Нелинейные структуры данных.

Нелинейные структуры данных позволяют выражать более сложные отношения между элементами, нежели линейные отношения соседства, как это было в линейных списках. К нелинейным структурам данных относят графы, деревья и леса. Эти структуры находят широкое применение при решении практических задач.

Графы.

Граф G=(V,E) включает в себя непустое множество V, называемое множеством вершин (узлов) графа, множество E, представляющее собой множество рёбер графа, и отображение множества рёбер на множество пар элементов из множества вершин.

Множества V и E конечны. Каждому ребру графа eE можно поставить в соответствие пару вершин (Vi,Vj) этого графа, тогда говорят, что ребро e соединяет вершины Vi и Vj.

Любые две вершины Vi и Vj, соединённые в графе ребром e, называются смежными, и говорят, что ребро e инцидентно вершинам . Граф, у которого каждая вершина смежна со всеми остальными вершинами, называется полным графом. Вершина, у которой нет ни одной смежной вершины, называется изолированной. Граф, состоящий только из таких вершин, называется нульграфом.

Ребро e, направленное от одной вершины Vi к другой Vj, называется ориентированным ребром, или дугой. Граф, содержащий только ориентированные рёбра, называется ориентированным графом (орграфом). Ребро, не имеющее определённого направления, называется неориентированным ребром. Граф, содержащий только такие рёбра, называется неориентированным. Граф, содержащий как ориентированные, так и неориентированные рёбра, называется смешанным.

Ребро, соединяющее вершину саму с собой, называется петлёй. В дальнейшем будем рассматривать графы, в которых петли не допускаются.

В случае ориентированных рёбер между парой вершин возможны два ребра, которые имеют противоположную направленность и считаются разными рёбрами:

E₂

e₁

V2

V1

Если между любой парой вершин графа имеется не более одного ребра в неориентированном графе или не более одного ребра данного направления в ориентированном графе, то такой граф называется простым, в противном случае – мультиграфом. В дальнейшем будем рассматривать только простые графы.

Граф, у которого каждому ребру приписан вес, называется взвешенным графом.

Для ориентированного графа число рёбер, исходящих из некоторой начальной вершины, называется полустепенью исхода; число рёбер, для которых вершина является конечной, называется полустепенью захода данной вершины; а сумма полустепеней исхода и захода называется полной степенью этой вершины.

Любая последовательность рёбер простого орграфа такая, что конечная вершина любого ребра этой последовательности является начальной вершиной следующего за ним ребра, задаёт путь в графе. Длиной пути считается число рёбер в пути. Простой путь (относительно рёбер) – это путь в графе, все рёбра которого различны. Элементарный путь (простой относительно вершин) – это путь, в котором все вершины различны. Все элементарные пути – простые, но не наоборот.

Цикл (контур) – путь, начинающийся и заканчивающийся в одной и той же вершине. Простой цикл – если соответствующий ему путь простой. Элементарный цикл проходит через любую вершину (кроме одной – начальной) не более одного раза. В элементарном цикле начальная вершина появляется два раза, в простом – может более двух раз. Ациклический орграф – граф, не имеющий ни одного цикла. Связный граф – это граф, в котором есть пути между любыми двумя вершинами.

Представление графов

Реализация алгоритмов обработки графов в виде машинных процедур в высокой степени зависит от способа представления графов на логическом и физическом уровнях. А способ представления графа существенно зависит от типов структур данных, до­пускаемых используемым языком программирования и типом компьютера.

Выбор наилучшего способа представления графа зависит **от** природы моделируемых данных (процессов) и операций, выполняемых над ними. На выбор подходящего представления влияют такие факторы, как число вершин, максимальная полустепень исхода, частота изменения числа вершин и ребер, является ли граф ориентированным или нет и т.д. Таким образом, какого-то одного наилучшего способа представления не существует, лучшее представление зависит от решаемой задачи. Выбор конкретного представления графа может существенно повлиять на эффектив­ность его обработки.

Граф G=(**V,E)** может быть полностью определен простым перечислением двух множеств **V** и **Е.** Однако в большинстве случаев этот способ представления не позволяет создавать эффективные алгоритмы обработки графов.

Широко распространенным способом представления графов являются матричный способ и соответствующее отображение их в векторной (смежной) памяти в виде двухмерных массивов. Этот способ имеет несомненные достоинства:

а) массивы легко хранить и обрабатывать в ЭВМ;

б) для получения характеристик графа могут быть использо­ваны операции матричной алгебры;

в) имеются хорошо известные алгоритмы обработки графов.

Однако матричный способ представления имеет и крупный

недостаток, связанный с тем, что массивы являются статически­ми по размерам структурами.

1. Матрица смежности

Очень часто граф с количеством вершин n представляется в виде матрицы смежности , в которой =1, если (,) принадлежит Е, в противном случае.

