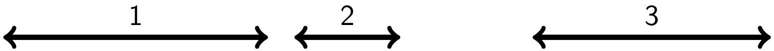
Задача об интервалах

По левой границе.



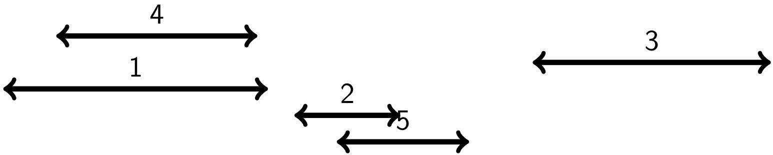
Задача об интервалах

Итоговая расстановка:



Задача об интервалах

По правой границе.

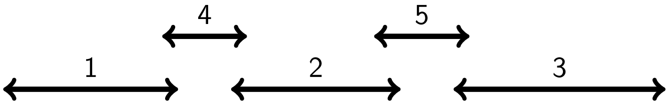


Задача об интервалах

Итоговая расстановка

Задача об интервалах

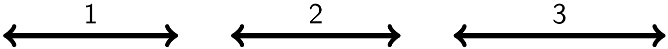
Как будто, все способы упорядочивания годятся?
А что насчёт такой расстановки?



Задача об интервалах

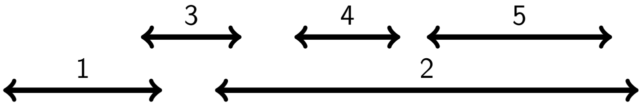
Сначала самые короткие:

По левой границе и по правой границе:



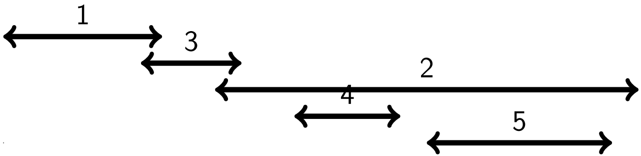
Первый вариант не всегда даёт точное решение.
Поиск контрпримера: мы хотим найти опровергающий какую-либо гипотезу вариант (*зелёную ворону*).

Рассмотрим следующее расположение отрезков:



Задача об интервалах

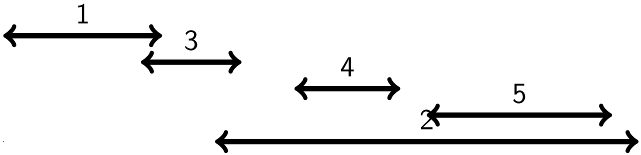
Упорядочивание по началу отрезка даёт нам:



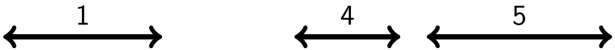
Решение:

Задача об интервалах

Упорядочивание по концу отрезка даёт нам:



И это приводит к верному решению:



Задача об интервалах

Как доказать, что данный алгоритм верно решает задачу?

1. Первый шаг — доказательство того, что существует оптимальное подмножество отрезков, которое содержит первый отрезок, получившийся при применении нашего алгоритма. Если в некотором оптимальном подмножестве мы поменяем отрезок с минимальным значением конца на первый, то количество отрезков в подмножестве не изменится и подмножество останется решением. Таким образом, существует оптимальное подмножество, содержащее первый отрезок.
2. Второй шаг — удаляем из множества отрезков все отрезки, пересекающиеся с первым.
3. Третий шаг — повторяем алгоритм для усечённого множества, в котором находит первый отрезок.

Применив метод математической индукции, мы показали, что предложенный жадный алгоритм приводит к одному из оптимальных решений задачи.

Жадные алгоритмы не заглядывают вперёд. Они повторяют локально оптимальные по какому-либо критерию шаги и надеются, что решение будет глобплльно оптимальным. Возможно, что найдётся такой локально оптимальный критерий и общее решение окажется верным. Это бывает отнюдь не всегда, но тщательный выбор критерия может найти приемлемое решение.

