Введение в дискретную математику и математическую логику

Лекция №4
Деревья, основные определения и приложения

Апанович Зинаида Владимировна

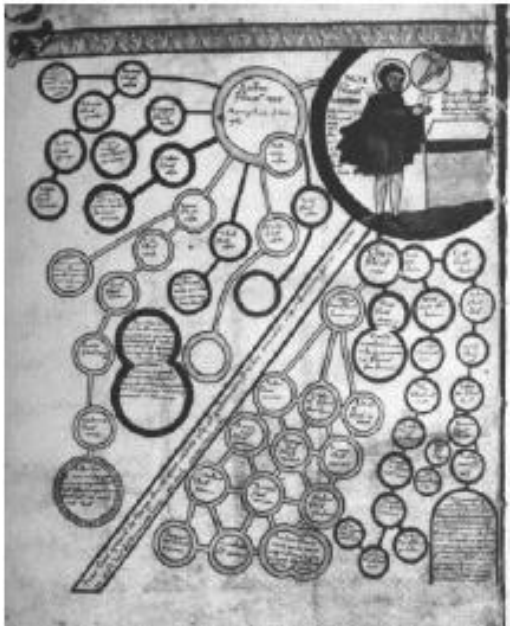
В этой лекции мы рассмотрим специальный тип графов, так называемые деревья.

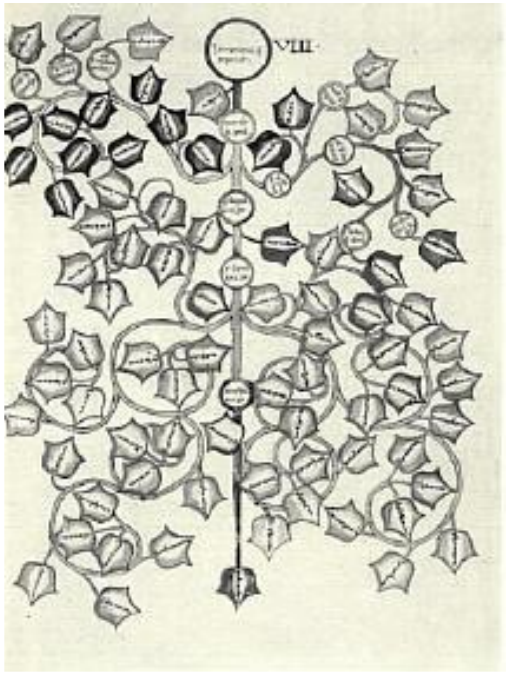
Они так названы потому, что такие графы напоминают деревья.

Например, генеалогические деревья — это графы, представляющие собой генеалогические схемы.

Вершины генеалогических деревьев представляют членов семьи, а ребра — отношения родитель-потомок.

Древние генеалогические деревья





ОПРЕДЕЛЕНИЕ 1

Дерево — это связный неориентированный граф, не содержащий простых циклов.

ИЛИ так : Дерево — это связный ациклический неориентированный граф.

Поскольку дерево не может иметь простых циклов , дерево не может содержать кратных ребер или петель .

Поэтому любое дерево должно быть простым графом .

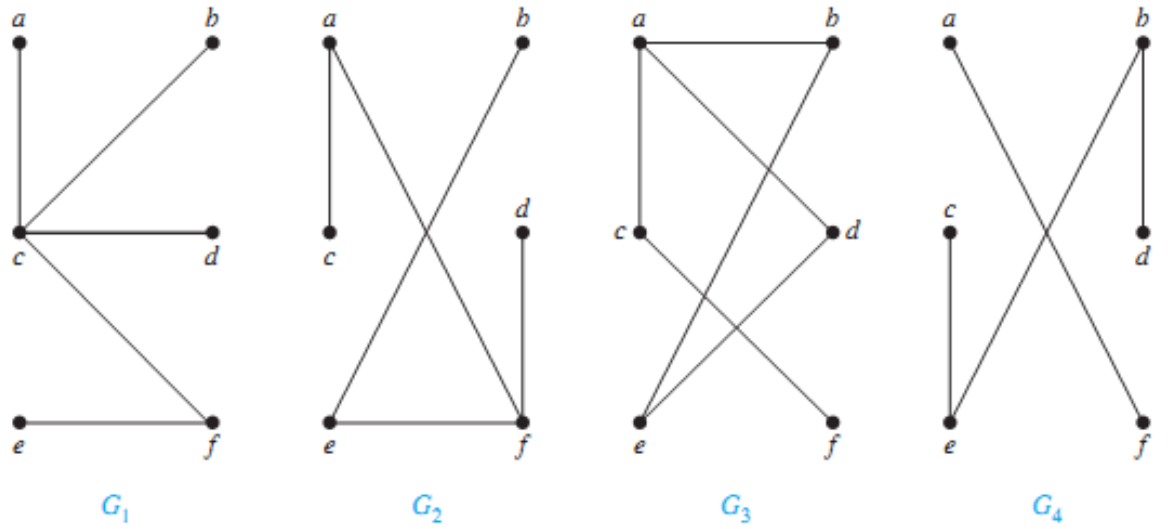
ПРИМЕР

Какие из графов G_1, G_2, G_3, G_4 являются деревьями?

Решение : графы G_1 и G_2 являются деревьями, поскольку оба являются связными графами без простых циклов.

Граф G_3 не является деревом, поскольку *e, b, a, d, e* — простой цикл в этом графе.

Наконец, граф G_4 не является деревом, поскольку он не связен.

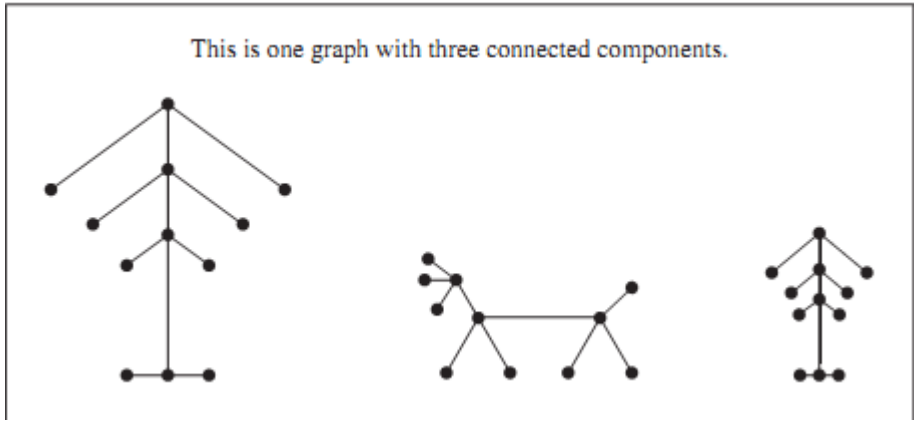


Лес

Любой связный граф, не содержащий простых циклов, является деревом.

Несвязные ациклические графы называются лесами, и обладают тем свойством, что каждая из их связных компонент является деревом.

Деревья часто определяются как неориентированные графы, обладающие свойством, что между каждой парой вершин существует единственный простой путь.



Пример леса.

Свойства деревьев

ТЕОРЕМА 1 Неориентированный граф является деревом ⇔ между любыми двумя его вершинами существует единственный простой путь.

Доказательство:

⇒ Сначала предположим, что *T* — это дерево .

Тогда *T* — связный граф без простых циклов.

Пусть *x* и *y* — две вершины *T.*

Поскольку *T* связен, по теореме 1 из лекции 2 существует простой путь между *x* и *y* .

Более того, этот путь должен быть уникальным, поскольку если бы существовал второй такой путь, то путь, образованный объединением первого пути из вершины *x* в вершину *y* с путем из вершины *y* в вершину *x,* полученным путем изменения порядка вершин второго пути, образует цикл.

Значит, между любыми двумя вершинами дерева существует единственный простой путь .

Свойства деревьев

<= Теперь предположим, что между любыми двумя вершинами графа *T* существует единственный простой путь .

Тогда *T* связен, поскольку между любыми двумя его вершинами существует путь.

Более того, *T* не может иметь простых циклов.

Чтобы убедиться в этом, предположим, что в *T* имеется простой цикл, содержащий вершины *x* и *y* .

Тогда между вершинами *x* и *y* будет 2 простых пути , поскольку простой цикл состоит из простого пути из *x в y* и второго простого пути из *y в x* .

