Учреждение образования
«Белорусский государственный педагогический университет
имени Максима Танка»

Факультет физико-математический
Кафедра информатики и методики преподавания информатики

СОГЛАСОВАНО
И.о. заведующего кафедрой
_____ С.В. Вабищевич
15 мая 2017 г.

СОГЛАСОВАНО
Декан факультета
_____ С.И. Василец
24 05 2017 г.
(рег. № 24-8-174 от 25.05 2017г.)

УЧЕБНО-МЕТОДИЧЕСКИЙ КОМПЛЕКС ПО УЧЕБНОЙ ДИСЦИПЛИНЕ

«Решение сложных и олимпиадных задач по программированию (дисциплина по выбору)»

Для специальности 1–02 05 01 Математика и информатика

Составители: В.М. Котов, доктор. физ.-мат. наук, профессор

Рассмотрено и утверждено
на заседании Совета БГПУ 28 июня 2017 г. протокол № 10

ОГЛАВЛЕНИЕ

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

Электронный учебно-методический комплекс (ЭУМК) по учебной дисциплине «Решение сложных олимпиадных задач по программированию» для специальности 1-02 05 01 Математика и информатика

Одной из задач школьного образования, связанной с применением компьютеров является подготовка способных учащихся к участию в олимпиадах по информатике и программированию. В условиях современного развития олимпиадного движения по информатике необходимо будущему педагогу иметь определенный банк знаний, включающий знания о классических структурах, данных стеках, очередях, линейных списках со связями, двоичных деревьях, графах, алгоритмах поиска и сортировки, рекуррентных соотношениях, комбинаторике, рекурсии, динамическому программированию и алгоритмах решения сложных и олимпиадных задач.

Цель учебной дисциплины по выбору – формирование у будущих преподавателей информатики профессиональных компетенций для построения эффективных алгоритмов решения сложных задач и решения оригинальных задач самого различного уровня, предлагавшихся на олимпиадах по программированию.

Основные задачи учебной дисциплины по выбору:

– разбор важнейших классов быстрых алгоритмов поиска и сортировки, их сравнительная характеристика;

– изучение динамических структур данных, применяемых в алгоритмах, приемов работы с ними;

– практическая реализация алгоритмов на компьютере;

– решение оригинальных олимпиадных задач.

ТЕОРЕТИЧЕСКИЙ РАЗДЕЛ

Раздел 1. ПРОЕКТИРОВАНИЕ И АНАЛИЗ АЛГОРИТМОВ

Тема 1.1 Проектирование и анализ алгоритмов.

ТРУДОЕМКОСТЬ АЛГОРИТМА

Прежде чем ввести такие понятия, как *размерность задачи* и *трудоемкость алгоритма*, нам будйг необходимо дать определение *информации.*

Определение 1.1. Пусть A — некоторая случайная величина, тогда количество информации, которого достаточно для знания того, что событие A произошло, определяется по следующему правилу:

$I(A) = - \log_b p(A)$,

где $p\{A)$ — вероятность, с которой событие A произойдет.

Если в качестве основания логарифма взять 2, то единицей измерения информации будет *бит.*

Впервые ввести меру информации нопытгшся Р. Хартли в 1928 В своих рассуждениях он исходил из интуитивной идеи о том, что сообщение, состоящее из n символов, должно нести в n раз больше информации, чем сообщение, состоящее из одного символа. Единственной функцией, которая удовлетворяет этому свойству, является логарифмическая фупкиия.

Пример 1.1. Пусть имеется 8 шаров с номерами от 1 до 8. Какого количества информации достаточно для установления номера выбранного случайным образом шара?

Решение. Так как любой шар может быть выбран с равной: вероятностью. то вероятность события A, заключающегося в определении номера выбранного шара, есть $P(A) = 1/8$ и $I(A) = —\log_2(1/8) = 3$. Заметим, что действительно трех битов достаточно, чтобы задать в памяти компьютера любое число от I до 8.

Пример 1.2. Предположим, что на вход поступает некоторое число A, которое может быть выбрано из интервала [1, • • •,»], при чем любой выбор является равновероятным. Какое количество битов необходимо для определения того, какое число мы выбради?

Решение. Из определения информации следует, что

$I(A) = [\log_2 m]$

есть то ка ш чес пи > битов, которое необходимо для определения того, какое конкретно число мы выбрали.

Здесь и далее символ! [ж] будет означать наименьшее целое, не меньшее ж.

Определение 1.2. Размерностью задачи I будем называть количество информации, которой достаточно для формального описания задачи.

Под формальным описанием задачимы понимаем такое ее описание, когда нас не интересует конкретный алгоритм решения задачи, а сама задача

описывается как некоторая функция, на входе которой поступают некоторые формальные параметры, задающие ее входные и выходные параметры.

В дальнейшем мы покажем, что если предположить, что размерность машинного слова достаточна для представления (разрушения неопределенности) любого числа, то размерность задачи будет равна числу ее исходных данных в ее формальном описании.

Пример 1.3. Предположим, что на вход некоторого алгоритма поступают Два числа: a и $Ь$. Определить размерность задачи.

Решение, Размерность задачи (т. е. количество информации, достаточной для задания этих чисел) равна $I = I_1 + h$, где $h = [\log_2»]$ и $h = |"\log_2 6]$.

Пример 1.4. Предположим, что на вход некоторого алгоритма поступает n чисел си,..., a_n. Определить размерность задачи.

Решение, Размерность задачи есть

$$I =]\Gamma\ [\log 0*1 \cdot$$
$$\Pi$$
$$r=1$$

Если предположить,, что размерности машинного слова достаточно для представления любого из чисел (т. е. maxj=i ,, $[\log_2 <$ const), то размерность задачи I можно оценить величиной n.

Определение 1.2. Время, затрачиваемое алгоритмом как функция от размерности задачи, называется временной сложностью алго ритма. Это функция, которая задаче размерности I ставит в соответствие время /(/), затрачиваемое алгоритмом для ее решения. Объем памяти, требуемый для реализации -алгоритма, как функция от размерности задачи, называется емкостной сложностью алгоритма. Поведение временной (емкостной) сложности при увеличении размерности задачи: до бесконечности называется асимптотической временной сложностью (асимптотической емкостной сложностью).

Если при данной размерности в качестве меры сложности берется наибольшая из сложностей (по всем входам этой размерности), то она называется сложностью В худшем случае. Если при данном размере в качестве меры Сложности берется средняя Сложность (по всем входам данной размерности), то она называется средней сложностью. Обычно среднюю сложность найти труднее, чем сложность в худшем случае. Однако не следует забывать, что алгоритм с наименьшей сложностью в худшем: случае не обязательно имеет лучшую скорость в среднем.

Если некоторый алгоритм обрабатывает входы длины n за время Cn^2, где C — некоторая константа, то говорят, что временная сложность этого алгоритм а есть $0(n^2)$ ("порядка n^2").

В общем случае будем говорить, что неотрицательная функция:

$T(n)$ есть $0(f(n))$, если существует такая константа C и n •. ч го $T(n) < Cf\{n)$ длД всех $n > na;$

$T(n)$ есть $f2(f(n))$, если существует такая константа C и $щ$, что $T(n) > Cf(n)$ для: всех $n > щ;$

$T(n)$ есть $0(f(n■)■■)■■$ тогда и только тогда, когда $W(n) = 0(f(n))$ и

mT(n) = H(f(n)).

Для асимптотик справедливы следующие правила.

Если *f(n)* есть *0(∂(n))* и *∂{n)* есть 0(/г(??.)), то *f(n)* есть *0(h(n)).*

Если *f(n)* есть *0(кд(n))* для некоторой константы *к* > 0, то /(») есть *0(∂(n)).*

Если *fi(n)* есть *0{∂1(n))* и *fain)* есть Qfjafs.)), то *fi(n)* + *f₂(n)* есть 0(max|pj{??.),*gain)))*.

Если *fi(n)* есть *0(gi(n))* и Jg|*n)* есть *Щдз$*n)),* то *fi(n)* ■ *fain)* есть *0{gi*(те) • *∂₂(n)).*

Разные алгоритмы имеют различную временную сложность. Для сравнения: эффективности алгоритмов часто применяется один простой подход, который заключается в различии между полиномиальными и экспоненциальными алгоритмами.

Метод рекурсивных деревьев

Данный метод заключается в том, что но рекуррентному уравнению итерационно строится древовидная структура, суммирование в которой осуществляется специальным образом.

На первой итерации формируется дерево следующего вида:

в корень дерева заносится свободный член исходного рекуррентного уравнения;

сыновьями этого корня являются рекуррентные функции правой части исходного соотношения.

На последующих итерациях для каждого из сыновей строится аналогичная древовидная структура.

Процесс построения древовидной структуры заканчивается, когда висячие вершины соответствуют начальным данным уравнения, при этом значения внутренних вершин дерева есть некоторые явные функции от размера задачи. Заметим, что висячие вершины построенной древовидной структуры не обязательно одинаково удалены от корня.

После построения дерева суммирование значений в вершинах производится следующим образом:

Определяются суммы значений для равноудаленных от корня вершин (эти вершины находятся на одном уровне).

Находится максимальная сумма по уровням.

Общая трудоемкость алгоритма ограничена сверху одним из следующих значений:

максимальной суммой, которая умножается на количество уровней;

суммой, полученной в результате суммирования сумм значений по уровням.

Заметим, что количество уровней для дерева есть (logn).

Пример 1.15. Предположим, что необходимо решить следующее рекуррентное уравнение:

$$T(n) = T(n/2) + T(n/2) + n$$

Решение. На рис. 1.1 изображена древовидная структура для рассматри ваем ого примера.

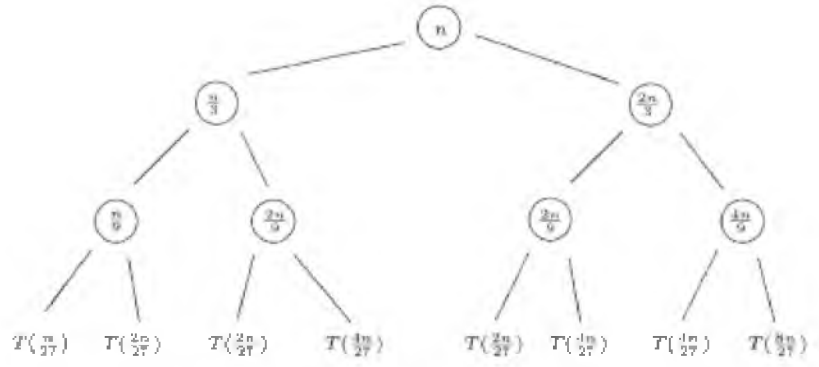


Рис. 1.1.

