МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
имени М.В.ЛОМОНОСОВА
факультет Вычислительной Математики и Кибернетики

Е. Корныхин

Задачи по курсу «Методы оптимизации»

Москва
2005

Оглавление

Глава 1

Введение в теорию сложности

Определение (ЗАДАЧА КОММИВОЯЖЕРА в форме распознавания свойства)**.** *[2, стр.35] Заданы конечное множество $C = \{c_1, c_2, \ldots, c_m\}$ "городов"мощности m, "расстояние"$d(c_i, c_j) \in \mathbb{N}$ для каждой пары различных городов $c_i, c_j \in C$ и граница $B \in \mathbb{N}$. Существует ли "маршрут", проходящий через все города C, длина которого не превосходит B? Другими словами, существует ли последовательность $\langle c_{\pi(1)}, c_{\pi(2)}, \ldots, c_{\pi(m)} \rangle$ элементов C такая, что*

$$\sum_{i=1}^{m-1} d(c_{\pi(i)}, c_{\pi(i+1)}) + d(c_{\pi(m)}, c_{\pi(1)}) \leqslant B \qquad ?$$

Определение (ВЫПОЛНИМОСТЬ)**.** *По данной булевой формуле, представленной в виде КНФ, выяснить, выполнима ли она, т.е. существует ли набор переменных, от которых зависит эта КНФ, на которых она обращается в истину [3, с.357].*

Определение (N-ВЫПОЛНИМОСТЬ)**.** *По данной булевой формуле, представленной в виде КНФ, в которой каждый дизъюнкт имеет длину N, выяснить, выполнима ли она, т.е. существует ли набор переменных, от которых зависит эта КНФ, на которых она обращается в истину [3, с.357].*

1.1 задача №1

Постановка задачи. *Предложить неизбыточную кодировку задачи коммивояжера (задачи КМ); оценить длину входа.*

Решение. Будем использовать десятичное кодирование натуральных чисел, а для их разделения в слове применять специальный символ ($\diamondsuit$). Итак,

пусть конечный алфавит для кодировки будет таким:

$$\Sigma = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, \diamondsuit\}.$$

Все данные в задаче расстояния поместим в матрицу так, чтобы на пересечении i-й строки и j-го столбца стоял $d(c_i, c_j)$. Такую матрицу будем представлять по строкам: сначала первая строка, затем вторая, третья и т.д., и, наконец, последняя строка. При кодировании этой матрицы для разделения строк будем использовать тот специальный символ-разделитель, который присутствует в алфавите кодирования.

Введем следующую кодировку e: сначала идет запись m, затем символ-разделитель, затем - запись B, символ-разделитель, запись матрицы $D = ||d(c_i, c_j)||_{1 \leqslant i,j \leqslant m}$. Т.к. D - симметричная матрица с нулевой главной диагональю, то для неизбыточности кодировки e будем кодировать только часть матрицы D, располагающаяся выше главной диагонали без самой диагонали так, как описано выше (по строкам), вставляя между элементами части такой матрицы уже известный нам символ-разделитель. Например, кодировка следующей индивидуальной задачи: $m = 3$, $B = 11$, $D = \begin{pmatrix} 0 & 1 & 5 \\ 1 & 0 & 3 \\ 5 & 3 & 0 \end{pmatrix}$ будет такой:

$$3 \diamondsuit 1\ 1 \diamondsuit 1 \diamondsuit 5 \diamondsuit 3$$

Исследуем некоторые свойства такой кодировки:

а) эта кодировка однозначна, т.к. по данной строке легко выделить сначала m, затем B и, наконец, по-элементно матрицу D.

б) очевидно, что и кодировка, и её декодировка полиномиально вычислимы (алгоритм кодировки просто состоит из делений по модулям - степеням десяти).

Оценим длину кодировки. Для начала заметим, что для представления натурального числа n в нашей записи потребуется $\lfloor \log_{10} n \rfloor + 1$ символов

алфавита Σ[1] (подчеркну, что n - натуральное, т.е. $n \geqslant 1$, а это значит, что $\log_{10} n$ определен и неотрицателен, а значит и длина записи n определена и натуральна). Тогда длина закодированной индивидуальной задачи КМ I может быть вычислена по следующей формуле: $|e(I)| = \lfloor \log_{10} m \rfloor + 1 + \lfloor \log_{10} B \rfloor + 1 + \sum\limits_{1 \leqslant i < j \leqslant m} (\lfloor \log_{10} D_{ij} \rfloor + 1) + 1 + 1 + (m-1+m-2+\ldots+1) - 1 \leqslant m^2 - m + 2 + \lfloor \log_{10} m \rfloor + \lfloor \log_{10} B \rfloor + \frac{m(m-1)}{2} \max\limits_{1 \leqslant i < j \leqslant m} \lfloor \log_{10} D_{ij} \rfloor$

□

1.2 задача № 2

Постановка задачи. *Предложить алгоритм проверки на простоту числа; найти его временную сложность.*

Определение (линейная двоичная кодировка)**.** *Назовем линейной двоичной кодировкой порядка n такое отображение $\mathbb{N}^n$ в Σ^* (где $\Sigma = \{0, 1, \diamondsuit\}$ - алфавит кодировки), переводящее числа $(x_1, x_2, \ldots, x_n)$ в слово $\sigma = \xi_{11}\xi_{12}\ldots\xi_{1l_1}\diamondsuit\xi_{21}\xi_{22}\ldots\xi_{2l_2}\diamondsuit\ldots\diamondsuit\xi_{n1}\xi_{n2}\ldots\xi_{nl_n}$, где $\xi_{i1}\xi_{i2}\ldots\xi_{il_i}$ - двоичная запись числа x_i.*

Например, если e - линейная двоичная кодировка порядка 4, то $e(3, 5, 1, 6) =$ 1 1 $\diamondsuit$ 1 0 1 $\diamondsuit$ 1 $\diamondsuit$ 1 1 0.

Лемма 1 (о сложности нахождения остатка)**.** *Существует алгоритм* `BMod`*, решающий массовую задачу нахождения остатка от деления одного натурального числа на другое с линейной двоичной кодировкой порядка 2, имеющий временную сложность $T_{\texttt{BMod}}(n) = O(n^2)$.*

Доказательство. Рассмотрим следующий алгоритм нахождения остатка от деления одного числа на другое:

```
function BMod(x, y: integer): integer;
```

[1] так как при увеличении n в 10 раз число десятичных разрядов в записи n увеличивается на 1

```
var k, i: integer;
begin
    k := 0;
    while k + |y| <= |x| do
    begin
        i := subint(x, |y|, k);
        if i >= y then
            minus(i, j);
        k := k + 1;
    end;
    return x;
end;
```

Поясним используемые в `BMod` функции.

|y| длина слова, отвечающего переменной y на ленте Машины Тьюринга.

i := subint(x, |y|, k) после этой функции число i будет идти в слове x, начиная с символа с номером k; при этом i будет иметь длину $|y|$.

minus(i, j) отнять от числа i число j, причем результат записать на место i.