Для простых неориентированных графов матрица смежности симметрична, т.е. . В случае взвешенного графа полагается , где - вес ребра.

Матрица смежности зависит от упорядочения графа (от по­рядка нумерации вершин). Для различных упорядочений получа­ются различные матрицы (изоморфизм), однако любая матрица смежности графа G может быть получена из другой матрицы смеж­ности этого же графа путем перестановки некоторых строк и со­ответствующих им столбцов.

Используя матрицу смежности, можно определить все пути между вершинами и и их длины, определить все возможные пути, циклы, простые и элементарные пути и т.п.

Рассмотрим представление орграфа и неориентированно­го графа , представленных на рис., в виде матриц смежности.

Если в графе ребер (дуг) много, то такое представление доста­точно компактное, в противном случае получается разреженная матрица.

При разработке программ, предназначенных для работы с графами, необходимо контролировать правильность задания параметров графа. Естественно, при числе вершин графа, мень­ших 2, не приходится говорить о матрице смежности. Если т — число вершин графа и n — число дуг, очевидными являются тре­бования: т2, n1 и nт\*(т - 1) для ориентированного или n т\*(т - 1)/2 для неориентированного графа. Здесь учитывает­ся тот факт, что в ориентированном графе вершины могут быть соединены прямыми и обратными дугами.

3

6

5

4

1

2

2

1

6

5

4

3

Рис. Ориентированный () и неориентированный () графы

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 0 | 1 | 1 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 1 | 0 |
| 0 | 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 1 | 0 | 0 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 0 | 1 | 0 | 1 | 0 | 0 |
| 1 | 0 | 1 | 1 | ,0 | 0 |
| 0 | 1 | 0 | 1 | 0 | 0 |
| 1 | 1 | 1 | 0 | 1 | 1 |
| 0 | 0 | 0 | 1 | 0 | 1 |
| 0 | 0 | 0 | 1 | 1 | 0 |

Матрицы смежности ориентированного () и неориентированного () графов

1. Векторы смежности

При представлении графа в виде векторов смежности создаётся матрица, число строк которой равно числу вершин графа. Каждая i-я строка матрицы содержит номера вершин, смежных с i-ой вершиной, петли исключаются. Число столбцов матрицы определяется максимальной полустепенью исхода в орграфе иле максимальной степенью в неориентированном графе. Рассмотренные выше графы ибудут представлены, как на рис.

Такие массивы занимают меньше памяти, время просмотра их сокращается. Однако и здесь не удается избежать разреженности матрицы, если полустепень исхода какой-либо отдельной вершины близка к числу вершин графа.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 2 | 4 | 0 | 0 | 0 |
| 1 | 3 | 4 | 0 | 0 |
| 2 | 4 | 0 | 0 | 0 |
| 1 | 2 | 3 | 5 | 6 |
| 4 | 6 | 0 | 0 | 0 |
| 4 | 5 | 0 | 0 | 0 |

|  |  |  |
| --- | --- | --- |
| 2 | 3 | 0 |
| 0 | 0 | 0 |
| 2 | 4 | 5 |
| 5 | 0 | 0 |
| 6 | 0 | 0 |
| 4 | 0 | 0 |

Рис.Представление векторов смежности массивами

1. Списки смежности

Список смежности для вершины v — это для орграфа список вершин, в которых оканчиваются дуги, исходящие из вершины v, а для неориентированного графа — список смежных с v вершин.

Для орграфа и графа списки смежности выглядят так:

: 1=>2->3 1=>2->4

1. => 2->4
2. => 1->3->4
3. => 2->4
4. => 1->2->3->5->6
5. =>4->6
6. =>4->5

1=> 2->3

2=> NULL

3=> 2->4->5

4=> 5

5=>6

6=>4

2=>NULL 2=>1->3->4

3=>2->4->5 3=>2->4

4=>5 4=>1->2->3->5->6

5=>6 5=>4->6

6=>4 6=>4->5

Здесь: => — указатель начала списка,

-> — связь вершин в списке смежности.

Списки смежности можно представить в виде связанных последовательных списков, причем для каждой вершины создается свой последовательный список с динамическим получением памяти для каждого элемента списка. Для указателей списков тогда создается массив указателей, число элементов которого равно числу вершин графа. Очевидно, что алгоритмы обработки графов при таком их представлении сильно усложняются.

1. Матрица инцидентности

Напомним, что любые две вершины и и *v*, соединенные в гра­фе ребром *a*, называются смежными**,** и говорят, что ребро а ин­цидентно вершинам и и v.

Любой граф можно задать матрицей инцидентности. Мат­рица инцидентности В**[]** размерностью n\*т **(**n— число вер­шин, т — число дуг) определяется следующим образом:

**,** если является начальной вершиной дуги , если яв­ляется конечной вершиной дуги в противном случае.

Если граф неориентированный, то его матрица инцидентности определяется аналогично, за исключением того, что элемент -1 поменяется на +1.

