Распределенные информационно-аналитические системы

Лекция № 13.

Временная сложность: поиск в глубину

Профессор кафедры КБ-2: д.т.н. Шатовкин Р.Р.

Учебные цели:

Изучить основы научных знаний по классическому алгоритму поиска в глубину; алгоритмам поиска в глубину за линейное время; алгоритмам поиска в глубину со знанием соседей; альтернативным определениям временной сложности. Сравнить алгоритмы поиска в глубину.

Учебные вопросы:

1. Классический алгоритм поиска в глубину.

2. Алгоритмы поиска в глубину за линейное время.

3. Алгоритм поиска в глубину со знанием соседей.

4. Сравнение алгоритмов поиска в глубину.

5. Альтернативные определения временной сложности.

Процессы в алгоритме Тарри достаточно свободны в выборе соседа, которому передается маркер, в результате чего возникает большой класс остовных деревьев.

Обсудим алгоритмы, которые вычисляют остовные деревья с дополнительным свойством: ***каждое листовое ребро соединяет две вершины, одна из которых является предком другой***.

Листовое ребро – это ребро, не принадлежащее остовному дереву. В данном корневом остовном дереве T сети G для каждого p T[p] обозначает множество процессов в поддереве p, а A[p] обозначает предков p, то есть вершины на пути в T от корня до p. Заметим, что $q \in T[p] \Leftrightarrow p \in A[q]$.

Определение 1. Остовное дерево T сети G называется ***деревом поиска в глубину***, если для каждого листового ребра pq $q \in T[p] \vee q \in A[p]$.

Деревья поиска в глубину, или DFS-деревья (DFS – Depth-First Search), используются во многих алгоритмах на графах, таких как проверка планарности, проверка двусвязности, и для построения интервальных схем маркировки.

Определение 2. ***Временная сложность распределенного алгоритма*** – это максимальное время, требуемое на вычисление алгоритма при следующих предположениях:

T1 – процесс может выполнить любое конечное количество событий за нулевое время;

T2 – промежуток времени между отправлением и получением сообщения - не более одной единицы времени.

Лемма 1. Для алгоритмов обхода временная сложность равна сложности сообщений.

Доказательство. Сообщения пересылаются последовательно, и каждое может занять одну единицу времени.

1. Классический алгоритм поиска в глубину

Классический алгоритм поиска в глубину получается, когда свобода выбора соседа для передачи маркера в алгоритме Тарри ограничивается ***следующим правилом:***

R3: когда процесс получает маркер, он отправляет его обратно через тот же самый канал, если это позволяется правилами R1 и R2, где правила:

R1: процесс никогда не передает маркер дважды по одному и тому же каналу.

R2: не-инициатор передает маркер своему родителю (соседу, от которого он впервые получил маркер), только если невозможна передача по другим каналам, в соответствии с правилом R1.

Теорема 1. Классический алгоритм поиска в глубину вычисляет остовное дерево поиска в глубину, используя 2|E| сообщений и 2|E| единиц времени.

Доказательство. Так как алгоритм реализует алгоритм Тарри, это алгоритм обхода и он вычисляет остовное дерево T. Каждый канал передает два сообщения (по одному в обоих направлениях), что доказывает оценку сложности сообщений, а оценка для временной сложности следует из того, что 2|E| сообщений передаются одно за другим, причем каждое занимает не более одной единицы времени. Остается показать, что из правила R3 следует, что получающееся дерево – дерево поиска в глубину.

Из R3 следует, что за первым проходом по листовому ребру немедленно следует второй, в обратном направлении. Пусть pq – листовое ребро и p первым использует ребро. Когда q получает маркер от p, q уже посещен (иначе q присвоит $father_q$ p и ребро не будет листовым) и $used_p$ [q] = false (так как по предположению p первый из двух процессов использовал ребро). Следовательно, по правилу R3, q немедленно отправляет маркер обратно p.

Теперь можно показать, что если pq – листовое ребро, используемое сначала p, то $q \in A[p]$. Рассмотрим путь, пройденный маркером, пока он не был переслан через pq. Так как pq – листовое ребро, q был посещен до того, как маркер попал в q через это ребро: ..., q, ..., p, q

Получим из этого пути возможно более короткий путь, заменив все комбинации r_1, r_2, r_1 (где r_1r_2 – листовое ребро) на r_1. Из предыдущих наблюдений, все листовые ребра теперь убраны, откуда следует, что получившийся путь – это путь в T, состоящий только из ребер, пройденных до первого прохождения pq. Если q не является предком p, то отсюда следует, что ребро от q до $father_q$ было пройдено до того, как было пройдено ребро qp, что противоречит правилу R2 алгоритма.

