№ 3604

51
С 873

СТРУКТУРЫ ДАННЫХ И АЛГОРИТМЫ

Методические указания

НОВОСИБИРСК
2008

Министерство образования и науки Российской Федерации

НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

51 № 3604
С 873

СТРУКТУРЫ ДАННЫХ И АЛГОРИТМЫ

Методические указания
к курсовой работе для студентов I курса ФПМиИ
(направление 010500 – Прикладная математика и информатика,
специальность 010503 – Математическое обеспечение
и администрирование информационных систем)
дневного отделения

НОВОСИБИРСК
2008

УДК 519.422.63 (07)
С 873

Составители: *В.П. Хищенко*, ст. преп.
Т.А. Шапошникова, ст. преп.

Рецензент *М.Э. Рояк*, д-р техн. наук, проф.

Работа подготовлена на кафедре прикладной математики

ВВЕДЕНИЕ

Курсовое проектирование является важным этапом процесса обучения и представляет собой самостоятельную работу студента (под руководством преподавателя), призванную развить и закрепить навыки инженерного творчества по специальности. Курсовая работа (КР) позволяет проверить умение студента применить полученные знания при решении конкретных задач.

Цель данной курсовой работы – практическое освоение абстрактных структур данных и алгоритмов их обработки при решении задач.

Тематика курсовой работы связана с решением задач на дискретных конечных математических структурах. Задания предполагают использование:

– абстрактных структур данных (списки, стеки, очереди, деревья, графы, таблицы);

– алгоритмов поиска (линейный поиск, бинарный поиск, исчерпывающий поиск);

– алгоритмов обхода иерархических структур в ширину, в глубину;

– алгоритмов, оперирующих со структурами типа дерево, граф, множество;

– алгоритмов порождения элементов конечного (как правило, чрезвычайно большого) множества, которые удовлетворяют определенным ограничениям.

Задания на курсовую работу разбиты по темам: графы, системы дорог, деревья; языки, грамматики; логические формулы, выражения; игры.

1. ТРЕБОВАНИЯ К ВЫПОЛНЕНИЮ КУРСОВОЙ РАБОТЫ

1.1. Решение задачи включает:

– формальную (на языке математики) постановку задачи;

– выбор продуманного и обоснованного метода решения;

– определение формы представления (внешнего и внутреннего) исходных данных и результатов задачи, т. е. спецификации данных;

– реализацию решения, с использованием принципов структурного и модульного программирования.

1.2. Отчет по КР включает следующие разделы:

1. Условие задачи (условие задачи должно быть записано полно, без сокращений).
2. Анализ задачи.
3. Структуры данных, используемые для представления исходных данных и результатов задачи.
4. Укрупненный алгоритм решения задачи.
5. Структура программы.
6. Текст программы на языке Си.
7. Набор тестов (в форме, соответствующей предметной области задачи).
8. Результаты работы программы.
9. Литература.

1.3. Оформление отчета

1. Отчет (желательно распечатанный на компьютере) оформляется на стандартных листах бумаги форматом А4 (297×210 мм) или листах школьной тетради. Записи ведутся только на одной стороне листа.
2. Титульный лист оформляется по образцу, приведенному на рис. 1.

2. РЕКОМЕНДАЦИИ ПО ВЫБОРУ И ОБОСНОВАНИЮ ФОРМАЛЬНОЙ МОДЕЛИ ЗАДАЧИ

Процесс перевода задачи на язык какой-то математической системы называется формализацией задачи.

Результатом формализации является выбор модели для представления изучаемой предметной области. Поскольку модель является объектом математической системы или теории, то все теоремы и алгоритмы данной теории могут быть применены к выбранной модели. Таким образом, решение конкретной задачи сводится к решению некоторой формальной задачи, являющейся ее аналогом.

Рассмотрим выбор формальной модели на примерах задач по предлагаемым в КР темам.

Министерство образования и науки Российской Федерации

НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

КАФЕДРА ПРИКЛАДНОЙ МАТЕМАТИКИ

Курсовая работа
по практикуму на ЭВМ: структуры данных и алгоритмы

Факультет	прикладной математики и информатики
Группа	ПМ-81
Студент	Иванов А.А.
Преподаватель	Петров В.В.

Новосибирск 2008

Рис. 1

Граф как формальная модель предметной области

Графами описываются организация транспортных систем, электрических цепей, взаимодействие людей, сетей телекоммуникации, структуры программ и алгоритмов, конечные автоматы, синтаксические диаграммы и др. Графы обладают несколькими фундаментальными свойствами, влияющими на выбор структуры для представления графа и на алгоритмы, которые могут быть применены к графу.

Свойства графа представлены в табл. 1.

Таблица 1

№ графа	Свойства графа $G = (V, E)$	Определение свойства	Предметная область
1	Неориентированный	Из того, что ребро $(x, y) \in E$, Следует, что $(y, x) \in E$	Дорожная сеть между городами Отношение «быть родственниками» Граф как математический объект
2	Ориентированный	Из того, что ребро $(x, y) \in E$, не следует, что $(y, x) \in E$	Уличная сеть внутри города, так как существует несколько улиц с односторонним движением Графы программ Отношение «быть ребенком», Иерархические деревья
3	Взвешенный	Каждому ребру или(и) каждой вершине G ставится в соответствие некоторая информация, характерная для данного ребра или данной вершины (эта информация называется весом) (вес вершины не путать с меткой вершины)	Дорожная сеть (вес ребра – расстояние, время поездки, стоимость, расход бензина, вес вершины – стоимость бензина в данном населенном пункте) Сеть аэродромов (вес вершины – количество самолетов, которое может принять аэродром) Автоматные диаграммы (вес дуги графа – символ)
4	Невзвешенный	Все вершины в графе и ребра (дуги) равнозначны	Граф как математический объект Отношение «быть родственниками»
5	Циклический	Существует путь из вершины с меткой i в вершину с меткой i	Граф как математический объект Транспортно-дорожная сеть, Блок-схемы программ, План замка, крепости, шахты

Продолжение табл. 1

№ графа	Свойства графа $G = (V, E)$	Определение свойства	Предметная область
6	Ациклический	Граф, не содержащий циклов (связный неориентированный ациклический граф называется деревом)	Некоторые транспортно-дорожные сети Граф как математический объект Генеалогические «деревья» Сеть работ или событий, в которой ребро (x, y) означает, что событие **y** должно произойти позже события **x** Деревья поиска Деревья вычисления выражений
7	Простой	Не содержит ребер (x, x), не содержит кратных ребер	Граф как математический объект Двудольные графы Все виды деревьев Схемы программ Некоторые транспортные сети (метро – сеть, электрические сети)
8	Псевдограф	Содержит ребра (x, x), или кратные ребра	Автоматные диаграммы Дорожная сеть между городами, сеть воздушного сообщения между городами Уличная сеть внутри города, др.
9	Помеченный	В помеченном графе каждой вершине присвоено уникальное имя, или идентификатор, отличающий ее от остальных вершин	При решении конкретной задачи метки возникают естественным образом: имя города в транспортной сети Состояние в конечных автоматах Имя в генеалогических «деревьях»