Применимость жадных алгоритмов. Приближённое решение экстремальных задач

Опять задача о рюкзаке.
Одно из точных решений имеет сложность $O(2^N)$.
Как найти приближённое решение?

Формализация условия задачи о рюкзаке

Задача о рюкзаке. Пусть имеется N предметов, стоимость $i-$го предмета v_i, а масса w_i. Найти набор предметов с наибольшей стоимостью и не превосходящей заданного W массой.

Приближённое решение Попробуем применить следующий локально оптимальный алгоритм:

1. Расположим предметы в порядке убывания отношения $\dfrac{v_i}{w_i}$. Пусть они образуют упорядоченное множество B.
2. Установим оставшийся вес $L = W$
3. Установим множество $S = \emptyset$.
4. Выбираем первый предмет I из упорядоченного множества, вес w_I которого не превосходит L.
5. Если такого предмета нет, то алгоритм закончен.
6. Кладём предмет в рюкзак, удаляя его из из B: $B \leftarrow B - I : L \leftarrow L - w_I; S \leftarrow SI$. Переходим к 4-му шагу.

Данный алгоритм приведёт к какому-либо решению.
Рассмотрим пример:

$N = 3$

$W = 40$

$w_1 = 10; v_1 = 60$

$w_2 = 20; v_2 = 100$

$w_3 = 20; v_3 = 100$

Алгоритм выберет последовательно первый и второй предметы. Их суммарная стоимость окажется 160.
Верное решение - выбрать второй и третий предметы. Их суммарная стоимость будет 200.

Задача о сумме подмножества

Задача 3. Дано множество натуральных чисел $S = \{a_1, a_2, \ldots, a_n\}$ и натуральное число L. Найти такое подмножество, сумма элементов Sum такова, что $Sum - L \geqslant 0$ и $Sum - L \to \min$.

Жадный алгоритм: сведение к жадной задаче с рюкзаком при стоимости равной a_i и весу, равному 1.

Quiz

http://rbtsv.ru/quiz/index

Сжатие информации. Алгоритм Хаффмана.

Сжатие информации: алгоритм Хаффмана

Задача 4. Имеется текст, состоящий из символов. Закодировать его таким образом, чтобы:

- каждый из встречающихся символов получил свой двоичный код
- множество кодов было префиксным
- суммарная длина всех кодов для всех символов была бы минимальной.

Алгоритм Хаффмана

Пример: пусть имеется текст, состоящий из множества из четырёх символов:

$$AAAABAABABABABCBCAAAD$$

Его длина — 21 символ.
Можно закодировать его следующим образом:

- $A \to 00$
- $B \to 01$
- $C \to 10$
- $D \to 11$

На кодирование каждого символа понадобится ровно два бита и общая длина кода составит 42 бита.

Префиксный код

Неформальное определение: код, в котором не имеется кодовых слов, начинающихся с других кодовых слов.
Формальное определение: если существует код со строкой a, то кодов с непустой строкой ab не существует.
Пример: код

- $A \to 00$
- $B \to 10$
- $C \to 01$
- $D \to 101$

префиксным не является, так как кодовое слово символа D начинается с кодового слова символа B.
Другое название: *безпрефиксный*.
Ещё одно название — код, удовлетворяющий условиям Фано.
Наша задача — найти *минимальный префиксный код* для множества.

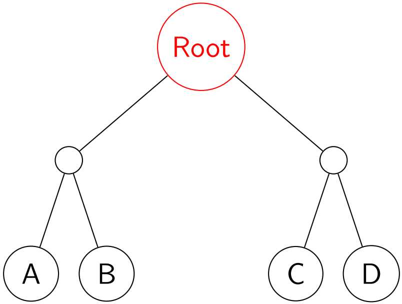
Кодирование с помощью дерева

Ещё раз рассмотрим равномерный код из четырёх символов

- $A \rightarrow 00$
- $B \rightarrow 01$
- $C \rightarrow 10$
- $D \rightarrow 11$

Попробуем представить его в виде двоичного дерева.