Следовательно, граф с единственным простым путем между любыми двумя вершинами является деревом.

Корневые деревья

Во многих приложениях деревьев какая-то выделенная вершина дерева обозначается как корень .

Указав корень, мы можем присвоить направление каждому ребру следующим образом.

Поскольку существует единственный путь из корня до каждой вершины графа (по теореме 1), мы ориентируем каждое ребро в направлении от корня.

Таким образом, дерево вместе со своим корнем образует ориентированный граф, называемый корневым деревом .

Корневые деревья

ОПРЕДЕЛЕНИЕ 2 Корневое дерево — это дерево, в котором одна вершина обозначена как корень, а все ребра направлены от корня.

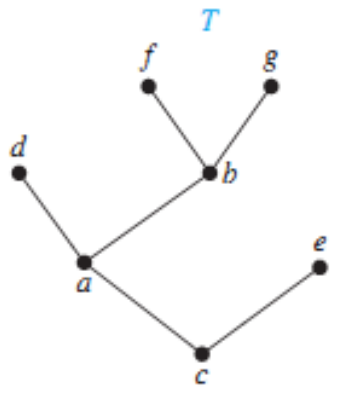
Мы можем превратить некорневое дерево в корневое, выбрав любую вершину в качестве корня.

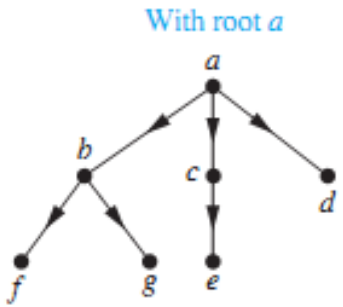
Обратите внимание, что разные варианты корня приводят к разным корневым деревьям.

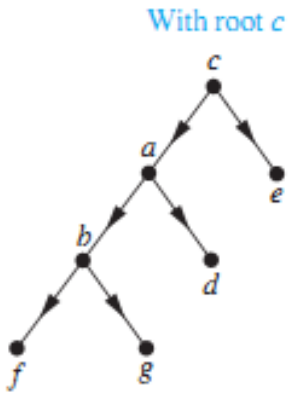
Например, корневые деревья, образованные путем выбора в качестве корня в дереве *T* вершин *a* или *c* соответственно, показаны ниже.

Обычно корневое дерево изображается с корнем в верхней части графа.

Стрелки, указывающие направления ребер в корневом дереве, можно опустить, поскольку выбор корня однозначно определяет направления ребер.







Дерево и корневые деревья, образованные путем выбора в качестве корня вершин *a* и *c* .

Терминология деревьев:отец,сын, братья

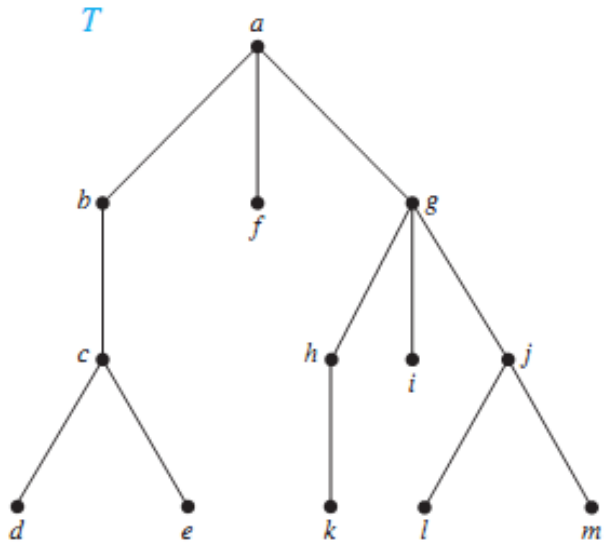
Терминология деревьев имеет ботаническое и генеалогическое происхождение.

Предположим, что T — корневое дерево.

Если v — вершина в T, отличная от корня, то отец вершины v — это единственная вершина u , такая, что существует ориентированное ребро из u в v (такая вершина уникальна).

Когда u является отцом вершины v, то v называется сыном вершины u.

Вершины, имеющие одного и того же отца, называются братьями.



Пример

Отцом вершины c является b .

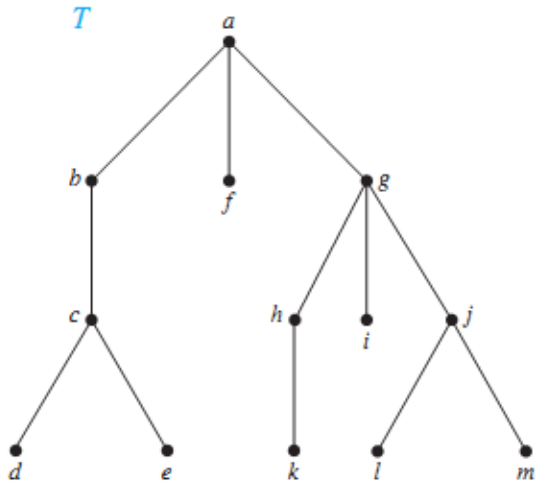
Сыновьями вершины g являются h, i и j .

Братья вершины h — это i и j .

Терминология деревьев:предки и потомки

Предки вершины, отличной от корня, — это вершины на пути от корня к этой вершине , исключая саму вершину и включая корень (то есть ее родителя, родителя ее родителя и т. д.) до тех пор, пока не будет достигнут корень).

Потомками вершины *v* являются те вершины, которые имеют *v в* качестве предка.



Пример

Предками вершины *e* являются *c*, *b* и *a* .
Потомками вершины *b* являются *c*, *d* и *e* .

Терминология деревьев:лист, внутренняя вершина, поддерево

Вершина корневого дерева называется листом, если у нее нет сыновей.

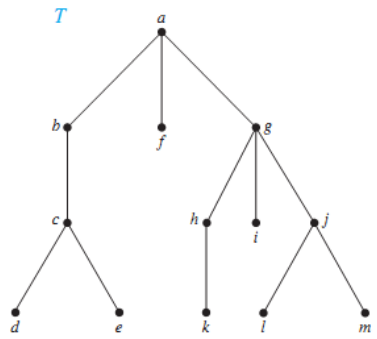
Вершины, у которых есть сыновние вершины, называются внутренними вершинами.

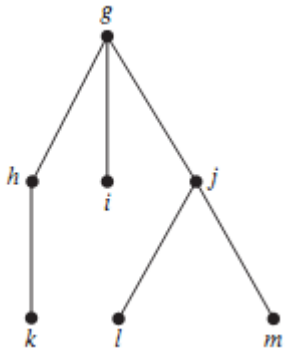
Корень является внутренней вершиной, если только он не является единственной вершиной в графе; в этом случае он является листом.

Если *a* — вершина дерева, то поддерево с корнем *a* — это *подграф* дерева, состоящий из *a* и его потомков, а также всех ребер, инцидентных этим потомкам .

Пример : Внутренние вершины — *a, b, c, g, h* и *j* .

Листья — это *d, e, f, i, k, l* и *m* .





Поддерево дерева *T* с корнем *g* .

Терминология деревьев *m* - арное дерево

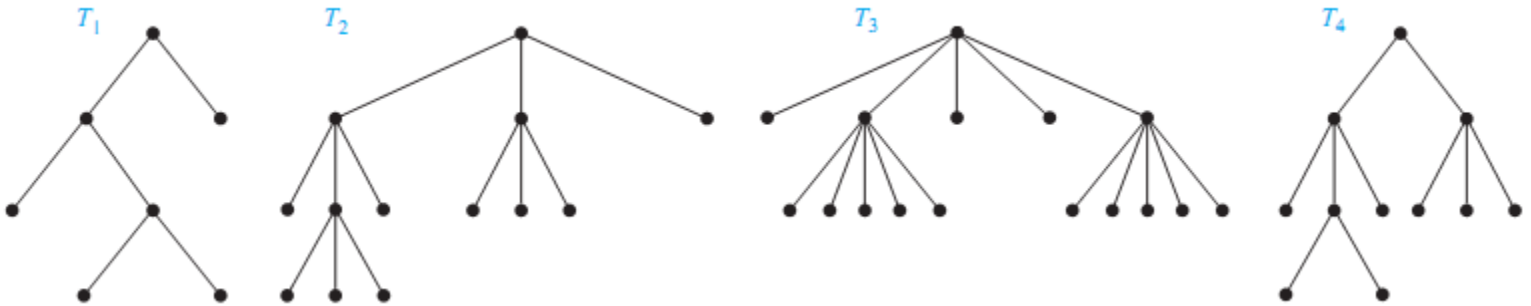
ОПРЕДЕЛЕНИЕ 3 Корневое дерево называется *m* - арным деревом, если каждая внутренняя вершина имеет *не* более *m* сыновей.