Оценим трудоемкость алгоритма. Так как сумма значений на каждом уровне не превосходит n и количество уровней есть $\log_a n = = C \log_2 n$, то $T(n) = T + T + {}^{n} \text{®} {}^{Cn \log_2 n}$-

Раздел 2. СТРАТЕГИИ РЕШЕНИЯ ЗАДАЧ

Тема 2.1 Принцип «Разделяй и властвуй».

Метод "разделяй и властвуй" заключается в следующем

Задача разбивается на независимые подзадачи (части), которые не пересекаются (под шдача — это та же задача, но меньшей размерности).

Каждая подзадача решается Отдельно.

Из отдельных решений подзадач строится решение исходной задачи.

Заметим., что если бы подзадачи пересекались, т. е. имели общие подиодзадачи, то метод "разделяй и властвуй" делал бы лишнюю работу, решая некоторые подиодзадачи по нескольку раз.

При разбиении задачи на подзадачи полезен принцип балансировки, который предполагает, что задача разбивается: на подзадачи приблизительно равных размерностей, т. е. идет поддержание равновесия. Обычно такая стратегия приводит к разделению исходной задачи пополам и обработке каждой из его частей тем же способом до тех пор, пока части не станут настолько малым и, что их можно будет обрабатывать непосредственно. Часто такой процесс приводит к логарифмическому множителю в формуле, описывающей трудоемкость алгоритма. Таким образом, в основе техники рассматриваемого метода лежит процедура разделения. Если разделение удается произвести без слишком больших затрат, то может быть построен эффективный алгоритм.

Пример 6.1. Отсортировать массив из n элементов, используя технику "разделяй и властвуй" (сортировку слиянием и сортировку Хоара).

Рассмотрим сначала *сортировку слиянием:* сортируемая последовательность элементов разбивается на два сегмента равных раз мерностей (принцип балансировки), элементы которых не пересекаются. Элементы каждого сегмента упорядочиваются, пос. к- чего происходит слияние отсортированных сегментов в один. Рекуррентное уравнение сортировки

слиянием имеет следующий вид:

$T\{n) = Ж\{n/Щ + Си.\ n > 1,$

$T(\ 1) = 1.$

По теореме о решении рекуррентного уравнения получаем, что трудоемкость алгоритма есть $0(nlogn)$.

Принцип "разделяй и властвуй" лежит в основе еще одного рассмотренного нами ранее эффективного алгоритма сортировки массива — *алгоритма, быстрой сортировки Хоара.* Согласно Этому принципу происходит разделение сортируемой части массива на два сегмента. Если при сортировке слиянием процедура разделения производилась простым делением сортируемой последовательности на два сегмента, то в сортировке Хоара она производится таким образом, что значения элементов из первого сегмента не больше каждого значения из второго сегмента. После выполнения процедуры разделения для упорядочения исходного массива остается лишь упорядочить сегменты. Напомним, что рекуррентное уравнение этого алгоритма в случае, когда предполагалось, что сегменты имеют одинаковое количество элементов (принцип балансировки), имеет следующий вид:

$T(n) = Cn, + 2T(\sim),$

и трудоемкость алгоритма есть 0(га log га). В худшем случае, когда процедура разделения разбивает элементы массива на сегменты размерности n и n — ₁ соответственно (не выполняется принцип балансировки), мы получали рекуррентное уравнение

$T(n) = Cn + T(n - 1) + Г(1),$

и трудоемкость алгоритма есть $0(n^2)$.

Таким образом, пример 6.1 явно иллюстрирует преимущества принципа балансировки в технике "разделяй и властвуй".

Пример 6.2. Найти максимальный и минимальный элементы массива двумя различными алгоритмами: исиользу'я технику "разделяй и властвуй" и без ее использования.

Рассмотрим сначала алгоритм решения поставленной задачи, который не основан на технике "разделяй и властвуй". Выбираем из первых двух элементов массива максимальный и минимальный элементы ₀ сравнение), затем, последовательно просматривая элементы массива, сравниваем каждый последующий из n — ₂ элементов с максимальным и минимальным элементами (₂(гг. — ₂) сравнения). Количество сравнений, которое выполняет алгоритм, есть

$T(n) = 2n - 3.$

Второй алгоритм решения поставленной: задачи основан на технике "разделяй и властвуй". Делим массив на две части (балансировка). Находим максимальный и минимальный элементы для каждой из частей; выбираем из максимальных элементов наибольший и из минимальных элементов наименьший. Рекуррентное уравнение данного алгоритма имеет следующий вид:

$T(n) = 2T\{n/2) + 2, n > 2, T(_2) = _1.$

Построим: точное решение данного уравнения, используя метод подстановки:

$$ШЩ = o^n \sim \text{“}$$

Таким образом, сложность алгоритма есть

$$\text{ЯР (IT.)} = -n - 2.$$

Заметим, что асимптотические сложности в первом и втором алгоритмах совпадают и равны $0(n)$, но применение техники во втором алгоритме дает лучшую константу.

Техника "разделяй и властвуй" является эффективной, когда задачу можно разбить на подзадачи за разумное время, а суммарный размер задач невелик. Однако, когда это не удается (применение техники приводит к экспоненциальным алгоритмам),, то применяется принцип "динамического программирования".

Тема 2.2 Динамическое программирование и градиентные алгоритмы

Суть данного метода заключается в следующем:

Задача погружается в семейство задач той же природы. Другими словами, разбивается на зависимые (могут пересекаться) подзадачи.

Каждая подзадача решается отдельно один раз. Оптимальные значения решений всех подзадач запоминаются, что позволяет не решать снова встречавшиеся ранее подзадачи.

Для исходной задачи строится возвратное соотношение, связывающее между собой оптимальные значения зависимых подзадач.

Стратегия метода динамического программирования — попытка свести рассматриваемую задачу к более простым задачам, тогда как стратегия предыдущей техники — "разделяй и властвуй".

Понятие "более простая задача" может в разных случаях пониматься: по-разному,. Задача может быть проще из-за тот, что опущены некоторые ограничения. Она может быть проще из-за того, что некоторые ограничения добавлены. Однако, как бы ни была изменена задача, если это изменение приводит к решению более простой задачи, то, возможно, удастся, опираясь на эту более простую, решить и исходную.

Основные отличия техники динамического программирования от техники "разделяй и властвуй" заключаются в следующем::

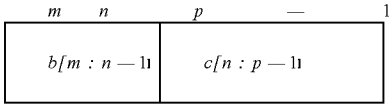
Разделяй и властвуй	Динамическое программирование
независимые	зависимые подзадачи
решения всех	решения всех подзадач

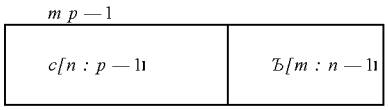
Данная техника находит свое применение, когда все подзадачи помещаются в память. Вычисление идет от малых подзадач к большим, и

ответы запоминаются в таблице, поэтому данный метод иногда называют *'табличным.* Одна из клеток таблицы и дает оитимаиь- ное решение исходной задачи.

Пример 6,3. Рассмотрим задачу перестановки сегментов. Пусть даны фиксированные целые переменные *m, n, p,* которые удовлетворяют условию: *m < n < p.* Дана часть массива *A[m : p* — 1], которая рассматривается как два сегмента — *Ь[m : n* — ₁] и *c[n : p* — IT,

Необходимо поменять местами два Сегмента массива, используя лишь постоянный (не зависящий от *:n%, n, p)* объем дополнительной памяти.





Более простой для данной задачи является задача перестановки двух Сегментов одинаковой длины.

Сведем нашу задачу перестановки сегментов неравной Длины к решению этой более простой задачи. Сначала предположим, что сегмент *b\m '■ n* — 1] больше, чем *c[n : p* — 1]. Предположим, что первый, больший сегмент состоит из двух сегментов — *b1* и *b2,* первый из которых имеет ту же длину, что и *c\n : p* — 1]. Тогда переставим местами дин сегмента — а и с.

Далее решение исходной задачи сводится к перестановке сегментов *Ъn* и *b\.*

Теперь предположим, что сегмент *b[m : n* — 1] меньше, чем *c[n : p* — 1]. Предположим, что второй, больший сегмент состоит из двух сегментов — q и *co,* второй из которых имеет ту же длину, что и *b[n". p* — 1]. Тогда переставим местами два сегмента — *b и* c_o.

Раздел 3. АЛГОРИТМЫ ЦЕЛОЧИСЛЕННОЙ АРИФМЕТИКИ

Тема 3.1 Алгоритмы целочисленной арифметики

В десятичной системе счисления, как и в любой другой позиционной системе счисления, натуральное число может быть записано в виде суммы разрядных слагаемых, т. е. в виде суммы единиц, десятков, сотен, тысяч и т. д. Так, например, число 6487 содержит 7 единиц, 8 десятков, 4 сотни и 6 тысяч и может быть записано в виде выражения;

6487 = 6-1000 + 4-100 + 8-10 + 7.

Цифры 6, 4, 8, 7 являются цифрами числа 6487, а 1000, 100, 10 — разрядные единицы. Каждая разрядная единица является степенью числа 10, поэтому представление числа в виде суммы разрядных слагаемых чаще записывают так:

$6487 = 6\text{-}10^3 + 4\text{-}10^2 + 8\text{-}10' + 7\text{-}10°$.

Вид числа в левой части равенства будем называть *обычным*

представлением числа в десятичной форме. При написании программ такие числа хранятся в стандартных числовых типах, например: ЦЕЛ, ВЕЩ. При работе с числами в обычном представлении мы не имеем возможности непосредственно обращаться к цифрам этого числа, однако иногда это бывает необходимо. Например, если требуется вычислить значение выражения, которое не помещается в стандартном числовом типе (100!), то число представляют в виде массива цифр. Такое представление будем называть *табличным* представлением числа.

Размещение цифр числа в массиве возможно двумя способами. Связано это с тем, что разряды чисел нумеруются справа налево, а число читается слева направо, поэтому цифры числа можно располагать в массиве, начиная с первой (старший разряд), а можно — начиная с последней (младший разряд). Порядок, при котором цифра старшего разряда является первым элементом массива, назовем *прямым. Обратным* вазоведа порядок, при котором первый элемент в массиве является цифрой младшего разряда.

Можно задавать число и как литерную величину, тогда мы получим доступ к каждой отдельной цифре, однако с ними, без преобразования в числовой тип, нельзя выполнять арифметические операции.

Рассмотрим способы преобразования обычного представления числа в табличное и наоборот.

Преобразование числа из обычного представления в табличное

Пусть задано число в обычном представлении, необходимо получить табличное представление данного числа.