Классический алгоритм поиска в глубину:

var $used_p[q]$: boolean init false для всех $q \in Neigh_p$;

(Признак того, отправил ли p сообщение q)

$father_p$: process init udef ;

Для инициатора (выполняется один раз):

begin $father_p := p$;

выбор $q \in Neigh_p$;

$used_p[q] := true$;

send <tok > to q ;

end

Для каждого процесса при получении <tok > от q_0:

begin if $father_p$ = udef then fatherp := q_0 ;

if $\forall q \in Neigh_p : used_p[q]$

then decide

else if $\exists q \in Neigh_p : (q \neq fatherp \ \& \ \neg used_p[q])$

then begin if $father_p \neq q_0 \ \& \ \neg used_p[q_0]$

then $q := q_0$

else выбор $q \in Neigh_p \setminus \{father_p\}$ с $\neg used_p[q]$;

$used_p[q] := true$;

send <tok > to q

end

else begin $used_p[father_p] := true$;

send <tok > to $father_p$

end

end

Сложность сообщений классического распределенного поиска в глубину равна 2|E|. Это наилучшая сложность (за исключением множителя 2), если идентификация соседей не известна изначально. Временная сложность также составляет 2|E|, что по Лемме 1, является наилучшей сложностью для алгоритмов обхода в этом случае. Распределенная версия поиска в глубину была впервые представлена Cheung.

2. Алгоритмы поиска в глубину за линейное время

Причина высокой временной сложности классического алгоритма поиска в глубину состоит в том, что все ребра, как принадлежащие дереву, так и листовые, обходятся последовательно. Сообщение-маркер <tok> проходит по всем листовым ребрам и немедленно возвращается обратно, как показано в доказательстве Теоремы 1. Все решения с меньшей временной сложностью основаны на том, что маркер проходит только по ребрам дерева. Очевидно, на это потребуется линейное время, так как существует только N – 1 ребро дерева.

Решение Авербаха. В алгоритм включается механизм, который предотвращает передачу маркера через листовое ребро. В алгоритме Авербаха гарантируется, что каждый процесс в момент, когда он должен передать маркер, знает, какие из его соседей уже были пройдены. Затем процесс выбирает непройденного соседа, или, если такого не существует, посылает маркер своему родителю.

Когда процесс p впервые посещается маркером (для инициатора это происходит в начале алгоритма), p сообщает об этом всем соседям r, кроме его родителя, посылая сообщения <vis>. Передача маркера откладывается, пока p не получит сообщения <ack> от всех соседей. При этом гарантируется, что каждый сосед r процесса p в момент, когда p передает маркер, знает, что p был посещен. Когда позже маркер прибывает в r, r не передаст маркер p, если только p не его родитель.

Из-за передачи сообщений <vis> в большинстве случаев $used_p\ [father_p] = true$, даже если p еще не послал маркер своему родителю. Поэтому в алгоритме должно быть явно запрограммировано, что только инициатор может принимать решения; а не-инициатор p, для которого $used_p\ [q] = true$ для всех соседей q, передает маркер своему родителю.

Теорема 2. Алгоритм Авербаха вычисляет дерево поиска в глубину за 4N –2 моментов времени и использует 4|E| сообщений.

Доказательство. По существу, маркер передается по тем же самым каналам, как и в классическом алгоритме поиска в глубину, за исключением того, что пропускается передача по листовым каналам. Так как передача по листовым каналам не влияет на конечное значение $father_p$ для любого процесса p, то в результате всегда получается дерево, которое может получиться и в классическом алгоритме поиска в глубину.

Маркер последовательно проходит дважды через каждый из N – 1 каналов дерева, на что тратится 2N – 2 единиц времени. В каждой вершине маркер простаивает перед тем, как быть переданным, не более одного раза из-за обмена сообщениями <vis> / <ack>, что приводит к задержке не более чем на 2 единицы времени в каждой вершине.