Дерево называется полным *m* - арным деревом, если каждая внутренняя вершина имеет ровно *m* сыновей.

m - арное дерево, где *m* = 2 называется бинарным деревом .

Терминология деревьев *m* - арное дерево

ПРИМЕР Являются ли корневые деревья T_1, T_2, T_3, T_4 полными *m* -арными деревьями для некоторого положительного целого числа *m* ?



Решение: T_1 — полное бинарное дерево, т.к. каждая из его внутренних вершин имеет 2 сына.

T_2 является полным 3-арным деревом, поскольку каждая из его внутренних вершин имеет 3 сына.

В T_3 каждая внутренняя вершина имеет 5 сыновей, поэтому T_3 — это полное 5-арное дерево .

T_4 *не* является полным *m*-арным деревом для любого *m,* т.к. некоторые его внутренние вершины имеют 2 сына, а другие — 3 сына.

УПОРЯДОЧЕННЫЕ КОРНЕВЫЕ ДЕРЕВЬЯ

Упорядоченное корневое дерево — это корневое дерево, в котором дочерние вершины каждой внутренней вершины упорядочены.

Упорядоченные корневые деревья рисуются таким образом, что дочерние элементы каждой внутренней вершины изображаются в порядке слева направо.

Обратите внимание, что представление корневого дерева обычным способом определяет порядок его ребер.

В упорядоченном бинарном дереве (обычно называемом просто бинарным деревом), если внутренняя вершина имеет 2 сына, первый сын называется левым сыном, а второй сын называется правым сыном .

Дерево, корнем которого является левый сын вершины, называется левым поддеревом этой вершины, а дерево, корнем которого является правый сын вершины, называется правым поддеревом вершины.

Обратите внимание, что в некоторых приложениях каждая вершина бинарного дерева, кроме корня, обозначается как правый или левый сын своего отца.

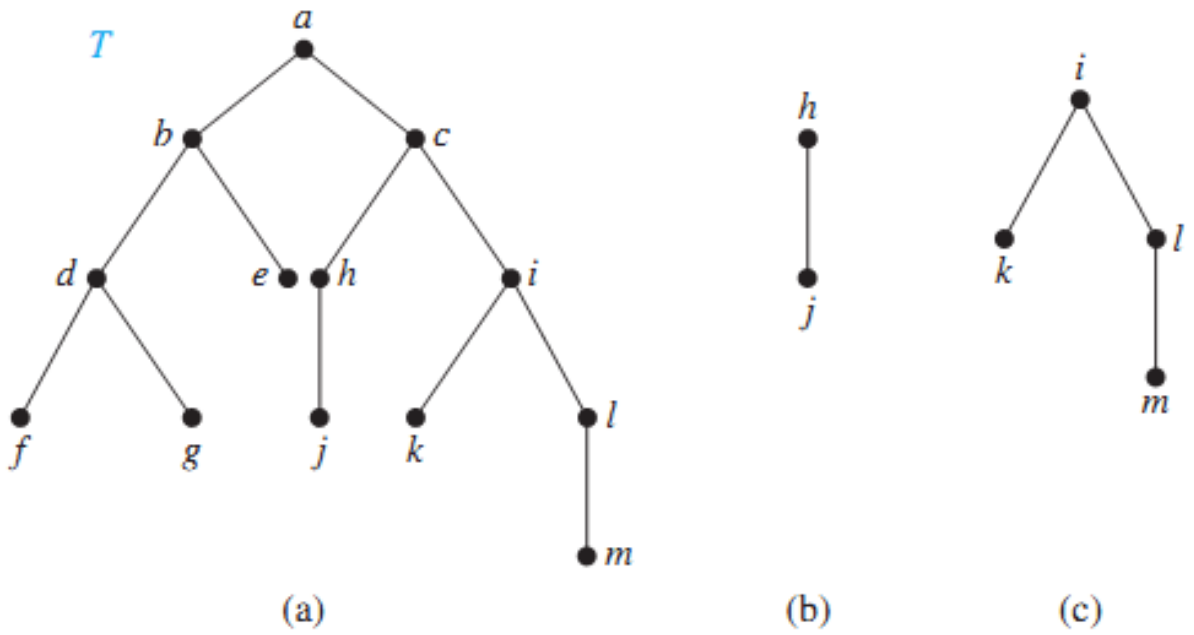
Это делается даже в том случае, если некоторые вершины имеют только 1 сына.

ПРИМЕР Чему равны левый и правый сын вершины *d* в бинарном дереве *T?*

Чему равны левое и правое поддеревья вершины *c* ?

Решение : Левый сын вершины *d* — это *f,* а правый сын — это *g* .

Левое и правое поддеревья вершины *c* показаны ниже на рисунках (b) и (c) соответственно.



Деревья в качестве графовых моделей

Деревья используются в качестве моделей в таких разнообразных областях, таких как информатика, химия, геология, ботаника, психология и др.

ПРИМЕР: Насыщенные углеводороды и деревья

Графы можно использовать для представления молекул, где атомы являются вершинами, а связи между ними — ребрами.

Английский математик Артур Кэли открыл деревья в 1857 году, когда пытался перечислить изомеры соединений вида $C_n H_{2n+2}$, которые называются *насыщенными углеводородами* .

В графовых моделях насыщенных углеводородов каждый атом углерода представлен вершиной степени 4, а каждый атом водорода представлен вершиной степени 1.

В графе, представляющем соединение вида $C_n H_{2n+2,}$ имеется $3n + 2$ вершин.

Количество ребер в таком графе равно 1/2 суммы степеней вершин.

В этом графе имеется $(4n + 2n + 2)/2 = 3n + 1$ ребер.

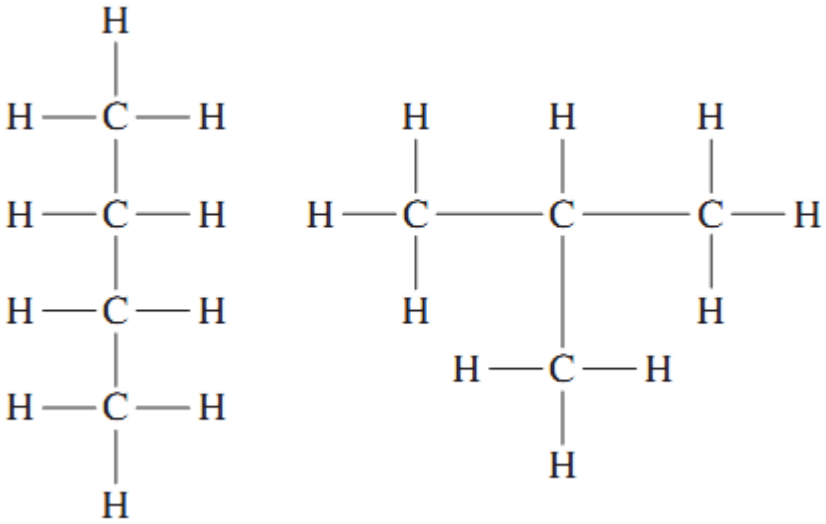
Поскольку граф связный и $m= n -1$, он обязан быть деревом .

Неизоморфные деревья с n вершинами степени 4 и $2n+2$ вершин степени 1 представляют собой различные изомеры C_nH_{2n+2} .

Например, при $n = 4$ существует ровно 2 неизоморфных дерева этого типа.

То есть, существует ровно 2 различных изомера C_4H_{10} .

Эти изомеры называются бутан и изобутан.

бутан  изобутан .

ПРИМЕР Представление организаций

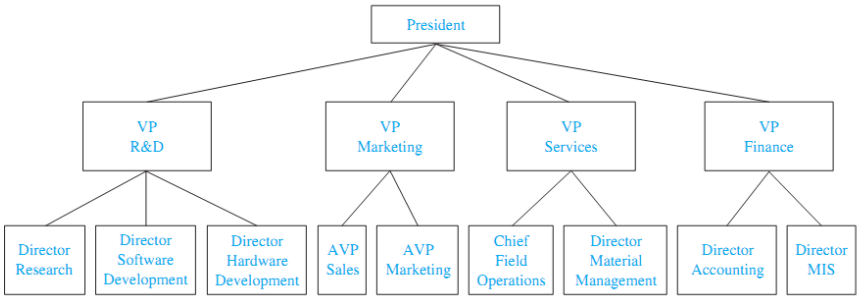
Структуру крупной организации можно смоделировать с помощью корневого дерева.