Двоичные деревья



Левая ветка - 0, правая ветка - 1.

Алгоритм Хаффмана

$$AAAABAABABABABCBCAAAD$$

Определим частоты символов:

- $F_A = 12$
- $F_B = 6$
- $F_C = 2$
- $F_D = 1$

Нужно построить дерево, вес которого минимален.

$$\sum_{i=1}^{n} F_i \cdot L_i,$$

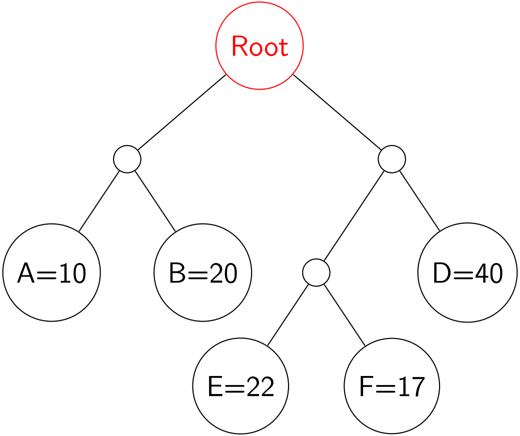
где L_i - глубина $i-$го символа.

Алгоритм Хаффмана

Свойства оптимального дерева:

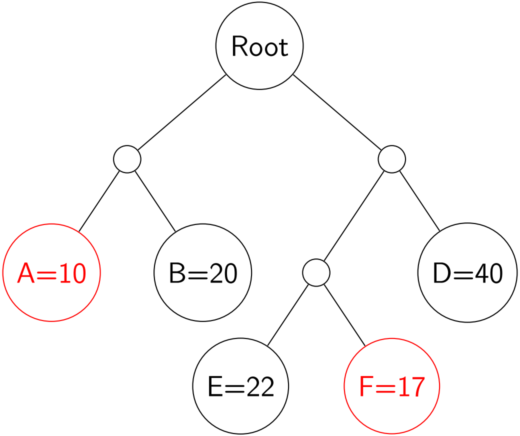
- Из каждого узла должно исходить ровно два пути.
- Не должно быть пустых вершин.
- Самое длинное кодовое слово должно быть парным.

Алгоритм Хаффмана



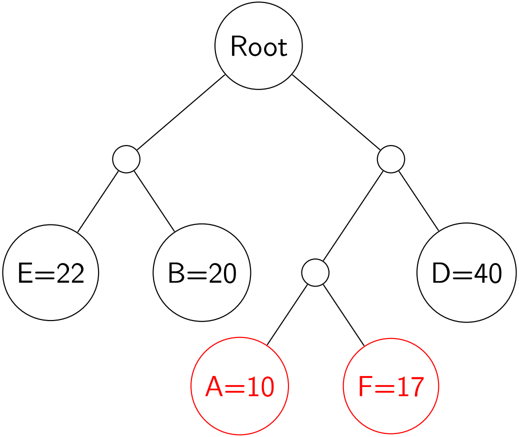
Алгоритм Хаффмана

Находим два самых редких символа.



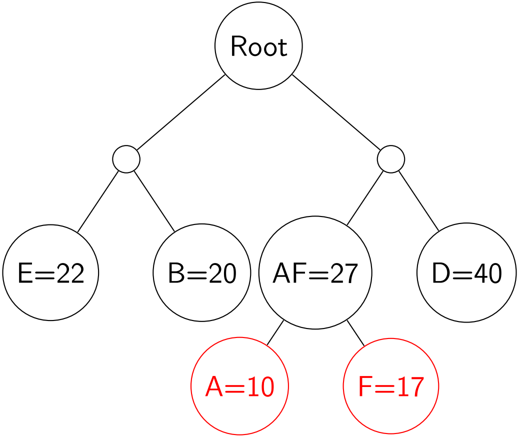
Алгоритм Хаффмана

Два самых редких символа должны находиться на самой длинной ветке. Если нет, то можно их поменять местами с символами на самой длинной ветке.