Поясним работу функции `BMod` на следующем примере: пусть $x = 15_{10} = 1111_2$, $y = 2_{10} = 10_2$ (индекс указывает основание системы счисления, в которой записано значение переменной).

$$
\begin{array}{lll}
k = 0 & k = 1 & k = 2 \\
x = \overbrace{1\,1}^{i}\;1\;1 & x = 0\overbrace{1\,1}^{i}\;1 & x = 0\;0\overbrace{1\,1}^{i} \\
- & - & - \\
y = \;1\;0 & y = \quad 1\;0 & y = \quad\;\; 1\;0 \\
\hline
x = \;0\;1\;1\;1 & x = \;0\;0\;1\;1 & x = \;0\;0\;0\;1
\end{array}
$$

В качестве ответа принимаем полученное значение x (т.е. 1, что верно: $15 \mod 2 = 1$).

Перейдем к вопросу о сложности данной функции. Для начала заметим, что переменная k играет роль *положения головки*, поэтому на работе с k можно будет сэкономить по сложности. Проверка в цикле $k+|y| <= |x|$ - это проверка на невыход k за пределы установленных границ: от 0 до $|x| - |y|$. Достаточно поставить ограничитель и проверять, чтобы k не перешло этот ограничитель. Поэтому на работу с k надо $O(|x| - |y|)$ шагов Машины Тьюринга. На выделение подслова в слове x с присвоением подслову имени i вообще не надо тратить время Машины Тьюринга, потому как выделение i просто влияет на то, где будет производиться вычитание из слова x слова y. На проверку $i \geqslant y$ надо $O(|y|)$ шагов. На вычитание слова y из слова x тоже достаточно $O(|y|)$ шагов Машины Тьюринга. И, наконец, ещё один шаг на сдвиг головки, отвечающий оператору $k := k + 1$.

В результате получим, что время работы алгоритма `BMod` будет равно $t_{\texttt{BMod}}\,(e(x,y)) \leqslant \sum\limits_{k=0}^{|x|-|y|} (2|y|+1) = (2|y|+1)(|x|-|y|+1)$.

Обозначим $|x| = \alpha$ и $|y| = \beta$. Тогда $T_{\texttt{BMod}}\,(n) \leqslant \max\limits_{\substack{\alpha+\beta+1\leqslant n\\ \alpha,\beta\geqslant 1}} (2\beta+1)(\alpha-\beta+1) = \max\limits_{\substack{1\leqslant\alpha\leqslant n-1-\beta\\ \beta\geqslant 1}} (2\beta+1)(\alpha-\beta+1) \leqslant \max\limits_{\substack{1\leqslant\alpha\leqslant n-1-\beta\\ \beta\geqslant 1}} (2\beta+1)(n-\beta) = \max\limits_{1\leqslant\beta\leqslant n-2} (2\beta+1)(n-\beta) = (2\beta+1)(n-\beta)\big|_{\beta=\frac{n-(-1/2)}{2}=\frac{2n+1}{4}} = \frac{(2n+3)(2n-1)}{8} = O(n^2)$.

В выводе использовано такое свойство квадратного трехчлена: если старший коэффициент отрицательный и существуют 2 вещественных корня, то максимум достигается в точке с абсциссой, равной полусумме корней. □

Решение задачи №2. Предложим очень простой алгоритм:

```
function is_prime(n: integer): boolean;
var m, k:integer;
begin
    m := 2;
```

```
    while m < n do
    begin
        k := BMod(n, m);
        if k = 0 then
            return false;
        inc(m);
    end;
    return true;
end;
```

По определению $T_A(n) = \max\limits_{\sigma\in\Sigma^*:|\sigma|\leqslant n} t_A(\sigma)$. По определению нашего алгоритма максимум сложности будет достигаться на простом числе, так как для него нужно наибольшее число итераций для доказательства того, что оно на самом деле простое. Пусть x - максимальное простое число, длина кодировки которого не больше n. Будем использовать двоичную кодировку e для натуральных чисел. Тогда длина записи x равна $\lfloor\log_2 x\rfloor + 1$. Значит, $\lfloor\log_2 x\rfloor + 1 \leqslant n \Leftrightarrow x \leqslant 2^{n-1}$ (это несложно сделать, используя определение $\lfloor\cdot\rfloor$).

Итак, $T_A(n) = t_A(e(x(n))) = \sum\limits_{i=2}^{x-1}(t_{i<x} + t_{k\ :=\ \texttt{BMod}\ (x,i)} + t_{k=0} + t_{inc(m)}) + t_{i:=2}$.

$t_{i<x} = O(n)$.

$t_{k\ :=\ \texttt{BMod}\ (x,i)} = O(n^2)$ (по лемме о сложности нахождения остатка).

$t_{k=0} = O(n)$.

$t_{inc(i)} = O(n)$.

$t_{i:=2} = 2$ (т.к. число «2» имеет в двоичной кодировке длину 2).

Тогда $T_A(n) = \sum\limits_{i=2}^{x-1}(n + O(n^2) + O(n)) + 2 = (x-2)O(n^2) = O(2^n)O(n^2) = O(2^n n^2)$.

□

Ответ. $T_A(n) = O(2^n n^2)$.

1.3 задача №3

Постановка задачи. *Доказать, что задача «Составные Числа»* $\in \mathbb{NP}$.

Доказательство (конструктивное). Поставим задачу «Составные Числа» (СЧ): задано натуральное число x. Составное ли оно, т.е. имеет ли оно натуральный неединичный и не равный x делитель?

Будем использовать линейную двоичную кодировку порядка 1 e для кодирования x.

По определению класса $\mathbb{NP}$: $\texttt{СЧ} \in \mathbb{NP} \Leftrightarrow \exists \mathcal{A} = \{A_s\}_s$ - недетерминированная Машина Тьюринга, решающая массовую задачу СЧ с линейной двоичной кодировкой порядка 1: $\exists p(n)$ - полином: $\hat{T}_{\mathcal{A}}(n) \leqslant p(n)$, где $\hat{T}_{\mathcal{A}}(n) = \max\limits_{\sigma:|\sigma|\leqslant n} \min\limits_{s:\sigma\in L_{A_s}} \{|s| + t_{A_s}(\sigma)\}$ - временная сложность алгоритма работы Машины Тьюринга $\mathcal{A}$.

Предложим такую недетерминированную Машину Тьюринга: $\mathcal{A}(\sigma) = \{A_s | A_s \equiv (\texttt{ВМod}(\sigma, s) == 0)\}_{\substack{s=e(y)\\ \sigma=e(x)\\ 2\leqslant y\leqslant x}}$. Из доказательства леммы о сложности нахождения остатка[2] следует, что $t_{A_s}(\sigma) = (m+|s|+1)(|\sigma|-|s|+1)+|\sigma|$ ($+|\sigma|$ идет на сравнение результата с нулем). Здесь m - число единиц в записи частого $e^{-1}(\sigma)$ и $e^{-1}(s)$. В этой формуле точно учтены только происходящие вызовы функции `minus`.

