- 1) найдена вершина, содержащая ключ, равный ключу X;
- 2) в дереве отсутствует вершина, к которой нужно перейти для выполнения очередного шага поиска.

В первом случае возвращается указатель на найденную вершину. Во втором - указатель на звено, где остановился поиск, (что удобно для построения дерева).

ДОБАВЛЕНИЕ НОВОГО УЗЛА

Для включения записи в дерево прежде всего нужно найти в дереве ту вершину, к которой можно "подвести" (присоединить) новую вершину, соответствующую включаемой записи. При этом упорядоченность ключей должна сохраняться.

Алгоритм поиска нужной вершины, вообще говоря, тот же самый, что и при поиске вершины с

заданным ключом. Эта вершина будет найдена в тот момент, когда в качестве очередного указателя, определяющего ветвь дерева, в которой надо продолжить поиск, окажется указатель NIL (случай 2 в поиске записи в дереве).

ОБХОД ДЕРЕВА.

Во многих задачах, связанных с деревьями, требуется осуществить систематический просмотр всех его узлов в определенном порядке. Такой просмотр называется прохождением или обходом дерева.

Бинарное дерево можно обходить тремя основными способами: нисходящим, смешанным и восходящим (возможны также обратный нисходящий, обратный смешанный и обратный восходящий обходы). Принятые выше названия методов обхода связаны с временем обработки корневой вершины: До того как обработаны оба ее поддерева, после того как обработано левое поддерево, но до того как обработано правое , после того как обработаны оба поддерева. Используемые в переводе названия методов отражают направление обхода в дереве: от корневой вершины вниз к листьям - нисходящий обход; от листьев вверх к корню - восходящий обход, и смешанный обход - от самого левого листа дерева через корень к самому правому листу.

Схематично алгоритм обхода двоичного дерева в соответствии с нисходящим способом может выглядеть следующим образом:

- 1. В качестве очередной вершины взять корень дерева. Перейти к пункту 2.
- 2. Произвести обработку очередной вершины в соответствии с требованиями задачи. Перейти к пункту 3.
- 3.а) Если очередная вершина имеет обе ветви, то в качестве новой вершины выбрать ту вершину, на которую ссылается левая ветвь, а вершину, на которую ссылается правая ветвь, занести в стек; перейти к пункту 2;
- 3.б) если очередная вершина является конечной, то выбрать в качестве новой очередной вершины вершину из стека, если он не пуст, и перейти к пункту 2; если же стек пуст, то это означает, что обход всего дерева окончен, перейти к пункту 4;
- 3.в) если очередная вершина имеет только одну ветвь, то в качестве очередной вершины выбрать ту вершину, на которую эта ветвь указывает, перейти к пункту 2.
- 4. Конец алгоритма.

Для примера рассмотрим возможные варианты обхода дерева (рис. 4.1).

При обходе дерева представленного на рис.1 этими тремя методами мы получим следующие последовательности: ABCDEFG (нисходящий); CBAFEDG (смешанный); CBFEGDA (восходящий).

Рис.4.1. Схема дерева

ВОСХОДЯЩИЙ ОБХОД

Трудность заключается в том, что в этом алгоритме каждая вершина запоминается в стеке дважды: первый раз - когда обходится левое поддерево, и второй раз - когда обходится правое поддерево. Таким образом, в алгоритме необходимо различать два вида стековых записей: 1-й означает, что в данный момент обходится левое поддерево; 2-й - что обходится правое, поэтому в стеке запоминается указатель на узел и признак (код-1 и код-2 соответственно).

Алгоритм восходящего обхода можно представить следующим образом:

- 1) Спуститься по левой ветви с запоминанием вершины в стеке как 1-й вид стековых записей;
- 2) Если стек пуст, то перейти к п.5;
- 3) Выбрать вершину из стека, если это первый вид стековых записей, то возвратить его в стек как 2-й вид стековых записей; перейти к правому сыну; перейти к п.1, иначе перейти к п.4;
- 4) Обработать данные вершины и перейти к п.2;

- 5) Конец алгоритма.

4. Графы и их представление в компьютере.