Для графов и матрицы инцидентности имеют вид:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |
| 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | -1 | 0 | -1 | 0 | 0 | 0 | 0 | 0 |
| 3 | 0 | -1 | 1 | 1 | 1 | 0 | 0 | 0 |
| 4 | 0 | 0 | 0 | -1 | 0 | 1 | 0 | -1 |
| 5 | 0 | 0 | 0 | 0 | -1 | -1 | 1 | 0 |
| 6 | 0 | 0 | 0 | 0 | 0 | 0 | -1 | 1 |

=

(В каждом столбце должно быть по одной 1 и -1).

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |
| 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| 3 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 |
| 4 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 0 |
| 5 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 |
| 6 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |

=

(В каждом столбце должно быть по две 1).

Пути в графе

Во многих задачах требуется определение путей в графах.

К ним относятся задачи определения наличия пути между двумя вершинами, всех путей между ними, экстремальных путей, путей от одной вершины до всех других, определения дуг, составляю­щих путь, и т.п.

Рассмотрим матрицу смежности. Единица в i-й строке и j-м столбце матрицы А, т.е. указывает на наличие ребра (,), т.е. на путь длиной 1 из . Элементами матрицы А2 будут:

Для каждого k условие выполняется тогда и только тогда, когда оба элемента равны 1, т.е. имеются ребра , и ,, следовательно, существует путь из в длиной 2 через .. Тогда приведенная выше сумма равна числу различных путей длиной 2 из в через различные вершины .

Аналогично элемент матрицы задает число путей дли­ной 3 из в, а по матрице Аr мы можем определить число путей длиной r из в.

Если мы хотим определить, существует ли в графе с n верши­нами путь из в, то нам необходимо проверить, имеются ли элементарные пути длиной, меньшей или равной (n - 1). Такие пути определяются с помощью матрицы = А + А2 + А3 +...+Аn-1.

Элемент показывает число существующих путей из вдлиной, меньшей или равной (n-1). Если элемент , то вершина достижима из .

Матрица = А + А2 + А3 +...+Аn дает возможность определить число элементарных путей и элементарных циклов в графе, представленном матрицей смежности А.

Путевая матрица (матрица достижимости)

Если нас интересует, достижима ли вершина из вершины , и не интересует число путей из вершины в , то достаточно в матрице все ненулевые элементы поменять на 1. Тогда полу­чим матрицу достижимости, или путевую матрицу , элемен­ты которой = 1, если существует путь из вв про­тивном случае. Матрицу Р называют также транзитивным замы­канием матрицы смежности.

Путевая матрица только показывает, имеется или отсутству­ет путь между парой вершин (или цикл в любой вершине), но не определяет все пути.

Рассмотренный способ определения путевой матрицы Рn из матрицы громоздок. Более рациональный способ получения путевой матрицы предложен Уоршаллом , в котором исполь­зуются реализуемые просто булевы операции. В этом алгоритме вначале путевая матрица Р устанавливается равной матрице смеж­ности, затем за k проходов, k=1,2,...,n, в циклах по i и j выполня­ются булевы операции:

Алгоритм выполняется за n проходов. После k-го прохода содержит 1 или 0 в зависимости от того, имеются ли пути между вершинами i и j, которые не проходят через вершины с номерами, большими или равными k, т.е. между вершинами *i* и j могут быть вершины только с номерами, менышими или равными k.

Минимальная путевая матрица

Если по путевой матрице мы можем определить, имеется ли путь между вершинами и, то минимальная путевая матрица содержит длины кратчайших путей (количество дуг) между вер­шинами и, i,j=1,2,…,n.

Алгоритм вычисления минимальной путевой матрицы полу­чается модификацией алгоритма Уоршалла. В матрице смежнос­ти все нулевые элементы меняются на бесконечность (число > n), а выражение для вычисления элемента минимальной путевой мат­рицы принимает вид:

при k=1,…n.

Алгоритм, как и при вычислении путевой матрицы, выполня­ется за n проходов. После k-го прохода содержит минимальную длину пути (в дугах) между вершинами i и j, который не проходит через вершины с номерами, большими k. После k-го прохода за минимальную длину пути между вершинами i и j принимается бо­лее короткая из двух:

1. длина пути, которая была вычислена до этого, т.е. пути, проходящего только через вершины с номерами, меньшими k;
2. длина пути, проходящего через вершину k, причем пути (i-k) и (k-j) также проходят только через вершины с номерами, меньшими k.

Иногда необходимо знать не только минимальный путь меж­ду вершинами i и j, но и вершины, через которые пролегает этот путь (маршрут). Для этого дополнительно вводится матрица , в которой элемент содержит номер вершины k, полученный при нахождении наименьшего значения при k-м прохождении.

Если , то значит, что вершины i и j смежны.

Алгоритмы поиска кратчайших путей в графе.

