**Лекция № 6.**

Методы поиска решений в пространстве состояний.

**Путь решения задачи**

Выполнение определённых действий и команд в системе или на объекте, находящихся в начальном или промежуточном состоянии меняет эти состояния, порождая новые, которых вообще-то может быть много и которые образуют то, что называется *пространством состояний.*

Среди таких состояний, если система или объект исправны, и работают без ошибок, как правило, имеется состояние, которое будет соответствовать желаемому или целевому, и процесс поиска решения при попадании в это состояние будет закончен.

Процесс поиска такого состояния может быть сложным, и поэтому возникает потребность в некотором едином методе представления множества состояний и поиска решений. Таким методом является метод, использующий представление состояний, в которых может находиться система или объект, в виде графа.

Если отождествить состояние , в котором находилась система или объект перед выполнением действий или команд, с корнем или начальной вершиной такого графа, то, применяя к состоянию какой-либо оператор , мы порождаем новое состояние , образуя тем самым следующую вершину графа (рис. 2.1).

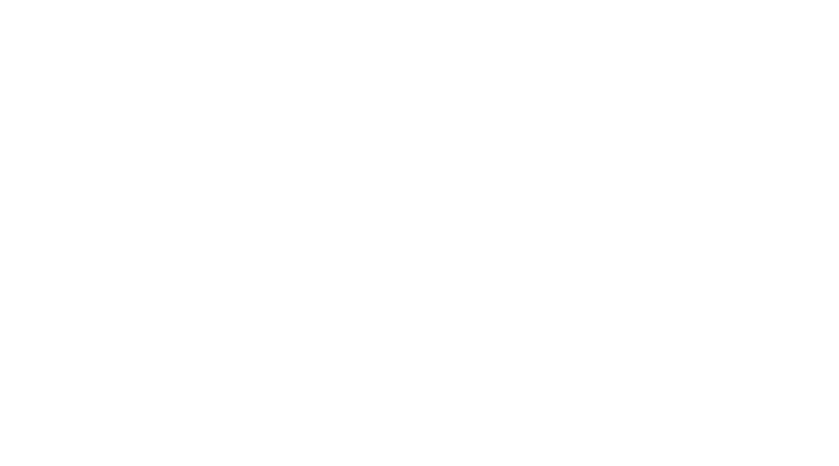






















































Рис. 6.1. Пример фрагмента графа, представляющего решение задачи.

Эта новая вершина может быть промежуточной или целевой. Если вершина промежуточная, то процесс порождения новых вершин (с помощью операторов ) будет продолжен, пока не найдется целевая. Процесс применения оператора к некоторой вершине называется *раскрытием вершины*. От каждой порожденной вершины к породившей ее расставляются указатели. Такие указатели позволяют найти путь к начальной вершине, после того, как будет обнаружена целевая.

Общая процедура построения дерева в пространстве состояний при этом выглядит следующим образом.

1). К корню дерева () применяются операторы из множества *G* (их может быть несколько). Полученные при этом вершины образуют первый уровень новых вершин.

2). Каждая из вновь полученных вершин проверяется, не является ли она целевой. Если нет, то процесс продолжается по отношению к каждой из них. Образуется второй уровень вершин. Если к какой-либо вершине никакой оператор из *G* не применим, то эта вершина становится терминальной (конечной). Как видим, на каждом шаге проводятся две операции: порождение новой вершины и проверка, не является ли новая вершина целевой, т.е. совпадающей с целевым состоянием.

3). Когда целевая вершина будет найдена, в обратном направлении (от цели к началу) просматриваются указатели дуг и выделяется путь решения. Практически этот путь удобнее отображать посредством операторов, связанных с этими дугами (см. рис. 2.1).

В общем случае число вершин может быть большим. Их последовательное раскрытие, анализ и пометка пути осложняют задачу. Возникает проблема перебора вершин, связанные с выбором порядка их порождения и анализа. Здесь возможны следующие варианты:

* если вершины раскрываются в том же порядке, в котором они порождаются, то такой процесс называется полным перебором в ширину *(breadth- first process*);
* если на каждом шаге первой раскрывается вершина, которая была построена последней, то такой процесс называется полным перебором в глубину *(depth-firstprocess).* В этих процессах расположение целевой вершины не влияет на порядок раскрытия. Поэтому эти процессы часто называют процессами слепого перебора;
* если есть некоторая дополнительная (эвристическая) информация о предметной области, которая позволяет делать суждения о характере графа пространства состояний и расположения цели, то такой метод построения графа называется эвристическим («эвристический» означает «служащий открытию»). Эвристическая информация, опирающаяся, как правило, на предыдущий опыт, позволяет выполнять поиск в наиболее перспективных направлениях.

Говоря о графе, будем рассматривать только один наиболее простой его тип — граф типа «дерево». Как известно, *деревом* называется граф, каждая вершина которого имеет только одну вершину, непосредственно предшествующую ей (родительскую), за исключением вершины-корня, которая предшествующих вершин не имеет.

