МИНОБРНАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное

учреждение высшего образования

«ЮЖНЫЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт математики, механики и компьютерных наук им. И.И. Воровича

Адигеев М.Г.

ПЕРЕБОРНЫЕ АЛГОРИТМЫ

Методические указания для студентов

Института математики, механики и компьютерных наук

Ростов-на-Дону

2018

Методические указания разработаны кандидатом технических наук, доцентом кафедры алгебры и дискретной математики М.Г. Адигеевым.

Печатается в соответствии с решением кафедры алгебры и дискретной математики Института математики, механики и компьютерных наук ЮФУ, протокол № _____ от _______________ 2018 г.

Содержание

Введение 4
1 Вычислительно сложные задачи 5
1.1 Задача о рюкзаке 5
1.2 Задача об упаковке ящиков 5
1.3 Задача о вершинном покрытии 6
1.4 Задача коммивояжёра 7
2 Алгоритмы полного перебора 8
2.1 Общая схема переборных алгоритмов 8
2.2 Двоичные строки 10
2.3 Строки в произвольном алфавите 13
2.3.1 Переборный алгоритм для задачи об упаковке ящиков 14
2.3.2 Переборный алгоритм для задачи коммивояжёра 15
2.4 Сочетания 17
2.5 Перестановки 19
3 Метод ветвей и границ 21
3.1 Идея метода ветвей и границ 21
3.2 Метод ветвей и границ для задачи коммивояжёра 22
3.2.1 Представление цикла в виде перестановки вершин 22
3.2.2 Представление цикла в виде подмножества дуг 23
Литература 24

Введение

Данные методические указания разработаны для поддержки курса «Алгоритмы решения вычислительно сложных задач», читаемого студентам 2-го курса магистратуры Института математики, механики и компьютерных наук.

В данных методических указаниях переборные алгоритмы разбираются на примере нескольких вычислительно сложных оптимизационных задач. Формулировки задач приведены в главе 1. Все рассмотренные задачи относятся к классу NP-трудных [1, 2, 3]. Что это означает на практике? Что в настоящее время не известны эффективные (имеющие полиномиальную временную сложность) алгоритмы для получения точного решения этих задач. Но, поскольку задачи важны для практики, остаются следующие альтернативы:

а) Решать задачи точно, пусть и за экспоненциальное (в худшем случае) время.

б) Находить эффективно решаемые частные случаи.

в) Искать приближённые решения.

В данных методических указаниях рассмотрен подход (а). Он основан на применении переборных алгоритмов. Эти алгоритмы так называются потому, что в худшем случае перебирают все допустимые решения задачи и выбирают из них оптимальное. Обратите внимание на «в худшем случае». Это уточнение важно: то, что задача является NP-трудной, не означает, что все экземпляры задачи требуют экспоненциального времени. Некоторые (а иногда – большинство) экземпляры можно решать гораздо быстрее, если применить специальные приёмы. Одним из таких приёмов является метод ветвей и границ, также рассмотренный в данных методических указаниях.

1 Вычислительно сложные задачи

Рассмотрим несколько вычислительно сложных задач, на примере которых будем изучать переборные алгоритмы решения.

1.1 Задача о рюкзаке

Задача о рюкзаке является известной комбинаторной оптимизационной задачей. Существует несколько вариантов этой задачи; мы рассмотрим вариант, который также называется «задача о 0-1 рюкзаке».

Дано:

- n предметов, для каждого задан вес w_i и стоимость c_i.
- b - предельный допустимый суммарный вес предметов, которые можно поместить в рюкзак.

Решением задачи является подмножество предметов $I \subseteq \{1,..., n\}$, при котором $c(I) = \sum_{i \in I} c_i \to max$ при условии, что $\sum_{i \in I} w_i \leq b$.

Этот вариант называется «0-1 рюкзак» потому, что каждый предмет присутствует в единственном экземпляре и может быть либо включён в решение, либо не включён. В других вариантах задачи допускается возможность положить в рюкзак несколько вариантов одного и того же предмета.

1.2 Задача об упаковке ящиков

Задача «Упаковка ящиков» также имеет несколько вариантов. Мы рассмотрим одномерный вариант (1D Bin Packing).

Дано:

- n предметов, для каждого задан вес w_i.
- вместимость ящика b.

В отличие от задачи о рюкзаке, разместить надо *все* предметы, но зато количество ящиков не ограниченно (все ящики идентичны и имеют одинаковую вместимость). Требуется найти решение, при котором используется минимальное

количество ящиков. Ниже (Рисунок 1) приведён пример задачи (веса предметов соответствуют их линейным размерам) и оптимальная упаковка в три ящика.