Рассмотрим связный граф G, в котором каждому ребру приписан положительный вес, равный длине ребра. Длина пути в таком графе равна сумме длин рёбер, составляющих путь.

Задачи о кратчайших путях относятся к фундаментальным задачам комбинаторной оптимизации, так как многие задачи можно свести к отысканию кратчайшего пути в графе. Существу­ют различные типы задач о кратчайшем пути между вершинами:

* двумя заданными вершинами;
* данной вершиной и всеми остальными вершинами;
* каждой парой вершин и т.д.

1. Алгоритм Дейкстры

Одним из наилучших алгоритмов отыскания кратчайшего пути между двумя вершинами является алгоритм Дейкстры [8], кото­рый по существу совпадает с алгоритмом Прима [8] отыскания остовного дерева минимального веса.

Рассмотрим алгоритм Дейкстры отыскания кратчайшего пути от заданной начальной вершины (источника) до конечной вершины w. Граф G=( V,E) имеет m-вершин, вершины пронумерованы 1,2,...,n. Граф представлен взвешенной матрицей смежности , где - дли­на дуги, соединяющей вершины i и j. Если дуга отсутствует, то принимается бесконечность (число, большее длины наиболь­шей дуги графа).

В результате вычислений получаем перечень дуг, лежащих на кратчайшем пути и представленных тройками чисел: начальная вершина, конечная вершина и длина пути от заданной вершины v до конечной вершины дуги.

Фактически этот перечень можно рассматривать как дерево с кор­нем v, узлы дерева содержат номера вершин графа, длины означают расстояния от корня (заданной вершины v) до этих вершин. Обрат­ный обход дерева от заданной конечной вершины w до начальной v дает перечень вершин, лежащих на искомом кратчайшем пути.

Алгоритм Дейкстры состоит в следующем:

Строится множество S вершин, для которых кратчайшие пути от источника v уже известны. На начальном шаге в S включается вершина v. На каждом шаге к множеству S добавляется та из оставшихся вершин, расстояние до которой от источника v меньше, чем для других оставшихся вершин. Если стоимости всех дуг неотрицательны, то кратчайший путь от источника v к конкретной вершине w проходит только через вершины множества S.

Такой путь назовём особым. На каждом шаге алгоритма используется также массив , в который записываются длины кратчайших особы путей для каждой вершины. Когда множество S будет содержать все вершины графа, т.е. для всех вершин будут найдены «особые» пути, тогда массив будет содержать длины кратчайших путей от источника v до каждой вершины i.

Приведём структуру программы, реализующей алгоритм Дейкстры. Пусть в орграфе G вершины наименованы целыми числами, т.е. V={1,2,…,n}, причём вершина 1 является источником. Найдём кратчайшие пути до всех остальных вершин.

Procedure Dijkstra;

Begin s:=[1];

For i:=2 to n do D[i]=A[1,i];

For i:=1 to n-1 do

begin

1. выбор из множества V\S такой вершины w, что значение D[w] минимально;
2. добавить w к множеству S;
3. for <цикл по каждой вершине V из множества V\S> do

D[V]:=min(D[V],D[w]+A[w,V]) или

If D[w]+A[w,V]< D[V] then D[V]:= D[w]+a[w,V]

end

end;

Рассмотрим работу алгоритма на примере:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **№ итерации** | **S** | **w** | **D[2]** | **D[3]** | **D[4]** | **D[5]** |
| 0 | {1} | - | 10 | ∞ | 30 | 100 |
| 1 | {1,2} | 2 | —//— | 60=min(∞,60) | 30=min(30,10+∞) | 100=min(100,∞+10) |
| 2 | {1,2,4} | 4 | —//— | 50=min(60,30+20) | —//— | 90=min(100,30+60) |
| 3 | {1,2,4,3} | 3 | —//— | —//— | —//— | 60=min(90,50+10) |
| 4 | {1,2,4,3,5} | 5 | 10 | 50 | 30 | 60 |
|  |  | Особый путь: | 1-2 | 1-4-3 | 1-4 | 1-4-3-5 |

Для определения последовательности вершин, составляющих кратчайший путь, в алгоритм надо ввести ещё один одномерный массив P вершин, где P[V] содержит вершину, непосредственно предшествующую вершине V в кратчайшем пути. Вначале положим P[V]=1 для всех V1.

1

2

5

3

4

10

100

20

60

10

30

50

В процедуру Dijkstra в тело внутреннего цикла надо ввести при выполнении условия:

If D[w]+A[w,V]<D[V] then P[V]:=w;

После выполнения алгоритма кратчайший путь к каждой вершине можно найти с помощью обратного прохождения по предшествующим вершинам массива P.

Алгоритм Дейкстры применим для любых графов: ориентированных, неориентированных, смешанных. Сложность алгоритма равна O() при использовании матрицы смежности. Если использовать списки смежности, то сложность алгоритма будет O(m n), где m- число дуг, n – число вершин. Это значительно меньше, чем O() при m .