**Метод полного перебора в ширин*у***

Как уже было сказано, в этом методе вершины раскрываются в том порядке, в котором они строятся. Основной алгоритм состоит в выполнении следующих действий.

1). Раскрывается начальная вершина . Она раскрывается до тех пор, пока ее можно раскрыть, применяя один и тот же оператор (или разные, смотря по условию). При этом образуются вершины первого уровня: , , ... . Их раскрывают в свою очередь, и образуются вершины второго уровня и т.д. (рисунок 2.1 может служить примером этого метода: и - вершины первого уровня, и - вершины второго уровня, и - третьего и т.д.).

2). Расставляются указатели, ведущие от новых вершин к корню. Это могут быть условные имена, буквы, цифры, имена операторов и т.п. Однако среди них могут быть и реальные величины, например, расстояния, стоимость, вес и т.д.

3). Проверяется, нет ли среди полученных вершин целевой . Если есть, то формируется решение на основе соответствующего оператора. Если целевых вершин нет, то рассматривается первая порожденная вершина и к ней применяется тот же алгоритм. После чего, переходят ко второй и т.д., пока среди получаемых вершин не окажется целевой.

Метод полного перебора в ширину гарантируют нахождение целевой вершины как раз потому, что перебор - полный. Путей достижения цели, вообще говоря, может быть много. В этом случае у нас имеется возможность выбрать наикратчайший (или самый дешевый, или самый легкий - критериев много) путь. Но может быть случай, когда граф поиска окажется бесконечным и тогда этот алгоритм никогда не кончит работу.

Таким образом, метод полного перебора гарантирует поиск оптимального решения, если дерево пространства состояний не бесконечно.

Синонимами названия метода являются: метод грубой силы, метод проб и ошибок. На рис. 6.2 показан фрагмент дерева полного перебора в ширину.

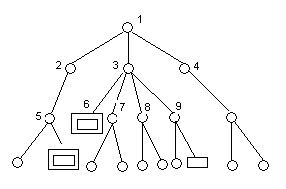


Рис. 6.2. Пример графа поиска, построенного при поиске в ширину.

**Метод полного перебора в глубину**

В методе перебора в глубину раскрываются, прежде всего вершины, которые были построены последними.

Первой раскрываемой вершиной, а следовательно, и последней, является корневая, но процесс всегда будет идти по самой левой ветви вершин. Чтобы как-то ограничить перебор, вводится понятие глубины вершины в дереве перебора. Полагаем, что глубина корня дерева равна нулю, а глубина любой последующей вершины равна единице плюс глубина вершины, непосредственно ей предшествующей.

Отсюда следует, что наибольшую глубину всегда будет иметь та вершина, которая должна быть в этот момент раскрыта. Если образующийся путь оказывается бесполезным, то есть при заданной глубине раскрытия целевой вершины не получилось, необходимо вернуться в вершину, предшествующую раскрытой и попытаться еще раз применить к ней операцию раскрытия. И так до тех пор, пока не будет получена целевая вершина. На рис. 6.3 показан элемент дерева полного перебора в глубину.

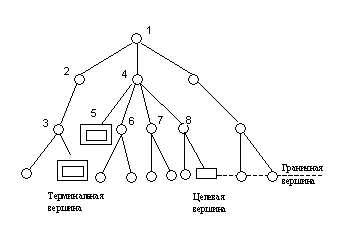


Рис. 6.3. Пример графа поиска, построенного при поиске в глубину.

Алгоритм перебора в глубину состоит в следующем.

1). Раскрывается начальная вершина соответствующая начальному состоянию *.*

2) Раскрывается первая вершина, получаемая в результате раскрытия . Ставится указатель.

3) Если она раскрывается, то следующей будет раскрываться вновь порожденная вершина. Если вершина не раскрывается, то процесс возвращается в предыдущую вершину .

4) По получении целевой вершины, процесс раскрытия заканчивается и по указателям строится путь, ведущий к корню.

5). Если для заданной глубины раскрытия целевая вершина не находится, то весь процесс повторяется снова, а в качестве новой вершины рассматривается самая левая из полученных на предыдущем этапе.

Соответствующие дугам операторы образуют решение задачи.

Так же, как и метод поиска “в ширину”, этот метод относится к методам “грубой силы”. Он обеспечивает перебор всех состояний, если, конечно, прежде не определит целевую вершину .

**Эвристические методы поиска в пространстве состояний**

Методы полного перебора гарантируют решение задачи, если оно существует, а при наличии нескольких решений, гарантирует оптимальное. Однако на практике эти методы используются для решения лишь небольших по размерности графа состояний задач. Для реальных случаев чаще всего используется дополнительная информация, основанная на предыдущем опыте или полученная на основании теоретических выводов.