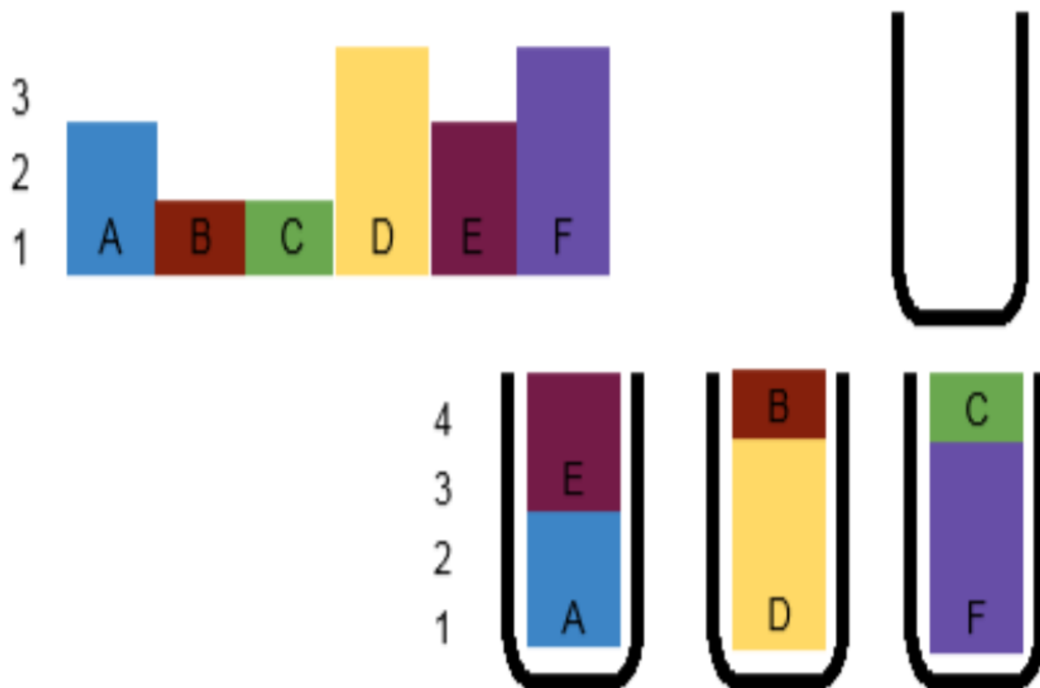

Рисунок 1

1.3 Задача о вершинном покрытии

Определение 1. Пусть дан граф G(V, E). *Вершинным покрытием* называется подмножество вершин $U \subseteq V$, такое что $\forall\ e \in E \quad \exists\ u \in U$: ребро e инцидентно вершине u.

Задача о вершинном покрытии заключается в том, чтобы на заданном графе найти минимальное по мощности вершинное покрытие U. Рисунок 2 демонстрирует минимальное покрытие (выделенные серым вершины) на заданном графе. Существует также обобщение данной задачи — для вершин графа заданы стоимости, и требуется найти покрытие минимальной суммарной стоимости. Однако в данной работе мы будем рассматривать описанный вариант, в котором стоимости всех вершин полагаются равными 1.

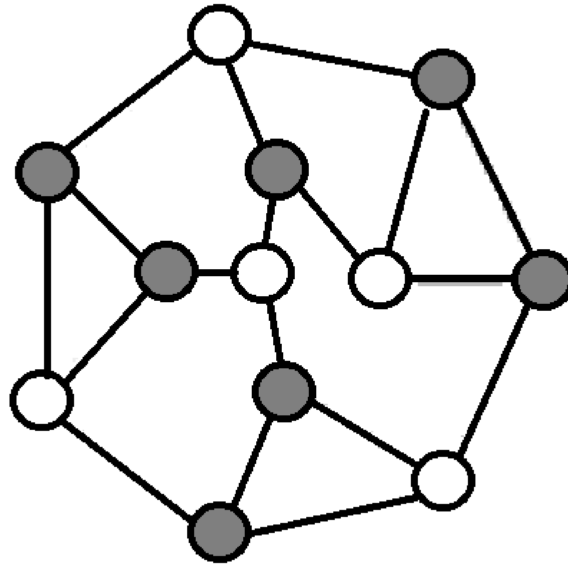

Рисунок 2

Задача о вершинном покрытии является частным случаем задачи о покрытии множества. Однако и сама по себе относится к NP-трудным задачам.

1.4 Задача коммивояжёра

Определение 2*.* Пусть дан граф. Путь или цикл на нём называется *гамильтоновым*, если он проходит через все вершины графа ровно по одному разу.

Определение 3*.* Граф называется *гамильтоновым*, если он содержит гамильтонов цикл.