Граф - это сложная нелинейная многосвязная динамическая структура, отображающая свойства и связи сложного объекта.

Многосвязная структура обладает следующими свойствами:

- 1) на каждый элемент (узел, вершину) может быть произвольное количество ссылок;
- 2) каждый элемент может иметь связь с любым количеством других элементов;
- 3) каждая связка (ребро, дуга) может иметь направление и вес.

В узлах графа содержится информация об элементах объекта. Связи между узлами задаются ребрами графа. Ребра графа могут иметь направленность, показываемую стрелками, тогда они называются ориентированными, ребра без стрелок - неориентированные.

Граф, все связи которого ориентированные, называется ориентированным графом или орграфом; граф со всеми неориентированными связями - неориентированным графом; граф со связями обоих типов - смешанным графом. Обозначение связей: неориентированных - (A,B), ориентированных - <A,B>. Примеры изображений графов даны на рис.1. Скобочное представление графов рис.5.1:

а).((A,B),(B,A)) и б).(< A,B >,< B,A >).

Рис.5.1. Граф неориентированный (а) и ориентированный (б).

Для ориентированного графа число ребер, входящих в узел, называется полустепенью захода узла, выходящих из узела -полустепенью исхода. Количество входящих и выходящих ребер может быть любым, в том числе и нулевым. Граф без ребер является нуль-графом.

Если ребрам графа соответствуют некоторые значения, то граф и ребра называются взвешенными. Мультиграфом называется граф, имеющий параллельные (соединяющие одни и те же вершины) ребра, в противном случае граф называется простым.

Путь в графе - это последовательность узлов, связанных ребрами; элементарным называется путь, в котором все ребра различны, простым называется путь, в котором все вершины различны. Путь от узла к самому себе называется циклом, а граф, содержащий такие пути - циклическим.

Два узла графа смежны, если существует путь от одного из них до другого. Узел называется инцидентным к ребру, если он является его вершиной, т.е. ребро направлено к этому узлу.

Логически структура-граф может быть представлена матрицей смежности или матрицей инцидентности.

Матрицей смежности для n узлов называется квадратная матрица adj порядка n. Элемент матрицы a(i,j) равен 1, если узел j смежен с узлом i (есть путь < i,j >), и 0 -в противном случае (рис.5.2).

Рис.5.2. Графа и его матрица смежности

Если граф неориентирован, то a(i,j)=a(j,i), т.е. матрица симметрична относительно главной диагонали.

Матрицы смежности используются при построении матриц путей, дающих представление о графе по длине пути: путь длиной в 1 - смежный участок - , путь длиной 2 - (< A,B >,< B,C >), ... в n смежных участков: где n - максимальная длина, равная числу узлов графа. На рис.5.3 даны путевые матирцы пути adj2, adj3, adj4 для графа рис.5.2.

Рис.5.3. Матрицы путей

Матрицы инцидентности используются только для орграфов. В каждой строке содержится упорядоченная последовательность имен узлов, с которыми данный узел связан ориетрированными (исходящими) ребрами. На рис.5.4 показана матрица инцидентности для графа

рис. 5.2.

Рис.5.4. Матрицы инцидентности

Существуют два основных метода представления графов в памяти ЭВМ: матричный, т.е. массивами, и связными нелинейными списками. Выбор метода представления зависит от природы данных и операций, выполняемых над ними. Если задача требует большого числа включений и исключений узлов, то целесообразно представлять граф связными списками; в противном случае можно применить и матричное представление.

Матричное представление орграфов.

При использовании матриц смежности их элементы представляются в памяти ЭВМ элементами массива. При этом, для простого графа матрица состоит из нулей и единиц, для мультиграфа - из нулей и целых чисел, указывающих кратность соответствующих ребер, для взвешенного графа - из нулей и вещественных чисел, задающих вес каждого ребра.

Например, для простого ориентированного графа, изображенного на рис.5.2 массив определяется как:

array[4][4]={{0,1,0,0},{0,0,1,1},{0,0,0,1},{1,0,1,0}}

Матрицы смежности применяются, когда в графе много связей и матрица хорошо заполнена.

Связное представление орграфов.