Как видно из представления числа в виде суммы разрядных слагаемых, каждое слагаемое, кроме последнего, разделится на 10 без остатка. Поэтому остаток от деления числа на 10 будет равен последней цифре числа. Эту цифру мы помещаем в массив. Если мы размещаем в массиве цифры числа, начиная с последней, то первому элементу массива присваивается значение полученной цифры. Если размещаем цифры, начиная с первой, то полученную цифру помещаем в элемент массива с индексом, равным количеству цифр в числе (количество цифр необходимо подсчитать до того, как начнем получать сами цифры). Далее находим целую часть от деления исходного числа на 10 (или, говоря другими словами, отбрасываем последнюю цифру числа) и снова на ходим остаток от деления данного числа на 10, это и будет следующая цифра. Так продолжаем до тех пор, пока в числе не останется ни одной цифры.

Данный алгоритм размещает цифры числа в массив в обратном порядке. Для хранения цифр числа N будем использовать массив B, k — номер текущей цифры в массиве B.

```
к: =0
нц пока N > — 1
k: =k~f 1
B[k]:=mod(N, 10)      (3.10)
N: = div (N, 10) кц
```

Первые *k* элементов массива *B* содержат цифры числа *N,* записанные в обратном порядке.

Изменим алгоритм для получения цифр числа в прямом порядке.

к: =0 L: —N

нц пока L> = 1 L: = div (L, 10) k: = k+1 кц

m: =k (3.11)

нц пока N > = 1 B[k]:=mod(N, 10)

N: = div (N, 10) k: = k— 1 кц

Первый цикл ПОКА считает количество цифр в числе, второй — получает цифры числа. Переменная *L* служит для того, чтобы сохранить значение исходного числа, так как внутри цикла переменная изменяет свое значение. Если бы в первом цикле использовалась переменная *N,* то после выполнения цикла ее значение

равнялось бы 0 (почему?) и второй цикл не выполнился бы ни разу. В переменной *m* сохраняется количество цифр числа, что, вообще говоря, не всегда необходимо.

Тема 3.2 Реализация алгоритмов целочисленной арифметики.

Преобразование табличного представления числа в обычное

Решим обратную задачу, т. е. получим обычное представление числа из табличного.

Пусть некоторое число о задано своими цифрами:

$a = a_1 a_2 a_z .., 'a'_n.$

Черта сверху обозначает, что a_u a_2, a_3, ..., a_n — цифры числа *a (*a_1 — цифра старшего разряда). Запишем это число в виде суммы разрядных слагаемых:

$a — a_1 \cdot 10^{n-1} + a_2 \cdot 10^{n-2} + a_3 \cdot 10^{n-3} + ... + a_{n-1} \cdot 10 + a_n$. (4)

Для того чтобы получить значение числа *a,* необходимо вычислить значение выражения, стоящего в правой части равенства (4). Для этого проделаем следующие преобразования: вынесем 10 за скобки из тех слагаемых, для которых это возможно.

$a = a_1 \cdot 10^{n-1} + a_2 \cdot 10^{n-2} + a_3 \cdot 10^{n-3} + ... 10 + a_n =$

$= 10 (a_1 \cdot 10^{n-2} + a_2 \cdot 10^{n-3} + a_3 \cdot 10^{n-4} + ... + a_{n-1}) + a_n$

Затем с выражением в скобках проделаем то же самое, т. е. вынесем 10 за скобки из тех слагаемых, для которых это возможно. Продолжим действовать таким образом до тех пор, пока это будет возможно:

$a — a_1 \cdot 10^{n-1} + a_2 \cdot 10^{n-2} + a_3 \cdot 10^{n-3} + ... + a_{n-1} \cdot 10 + a_n =$ — $10(a_1 \cdot 10^{n-2} + a_2 \cdot 10^{n-3} + a_3 \cdot 10^{n-4} + ... + a_{n-1}) + a_n =$ $= 10(10(a_1 \cdot 10^{n-3} + a_2 \cdot 10^{n-4} + a_3 \cdot 10^{n-5} + ...) + a_{n-1}) + a_n =$

$= 10 (10 \cdot ... (10 (a_1 \cdot 10 + a_2) + a_3) + ... + a_{n-1}) + a_n.$

Теперь для вычисления значения числа *a* достаточно:

первую цифру числа умножить на 10 и к произведению прибавить вторую цифру;

полученную сумму умножить на 10 и прибавить следующую цифру;

пункт 2) выполнять до тех пор, пока не используем все заданные цифры числа.

Описанный выше алгоритм называется *схемой Горнера.* Работа алгоритма

рассмотрена для случая, когда цифры числа расположены в массиве в прямом (а, — цифра старшего разряда) порядке.

Число будем получать в переменной *a.* Цифры числа хранятся в массиве *B* (элемент j3[1] содержит цифру старшего разряда).

```
a: =0
нц для i от 1 до N
[ a:=a*10 + B[i]          (3.12)
кц
```

Для получения числа *a,* цифры которого заданы в обратном порядке (первый элемент массива содержит цифру младшего разряда), можно воспользоваться схемой Горнера, начиная обработку с последнего элемента массива.

Можно воспользоваться и другим алгоритмом. Каждая цифра должна умножаться на соответствующую ей разрядную единицу; первая — на 1, вторая — на 10, третья — на 100 и т. д. Каждая следующая разрядная единица в 10 раз больше предыдущей. Число *a* будем накапливать следующим образом: будем брать очередную цифру, умножать ее на соответствующую ей разрядную единицу и прибавлять полученное произведение к уже накопленному числу. Разрядную единицу увеличим в 10 раз. Так продолжаем до тех пор, пока не используем все цифры числа.

В переменной *г* будем хранить значение разрядной единицы, цифры числа будем хранить в массиве *B.*

```
a: = 0 г: = 1
нц для i от 1 до N ! a: = a-{-B [ij*r
! г: = г* 10        (3.13)
кц
```

Проанализируйте алгоритмы (3.12) и (3.13) на скорость работы (посчитайте количество выполненных арифметических операций).

Раздел 4. РЕШЕНИЕ ГЕОМЕТРИЧЕСКИХ ЗАДАЧ

Тема 4.1 Решение геометрических задач.

Геометрия развивается по многим направлениям. Возникновение компьютеров привело к появлению такой области математики, как вычислительная геометрия. При создании современных приложений часто требуется разработка эффективных алгоритмов для определения взаиморасположения различных объектов на плоскости, вычисления расстояний между ними, вычисления площадей фигур и др.

В данной главе излагается материал, частично известный вам из курса математики. Мы рассмотрим методы решения геометрических задач, которые эффективно реализуются с помощью компьютера, что позволит вам по другому взглянуть на вопросы, изучаемые в рамках школьного курса геометрии. Для этого придется воспользоваться аналитическим представлением геометрических объектов.

Положение точек относительно прямой

Множество точек прямой, проходящей через две точки с координатами (xf, y_t) и (x_2; y_2), удовлетворяет уравнению

(x-;*i)(0_2 — *Ух)—(У — Щ)&з* — Jt,) = 0.

Это значит, что если имеется точка с координатами (x_0; y_0) и $(x_Q - x_t)(y_2 - y_l) - (y_0 - y_l)(x_2 - x_l) = 0$, то эта точка лежит на прямой. В дальнейшем вместо выражения

$(x-x_i)(y_2-y_i)\sim-(y-y_l)(x_2 - x_i)$ мы иногда будем использовать для краткости обозначение

$Ax + By + C$ или $f\{x_1, y_u x_2, y_2, x, y)$.

Прямая $Ax-\-By-\-C$ — 0, проходящая через две заданные точки с координатами (х,; г/,) и (x_2; y_2), разбивает плоскость на две полуплоскости. Рассмотрим возможные значения выражения $Ax-\- By-\- C$.

Лх+Вг/ + С = 0 — определяет геометрическое место точек, лежащих на прямой.

Запишем алгоритм для определения, лежит ли точка с координатами (x_3; y_3) на прямой, проходящей через точки (х,; y_x) и (x_2; y_2)■ Переменная *Р* — переменная логического типа, которая имеет значение «истина», если точка лежит на прямой, и имеет значение «ложь» в противном случае.

```
Р: = «ложь»
если (х3—-xl)*(y2 — yl) — (y3 — yl)*(x2 — x1)~0
| то Р:=«истина»       0-2)
все
```

$Ax-\-By-\-C>0$ — определяет геометрическое место точек, лежащих по одну сторону от прямой.

$Ax-\-By-\-C<0$ — определяет геометрическое место точек, лежащих по другую сторону от прямой.

Это значит, что если для двух точек с координатами (x_3; y_3) и (л;$_4$; г/$_4$) значения выражений Ax_3 + -\-By_3-{-*C* и Лх$_4$ + Бг/$_4$+С имеют разные знаки, то эти точки лежат по разные стороны от прямой, проходящей через точки с координатами (я,; г/,) и $(x_2; y_2)$, а если одинаковые, то эти точки лежат по одну сторону от прямой. При этом число 0 имеет знак и « + » и «—■».

На рисунке 3 точки $(x_3; y_3)$ и (x_4; г/$_4$) лежат по одну сторону от прямой, точки (дс$_3$; г/$_3$) и (x_5; $y_Б$) — по разные стороны от прямой, а точка (x_6; г/$_6$) лежит на прямой.

Рассмотрим пример: jc, = 1, г/, = 2, $x_2 = 5$, y_2 — *Q*-

Уравнение прямой, проходящей через точки (х,; y_t) и $(x_2\ y_2)$, будет следующим:

Л = *У2*~ #1 = 6 —2 = 4, *В*—x_x—x_2= 1 — 5= —4, *С* = - $x_1(y_2$~«/,) + + г/,(x_2 —х,) — — 1 -4 + + 2 -4 — 4.

Следовательно, уравнение прямой будет иметь вид: $4x$ — 4у + 4 — 0, или x — у+1==0. Подставим координаты точек (3; 4), (1; 1), (2; 0), (0; 2) в уравнение прямой. - Рис. 3

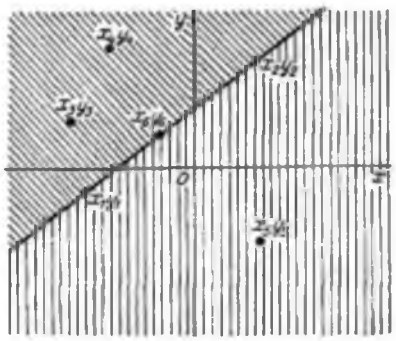


Рис. 3

Получим: 1 - 3 — 1 - 4 + 1=0, Ы-Ы + 1>0,

1 - 2 — 1 - 0 + 1 >0, 1-0—1 - 2 + 1 <0.

Следовательно, точка (3; 4) лежит на прямой, точки (1; 3) и (2; 0) лежат по одну сторону от прямой, а точки (1; 1) и (0; 2) — по разные стороны от прямой.