Окончание табл. 1

№ графа	Свойства графа $G = (V, E)$	Определение свойства	Предметная область
10	Непомеченный	В непомеченных графах не делается различий между вершинами	Граф как математический объект не имеет размеченных вершин, но граф, как структура данных, должен быть размечен. В качестве меток вершин используется множество натуральных чисел
11	Связный	Имеет одну компоненту связности	Конечный автомат Уличная сеть внутри города Транспортно-дорожная сеть Метро-сеть, Родственники
12	Несвязный	Имеет несколько компонент связности	Граф как математический объект Множество дорог Кланы родственников
13	Привязанный или топологический	Граф называется привязанным, если его вершинам и ребрам было присвоено определенное геометрическое положение Топологическими графами являются клеточные структуры, в которых множество вершин – это клетки, множество ребер явно не перечисляются. Ребра неявным образом определяются из геометрии объекта	Множество точек на плоскости и необходимо найти обход минимальной стоимости Шахматная доска – ребра, соединяющие различные клетки шахматного поля, определяются правилом перемещения шахматной фигуры Клеточное поле – перемещение между клетками по вертикали или горизонтали (ребра соединяют смежные клетки-вершины)

Рассмотрим несколько задач, для которых граф может быть математической моделью предметной области.

Пример 1

В комнате замка N во время завтрака юный принц не поладил со своей гувернанткой. Для того чтобы избежать наказания, принц решил спрятаться в одной из комнат замка. Передвигаясь из комнаты в комнату, принц обязательно закрывал на ключ дверь, через которую он проходил, после чего ни он, ни гувернантка не могли воспользоваться этой дверью. Через любую комнату (в том числе и через ту, в которой находится гувернантка) принц может проходить несколько раз, если это позволяют еще не запертые им двери. Изначально все двери замка не заперты. Между двумя комнатами замка не более одной двери. Общее количество дверей не превосходит 5000. Гувернантка начнет искать принца только после того, как он спрячется. Избежать наказания принц может только в том случае, если гувернантка, проходя через оставшиеся незапертыми двери, не сумеет попасть в комнату, где он спрятался.

Требуется написать программу, которая, проверяет, сможет ли принц спрятаться от гувернантки. Если скрыться удастся, то программа определит любой из возможных путей спасения.

Что же известно о предметной области из условия задачи:

1) план замка. При этом между двумя комнатами замка не более одной двери и изначально все двери замка не заперты;

2) местонахождение принца и гувернантки: комната, где они завтракали;

3) способ избежать наказания: спрятаться от гувернантки в одной из комнат замка. При этом принц может передвигаться по комнатам, связанным дверями. Передвигаясь из комнаты в комнату, принц обязательно закрывал на ключ дверь, через которую он прошел, после чего ни он, ни гувернантка не могли воспользоваться этой дверью. Через любую комнату (в том числе и через ту, в которой находится гувернантка) принц может проходить несколько раз, если это позволяют еще не запертые им двери.

Что требуется получить:

1) сможет ли принц спрятаться от гувернантки?

2) если принцу удалось спрятаться, то путь спасения.

Выбор формальной модели

План замка удобно представить в виде графа, в котором каждой комнате замка сопоставлена вершина графа, а ребро связывает две вершины графа в том случае, если соответствующие этим вершинам комнаты замка имеют общую дверь. Так как комната, где проходил завтрак, известна, то в графе ей должна соответствовать выделенная вершина. Чтобы отобразить способ передвижения принца по замку и отметить пройденный путь, все комнаты замка должны быть помечены.

Отсюда следует, что граф должен быть:

– **неориентированный** – если существует дверь из комнаты *i* в комнату *j*, то можно попасть как из комнаты *i* в комнату *j*, так и из комнаты *j* в комнату *i*.

– **размеченный**, то есть каждой вершине необходимо сопоставить целое число – номер комнаты.

– **простой,** так как между двумя комнатами не более одной двери и не может существовать дверь из *i* комнаты в *i* комнату.

– **невзвешенный** – в задаче отсутствуют какие-либо характеристики перехода из одной комнаты в другую.

Так как из условия задачи не следует, что все комнаты связаны между собой дверями, а принц может перемещаться по замку только в том случае, если существует дверь между комнатами, то в замке имеется некоторое подмножество смежных комнат. Отсюда **граф может иметь несколько компонент связности.**

Таким образом, формальной моделью предметной области поставленной задачи (замок и способ перемещения по нему) является граф с перечисленными выше характеристиками.

Формальная постановка задачи

Для решения задачи достаточно рассмотреть только ту **компоненту связности, которая содержит выделенную вершину.**

Убегая от гувернантки, принц закрывает двери, через которые он прошел. В формальной модели это равносильно удалению ребра, соединяющего две вершины, сопоставленные этим комнатам, вершины при этом не удаляются.

Если существует такая последовательность ребер в графе, удаление которых делает данный граф несвязным, то тогда принц может спрятаться от гувернантки, в противном случае – нет.

Таким образом, дан размеченный неориентированный простой граф, одна из вершин графа выделена. Для связной компоненты графа, содержащей выделенную вершину, нужно найти последовательность ребер начиная с выделенной вершины, удаление которых сделает данный компонент графа несвязным. Если такая последовательность ребер существует, то она и есть путь спасения для принца, иначе пути нет.

Пример 2

Дана система односторонних дорог, определяемая набором пар городов. Каждая такая пара (*i, j*) указывает, что из города *i* можно проехать в город *j*, но это не значит, что можно проехать в обратном направлении. Необходимо определить, можно ли проехать из заданного города *A* в заданный город *B* таким образом, чтобы посетить город *C* и не проезжать ни по какой дороге более одного раза. Если такое путешествие возможно, то перечислите все города, которые можно посетить, проезжая из города *A* в город *B* с обязательным посещением города *C*.

Что же известно о предметной области из условия данной задачи:

1) множество городов;

2) множество односторонних дорог, связывающих некоторые пары городов;

3) выделены три города: город – начало путешествия, город – конец путешествия, город, который необходимо посетить во время путешествия;

4) условие путешествия: не проезжать ни по какой дороге более одного раза.

Что требуется получить:

1) можно ли осуществить такое путешествие,

2) если да, то перечислить все города, которые посетит путешественник.

Выбор формальной модели

Систему односторонних дорог представим в виде графа, в котором каждой вершине графа сопоставлен город. Дуга соединяет вершины графа, если между двумя городами, соответствующими данным вершинам, существует односторонняя дорога. По условию задачи в графе выделены три вершины.

Характеристики графа:

– **размеченный**, так как имя города становится именем вершины**;**

– **ориентированный**, поскольку дороги односторонние;

– **несвязный**, так как в задаче ничего не говориться о том, что все города связаны дорогами;

– **невзвешенный**, поскольку отсутствуют характеристики дорог и вершин, для решения данной задачи эти характеристики не важны;

– **псевдограф (мультиграф)** – один город с другим городом могут связывать несколько дорог.