Каждое листовое ребро передает два сообщения <vis> и два сообщения <ack>. Каждое ребро в дереве передает два сообщения <tok>, одно <vis> (посылаемое от родителя потомку), и одно <ac> (от потомка родителю). Следовательно, передается 4|E| сообщений.

***Алгоритм поиска в глубину Авербаха*:**

var $used_p$ [q] : boolean init false для всех $q \in Neigh_p$;

(Признак того, отправил ли p сообщение q)

$father_p$: process init udef ;

Для инициатора (выполняется один раз):

begin $father_p := p$; выбор $q \in Neigh_p$;

forall $r \in Neigh_p$ do send <vis> to r ;

forall $r \in Neigh_p$ do receive <ack> from r ;

$used_p$ [q] := true ; send <tok> to q ;

end

Для каждого процесса при получении <tok> от q_0:

begin if $father_p$ = udef then

begin $father_p := q_0$;

forall $r \in Neigh_p \setminus \{father_p\}$ do send <vis> to r ;

forall $r \in Neigh_p \setminus \{father_p\}$ do receive <ack> from r ;

end ;

if p – инициатор and $\forall q \in Neigh_p$: $used_p$ [q]

then decide

else if $\exists q \in Neigh_p$: $(q \neq father_p \ \& \ \neg used_p [q])$

then begin if $father_p \neq q_0 \ \& \ \neg used_p [q_0]$

then $q := q_0$

else выбор $q \in Neigh_p \setminus \{father_p\}$

с $\neg used_p$ [q] ;

$used_p$ [q] := true ; send <tok> to q

end

else begin $used_p [father_p] := true$;

send <tok> to $father_p$

end

end

Для каждого процесса при получении <vis> от q_0:

begin $used_p [q_0] := true$; send <ack> to q_0 end

Передачу сообщения <vis> соседу, которому процесс передает маркер, можно опустить. Это усовершенствование (не выполняемое в алгоритме Авербаха) сберегает по два сообщения для каждого ребра дерева и, следовательно, уменьшает сложность сообщений на 2N – 2 сообщения.

Решение Сайдона. Алгоритм Сайдона улучшает временную сложность алгоритма Авербаха, не посылая сообщения <ack>. В модификации Сайдона маркер передается немедленно, то есть без задержки на 2 единицы времени, внесенной в алгоритм Авербаха ожиданием подтверждения. Этот же алгоритм был предложен Лакшмананом и др.

В алгоритме Сайдона может возникнуть следующая ситуация. Процесс p получил маркер и послал сообщение <vis> своему соседу r. Маркер позже попадает в r, но в момент, когда r получает маркер, сообщение <vis> процесса p еще не достигло r. В этом случае r может передать маркер p, фактически посылая его через листовое ребро. (Заметим, что сообщения <ack> в алгоритме Авербаха предотвращают возникновение такой ситуации.)

Чтобы обойти эту ситуацию процесс p запоминает (в переменной mrs_p), какому соседу он переслал маркер в последний раз. Когда маркер проходит только по ребрам дерева, p получает его в следующий раз от того же соседа mrs_p. В ситуации, описанной выше, p получает сообщение <tok> от другого соседа, а именно, от r; в этом случае маркер игнорируется, но p помечает ребро rp, как пройденное, как если бы от r было получено сообщение <vis>. Процесс r получает сообщение <vis> от p после пересылки маркера p, то есть r получает сообщение <vis> от соседа mrs_r. В этом случае r действует так, как если бы он еще не послал маркер p; r выбирает следующего соседа и передает маркер.

Теорема 3. Алгоритм Сайдона вычисляет DFS-дерево за 2N – 2 единиц времени, используя 4|E| сообщений.

Доказательство. Маршрут маркера подобен маршруту в классическом алгоритме поиска в глубину. Прохождение по листовым ребрам либо пропускается (как в алгоритме Авербаха), либо в ответ на маркер через листовое ребро посылается сообщение <vis>. В последнем случае, процесс, получая сообщение <vis>, продолжает передавать маркер, как если бы маркер был возвращен через листовое ребро немедленно.

