

Алгоритмы и структуры данных

Лекция 3

Жадные алгоритмы

Сергей Леонидович Бабичев

План лекции

- 1 Задача нахождения оптимальных значений
- 2 Жадные алгоритмы. Задача об интервалах.
- 3 Жадные алгоритмы. Задача о резервных копиях.
- 4 Применимость жадных алгоритмов.
- 5 Приближённое решение задачи о рюкзаке.
- 6 Префиксное дерево. Задача о покрытии строки.

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.

Покомпонентный спуск: типичный жадный алгоритм

Имеется непрерывная функция n переменных $f(x_1, x_2, \dots, x_n)$, 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, \dots, x_n) , она становится текущей точкой алгоритма.
- 2 обследуя точки вокруг текущей находим такую, в которой $f(x'_1, x'_2, \dots, x'_n)$ имеет минимальное значение.
- 3 если найденная точка отлична от текущей, то делаем его текущей и переходим к второму шагу алгоритма.
- 4 конец.

Пример покомпонентного 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.

$$f(0, 0) = 3^2 + 2^2 = 13$$

$$f(0 + 0.1, 0) = 2.9^2 + 2^2 = 8.41 + 4 = 12.41$$

$$f(0 - 0.1, 0) = 3.1^2 + 2^2 = 9.61 + 4 = 13.61$$

$$f(0, 0 + 0.1) = 9 + 4.41 = 13.41$$

$$f(0, 0 - 0.1) = 9 + 3.61 = 12.61$$

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 покомпонентного 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.

Покомпонентный 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 .

Покомпонентный спуск

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

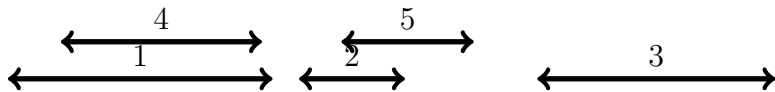
Покомпонентный спуск

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:

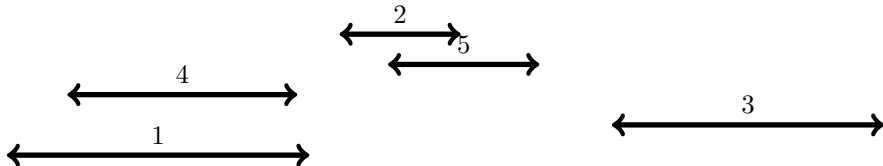
- упорядочить отрезки по какому-либо признаку.
- рассматриваем отрезки по-одному. Если он не перекрывается с каким-либо из уже внесённым в выходное множество, то добавляем её в это множество.

Жадность — каждый раз, когда мы видим подходящий вариант (рассматривая очередной отрезок), то сразу его хватаем.

Принципов упорядочивания может быть несколько. Не все одинаково полезны.

Задача об интервалах

По длительности. Сначала выберем самые короткие отрезки.

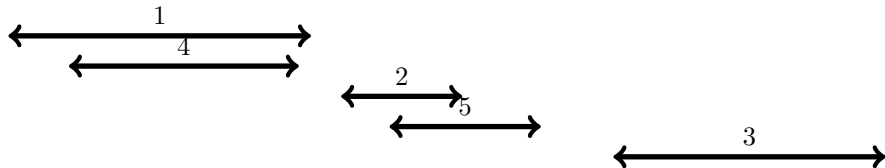


Задача об интервалах

По длительности: итоговая расстановка:

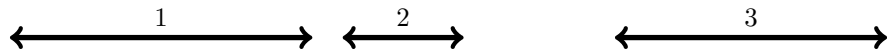


Задача об интервалах



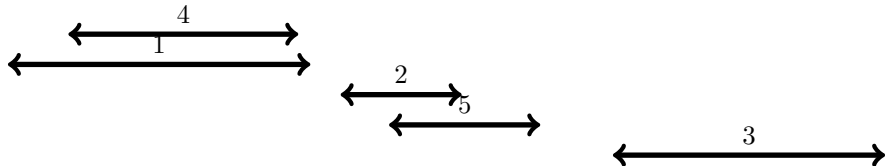
Задача об интервалах

Итоговая расстановка:



Задача об интервалах

По правой границе.