Задача коммивояжёра (ЗК) формулируется следующим образом: для заданного графа $G(\mathrm{V},E)$ и заданных стоимостей дуг c: $E \to R_+$ найти гамильтонов цикл наименьшей стоимости. Под стоимостью цикла понимают ***сумму*** стоимостей дуг, принадлежащих циклу.

Рассматривают также «незамкнутую» версию задачи коммивояжёра, в которой требуется найти гамильтонов путь наименьшей стоимости. Но мы будем рассматривать только классический вариант задачи, заключающийся в нахождении гамильтонова цикла.

Задача коммивояжёра может рассматриваться как на ориентированном, так и на неориентированном графе. Точные алгоритмы решения для этих вариантов аналогичны, однако для неориентированных графов с дополнительными ограничениями (функция стоимости удовлетворяет неравенству треугольника) существуют эвристические алгоритмы с гарантированной оценкой точности. Задача коммивояжёра, удовлетворяющая таким ограничениям, называется ***метрической***.

Мы будем рассматривать общий случай (то есть большинство наших рассуждений будут применимы как для ориентированного варианта, так и для неориентированного); в тех случаях, когда вид графа важен, это будет оговорено дополнительно.

Без потери общности можно считать, что граф является полным. Если это не так, то отсутствующим рёбрам назначим «бесконечную» стоимость. На практике это означает, что им в качестве стоимости устанавливается число, настолько большое, что такая фиктивная дуга заведомо не попадёт в решение (при условии, конечно, что на исходном графе есть гамильтонов цикл). Например, в качестве такой «бесконечности» можно взять максимальную стоимость реальной дуги графа, умноженную на количество вершин. Заметим, что в упомянутом выше варианте метрической задачи коммивояжёра уже исходный граф обязательно является полным, иначе функция стоимости не будет удовлетворять неравенству треугольника.

2 Алгоритмы полного перебора

2.1 Общая схема переборных алгоритмов

В основе алгоритмов полного перебора лежит очень простая идея:

1. Последовательно перебирать все варианты, которые потенциально могут задавать решение задачи.

2. Для каждого варианта потенциального решения проверять, является ли оно допустимым.
3. Если это допустимое решение, то проверять его на оптимальность, сравнивая с текущим «рекордом».

В итоге общая схема переборного алгоритма оказывается очень похожей на алгоритм поиска минимального или максимального значения в массиве – только вместо перебора элементов заранее заданного массива необходимо перебирать потенциальные решения задачи. Ниже (Рисунок 3) приведён «псевдокод» обобщённого переборного алгоритма решения задачи для случая минимизации. Модификация алгоритма для задач максимизации остаётся в качестве упражнения.

Процедура Process() выполняет операции, специфические для конкретной задачи: проверяет сгенерированный вариант на допустимость, и если он допустим – сравнивает с рекордом. В случае если текущий допустимый вариант лучше рекордного (для минимизации – его стоимость меньше, чем стоимость рекорда), выполняется обновление рекорда.

```
GeneralizedBruteForce()
{
    RecordSolution := NULL;
    RecordCost := +∞;
    While (перебрали не все потенциальные решения):
    {
        CurSolution := Yield(); // Генерируем очередной вариант
        Process(CurSolution, RecordSolution, RecordCost);
    }
    Return RecordSolution;
}
```

Рисунок 3

Ключевым моментом описанной процедуры является генерация потенциальных решений. В общем случае, эта задача непроста сама по себе. К счастью, во многих важных задачах потенциальное решение задачи можно представить в виде комбинаторной конфигурации, принадлежащей небольшому перечню. Для переборного решения задач, рассматриваемых в данной работе, достаточно научиться перебирать следующие комбинаторные конфигурации [4]:

- Двоичные строки (вектора) заданной длины.
- Строки (вектора) в произвольном заданном алфавите.
- Сочетания.
- Перестановки.

Ниже рассмотрены алгоритмы генерации этих конфигураций, и примеры применения таких алгоритмов для решения рассматриваемых задач.

2.2 Двоичные строки

Определение 4. *Алфавитом* называется непустое конечное множество. Элементы алфавита называются *символами* или *буквами*.

Определение 5. *Строкой* в заданном алфавите называется конечная последовательность символов из этого алфавита. Один и тот же символ может встречаться в строке несколько раз. Количество символов в строке, с учётом их кратности (количества повторений) называется *длиной* строки. Например, ‘aabacb’ – строка длины 6 в алфавите {a,b,c}.

Определение 6. *Двоичной* (*бинарной*) строкой называется строка в алфавите {0,1}.