Алгоритм определения взаимного расположения точек (x_3; г/$_3$) и (x_4; y_4) относительно прямой, проходящей через точки (х,; г/,) и (x_2; y_2), можно записать следующим образом:

L: = «по одну»
Z1: =(x3 — xl)*(y2 — yl) — (y3 — yl)*(x2 — xl)
Z2: =(x4 — x1)*(y2 —y1) —(y4 —y1)*(x2— x1)
если Z1*Z2<0
| то L: = «no разные» (1.3)

Взаимное расположение двух отрезков

Пусть нам необходимо определить взаимное расположение двух отрезков. Отрезки на плоскости заданы координатами своих концевых точек. Предположим, что концевые точки одного из отрезков имеют координаты (х,; y_x) и (x_2; y_2), а концевые точки другого — (x_3; y_3) и (x_4; z/$_4$). Пусть общее уравнение первой прямой, проходящей через точки (%; y_x) и (x_2; y_2), имеет вид Л|Х + Б^+ С, = 0, а уравнение второй прямой, проходящей через точки (x_3; г/$_3$) и (x_4; y_A), выглядит так:

$Л_2x + B_2y + C_2 = 0$.

Определим расположение точек (x_3; y_3) и (x_4; y_4) относительно первой прямой. Если они расположены по одну сторону от прямой, то отрезки не могут пересекаться. Аналогично можно определить положение точек (Xj; $y_:$\) и (x_2; y_2) относительно другой прямой.

Таким образом, если значения пары выражений Z\ —$A_x x\%$-\-$B\{y_л$-{- С, и $Z_2 = A_I x_i$-\-B\y^-{-C_t имеют разные знаки или Z(-Z_2 = 0, а также пары -£з = ^2*1 + ^2#1 + C_2 и 2_4 = $Л_2$а:$_2$-|~ ВаУгН$^{-2}$ имеют разные знаки или Z_3-Z_4 = 0, то отрезки пересекаются. Если же значения пар выражений Z, и Z_2 или Z_3 и Z_4 имеют одинаковые знаки, то отрезки не пересекаются.

Различные случаи расположения отрезков показаны на рисунке 4.

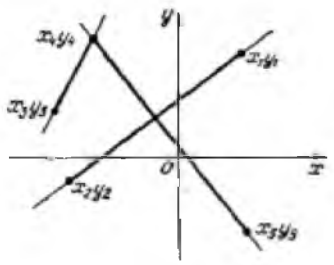


Рис. 4

На этом рисунке отрезки с концами в точках (*,*, y_x), $У_2$) и (~7 $У*)у$ (%> $У_3$) пересекаются, отрезки с концами в точках (*[; г/,), (x_2; г/$_2$) и (x_3; y_3), (x_4; г/$_4$) не пересекаются, а отрезки с концами в точках (x_3; $y_&$), (x_4; y_A) и (x_4; г/$_4$) и $Уъ$) имеют общую вершину. Последний случай можно считать частным случаем пересечения.

Алгоритм для определения, пересекаются ли два отрезка с концами в точках (x_t\ y_t), (x_2; y_2) и (лс$_3$; y_3), (лс$_4$; г/$_4$), будет следующим:

Р:=«истина»

Z1: =(x3 — xl)*(y2 — yl) — (y3 — yl)*(x2 — xl)

Z2: = (x4 — xl)*(y2 — y1)~ (y4 — y1)*(x2—• xl) если Z1*Z2>0

J то Р:=«ложь»(1-4)

все

Z3: =(x1 — x3)*(y4—-y3) —(y ! — y3)*(x4 —x3)

Z4: = (x2--x3)*(y4 —y3)— (y2 — y3)*(x4— x3) если Z3*Z4>0 | то Р: = «ложь» все

Приведенный фрагмент алгоритма не учитывает

крайней ситуации, когда два отрезка лежат на одной прямой. В этом случае

{x_3 — x_l)(y_2~y_l)—(y_3~y_1)(x_2~-x_i)=0 и (x_4 **— X,)x**

X(ft—0i)—(0$_4$—0i)(*s—*i)=0-

На рисунке 5 отрезки, лежащие на одной прямой, не пересекаются, а на рисунке 6 — пересекаются.

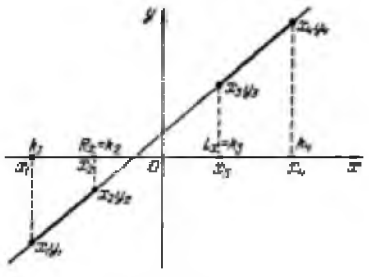


Рис. 5

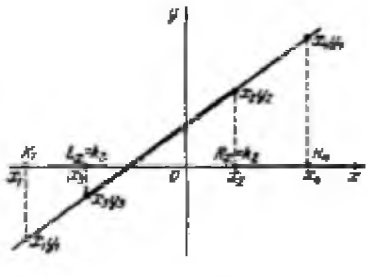


Рис. 6

Для того чтобы определить взаимное расположение таких отрезков, поступим следующим образом. Обозначим ^[~min(x,; ж$_2$); fc$_2$ —max^; x_2)\ fe$_3$ = min(x_3; лс$_4$); ft$_4$ = — шах (x_3; x_4).

Здесь k_x является левой, а k_2 — правой точкой проекции первого отрезка (отрезка, заданного координатами (■*ь *Л)> ($^{X}_{2}$< $У_2$)) на ось Ох. Аналогично k_3 является левой, а &$_4$ — правой точкой проекции второго

отрезка (отрезка, заданного координатами (x_3; $y_ъ$), (x,; y_4)) на ось Ох. Аналогично ищем проекции на ось Оу.

Отрезки, лежащие на одной прямой, будут пересекаться тогда, когда их проекции на каждую ось пересекаются. (Следует заметить, что если проекции двух произвольных отрезков пересекаются, то это не значит, что и сами отрезки пересекаются, что видно на рисунке Для определения взаимного расположения проекций на ось Ох воспользуемся следующим фактом (см. рис. 5 и 6): координата левой точки пересечения проекций L_x равна тах(£,; й$_3$), т. е. максимальной из координат левых точек проекций. Рассуждая аналогично для правых точек проекций, получим, что координата правой точки R_x пересечения равна min (k_2; й$_4$). Для того чтобы отрезки пересекались, необходимо, чтобы левая координата пересечения проекций была не больше правой координаты пересечения отрезков (такой случай имеет место на рис. 5, когда L_x — x_3, а R_x = x_2). Поэтому условием пересечения проекций является выполнение неравенства L_X^.R_X.

Аналогично можно вычислить величины L_y и $R_{y>}$ взяв соответствующие проекции на ось Оу.

Следует отметить, что длина пересечения проекций в этом случае равна величине R_x—L_x (если R_x—L_x = О, то проекции имеют только общую точку).

Точка пересечения отрезков

Для определения места пересечения отрезков (если известно, что они пересекаются), достаточно определить точку пересечения прямых, на которых эти отрезки лежат.

Пусть Лijc-f-5,# + €; = () — уравнение прямой, проходящей через концевые точки первого отрезка, а A_2х-\- -\-B_2у-\-C_2=® — уравнение прямой, проходящей через концевые точки второго отрезка.

Тогда для определения точки пересечения отрезков достаточно решить систему уравнений

A_xх-\-B_s у—— — С,, $A_{2x}B_2$у = — C_2.

Домножив первое уравнение на Л$_2$, а второе — на $A_и$

получим

A_2A_xх-\- A_2B_xу— —$A_2C_Ь$ A_1A_2X-\- A]B_2y = —A_1C_2.

Вычитаем из первого уравнения второе и находим значение у:

A,C, ^2^4

A_2B,-A\B_2-

Аналогично вычисляем значение х:

B_jC_j—B_2C_l B_2A_2 — В

Это справедливо в случае, если Л$_2$-б, — A_1-B_2Ф0. Но мы уже знаем, что отрезки пересекаются и не лежат на одной прямой, а это невозможно, если A_2-B_x — A_x-B_2 = 0.

Раздел 5. СТРУКТУРЫ ДАННЫХ

Тема 5.1 Структуры данных.

Часто задача может быть сформулирована на языке основных математических понятий, например таких, как множество, и тогда алгоритм решения ее может быть изложен в терминах основных операций над основными объектами. Для разработки эффективного алгоритма решения поставленной задачи нужно проанализировать несколько различных структур данных. Может оказаться, что каждая из структур данных позволяет нам выполнять некоторую из операций легко-, а другие — с большим трудом. В этом случае часто выбирают ту структуру, которая лучше всего подходит для решения всей задачи в целом (не делает максимально лёгким выполнение ни одной операции, но позволяет выполнить всю работу лучше, чем при любом очевидном подходе). Поэтому разработка эффективного алгоритма напрямую связана с разработкой хорошей структуры данных.

В этой главе сначала вводится ряд элементарных структур данных: списки, стеки, очереди — и показывается, как с помощью этих структур можно представлять графы и деревья. Затем вводятся сне- циааьные структуры данных: множества и приоритетные очереди.

Массивы

Наиболее известной и распространенной структурой данных являются *массивы.*. Эта структура однородна, так как все компоненты имеют один и тот же тип и все они равнодоступны. Массивы относятся к структурам данных со случайным доступом: для выделения некоторой компоненты к имени массива добавляется индекс (значение специального тина), который можно вычислить, ношшгу элементы массива иногда называют переменными с индексами.

К базовым операциям над массивами относятся поиск элемента и поиск максимального (минимального) элемента.

Поиск элемента *x.*

а) в произвольном массиве трудоемкость операции *O(n);*

б) в отсортированном массиву трудоемкость операции 0(log ??.).,

Поиск элемента в отсортированном массиве можно выполнить

за время O(logrc), если в качестве алгоритма поиска используется поиск делением пополам (двоичный поиск), так как в этом случае максимальное число сравнений равно log??- (для линейного поиска надо в среднем *n/2* сравнений).

Пусть *L* и *R* обозначают левый и правый концы интервала поиска элемента *x* в массиве *a.*

Алгоритм двоичного поиска

begin *L* **:= 0;**

R := n;

while *(L* •: *R)* do

= m;

if *a[m]* < *x* then *L* := *m* + **1** else *R* := *Щ*
}
end;

Поиск максимального. (минимального) элемента.

а) в произвольном массиве трудоемкость операции *0(n.*);

б) в отсортированном массиве трудоемкость операции 0(1).

Упражнение. Написать рекуррентное уравнение алгоритма двоичного поиска.

Следует отмётить, что обычный прием работы с массивами — это поиск и выборочное изменение отдельных компонент массива.

Основные операции над массивами не предполагают включения новых или исключения элементов.