Каждая вершина этого дерева представляет собой должность в организации.

Ребро от одной вершины к другой указывает на то, что лицо , представленное начальной вершиной, является (прямым) начальником лица, представленного конечной вершиной.

Приведенный ниже граф отображает такое дерево.

В организации, представленной этим деревом, директор по разработке аппаратного обеспечения работает непосредственно под руководством вице-президента по исследованиям и разработкам.



ПРИМЕР Компьютерные файловые системы

Файлы в памяти компьютера можно организовать в каталоги.

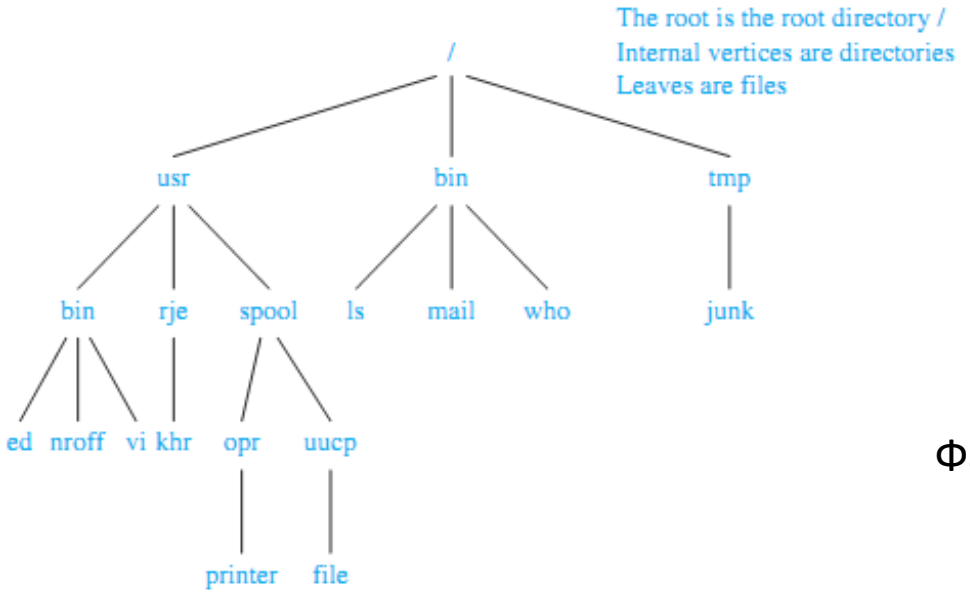
Каталог может содержать как файлы, так и подкаталоги.

Корневой каталог содержит всю файловую систему.

Таким образом, файловая система может быть представлена корневым деревом, где корень представляет корневой каталог , внутренние вершины представляют подкаталоги, а листья представляют обычные файлы или пустые каталоги.

Одна из таких файловых систем показана ниже.

В этой системе файл khr находится в каталоге rje .



Файловая система компьютера.

Свойства деревьев

Нам часто будут нужны результаты, связывающие количество ребер и вершин в деревьях различных типов.

ТЕОРЕМА 2 Дерево с n вершинами имеет $n - 1$ ребро.

Доказательство : Для доказательства этой теоремы мы воспользуемся методом математической индукции.

Обратите внимание, что у любого дерева можно выбрать корень и считать дерево корневым .

БАЗОВЫЙ ШАГ : Когда $n = 1$, дерево с $n = 1$ вершиной не имеет ребер.

Отсюда следует, что теорема верна для $n = 1$.

Свойства деревьев

ИНДУКТИВНЫЙ ШАГ : Индуктивная гипотеза утверждает, что каждое дерево с k вершинами имеет $k - 1$ ребро, где k — положительное целое число.

Предположим теперь, что дерево T имеет $k + 1$ вершину, и что v является листом T (который должен существовать , поскольку дерево конечно), и пусть w является отцом листа v .

Удаляем из T вершину v и ребро, соединяющее w с v.

В результате получится дерево T' с k вершинами, поскольку полученный граф по-прежнему связен и не имеет простых циклов.

По индуктивному предположению T' имеет $k - 1$ ребро.

Отсюда следует, что T имеет k ребер, поскольку у него на одно ребро больше, чем у T' , а именно, имеется ребро, соединяющее v и w .

На этом индуктивный шаг завершен.

Свойства деревьев

Напомним, что дерево — это связный ациклический неориентированный граф.

Итак, когда G — это неориентированный граф с n вершинами, Теорема 2 говорит нам, что из двух условий

(*i*) G связен и

(*ii*) G является ациклическим,

следует (*iii*) G имеет $n - 1$ ребро.

Кроме того, когда выполняются (*i*) и (*iii*), то (*ii*) также должно выполняться, а когда выполняются (*ii*) и (*iii*), то (*i*) также должно выполняться.

То есть, если G связен и G имеет $n - 1$ ребро, то G ацикличен, так что G является деревом.

и если G *ацикличен* и G имеет $n - 1$ ребро , то G связен, и G должен быть деревом.

Следовательно, когда выполняются два из условий (*i*), (*ii*) и (*iii*), третье условие также должно выполняться.

Свойства *m* -арных деревьев

ТЕОРЕМА 3 Полное *m* -арное дерево, имеющее *i* внутренних вершин, всего содержит $n = mi + 1$ вершин.

Доказательство : Каждая вершина, за исключением корня, является сыном внутренней вершины.

Поскольку каждая из *i* внутренних вершин имеет *m* сыновей, в дереве есть *mi* вершин, отличных от корня. Поэтому, дерево содержит $n = mi + 1$ вершин.

Предположим, что *T* — полное *m* - арное дерево.

Пусть *i* будет количеством внутренних вершин,

а *l* - количеством листьев на этом дереве.

Как только одна из величин *n*, *i* или *l* известна,то две другие величины легко вычислить.

ТЕОРЕМА 4 Полное m -арное дерево, имеющее

(*1*) n вершин имеет $i = (n - 1)/m$ внутренних вершин и $l = [(m - 1)n + 1]/m$ листьев,

(*2*) i внутренних вершин имеет $n = mi + 1$ вершин и $l = (m - 1)i + 1$ листьев,

(*3*) l листьев имеет $n = (ml - 1)/(m - 1)$ вершин и $i = (l - 1)/(m - 1)$ внутренних вершин.

Доказательство : Пусть n представляет общее количество вершин, i - количество внутренних вершин, а l - количество листьев.

Все три части теоремы можно доказать, используя равенство, данное в теореме 3, то есть $n = mi + 1$,

вместе с равенством $n = l + i$, что верно, поскольку каждая вершина является либо листом, либо внутренней вершиной.

Докажем часть (*1*) в качестве примера.

Доказательства частей (*2*) и (*3*) выполняются аналогично.

Решение для i в $n = mi + 1$ дает $i = (n - 1)/m$.

Затем вставляем это выражение для i в уравнение $n = l + i$, получим

$l = n - i = n - (n - 1)/m = [(m - 1)n + 1]/m$.

Свойства *m* -арных деревьев

ПРИМЕР. Предположим, что кто-то рассылает «письма счастья».

Каждого, кто получит письмо, просят переслать его копии еще 4 людям.

Некоторые так делают, но другие вообще не отправляют писем.

Сколько человек видели письмо, включая первого человека,

если никто не получит больше 1 письма, и

если цепочка писем заканчивается после того, как 100 человек прочитали его, но не отправили?

Сколько человек отправили письмо?

Свойства *m* -арных деревьев

Решение: Цепочки рассылки писем можно представить с помощью 4-арного дерева.

Внутренние вершины соответствуют людям, отправившим письмо, а листья соответствуют людям, которые его не отправили.

Поскольку 100 человек не отправили письмо, количество листьев в этом корневом дереве равно $l = 100$.

Следовательно, надо применить часть (*3*) теоремы 4, которая показывает,

что количество людей, видевших письмо,

является $n = (ml - 1)/(m - 1) = (4 \cdot 100 - 1)/(4 - 1) = 133$.

Также количество внутренних вершин равно $133 - 100 = 33$.