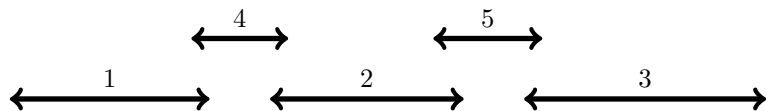


Задача об интервалах



Задача об интервалах

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

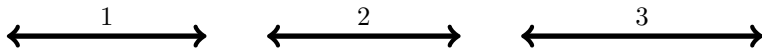


Задача об интервалах

Сначала самые короткие:



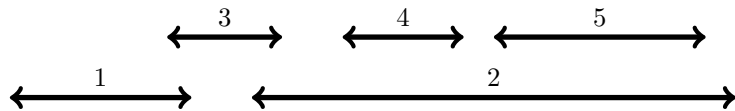
По левой границе и по правой границе:



Первый вариант не всегда даёт точное решение.

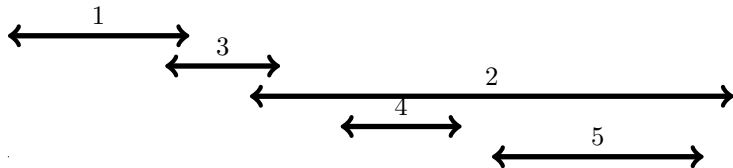
Поиск контрпримера: мы хотим найти опровергающий какую-либо гипотезу вариант (*зелёную ворону*).

Рассмотрим следующее расположение отрезков:



Задача об интервалах

Упорядочивание по началу отрезка даёт нам:

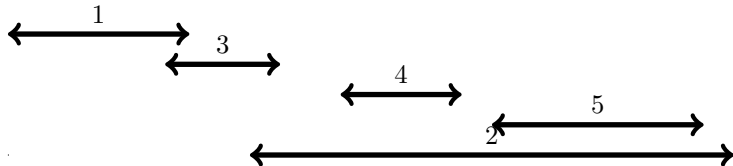


Решение:

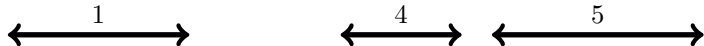


Задача об интервалах

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.

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.

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 находится 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 глобально 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.

Задача об резервных копиях

Задача о резервных копиях

Задача 2. Имеется распределённая система, состоящая из N хранилищ различной ёмкости, причём в i -м хранилище можно разместить A_i блоков информации.

Хранение одного блока считается надёжным, если имеется две его копии в различных хранилищах. Требуется определить наибольшее количество надёжных блоков, которое можно разместить во всех хранилищах.

Задача о резервных копиях: пример

4 хранилища размерами (8, 7, 4, 3).

8	7	4	3
1	1	2	2
3	3	4	4
5	5	6	6
7	7	-	-
8	8	-	-
9	9	-	-
10	10	-	-
11	-	11	-

Задача о резервных копиях

- Попробуем решить жадным алгоритмом.
- Задачу можно решать многими стратегиями.
- Какая из стратегий оптимальная?

Для удобства сведём задачу к эквивалентной: имеется N кучек камней, каждым ходом можно выбрать по одинаковому количеству камней из любой пары кучек. Найти такой порядок игры, при котором останется минимальное количество камней.

Задача о резервных копиях: стратегия 1

Первая стратегия: выбрать две наименьших кучи и взять из них одинаковое наибольшее количество камней.

8	7	4	3
8	7	1	0
8	6	0	0
2	0	0	0

Интуиция подсказывает, что решение неверное.

Задача о резервных копиях: стратегия 2

Вторая стратегия: выбрать наибольшую и наименьшую кучи и взять из них одинаковое наибольшее количество камней.

8	7	4	3
5	7	4	0
5	3	0	0
2	0	0	0

Опять не повезло.

Задача о резервных копиях: стратегия 3

Третья стратегия: выбрать две наибольших кучи и взять из них одинаковое наибольшее количество камней.

8	7	4	3
1	0	4	3
1	0	1	0
0	0	0	0

Повезло. Как обычно в задачах на жадность: нужен контрпример.

Задача о резервных копиях: стратегия 3

Новая попытка:

8	7	7	6	5
1	0	7	6	5
1	0	1	0	5
0	0	1	0	4
0	0	0	0	3

И снова неудача. Есть ли вообще нужная стратегия?

Задача о резервных копиях: стратегия 4

Четвёртая стратегия: выбрать наибольшую и наименьшую кучи, и взять из них по одному камню.