Положим $t_{\mathcal{A}}(\sigma) = \min\limits_{\substack{s:s=e(y)\\ \sigma=e(x)\\ 2\leqslant y\leqslant x-1\\ x \mod y=0\\ m=|e([\frac{x}{y}])|_{(1)}}} \{(m+|s|+1)(|\sigma|-|s|+1)+|\sigma|\}$. Т.к. $m \leqslant |\sigma|$,

[2] доказательство можно найти в решении домашнего задания №1

то $t_{\mathcal{A}}(\sigma) \leqslant \min\limits_{\substack{s:s=e(y)\\ \sigma=e(x)\\ 2\leqslant y\leqslant x-1\\ x \mod y=0}} \{(|\sigma|+1)^2-|s|^2+|\sigma|\} = (|\sigma|+1)^2+|\sigma|-(\max\limits_{\substack{s:s=e(y)\\ \sigma=e(x)\\ 2\leqslant y\leqslant x-1\\ x \mod y=0}} \{|s|\})^2 \leqslant$
$(|\sigma|+1)^2+|\sigma|-4$, т.к. $y \geqslant 2 \Rightarrow |s| \geqslant 2 \Rightarrow max|s| \geqslant 2$.

Итак, $t_{\mathcal{A}}(\sigma) \leqslant (|\sigma|+1)^2+|\sigma|-4 = |\sigma|^2+3|\sigma|-3$.

Осталось посчитать $\hat{T}_{\mathcal{A}}(n) \leqslant \max\limits_{\sigma:|\sigma|\leqslant n} (|\sigma|^2+3|\sigma|-3)$.

Так как квадратичный трехчлен $y(x) = x^2+3x-3$ возрастает на промежутке $[1,n]$ (старший коэффициент положительный и абсцисса вершины отрицательна), то $\max\limits_{\sigma:|\sigma|\leqslant n} (|\sigma|^2+3|\sigma|-3) = (|\sigma|^2+3|\sigma|-3)|_{|\sigma|=n} = n^2+3n-3$.

Таким образом, мы доказали, что $\hat{T}_{\mathcal{A}}(n) \leqslant n^2+3n-3$. Это и означает, что $\mathtt{СЧ} \in \mathbb{NP}$ с полиномом $p(n) = n^2+3n-3$.

□

1.4 задача №4

Постановка задачи. *Доказать полиномиальную сводимость задачи* 3-ВЫПОЛНИМОСТЬ *к задаче* ВЫПОЛНИМОСТЬ.

Доказательство (конструктивное). Рассмотрим задачу ВЫПОЛНИМОСТЬ. Будем преобразовывать каждый дизъюнкт, входящий в исходную КНФ, в конъюнкцию дизъюнктов длины 3. Такии образом, из КНФ будет получена 3-КНФ, обладающая тем свойством, что проекция выполняющего набор 3-КНФ на переменные, входящие в КНФ, будет выполняющим набором КНФ. И наоборот, для любого выполняющего набора КНФ найдется выполняющий набор 3-КНФ, проекция которого будет совпадать с выполняющий набором КНФ.

Преобразование дизъюнктов длины не меньшей 2 описано в лекциях.

Докажем следующее простое утверждение: для любых булевых x и y выполнено: $x \equiv (x \vee y)(x \vee \overline{y})$. Докажем его следующей последовательностью

эквивалентных преобразований: $(x \vee y)(x \vee \overline{y}) \equiv xx \vee xy \vee x\overline{y} \vee y\overline{y} \equiv x \vee x(y \vee \overline{y}) \vee 0 \equiv x \vee x1 \equiv x$.

Преобразование дизъюнктов длины 2 будем выполнять следующим образом: $y_1 \vee y_2 \equiv (y_1 \vee y_2 \vee u)(y_1 \vee y_2 \vee \overline{u})$, где u - новая булева переменная. Преобразование дизъюнктов длины 1: $x \equiv (x \vee y)(x \vee \overline{y}) \equiv (x \vee y \vee u)(x \vee y \vee \overline{u})(x \vee \overline{y} \vee u)(x \vee \overline{y} \vee \overline{u})$, где y, u - новые булевы переменные. □

Глава 2

Основы линейного программирования

Определение. *Основная задача линейного программирования (*ОЗЛП*): при начальных данных* $A \in \mathbb{R}_{n,m}$, $b \in \mathbb{R}^m$, $c \in \mathbb{R}^n$ *найти* $\eta \in \mathbb{R}^n$ *такой, что* $(c, \eta) = \max\limits_{Ax \leqslant b} (c, x)$, *где* $(x, y) = \sum\limits_{i=1}^{n} x_i y_i$.

Определение. *Каноническая задача линейного программирования (*КЗЛП*): при начальных данных* $A \in \mathbb{R}_{n,m}$, $b \in \mathbb{R}^m$, $c \in \mathbb{R}^n$ *найти* $\xi \in \mathbb{R}^n$ *такой, что* $(c, \xi) = \max\limits_{Ax=b,\ x \geqslant \overline{0}_n} (c, x)$, *где* $(x, y) = \sum\limits_{i=1}^{n} x_i y_i$.

2.1 задача №5

Постановка задачи. *Свести каноническую задачу линейного программирования (*КЗЛП*) к основной задаче линейного программирования (*ОЗЛП*).*

Решение (конструктивное)**.** Покажем сведение КЗЛП к ОЗЛП: $\max\limits_{Ax=b,\ x \geqslant \overline{0}_n} (c, x) = \max\limits_{Ax \leqslant b,\ Ax \geqslant b,\ x \geqslant \overline{0}_n} (c, x) = \max\limits_{Ax \leqslant b,\ (-A)x \leqslant -b,\ -x \leqslant \overline{0}_n} (c, x) = \max\limits_{\begin{pmatrix} A \\ -A \\ -E_n \end{pmatrix} x \leqslant \begin{pmatrix} b \\ -b \\ \overline{0}_n \end{pmatrix}} (c, x) = \max\limits_{\widetilde{A}x \leqslant \widetilde{b}} (c, x)$,

где $\widetilde{A} = \begin{pmatrix} A \\ -A \\ -E_n \end{pmatrix}$ (E_n - единичная матрица порядка n) и $\widetilde{b} = \begin{pmatrix} b \\ -b \\ \overline{0}_n \end{pmatrix}$. $\square$

2.2 задача №6

Постановка задачи. *Доказать, что* $\mathcal{L} \geqslant O(\ln n\triangle)$, *где* $\mathcal{L}$ *- длина входа* ОзЛП D *(входом* ОзЛП *считается 3 матрицы:* $A \in \mathbb{R}_{m,n}$, $b \in \mathbb{R}^m$, $c \in \mathbb{R}^n$*),* $a\ \triangle = \triangle(A) = \max\limits_{A' \subseteq A} |\det A'|$.

Доказательство (от противного). Первым делом хочу заметить, что условие задачи допускает несколько интерпретаций (причина этого в том, что символ O нельзя использовать для сравнения на "больше-меньше").

Первая интерпретация: доказать, что $\mathcal{L} \geqslant f(n)$ для некоторой функции $f(n) = O(\ln n\triangle)$, т.е. по определению O существует такая константа (для n) $C > 0$ и номер N такие, что $f(n) \leqslant C \ln n\triangle$ для всех $n \geqslant N$. Но я не думаю, что имелось именно это, потому как доказывается слишком просто: в качестве f можно взять константу 0.