В текущем параграфе мы рассмотрим алгоритм генерации всех двоичных строк заданной длины, и его практическое применение. Алгоритм генерации строк в произвольном алфавите рассмотрен в следующем параграфе.

Рекурсивный вариант алгоритма представлен ниже (Рисунок 4). В него сразу «встроена» обработка генерируемых строк. Этот алгоритм, как и остальные рассматриваемые далее, состоит из двух процедур: одна процедура выполняет

этап инициализации (создаёт и инициализирует необходимые структуры данных) и вызывает вторую процедуру. Вторая процедура рекурсивно перебирает требуемые комбинаторные конфигурации.

```
GenerateBinaryStrings(n):
{
    Создать массив S[1..n];
    S := [NULL, NULL, …, NULL];
    ProcessBinaryStrings(S, 1, n);
}

ProcessBinaryStrings(S, k, n)
{
    if k > n then Process(S)
    else
    {
        S[k] := 0;
        ProcessBinaryStrings(S, k+1);
        S[k] := 1;
        ProcessBinaryStrings(S, k+1);
    }
}
```

Рисунок 4

Обратите внимание, что последовательность, в которой генерируются строки, можно представить в виде обхода в глубину виртуального дерева вариантов, представленного ниже (Рисунок 5). Дерево названо *виртуальным* потому, что оно никогда не строится в явном виде и не хранится в памяти компьютера (это привело бы к быстрому переполнению памяти при решении практических задач более-менее реалистичного масштаба), а просто описывает

логическую связь между генерируемыми вариантами и порядок их генерации и обработки. Представление перебора вариантов в виде дерева очень полезно при рассмотрении различных способов оптимизации перебора – в частности, метода ветвей и границ.

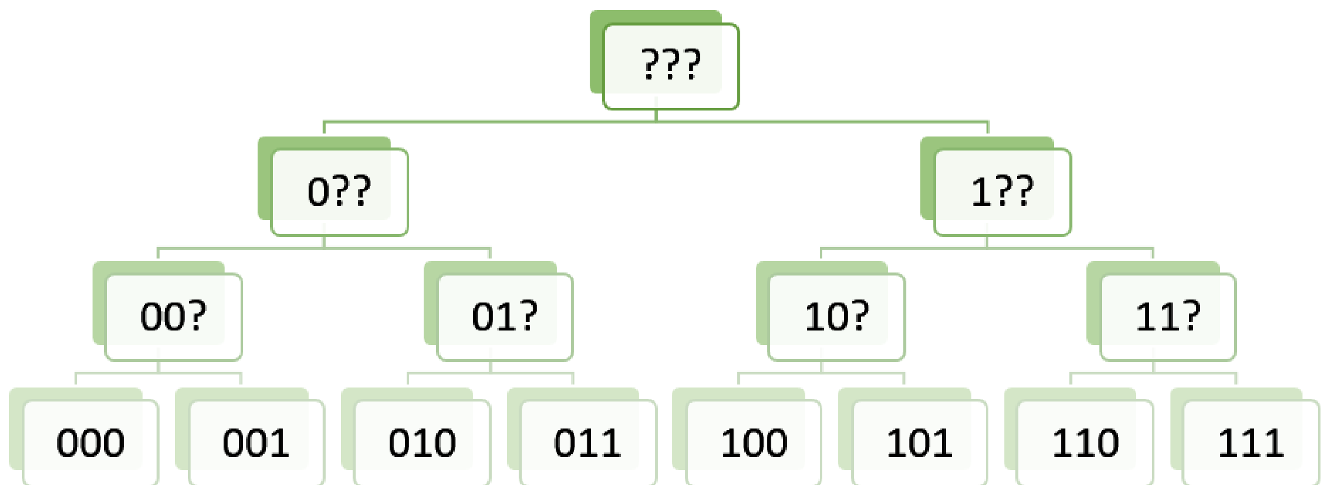

Рисунок 5

Как рассмотренный алгоритм можно применить для решения задач комбинаторной оптимизации? Из курса дискретной математики [4] известно, что существует взаимно-однозначное соответствие между двоичными строками длины n (или, что в данном случае равносильно, векторами из $\{0,1\}^n$) и всеми подмножествами n-элементного множества. Потенциальные решения одной из рассмотренных ранее задач — задаче о рюкзаке — как раз и являются подмножествами множества предметов. Это позволяет применить рассмотренный алгоритм для решения задачи о рюкзаке. Для этого достаточно доопределить процедуру Process, включив в неё проверку текущей двоичной строки на допустимость (найти сумму весов предметов, соответствующих ‘1’-м в строке, и сравнить её с вместимостью рюкзака) и на оптимальность (вычислить суммарную стоимость предметов, соответствующих ‘1’-м в строке, и сравнить с рекордной стоимостью). Детали реализации остаются на самостоятельную проработку.