Алгоритм Хаффмана

Два самых редких символа должны находиться на самой длинной ветке. Если нет, то можно их поменять местами с символами на самой длинной ветке.

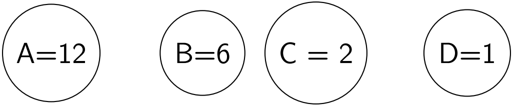


Алгоритм Хаффмана

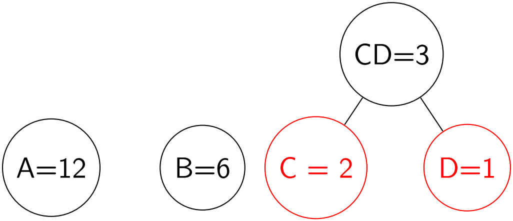
Жадный алгоритм:

1. Помещаем в каждый узел частоту символа
2. Располагаем узлы согласно убыванию частот.
3. Для двух узлов с наименьшей частотой добавляем узел, который их соединяет.
4. В узел помещаем сумму частот детей
5. Помечаем узлы или вершины, как уже обработанные (отправляем вниз)
6. Если необработанных не осталось, то конец алгоритма
7. Переходим к шагу 2.

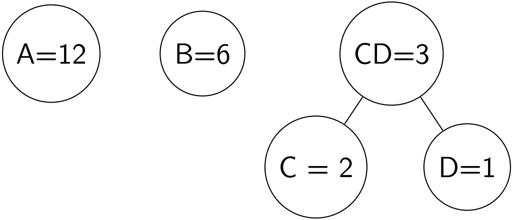
После шага 1



Шаги 2,3,4

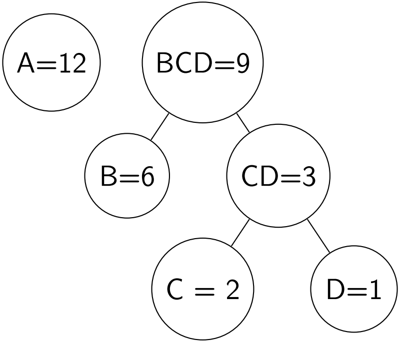


Шаг 5



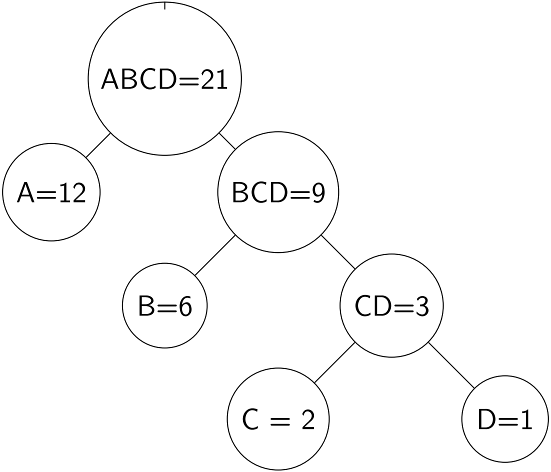
Алгоритм Хаффмана

Следующая итерация:



Алгоритм Хаффмана

Заключительное состояние:



Алгоритм Хаффмана

Получившиеся коды:

- $A \to 0$
- $B \to 10$
- $C \to 110$
- $D \to 111$

Общая длина всех кодовых слов
$12 \cdot 1 + 6 \cdot 2 + 2 \cdot 3 + 1 \cdot 1 = 31 < 42$

Задача о покрытии строки

Рассмотрим следующую задачу:
Имеется набор строк $s_i, i = 1 \ldots N$ - «слова», каждое из которых не начинается с другого (префиксный код).
Имеется длинная строка p.
Требуется определить, можно ли составить слово p из слов s_i.