2. Алгоритм Флойда

Иногда требуется решать общую задачу нахождения кратчайших путей, т.е. нахождения для каждой пары вершин (u,v) пути от вершины v к вершине w, длина (расстояние) которого мини­мальна среди всех возможных путей от v к w. Эту задачу можно решить, последовательно применяя алгоритм Дейкстры нахождения кратчайших путей для каждой вершины. Тогда временная сложность будет равна 0(n3) при использовании матрицы смеж­ности 0(n\*e\* (n)) при использовании списков смежности.

Существует прямой способ решения этой задачи, использу­ющий алгоритм Флойда. Граф G=(V,E) с n вершинами пред­ставляется взвешенной матрицей смежности An\*n. В результате решения задачи формируется матрица Bn\*n, в которой элемент будет содержать значение кратчайшего пути от вершины i до вершины j.

Сначала исходная матрица A копируется в матрицу B, при­чем если дуга (i,j) отсутствует, то =∞, а диагональные эле­менты = 0.

Алгоритм выполняется за n проходов. После k-го прохода содержит значение наименьшей длины путей из вершины i к j, которые не проходят через вершины с номером, большим k, т.е. между вершинами i и j могут быть вершины только с номерами, меньшими или равными k.

При k-м проходе элемент вычисля­ется как [i,j] = min([i,j], [i,k] +[k,j]).

Алгоритм Флойда реализуется посредством трех вложенных циклов и имеет сложность 0(n3).

Программа нахождения кратчайших путей по алгоритму Флойда :

Type MATRIX=array[1..n,1..n] of real; {n задаётся как константа}

Type MAS=array[1..n,1..n] of byte;

Procedure Floyd (var B: MATRIX; A:MATRIX; n:byte; var P:MAS);

Var i,j,k: integer;

Begin

For i:=1 to n do

For j:=1 to n do begin B[i,j]:=A[i,j]; P[i,j]:=0; End;

For i:=1 to n do B[i,j]:=0;

For k:=1 to n do

For i:=1 to n do

For j:=1 to n do

If B[i,k]+ B[k,j]< B[i,j] then

Begin B[i,j]:= B[i,k]+ B[k,j]; P[i,j]:=k;

End;

End;

Для определения самого кратчайшего пути следует ввести матрицу **,** элемент которой P[i,j] содержит вершину k, полученную при нахождении наименьшего значения B[i,j]. Если P[i,j]=0, то кратчайший путь из i в j состоит их одной дуги i→j.

**3. Динамическое программирование**

Динамическое программирование(ДП)- это вычислительный метод, использующий аппарат рекуррентных соотношений. Разработан Р.Беллманом(США).

В ДП рассматриваются многоэтапные процессы принятия решений. При постановке и решении любой задачи в ДП формируется целевая функция и критерий, подлежащий оптимизации, и на каждом шаге принимается такое решение, чтобы досталась поставленная цель.

Метод ДП основан на **принципе оптимальности**:

Оптимальная стратегия обладает таким свойством, что каково бы ни было начальное состояние и начальные решения, последующие решения должны приниматься, исходя из оптимальной стратегии с учетом состояния, достигнутого путем предыдущих решений. Другими словами, планируя многошаговую операцию, необходимо выбирать решение на каждом шаге с учетом его будущих последствий на еще предстоящих шагах.

Основное преимущество метода ДП. Рассмотрим процесс с последовательным принятием решений, состоящий из n этапов, на каждом из которых принимается одно из k возможных решений. Тогда для определения оптимального решения методом полного перебора необходимо перебрать kn что крайне велико ( при n=10 и k=3 kn≈ 59000). Метод ДП исключает необходимость полного перебора всех kn возможных решений.

Рассмотрим применение ДП на примере задачи определения кратчайшего пути в графе.

7 Шаги процесса принятия решений определим

1 3 как последовательный выбор ребер графа,

6 5 приводящих в каждую вершину.

2 Выведем функциональное уравнение

Метода ДП. Пусть кратчайшее расстояние от вершины j

8 4 3 до конечной n=6 равно Wj, j=n-1,n-2,…,1. Рассмотрим

6 7 2 любую вершину k, k=1,2,…,n-1. Кратчайшее расстояние от

4 от k-й вершины до последней (n-й) будет равно

6 2 Wk=minj(Wj+wkj), где wkj- длина ребра k→j.

(минимум берется по всем вершинам j, в которые можно

1. попасть из k-й).

Это и есть функциональное уравнение (k=n,n-1,…,1)

W6=0 → для удобства сразу обозначаем на графе около соответствующей вершины.

W5=0+2=2

W4= min 4+2 =6 Кратчайший путь имеет длину 7 и соответствует дугам 1→2→5→6.