2.3 Строки в произвольном алфавите

Рассмотрим обобщение задачи: пусть вместо алфавита {0, 1} задан произвольный алфавит (множество символов) $\{a_1,a_2,\ldots,a_m\}$. Ниже (Рисунок 6) приведён алгоритм для генерации всех строк в заданном алфавите. Нетрудно заметить, что он является обобщением ранее рассмотренного алгоритма для двоичных строк.

```
GenerateGeneralizedStrings(n):
{
    Создать массив S[1..n];
    S := [NULL, NULL, …, NULL];
    ProcessGeneralizedStrings (S, 1, n);
}

ProcessGeneralizedStrings(S, k, n)
{
    if k > n then Process(S)
    else
    {
        for i:=0 to m-1 do
            S[k] := a_i;
            ProcessGeneralizedStrings (S, k-1, n);
    }
}
```

Рисунок 6

Процесс генерации и обработки вариантов также можно представить в виде дерева перебора — см.Рисунок 7 для случая алфавита {a,b,c}.

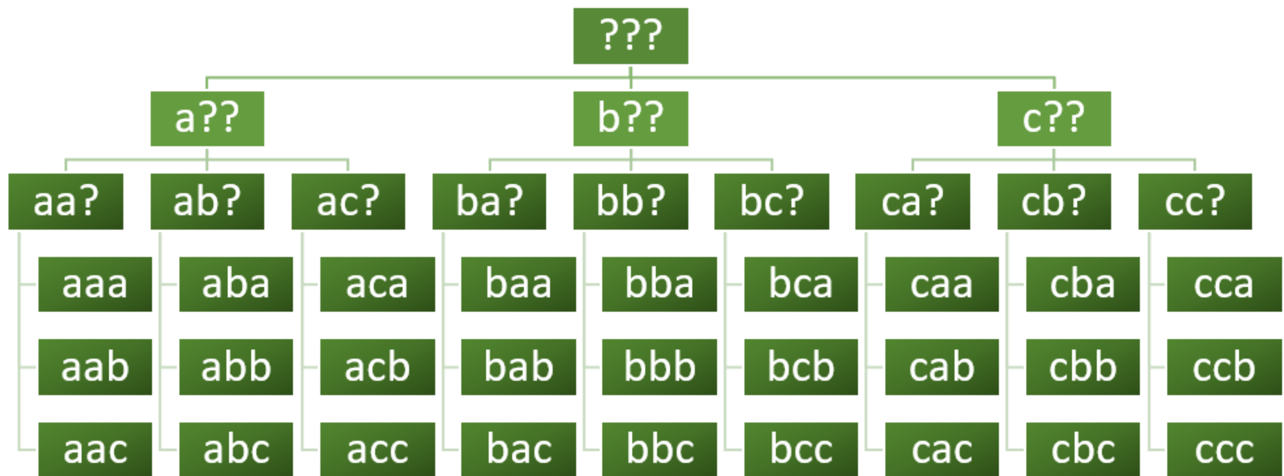

Рисунок 7

Вместе с тем, рассмотренный вариант алгоритма можно обобщить ещё больше, и допустить случаи, когда в разных позициях строки используются разные алфавиты. Для этого в цикле по i нужно устанавливать разные границы для разных *k*:

```
for i:=0 to m[k]-1 do
```

Более того, верхние границы (а также сами «алфавиты») можно динамически изменять в процессе решения. Этот приём мы сейчас и используем при решении задачи об упаковке ящиков.

2.3.1 Переборный алгоритм для задачи об упаковке ящиков

Идея переборного решения:

- Выбрать m — оценку сверху для количества ящиков.
- Перебрать все слова в алфавите $\{1,..., m\}$.
- При обработке потенциального решения A[i] = j означает, что i-й предмет уложен в j-й ящик.

Как выбирать оценку количества ящиков? Тривиальный выбор: $m = n$. Действительно, если предположить (что логично, иначе задача не имеет решения), что вес каждого предмета не превышает вместимость ящика, то для размещения n предметов достаточно использовать $m=n$ ящиков. Возможны и более точные

оценки (придумайте 1-2 варианта). При этом, независимо от первоначальной оценки m, мы можем (используемые алгоритм это допускает, без ущерба для корректности) уменьшать m по мере обнаружения всё более хороших допустимых решений. А именно, в качестве m можно использовать количество ящиков в текущем рекордном решении. Очевидно, это позволяет сильно сократить количество перебираемых вариантов по сравнению с «наивным» алгоритмом, перебирающим все строки в фиксированном алфавите $\{1,\ldots,m\}$.