Например, если слова $s_i = \{ab, ca, ra, dab\}$, то строку *abracadabra* из них составить можно $ab + ra + ca + dab + ra$, а вот слова *barca*, *abracadabraa* — нет.

Задача о покрытии строки

Цель задачи — не просто найти решение, а найти оптимальное решение.

Главные параметры алгоритма:

- N — длина строки p.
- M — сумма длин строк s_i.

Решается ли эта задача жадным алгоритмом?

Задача о покрытии строки

Жадный алгоритм:

1. Устанавливаем указатель позиции на начало «длинной» строки
2. Выбираем слово, которое полностью совпадает с подстрокой, начинающейся с указателя
3. Если такого слова не найдено, выводим «нет», завершение алгоритма
4. Если такое слово есть, перемещаем указатель на длину слова.
5. Если слово закончилось, то выводим «да» и завершаем алгоритм
6. Возвращаемся к пункту 2

Задача о покрытии строки

C	A	B	B	A	C	A	A	C	A	B	A	B	A	C	C	A	B

A	B	A	C

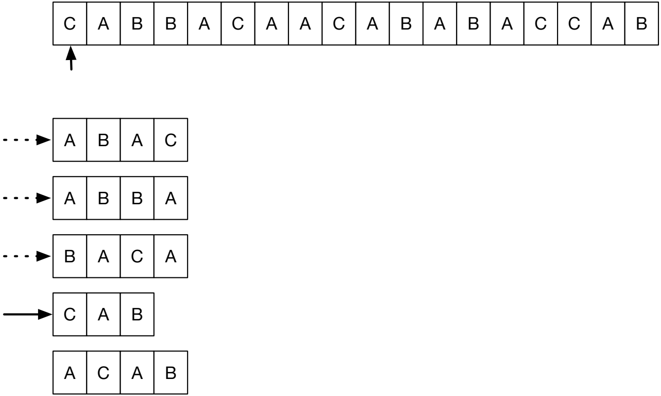
A	B	B	A

B	A	C	A

C	A	B

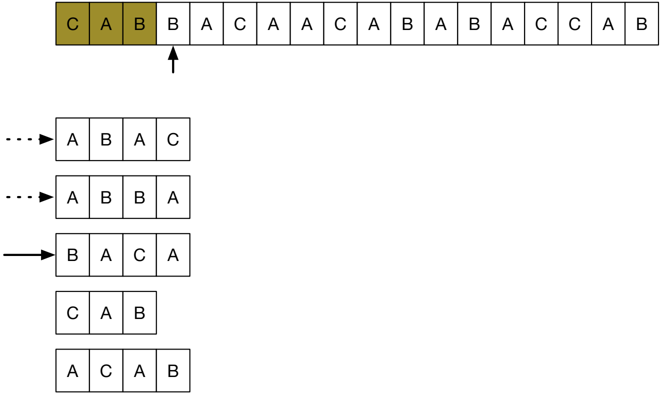
A	C	A	B

Задача о покрытии строки



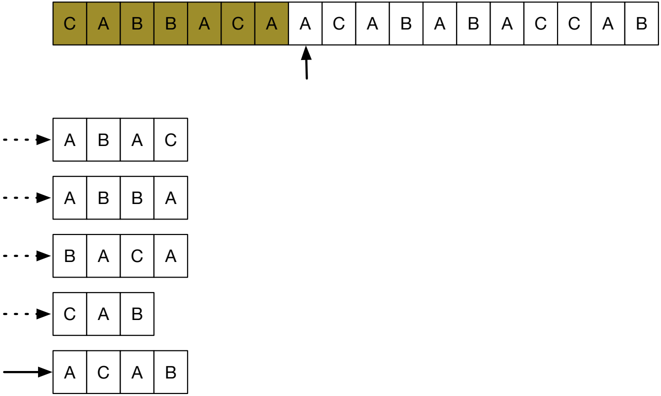
Этап 1. Нашли первое слово.