Стало быть, письмо отправили дальше 33 человека.

Сбалансированные *m*-арные деревья

Часто желательно использовать корневые деревья, которые «сбалансированы», то есть, такие, что поддеревья в каждой вершине содержат пути примерно одинаковой длины.

Введем определения, которые прояснят это понятие.

Уровень вершины *v* в корневом дереве — это длина уникального пути от корня до этой вершины.

Уровень корня определяется как нулевой .

Высота корневого дерева — это максимальный из уровней вершин.

Другими словами, высота корневого дерева равна длине самого длинного пути от корня до любой вершины .

Сбалансированные *m*-арные деревья

ПРИМЕР. Найдите уровень каждой вершины в корневом дереве *T.*

Какова высота этого дерева?

Решение: Корень *a* находится на уровне 0.

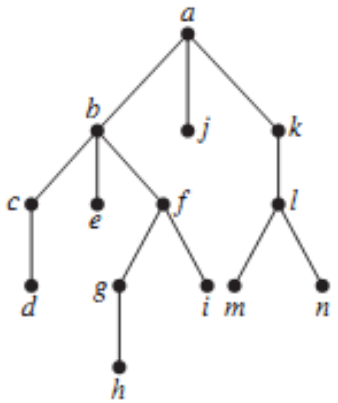
Вершины *b*, *j* и *k* находятся на уровне 1.

Вершины *c*, *e*, *f* и *l* находятся на уровне 2.

Вершины *d*, *g*, *i*, *m* и *n* находятся на уровне 3.

Наконец, вершина *h* находится на уровне 4.

Поскольку наибольший уровень любой вершины равен 4, это дерево имеет высоту 4.



T

Сбалансированные m-арные деревья

Корневое m -арное дерево высоты h является сбалансированным, если все его листья находятся на уровнях h или $h - 1$.

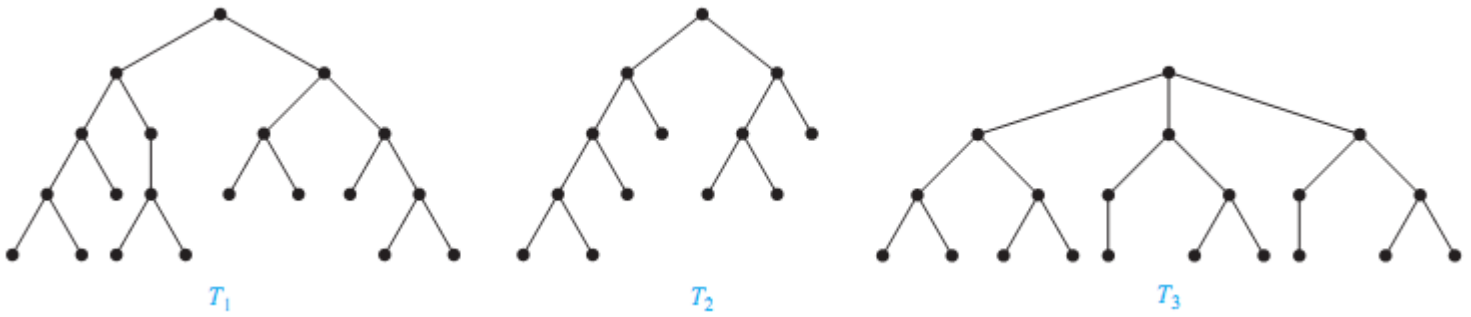
Сбалансированные *m*-арные деревья

ПРИМЕР. Какие из корневых деревьев T_1, T_2, T_3 сбалансированы?

Решение : T_1 сбалансировано, поскольку все его листья находятся на уровнях 3 и 4.

Однако T_2 не сбалансировано, поскольку имеет листья на уровнях 2, 3 и 4.

Наконец, T_3 сбалансировано, поскольку все его листья находятся на уровне 3 .



Связь между высотой дерева и количеством листьев

ТЕОРЕМА 5. В m -арном дереве высотой h имеется не более m^h листьев.

Доказательство: Доказательство использует математическую индукцию по высоте дерева.

Сначала рассмотрим m-арные деревья высоты 1 .

Эти деревья состоят из корня, имеющего не более m сыновей, каждый из которых является листом.

Следовательно, в m -арном дереве высотой 1 имеется не более $m^1 = m$ листьев.

Это базовый шаг индукции.

Связь между высотой дерева и количеством листьев

Теперь предположим, что утверждение верно для всех m -арных деревьев высоты $< h$; это индуктивная гипотеза.

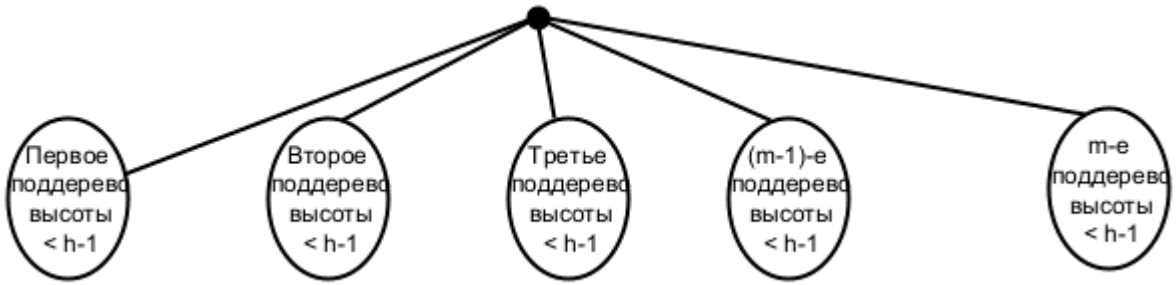
Пусть T — m -арное дерево высоты h .

Листья дерева T — это листья поддеревьев T, полученные путем удаления ребер от корня до каждой из вершин на уровне 1, как показано на рисунке ниже.

Каждое из этих поддеревьев имеет высоту не более чем $h - 1$.

Таким образом, по индуктивной гипотезе каждое из этих корневых деревьев имеет не более чем m^{h-1} листьев.

Поскольку существует не более m таких поддеревьев, каждое из которых имеет максимум m^{h-1} листьев, то существует $l \leq m \cdot m^{h-1} = m^h$ листьев в корневом дереве высоты h.



Индуктивный шаг доказательства.

Связь между высотой дерева и количеством листьев

СЛЕДСТВИЕ 1 Если m - арное дерево высоты h имеет l листьев, то $h \geq \lceil \log_m l \rceil$.
Если m - арное дерево полное и сбалансированное , то $h = \lceil \log_m l \rceil$.
(Здесь мы используем функцию потолка.
Напомним, что $\lceil x \rceil$ — наименьшее целое число $\geq x$.)

Доказательство : Мы знаем, что $l \leq m^h$ из теоремы 5.
Возьмем логарифм по основанию m от обеих частей: $\log_m l \leq h$.
Поскольку h — целое число, то $h \geq \lceil \log_m l \rceil$.
Теперь предположим, что дерево сбалансировано.
Тогда каждый лист находится на уровне h или $h - 1$, и поскольку высота дерева равна h, на уровне h имеется по крайней мере 1 лист .
Из этого следует, что листьев должно быть $\geq m^{h-1}$.
Поскольку $l \leq m^h$, то $m^{h-1} < l \leq m^h$.
Взяв логарифмы по основанию m в этом неравенстве, получаем $h - 1 < \log_m l \leq h$.
Следовательно, $h = \lceil \log_m l \rceil$.

Приложения деревьев

Мы обсудим две проблемы, которые можно изучить с помощью деревьев.

1) Как оценить сложность задачи, решаемой при помощи деревьев принятия решений?
2) Как эффективно кодировать набор символов с помощью битовых строк?

Деревья решений

Корневые деревья можно использовать для моделирования задач, в которых ряд сделанных выборов приводит к решению проблемы.

Например, двоичное дерево поиска можно использовать для поиска элементов на основе серии сравнений, где каждое сравнение сообщает нам, нашли ли мы элемент или следует ли нам идти вправо или влево в поддереве.

Деревья решений

Корневое дерево, в котором каждая внутренняя вершина соответствует принятию решения, а поддерево с корнем в этой вершине соответствует каждому возможному результату принятия решения, называется деревом решений.