2.3.2 Переборный алгоритм для задачи коммивояжёра

Перебор слов в заданном алфавите можно использовать и для решения задачи коммивояжёра. Пусть что граф задан списком смежности.

Введём обозначения:

- N[i] — список смежности для i-й вершины
- D[i] — степень i-й вершины, т. е. D[i] = | N[i] |

Пусть Z — искомый гамильтонов цикл. Последовательность вершин Z храним в массиве S[1..n] в обратном порядке: S[n] → S[n–1] → S[n–2] → … → S[1] → S[n].

В массиве S перебираем все «обобщённые» строки длины n. В обобщённой строке в i-й позиции может стоять символ из алфавита {1,..., D[S[i+1]]}. А именно: значение S[i]=j означает, что цикл Z из вершины S[i+1] переходит в вершину N[i][j], т.е. в j-ю вершину из списка смежности для i-й вершины графа.

Чтобы не допускать появление дублей вершин в Z, введём ещё один массив U[1..n]; в массиве U[i] отмечаем номера ещё не использованных вершин (т. е. U[i] = True означает, что i-я вершина ещё не включена в Z).

Итоговый вариант алгоритма приведён ниже (Рисунок 8).

Оценим сложность алгоритма. Пусть максимальная степень вершины на графе G (для ориентированного случая — максимальная степень исхода) равна d. Тогда временная сложность данного алгоритма оценивается величиной O(d^n).

```
TSP_GeneralizedStrings(G):
{
    // Инициализация
    for i:=1 to n-1 do  U[i] := True;
    U[n] := False;     S[n]:=n; // фиксируем последнюю вершину
    // Вызываем рекурсивную процедуру
    TSP_ProcessGeneralizedStrings(S, U, n-1);
}

TSP_ProcessGeneralizedStrings(S, U, k):
{
    if k = 0 then Process(S)
    else
        for j:=1 to D[S[k+1]] do
        {
                m := N[S[k+1], j];
                if U[m] then
                {
                    U[m] := False;
                    S[k] := m;
                    TSP_ProcessGeneralizedStrings(S,U,k-1);
                    U[m] := True;
                }
        }
}
```

Рисунок 8

2.4 Сочетания

Рассмотрим задачу о вершинном покрытии (параграф 1.3). Её решением является подмножество вершин на графе. Поэтому для нахождения вершинного покрытия, в принципе, можно применить алгоритм перебора двоичных строк, рассмотренный в параграфе 2.2. Однако, этот алгоритм генерирует и обрабатывает строки (=подмножества) в порядке, не удобном для нахождения минимального по мощности подмножества, и для точного решения задачи придётся перебрать почти все варианты. Более удобным было бы перебирать варианты, начиная с наиболее перспективных (содержащих меньшее количество элементов). Тогда общую схему алгоритма можно представить так, как на рисунке ниже (Рисунок 9). Действительно, при таком порядке перебора первое допустимое решение (подмножество вершин, являющееся покрытием) является оптимальным решением задачи.

```
FindMinVertexCover(G)
{
    for k:=1 to n:
    {
        Перебрать все k-элементные подмножества вершин;
        if (найдено покрытие) then Return найденное покрытие;
    }
}
```

Рисунок 9

Для того чтобы воспользоваться данным алгоритмом, необходимо научиться перебирать все k-элементные подмножества заданного n-элементного множества. Такие подмножества называются *сочетаниями* [4]. Алгоритм перебора всех сочетаний приведён на рисунке ниже (Рисунок 10).

```
GenerateCombinations(n, k):
{
    Создать массив S[1..k];
    ProcessCombinations(S, n, k);
}

ProcessCombinations(S, n, k):
{
    if k = 0 then Process(S)
    else
      for i = k to n do
      {
         S[k] := i;
        ProcessCombinations(S, i-1, k-1);
      }
}
```

Рисунок 10

Теорема. Алгоритм *GenerateCombinations*

1) Корректен, т. е.

- Обрабатывает только сочетания из n по k.
- Обрабатывает все сочетания из n по k.
- Каждое сочетание обрабатывается только 1 раз.

2) Оптимален по затратам для $k \le n/2$.

Обратите внимание, что рассмотренный алгоритм оптимален по затратам только при $k \le n/2$. При $k > n/2$ этот алгоритм не оптимален по затратам, т.е. генерирует и обрабатывает «лишние» варианты. Однако его легко

модифицировать и для случая $k > n/2$. Достаточно применить этот алгоритм для генерации всех сочетаний из n по $(n-k)$, и для каждого сгенерированного сочетания обрабатывать (проверять на допустимость и оптимальность) не само это сочетание, а его дополнение.