Орграф представляется связным нелинейным списком, если он часто изменяется или если полустепени захода и исхода его узлов велики. Рассмотрим два варианта представления орграфов связными нелинейными списковыми структурами.

В первом варианте два типа элементов - атомарный и узел связи На рис.5.5 показана схема такого представления для графа рис.5.2. Скобочная запись связей этого графа:

(< A,B >, < B,C >, < C,D >, < B,D >, < D,C >)

Рис.5.5. Машинное представление графа элементами двух типов

Более рационально представлять граф элементами одного формата, двойными: атом-указатель и указатель-указатель или тройными: указатель-data/down-указатель . На рис.5.6 тот же граф представлен элементами одного формата.

Рис.5.6. Машинное представление графа однотипными элементами

Многосвязная структура - граф - находит широкое применение при организации банков данных, управлении базами данных, в системах программного иммитационного моделирования сложных комплексов, в системах исскуственного интеллекта, в задачах планирования и в других сферах.

5. Алгоритмы, оперирующие со структурами типа графа.

Многие задачи оптимального выбора при прямом их решении предполагают перебор большого количества счетного числа вариантов. Теория графов позволяет во многих случаях решить

задачу гораздо эффективнее, чем непосредственный перебор вариантов. Граф - это геометрическая структура, состоящая из множества точек (вершин) и множества кривых (ребер). В алгоритмах, приведенных ниже, используются следующие обозначения: -{v} – множество вершин графа, – i-тая вершина,-{e} -- множество ребер графа,-- i-тое ребро,Смежными называют ребра, имеющие общую вершину. Граф, у которого ребрам приписаны направления, называется ориентированным, или орграфом. Последовательность попарно различных ребер, начинающихся и заканчивающихся в одной вершине, такая, что соседние ребра смежны, в неориентированном графе называется циклом, а в ориентированном - контуром. Если последовательность смежных ребер соединяет разные вершины, то она называется цепью (путем). Для последовательности с повторяющимися ребрами употребляют название маршрут. Если между любой парой вершин существует маршрут, то граф называют связанным. Граф без циклов называют лесом, а связанный граф без циклов называют деревом. Цепь без повторяющихся вершин называют простой. Граф называют планарным, если его можно расположить на плоскости так, что два различных ребра в качестве общей точки могут иметь только вершину. Остовом называется дерево, включающее все вершины графа. Эйлеровым циклом называется цикл, включающий все ребра графа по одному разу. Гамильтонов цикл проходит через все вершины графа по одному разу.

В зависимости от решаемой задачи используют различные способы задания графов:- задание матрицей весов ;- задание списком ребер, массивы END(2,M) и W(M); - задание списком смежности. Каждой вершине графа ставится в соответствие список смежных вершин.

Поиск в глубину

Данный алгоритм используется для поиска пути между двумя вершинами. В алгоритме используется вспомогательный стековый массив вершин Q, q – число элементов в стеке, массив меток вершин l(n) и массивы меток ребер. Изначально все вершины считаются непомеченными.

1. Поместим стартовую вершину в стек q=1, Q(q)=s.2. Если стек не пуст, то берем вершину из стека x=Q(q), q=q-1, в противном случае пути нет.3. Если все смежные вершины просмотрены, то выполняем пункт два. В противном случае рассматриваем очередную смежную с x вершину y. Если y - конечная вершина, то путь найден.4. Проверяем наличие метки у вершины y. Если l(y)>0, то помечаем реброкак обратное, в противном случае ребропомечаем как прямое и присваиваем метку l(y)=l(x)+1. Вершина y помещается в стек q=q+1, Q(q)=y.5. Выполнить пункт 3.

Построение остова минимального веса

1. Упорядочить ребра графа по возрастанию весов.2. Выбрать ребро с минимальным весом, не образующее цикл с выбранными ранее ребрами. Занести выбранное ребро в список ребер строящегося остова.3. Проверить, все ли вершины графа вошли в построенный остов. Если нет, то выполнить пункт 2.

Построение эйлерова цикла

1. Выходим из произвольной точки и помечаем ребра и вершины графа. На каждом шаге идем по ребру, удаление которого нарушает связанность графа, если другого пути нет.