Формальная постановка задачи

В данной задаче важным признаком предметной области является возможность попасть из города *i* в город *j*, посетив некоторый город *h,* а не способы попадания. Следовательно, при решении задачи, если два города связывают несколько дорог, достаточно **учесть одну дорогу** и тем самым свести исходный **псевдограф к простому графу.**

В заданном простом ориентированном размеченном несвязном и невзвешенном графе с тремя выделенными вершинами определить, существует ли путь из одной выделенной вершины в другую выделенную вершину, проходящий через третью выделенную вершину; не проезжать ни по какой дороге более одного раза. Если такой путь существует, то перечислите все вершины, лежащие на этом пути.

Пример 3

Клуб юных хакеров организовал на своем сайте форум. Форум имеет следующую структуру: каждое сообщение либо начинает новую тему, либо является ответом на какое-либо предыдущее сообщение и принадлежит той же теме. После нескольких месяцев использования форума юных хакеров заинтересовало, какая тема на их форуме наиболее популярна.

Помогите им выяснить это. Сообщения идут в хронологическом порядке и нумеруются начиная с единицы. Описание сообщения, которое представляет собой начало новой темы, состоит из трех строк. Первая строка содержит число 0. Вторая строка содержит название темы. Длина названия не превышает 30 символов. Третья строка содержит текст сообщения.

Описание сообщения, которое является ответом на другое сообщение, состоит из двух строк. Первая строка содержит целое число – номер сообщения, ответом на которое оно является. Вторая строка со-

держит текст сообщения. Ответ всегда появляется позже, чем сообщение, на которое он ссылается. Длина всех сообщений не превышает 100 символов.

Что же известно о предметной области из описания данной задачи?

Дано: 1) темы, обсуждаемые на форуме (выделены специальным символом), 2) сообщения двух видов: связанные непосредственно с обсуждаемой темой и сообщения-ответы (ответы могут ссылаться как на саму тему, так и на ответ, связанный с какой-то темой).

Все сообщения пронумерованы в порядке их появления на форуме;

Ответы всегда появляются позже, чем сообщение, на которое они ссылаются.

Что же является результатом? Имя темы, обсуждаемой на форуме наибольшее число раз.

Выбор формальной модели

Каждому сообщению на форуме поставим в соответствие вершину графа. Вершины, соответствующие сообщениям, поднимающим новую тему, должны быть выделены. Имя вершины совпадает с номером сообщения. Две вершины с номерами i и j соединяются дугой (ij), если сообщение с номером i ссылается на сообщение с номером j, при этом $i > j$.

Характеристики графа:

– **ориентированный**, так как более поздние сообщения ссылаются на предшествующие сообщения;

– **размеченный,** поскольку каждой вершине приписан номер сообщения в порядке его поступления;

– **несвязный,** так как ответ может быть откликом только на одно сообщение, существуют сообщения-темы, которые не ссылается ни на какие другие сообщения. Таким образом, количество компонент связности в этой модели будет равно количеству тем, обсуждаемых на форуме;

– **нециклический**, так как все ссылки упорядочены во времени. Ссылаться можно только на предыдущее сообщение или вообще не на что не ссылаться;

– **простой**, так как ответ может быть откликом только на одно сообщение, ссылаться можно только на предыдущее сообщение;

– **взвешенный**, поскольку каждой вершине сопоставлено сообщение.

Формальная постановка задачи

Анализируя поставленную задачу, можно заметить, что сами сообщения не играют никакой роли в решении задачи. Следовательно, текст сообщений можно не учитывать и упростить формальную модель, считая граф не взвешенным.

Кроме того, учитывая, что корневая вершина выделена и направление дуг не влияет на решение поставленной задачи, дуги можно заменить ребрами. Тогда уточненной формальной моделью предметной области будет «лес», состоящий из множества размеченных неориентированных деревьев с выделенными корневыми вершинами.

Таким образом, в заданном «лесе», определить компоненту связности с максимальным числом вершин. Имя корневой вершины найденной компоненты связности есть результат.

В некоторых задачах предметная область уже формализована, т. е. совпадает с некоторой математической реальностью. В этом случае под формализацией задачи понимается выбор характеристик данной формальной модели с учетом поставленной задачи.

Пример 4

Определить диаметр графа.

Исходя из определения диаметра графа уточнить характеристики графа, для которого поставленная задача имеет решение.

Диаметр графа – это максимум расстояния между вершинами для всех пар вершин.

Расстояние между вершинами – наименьшее число ребер, которые необходимо пройти, чтобы добраться из одной вершины в другую.

Выбор формальной модели

Исходя из определения диаметра графа его можно определить только для связного графа. Если граф несвязный, то решения задачи не существует. Таким образом, заданный граф **неориентированный** и может состоять из нескольких компонент связности.

Граф **необходимо разметить**, т. е. приписать каждой вершине некоторое натуральное число, например начиная с 1.

Граф **простой**, так как наличие кратных ребер и петель не изменяет диаметр графа.

Граф **невзвешенный**, так как абстрактным графам не ставятся в соответствие весовые функции.

Формальная постановка задачи

Проверить связность графа. Если граф связный, то найти максимум среди расстояний между всеми парами вершин простого неориентированного связного невзвешенного графа.

Кроме графов можно выделить несколько формальных моделей, которые могут служить аналогом предметной области. Некоторые из них приведены в табл. 2.

Таблица 2

№ графа	Предметная область	Формальные модели
1	Грамматика	$G = <T, N, S, P>$ T – конечное множество терминальных символов N – конечное множество нетерминальных символов $S \in N$ $P = \{\alpha \rightarrow \beta \mid \alpha, \beta \in (N \cup T)^+\}$
2	Язык	Множество слов в некотором алфавите $L = \{a_1 a_2 \ldots a_n \mid n \geq 1, a_i \in A\}$ $A = \{b_1, b_2, \ldots, b_k\}$
3	Автоматная диаграмма	Ориентированный связный взвешенный псевдограф с двумя выделенными вершинами
4	Множество логических формул	Формальная грамматика: <логическая формула> ::= <конъюнкция> {OR<конъюнкция>}* <конъюнкция>::= <множитель>{AND<множитель>}* <>::= {NOT символ пробела, буква (<логическая формула>) TRUEсимвол пробела FALSEсимвол пробела}
5	Логическая формула	ДНФ, ориентированное двоичное размеченное дерево вычислений, обратная польская запись, строка символов, построенная по определенным правилам (см. № 4)
6	Логический фрагмент	Ориентированный связный размеченный взвешенный не простой граф (есть петли) с одной выделенной вершиной