Вторая интерпретация: доказать, что $\mathcal{L} = \Omega(\ln n\triangle)$, т.е. по определению Ω найдется такая константа (для n) $C > 0$ и номер N такие, что $\mathcal{L} \geqslant C \ln n\triangle$. Но тогда противное утверждение (о том, что в общем случае такого конечного N может не найтись и $\mathcal{L} = o(\ln n\triangle)$ будет выполнено на целой последовательности различных задач $\mathcal{D}_n$) будет иметь такое подтверждение. Возьмем кодировку, которая задачи со входом D_n (где n - есть степень двойки, а сама D_n, как матрица, состоит из одних единиц) переводила в строку, являющуюся двоичной записью показателя степени в n, а входы всех остальных задач переводила каким-то определенным, но не регламентируемым здесь, способом так, чтобы не нарушались свойства однозначности, полиномиальной кодируемости и декодируемости и неизбыточности. Тогда в качестве $\mathcal{D}_n$ возьмем ту самую последовательность задач, размерность которых есть степень двойки. Тогда задачи $\mathcal{D}_n$ будут кодироваться словами длиной $\mathcal{L} = O(\log\log n)$, т.к. размерность задачи n в данном случае равна 2^{n_1}, а нужно получить двоичную запись числа $n_1 = \log n$, что добавляет ещё один логарифм. Итак, вторая интерпретация привела нас к контрпримеру к своей формулировке.

И, наконец, ТРЕТЬЯ ИНТЕРПРЕТАЦИЯ: доказать, что $\mathcal{L} \gtrsim \ln n\triangle$, т.е. всегда порядок роста $\mathcal{L}$ не может быть ниже, чем $\ln n\triangle$. Вот это утверждение попытаемся доказать от противного. Предположим, что нашлась такая кодировка e', что $|e'(D_n)|$ имеет меньший порядок роста, чем $\ln n\triangle$, т.е. что

$$|e'(D_n)| = o(\ln n\triangle)$$

Докажем тогда, что такая кодировка не будет однозначной (слишком она умудряется ужать D). Предположим противное, т.е. что кодировка e' однозначна. Рассмотрим задачи $\mathcal{D}_n$ порядков n, матрицы которых содержат только единицы. Пусть $\mathcal{A}_n$ - матрица коэффициентов задач $\mathcal{D}_n$. Тогда $\triangle(\mathcal{A}_n) = 1$. Отсюда получаем, что

$$|e'(\mathcal{D}_n)| = o(\ln n)$$

Таким образом, кодировка e' кодирует единичную матрицу порядка n словом длины $o(\ln n)$. Покажем, что такой кодировки существовать не может.

Пусть $F(n) = \max\limits_{1\leqslant k\leqslant n} |e'(\mathcal{D}_k)|$. Так как $|e'(\mathcal{D}_k)| = o(\ln k)$, то $F(n) = o(\ln n)$.

Так как e' однозначна и $\{\mathcal{D}_k\}_{k=1}^n$ различны, то различны будут и $\{e'(\mathcal{D}_k)\}_{k=1}^n$ и их ровно n штук.

Найдем число кодировок единичных матриц порядка $\leqslant n$ словами из конечного алфавита Σ. Их не больше, чем

$$\sum_{k=1}^{F(n)} |\Sigma|^k = \frac{|\Sigma|(|\Sigma|^{F(n)} - 1)}{|\Sigma| - 1} \sim |\Sigma|^{F(n)} = |\Sigma|^{o(\ln n)}$$

(это число слов длины не большей, чем $F(n)$ в алфавите Σ). Докажем, что $|\Sigma|^{o(\ln n)} = o(n)$.

Пусть $f(n) = o(\ln n)$. Тогда по определению символа o $\lim\limits_{n\to\infty} \frac{f(n)}{\ln n} = 0$. Тогда $\lim\limits_{n\to\infty} \frac{|\Sigma|^{f(n)}}{n} = \lim\limits_{n\to\infty} \frac{e^{f(n)\ln|\Sigma|}}{n} = \lim\limits_{n\to\infty} e^{f(n)\ln|\Sigma| - \ln n} = \lim\limits_{n\to\infty} e^{\ln n\cdot(\ln|\Sigma|\frac{f(n)}{\ln n} - 1)} = \lim\limits_{n\to\infty} e^{\ln n(\ln|\Sigma|\cdot 0 - 1)} = \lim\limits_{n\to\infty} e^{-\ln n} = \lim\limits_{n\to\infty} \frac{1}{n} = 0.$

Итак, получили, что $n \leqslant o(n)$, что на самом деле неправда. Осталось вспомнить все допущения, которые мы делали и удостовериться, что все они приводят к противоречиям. Что и доказывает поставленную перед нами задачу в третьей формулировке. □

2.3 задача №6.1

Постановка задачи. *Доказать, что* $\mathcal{L} > O(\ln n \triangle)$*, где* $\mathcal{L}$ *-* ***битовая*** *длина входа* ОзЛП D.

Лемма 2. *Для любого целого неотрицательного числа* n *длина его двоичной записи числа* $\geqslant \log{}^{1}(n+1)$.

Доказательство. Пусть $n = 0$. Тогда длина двоичной записи числа n равна $1 = \log(0+1)$. То есть для $n = 0$ неравенство выполнено.

Пусть длина двоичной записи n равна k. Тогда $2^{k-1} \leqslant n < 2^k \Rightarrow n+1 \leqslant 2^k \Rightarrow \log(n+1) \leqslant k$. □

Лемма 3. *Для любых неотрицательных чисел* $x_1, x_2, \cdots, x_n \geqslant 0$ *выполнено неравенство* $x_1 + x_2 + \cdots + x_n \leqslant (x_1+1)(x_2+1)\cdots(x_n+1)$.

Доказательство. Раскрытием скобок, получаем такое равенство: $(x_1+1)(x_2+1)\cdots(x_n+1) = x_1 + x_2 + \cdots + x_n + A$, причем $A \geqslant 0$, т.к. есть сумма произведений неотрицательных чисел. □

Лемма 4. *Для любой квадратной матрицы* A *выполнено неравенство* $|\det A| \leqslant \prod_i \sum_j |a_{i,j}|$.

Доказательство. По определению определителя $|\det A| = |\sum (-1)^{a(x)} \prod a_{i_1} a_{i_2} \cdots a_{i_n}| \leqslant \sum |\prod a_{i_1} a_{i_2} \cdots a_{i_n}|$.

[1] далее логарифмы без основания - двоичные

Раскроем скобки в произведении сумм $\prod_i \sum_j |a_{i,j}|$. В полученной сумме будут все произведения из полученной только что оценки модуля определителя плюс ещё некоторая сумма произведений модулей элементов матрицы A. □

Доказательство (нашей задачи). Доказательство проведем для матриц D, состоящих из неотрицательных целых чисел. Для остальных чисел неравенство тоже будет доказано, т.к. длина двоичного представления отрицательного числа больше на единицу (знак «минус»), а рационального числа - на длину знаменателя.

$$L = \sum_{i,j}(blen(a_{i,j}) + 1) + \sum_i (blen(b_i) + 1) + \sum_j (blen(c_j) + 1) + blen(n)$$

($blen(x)$ - это длина двоичной записи x) (здесь использовано двоичное кодирование элементов матрицы + разделители + надо закодировать n, чтобы однозначно восстановить положение элементов матрицы).