Поиск кратчайшего пути

Будем характеризовать граф матрицей весов Wи дополнительными вспомогательными массивами last(j) - номер предыдущей вершины при шаге в вершину j, lenf(j) - длина пути из стартовой вершины s до вершины j, label(n) - массив вспомогательных меток.1. Положим lenf(j)=w(s,j), last(j)=s, label(j)=0, j=1,n.2. Найти i: lenf(i)<lenf(j) и label(j)=0 для всех j. Присвоить label(i)=1. Если i конечная вершина, то задача решена.3. Для всех label(j)=0 и lenf(j)>lenf(i)+w(i,j) присвоить lenf(j)=lenf(i)+w(i,j), last(j)=i.4. Выполнить пункт 2.

Алгоритм поиска в ширину

Вводится вспомогательный массив номеров вершин Q, массив меток l(j) и матрица смежности v(i,k). Значение метки равно удаленности вершины от исходной.1. q=1, Q(q)=s, j=1, l(s)=0.2. k=Q(j) - берем вершину из рабочего массива,3. i=1, y=v(i,k) - первая смежная с k вершина,4. Если

смежная с k вершина y имеет метку, то выполнить п.5, иначе l(y)=l(k)+1, last(y)=k, q=q+1, Q(q)=y.5. Если у k-ой вершины есть еще смежные вершины, то i=i+1 и выполнить пункт 4, иначе j=j+1.6. Если j <=q, то выполнить пункт 2.

Построение наибольшего паросочетания в двудольном графе

Паросочетанием называется множество несмежных ребер графа. Вершины, смежные с ребрами из паросочетания, называются насыщенными. Чередующимся путем называют путь, состоящий из чередующихся ребер графа, включенных и не включенных в паросочетание. Построение наибольшего паросочетания в графе основывается на отыскании чередующегося пути из ненасыщенной вершины до ненасыщенной вершины.1. Построим какое-либо паросочетание M в графе G.2. По графу G и паросочетанию M построить вспомогательный граф Г=X+Y: всем ребрам из паросочетания M присваивается ориентация из X в Y, остальные ребра графа G получают обратное направление.3. Выполнить в графе Г поиск в ширину из множества Y ненасыщенных вершин.4. Если в результате поиска одна из вершин множества X получила метку, то найден добавляющий путь. Произвести инверсию ребер добавляющего пути относительно паросочетания.

Алгоритм укладки графа на плоскости

Назовем гранью графа G часть плоскости, ограниченную ребрами графа, такую, что любые две ее точки могут быть соединены жордановой кривой, непересекающей ребра графа. Сегментом графа G относительно его подграфа G–G1 назовем ребро графаG1 =C, смежное с вершинами графаG–G1, или связанную компоненту графаG1 =C, дополненную смежными вершинами графаG–G1.1. Выберем некоторый простой цикл C графа G и уложим его на плоскости. ПоложимE.2. Найдем грани графаG–G1 и сегменты относительноG–G1. Если множество сегментов пусто, то задача решена.3. Для каждого сегмента определим число граней, в которых он может быть расположен. 4. Выберем сегмент, для которого число возможных расположений минимально. Если это число нуль, то граф невозможно расположить на одной плоскости.5. Присоединим выбранный сегмент к графу G–G1. Повторим п. 2.

6. Задачи поиска.

Основной вопрос задачи поиска: где в заданной совокупности данных находится элемент, обладающий заданным свойством? Большинство задач поиска сводится к простейшей — к поиску в массиве элемента с заданным значением.

Рассмотрим именно эту задачу. Пусть требуется найти элемент X в массиве A. В данном случае известно только значение разыскиваемого элемента, никакой дополнительной информации о нем или о массиве, в котором производится поиск, нет. Наверное, единственным способом является последовательный просмотр массива и сравнение значения очередного рассматриваемого элемента массива с X. Напишем фрагмент:

```
for(i=1;i<N;i++)

if (A[i]==X) k=i;
```

В этом цикле находится индекс последнего элемента A, равного X, при этом массив просматривается полностью.

Эффективность алгоритма: O(N).