Продолжение табл. 2

№ графа	Предметная область	Формальные модели
7	Множество простых выражений	Формальная грамматика: <простое выражение>::=<терм>{{+,–}<терм>}* <терм>::=<множитель>{*<множитель>}* <>::={ <целое без знака> буква символ пробела (<простое выражение>)} <целое без знака>::= <цифра>$^+$ <цифра>::={0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
8	Выражение	Прямая польская запись, обратная польская запись, ориентированное двоичное размеченное дерево вычислений, строка символов, построенная по определенным правилам (см. № 7)
9	Полином	Формальная грамматика [2; с. 109], прямая польская запись, обратная польская запись, ориентированное двоичное размеченное дерево вычислений, строка символов, построенная по определенным правилам, одномерный массив коэффициентов (для полиномов от одной переменной)
10	Множество отрезков	$\{(x_i, y_i;\ x_j, y_j)\mid\ x_i, y_i,\ x_j, y_j \in R\}$
11	Множество пересекающихся отрезков	Неориентированный размеченный простой несвязный граф, $\{(x_i, y_i;\ x_j, y_j)\mid\ x_i, y_i,\ x_j, y_j \in R\}$
12	Множество точек в n-мерном пространстве	$\{(x_1, x_2, \ldots, x_n) \mid x_1, x_2, \ldots, x_n \in R\}$
13	Множество материальных точек в n-мерном пространстве	$\{(x_1, x_2, \ldots, x_n; m) \mid x_1, x_2, \ldots, x_n, m \in R\}$
14	Множество прямых	$\{ax + by + c = 0\mid a, b, c \in R\}$
15	Множество окружностей	$\{(x - x_1)^2 + (y - y_1)^2 = r^2 \mid\ x_1, y_1, r \in R\}$
16	Множество геометрических фигур	Формальная грамматика [2; с. 112], строка символов, построенная по определенным правилам

Окончание табл.2

№ графа	Предметная область	Формальные модели
17	Комплект домино	Множество $\{n_1, n_2 \mid n_1, n_2 \in \{0, 1, 2, 3, 4, 5, 6\}\}$
18	Лабиринт	Неориентированный несвязный простой размеченный невзвешенный граф, клеточное поле с выделенными клетками (топологический граф)
19	Клеточное поле	Неориентированный размеченный связный простой невзвешенный (взвешенный) топологический граф, в котором каждой клетке поля сопоставлена вершина графа, смежная с четырьмя вершинами по вертикали и горизонтали
20	Шахматная доска	Неориентированный размеченный связный простой невзвешенный топологический двудольный граф

Пример 6

Правило грамматики G называется бесполезным, если его нельзя применить ни в одном выводе слов грамматики G. Найти и удалить все бесполезные правила заданной грамматики.

Выбор формальной модели

Поскольку в условии задачи не конкретизирована грамматика, то в качестве модели будем рассматривать формальную грамматику в общей форме: $G = <T, N, S, P>$.

Для решения задачи должны быть определены множества T, N, P и $S \in N$. На правила вывода (множество P) не накладывается никаких ограничений.

Формальная постановка задачи

Формальная постановка: по заданной грамматике $G = <T, N, S, P>$ построить грамматику, эквивалентную исходной $G' = <T', N', S', P'>$, в которой:

1) $T' = \{a \mid \forall\, a \in T' \; S' \xrightarrow{*} a, a \in T\}$

2) $N' = \{x \mid \forall\, x \in N' \; S' \xrightarrow{*} x \text{ и } x \xrightarrow{*} \alpha,$
где $\alpha \in T^*\}$;

3) $S' = S$

4) $\forall x \in N' S' \xrightarrow{*} x$ и $x \xrightarrow{*} \alpha$,

где $\alpha \in T'^{*}$;

$P' = \{\alpha \underset{*}{\rightarrow} \beta \mid \alpha \in (N' \cup T')^{+}, \beta \in (N' \cup T')^{*}\}$.

Символ $\xrightarrow{*}$ означает выводимость.

Пример 7

Найти кратчайший путь передвижения коня по шахматной доске, соединяющий два заданных поля доски.

Выбор формальной модели

Формальной моделью шахматной доски в данной задаче можно считать граф.

Каждой клетке доски ставится в соответствие вершина графа. Две вершины *i* и *j* соединяются ребром, если шахматная фигура «конь» стоит на клетке, сопоставленной вершине *i*, и за один ход данная шахматная фигура может попасть в клетку, которой сопоставлена вершина *j*. Граф, заданный таким образом, имеет следующие свойства:

– **неориентированный**, так как если «конь» может попасть из клетки *i* в клетку *j*, то он может попасть из клетки *j* в клетку *i*;

– **размеченный**, так как каждая вершина графа имеет метку – *ав*, где $\boldsymbol{b} \in \{ a, b, c, d, e, f, g, h \}$, $a \in \{ 1, 2, 3, 4, 5, 6, 7, 8 \}$;

– **невзвешенный**, так как все ходы данной фигуры равнозначны;

– **с двумя выделенными вершинами** – старт и финиш;

– **связный**, так как данной фигурой можно обойти все клетки шахматной доски;

– **простой**, поскольку нет кратных ребер по построению и нет петель по правилам игры, «конь» не может за один ход перейти из клетки *i* в клетку *i*;

– **топологический**, так как множество ребер не перечисляется явно, а определяется правилами передвижения фигуры по шахматной доске. Это определяет форму внутреннего представления графа – ребра графа хранить не нужно.

Формальная постановка задачи

Так как требуется найти кратчайшее расстояние между двумя клетками, то в неориентированном размеченном невзвешенном связном простом топологическом графе найти кратчайший путь из одной выделенной вершины в другую выделенную вершину.

Замечание. Как вытекает из табл. 2, некоторые предметные области можно формализовать по-разному. Какую формальную модель выбрать, зависит от поставленной задачи или от предпочтений разработчика проекта. В любом случае выбор должен быть обоснован.

Например, в задаче «Построить синтаксический анализатор для определения, является ли заданная строка символов логической формулой», моделью логической формулы будет формальная грамматика.

В задаче «Вычислить значение заданной логической формулы, не содержащей вхождения переменных», моделью логической формулы может быть либо обратная польская запись логического выражения, либо двоичное дерево вычислений, либо прямая польская запись.

В задаче «По заданной логической формуле построить эквивалентную логическую формулу, в которой знак отрицания встречается только перед переменными», моделью логической формулы может быть строка символов, обратная польская запись, прямая польская запись, дерево вычислений.

В задачах преобразования формул на основе множества эквивалентных преобразований в качестве формальной модели лучше выбрать дерево вычислений.

3. РЕКОМЕНДАЦИИ ПО АНАЛИЗУ ЗАДАЧИ И РАЗРАБОТКЕ АЛГОРИТМА

Анализ задачи – первый этап процесса решения. Условие задачи, как правило, задается в словесной формулировке. На основе словесной формулировки задачи студент должен выбрать объекты (переменные), подлежащие определению, найти и записать ограничения и связи между объектами задачи, в совокупности образующие ее математическую модель.

Таким образом, анализ задачи позволяет установить, что является входом и выходом будущего алгоритма, выделить основные необходимые для решения задачи отношения между входными и выходными объектами и их компонентами и, как следствие этого, выработать подход к построению алгоритма. Результатом этого этапа должна быть формулировка в самом общем виде того, что должен делать алгоритм, чтобы переработать входные данные в выходные. Формулировать необходимо на языке математики в терминах предметной области задачи (например, теории графов).