Однако на практике часто нужно выполнять процедуры включения и исключения отдельных элементов, что приводит к необходимости разработки структур данных, поддерживающих эти операции.

Линейные списки

Список — конечная последовательность элементов, порядок которых определяется с помощью ссылок.

К базовым операциям для списка относятся:

поиск некоторого заданного элемента с ключом *ж* (трудоемкость *O(n))* |

поиск максимального (минимального.) элемента (трудоемкость ***O(n))-,***:

включение элемента (трудоемкость 0(1));

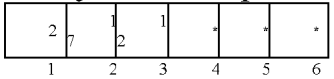
исключение элемента (трудоемкость 0(1));

формирование списка (трудоемкость *O(n)):*

просмотр списка (трудоемкость *O(n)).*

Существуют различные способы реализации списка. Наиболее распространенными являются следующие: реализация в виде двух массивов и в виде последовательно связанных компонент.

Рассмотрим сначала реализацию в виде двух массивов — *A* и *B*. Пусть *г* — индекс элемента в списке, тогда значение *A[i]* — сам элемент, значение *B[i]* — индекс следующего за *A[i]* элемента в списке *A*. Если *к* — индекс последнего элемента в списке, то *B[к]* = 0. Кроме того, существуют две переменные: *ns* — индекс начала занятых компонент и: *nf* — индекс начала свободных компонент. Тогда, массив 2,12,17 может быть реализовав следующим образом: *nf* = 4, *ns* = 1.



Упражнение. Написать процедуры, реализующие базовые операции над списком, который моделируется с помощью двух массивов.

Рассмотрим теперь реализацию списка, которая состоит в использовании последовательно связанных компонент. При такой реализации каждая

компонента списка состоит из двух ячеек памяти:

первая содержит сам элемент либо указатель на его местоположение, а вторая — указатель на с.киукнции ^леяёнт (такие списки часто называют *линейными* или *однонаправленными с писками);*

Тогда массив 2,12,17 будет реализован следующим образом:

|-1—I H 121 —I >1171 *nil* ~-|

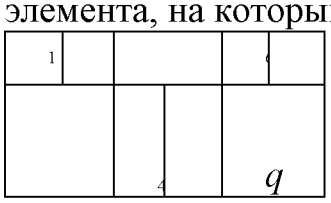
Приведем реализацию некоторых базовых операций для работы с линейным списком, который описан следующим образом:

и к =| el] el = record *key* : *data] next : uk]* end;

Включение элемента в список

Включение Элемента может быть осуществлено несколькими способами. В связи с тем, что в списке всегда известно его начало, новый элемент традиционно включают в начало списка.

Приведем пример процедуры включения элемента с ключом *x* после элемента, на который указывает ссылка p.



Программная реализация данной процедуры может быть следующей :

new*(q)] q | Jtey := x] q | .next '. = p* t *.next] p | .next : q:*

Поиск элемента ж в неупорядоченное списке осуществляется последовательный просмотром элементов. Он заканчивается либо при обнаружении требуемого элемента, либо при достижении конца списка. Дй$ того чтобы оптимизировать условие окончания просмотра, будем использовать технику барьера (s). Переменная *head будет* указывать на начало списка.

Ниже приводится программная реализация процедуры поиска элемента *x.*

s | ,кеу '■= x; w : = head;

while *(w | .key ^ x)* do *iv : = w | ,next'v*

if *w s* then { *W* - ссылка на элемент с ключом *x }*

else { элемент не найден: }

Для того чтобы процедура отработана правильно в случае, когда список пустой, необходимо при формировании списка выполнить оператор: *head : = s.*

Трудоемкость описанной выше процедуры поиска есть *O(n).*

Усовершенствовать метод поиска можно, если осуществлять поиск в упорядоченном списке. В упорядоченном списке поиск элемента *x* можно заканчивать при обнаружении первого ключа со значением большим, чем *x.* Упорядоченность списка Достигается путем включения нового элемента в подходящее для него место, что позволяет полностью использовать гибкость структуры списка. Однако даже упорядоченные линейные списки не позволяют организовать ничего подобного на двоичный поиск в массивах.

К операциям: работы: со списками также можно отнести:

Конкатенацию (сцепление) двух списков, в результате чего образуется:

единый список. Эту операцию можно выполнить за время 0(1), если имеются адреса первого и последнего элементов списков, •что позволит выполнить операцию сцепления без полного просмотра одного из списков для определения адреса его последнего элемента. Если известны только начальные адреса, то трудоемкость данной операции есть *0(n + m),* где *n, m* — количество элементов в списках,

Расцепление списка. Эту операцию можно выполнить за время 0(1), если извейтён указатель на Элемент, непосредственно предшествующий месту расцепления.

Стеки

Иногда при работе со списковой структурой возникает потребность включать и исключать элементы в определенном порядке. Если реализуется принцип "последний вошел — первый вышел" *(LIFO),* то такую списковую структуру называют *стеком (магазином).* Напомним, что алгоритмы решения таких известных задач, как проверка баланса скобок в арифметическом выражении, вычисление значения арифметического выражения, используют стек. В дальнейшем мы рассмотрим еще ряд задач, в которых будет использоваться эта структура данных.

Если заранее известно максимальное количество элементов *(I),* одновременно хранящихся в стеке, то целесообразно моделировать стек на массиве постоянной длины. Пусть переменная *top* содержит индекс последнего включенного в стек элемента (сначала *top* = 0), тогда процедуры включения и исключения элемента из стека реализуются следующим образом:

Включение элемента в стек, реализованный в виде одномерного массива

if *top* > *I* then { нет места } else { inc*(top)*-,

A\top := *x: }*

Исключение элемента из стека, реализованного в виде одномерного массива

if top < 1 then { стек пуст } else { *x* := i4[fop]; *dec(top); }*

В случае, когда количество элементов заранее неизвестно, стек может быть реализован в виде списковой структуры:

uk =f *11: el* = record *ml . dal t/.*

next : *uk;* end;

Тогда переменная *top* uv;u' г содержать адрес последнего включенного в Стек элемента (первоначально *top = nil), а* процедуры включения и исключения элемента будут иметь следующий вид:

Включение элемента в стек, реализованный в виде списковой структуры

n *ew(dop)*; *dop | .val := x] dop | .in .rl* := *top*; *top* := *dop*;

Исключение элемента из стека, реализованного в виде списковой структуры

if *top = nil* then { стек пуст } else ;

x : top | .rnl:
dop := top; top : hip j *.next]*
dispose^*iop)]*
}

Трудоемкость процедур включения и исключения элемента из стека есть 0(1).

Раздел 6. ТЕОРИЯ ГРАФОВ

Тема 6.1 Теория графов.

Существует много алгоритмов на графах, в основе которых лежит такой перебор вершин графа, при котором каждая вершина просматривается в точности один раз. Поэтому важной задачей является нахождение хороших методов поиска в графе.

Будем: говорить, что:

вершина является *помеченной,* если она занесена в структуру (т. е. ей Дается некоторая метка);

вершина является *просмотренной,* если она удалена из структуры первый раз (иногда в структуре некоторая вершина может храниться несколько раз, но с разными метками, и первое удаление данной вершины из структуры делает ее просмотренной).

ПОИСК В ГЛУБИНУ В ГРАФЕ

Поиск в глубину — один из методов обхода всех вершин (и дуг) графа *G = (V,Mh* который послужил основой для разработки ряда эффективных алгоритмов на графах.

В процессе поиска в глубину вершинам присваиваются метки, которые соответствуют порядку обхода вершин графа, а его ребра помечаются как "древесные" или "обратные". В начале работы алгоритма вершины не имеют меток, а ребра не помечены.

Для программной реализации алгоритма поиска в глубину в графе будем использовать стек. В процессе работы алгоритма каждая вершина включается и исключается из стека только один раз. Она включается в стек после того, как ей присваивается метка, и исключается, когда происходит возвращение из этой вершины.

Алгоритм поиска в глубину в графе

Некоторой стартовой вершине *s* присваивается метка & = 1, и вершина *s* заносится в стек.

2. Пока все вершины не будут помечены или стек не станет пуст (произойдет возвращение в вершину *s,* а из нее все ребра уже помечены), повторять следующие действия: пусть *v* — последняя занесенная в стек вершина и *к* — последняя присвоенная метка, тогда возможна одна из ситуаций:

имеется непомеченное ребро *(v,* ш), и у вершины *m* уже имеется метка, тогда ребропомечаем как "обратное" и продолжаем поиск непомеченного

ребра, инцидентного вершине *v;*

имеется непомеченное ребро *(v,w),* и вершина *w* не имеет метки (является новой), тогда присваиваем вершине *Я* метку, равную *к* = = *к* + 1, помечаем ребро *(v,iv)* как "древесное", заносим вершину *w* в стек и возвращаемся к шагу 2 алгоритма;

все ребра, инцидентные вершине *v,* уже помечены, тогда уд;>- ляем вершину *v* из стека (вершина *v* становится использованной или просмотренной) и возвращаемся к шагу 2 алгоритма.

Таким образом, поиск в глубину из вершины *n* заключается в поиске в глубину для всех новых вершин, смежных с *v.*

Оценим трудоемкость алгоритма. Каждое включение и исключение вершины из стека выполняется за время 0(1). Так как каждая вершина помечается и просматривается только один раз, то для всей работы, связанной со стеком, требуется время 6§4V|), Каждое ребро графа помечается только один раз, и Это требует времени порядка 0(1), поэтому трудоемкость пометки ребер есть *0(\E\).* Следовательно, трудоемкость поиска в глубину в графе есть 0(\V\ + |£|).

Пример 5.1. На рис. 5.1 приведены два примера реализации алгоритма поиска в глубину. В результате работы алгоритма:

каждое ребро помечено как ’оГ — древесное или V — обратное;

каждая вершина характеризуется следующим образом: [метка; номер вершины, из которой данная вершина получила метку].

Замечание 5.1. Заметим, что множество всех ребер графа *G,* которые были помечены как "древесные", в результате поиска в игу- бину из некоторой стартовой вершины *s* образуют ориентированное корневое дерево *T* графа *G* с корнем: в вершине *s* (это дерево часто называют корневым глубинным деревом).

Замечание 5.2. Поиск в глубину может быть использован для выделения связных компонент неориентированного графа.

Для выделения связных компонент неориентированного графадостаточно выполнить поиск в глубину, начиная с произвольной начальной вершины *s.*

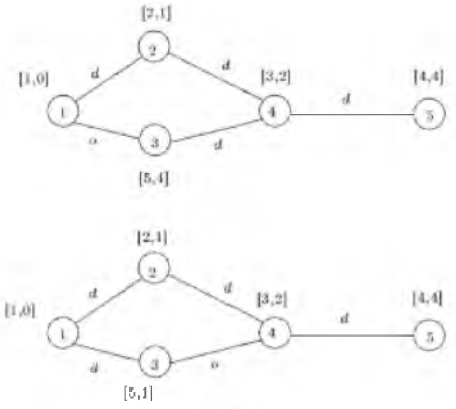


Рис. 5.1.