Время между двумя успешными передачами маркера по дереву ограничено одной единицей времени. Если маркер послали по ребру дерева процессу p в момент времени t, то в момент t все сообщения <vis> ранее посещенных соседей q процесса p были посланы, и, следовательно, эти сообщения прибудут не позднее момента времени t + 1. Итак, хотя p мог несколько раз послать маркер по листовым ребрам до t + 1, не позднее t + 1 p восстановился после всех этих ошибок и передал маркер по ребру, принадлежащему дереву. Так как должны быть пройдены 2N – 2 ребра дерева, алгоритм завершается за 2N – 2 единиц времени.

Через каждый канал передается не более двух сообщений <vis> и двух <tok>, откуда граница сложности сообщений равна 4|E|.

***Алгоритм поиска в глубину Сайдона (часть 1)*:**

var $used_p$ [q] : boolean init false для всех q $\in$ $Neigh_p$;

$father_p$: process init udef ;

mrs_p : process init udef ;

Для инициатора (выполняется один раз):

begin $father_p$:= p ; выбор q $\in$ $Neigh_p$;

forall r $\in$ $Neigh_p$ do send <vis> to r ;

$used_p$ [q] := true ;

mrs_p := q ; send <tok> to q ;

end

Для каждого процесса при получении <vis> от q_0 :

begin $used_p$ [q_0] := true ;

if q_0 = mrs_p then (интерпретировать, как <tok >)

передать сообщение <tok> как при получении <tok>

end

Алгоритм поиска в глубину Сайдона (часть 2):

Для каждого процесса при получении <tok> от q_0:

begin if $mrs_p \neq$ udef and $mrs_p \neq q_0$

(это листовое ребро, интерпретируем как сообщение <vis>)

then $used_p [q_0]$:= true

else (действовать, как в предыдущем алгоритме)

begin if $father_p$ = udef then

begin $father_p := q_0$;

forall $r \in Neigh_p \setminus \{father_p\}$

do send <vis> to r ;

end ;

if p – инициатор and $\forall q \in Neigh_p : used_p [q]$

then decide

else if $\exists q \in Neigh_p : (q \neq father_p \& \neg used_p [q])$

then begin if $father_p \neq q_0 \& \neg used_p [q_0]$

then $q := q_0$

else выбор $q \in Neigh_p \setminus \{father_p\}$

с $\neg used_p [q]$;

$used_p [q]$:= true ; $mrs_p := q$;

send <tok> to q

end

else begin $used_p [father_p]$:= true ;

send <tok > to $father_p$

end

end

end

Во многих случаях этот алгоритм будет пересылать меньше сообщений, чем алгоритм Авербаха. Оценка количества сообщений в алгоритме Сайдона предполагает наихудший случай, а именно, когда маркер пересылается через каждое листовое ребро в обоих направлениях. Можно ожидать, что сообщения <vis> помогут избежать многих нежелательных пересылок, тогда через каждый канал будет передано только два или три сообщения.

Сайдон замечает, что хотя алгоритм может передать маркер в уже посещенную вершину, он обладает лучшей временной сложностью (и сложностью сообщений), чем алгоритм Авербаха, который предотвращает такие нежелательные передачи. Это означает, что на восстановление после ненужных действий может быть затрачено меньше времени и сообщений, чем на их предотвращение.

Сайдон оставляет открытым вопрос о том, существует ли DFS-алгоритм, который достигает сложности сообщений классического алгоритма, то есть 2|E|, и который затрачивает O(N) единиц времени.

3. Алгоритм поиска в глубину со знанием соседей

Если процессам известны идентификаторы их соседей, проход листовых ребер можно предотвратить, включив в маркер список посещенных процессов. Процесс p, получая маркер с включенным в него списком L, не передает маркер процессам из L. Переменная $used_p$ [q] не нужна, так как если p ранее передал маркер q, то $q \in L$.

Теорема 4. DFS-алгоритм со знанием соседей является алгоритмом обхода и вычисляет дерево поиска в глубину, используя 2N – 2 сообщений за 2N – 2 единиц времени.

У этого алгоритма высокая битовая сложность; если w – количество бит, необходимых для представления одного идентификатора, список L может занять до Nw бит.