Такая информация называется *эвристической*, а организованная в правила - *эвристическими правилами* или *эвристиками*. Эвристическая информация носит сугубо специальный характер и может применяться только в рамках данной задачи, в лучшем случае, в рамках задач данного класса.

Чаще всего практические задачи решаются с помощью эвристического поиска. Эвристический поиск основан на функции оценки состояния. В отличие от алгоритмов полного перебора, эвристический поиск позволяет ранжировать пространственные состояния на основе их «перспективности». Эвристический поиск ищет в пространстве состояний более целенаправленно, чем алгоритмы полного перебора. Важно, что во многих задачах оценка состояния есть наилучшая оценка пути достижения данного состояния из начального. Если в нашей задаче возможна такая оценка, то алгоритмы эвристического поиска значительно упрощаются.

Основной класс алгоритмов эвристического поиска – это поиск от наилучшего состояния. Он включает три основных алгоритма: это жадный поиск, лучевой поиск и А\*. Общая их идея основана на поддержании в процессе поиска множества достигнутых состояний и выборе на каждом шаге одного или нескольких лучших состояний.

Простейший из них — **жадный поиск**. Если его применить в задаче поиска наилучшего пути в графе, это даст известный [алгоритм Дейкстры](http://ru.wikipedia.org/wiki/%C0%EB%E3%EE%F0%E8%F2%EC_%C4%E5%E9%EA%F1%F2%F0%FB). Жадный поиск, выбирая состояние, из которого будет продолжаться поиск, ищет состояние с наилучшей оценкой пути от начального в данное. Поэтому он и называется «жадным» — поскольку «хватает» самое лучшее на данный момент состояние, не думая о последствиях. Естественно, как и многие жадные алгоритмы, такая стратегия не приводит к оптимальному решению. Конечные состояния зачастую достигаются слишком длинным, неоптимальным путем. Кроме того, жадный поиск постоянно должен поддерживать множества всех достигнутых состояний, которых может быть слишком много, отчего чрезмерно расходуется память.

Примером, где могут успешно использоваться идеи жадных алгоритмов, является задача составления числа из разрядов. Простейшая постановка такой задачи имеет следующий вид.

Пусть заданы цифры 1, 3, 9, 3, 4, 2, из которых надо составить наибольшее число. Каждая цифра может быть использована только один раз.

Используя идеи и принципы жадных алгоритмов, построим для данной задачи жадный алгоритм следующего вида по шагам.

Шаг 1. Выбираем наибольшее из имеющихся цифр и ставим, его справа к ранее установленным цифрам или первым, если таковых ещё нет. Следует переход к Шагу 2.

Шаг 2. Производится проверка. Если в заданном множестве после вычёркивания использованной цифры остались другие, то следует переход к Шагу 1. В противном случае к Шагу 3.

Шаг 3. Формирование числа закончено.

Решение. 9 → 4 → 3 → 3 → 2 → 1 = 943321.

К сожалению, жадный алгоритм является эвристическим и не всегда позволяет получать оптимальное решение задач.

**Алгоритм лучевого поиска и алгоритм А\***— это попытки улучшить поведение жадного поиска и исправить эти две присущие ему проблемы. Лучевой поиск работает следующим образом: на каждом шаге мы поддерживаем некоторое множество из N лучших состояний. Далее из каждого из этих состояний делаем все возможные шаги и получаем множество состояний следующего поколения. В этом множестве мы удаляем дубликаты, то есть одинаковые состояния. Оцениваем оставшиеся и сортируем в порядке ухудшения оценки. Далее выбираем N лучших, и так до тех пор, пока мы не найдем интересующее нас конечное состояние.

У лучевого поиска есть свои достоинства и недостатки. Основной его минус в том, что, в отличие от жадного, лучевой поиск не гарантирует нахождения конечного состояния с наилучшим качеством, потому что в процессе движения фронта лучшее состояние может из него выпасть. На практике с этим можно бороться, настраивая ширину фронта. Чем фронт уже, тем быстрее работает алгоритм, но тем чаще он ошибается. Чем шире фронт, тем алгоритм работает лучше, но дольше. Это одно из его важных преимуществ – возможность легко балансировать между скоростью и качеством. Лучевой поиск очень популярен в академической среде, особенно среди китайских ученых. Он прост в реализации и работает достаточно неплохо.

Идея алгоритма А\* заключается в том, что мы выбираем на каждом шаге не лучшее, а наиболее перспективное на данный момент состояние. То состояние, через которое с наибольшей вероятностью проходит путь до лучшего конечного состояния.

К оценке текущего состояния в алгоритме А\* добавляется эвристическая оценка снизу остатка пути до конечного состояния. Если эвристическая оценка снизу достаточно хороша, то А\* работает эффективно и быстро находит наилучшее состояние. В этом случае А\* будет полиноминальным по числу шагов, а не экспоненциальным. А\* — это действительно хороший алгоритм, в своей работе я использую его гораздо чаще, чем лучевой поиск.