В отчете данный пункт (анализ задачи) включает:

– определение исходных данных и результатов решения задачи. Отвечая на вопрос, что является исходными данными задачи, требуется перечислить величины (объекты) задачи, значения которых необходимо знать, чтобы ее решить. Каждая величина задается своим именем и диапазоном значений. Диапазон значений определяется структурой и размером данных. В условиях задач, как правило, не указываются ограничения на размер входных данных, поэтому при выборе ограничений студент должен не сужать, а максимально расширять класс входных данных. Например, если входной величиной является текст, то определить его структуру можно как последовательность элементов (в свою очередь аналогично определяется элемент. Например, если элемент текста это слово, то оно может быть определено как последовательность каких-то символов). Размер текста определяется как максимально допустимая его длина. Также необходимо указать и минимальную длину текста.

Отвечая на вопрос, что является результатами решения задачи (выходными данными), требуется перечислить величины, значения которых необходимо вычислить, и определить их аналогично входным данным. Поскольку задача не всегда может быть решена (или решена, но не полно), то среди результатов задачи должен быть такой, который говорит об этом.

– математическую (формальную) постановку задачи. Математическая постановка задачи включает формальное определение входных и выходных данных (как математических объектов) и математически строгого описания основных необходимых отношений (функциональных зависимостей) между входными данными и результатами (см. п. 2).

– выбор и описание способа (метода) решения задачи. Способ решения задачи должен основываться на описанных в предыдущем пункте отношениях между входными данными и результатами и показывать, как вычислить результаты. При выборе способа решения необходимо использовать известные для данной предметной области методы и алгоритмы. Способ решения должен быть описан в виде общей спецификации на языке математики и предметной области задачи.

– выделение основных подзадач, которые надо решить, чтобы достичь результата. Разработка алгоритма представляет собой задачу на проектирование. Разработка алгоритма существенно усложняется, если разработчик не придерживается с самого начала некоторой дисципли-

ны (метода) проектирования. Для выполнения курсовой работы такой дисциплиной является метод пошаговой разработки (декомпозиции). Суть метода состоит в том, что алгоритм разрабатывается «сверху вниз», начиная с его спецификации, полученной в результате анализа задачи. Затем на последующих этапах решение постепенно детализируется. Таким образом, получается последовательность все более детальных спецификаций алгоритма, приближающихся к окончательной версии программы.

Этот подход позволяет разбить решение задачи на части (модули), каждая из которых решает самостоятельную подзадачу. В результате решение сложной задачи разбивается на решение более простых подзадач, которые можно решать независимо друг от друга. Решение каждой подзадачи можно оформить в виде отдельных подпрограмм. Связи по управлению между ними осуществляются посредством обращений к ним, передача данных от одного модуля к другому производится через параметры.

На разработку алгоритма существенное влияние оказывают:

1) средства, предоставляемые тем языком, на котором алгоритм будет запрограммирован;

2) структура данных, которая будет использована для отображения данных задачи.

Разрабатывая алгоритм, необходимо на каждом шаге декомпозиции проверять его правильность. Строгое математическое доказательство правильности работы алгоритма – трудная задача. Проверка правильности программы при наборе тестов не дает основания считать, что алгоритм всегда будет работать правильно, так как различных комбинаций входных данных практически бесконечно много. Поэтому реальным подходом к обоснованию алгоритма является его детальное построение в процессе пошаговой разработки. Чтобы получить правильный алгоритм, необходимо следить за строгостью детализации его шагов в ходе построения, сопровождая их логическими рассуждениями.

На каждом шаге декомпозиции алгоритм необходимо описывать. Важно понимать, что это описание предназначено для человека. По этому описанию человек должен получить представление об общей структуре алгоритма, смысле его шагов и их логической взаимосвязи. Сохранение достаточно высокого уровня описания алгоритма облегчает его обоснование. Шаги алгоритма должны описываться в терминах тех объектов и отношений между ними, о которых идет речь в форму-

лировке задачи. Например, для задачи из теории графов шаги алгоритма следует описывать как действия над вершинами, ребрами, подграфами и т. п. Но не должно быть работы с внутренним представлением графа, например с матрицей смежности. (Это не исключает использования математической и другой удобной символики.)

Степень детализации при описании алгоритма определяется так, чтобы было ясно, какие действия надо выполнять, реализуя выбранный метод решения, и в чем заключаются основные проблемы данной задачи. Поэтому в отчете по КР необходимо привести несколько спецификаций алгоритма, позволяющих раскрыть решение основных проблем задачи (например, укрупненный алгоритм решения всей задачи и более детальные алгоритмы решения основных подзадач). Алгоритм может быть описан на языке блок-схем или псевдоязыке (псевдоСи).

4. РЕКОМЕНДАЦИИ ПО ВЫБОРУ ПРЕДСТАВЛЕНИЯ ДАННЫХ

Данные, необходимые для решения задачи, делятся на входные, промежуточные и выходные (результаты решения задачи). Специфика входных и выходных данных состоит в том, что ими оперирует не только алгоритм, но и пользователь программы. Поэтому различают внутреннее и внешнее представления этих данных.

С внешним представлением работает пользователь. Поэтому оно должно быть понятно, максимально удобно и естественно (привычно) для пользователя, чтобы он мог достаточно легко подготовить данные для ввода и интерпретировать результат по выходным данным. (Человек не должен при подготовке и интерпретации данных их структурировать, сортировать или кодировать.) При выборе формы внешнего представления необходимо опираться на предметную область задачи. Например, в задаче по теме «Системы дорог» входные данные – это множество городов и множество дорог. Внешнее представление множества городов может быть представлено как список городов – последовательность слов – названий этих городов, а множество дорог как список дорог – последовательность пар слов: первое слово – название города, являющегося началом дороги, второе слово – название города – конец дороги. Результатом решения такой задачи может быть путь из одного города в другой. Внешнее представление этого пути также

может быть отображено как список городов – последовательность слов – названий городов, через которые проходить путь.

Если выбор внешнего представления данных определяется естественностью для человека и обычно не зависит от алгоритма решения задачи, то внутреннее представление должно обеспечивать возможность построения эффективного алгоритма решения задачи. Поэтому внутреннее представление отличается от внешнего. Структура алгоритма и используемые структуры для представления данных должны разрабатываться одновременного в ходе шагов последовательного уточнения алгоритма. Надлежащий выбор внутреннего представления данных, как правило, обеспечивает более эффективный способ обращения к данным (например, при использовании массивов прямой способ обращения) и меньший объем используемой памяти (например, если в задачах на графах использовать в качестве внутреннего представления не множество вершин и множество ребер, а матрицу смежности). Перевод из внешнего представления во внутреннее (и наоборот) – существенная часть процесса решения задачи.