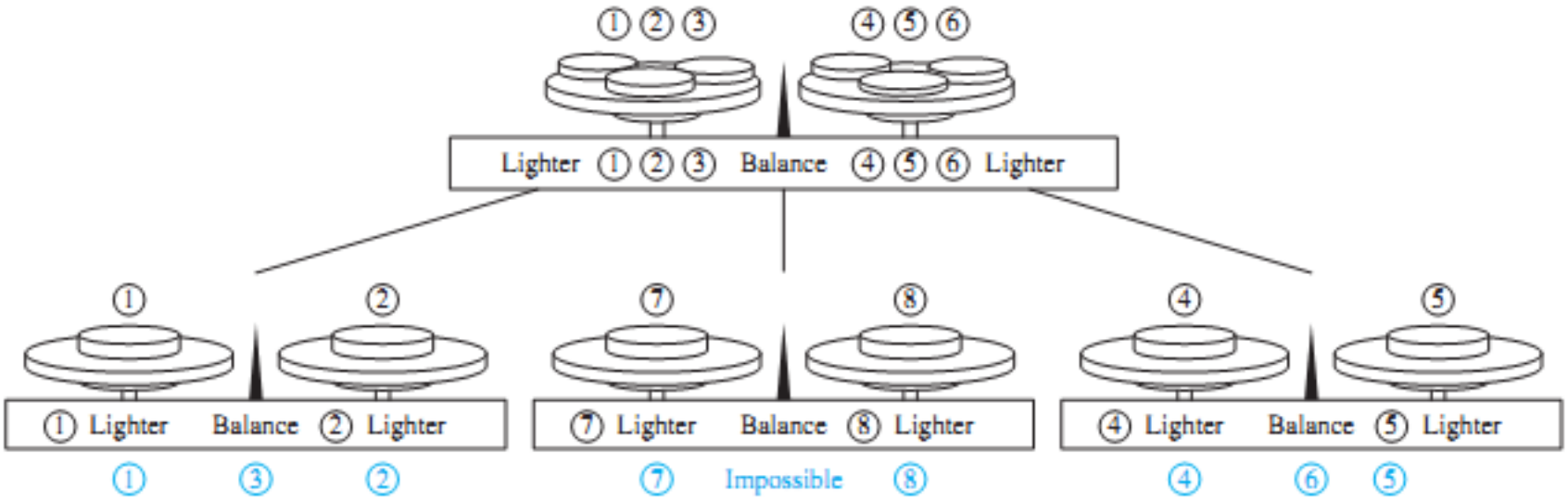
Возможные решения проблемы соответствуют путям к листьям этого корневого дерева

Оценка количества взвешиваний при помощи дерева решений

ПРИМЕР 1. Имеется 8 монет : 7 одинаковых по весу и одна фальшивая монета, которая весит меньше остальных.

Сколько взвешиваний необходимо произвести на весах, чтобы определить, какая из 8 монет фальшивая?

Ниже показан алгоритм поиска этой фальшивой монеты.



Оценка количества взвешиваний при помощи дерева решений

Решение : Для каждого взвешивания на весах существует 3 варианта.

Две чаши могут иметь одинаковый вес, первая чаша может быть тяжелее, или вторая чаша может быть тяжелее .

Следовательно, дерево решений для последовательности взвешиваний представляет собой 3-арное дерево .

В дереве решений имеется не менее 8 листьев, поскольку существует 8 возможных результатов (каждая из 8 монет может быть фальшивой более легкой монетой), и каждый возможный результат должен быть представлен не менее чем 1 листом.

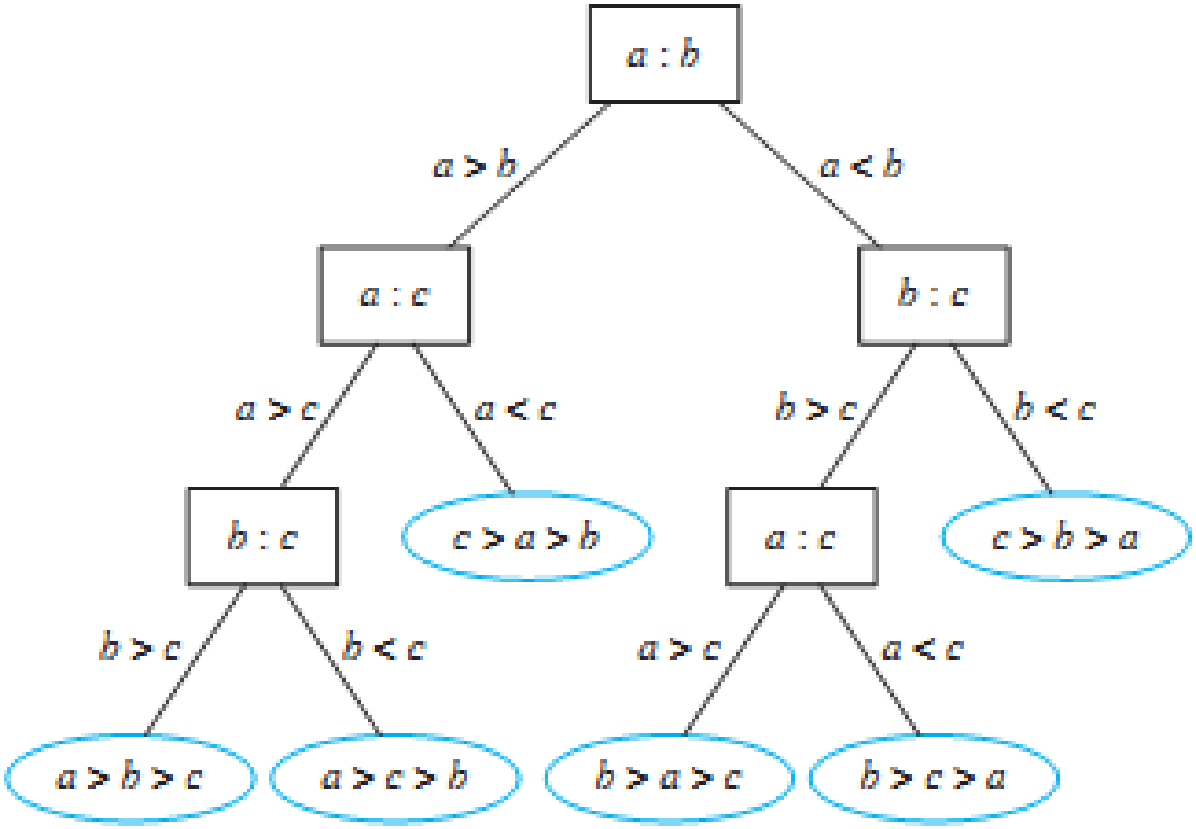
Наибольшее количество взвешиваний, необходимое для определения фальшивой монеты равно высоте дерева решений.

Из следствия 1 следует, что высота дерева решений $\geq \log_3 8 = 2$.

Следовательно, в данном случае необходимо 2 взвешивания .

Определить фальшивую монету можно с помощью 2 взвешиваний.

Пример 2. Дерево решений для сортировки трех различных элементов.



Сложность алгоритмов сортировки, основанных на попарном сравнении

Разработано множество различных алгоритмов сортировки.

Чтобы решить, эффективен ли конкретный алгоритм сортировки, определяется его сложность.

Используя деревья решений в качестве моделей, можно найти нижнюю границу для наихудшего случая сложности алгоритмов сортировки , основанных на двоичных сравнениях.

Мы можем использовать деревья решений для моделирования алгоритмов сортировки и определения оценки наихудшей сложности этих алгоритмов.

Обратите внимание, что при наличии n элементов существует $n!$ возможных упорядочений этих элементов, поскольку каждая из $n!$ перестановок этих элементов может быть правильным порядком.

Наиболее часто используемые алгоритмы сортировки основаны на попарных сравнениях , то есть сравнении двух элементов.

Таким образом, алгоритм сортировки, основанный на попарных сравнениях, можно представить в виде бинарного дерева решений, в котором каждая внутренняя вершина представляет собой сравнение двух элементов.

Каждый лист представляет собой одну из $n!$ перестановок n элементов.

Сложность алгоритмов сортировки, основанных на попарном сравнении

Сложность сортировки, основанной на попарных сравнениях, измеряется количеством используемых сравнений.

Наибольшее количество попарных сравнений, которое может потребоваться для сортировки списка из n элементов, дает наихудший случай производительности алгоритма.

Наибольшее количество использованных сравнений соответствует наибольшей длине пути в дереве решений, представляющем процедуру сортировки.