6

W3=min 3+7 =5

7+6

2 5

4+2

W2=min 8+6 =6

2+5

1 2

W1=min 1+6 =7

3+5

Остовные деревья графа

Остовным деревом для связного неориентированного графа G=(V,E) с n вершинами называется неориентированное дерево, содержащее все n вершин и (n - 1) ребер графа. Таким образом, остовное дерево связывает все вершины графа и из каждой вер­шины графа можно попасть в любую другую. В полном графе с n вершинами имеется n n-2 остовных деревьев. Для одного и того же графа можно построить несколько остовных деревьев.

2

3

1

4

5

2

3

1

4

5

2

3

1

4

5

а б в

**Рис.** Граф **(а)** и его остовные деревья (**б)** и (в)

Пусть G=(V,E) — связный неориентированный граф и T = (V,F) — остовное дерево для него. Тогда:

а) для любых двух вершин vi и vj путь между ними единствен;

б) если к остовному дереву Т добавить ребро из (E-F), т.е. из оставшихся, не включенных в дерево ребер графа, то возникнет ровно один цикл и Т перестанет быть деревом.

Под остовным деревом ориентированного графа понимают такое дерево, в котором одна из вершин графа связана со всеми остальными.

Основными задачами нахождения остовных деревьев графа являются:

* нахождение остовного дерева;
* нахождение минимального и максимального остовного дерева.

Обходы графов.

Поиск в глубину и поиск в ширину

При решении многих задач, связанных с графами, необходим систематический обход вершин и ребер графа. Широкое применение получили два метода обхода:

* поиск в глубину;
* поиск в ширину.

При поиске в глубину осуществляется регулярный обход графа по следующим правилам.

1. Находясь в вершине v, нужно двигаться в любую другую w, ранее не пройденную вершину (если таковая найдется), одновре­менно запоминая дугу z, по которой мы впервые попали в вер­шину v.

1. Если из вершины v мы не можем попасть в ранее не прой­денную вершину или таковой вообще нет, то мы возвращаемся в вершину дуги z, из которой впервые попали в вершину v, и про­должим поиск в глубину из этой вершины.

При выполнении обхода по этим правилам мы стремимся про­никнуть вглубь графа как можно дальше, затем отступаем на шаг и снова стремимся пройти вперед и т.д.

При поиске в глубину в ориентированном графе мы можем попасть в вершину w из вершины v только в том случае, есливорграфе есть дуга (v,w), то есть двигаемся вперед только в направ­лении ориентации дуг, а возвращаемся против ориентации. В неориентированном графе таких ограничений нет. В неориентирован­ном графе обход можно начинать с любой вершины. Топология полученного остовного дерева зависит от начальной вершины.

Рассмотрим особенности поиска в глубину в орграфе. Те дуги, которые ведут к новым вершинам, называются дугами дерева, или древесными дугами. Они формируют для данного графа либо *ос*товное дерево (дерево поиска в глубину), либо остовный лес (глу- бинное остовное дерево, глубинный остовный лес). Для того чтобы после завершения обхода в глубину все вершины оказались прой­денными и образовалось остовное дерево, (а не лес), необходимо и доста­точно, чтобы обход начинался во входе графа (вершине, имею­щей только исходящие дуги) и этот вход был единственным. В противном случае при обходе в глубину образуется глубинный остовный лес. Если при поиске в глубину вершины графа нумеру­ются в порядке их посещения, то такая нумерация называется глубинной нумерацией графа или М-нумерацией вершин графа [14].

После завершения поиска в глубину все дуги орграфа разби­ваются на 4 класса:

1. класс Т (Tree) древесных дуг, порождающих дерево поиска в глубину либо глубинный остовный лес. Для каждой дуги (x,y) Т имеем М(х) < М(у), М[i] — номер вершины i;
2. класс F (Forward) прямых дуг, к которому относятся дуги (x,y) такие, что М(х) < М(у) и вершина х соединена с y альтернативным путём, состоящим из древесных дуг. Другими словами, прямые дуги идут от предков к непосредственным потомкам, но не являются дре­весными дугами, а дублируют путь из древесных дуг;
3. класс В (Back) обратных дуг, представляющих дуги (х,у), такие, что М(х) > М(у) и вершина у соединена с х путем, состоя­щим из древесных дуг. По-другому обратные дуги идут от по­томков к предкам;
4. класс С (Cross) поперечных дуг, представляющих собой дуги (х,у), такие, что М(х) > М(у) и вершины х и у не соединены путем, состоящим из древесных дуг, т.е. поперечные дуги соединяют вер­шины, не являющиеся ни потомками, ни предками друг друга.

Рассмотрим пример:

Древесные дуги: (а, Ь), (Ь, е), (е, *h), (h*, *j*), (е, d), (а, с), (с,f), (f, *i), (i,g).*

Прямые дуги: (*b*, d), (с, g).

Обратные дуги: (g, *i*), (d, b).