Выбор форм представления исходных данных и результатов задачи необходимо обосновать, отметив их достоинства и недостатки.

5. РЕКОМЕНДАЦИИ ПО РАЗРАБОТКЕ ПРОГРАММЫ

Написание программы это, по сути, продолжение разработки алгоритма с дальнейшей детализацией его шагов и фиксированием их на языке программирования. Для улучшения качества программы необходимо использовать структурное и модульное программирование. Язык С ориентирован на структурное программирование (операторы языка имеют один вход и один выход). Поэтому правильное использование его управляющих структур способствует написанию программы, хорошо структурированной, наглядной, легко читаемой и понимаемой не только составителем, но и другими программистами, в том числе преподавателем.

Принципы модульного программирования являются поддержкой пошаговой разработки алгоритма. Полученные при пошаговой разработке подзадачи необходимо оформить в виде отдельных модулей (подпрограмм). Каждый модуль может программироваться независимо от других, а программирование всей задачи в терминах подзадач может быть осуществлено без предварительных знаний о внутренних

действиях модулей этих подзадач. Следует стремиться к тому, чтобы каждый модуль имел один вход и один выход и выполнял самостоятельное действие, которое можно описать одной фразой. Необходимо стремиться к тому, чтобы модули были максимально независимыми, в частности не имели общих переменных (глобальных объектов), за исключением модулей, реализующих переменные сложных типов. Операции по вводу-выводу не должны быть рассеяны по всей программе, их нужно отделять от вычислительных частей программы.

Проектирование модуля следует начинать с формализованного и строгого описания его входных данных, выходных и действий. Принятие основных решений по управлению модулями нужно выносить на максимально высокий уровень в их иерархии – обычно это главный модуль. Таким образом, главный модуль можно рассматривать как краткое описание решения всей задачи. Модуль низкого уровня должен передавать модулю более высокого уровня результаты своих действий, в том числе и в случае, если он не может успешно выполнить те функции, которых ждет от него вызвавший его модуль. На основании этих данных модуль высокого уровня принимает решение о дальнейших действиях.

В п. 5 отчета описываются составные части программы, т. е. функции, реализующие отдельные действия алгоритма, и их взаимосвязь. Описание функции включает назначение функции, ее прототип, назначение и описание параметров функции (описание параметра включает его имя, назначение и тип).

Взаимосвязь функций можно представить в виде схемы, отображающей связь между функциями по управлению.

Для повышения наглядности программы рекомендуется:

– снабжать текст программы комментариями, кратко и точно поясняющими основное назначение фрагмента программы (не следует делать их слишком длинными и частыми);

– использовать содержательно осмысленные (но не чрезвычайно длинные) имена переменных, функций;

– при расположении операторов, реализующих базовые управляющие структуры, использовать отступы на 2-3 позиции для выделения как внутренней конструкции управляющей структуры, так и отделения одной управляющей структуры от другой (это позволяет структурировать сам текст программы);

– разумно ограничивать длину строк в программе, что обеспечит более удобную дальнейшую отладку и исправление ошибок.

6. РЕКОМЕНДАЦИИ ПО РАЗРАБОТКЕ НАБОРА ТЕСТОВ

Тест представляет собой контрольный пример, на котором будет проверяться правильность работы программы. Тест задает значения не только входных данных задачи, но и всех результатов, которые должна вычислить программа на этих данных.

Набор тестов должен быть необходимым и достаточным, включающим частные и особые случаи решения, а также позволять проверку (на одном из тестов) каждого перехода в программе. При выполнении КР входные данные считаем корректными. Как правило, сложная задача не всегда имеет решение, поэтому должны быть отражены случаи, когда решение есть и когда его нет. Когда в задаче используются сложные данные, то возможно много вариантов их значений. Например, если входные данные представляют множество точек на плоскости, то это множество может быть пустым или непустым и состоящим только из одной или нескольких точек. В случае пустого множества задача, как правило, не имеет решения. В случае множества, состоящего из одной точки, решение может быть получено более простым способом (частное решение), чем для множества, число точек в котором больше единицы. Таким образом, одна из задач, которая решается при выборе набора тестов, состоит в том, чтобы проверить адекватность разработанного алгоритма возможному разнообразию входных данных.

Если в программе есть ветвления (циклы), то должны быть тесты для прохождения каждой из ветвей (условий управления циклом).

Другой важный вопрос, решаемый при выборе тестов, – это количественные ограничения размеров используемых в программе данных (для корректно заданных данных это ограничения сверху). Обязательно должны быть тесты, в которых проверяется поведение программы при граничных значениях. Но следует отказаться от тестов, для которых время решения не укладывается в разумные рамки (для КР это не более 20 мин). Например, если в задаче нужен перебор всех подмножеств заданного множества, то при мощности множества n, равной 20, имеется 2^n различных подмножеств. Такой тест требует значительного времени.

В отчете набор тестов желательно описать в виде таблицы, содержащей исходные данные, результаты и назначение для каждого теста. Необходимо для наглядного представления теста (если это возможно) иллюстрировать его графически.

7. ПРИМЕР РЕШЕНИЯ ЗАДАЧИ И ОФОРМЛЕНИЯ ОТЧЕТА ПО КР

1. Условие задачи

Множество из N городов связано односторонними дорогами. Определить, есть ли среди городов такой, из которого можно добраться до каждого из остальных, проезжая не более 100 км.

2. Анализ задачи

2.1. Исходные данные задачи: n – количество городов, $n \in N$, $n \le 20$,

S – система дорог, $S = \{ (\text{name1}_i , \text{name2}_i , \text{weight}_i) \mid \text{name1}_i , \text{name2}_i \in$ город, $\text{weight}_i \in R, i = 1, \ldots, n \}$,

$$\text{город} = \{g_j \mid g_j \in \{\text{подмножество литер-букв}\}, j = 1, \ldots, 20\},$$

где name1_i – город, задающий начало дороги, name2_i – город, задающий конец дороги, weight_i – длина дороги;

2.2. Результат: result $\in \{0;1\}$, town $\in$ город; если искомый город существует, то result = 1, иначе result = 0, town – искомый город.