7. Исчерпывающий поиск: перебор с возвратом, метод ветвей и границ, динамическое программирование.

Поиск с возвращением

Во многих задачах, формулируемых в виде вопросов вроде "сколько существует способов...", "найти множество всех возможных способов...", "определить, есть ли способ..." и т.д., обычно требуется осуществить исчерпывающий поиск в некотором конечном множестве, заведомо содержащем все искомые способы, и называемом множеством всех возможных решений. Например, при решении задачи отыскания всех простых чисел, меньших 10000, в качестве

множества всех возможных решений можно взять множество всех целых чисел от 1 до 10000, а при нахождении пути через лабиринт множеством всех возможных решений может служить множество всех путей, начинающихся входом в лабиринт.

Лабиринт (целью пути является черный квадрат; стрелками обозначены возможные продолжения текущего пути)

Общим методом организации исчерпывающего поиска является так называемый *поиск с возвращениями*или*возвратный ход*по упорядоченному множеству частичных возможных решений. Понятие*частичное возможное решение*, с помощью которого формулируется алгоритм поиска с возвращением, мы иллюстрируем ниже в примере решения задачи поиска выхода из лабиринта. Заметим здесь, что каждое частичное решение фиксирует определенное подмножество возможных (*полных*) решений, не противоречащих данному частичному решению, т.е. являющихся его*расширениями*. Этот метод -- метод поиска с возвращением -- основан на том, что мы многократно пытаемся расширить текущее частичное решение, или, что то же самое, сузить множество тех возможных полных решений, которые не противоречат текущему частичному решению. Если расширение невозможно на текущем шаге поиска, то происходит возврат к предыдущему более короткому частичному решению и делается попытка его расширения, но уже другим способом.

Идею поиска с возвращением можно продемонстрировать на примере решения задачи поиска выхода из лабиринта, в которой требуется попасть из входного квадрата в некоторый заданный квадрат путем последовательного перемещения по соединенным смежным квадратам. Поиск с возвращением осуществляет движение по лабиринту в соответствии со следующими двумя правилами:

1) *продвижение:*из текущего квадрата нужно переходить в еще не исследованный соседний квадрат,

2) *возврат:* если все соседние квадраты уже исследованы, то нужно вернуться на один квадрат назад по пройденному пути.

Первое правило говорит о том, как расширить исследуемый путь, если это возможно, а второе -- о том, как выходить из тупика. Заметим, что каждый исследуемый путь определяет множество всех тех возможных путей по лабиринту, начальным отрезком которых он является, и, таким образом, удлинение исследуемого пути сужает связанное с ним множество возможных путей по лабиринту, а сокращение расширяет это множество. В этом и состоит сущность поиска с возвращением: продолжать расширение найденного частичного решения до тех пор, пока это возможно, и, когда текущее частичное решение нельзя расширить, возвращаться по нему и пытаться сделать другой выбор по расширению частичного решения на самом близком шаге, где имеется такая возможность.

Алгоритм поиска с возвращением

Рассмотрим общий случай, когда решение задачи имеет вид вектора (a1,a2,a3,...), длина которого не определена, но ограничена сверху некоторым (известным или неизвестным) числомr, а каждоеa[i] является элементом некоторого конечного линейно упорядоченного множестваAi. Таким образом, при исчерпывающем поиске в качестве возможных решений мы рассматриваем элементы множестваA1xA2x...xAnдля любогоi, гдеi<=r, и среди них выбираем те, которые удовлетворяют ограничениям, определяющим решение задачи.

В качестве начального частичного решения берется пустой вектор () и на основе имеющихся ограничений выясняется, какие элементы из A1 являются кандидатами для их рассмотрения в качествеa1 (множество таких элементова1 изA1 ниже обозначается черезS1). В качествеa1 выбирается наименьший элемент множестваS1, что приводит к частичному решению (a1). В общем случае ограничения, описывающие решения, говорят о том, из какого подмножестваSkмножестваAkвыбираются кандидаты для расширения частичного решения от (a1,a2,...,a(k-1)) до (a1,a2,...,a(k-1),ak) . Если частичное решение (a1,a2,...,a(k-1)) не предоставляет других возможностей для выбора новогоak(т.е. у частичного решения (a1,a2,...,a(k-1)) либо нет кандидатов для расширения, либо все кандидаты к данному моменту уже использованы), то происходит возврат и осуществляется выбор нового элементаa(k-1) изS(k-1). Если новый элемента(k-1) выбрать нельзя, т.е. к данному моменту множествоS(k-1) уже пусто, то происходит еще один возврат и делается попытка выбрать новый элемента(k-2) и т.д.