8	7	4	3
7	7	4	2
7	6	4	1
6	6	4	0
5	6	3	0
5	5	2	0
4	5	1	0
4	4	0	0
0	0	0	0

Задача о резервных копиях: стратегия 4

8	7	7	6	5
7	7	7	6	4
7	7	6	6	3
7	6	6	6	2
6	6	6	6	1
6	6	6	5	0
6	6	5	4	0
6	5	5	3	0
5	5	5	2	0
5	5	4	1	0
5	4	4	0	0
4	4	3	0	0
4	3	2	0	0
3	3	1	0	0
3	2	0	0	0
1	0	0	0	0

Задача о резервных копиях: корректность алгоритма

- Очевидно, что на каждой «большой» итерации наименьшая из куч опустошается, так как из неё по условиям алгоритма всегда производится взятие.
- Когда останется три кучи $A \geq B \geq C$, то возможны следующие ситуации:
 - ▶ $A > B + C$. Тогда ответ: $A - B - C$. Так как на каждом ходе A оставалось наибольшим, на каждом их ходов, приведшем к позиции происходило вычитание из A , это значит, что $A > \sum$ всех оставшихся, что даёт верное решение.
 - ▶ $A = B + C$. Тогда результат равен нулю.
 - ▶ $A < B + C$. Тогда, после некоторой, возможно нулевой последовательности ходов достигается ситуация, когда $A' = B' > C'$, которая сведётся к позиции $A' - \left\lfloor \frac{C'}{2} \right\rfloor, A' - \left\lfloor \frac{C'}{2} \right\rfloor$.

Применимость жадных алгоритмов. Приближённое решение экстремальных задач

Применимость жадных алгоритмов. Приближённое решение экстремальных задач

Вернёмся к задаче о рюкзаке.

Одно из точных решений имеет сложность $O(2^N)$.

Как найти приближённое решение?

Формализация условия задачи о рюкзаке

Задача 3. (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 $\frac{v_i}{w_i}$. 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 B :
 $B \leftarrow B - I : L \leftarrow L - w_I; S \leftarrow S \cup I$. Переходим к 4-му шагу.

Приближённое решение задачи о рюкзаке

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 4. Дано множество натуральных чисел $S = \{a_1, a_2, \dots, a_n\}$ и натуральное число L . Найти такое подмножество, сумма элементов Sum такова, что $Sum - L \geq 0$ и $Sum - L \rightarrow \min$.

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

Префиксное дерево

Задача о покрытии строки

Задача 5. Имеется набор строк $s_i, i = 1 \dots N$ — «слова», каждое из которых не начинается с другого (префиксный код).

Имеется «длинная» строка, «предложение» p .

Требуется определить, можно ли составить предложение 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 предложения 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 предложения.
- 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

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

Этап 1. Нашли первое слово.

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
---	---	---	---

Этап 2. Переместили указатель и нашли второе слово.

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

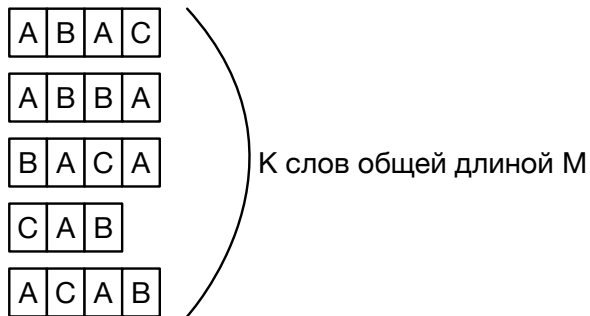
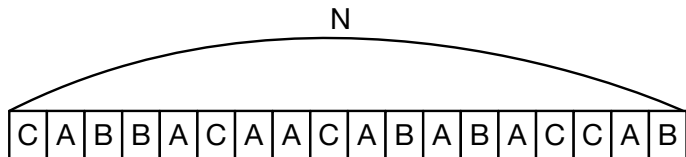
... → B A C A

... → C A B

→ A C A B

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 предложении, 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 $N \geq M$).
- Тогда каждый успешный поиск продвигает нас по предложению 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

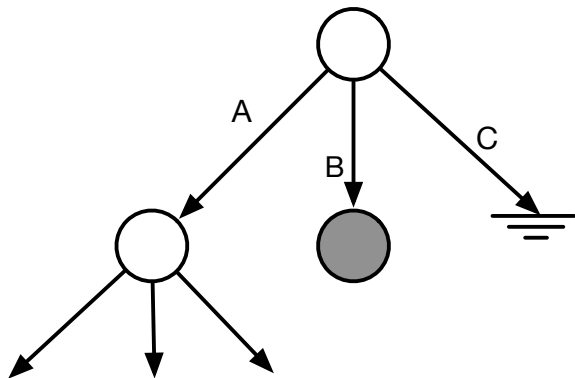
- Для большого K алгоритм становится неэффективен.
- Имеется ли более короткое решение?