2.3. Решение. Математическая модель системы односторонних дорог – это ориентированный взвешенный мультиграф $G = (V, E)$ с весовой функцией w: $E \to R$, причем $w(u, v) \ge 0$ для всех $(u, v) \in E$. При этом вершины соответствуют городам, а дуги – односторонним дорогам. Метка вершины – название города. Дуга (u, v) ведет от вершины u, соответствующей городу–началу дороги, в вершину v, соответствующую городу, являющемуся концом дороги. Вес дуги – это длина дороги. Для того чтобы определить, есть ли в системе дорог S город, от которого можно добраться до каждого из остальных городов, надо в ориентированном графе G найти вершину u, от которой можно добраться до всех остальныех вершин, т. е. проверить, что в орграфе существуют пути между вершиной u и каждой из остальных вершин. Поскольку условие задачи содержит еще и ограничения на длину пути, то определения существования пути между вершинами недостаточно. Длина пути, соединяющего вершину u и любую из других вершин, – это сумма весов дуг графа, т. е. сумма длин дорог этого пути. При этом длина любого пути от вершины u до остальных вершин не должна превышать 100 км. Таким образом, чтобы решить задачу, надо не только искать вершину, от которой достижимы все другие, но и определять длины путей (а значит, и сами пути). Для поиска такой вершины можно

последовательно рассматривать каждую вершину орграфа и для текущей вершины искать пути (вычисляя их длины) от нее до остальных вершин. Просмотр продолжать, пока не будет обнаружена искомая вершина или рассмотрены все вершины орграфа. Вершин, удовлетворяющих условию задачи, может быть несколько. Для ответа на вопрос, существует ли город, от которого можно добраться до каждого из остальных, достаточно найти любую из них (первую обнаруженную в порядке последовательного просмотра). Если просмотрены все вершины, то искомой вершины в графе нет. Так как в задаче существует ограничение, накладываемое на длину пути, то достаточно искать кратчайшие пути длиною не больше 100 км. Если в графе нет кратчайшего пути длиною не больше 100 км, то нет и никакого другого, удовлетворяющего ограничению задачи. Учитывая это, поиск кратчайших путей от текущей вершины можно завершить, если расстояние от нее до очередной вершины превышает ограничение на его длину. Кратчайших путей от одной вершины до другой может быть несколько, для решения данной задачи достаточно найти один (любой из возможных).

Если орграф состоит из нескольких компонент связности, то кратчайший путь от текущей вершины до каждой из остальных не будет найден.

Формальная постановка задачи

В ориентированном взвешенном мультиграфе найти вершину, от которой существуют кратчайшие пути длиной не более 100 до всех остальных вершин.

Таким образом, решение задачи заключается в поочередном рассмотрении вершин орграфа и поиске кратчайших путей от очередной вершины до всех остальных.

Начальная спецификация алгоритма решения:

при v = 1-й город S, t = город не найден

повторять

если есть кратчайшие пути длиною ≤ 100 от города v до остальных, то t = город найден

v = следующий город S, пока t = город не найден **и** не все города просмотрели,

если t = город найден, то result =1, town= v, иначе result =0

Для нахождения кратчайших путей от одной вершины будем использовать алгоритм Дейкстры. Алгоритм Дейкстры решает задачу о кратчайших путях из одной вершины *k* для взвешенного ориентированного графа $G(V, E)$, в котором веса всех ребер неотрицательны ($w(u, v) \geq 0$ для всех $(u, v) \in E$).

Если система односторонних дорог содержит только один город, то искомого города нет, result = 0. Если система односторонних дорог содержит только два города, то задача сводится к проверке длины дороги (дорог) между ними. Если в системе есть дорога длиною не более 100 км, то искомый город существует и result = 1, иначе нет и result = 0.

Если в системе дорог есть города, до которых нет дорог (в орграфе есть изолированные вершины), то искомого города нет, result = 0.

Основные подзадачи:

1) поиск кратчайших путей от выделенной вершины до остальных с учетом ограничения на длину пути,

2) ввод системы дорог.

3. Структуры данных, используемых для представления исходных данных и результатов задачи

3.1. Входные данные

Внешнее представление системы односторонних дорог:

<система – дорог> : : = *<количество – городов> <список – дорог>*

<список – дорог> : : = пробел | пробел <дорога> <список – дорог>

<дорога> : : = *<откуда> пробел <куда> пробел <длина – дороги>*

<откуда> : : = *<куда>* : : = *<название>*

<название> : : = *<элемент> | <элемент> <название>*

<длина – дороги> : : = *вещественное – число*

<количество – городов> : : = *натуральное – число*

<элемент> : : = *символ из подмножества латинских букв*

Если в системе дорог есть изолированные города, то они во внешнем представлении системы дорог не отображаются. Это позволяет после ввода системы дорог определить, связен граф или нет.

Внутреннее представление системы дорог:

Для алгоритма Дейкстры хорошим представлением орграфа является представление списками смежности. Список смежности дает компактное представление для разреженных графов – тех, у которых множество ребер много меньше множества вершин, что характерно для системы дорог. Так как решение задачи сводится к отысканию кратчайших путей, то из всех кратных дуг мультиграфа, связывающих две вершины, достаточно в списке смежности хранить дугу с наименьшим весом. Чтобы не усложнять

списки смежности, будем вместо названия города помечать вершину номером. Вершины орграфа (а значит, и города) пронумеруем начиная с нуля. Список смежных вершин представим линейным односвязанным списком. Это позволяет не вносить ограничений на число дорог между двумя городами. Названия городов будем представлять также списком. Так как число городов ограничено, то списки смежных вершин и список названий городов будем представлять массивами размером n, $n \leq 20$ (рис. 2):

– структура смежности (списки смежных вершин графа)

```
listinc * adj [n];
struct listinc
{
  int town;
  float weight;
  listinc *next;
  }
```

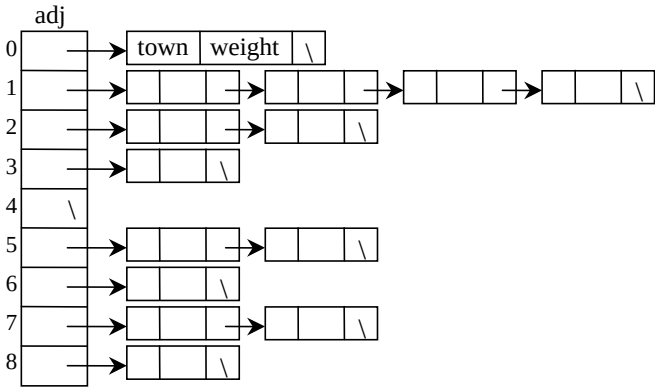


Рис. 2

– список названий городов

```
char z[n][20];
```

3.2. Выходные данные

Внешнее представление:

result – текстовое сообщение «город есть» или «города нет»

town $\in$ *название*

Внутреннее представление:

```
int result, town;
```

4. Укрупненный алгоритм решения задачи

4.1. Укрупненный алгоритм решения задачи

```
{  ввод числа городов n;
   if (n > 1)
     {ввод информации о системе дорог и создание структуры смежности орграфа;
      if (n = 2)
          if (есть дорога между городом 0 и городом 1 и ее длина ≤ 100)
              {town ← город–начало дороги; result ← «город есть»}
            else  result ← «города нет»;
          else { flag ←город не найден;
              while (flag =  город не найден и не все города просмотрели)
                  if (есть кратчайшие пути длиною ≤ 100 от очередного города v)
                          flag ←город найден;
                if (flag = город найден)
                    {result ← «город есть»; town ←  v; }
                   else  result ← «города нет»;
               }
     else result ← «города нет»;
     }
 }
```

Так как основная проблема решения задачи это отыскание кратчайших путей, то приведем более детальное описание алгоритма (спецификацию) поиска кратчайших путей от выделенной вершины.