Для иллюстрации того, как описанный метод применяется при решении конкретных задач,

рассмотрим задачу нахождения таких расстановок восьми ферзей на шахматной доске, в которых ни один ферзь не атакует другого. Решение расстановки ферзей можно искать в виде вектора (a1,a2,a3,a4,a5,a6,a7,a8), гдеai-- номер вертикали, на которой стоит ферзь, находящийся ві-й горизонтали, т.е.A1=A2=A3=A4=A5=A6=A7=A8={1,2,3,4,5,6,7,8}. Каждое частичное решение -- это расстановкаNферзей (где 1<=N<=8) в первыхNгоризонталях таким образом, чтобы эти ферзи не атаковали друг друга. Заметим, что общая процедура поиска с возвращением при применении ее к задаче о расстановке ферзей уточняется таким образом, что в ней не вычисляются и не хранятся явно множестваSk.

Процесс поиска с возвращением удобно описывать в терминах обхода в глубину (см. ниже) дерева поиска решения, которое строится следующим образом. Корень дерева поиска решения (нулевой уровень) соответствует пустому вектору, являющемуся начальным частичным решением. Для любого k>=1 вершиныk-го уровня, являющиеся сыновьями некоторой вершиныr, соответствуют частичным решениям (a1,a2,...,a(k-1),ak) , где (a1,a2,...,a(k-1)) -- это то частичное решение, которое соответствует вершинеr, aakпринадлежитSk; при этом упорядоченность сыновей вершиныrотражает упорядоченность соответствующих элементовakвSk.

Метод ветвей и границ

Широко используемый вариант поиска с возвращением, называется *методом ветвей и границ*, фактически является лишь специальным частным случаем метода поиска с ограничениями. Ограничения в данном случае основываются на предположении, что на множестве возможных и частичных решений задана некоторая функция*цены*и что нужно найти оптимальное решение, т.е. решение с наименьшей ценой. Для применения метода ветвей и границ функция цены должна обладать тем свойством, что цена любого частичного решения не превышает цены любого расширения этого частичного решения (Заметим, что в большинстве случаев функция цены неотрицательна и даже удовлетворяет более сильному требованию

ЦЕНА(a1,a2,...,ak) = ЦЕНА(a1,a2,...,a(k-1))+C(ak), гдеC(ak)>=0 – функция, определенная для всехak).

Это свойство позволяет отбрасывать любое частичное решение в процессе поиска, если его цена больше цены ранее вычисленного решения. Такие ограничения использованы в приведенном ниже варианте алгоритма поиска:

```
минимум := бесконечность;  (* максимальное представимое число *)

k:= 1; ВычислитьS1;while k >0do        while(Не пустоSk)and(цена < минимум)do
(* Продвижение:*)

            В качестве akвзять наименьший элемент изSk, удалив его изSk;                цена :=
ЦЕНА (a1,a2,...,ak);              if((a1,a2,...,ak) -- решение) & (цена < минимум)thenминимум := цена;

                Хранить далее (a1,a2,...,ak) как решение с наименьшей ценой
end;          if k < rthen                  k:=k- 1; ВычислитьSk          end       end;

      (* Возврат *)

      k:=k-1; цена := ЦЕНА (a1,a2,...,ak)
end.
```

При применении метода ветвей и границ рекомендуется использовать технику перестройки дерева поиска, с тем чтобы решения, близкие к оптимальному, находились на ранних этапах поиска.

Динамическое программирование

Пусть имеется представление исходной задачи в виде иерархии вложенных подзадач разного размера таким образом, что есть прямые алгоритмы решения элементарных (не содержащих других) подзадач и есть простое сведение любой неэлементарной подзадачи (в том числе и самой задачи) к подзадачам меньшего размера.