2.5 Перестановки

Определение 7. Перестановкой степени n называется взаимно однозначное отображение множества $\{1, \ldots, n\}$ в себя [4].

С алгоритмической точки зрения, перестановка задаётся массивом размера n, в котором присутствуют все числа от 1 до n ровно по одному разу. Поэтому перебор перестановок естественным образом применяется для решения задачи коммивояжёра.

Алгоритм перебора всех перестановок (степени n) приведён ниже (Рисунок 11). Идея алгоритма:

- Храним текущую перестановку в массиве S[1..n].
- Инициализация: для всех i устанавливаем: $A[i] := i$.
- Для всех k последовательно переставляем $A[k]$ с элементами в позициях $1, \ldots, k-1$.

```
ProcessPermutations(S,k):
{
    if k = 1 then Process(S)
    else
        ProcessPermutations(S, k-1);
        for i = k-1 downto 1 do
            Поменять S[k] и S[i];
            ProcessPermutations(S, i-1);
            Поменять S[k] и S[i];
}
```

Рисунок 11

Дерево перебора вариантов представлено ниже (Рисунок 12).

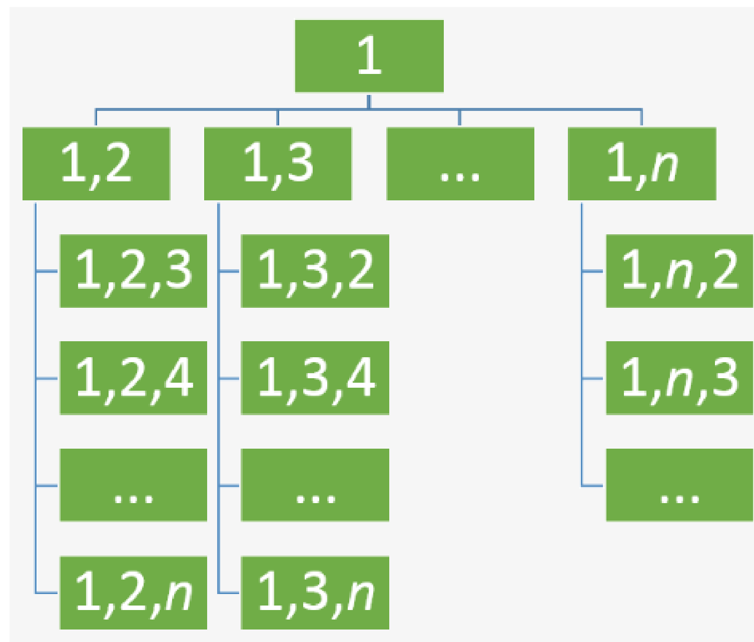

Рисунок 12

Решение задачи коммивояжёра (гамильтонов цикл на графе) ищется в виде перестановок вершин графа. Поскольку в цикле нет выделенной начальной вершины, можно считать, что на первом месте в перестановке всегда находится некоторая фиксированная вершина — например, v_1. Эту вершину можно не указывать в перестановке, и в качестве решений всегда рассматривать только перестановки (n–1) вершин (все кроме v_1). Поэтому сложность данного переборного алгоритма равна $O((n-1)!)$.

Ранее (параграф 2.3.2) мы рассматривали другой переборный алгоритм для этой же задачи, основанный на переборе всех строк в заданном алфавите. Тот алгоритм имеет сложность $O(d^n)$. Какой алгоритм является более быстрым? Воспользовавшись формулой Стирлинга для оценки факториала, мы получим, что это зависит от характеристик графа. Для разреженных графов [5], для которых выполняется условие $d < n/e$ (e — основание натурального логарифма) более быстрым является алгоритм, основанный на переборе всех строк в заданном

алфавите (параграф 2.3.2). Для плотных графов ($d > n/e$) более эффективен алгоритм перебора перестановок, рассмотренный в данном параграфе.

3 Метод ветвей и границ

3.1 Идея метода ветвей и границ

В предыдущих параграфах рассмотрены переборные алгоритмы решения вычислительно сложных задач, с примерами применения к нескольким «модельным» задачам. Общей особенностью этих алгоритмов является необходимость перебора всех допустимых решений, для получения оптимального (минимального или максимального) решения. Поскольку речь идёт об NP-трудных задачах, полностью избежать полного перебора не получится — любой алгоритм, получающий гарантированно точное решение такой задачи, может потребовать для своей работы экспоненциального времени на полный перебор всех вариантов. Но при этом речь идёт об оценке *в худшем случае*, т.е. при решении конкретного экземпляра, в принципе, можно затратить и гораздо меньше времени. Одним из универсальных методов, позволяющих достичь такого ускорения, является ***метод ветвей и границ***. Идея его в следующем:

1) Процесс перебора вариантов представляется в виде виртуального дерева перебора. Примеры таких деревьев приведены ранее. Листья такого дерева представляют собой потенциальные решения задачи, а внутренние вершины (вершины, не являющиеся листьями) представляют множества таких решений, объединённые какой-то общей характеристикой. В качестве общей характеристики выступает, как правило, либо часть потенциального решения (например, уже построенная часть гамильтонова цикла), либо ограничение на включение или невключение элементов (списки рёбер, входящих и не входящих в цикл из данного множества). Такое множество потенциальных решений называется ***ветвью***.