Множество помеченных вершин и множество "древесных" и "обратных" дуг Образуют компоненту связности графа. Если имеются непомеченные

вершины, то необходимо найти одну из них и, налагая ее в качестве начальной вершины s, повторить для нее описанный выше процесс поиска в глубину (при этом будет получена новая компонента связности графа).

Процесс выделения: связных компонент графа заканчивается, когда все вершины будут помечены.

Трудоемкость алгоритма выделения связных компонент неориентированного графа есть $0(|V| + |£J|)$.

Связные компоненты: орграфа находятся: так же, как в неориентированном графе, если: рассматривать исходный орграф как неориентированный.

ПРАКТИЧЕСКИЙ РАЗДЕЛ

Лабораторная работа 1

Лабораторная работа
Стратегии решения задач
Принцип «Разделяй и властвуй»

Пример 6.1. Отсортировать массив из n элементов, используя технику "разделяй и властвуй" (сортировку слиянием и сортировку Хоара).

Пример 6.2. Найти максимальный и минимальный элементы массива двумя различными алгоритмами: используя технику "разделяй и властвуй" и без ее использования.

Упражнение. Решить описанную в примере 6.7 задачу в предположении, что строка y может содержать символы '$*$' и '?', где '$*$' — означает любую подпоследовательность символов, а '?' — некоторый одиночный символ.

Динамическое программирование и градиентные алгоритмы

Разделяй и властвуй	Динамическое программирование
независимые подзадачи	зависимые подзадачи
решения всех подзадач не запоминаются	решения всех подзадач запоминаются

Пример 6.4. Заданы две строки символов $A = \{a_1, a_2, \ldots, a_m\}$ и $B = \{b_1, b_2, \ldots, b_n\}$. Необходимо определить $d(A, B)$ — минимальное число вставок, удалений и замен символа, требуемое для преобразования одной строки в другую.

Пример 6.3. Рассмотрим задачу перестановки сегментов. Пусть даны фиксированные целые переменные m, n, p, которые удовлетворяют условию: $m < n < p$. Дана часть массива $A[m : p-1]$, которая рассматривается как два сегмента — $b[m : n-1]$ и $c[n : p-1]$.

Лабораторная работа 2

Лабораторная работа
Алгоритмы целочисленной арифметики
Реализация алгоритмов целочисленной арифметики

1. Определить, является ли данное число четным.
2. Два натуральных числа называют дружественными, если каждое из них равно сумме всех делителей другого, кроме самого этого числа. Найти все пары дружественных чисел, лежащих в диапазоне от n до k.
3. Найти натуральное число из диапазона от n до k, которое имеет наибольшее количество делителей.
4. Разложить дробь p/q на сумму дробей вида $1/n$. Например, при $p=3$, $q=7$

$$\frac{3}{7}=\frac{1}{3}+\frac{1}{11}+\frac{1}{231}.$$

5. Два двузначных числа, записанных одно за другим, образуют четырехзначное число, которое делится на их произведение. Найти эти числа.
6. Найти натуральные числа из отрезка $[n; k]$, количество делителей у которых является произведением двух простых чисел.

Реализация алгоритмов целочисленной арифметики

1. Сократить дробь a/b (a, b — натуральные числа).
2. Даны 4 целых числа a, b, c, d. Написать программу, вычисляющую сумму обыкновенных дробей $a/b + c/d$ в виде x/y.
3. Дано натуральное число n. Определить количество сотен (тысяч, миллионов) в нем.
4. Ввести период дроби. Напечатать числитель и знаменатель.

Лабораторная работа 3

Лабораторная работа
Структуры данных
Использование структур данных

1. Упражнение. Написать процедуры, реализующие базовые операции над списком, который моделируется с помощью двух массивов.

2. Написать программу, проверяющую своевременность закрытия скобок в строке символов.

Задача Игра в пьяницу

В игре в пьяницу карточная колода раздается поровну двум игрокам. Далее они вскрывают по одной верхней карте, и тот, чья карта старше, забирает себе обе вскрытые карты, которые кладутся под низ его колоды. Тот, кто остается без карт – проигрывает.

Для простоты будем считать, что все карты различны по номиналу, а также, что самая младшая карта побеждает самую старшую карту ("шестерка берет туза").

Игрок, который забирает себе карты, сначала кладет под низ своей колоды карту первого игрока, затем карту второго игрока (то есть карта второго игрока оказывается внизу колоды).

Напишите программу, которая моделирует игру в пьяницу и определяет, кто выигрывает. В игре участвует 10 карт, имеющих значения от 0 до 9, большая карта побеждает меньшую, карта со значением 0 побеждает карту 9.

Формат входных данных

Программа получает на вход две строки: первая строка содержит 5 карт первого игрока, вторая – 5 карт второго игрока. Карты перечислены сверху вниз, то есть каждая строка начинается с той карты, которая будет открыта первой.

Формат выходных данных

Программа должна определить, кто выигрывает при данной раздаче, и вывести слово `first` или `second`, после чего вывести количество ходов, сделанных до выигрыша. Если на протяжении 10^6 ходов игра не заканчивается, программа должна вывести слово `botva`.

Пример

Входные данные	Выходные данные
1 3 5 7 9 2 4 6 8 0	second 5

РАЗДЕЛ КОНТРОЛЯ ЗНАНИЙ

Список задач к зачёту

Задание 1. Ручка стоила K рублей. Первого сентября стоимость ручки увеличилась ровно на P процентов. Определите, сколько ручек можно купить на S рублей после подорожания.

Программа получает на вход три целых положительных числа. Первое число K – стоимость ручки в рублях до подорожания. Второе число P – величина подорожания ручки в процентах. Третье число S – имеющаяся сумма денег. Числа K и S не превосходят 10^7, число P не превосходит 100.

Пример входных и выходных данных

Ввод	Вывод
33 5 100	2

Задание 2. В некотором царстве жил Змей Горыныч. У него было N голов и M хвостов. Иван-царевич решил уничтожить губителя человеческих душ, для чего ему его кума Баба Яга подарила волшебный меч, так как только им можно убить Змея Горыныча. Если отрубить одну голову, то на её месте вырастает новая, если отрубить хвост, то вместо него вырастет 2 хвоста. Если отрубить два хвоста, то вырастает 1 голова, и только когда отрубить 2 головы, то не вырастет ничего. Змей Горыныч гибнет только в том случае, когда ему отрубить все головы и все хвосты. Определить минимальное количество ударов мечом, нужное для уничтожения Змея Горыныча.

Входные данные

В единственной строке записаны через пробел два числа N, M ($0 \le N, M \le 1000$).

Выходные данные

В единственную строку нужно вывести одно число – минимальное количество ударов мечом, или -1, если уничтожить Змея Горыныча невозможно.

Пример входных данных:

3 3

Пример выходных данных:

9

Задание 3. Вот что записано в тетради ученика:

101-10=11

101+10=111

101*10=1010

Понятно, что арифметические действия выполняются не с десятичными числами, а с двоичными. Надо написать программу получения таких действий в указанной системе счисления p ($2 \le p \le 16$) для заданных чисел A и B ($1 \le B \le A \le 10000$), записанных первоначально в десятичной системе

счисления. При выводе чисел в системе счислению больше 10 используются заглавные латинские буквы.

Пример входных данных:

2
5 2

Пример выходных данных:

101-10=11
101+10=111
101*10=1010

Задание 4. В Древнем Египте были очень распространены карточные гадания, которые заключались в следующем: двум участникам поровну раздавали колоду. Далее они вскрывали по одной верхней карте; тот, чья карта старше, забирал обе вскрытые карты. На этом шаг гадания завершался и начинался следующий: участники снова вскрывали по одной верхней карте и т.д.

Процесс продолжался, пока один из участников не оставался без карт, что и было результатом гадания.

Будем считать, что число карт n четно, и на картах без повторов записаны числа от 0 до $n-1$. Старшей считается та карта, на которой записано большее число, за одним исключением: карта с числом 0 старше карты с числом $n-1$.

Участник, который забирает себе карты, сначала кладет под низ своей колоды карту первого участника, затем карту второго участника гадания (то есть карта второго участника оказывается внизу колоды).

Напишите программу, которая определяет результат гадания – кто из участников останется без карт, и сколько шагов совершили участники.

Формат входных данных: Программа получает на вход три строки: первая строка содержит четное число n ($2 \le n \le 20$) – общее количество карт. Вторая строка содержит номера $n/2$ карт первого участника, вторая – номера $n/2$ карт второго участника.

Карты перечислены сверху вниз, то есть каждая строка начинается с той карты, которая будет открыта первой.

Формат выходных данных: Программа должна определить, кто из участников останется без карт, и вывести слово «first» или «second», после чего через пробел вывести количество ходов, сделанных до окончания.

Если же на протяжении 10^6 ходов гадание не заканчивается, программа должна вывести одно слово «Fatum».

Пример

Входные данные	*Выходные данные*
10 1 3 5 7 9 2 4 6 8 0	first 5

ВСПОМОГАТЕЛЬНЫЙ РАЗДЕЛ

Учреждение образования
«Белорусский государственный педагогический университет
имени Максима Танка»

УТВЕРЖДАЮ
Проректор по учебной и
информационно-аналитической работе
БГПУ
_____ В.М.Зеленкевич
_____ 2016 г.

Регистрационный № УД-

РЕШЕНИЕ СЛОЖНЫХ И ОЛИМПИАДНЫХ ЗАДАЧ ПО ПРОГРАММИРОВАНИЮ

Учебная программа учреждения высшего образования
по учебной дисциплине (по выбору студента) для специальности:
1-02 05 01 Математика и информатика

2016 г.

Учебная программа составлена на основе Образовательного стандарта высшего образования первая ступень специальность 1-02 05 01 Математика и информатика (ОСВО 1-02 05 01 – 2013) и Учебного плана специальности 1-02 05 01 Математика и [redacted] № 152 – 2013/у от 25.07.2013 г.)

СОСТАВИТЕЛИ:

С.И. Зенько, заведующий кафедрой информатики и методики преподавания информатики учреждения образования «Белорусский государственный педагогический университет имени Максима Танка», кандидат педагогических наук, доцент;

В.М. Котов, профессор кафедры информатики и методики преподавания информатики учреждения образования «Белорусский государственный педагогический университет имени Максима Танка», доктор физико-математических наук, профессор

РЕКОМЕНДОВАНА К УТВЕРЖДЕНИЮ:

Кафедрой информатики и [redacted] информатики
(протокол № 10 от 26.05.2016 г.)