Другими словами, наибольшее количество сравнений, которое может потребоваться, равно высоте дерева решений.

Поскольку высота бинарного дерева с $n!$ листьями $\geq \lceil \log n! \rceil$ (используя следствие 1), необходимы $\geq \lceil \log n ! \rceil$ сравнений, как указано в теореме 1.

Сложность алгоритмов сортировки, основанных на попарном сравнении

ТЕОРЕМА 1 Алгоритм сортировки, основанный на двоичных сравнениях, требует $\geq \lceil \log n! \rceil$ сравнений.

Сложность алгоритмов сортировки, основанных на попарном сравнении

Мы можем использовать теорему 1 , чтобы получить Ω - оценку для числа сравнений с помощью алгоритма сортировки, основанного на двоичном сравнении.

Нам нужно только отметить, что $\lceil \log n! \rceil$

это $\Theta(n \log n)$,

Следствие 1 является следствием этой оценки.

Сложность алгоритмов сортировки, основанных на попарном сравнении

$O(log_2(n!)) = O(nlogn)$

(a) $O(\log(n!)) \in O(n\log n)$

$\log_2(n!) = \log_2(1) + \log_2(2) + \log_2(3) + \ldots \log_2(n-1) + \log_2(n)$

$\leq \log_2(n) + \log_2(n) + \log_2(n) + \ldots + \log_2(n) = n\log_2 n$

Пусть $c = 1, N_0 = 1 \quad \forall n \geq N_0 \; log_2(n!) \leq c \cdot n log_2(n)$

Сложность алгоритмов сортировки, основанных на попарном сравнении

б) $\log_2(n!) = \log_2(n) + \log_2(n-1) + \ldots + \log_2(n/2) +$

$\log_2(n/2 - 1) + \ldots + \log_2(4) + \log_2(3) + \log_2(2) + \log_2(1) >$

$(n/2)\log_2(n/2) + (n/2)\log_2(2) =$

$= (n/2)\log_2(n) - (n/2)\log_2(2) + (n/2)\log_2(2) =$

$= (n/2)\log_2(n)$

$= (1/2)(n\log_2 n);$

$2\log_2(n!) > n\log_2(n).$

Пусть $c = 2$, $N_o = 1$ $\forall$ $n \geq N_0$

$n\log_2(n) \leq c\log_2(n!)$

$N\log_2(n) \in O(\log_2(n!))$

Сложность алгоритмов сортировки, основанных на попарном сравнении

СЛЕДСТВИЕ 2 Число сравнений, используемых алгоритмом сортировки для сортировки n элементов на основе попарных сравнений, равно $\Omega(n \log n)$.

Сложность алгоритмов сортировки, основанных на попарном сравнении

Следствием следствия 2 является то, что алгоритм сортировки, основанный на попарных сравнениях, который в худшем случае использует $\Theta(n \log n)$ сравнений для сортировки n элементов, является оптимальным в том смысле, что ни один другой такой алгоритм не имеет лучшей сложности в наихудшем случае.

Префиксные коды

Рассмотрим задачу использования битовых строк для кодирования букв английского алфавита (где не делается различий между строчными и заглавными буквами).

Мы можем представить каждую букву английского алфавита битовой строкой длиной 5, поскольку существует всего 26 букв и 32 битовые строки длиной 5.

Общее количество битов, используемых для кодирования данных, в 5 раз превышает количество символов в тексте, если каждый символ кодируется 5 битами.

Можно ли найти такую схему кодирования этих букв, чтобы при кодировании данных использовалось меньше битов?

Если это возможно, мы можем сэкономить память и сократить время передачи.

Префиксные коды

Рассмотрим возможность использования битовых строк разной длины для кодирования букв.

Буквы, которые встречаются чаще, следует кодировать с помощью коротких битовых строк, а для кодирования редко встречающихся букв следует использовать более длинные битовые строки .

Когда буквы кодируются с использованием различного количества битов, необходимо использовать какой-то метод для определения того, где начинаются и заканчиваются биты для каждого символа.

Например, если буква *e* была закодирована с помощью 0, *a* с помощью 1 и *t* с помощью 01, то битовая строка 0101 может соответствовать словам eat, tea, eaea или tt .

Префиксные коды

Один из способов убедиться, что ни одна строка битов не соответствует более чем одной последовательности букв — это кодировать буквы таким образом, чтобы битовая строка для буквы никогда не встречалась как начало битовой строки для другой буквы.

Коды с таким свойством называются **префиксными** кодами.

Например, кодирование *e* как 0, *a* как 10 и *t* как 11 является префиксным кодом.

Слово можно восстановить из уникальной битовой строки, кодирующей его буквы.

Например, строка 10 11 0 является кодировкой слова ate .

Чтобы увидеть это, обратите внимание, что начальная цифра 1 не представляет никакую букву, а вот 10 представляет букву *a* (и не может быть первой частью битовой строки другой буквы).

Тогда следующая 1 не представляет никакую букву, а 11 представляет букву *t*.

Последний бит, 0 , представляет *e*.

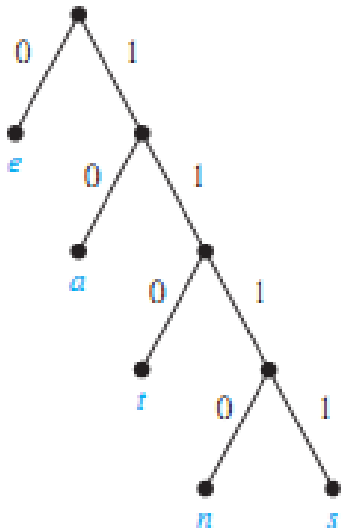
Префиксные коды

Префиксный код можно представить с помощью двоичного дерева, где символы являются метками листьев в дереве.

Ребра дерева помечены так, что ребру, ведущему к левому сыну , присваивается 0, а ребру, ведущему к правому сыну, присваивается 1 .

Битовая строка, используемая для кодирования буквы, представляет собой последовательность меток ребер на уникальном пути от корня до листа, имеющего эту букву в качестве метки.

Например, показанное ниже дерево представляет собой кодировку *e* с помощью 0, *a* с помощью 10, *t* с помощью 110, *n* с помощью 1110 и *s* с помощью 1111 .

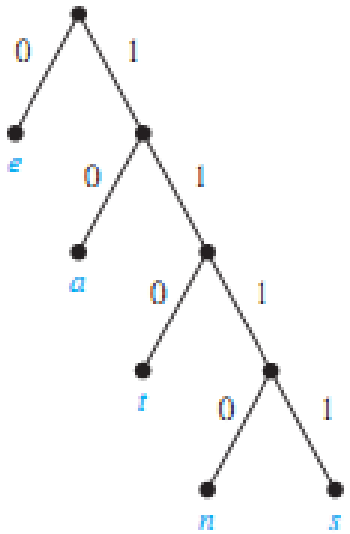


Префиксные коды

Дерево, представляющее префиксный код, можно использовать для декодирования строки битов.

Например, рассмотрим слово, закодированное как 1111 10 1110 0, используя приведенный ниже код.

Эту строку битов можно декодировать, начиная с корня, используя последовательность битов для формирования пути, который останавливается при достижении листа.



КОДИРОВАНИЕ ХАФФМАНА

Теперь мы представим алгоритм, который принимает в качестве входных данных частоты (вероятности появления) символов в строке и выдает в качестве выходных данных префиксный код, который кодирует строку, используя наименьшее возможное количество битов среди всех возможных двоичных префиксных кодов для этих символов.

Этот алгоритм, известный как кодирование Хаффмана, был разработан Дэвидом Хаффманом в курсовой работе, которую он написал в 1951 году, будучи аспирантом Массачусетского технологического института.

(Этот алгоритм предполагает, что мы уже знаем, сколько раз каждый символ встречается в строке, поэтому мы можем вычислить частоту каждого символа, разделив количество появлений этого символа на длину строки.)

Кодирование Хаффмана — это фундаментальный алгоритм сжатия данных, направленный на сокращение количества битов, необходимых для представления информации.

Кодирование Хаффмана широко используется для сжатия строк битов, представляющих текст, а также играет важную роль в сжатии аудиофайлов и файлов изображений .

КОДИРОВАНИЕ ХАФФМАНА

АЛГОРИТМ 2 Кодирование Хаффмана.