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*.
- Алгоритм останется жадным, но с другой структурой данных.

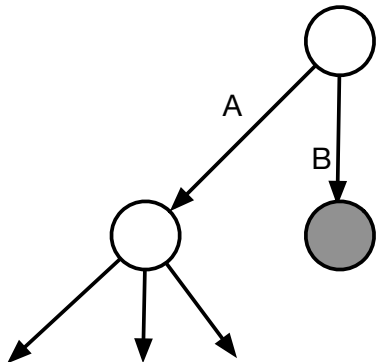
Префиксное дерево

- 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

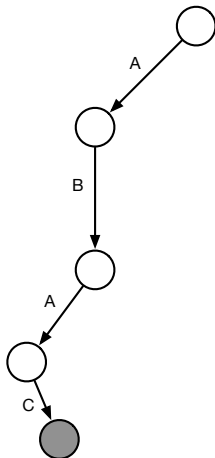
- Для простоты не будем рисовать ветви, уходящие в никуда.



Построение префиксного дерева

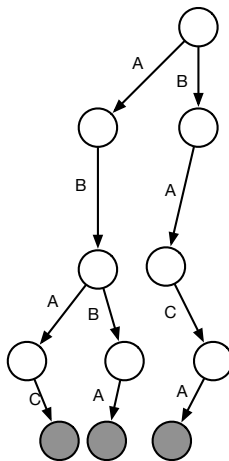
Слова: АВАС, АВВА, ВАСА, САВ, АСАВ

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



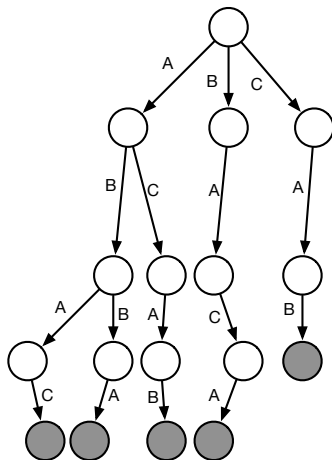
Построение префиксного дерева

- Добавим слова АВВА и ВАСА:



Построение префиксного дерева

- Теперь слова CAB и ACAB:



Алгоритм построения префиксного дерева

- 1 Создаём вершину дерева — узел с пустыми ветвями.
- 2 Для каждого слова выполняем:
 - 1 Устанавливаем указатель в вершину дерева.
 - 2 Считываем очередную букву.
 - 3 Если текущий узел не содержит нужной ветви, создаём эту ветвь и пустой узел на ней.
 - 4 Переходим в нужную ветвь.
 - 5 Если узел уже серый или не пустой, завершаем алгоритм с неудачей.
 - 6 Если слово закончилось, то помечаем узел серым цветом.
 - 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:

Алгоритм поиска по префиксному дереву

- 1 Устанавливаем указатели на начало строки и на вершину дерева.
- 2 Если указатель стоит за последним символом в строке то:
 - 1 Если указатель в дереве находится в корне — алгоритм завершён с успехом.
 - 2 Иначе алгоритм завершён с неудачей.
- 3 Считываем очередной символ строки и передвигаем указатель.
- 4 Если такой ветви дерева нет — завершаем алгоритм с неудачей.
- 5 Переходим по дереву по соответствующей ветке.
- 6 Если мы попали в серый узел — возвращаемся в корень дерева.
- 7 Переходим к пункту 2.

Оценка сложности алгоритма поиска

- Каждое перемещение символа в исходной строке приводит к перемещению указателя в дереве.
- Сложность каждого перемещения постоянна ($O(1)$).
- Количество перемещений в случае успеха равно N , где N - длина строки.
- Сложность алгоритма поиска — $O(N)$.
- Общая сложность алгоритма — $(N + M)$
- Сложность первой версии алгоритма — $O(N \times K)$.

Спасибо за внимание.

Следующая лекция —
сортировка