Поперечные дуги: (f, h), (*i*, h):

Если начать с “C”

a

b

c

e

g

i

j

h

f

d

1

2

6

3

8

7

10

9

4

5

j

h

8

5

a

b

e

d

c

f

g

i

Рис. Обход дерева в глубину

j

h

7

a

b

e

c

f

g

i

d

6

2

3

1

4

8

5

9

10

Глубинное остовное дерево

Если начать обход в глубину этого же графа, начиная с другой вершины, то образуется глубинный остовный лес. Например, обход, начиная с вершины с, приведет к образованию леса с двумя остовными деревьями. Первое дерево включает ребра (с, f), *(*f, h), (h, j), (f, i), (i,g) .

Второе дерево с началом обхода в вершине а состоит из ребер (a, b), (Ь, e), (e, d). Для упражнения определите прямые, обратные и поперечные дуги графа, полученные в результате такого обхода.

При поиске в глубину в связном неориентированном графе образуется одно единственное глубинное дерево. В нем различа­ют только два класса ребер: древесные *и* обратные**.**

Сложность алгоритма поиска в глубину равна 0(m), где m — число дуг (ребер) графа.

При поиске в ширину последовательно просматриваются, начиная с заданной вершины, все вершины дерева на уровне k=1, 2,… .

На рисунке представлены граф и остовные деревья полученные поиском в глубину и ширину. Начало поиска идёт с вершины 3.

2

3

1

4

5

а б в

Граф и его остовные деревья:

а) исходный граф; б) поиск в глубину; в) поиск в ширину

2

3

1

4

5

2

3

1

4

5

Если начать поиск с вершины 1, то получим:

2

3

1

4

5

При поиске в глубину; При поиске в ширину

2

3

1

4

5

Остовное дерево наименьшей стоимости (минимального веса)

Пусть G = (*V*, Е) — связный взвешенный неориентирован­ный граф, для которого задана матрица смежности, отобража­ющая веса ребер в числа (вещественные или целые). Стоимость (вес) остовного дерева определяется как сумма стоимостей (весов) его рёбер. Цель — найти для графа G остовное дерево наимень­шей стоимости (минимального веса).

Применение остовных деревьев минимальной стоимости можно найти при разработке коммуникационных сетей. Здесь вершины графа представляют города, рёбра – возможные коммуникационные линии между городами, а стоимость рёбер соответствует стоимости коммуникационных линий. В этом случае остовное дерево минимальной стоимости представляет коммуникационную сеть, объединяющую все города коммуникационными линиями минимальной стоимости.

Полный граф с n верши­нами содержит *n*n-2 остовных деревьев. Поиск каждого остовно­го дерева занимает 0(m) времени. В полном дереве m=n\*(n-1)/2 (количество рёбер).

Тогда решение задачи прямым поиском имело бы сложность 0(nn-2 n(n - 1)/2)=0(nn).

Построение остовного дерева минимальной стоимости.

1. Алгоритм Прима

Наиболее простым алгоритмом поиска остовного дерева ми­нимального веса является алгоритм Прима, похо­жий на алгоритм Дейкстры поиска кратчайшего пути.

Алгоритм Прима заключается в следующем.

1. Вначале выбираем некоторую вершину V и включаем её в множество выбранных величин, остальные (n-1) вершины графа отмечаются как невыбранные.
2. Определяются веса между всеми выбранными вершинами из и остальными невыбранными вершинами.
3. Выбираем вершину w с наименьшим весом до нее, фиксируем выбранные ребро и его вес.
4. Выбранную вершину w исключаем из множества невыбранных, включаем её в множество , число невыбранных вершин уменьшаем на 1.
5. Пункты 2— 4 повторяем до тех пор, пока не будут выбра­ны все вершины, т.е. (n- 1) раз.

Процесс поиска остовного дерева наименьшего веса по алго­ритму Прима можно проследить на примере графа, приведенно­го на рисунке.

3

5

1

2

4

6

7

8

9

21

31

10

27

15

32

7

8

9

11

15

17

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Номер итерации | w | Множество U выбранных вершин | Невыбранные вершины V\U и веса дуг до них | Выбранная дуга и её вес |
| 1 | – | 1 | 2 3 4 5 6 7 8 9  7 10 - - - - - - - | 1→2:7 |
| 2 | 2 | 1,2 | 3 4 5 6 7 8 9  10 9 27 - - - - - - | 2→4:9 |
| 3 | 3 | 1,2,4 | 3 5 6 7 8 9  10 (1→3) 27 11 - -  8 (4→3) | 4→3:8 |
| 4 | 4 | 1,2,4,3 | 5 6 7 8 9  27 11 - - 31 | 4→6:11 |
| 5 | 5 | 1,2,4,3,6 | 5 7 8 9  27 15 17 31 | 6→7:15 |
| 6 | 6 | 1,2,4,3,6,7 | 5 8 9  27 17 31  15(7→5) 21(7→8) | 7→5:15 |
| 7 | 7 | 1,2,4,3,6,7,5 | 8 9  (6→8)17 31  (7→8)21 | 6→8:17 |
| 8 | 8 | 1,2,4,3,6,7,5,8 | 9  31(3→9)  32(8→9) | 3→9:31 |
| 9 | 9 | 1,2,4,3,6,7,5,8,9 | **–** | **–** |

Временная сложность алгоритма Прима равна O(). При большом значении n его использовании нерационально.