По лемме 1 $L \geqslant \sum_{i,j} \log(2a_{i,j}+2) + \sum_i \log(2b_i+2) + \sum_j \log(2c_j+2) + \log(n) = \log(\prod(a_{i,j}+1)\prod(b_i+1)\prod(c_j+1)) + (n+1)(m+1) + \log n \geqslant \log\prod(a_{i,j}+1) + \log n$, т.к. $b_i \geqslant 0, c_j \geqslant 0$.

Пусть $A1$ - та квадратная подматрица матрицы A, на которой достигается максимум модуля определителя. Тогда по леммам 2 и 3 $\triangle = |\det A1| \leqslant \prod_i \sum_j |a_{i,j}| \leqslant \prod_i \prod_j (a_{i,j}+1)$, т.к. $a_{i,j} \geqslant 0$.

Таким образом $L \geqslant \log\prod(a_{i,j}+1) + \log n \geqslant \log\triangle + \log n = \log(n\triangle) = O(\ln(n\triangle))$. □

2.4 задача №7

Постановка задачи. *Доказать, что двойственная задача к двойственной задаче линейного программирования есть прямая задача линейного*

программирования.

Доказательство. Пусть дана следующая прямая задача линейного программирования в основной форме:

$$\text{найти } d = \max_{\substack{x\in\mathbb{R}^n \\ Ax\leqslant b}} (c,x) = (c,x^*) \tag{2.1}$$

где $A \in \mathbb{R}_{m,n}$, $b \in \mathbb{R}^m$, $c \in \mathbb{R}^n$. Здесь и далее элементы пространства $\mathbb{R}^k$ будем представлять *столбцами* из k вещественных чисел.

Тогда по определению двойственная задача для неё будет выглядеть так:

$$\text{найти } d^* = \min_{\substack{y\in\mathbb{R}^m \\ A^Ty=c \\ y\geqslant 0_m}} (b,y) = (b,y^*) \tag{2.2}$$

Приведем двойственную задачу к виду прямой: $\min\limits_{\substack{A^Ty=c \\ y\geqslant 0_m}} (b,y) = -\max\limits_{\substack{A^Ty\leqslant c \\ -A^Ty\leqslant -c \\ -y\leqslant 0_m}} (-b,y) =$

$-\max\limits_{\begin{pmatrix} A^T \\ -A^T \\ -E_m \end{pmatrix} y \leqslant \begin{pmatrix} c \\ -c \\ 0_m \end{pmatrix}} (-b,y).$

Построим двойственную задачу к полученной задаче:

$$-\min_{\substack{z\in\mathbb{R}^{2n+m} \\ z\geqslant 0_{2n+m} \\ (A|-A|-E_m)z=-b}} \left(\begin{pmatrix} c \\ -c \\ 0_m \end{pmatrix}, z\right) \tag{2.3}$$

Представим $z = \begin{pmatrix} \alpha \\ \beta \\ \gamma \end{pmatrix}$, где $\alpha, \beta \in \mathbb{R}^n$, $\gamma \in \mathbb{R}^m$.

Тогда задача (2.3) запишется так: $-\min\limits_{\substack{\begin{pmatrix} \alpha \\ \beta \\ \gamma \end{pmatrix} \geqslant \begin{pmatrix} 0_n \\ 0_n \\ 0_m \end{pmatrix} \\ (A|-A|-E_m)\begin{pmatrix} \alpha \\ \beta \\ \gamma \end{pmatrix} = -b}} \left(\begin{pmatrix} c \\ -c \\ 0_m \end{pmatrix}, \begin{pmatrix} \alpha \\ \beta \\ \gamma \end{pmatrix}\right) =$

$$\max_{\substack{\begin{pmatrix}\alpha\\\beta\\\gamma\end{pmatrix}\geqslant\begin{pmatrix}0_n\\0_n\\0_m\end{pmatrix}\\(A|-A|-E_m)\begin{pmatrix}\alpha\\\beta\\\gamma\end{pmatrix}=-b}} \left(\begin{pmatrix}-c\\c\\0_m\end{pmatrix},\begin{pmatrix}\alpha\\\beta\\\gamma\end{pmatrix}\right) = \max_{\substack{\alpha\geqslant 0_n\\\beta\geqslant 0_n\\\gamma\geqslant 0_m\\A\alpha-A\beta-\gamma=-b}} (c,\beta-\alpha) = \max_{\substack{\alpha\geqslant 0_n\\\beta\geqslant 0_n\\\gamma=b-A(\beta-\alpha)\geqslant 0_m}} (c,\beta-\alpha) =$$

$$\max_{\substack{\alpha\geqslant 0_n\\\beta\geqslant 0_n\\A(\beta-\alpha)\leqslant b}} (c,\beta-\alpha).$$

Итак, двойственная задача к двойственной (2.1) задаче имеет вид:

$$\max_{\substack{\alpha\geqslant 0_n\\\beta\geqslant 0_n\\A(\beta-\alpha)\leqslant b}} (c,\beta-\alpha) \tag{2.4}$$

Осталось доказать эквивалентность задач (2.1) и (2.4). Для этого докажем следующие два утверждения:

1. для любого x - решения задачи (2.1) конструктивным образом найдется (α,β) - решение задачи (2.4)
2. для любого (α,β) - решения задачи (2.4) конструктивным образом найдется x - решение задачи (2.1)

Доказательство второго утверждения очевидно: в качестве x достаточно взять разность $\beta-\alpha$.

Впрочем и доказать первое утверждение не составит для нас большого труда: достаточно научиться подбирать такие значения α и β, чтобы их разность $\beta-\alpha$ равнялась x, а сами эти вектора состояли бы из неотрицательных чисел. То есть в качестве α можно взять любой вектор из неотрицательных чисел, а тогда $\beta=x+\alpha$ и тоже должен состоять из неотрицательных чисел. Для удовлетворения этих условий можно, например, в качестве α взять $|x|$ (то есть вектор, состоящий из модулей элементов вектора x, тогда в качестве β - взять $x+|x|$. При этом β будет состоять из неотрицательных чисел (по определению модуля) и на такой паре (α,β) будет достигаться максимум скалярного произведения, так как фактически

максимизируемое скалярное произведение зависит от $\beta - \alpha$, а оно равно x, то есть такому вектору, на котором достигается максимум; а на тех парах (α, β), разность которых не равна x, максимум достигаться не будет, потому как не достигается максимум на их разности. □

2.5 задача №8

Постановка задачи. *Доказать, что основная задача линейного программирования (*ОзЛП*) эквивалентна решению некоторой системы линейных уравнений $Pz = q$, причем $z \geqslant 0$.*

Доказательство. В лекциях была доказана эквивалентность ОзЛП

$$\max_{\substack{x \in \mathbb{R}^n \\ Ax \leqslant b}} (c, x) = (c, x^*)$$

системе линейных уравнений и неравенств

$$\begin{cases} Ax^* \leqslant b \\ A^T y^* = c \\ y^* \geqslant \overline{0} \\ (b, y^*) = (c, x^*) \end{cases}$$

Эта система следует из теоремы о двойственности линейного программирования: x^* - это решение прямой задачи линейного программирования (это - первое неравенство), y^* - это решение двойственной ей задачи (второе равенство и третье неравенство) - на них должно совпадать значение оптимизируемых скалярных произведений (четвертое равенство). Разрешимость этой системы означает существование x^* - решения ОзЛП, т.е. её разрешимость - и наоборот. Далее для простоты обсуждения звездочки будут опущены.