Задача о покрытии строки



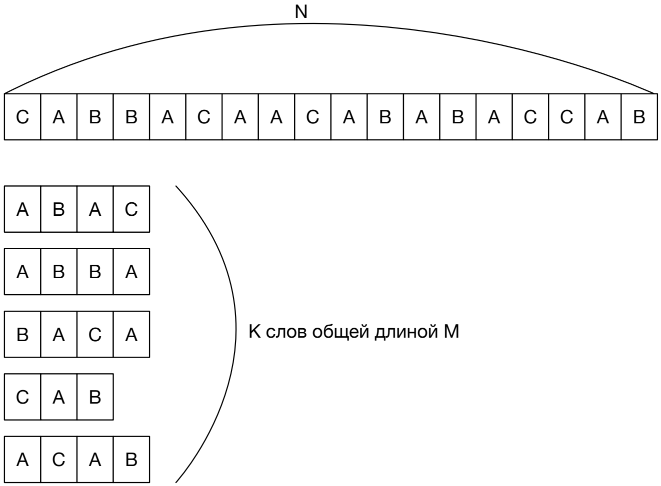
Этап 2. Переместили указатель и нашли второе слово.

Задача о покрытии строки



Этап 3. Переместили указатель и нашли третье слово.

Оценка сложности алгоритма



Оценка сложности алгоритма

- Определить, что слово подошло, мы можем только просмотрев всё слово.
- Определить, что слово не подошло, можно даже с первого символа слова.
- Грубо оценим количество попыток на слово длины L как $\frac{L}{2}$
- Средняя длина слова есть $L = \frac{M}{K}$
- При одном этапе поиска мы в среднем перебираем $\frac{K}{2}$ слов, каждое средней длиной L.
- Каждый этап продвигает нас в среднем на L позиций в «длинной» строке, итого количество этапов $T = \frac{N}{L}$
- Итого $F = \frac{N}{L} \times \frac{K}{2} \times \frac{L}{2} = \frac{NK}{4} = O(NK)$

Оценка сложности алгоритма

- Парадоксально, но сложность алгоритма не зависит от общей длины всех слов!
- Представим, что $K = 1$, то есть, ищется покрытие ровно одним словом длины M (пусть $N \vdots M$).
- Тогда каждый успешный поиск продвигает нас по «длинной» строке на M позиций. Всего наибольшее количество поисков $\frac{N}{M}$, в каждом из которых сравнивается M символов.
- Итого $F = O(N)$

Исследование задачи для поиска другого алгоритма

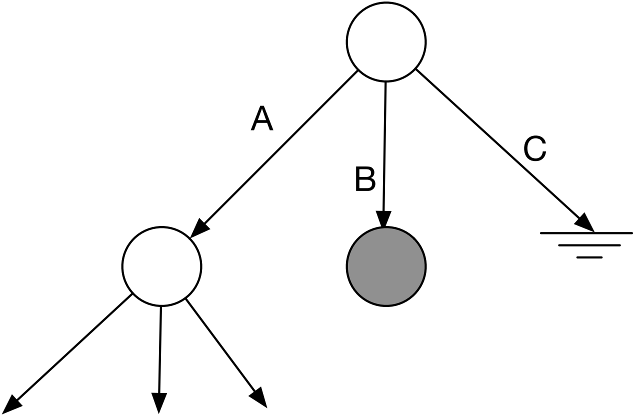
- Для большого K алгоритм становится неэффективен.
- Имеется ли более короткое решение?