Алгоритм поиска в глубину со знанием соседей:

var $father_p$: process init udef ;

Для инициатора (выполняется один раз):

begin $father_p$:= p ; выбор q $\in$ $Neigh_p$;

send <tlist , {p}> to q

end

Для каждого процесса при получении <tlist , L> от q_0 :

begin if $father_p$ = udef then $father_p$:= q_0 ;

if $\exists q \in Neigh_p \setminus L$

then begin выбор q $\in$ $Neigh_p \setminus L$;

send < tlist , $L \cup \{p\}$ > to q

end

else if p – инициатор

then decide

else send < tlist , $L \cup \{p\}$ > to $father_p$

end

4. Сравнение алгоритмов поиска в глубину

В таблице 1 для сравнения приведены рассмотренные алгоритмы поиска в глубину.

Таблица 1 – Алгоритмы поиска в глубину

Алгоритм	Топология	C/D	T	Сообщения	Время
Классический	произвольная	C	да	2|E|	2|E|
Авербаха	произвольная	C	нет	4|E|	4N – 2
Сайдона	произвольная	C	нет	4|E|	2N – 2
Со знанием соседей	произвольная	C	да	2N – 2	2N – 2

В столбце «C/D» отмечено, является ли алгоритм централизованным (C) или децентрализованным (D); столбец «T» определяет, является ли алгоритм алгоритмом обхода; в столбце «Сообщения» дана сложность сообщений; в столбце «Время» дана временная сложность. В этих столбцах N – количество процессов, |E| – количество каналов.

5. Альтернативные определения временной сложности

Временную сложность распределенного алгоритма можно определить несколькими способами. При рассмотрении временной сложности всегда имеется в виду Определение 2, но обсуждаются другие возможные определения.

Определение, основанное на более строгих временных предположениях. Время, потребляемое распределенными вычислениями, можно оценить, используя более строгие временные предположения в системе.

Определение 3. ***Единичная сложность алгоритма*** (one-time complexity) – это максимальное время вычисления алгоритма при следующих предположениях:

O1 – процесс может выполнить любое конечное количество событий за нулевое время;

O2 – промежуток времени между отправлением и получением сообщения – ровно одна единица времени.

Сравним это определение с Определением 2 и заметим, что предположение O1 совпадает с T1. Так как время передачи сообщения, принятое в T2, меньше или равно времени, принятому в O2, можно подумать, что единичная сложность всегда больше или равна временной сложности. Далее можно подумать, что каждое вычисление при предположении T2 выполняется не дольше, чем при O2, и, следовательно, вычисление с максимальным временем также займет при T2 не больше времени, чем при O2. Упущение этого аргумента в том, что отклонения во времени передачи сообщения, допустимые при T2, порождают большой класс возможных вычислений, включая вычисления с плохим временем. Это иллюстрируется ниже для эхо-алгоритма.

Фактически, верно и обратное: временная сложность алгоритма больше или равна единичной сложности этого алгоритма. Любое вычисление, допустимое при предположениях O1 и O2, также допустимо при T1 и T2 и занимает при этом такое же количество времени. Следовательно, наихудшее поведение алгоритма при O1 и O2 включено в Определение 2 и является нижней границей временной сложности.

Теорема 5. Единичная сложность эхо-алгоритма равна O(D). Временная сложность эхо-алгоритма равна Q(N), даже в сетях с диаметром 1.

Доказательство. Для анализа единичной сложности сделаем предположения O1 и O2. Процесс на расстоянии d переходов от инициатора получает первое сообщение <tok> ровно через d единиц времени после начала вычисления и имеет глубину d в возникающем в результате дереве T. (Это можно доказать индукцией по d.)

Пусть DT – глубина T; тогда DT ≤ D и процесс глубины d в T посылает сообщение <tok> своему родителю не позднее (2DT + 1) – d единиц времени после начала вычисления. (Это можно показать обратной индукцией по d.) Отсюда следует, что инициатор принимает решение не позднее 2DT + 1 единиц времени после начала вычисления.

Для анализа временной сложности сделаем предположения T1 и T2. Процесс на расстоянии d переходов от инициатора получает первое сообщение <tok> не позднее d единиц времени после начала вычисления. (Это можно доказать индукцией по d.)

Предположим, что остовное дерево построено через F единиц времени после начала вычисления, тогда F ≤ D. В этом случае глубина остовного дерева DT необязательно ограничена диаметром (как будет показано в вычислении ниже), но так как всего N процессов, глубина ограничена N – 1.