Заведующий кафедрой _____ С.И.Зенько

Советом физико-математического факультета
(протокол № 12 от 29.06.2016 г.)

Оформление учебной программы и сопровождающих её материалов действующим требованиям Министерства образования Республики Беларусь соответствует

Методист учебно-методического
управления БГПУ

_____ С.А.Стародуб

Ответственный за редакцию: С.И.Зенько
Ответственный за выпуск: С.И.Зенько

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

Одной из задач школьного образования, связанной с применением компьютеров является подготовка способных учащихся к участию в олимпиадах по информатике и программированию.

В условиях современного развития олимпиадного движения по информатике необходимо будущему педагогу иметь определенный банк знаний, включающий знания о классических структурах, данных стеках, очередях, линейных списках со связями, двоичных деревьях, графах, алгоритмах поиска и сортировки, рекуррентных соотношениях, комбинаторике, рекурсии, динамическому программированию и алгоритмах решения сложных и олимпиадных задач.

Цель учебной дисциплины по выбору – формирование у будущих преподавателей информатики профессиональных компетенций для построения эффективных алгоритмов решения сложных задач и решения оригинальных задач самого различного уровня, предлагавшихся на олимпиадах по программированию.

Основные задачи учебной дисциплины по выбору:

– разбор важнейших классов быстрых алгоритмов поиска и сортировки, их сравнительная характеристика;

– изучение динамических структур данных, применяемых в алгоритмах, приемов работы с ними;

– практическая реализация алгоритмов на компьютере;

– решение оригинальных олимпиадных задач.

Место учебной дисциплины в системе подготовки специалиста

Изучение учебной дисциплины по выбору «Решение сложных и олимпиадных задач по программированию» опирается на основные академические, социально-личностные и профессиональные компетенции, сформированные у студентов в процессе изучения ими таких учебных дисциплин как «Технологии программирования и методы алгоритмизации», «Вычислительные методы и компьютерное моделирование», «Практикум по решению задач по информатике». Благодаря ее изучению и приобретению умений создавать интерактивные динамические веб-страницы студенты смогут реализовать свои профессиональные потребности на современном уровне.

Профессиональные компетенции студентов

Учебная дисциплина по выбору «Решение сложных и олимпиадных задач по программированию» входит в компонент учреждения высшего образования. Изучение дисциплины по выбору студента «Решение сложных и олимпиадных задач по программированию» должно обеспечить формирование у студентов академических, социально-личностных и профессиональных компетенций.

Требования к академическим компетенциям

Студент должен:

– АК–3. Владеть исследовательскими навыками;

– АК–4. Уметь работать самостоятельно;

– АК–5. Быть способным порождать новые идеи (обладать креативностью);

– АК–6. Владеть междисциплинарным подходом при решении проблемы;

– АК–7. Иметь навыки, связанные с использованием технических средств устройств, управлением информацией и работой с компьютером;

– АК–9. Уметь учиться, повышать свою квалификацию в течение всей жизни;

– АК–10. Уметь осуществлять учебно-исследовательскую деятельность.

Требования к социально-личностным компетенциям

Студент должен:

– СЛК–3. Обладать способностью к межличностным коммуникациям;

– СЛК–7. Быть способным к осуществлению самообразования и самосовершенствования профессиональной деятельности.

Требования к профессиональным компетенциям

Студент должен быть способен:

Обучающая деятельность

– ПК–1. Управлять учебно-познавательной и учебно-исследовательской деятельностью обучающихся;

– ПК–2. Использовать оптимальные методы, формы и средства обучения;

– ПК–3. Организовывать и проводить учебные занятия различных видов;

– ПК–4. Организовывать самостоятельную работу учащихся.

Воспитательная деятельность

– ПК–8. Формировать базовые компоненты культуры личности воспитанника.

Развивающая деятельность

– ПК–12. Развивать навыки самостоятельной работы обучающихся с учебной, справочной, научной литературой и др. источниками информации.

Ценностно-ориентационная деятельность

– ПК–16. Оценивать учебные достижения учащихся, а также уровни их воспитанности и развития;

– ПК–17. Осуществлять профессиональной самообразование и самовоспитание с целью совершенствования профессиональной деятельности.

В результате изучения учебной дисциплины по выбору студент должен **знать**:

– алгоритмы решения наиболее широко распространенных классов задач обработки данных;

– структуры данных, используемые для хранения данных и рациональные приемы работы с ними;

– алгоритмы динамического программирования;

– основные алгоритмы на графах.

В результате изучения учебной дисциплины по выбору студент должен **уметь**:

– решать на компьютере задачи обработки данных;

– применять рекуррентные соотношения, переборные и рекурсивные методы при реализации алгоритмов;

– использовать динамические структуры данных;

– решать задачи, в которых используются алгоритмы на графах.

В результате изучения учебной дисциплины по выбору студент должен **владеть**:

– приемами оптимизации алгоритмов решения олимпиадных задач по информатике;

– методами анализа сложности алгоритмов;

– подходами динамического программирования.

Структура и содержание учебной дисциплины

Учебная дисциплина по выбору «Решение сложных и олимпиадных задач по программированию» изучается на протяжении одного семестра и содержит шесть тем. Первая тема посвящена проектированию и анализу алгоритмов. Во второй теме рассматривается стратегия решения задач динамического программирования и градиентные алгоритмы. В третьей теме рассматриваются алгоритмы целочисленной математики. Четвертая тема предполагает решение геометрических задач. Использование структур данных в задачах повышенного уровня изучается в пятой теме. Шестая тема посвящена решению задач, в которых используются алгоритмы на графах.

Данная учебная программа является основным документом, определяющим объем и содержание учебной дисциплины по выбору «Решение сложных и олимпиадных задач по программированию» для специальностей 1-02 05 01 Математика и информатика.

Методы обучения

Обучение учебной дисциплине по выбору проходит в рамках организации лекционных и лабораторных занятий. При чтении лекций особое внимание следует уделять использованию мультимедийных технологий.

Организация лабораторных занятий предполагает использование личностно-ориентированных методов обучения, что способствует развитию индивидуально-творческих способностей студентов и приобретению умений самостоятельной работы. Лабораторные работы направлены на формирование навыков решения профессионально-методических задач учителя информатики.

Содержание и формы самостоятельной работы студентов разрабатываются в соответствии с целями и задачами подготовки специалиста. Среди видов самостоятельной работы студентов представляется возможным применять: самостоятельную работу во время основных аудиторных занятий (лекций, лабораторных занятий); самостоятельную работу в форме консультаций; внеаудиторную самостоятельную работу при выполнении студентами домашних заданий учебного и творческого характера. Для управления самостоятельной работой рекомендуется использовать электронные средства обучения, тестирующие программы. Текущий контроль осуществляется в ходе выполнения и защиты лабораторных работ.

Распределение общего количества часов по формам обучения и семестрам

Дневная форма получения высшего образования:

Всего на учебную дисциплину – 50 часов.

8 семестр – 26 часов аудиторных (14 часов – лекции, 12 часов – лабораторные занятия), 24 часа – самостоятельная работа.

Форма контроля – зачет (8 семестр).

Заочная форма получения образования:

Всего на учебную дисциплину – 50 часов.

9 семестр – 8 часов аудиторных (4 часа – лекции, 4 часа – лабораторные занятия), 42 часа – самостоятельная работа.

Форма контроля – зачет (10 семестр).

СОДЕРЖАНИЕ УЧЕБНОГО МАТЕРИАЛА

Тема 1. Проектирование и анализ алгоритмов

Рекуррентные соотношения. Решение рекуррентных соотношений. Метод подстановки, итераций, рекурсивных деревьев. Способы упорядочения информации. Основные алгоритмы внутренней и внешней сортировки и их трудоемкость.

Тема 2. Стратегии решения задач

Принцип «Разделяй и властвуй». Динамическое программирование. Градиентные алгоритмы. Сведение задачи к подзадачам. Использование рекурсии. Использование таблиц.

Тема 3. Алгоритмы целочисленной арифметики

Представление чисел. Алгоритмы работы с большими числами. Поиск делителей числа, простые числа, разложение на простые множители. Поиск наибольшего общего делителя и наименьшего общего кратного. Операции с дробями. Нестандартные задачи.

Тема 4. Решение геометрических задач

Уравнения прямой, размещение точек плоскости относительно заданной прямой. Расстояние между точками и от точки до прямой. Точка пересечения отрезков. Понятие многоугольника. Площадь многоугольника. Взаимное расположение фигур.

Тема 5. Структуры данных

Организация списковых структур. Понятие списка, стека, очереди, кучи. Стандартные методы работы. Использование структур данных. Задачи повышенного уровня. Олимпиадные задачи.

Тема 6. Теория графов

Представления графов. Поиск в глубину и ширину. Поиск кратчайших путей. Алгоритмы на графах. Нахождение кратчайших путей в графе. Обходы графа.

УЧЕБНО-МЕТОДИЧЕСКАЯ КАРТА
для специальности 1-02 05 01 Математика и информатика
для дневной формы получения образования

Номер раздела, темы	Название раздела, темы	Количество аудиторных часов				Количество часов самостоятельной работы	Формы контроля знаний
		лекции	практические занятия	лабораторные занятия	Иное		
1	2	3	4	5	6	7	8
1	**Проектирование и анализ алгоритмов**	**2**				**2**	
1.1	Проектирование и анализ алгоритмов. 1. Рекуррентные соотношения. 2. Решение рекуррентных соотношений. Метод подстановки, итераций, рекурсивных деревьев. 3. Способы упорядочения информации. Основные алгоритмы внутренней и внешней сортировки и их трудоемкость	2				2	Устный опрос
2	**Стратегии решения задач**	**2**		**4**		**6**	
2.1	Стратегии решения задач. 1. Принцип «Разделяй и властвуй». 2. Динамическое программирование. 3. Градиентные алгоритмы.	2				2	Устный опрос
2.2	Принцип «Разделяй и властвуй». 1. Сведение задачи к подзадачам. 2. Использование рекурсии.			2		2	Проверка лабораторной работы
2.3	Динамическое программирование и градиентные алгоритмы. 1. Использование таблиц.			2		2	Проверка лабораторной