Преобразуем эту систему уравнений-неравенств к виду системы уравнений, в которой все переменные должны принимать неотрицательные значения. Пусть $z = b - Ax \geqslant \overline{0}$. Пусть $x = \alpha - \beta$, $\alpha \geqslant \overline{0}$, $\beta \geqslant \overline{0}$. Например, в

качестве α можно взять вектор $x+|x|$, а в качестве β взять $|x|$ (вектор под модулем означает вектор из модулей компонент). Аналогично, $y = \gamma - \delta$, $\gamma \geqslant \overline{0}$, $\delta \geqslant \overline{0}$. В результате получаем такую систему уравнений-неравенств:

$$\begin{cases} A\alpha - A\beta + z = b \\ A^T\gamma - A^T\delta = c \\ c^T\alpha - c^T\beta - b^T\gamma + b^T\delta = \overline{0} \\ z, \alpha, \beta, \gamma, \delta \geqslant 0 \end{cases}$$

или в виде $Pz = q$, $z \geqslant 0$:

$$\begin{pmatrix} E_m & A & -A & 0_{m,m} & 0_{m,m} \\ 0_n & 0_{n,n} & 0_{n,n} & A^T & -A^T \\ 0 & c^T & -c^T & -b^T & b^T \end{pmatrix} \begin{pmatrix} z \\ \alpha \\ \beta \\ \gamma \\ \delta \end{pmatrix} = \begin{pmatrix} b \\ c \\ 0 \end{pmatrix}$$

$0_{m,m}$ - матрица $m \times m$ из нулей,
E_m - матрица $m \times m$, у которой на диагонали стоят единицы, а на остальных местах - нули,
$A \in \mathbb{R}_{m,n}$ - матрица вещественных чисел $m \times n$,
$b \in \mathbb{R}^m$, $c \in \mathbb{R}^n$ - вектора-столбцы вещественных чисел соответствующей размерности,
z - вещественное число (скаляр) □

Глава 3

Элементы математического программирования

3.1 задача №9

Постановка задачи. *Доказать* $G^o(x) \neq \varnothing \Rightarrow \overline{G^o(x)} = G(x)$ *для любой точки* $x \in X \subset \mathbb{R}^n$ *в задаче минимизации некоторой функции* $f(x)$ *на множестве* X*, где*
$X = \{x \in \mathbb{R}^n \mid \forall i = \overline{1,m}(g_i(x) \leqslant 0)\}$,
$G^o(x) = \{y \in \mathbb{R}^m \mid \forall i \in \mathcal{J}(x)((\nabla g_i(x), y) < 0)\}$,
$G(x) = \{y \in \mathbb{R}^m \mid \forall i \in \mathcal{J}(x)((\nabla g_i(x), y) \leqslant 0)\}$,
$\mathcal{J}(x) = \{i^* = \overline{1,m} \mid g_{i^*}(x) = 0\}$

Доказательство. 1. Докажем, что $G^o(x) \neq \varnothing \Rightarrow \overline{G^o(x)} \subseteq G(x)$, т.е. все предельные точки $G^o(x)$ лежат в $G(x)$. Выберем произвольно предельную точку множества $G^o(x)$. Пусть это будет некоторая точка y. Тогда по определению предельной точки, существует целая последовательность $y_n \in G^o(x)$, сходящаяся к y.

Заметим, что $y_n \in G^o(x) \subseteq G(x)$, т.е. $y_n \in G(x)$. Т.к. $G(x)$ - это пересечение замкнутых множеств (полупространств), а значит оно само замкнуто. А значит, из $y_n \in G(x)$ будет следовать, что $\lim\limits_{n\to\infty} y_n \in G(x)$, т.е $y \in G(x)$. Первая часть доказательства сделана.

2. Докажем, что $G^o(x) \neq \varnothing \Rightarrow \overline{G^o(x)} \supseteq G(x)$, т.е. для любого элемента $\xi \in G(x)$ найдется последовательность $\{\xi_n\} \subseteq G^o(x)$, сходящаяся к ξ.

Зафиксируем $x \in X$. Очевидно, что $G^o(x) \subseteq G(x)$.

Если $\xi \in G^o(x)$, то в качестве ξ_n достаточно взять $\xi_n = \xi$. И все требования будут соблюдены.

В противном случае заметим, что так как $G^o(x) \neq \varnothing$, то обязательно найдется точка $\xi^* \in G^o(x)$. Так как $G^o(x)$ - есть пересечение открытых полупространств, то $G^o(x)$ - открытый многогранник, а, значит, $G^o(x)$ - выпуклое множество. Тогда в качестве ξ_n возьмем такую последовательность точек, которая вся лежит на отрезке из ξ^* в ξ, причем вся лежит в $G^o(x)$. Для этого достаточно взять ξ_n так: $\xi_n = \frac{1}{n}\xi^* + (1 - \frac{1}{n})\xi$. Покажем формально, что $\xi_n \in G^o(x)$: для любого $i \in \mathcal{J}(x)$ $(\nabla g_i, \xi_n) = (\nabla g_i, \frac{1}{n}\xi^* + (1 - \frac{1}{n})\xi) = \frac{1}{n}(\nabla g_i, \xi^*) + (1 - \frac{1}{n})(\nabla g_i, \xi)$. Так как $\xi \in G(x)$, то $(\nabla g_i, \xi) \leqslant 0$, а, значит, $(\nabla g_i, \xi_n) \leqslant \frac{1}{n}(\nabla g_i, \xi^*) < 0$, так как $(\nabla g_i, \xi^*) < 0$, что следует из того, что $\xi^* \in G^o(x)$.

Вторая часть доказана, а вместе с ней и вся теорема. □

3.2 задача №10

Постановка задачи. *Доказать теорему двойственности в линейном программировании с использованием теоремы Куна-Таккера.*

Теорема (двойственность в линейном программировании)**.** *Прямая задача разрешима $\Leftrightarrow$ разрешима двойственная ей задача, и при этом результаты этих задач совпадают.*

Определение. *Прямая задача линейного программирования - найти*

$$d^* = \max_{\substack{x \in \mathbb{R}^n \\ Ax \leqslant b}} (c, x) = (c, x^*)$$

Определение. *Задача линейного программирования, двойственная зада-*

че из определения 3.2 - найти

$$d^{**} = \min_{\substack{y\in\mathbb{R}^m \\ A^T y = c \\ y \geqslant 0_m}} (b, y) = (b, y^*)$$

Теорема (Куна-Таккера)**.** *Пусть решается следующая задача математического программирования: ищется* $\min\limits_X f(x)$*, где* $X = \{x \in \mathbb{R}^n | \forall i = \overline{1,m}(g_i(x) \leqslant 0)\}$*. Тогда если* $f, g_i \in C^1(\mathbb{R}^n)$*, выпуклы, а* X *регулярно, то справедливо следующее:*