Процесс глубины d в T посылает сообщение <tok> своему родителю не позднее (F + 1) + (DT – d) единиц времени после начала вычисления. (Это можно показать обратной индукцией по d.) Отсюда следует, что инициатор принимает решение не позднее (F + 1) + DT единиц времени после начала вычисления, то есть O(N).

Чтобы показать, что W(N) – нижняя граница временной сложности, построим на клике из N процессов вычисление, которое затрачивает время N. Зафиксируем в клике остовное дерево глубины N – 1 (на самом деле, линейную цепочку вершин). Предположим, что все сообщения <tok>, посланные вниз по ребрам дерева, будут получены спустя 1/N единиц времени после их отправления, а сообщения <tok> через листовые ребра будут получены спустя одну единицу времени. Эти задержки допустимы, согласно предположению T2, и в этом вычислении дерево полностью формируется в течение одной единицы времени, но имеет глубину N – 1. Допустим теперь, что все сообщения, посылаемые вверх по ребрам дерева также испытывают задержку в одну единицу времени; в этом случае решение принимается ровно через N единиц времени с начала вычисления.

Можно спорить о том, какое из двух определений предпочтительнее при обсуждении сложности распределенного алгоритма. Недостаток единичной сложности в том, что некоторые вычисления не учитываются, хотя они и допускаются алгоритмом. Среди игнорируемых вычислений могут быть такие, которые потребляют очень много времени. Предположения в Определении 2 не исключают ни одного вычисления; определение только устанавливает меру времени для каждого вычисления. Недостаток временной сложности в том, что результат определен вычислениями (как в доказательстве Теоремы 5), что хотя и возможно, но считается крайне маловероятным. Действительно, в этом вычислении одно сообщение «обгоняется» цепочкой из N – 1 последовательно передаваемого сообщения.

В качестве компромисса между двумя определениями можно рассмотреть a-временную сложность, которая определяется согласно предположению, что задержка каждого сообщения – величина между a и 1 (a – константа ≤1). К сожалению, этот компромисс обладает недостатками обоих определений: a-временная сложность эхо-алгоритма равна O(min(N,D/a)).

Наиболее точная оценка временной сложности получается, когда можно задать распределение вероятностей задержек сообщений, откуда может быть вычислено ожидаемое время вычисления алгоритма. У этого варианта есть два основных недостатка.

Во-первых, анализ алгоритма слишком зависит от системы, так как в каждой распределенной системе распределение задержек сообщений различно.

Во-вторых, в большинстве случаев анализ слишком сложен для выполнения.

Определение, основанное на цепочках сообщений. Затраты времени на распределенное вычисление могут быть определены с использованием структурных свойств вычисления, а не идеализированных временных предположений. Пусть C – вычисление.

Определение 4. ***Цепочка сообщений в C*** – это последовательность сообщений m_1, m_2, ..., m_k такая, что для любого i ($0 \le i \le k$) получение m_i каузально предшествует отправлению $m_i + 1$.

Цепочечная сложность распределенного алгоритма (chain-time complexity) – это длина самой длинной цепочки сообщений во всех вычислениях алгоритма.

Это определение, как и Определение 2, рассматривает всевозможные выполнения алгоритма для определения его временной сложности, но определяет другую меру сложности для вычислений.

Рассмотрим ситуацию (происходящую в вычислении, определенном в доказательстве Теоремы 5), когда одно сообщение «обгоняется» цепочкой из k сообщений. Временная сложность этого (под)вычисления равна 1, в то время, как цепочечная сложность того же самого (под)вычисления равна k. В системах, где гарантируется существование верхней границы задержек сообщений (как предполагается в определении временной сложности), временная сложность является правильным выбором. В системах, где большинство сообщений доставляется со «средней» задержкой, но небольшая часть сообщений может испытывать гораздо большую задержку, лучше выбрать цепочечную сложность.

Выводы

В ходе лекции рассмотрены следующие вопросы:

- классический алгоритм поиска в глубину;

- алгоритмы поиска в глубину за линейное время;

- алгоритм поиска в глубину со знанием соседей;

- сравнение алгоритмов поиска в глубину;

- альтернативные определения временной сложности.

Задание на самостоятельную работу

1. Конспект лекций.

Вид и тема следующего занятия

Практическое занятие №13. Работа с базой данных. CORS и кросс-доменные запросы