2. Алгоритм Крускала

Пусть имеется связный взвешенный граф G(V,E) с n вершинами. Построение остовного дерева минимального веса начинается с графа T(V,Ф), имеющего только данные n вершин без рёбер, т.е. состоящего из n связных компонент.

Построение дерева сводится к формированию набора связных компонент, постепенным объединением которых и формируется остовное дерево.

Алгоритм Крускала заключается в следующем.

1. Примем номер итерации i=1, упорядочим рёбра множества E по возрастанию их веса.
2. Выберем из множества E ребро с наименьшим весом.
3. Проверим выполнения условия: если ребросвязывает две вершины из разных компонент, то оно добавляется в граф T, в противном случае это ребро отбрасывается (т.к. его добавление в графу приведёт к образованию цикла).
4. Ребро исключается из множества E;

Осуществляется переход к новой итерации i=i+1, и шаги 2-4 повторяются до тех пор, пока все вершины не будут принадлежать одной компоненте.

Продемонстрируем работу алгоритма Крускала на примере графа, рассмотренного в предыдущем разделе.

|  |  |  |  |
| --- | --- | --- | --- |
| Вес | Вершины | | Признак выборки |
| 7 | 1 | 2 | + |
| 8 | 3 | 4 | + |
| 9 | 2 | 4 | + |
| 10 | 1 | 3 | - |
| 11 | 4 | 6 | + |
| 15 | 5 | 7 | + |
| 15 | 6 | 7 | + |
| 17 | 6 | 8 | + |
| 21 | 7 | 8 | - |
| 27 | 2 | 5 | - |
| 31 | 3 | 9 | + |
| 32 | 8 | 9 | Конец выборки |

Упорядоченная таблица весов рёбер графа

3

5

1

2

4

6

7

8

9

21

31

10

27

15

32

7

8

9

11

15

17

Временная сложность алгоритма Крускала O(), где m – количество рёбер в графе. При m алгоритм Крускала предпочтительнее алгоритма Прима, но если m близко к , то рекомендуется применять алгоритм Прима.

Упорядочение ориентированного графа (топологическая сортировка)

Часто встречаются ситуации, когда необходимо решать задачу сортировки элементов множества, для которых определен частичный порядок, т.е. упорядочение задано не на всех, а только на некоторых парах элементов. Вот два примера:

1. В учебных программах вузов одни предметы опираются на материал других. Топологическая сортировка означает чтение курсов в таком порядке, чтобы ни один курс не читался раньше того курса, на материале которого он основан.
2. Сетевые методы планирования и управления предполагают, что любая работа может быть выполнена только после того, как будут выполнены опорные работы.

Пусть имеется ориентированный граф *G=(V,E)* с произвольно занумерованными вершинами *vi­, i­­= 1,2,…,n*. Граф является упорядоченным, если в отношении каждой его дуги *(vi­, vj)* справедливо неравенство *vi­<vj* . Для этого необходимо выполнить перенумерацию вершин в соответствии с их рангами. Будем считать, что единственная вершина *vi­* , не имеющая входов, имеет нулевой ранг. Для любой другой вершины *vj* ее ранг равен максимальной длине пути (максимальному числу дуг) от вершины нулевого ранга *vi­* до вершины *vj* .

Рассмотрим граф, представленный на рисунке. Из рисунка видно, что вершина 3 имеет ранг R=0, вершина 1 имеет ранг R=1, вершины 5 и 7 имеют ранг R=2, вершина 4 имеет ранг R=3, вершина 2 имеет ранг R=4 и вершина 6 имеет ранг R=6.

0

1

2

5

4

2

3

а)Неупорядоченный граф б) Упорядоченный граф

Алгоритм упорядочения графа состоит из двух этапов. На первом этапе определяются ранги вершин, на втором этапе осуществляется перенумерация их в соответствии с рангами. Единственной вершине *vj* , не имеющей входов, присваивается номер 0, а затем нумерация вершин одного ранга безразлична. Упорядоченный граф имеет вершины *vj* с номерами 0,1,2,…,n – 1.

Рассмотренный выше граф после упорядочения будет иметь нумерация вершин, как показано на рисунке.

Таким образом, исходным номерами вершин 1,2,3,4,5,6,7 после упорядочения соответствуют новые номера 1,5,0,4,2,6,3.