Исследование задачи для поиска другого алгоритма

- Для большого K алгоритм становится неэффективен.
- Имеется ли более короткое решение?
- Да, если мы изменим структуру данных.
- Проблема в том, что мы, обнаружив несовпадение с одной подстрокой, ничего не получаем для следующих.
- Усложним представление тех строк, которые мы ищем.
- Попробуем производить поиск параллельно по всем подстрокам.
- Построим *префиксное дерево*

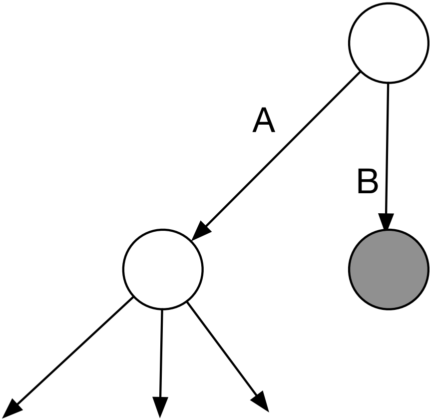
Префиксное дерево

- Из каждого узла всегда выходит ровно три ветви, A, B и C.
- Каждая из веток приходит или в узел, либо в вершину, либо в никуда.



Префиксное дерево

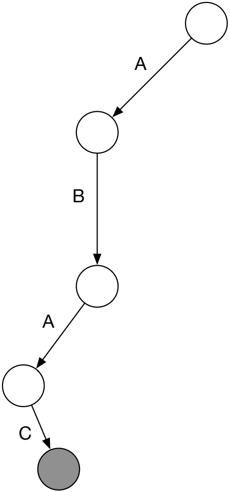
- Для простоты не будем рисовать ветви, уходящие в никуда.



Построение префиксного дерева

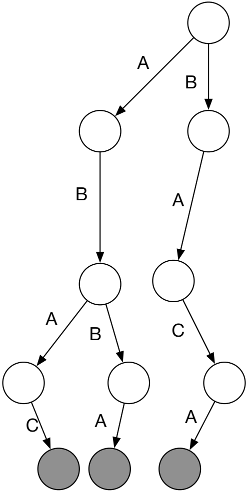
Слова: ABAC, ABBA, BACA, CAB, ACAB

- Строим дерево для каждого слова посимвольно.
- Для первого слова, ABAC дерево будет таким:



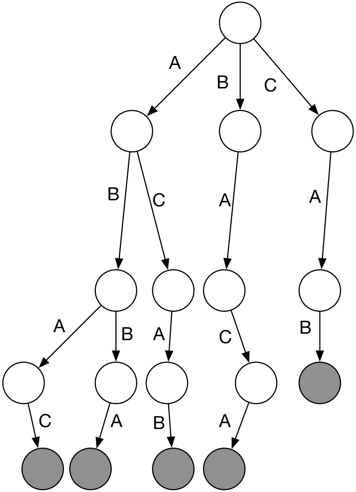
Построение префиксного дерева

- Добавим слова ABBA и BACA:



Построение префиксного дерева

- Теперь слова CAB и ACAB:



Алгоритм построения префиксного дерева

1. Создаём вершину дерева — узел с пустыми ветвями.
2. Для каждого слова исполняем:
   2.1 Устанавливаем указатель в вершину дерева.
   2.2 Считываем очередную букву.
   2.3 Если текущей узел не содержит нужной ветви, создаём эту ветвь и пустой узел на ней.
   2.4 Переходим в нужную ветвь.
   2.5 Если узел уже серый или не пустой, завершаем алгоритм с неудачей.
   2.6 Если слово закончилось, то помечаем узел серым цветом.
   2.7 Переходим к 2.2
3. Завершаем алгоритм с успехом.

Оценка сложности алгоритма

- Один шаг алгоритма продвигает нас на ровно один символ в образце.
- Повторно образцы не обрабатываются.
- Количество операций есть $O(M)$, где M — сумма длин образцов.

Поиск с помощью префиксного дерева

- Поиск по получившемуся дереву стал совсем простым:

C	A	B	B	A	C	A	A	C	A	B	A	B	A	C	C	A	B