	2. Градиентные алгоритмы.						работы
3	**Алгоритмы целочисленной арифметики**	**2**		**4**		**6**	
3.1	Алгоритмы целочисленной арифметики. 1. Представление чисел. 2. Алгоритмы работы с большими числами.	2				2	Устный опрос
3.2	Реализация алгоритмов целочисленной арифметики. 1. Поиск делителей числа, простые числа, разложение на простые множители. 2. Поиск наибольшего общего делителя и наименьшего общего кратного.			2		2	Проверка лабораторной работы
3.3	Реализация алгоритмов целочисленной арифметики. 1. Операции с дробями. 2. Нестандартные задачи.			2		2	Проверка лабораторной работы
4	**Решение геометрических задач**	**4**				**2**	
4.1	Решение геометрических задач. 1.Уравнения прямой, размещение точек плоскости относительно заданной прямой. 2. Расстояние между точками и от точки до прямой. 3. Точка пересечения отрезков.	2				1	Устный опрос
4.2	Решение геометрических задач. 1. Понятие многоугольника. 2. Площадь многоугольника. 3. Взаимное расположение фигур.	2				1	Устный опрос
5	**Структуры данных**	**2**		**2**		**4**	
5.1	Структуры данных. 1. Организация списковых структур 2. Понятие списка, стека, очереди, кучи. 3. Стандартные методы работы.	2				2	Проверка лабораторной работы

5.2	Использование структур данных. 1. Задачи повышенного уровня. 2. Олимпиадные задачи.			2		2	Проверка лабораторной работы
6	**Теория графов**	**2**		**2**		**4**	
6.1	Теория графов. 1. Представления графов. 2. Поиск в глубину и ширину. 3. Поиск кратчайших путей.	2				2	Устный опрос
6.2	Алгоритмы на графах 1. Нахождение кратчайших путей в графе. 2. Обходы графа.			2		2	Проверка лабораторной работы
	Итого:	**14**		**12**		**24**	**Зачет**

УЧЕБНО-МЕТОДИЧЕСКАЯ КАРТА
для специальности 1-02 05 01 Математика и информатика
для заочной формы получения образования

Номер раздела, темы	Название раздела, темы	Количество аудиторных часов				Количество часов самостоятельной работы	Формы контроля знаний
		лекции	практические занятия	лабораторные занятия	Иное		
1	2	3	4	5	6	7	8
1	**Проектирование и анализ алгоритмов**					**4**	
1.1	Проектирование и анализ алгоритмов. 1. Рекуррентные соотношения. 2. Решение рекуррентных соотношений. Метод подстановки, итераций, рекурсивных деревьев. 3. Способы упорядочения информации. Основные алгоритмы внутренней и внешней сортировки и их трудоемкость					4	
2	**Стратегии решения задач**	**2**				**10**	
2.1	Стратегии решения задач. 1. Принцип «Разделяй и властвуй». 2. Динамическое программирование. 3. Градиентные алгоритмы.	2				2	
2.2	Принцип «Разделяй и властвуй». 1. Сведение задачи к подзадачам. 2. Использование рекурсии.					4	
2.3	Динамическое программирование и градиентные алгоритмы. 1. Использование таблиц.					4	

	2. Градиентные алгоритмы.						
3	**Алгоритмы целочисленной арифметики**			**2**		**10**	
3.1	Алгоритмы целочисленной арифметики. 1. Представление чисел. 2. Алгоритмы работы с большими числами.					4	
3.2	Реализация алгоритмов целочисленной арифметики. 1. Поиск делителей числа, простые числа, разложение на простые множители. 2. Поиск наибольшего общего делителя и наименьшего общего кратного.			2		2	
3.3	Реализация алгоритмов целочисленной арифметики. 1. Операции с дробями. 2. Нестандартные задачи.					4	
4	**Решение геометрических задач**	**2**				**4**	
4.1	Решение геометрических задач. 1.Уравнения прямой, размещение точек плоскости относительно заданной прямой. 2. Расстояние между точками и от точки до прямой. 3. Точка пересечения отрезков.	2				1	
4.2	Решение геометрических задач. 1. Понятие многоугольника. 2. Площадь многоугольника. 3. Взаимное расположение фигур.					3	
5	**Структуры данных**			**2**		**6**	
5.1	Структуры данных. 1. Организация списковых структур 2. Понятие списка, стека, очереди, кучи. 3. Стандартные методы работы.					4	

5.2	Использование структур данных. 1. Задачи повышенного уровня. 2. Олимпиадные задачи.			2		2	
6	**Теория графов**					**8**	
6.1	Теория графов. 1. Представления графов. 2. Поиск в глубину и ширину. 3. Поиск кратчайших путей.					4	
6.2	**Алгоритмы на графах** 1. Нахождение кратчайших путей в графе. 2. Обходы графа.					4	
	Итого:	**4**		**4**		**42**	**Зачет**

ИНФОРМАЦИОННО-МЕТОДИЧЕСКАЯ ЧАСТЬ

Основная литература:

1. Кормен, Т., Лейзерсон, Ч., Ривест, Р. Алгоритмы: построение и анализ / Т. Кормен, Ч. Лейзерсон, Р. Ривест. – 2-е издание: пер. с англ. – Москва: Вильямс, 2005. – 1296 с.
2. Котов, В.М., Соболевская, Е.П. Структуры данных и алгоритмы: теория и практика: учебное пособие / В.М. Котов, Е.П. Соболевская. – Минск: БГУ, 2004. – 255 с.
3. Котов, В.М., Соболевская, Е.П. Разработка и анализ алгоритмов: учебное пособие / В.М. Котов, Е.П. Соболевская. – Минск: БГУ, 2009. – 251 с.
4. Котов, В. М. Алгоритмы и структуры данных: учебное пособие / В. М. Котов, Е. П. Соболевская, А.А. Толстиков. – Минск: БГУ, 2011. – 267 с.
5. Волчкова, Г.П. Сборник задач по теории алгоритмов для студентов физико-математических спец. БГУ/ Г.П. Волчкова, В.М. Котов, Е. П. Соболевская. – Минск: БГУ, 2005. – 59 с.

Дополнительная литература:

6. Ахо, А.В., Хопкрофт, Д.Э., Ульман, Д.Д. Структуры данных и алгоритмы / А.В. Ахо, Д.Э. Хопкрофт, Д.Д. Ульман. – Москва: Вильямс, 2000. – 384 с.
7. Вирт, Н. Алгоритмы и структуры данных / Н. Вирт. – Санкт-Петербург: Невский Диалект, 2001. – 352 с.
8. Емеличев, В.А., Мельников, О.И., Сарванов, В.И., Тышкевич, Р.И. Лекции по теории графов / В.А. Емеличев, О.И. Мельников, В.И. Сарванов, Р.И. Тышкевич. – Москва: Наука, 1990. – 383 с.
9. Липский, В. Комбинаторика для программистов / В. Липский. – Москва: Мир, 1988. – 214 с.
10. Пападдимитриу, Х., Стайглиц, К. Комбинаторная оптимизация: Алгоритмы и сложность / Х. Пападдимитриу, К. Стайглиц. – Москва: Мир, 1971. – 512 с.
11. Рейнгольд, Э., Нивергельт, Ю., Део, Н. Комбинаторные алгоритмы теория и практика / Э. Рейнгольд, Ю. Нивергельт, Н. Део. – Москва: Мир, 1980. – 476 с.
12. Mark Allen Weiss. Data structures and algorithm analysis. – Benjamin/Cummings Publishing Company, 1992. – 455 p.
13. C. Shaffer. A Practical Introduction to Data Structure and Algorithm Analysis. – London: Prentice Hall International, 1997. – 494 p.

ПРИМЕРНЫЙ ТЕМАТИЧЕСКИЙ ПЛАН

№	Наименование разделов и тем	Всего	ЛК	ПР	ЛБ
1	Проектирование и анализ алгоритмов	**2**	**2**		
2	Стратегии решения задач	**6**	**2**		**4**
3	Алгоритмы целочисленной арифметики	**6**	**2**		**4**
4	Решение геометрических задач	**4**	**4**		
5	Структуры данных	**4**	**2**		**2**
6	Теория графов	**4**	**2**		**2**
	Итого:	**26**	**14**		**12**

Материалы на электронных носителях
(сайт физико-математического факультета, локальная сеть физико-математического факультета, кафедральные компьютеры)

1. Тексты методических указаний к лабораторным работам.
2. Вопросы к зачету.
3. Задания для самостоятельной работы

Методические рекомендации по организации и выполнению самостоятельной работы студентов

Содержание и формы самостоятельной работы студентов разрабатываются кафедрами в соответствии с целями и задачами подготовки специалиста. Для управления самостоятельной работой рекомендуется использовать электронные средства обучения, тестирующие программы. Текущий контроль осуществляется в ходе выполнения и защиты лабораторных работ.

Перечень используемых средств диагностики результатов учебной деятельности

Основным средством диагностики усвоения знаний, умений и овладения необходимыми навыками по учебной дисциплине являются:

– *фронтальный опрос* на лекционных занятиях, направлен систематизацию знаний студентов, определение уровня готовности аудитории к восприятию нового материала, а также на формирование у преподавателя представление об усвоении студентами основополагающих понятий и фактов изучаемой учебной дисциплины;

– *проверка заданий* разнообразного типа (рецептивные, репродуктивные, продуктивные, творческие), выполняемых в рамках часов, отводимых на учебные занятия (лабораторные), представляет собой диагностику систематичности подготовки студентов к занятиям, уровень усвоения ими практико- ориентированного содержания программного материала учебной дисциплины;

– *групповые и индивидуальные консультации студентов* предназначены для диагностики уровня овладения определенными знаниями, умениями и навыками, как теоретического материала, так и практического; устранения типичных ошибок и пробелов в знаниях обучающихся;

– *самостоятельные работы* используются для определения индивидуальных особенностей, темпа продвижения студентов и усвоения ими необходимых знаний;

– *компьютерное тестирование* позволяет относительно быстро провести диагностику усвоения студентами учебного материала как по отдельным темам и разделам учебной дисциплины, так и по учебной дисциплине в целом;

– *зачет* используется для осуществления итоговой диагностики усвоения учащимися содержания учебной дисциплины за учебный семестр и оценивается обычно в форме «зачтено» или «не зачтено» в соответствии с критериями оценки результатов учебной деятельности обучающихся в учреждениях высшего образования по десятибалльной шкале.

ПРОТОКОЛ СОГЛАСОВАНИЯ УЧЕБНОЙ ПРОГРАММЫ

Название учебной дисциплины, с которой требуется согласование	Название кафедры	Предложения об изменениях в содержании учебной программы учреждения высшего образования по учебной дисциплине	Решение, принятое кафедрой, разработавшей учебную программу (с указанием даты и номера протокола)
Алгебра; Теория чисел; Методы изображений фигур и основания геометрии	Кафедра математики и методики преподавания математики	При рассмотрении вопросов, связанных с решением практико-ориентированных задач, использовать согласованную терминологию в соответствии с действующими учебными пособиями для учреждений общего среднего образования	Протокол № 10 от 26.05.2016