В этом случае для решения задачи можно использовать очевидный рекурсивный алгоритм. Однако, не всегда рекурсивная техника дает удовлетворительное решение. Например, если разбиение таково, что решение размера nсводится кnподзадачам размераn- 1, то рекурсивный

алгоритм будет, по-видимому, не приемлемым из-за его экспоненциальной сложности. Другой пример, это возможная неэффективность рекурсивного алгоритма, возникающая из-за большого числа повторений решения одной и той же подзадачи, как это имеет место, например, при вычислениин-ого числа Фибоначчи по формулеFib(n)=Fib(n-1)+Fib(n-2).

Часто в этом случае можно получать эффективные алгоритмы с помощью табличной техники, называемой *динамическим программированием*. Динамическое программирование предполагает последовательное решение всех подзадач исходной задачи в порядке возрастания их размера с запоминанием ответов в специальной таблице по мере их получения. Преимущество этого метода по сравнению с рекурсивным алгоритмом состоит в том, что раз уж подзадача решена, ее ответ размещен в таблицу и в дальнейшем никогда не вычисляется повторно. Ответ подзадачи берется из таблицы каждый раз, когда он нужен для нахождения решения подзадачи большего размера. Понятно, что эта техника позволяет получать эффективные алгоритмы в тех случаях, когда объем хранимой в таблице информации оказывается не слишком большим.

Рассмотрим применение техники динамического программирования еще на одном простом примере. Рассматривается множество так называемых *раскроев*выпуклогоn-угольника или, что то же самое, различных разбиений его на треугольники с помощьюn- 3 непересекающихся диагоналей.*Стоимость*конкретного раскроя -- это суммарная длина проведенных диагоналей. Пусть дан выпуклыйn-угольник. Требуется найти его раскрой минимальной стоимости.

Будем считать, что все вершины исходного многоугольника пронумерованы в порядке обхода n-угольника по часовой стрелки, т.е.-угольник образован отрезками (1,2),(2,3),…, (n-1,n),(n,1). ЧерезM(k,l) обозначим многоугольник, отрезаемый от исходного диагональю (k,l), гдеk<=l. Он образован отрезками (k,l), (k+1,k+2),… , (l-1,l). Ясно, что приl=k+ 1 многоугольникM(k,l) вырождается в отрезок, приl=k+ 2 он – треугольник, аM(1,n) -- это исходныйn-угольник.

Для любых вершин kиlчерезq(k,l) обозначим минимальную стоимость раскроя многоугольникаM(k,l), еслиl>k+2, и полагаемq(k,l)=0, еслиl=k+1 илиl=k+2. Получаем следующее очевидное рекуррентное соотношение:

q(k,l)=min{q(k,i)+q(i,l)+d(k,i)+d(i,l):k < i < l},

где d(s,r) – это длина диагонали (s,r), еслиr>s+ 1, иd(s,r), еслиr=s+ 1.

Указанное соотношение позволяет составить таблицу для q(k,l), заполняя ее в порядке возрастания числа (l–k+ 1) -- количества вершин в многоугольникеM(k,l).

После того, как вычислена таблица значений q(k,l), мы получаем минимальную стоимость раскроя (она равнаq(1,n)) и легко можем вычислить раскрой минимальной стоимости. Для этого можно реализовать процесс обработки подзадач в порядке убывания их размера. Процесс начинается обработкой исходной задачи и постепенно шаг за шагом находит диагонали, образующие разрез минимальной стоимости. На каждом шаге рассматривается многоугольникM(k,l) и ищется такая вершинаi,k<i<l, чтоq(k,l)=q(k,i)+q(i,l)+d(k,i)+d(i,l).

< Предыдущая 1 2 3 4 5 Следующая >

Соседние файлы в папке Лекции

kontr_sroki.doc	0	#	20.48 Кб	20.06.2014
Primery_lectiy.doc	13	#	722.43 Кб	20.06.2014
Rabochaya prog.doc	1	#	56.83 Кб	20.06.2014