$x^* - $*точка* $\min f(x) \Leftrightarrow \exists \lambda \geqslant 0_m : \nabla_x L(x,\lambda) \mid_{x=x^*} = 0_n \ \&(g_i(x^*), \lambda) = 0, i = \overline{1,m}$

где $L(x,\lambda) = f(x) + \sum\limits_{i=1}^{m} \lambda_i g_i(x)$ *- функция Лагранжа.*

Лемма 5 (самодвойственность задачи линейного программирования)**.** *Пусть Пр - прямая задача линейного программирования. Будем далее обозначать её двойственную задачу приписыванием звездочки, т.е. Пр* - задача, двойственная задаче Пр. Тогда Пр** = Пр.*

Доказательство леммы 5. см. домашнюю работу №6, задачу №7. □

Лемма 6 (наследование разрешимости двойственной задачей)**.** *Пусть Пр - прямая задача линейного программирования. Тогда если Пр разрешима, то Пр* тоже разрешима, причем ответы в этих задачах совпадают.*

Доказательство леммы 6. Введем следующие соглашения:

1. Пусть $A \in \mathbb{R}_{m,n}$. Тогда a_i - i-я **строка** матрицы , а a^i - i-й **столбец** матрицы
2. Аргументами скалярного произведения могут быть только вектора-столбцы

Приведем прямую задачу линейного программирования Пр в обозначениях определения 3.2 к виду задачи математического программирования, пригодному для применения теоремы Куна-Таккера. Пусть $f(x) = (-c, x)$,

$g_i(x) = ((a_i)^T, x) - b_i$. Так как скалярное произведение - линейная функция, то $f(x)$ и $g(x)$ непрерывно дифференцируемы и выпуклы на $\mathbb{R}^n$.

Пусть Пр разрешима. Тогда по теореме Куна-Таккера, найдется вектор $\lambda \in \mathbb{R}^m$ из неотрицательных компонент такой, какой описан в теореме Куна-Таккера. Посчитаем функцию Лагранжа: $L(x, \lambda) = f(x) + \sum\limits_{i=1}^{m} \lambda_i g_i(x) = (-c, x) + \sum\limits_{i=1}^{m} \lambda_i^*((a_i)^T, x) - \sum\limits_{i=1}^{m} \lambda_i^* b_i = -(c, x) + (A^T\lambda^*, x) - (\lambda^*, b) = (A^T\lambda^* - c, x) - (\lambda^*, b)$. Так как $\nabla_x(c, x) = c$, то $\nabla_x L(x, \lambda^*) = A^T\lambda^* - c = 0_n$ (по теореме Куна-Таккера). И второе условие: $(g_i(x^*), \lambda^*) = \sum\limits_{i=1}^{m} \lambda_i^*(((a_i)^T, x^*) - b_i) = (A^T\lambda^*, x^*) - (b, \lambda^*) = (c, x^*) - (b, \lambda^*) = 0$ (используя только что доказанное свойство)

Итак, по причине разрешимости исходной задачи Пр (точка, на которой достигается решение, есть x^*) нашелся вектор $\lambda^* \in \mathbb{R}^m$ такой, что $\lambda^* \geqslant 0_m$, $A^T\lambda^* = c$ и $d^* = (c, x^*) = (b, \lambda^*)$.

Пусть $\alpha^* = |x^*| \geqslant 0_n$ (это вектор-столбец из модулей компонент вектора x^*), $\beta^* = x^* + |x^*| \geqslant 0_n$, $\gamma^* = b - Ax^* \geqslant 0_m$.

Пусть $\mu^* = \begin{pmatrix} \alpha^* \\ \beta^* \\ \gamma^* \end{pmatrix} \geqslant 0_{2n+m}$.

Пусть $y^* = \lambda^*$. Тогда $y^* \geqslant 0_m$, $A^T y^* = c$.

Пусть $\widetilde{f}(x) = (b, x)$. Пусть $\overline{\widetilde{g}}(x) = \begin{pmatrix} A^T \\ -A^T \\ -E_m \end{pmatrix} x + \begin{pmatrix} -c \\ c \\ -0_m \end{pmatrix}$, $x \in \mathbb{R}^{2n+m}$.

Составим функцию Лагранжа для такой задачи:

$$\min_{\substack{y \in \mathbb{R}^{2n+m} \\ \overline{\widetilde{g}}(y) \leqslant 0_{2n+m}}} \widetilde{f}(y) \tag{3.1}$$

$L(x, \mu) = \widetilde{f}(x) + \sum\limits_{i=1}^{2n+m} \mu_i \widetilde{g}_i(x) = (b, x) + \sum\limits_{i=1}^{n} (\alpha_i - \beta_i)((a^i, x) - c_i) - \sum\limits_{i=1}^{m} \gamma_i x_i = (b, x) + (A(\alpha - \beta), x) - (\alpha - \beta, c) - (\gamma, x) = (A(\alpha - \beta) + b - \gamma, x) - (\alpha - \beta, c)$

И посчитаем градиент по x: $LGr(\mu) = \nabla_x L(x, \mu) = A(\alpha - \beta) + b - \gamma$. Посчитаем $LGr(\mu^*) = A(\alpha^* - \beta^*) + b - \gamma^* = A(|x^*| - (x + |x^*|)) + b - (b - Ax^*) =$

0_{2n+m}.

Теперь посчитаем $(\overline{\widetilde{g}}(y^*), \mu^*) = (\begin{pmatrix} A^T y^* - c \\ -A^T y^* + c \\ -y^* \end{pmatrix}, \begin{pmatrix} \alpha^* \\ \beta^* \\ \gamma^* \end{pmatrix}) = (A^T y^*, \alpha^* - \beta^*) - (c, \alpha^* - \beta^*) - (y^*, \gamma^*) = (A(\alpha^* - \beta^*), y^*) - (c, \alpha^* - \beta^*) - (y^*, \gamma^*) = (A(\alpha^* - \beta^*) - \gamma^*, y^*) - (c, \alpha^* - \beta^*) = (A(\alpha^* - \beta^*) - (b + A(\alpha^* - \beta^*)), y^*) - (c, \alpha^* - \beta^*) = -(b, y^*) + (c, \beta^* - \alpha^*) = -(b, \lambda^*) + (c, x^* + |x^*| - |x^*|) = (c, x^*) - (b, \lambda^*) = 0.$

Итак, получили, что нашелся такой вектор $\mu^* \geqslant 0_{2n+m}$, что $\nabla_x L(x, \mu^*) = 0$ и $(\overline{\widetilde{g}}(y^*), \mu^*) = 0$. Значит, по теореме Куна-Таккера существует решение задачи (3.1), которое реализуется в точке y^*. Причем $d^* = (c, x^*) = (b, \lambda^*) = (b, y^*) = d^{**}$, т.е. есть совпадение результатов. Лемма доказана □

Доказательство теоремы. Пусть Пр - исходная задача линейного программирования. Тогда из лемм 6 и 5 следует, что если (Пр*)* = Пр разрешима, то Пр* - тоже разрешима. Но по той же лемме 5 Пр* - есть некоторая прямая задача. Назовём её ПДр. Применяя лемму 6 для ПДр, получаем: если ПДр разрешима, то ПДр* - тоже разрешима.