2) Находясь во внутренних вершинах дерева перебора, алгоритм вычисляет оценки (***границы***) стоимости решений, входящих в соответствующую ветвь. Для задач минимизации вычисляется оценка снизу, для максимизации — сверху.
3) Если вычисленная оценка хуже, чем стоимость уже найденного рекордного значения, то алгоритм ***отсекает*** текущую ветвь, т.е. исключает из перебора все входящие в неё потенциальные решения.

Операция отсечения является ключевой для метода ветвей и границ, поскольку позволяет, при правильном выборе способа ветвления и вычисления границ значительно сокращать количество обрабатываемых вариантов.

Рассмотрим применение метода ветвей и границ на примере задачи коммивояжёра.

3.2 Метод ветвей и границ для задачи коммивояжёра

Существует несколько вариантов применения метода ветвей и границ для задачи коммивояжёра. Два наиболее популярных варианта основаны на одном из двух альтернативных представлений цикла:

1. виде последовательности вершин,
2. в виде последовательности (множества) дуг.

Поскольку цикл должен быть гамильтоновым, в первом варианте цикл задаётся перестановкой всех вершин. Во втором варианте цикл представляется в виде подмножества дуг, содержащего ровно n дуг и удовлетворяющего дополнительным условиям (подробно описано ниже).

Оба подхода являются популярными, каждый имеет свои плюсы и минусы.

3.2.1 Представление цикла в виде перестановки вершин

В каждый момент времени текущее множество задач определяется частичной перестановкой — например, первыми k элементами перестановки: $\pi_1, \pi_2, \ldots, \pi_k$.

При ветвлении формируются (n–k) ветвей, каждая получается добавлением к имеющимся k элементам нового, (k+1)-го, не совпадающего с уже имеющимися. Оценка стоимости решения для выбранной ветви может быть получена, например, сложением следующих величин:

- Сумма весов всех рёбер, по которым проходит уже построенный участок пути: $w(\pi_1, \pi_2) + w(\pi_2, \pi_3) +\ldots+ w(\pi_{k-1}, \pi_k)$.
- Величины $\frac{1}{2}\left(w(e'_j) + w(e''_j)\right)$ для всех вершин v_j, по которым путь ещё не проходит. Здесь e'_j и e''_j — два ребра с максимальным весом, инцидентные вершине v_j и не ведущие в ранее пройденные вершины.

3.2.2 Представление цикла в виде подмножества дуг

Разделение множества потенциальных решений (гамильтоновых циклов) на два подмножества: содержащие выбранное ребро e_j, и не содержащие этого ребра. В каждый момент времени текущая задача определяется двумя подмножествами рёбер: E^+ и E^-. То есть текущая задача рассматривает множество циклов, проходящих по рёбрам из E^+ и не проходящих по рёбрам из E^-. Оценка снизу для веса потенциальных решений из заданной ветви может быть построена аналогично первому варианту алгоритма. Если при добавлении ребра e_j к множеству E^+ образуется запрещённый подграф (например, вершина, инцидентная более чем двум рёбрам из E^+, или цикл, включающий не все вершины графа), то ветвь, соответствующая подмножествам ($E^+ \cup \{ e_j \}$, E^-) исключается из дальнейшего рассмотрения («отсекается») даже без вычисления границы.

Литература

1. Адигеев М.Г. Введение в теорию сложности. Метод. указания. — Ростов-на-Дону: Изд-во Ростовского гос. ун-та, 2004 г. — 35 с.
2. Ахо А., Хопкрофт Дж., Ульман Дж. Постороение и анализ вычислительных алгоритмов. — М.: Мир, 1979.
3. Кормен Т.Х., Лейзерсон Ч.И., Ривест Р.Л. Алгоритмы: построение и анализ. — М.: Бином, 2004. — 960 с.
4. Ерусалимский Я.М. Дискретная математика. Теория и практикум. — М: Лань, 2018. – 476 с.
5. Харари Ф. Теория графов. — М.: «Мир», 1973. — 301 с.