процедура Huffman (C : буквы a_i с частотами w_i , $i = 1, ..., n$)

F := лес из n корневых деревьев, каждое из которых состоит из одной вершины a_i и присвоеннного веса w_i

пока F не дерево

Заменить корневые деревья T и T' наименьших весов из F на дерево, имеющее новый корень и имеющее T в качестве своего левого поддерева и T' в качестве правого поддерева

$w(T) \geq w(T')$.

Пометить новое ребро к T нулем, а новое ребро к T' - единицей.

Присвоить $w(T) + w(T')$ в качестве веса нового дерева.

{код Хаффмана для символа a_i представляет собой конкатенацию меток ребер в уникальном пути от корня до вершины a_i }

КОДИРОВАНИЕ ХАФФМАНА

Алгоритм 2 представляет собой алгоритм кодирования Хаффмана.

Для заданных букв и их частот надо построить корневое двоичное дерево, где буквы являются метками листьев .

Алгоритм начинается с леса деревьев, каждое из которых состоит из 1 вершины, где каждая вершина имеет букву в качестве своей метки, а вес этой вершины равен частоте буквы, являющейся ее меткой.

На каждом шаге мы объединяем 2 дерева с наименьшим весом в одно дерево, вводя новый корень и помещая дерево с большим весом в качестве его левого поддерева, а дерево с меньшим весом — в качестве его правого поддерева .

Кроме того, мы присваиваем сумму весов двух поддеревьев этого дерева в качестве общего веса дерева .

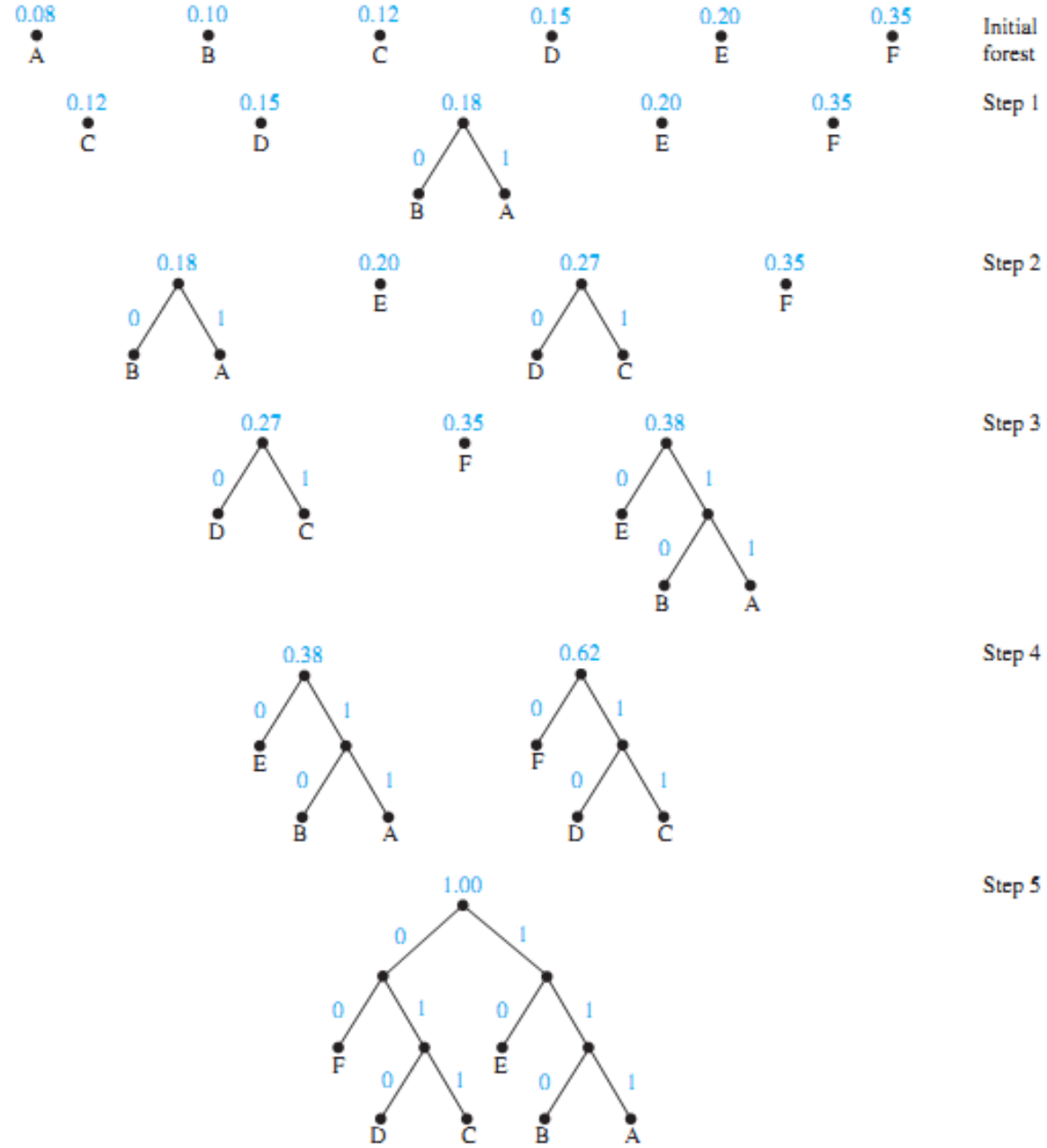
Алгоритм завершается, когда построено дерево, то есть когда лес сводится к одному дереву.

КОДИРОВАНИЕ ХАФФМАНА

ПРИМЕР Используйте кодирование Хаффмана для кодирования следующих символов с указанными частотами:

A : 0,08, *B* : 0,10, *C* : 0,12, *D* : 0,15, *E* : 0,20, *E* : 0,35.

Какое среднее количество битов используется для кодирования символа?

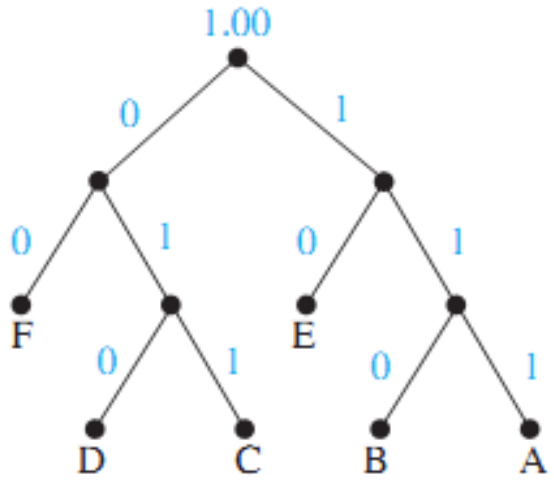

0.08
A
0.10
B
0.12
C
0.15
D
0.20
E
0.35
F
Initial forest
0.12
C
0.15
D
0.18
0
1
B
A
0.20
E
0.35
F
Step 1
0.18
0
1
B
A
0.20
E
0.27
0
1
D
C
0.35
F
Step 2
0.27
0
1
D
C
0.35
F
0.38
0
1
E
0
1
B
A
Step 3
0.38
0
1
E
0
1
B
A
0.62
0
1
F
0
1
D
C
Step 4
1.00
0
1
0
1
F
0
1
E
0
1
D
C
B
A
Step 5

Решение: полученное кодирование кодирует

A как 111, *B* как 110, *C* как 011, *D* как 010, *E* как 10 и *F* как 00.

Среднее число битов, используемых для кодирования символа с помощью этой кодировки, равно

$3 \cdot 0{,}08 + 3 \cdot 0{,}10 + 3 \cdot 0{,}12 + 3 \cdot 0{,}15 + 2 \cdot 0{,}20 + 2 \cdot 0{,}35 = 2{,}45$.



КОДИРОВАНИЕ ХАФФМАНА

Обратите внимание, что кодирование Хаффмана — это жадный алгоритм.

Замена двух поддеревьев наименьшего веса на каждом шаге приводит к оптимальному коду в том смысле, что не существует двоичного префиксного кода для этих символов, которым можно закодировать эти символы, используя меньшее количество битов.

Существует множество разновидностей кодирования Хаффмана.

Для кодирования исходных букв можно использовать более двух символов.

Кроме того, вместо кодирования одной буквы, мы можем кодировать блоки букв определенной длины, например, блоки из 2 букв.

Это может сократить количество битов, необходимых для кодирования строки.

- Ваши вопросы?
- Контакты лектора:
  apanovich_09@mail.ru