Объединяя последнее следствие с леммой 6 для Пр, получаем: Пр разрешима $\Leftrightarrow$ Пр* разрешима (в одну сторону - это лемма 6 для Пр, в другую - лемма 6 для ПДр, т.е. для Пр*), причем результаты решения задач совпадают. Теорема доказана □

Глава 4

Экзаменационные задачи

4.1 Построение двойственной задачи линейного программирования

Постановка задачи. *Дана прямая задача линейного программирования. Построить по ней двойственную задачу и проверить теорему двойственности линейного программирования.*

Решение. Покажем задачу на примере. Пусть дана такая задача линейного программирования:

$$\max_{x \geqslant 1}(2 - x)$$

Строим двойственную ей задачу: $\max\limits_{x \geqslant 1}(2 - x) = 2 + \max\limits_{-x \leqslant -1}(-x)$. $A = (-1)$, $b = (-1)$, $c = (-1)$. Тогда двойственная имеет вид: $2 + \min\limits_{\substack{(-1)^T y = -1 \\ y \geqslant 0}}(-y) = 2 + \min\limits_{y=1}(-y) = \min\limits_{y=1}(2 - y)$.

Проверим теорему двойственности. Очевидно, двойственная задача разрешима с результатом: $y^* = 1$, $d^{**} = 1$. Решим прямую задачу. В ней надо найти максимум убывающей функции. Значит, максимум будет достигаться на левой границе, т.е. при $x = 1$. Таким образом, $x^* = 1$, $d^* = 1$. Получили, что прямая и двойственная задача разрешимы одновременно, причем оптимумы совпадают: $d^* = d^{**} = 1$. Теорема двойственности выполнена.

Для самостоятельного решения:

1. $\max\limits_{2\leqslant x\leqslant 4}(3x+5)$.
2. $\min\limits_{x\geqslant 1}(2-x)$. Указание: $\min x = -\max(-x)$.
3. $\min\limits_{\substack{x\geqslant 1\\ x\leqslant 3}}(2-x)$.
4. $\max\limits_{\substack{2x-3y\geqslant 1\\ x\leqslant y\\ x+y\geqslant -4}}(x+y+2)$. Указание: для решения системы линейных неравенств использовать симплекс-метод
5. $\min\limits_{\substack{2x-3y\geqslant 1\\ x\leqslant y\\ x+y\geqslant -4}}(x+y+2)$.

Ответы: 17, ∞, -1, 0, -2.

4.2 Применение градиентного метода

Постановка задачи. *Записать градиентный метод решения задачи*

$$\min(x^2+y^2)$$

а) с постоянным шагом $\alpha = \frac{1}{4}$

б) наискорейшим спуском

Выписать несколько итераций с начальным приближением $x_1 = 1$, $y_1 = 1$.

Решение. Градиентный метод:

$$z_{t+1} = z_t - \alpha_t \cdot (\nabla f)(z_t)$$

$$f = (x^2+y^2),\ \nabla f = \begin{pmatrix} \frac{\partial}{\partial x}(x^2+y^2) \\ \\ \frac{\partial}{\partial y}(x^2+y^2) \end{pmatrix} = \begin{pmatrix} 2x \\ \\ 2y \end{pmatrix},\ z_t = \begin{pmatrix} x_t \\ \\ y_t \end{pmatrix}$$

Тогда $\begin{pmatrix} x_{t+1} \\ \\ y_{t+1} \end{pmatrix} = \begin{pmatrix} x_t \\ \\ y_t \end{pmatrix} - \alpha_t \begin{pmatrix} 2x_t \\ \\ 2y_t \end{pmatrix}$ или в виде системы:

$$\begin{cases} x_{t+1} = x_t - \alpha_t 2x_t = (1 - 2\alpha_t)x_t \\ y_{t+1} = y_t - \alpha_t 2y_t = (1 - 2\alpha_t)y_t \end{cases}$$

Для *постоянного шага*: $\alpha_t = \frac{1}{4}$. Тогда

$$\begin{cases} x_{t+1} = (1 - 2\frac{1}{4})x_t = \frac{1}{2}x_t \\ y_{t+1} = (1 - 2\frac{1}{4})y_t = \frac{1}{2}y_t \end{cases} \quad t = 1, 2, ...$$

Выпишем несколько итераций:

t	1	2	3	4
x_t	1	$\frac{1}{2}$	$\frac{1}{4}$	$\frac{1}{8}$
y_t	1	$\frac{1}{2}$	$\frac{1}{4}$	$\frac{1}{8}$

Видно, что сходимость происходит со скоростью геометрической прогрессии с $q = \frac{1}{2}$, что подтверждается соответствующим устверждением о скорости сходимости градиентного метода из лекций.

Наискорейший спуск: $\alpha_t : f(z_{t+1}) \to \min$.

$f(z_{t+1}) = f\begin{pmatrix} x_t(1 - 2\alpha_t) \\ y_t(1 - 2\alpha_t) \end{pmatrix} = (1 - 2\alpha_t)^2(x_t^2 + y_t^2)$. Получился квадратный трехчлен (α_t - переменная, x_t, y_t - константы). Он достигает своего минимума в точке $1 - 2\alpha_t^* = 0$, т.е. $\alpha_t^* = \frac{1}{2}$. Тогда градиентный метод с таким шагом запишется так:

$$\begin{cases} x_{t+1} = (1 - 2\frac{1}{2})x_t = 0 \\ y_{t+1} = (1 - 2\frac{1}{2})y_t = 0 \end{cases} \quad t = 1, 2, ...$$

Несколько замечаний. Получилось, что на всех итерациях, кроме первой, $z_t = \overline{0}$. Т.е. в данном случае метод сойдётся за одну итерацию. Интересно,

что наискорейший спуск получился с постоянным шагом! Не часто такое случается. Несколько итераций:

t	1	2	3	4
x_t	1	0	0	0
y_t	1	0	0	0

Для самостоятельного решения:

1. $\min(x^3 + y^3)$. $x_1 = y_1 = -1$. Постоянный шаг $\alpha_t = \frac{1}{3}$, $\alpha_t = \frac{2}{3}$, $\alpha_t = 1$. Что Вы можете сказать относительно сходимости метода при разных α_t? Насколько существенно начальное приближение x_1, y_1 - как оно влияет на сходимость метода? на скорость сходимости? Исследуйте наискорейший спуск.

2. $\min(x+y)$. $x_1 = y_1 = -1$. Постоянный шаг $\alpha_t = \frac{1}{3}$, $\alpha_t = \frac{2}{3}$, $\alpha_t = 1$. Что Вы можете сказать относительно сходимости метода при разных α_t? Насколько существенно начальное приближение x_1, y_1 - как оно влияет на сходимость метода? на скорость сходимости? Исследуйте наискорейший спуск. Предложите сходящийся метод решения этой задачи МП.

Ответы: сходится, сходится за одну итерацию, расходится; расходится

Литература

[1] Новикова Н.М. Основы оптимизации. Курс лекций в МГУ. 1999

[2] Гери М., Джонсон Д. Вычислительные машины и труднорешаемые задачи. — М.: Мир, 1982.

[3] Пападимитриу К.Х., Стайглиц К. Комбинаторная оптимизация: алгоритмы и сложность. — М.: Мир, 1984.