4.2. Укрупненный алгоритм поиска кратчайших путей от выделенной вершины k

```
{ Q = ∅ ;
flag ← 1;
    for ( i от 0 до n–1 )
         d [i] ← 1000;
         d[k]← 0;
    for ( i от  0 до n-1 )
          в_очередь (Q, i);
    while (Q ≠ ∅ и flag = 1)
        { u ← из_очереди_вершину_с_мин.расстоянием (Q);
          if  ( d[u] > 100 ) flag ← 0;
          else
              while ( v ∈adj [u] )
```

```
                { if ( d[v] > d [u] + weight (u, v) )
                        d [v]← d [u] + weight (u, v);
                    }
            }
        if ( flag = 1)
            { i ← 0; while ( flag = 1 и  i < N )
                  else i ← i + 1;}  if ( d[i] > 100 )  flag ← 0;
        }
```

4.3. Укрупненный алгоритм ввода системы дорог

```
{ список_названий_городов = ∅;
  while ( не конец файла )
  { ввод названия города i;
    if ( города i в списке_названий_городов нет )
       добавить в список_названий_городов город i;
    ввод названия города j;
    if ( города j в списке_названий_городов нет )
       добавить в список_названий_городов город j;
    ввод длины дороги m между i и j городами;
    if ( дороги i j нет в списке смежности города i )
     добавить к списку смежности города i город j с длиной дороги m;
    else
     if ( m < длины дороги между i и j городами в списке смежности
города i )
          длина дороги в списке смежности города i  ←  m;
  }
  if ( число городов в списке_названий_городов < числа городов в сис-
теме дорог )
     result ← 0;
  else
     result ← 1;
  }
```

5. Структура программы

Текст программы разбит на два модуля.

Модуль 1 содержит функции решения основных подзадач и определения входных и выходных данных.

Модуль 2 содержит вспомогательные функции, реализующие абстрактный тип очередь.

5.1. Состав модуля 1

Главная функция main:

– назначение:

определение входных и выходных данных задачи, ввод входных данных, последовательный просмотр городов и поиск искомого города, вывод результатов.

– прототип функции: void main ( )

Функция input:

– назначение:

ввод системы дорог. Функция возвращает число 1, если система дорог есть связный граф; возвращает число 0, если в системе есть изолированные города.

– прототип функции:

int input (listinc *adj [], int N, FILE *p)

– параметры:

adj – (выходной параметр) массив указателей на списки смежности вершин графа, задающие систему дорог;

N – (входной параметр) число городов в системе дорог;

p – (входной параметр) файловая переменная, посредством которой устанавливается связь с файлом с входными данными.

Функция search_ver:

– назначение:

поиск кратчайших путей от выделенной вершины по алгоритму Дейкстры. Если существуют кратчайшие пути от выделенной вершины до всех остальных вершин длиною не более 100, то функция возвращает число 1, иначе число нуль.

–прототип функции:

int search_ver (int k, int N, listinc *adj [])

–параметры:

k – (входной параметр) выделенная вершина графа, для которой осуществляется поиск кратчайших путей;

adj – (входной параметр) массив указателей на списки смежности вершин графа;

N – (входной параметр) число городов в системе дорог.

5.2. Состав модуля 2

Функция in_line:

– назначение:

добавление элемента в очередь. Элемент очереди – вершина и расстояние до нее.

– прототип функции:

void in_line (line **s, int v, float d [])

– параметры:

s – (входной и выходной параметр) очередь;

v – (входной параметр) вершина;

d – (входной параметр) массив расстояний от каждой вершины до остальных.

Функция change:

– назначение:

изменение расстояния (приоритета) у элемента очереди.

–прототип функции:

void change (line *s, int N, float d [])

–параметры:

s – (входной параметр) очередь;

N – (входной параметр) число вершин (городов) графа;

d – (входной параметр) массив расстояний от каждой вершины до остальных.

Функция out_line_min:

– назначение:

взятие из непустой очереди элемента (вершины) с минимальным расстоянием до нее. Функция возвращает вершину с минимальным расстоянием до нее.

– прототип:

int out_line_min (line **s)

– параметры:

s – (входной и выходной параметр) очередь.

5.3. Структура программы по управлению

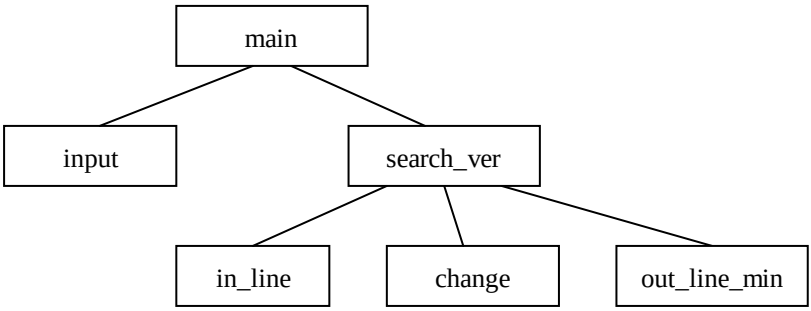


Рис. 3

6. Текст программы на языке Си

Замечание. В приведенной программе для наглядности результаты выдаются в виде текстового сообщения. В студенческих отчетах результаты должны быть выведены в пользовательский файл в удобной для использования форме.

MODUL_1.CPP

```
//Программа поиска города, от которого достижимы все города
    #include <stdio.h>
    #include <string.h>
    #include "MODYL_2.CPP"
    #define n  20                          // максимальное число городов
    char z[n][20];                         //массив названий городов
       struct listinc
        {
          int town;
          float weight;
          listinc *next;
         }

//функция ввода информации о системе дорог
int input(listinc *adj [],int N,FILE *p)
{
    char city[20];
listinc *b;
int r,i,g,j=0;
float m;
for(i=0;i<N;i++)
  z[i][0]='\0';
while(!feof(p))
{
   fscanf(p,"%s", city);                 //ввод названия города–начала дороги
 r=0;
 for(i=0,r=0;i<j&&!r;i++)
       if(!strcmp(city,z[i])) r=1;
 if(r) i--;                                 //если город уже есть в списке
           else
               {    i=j;
                    j++;
                    strcpy(z[i],city);          //добавить город в список городов
```

```
                    }

fscanf(p,"%s", city);              //ввод названия города–конца дороги
for(g=0,r=0;g<j&&!r;g++)
        if(!strcmp(city,z[g])) r=1;
if(r) g--;                         //если город уже есть в списке
   else
       {    g=j;
                    j++;
                    strcpy(z[g],city);        //добавить город в список городов
                   }
fscanf(p,"%f ",&m);                //ввод длины дороги
for(b = adj [i]; b!= NULL&&b->town!=g; b = b->next) ;
  if (!b)
  { b=adj [i];
    adj  [i]=new  listinc;          //добавление нового элемента к списку
                                    //смежности i-й вершины
    adj[i]->top=g;                  //в список смежности i-го города занести
                                    //номер смежной вершины
   adj[i]->next=b;
   adj  [i]->weight=m;                  //занесение длины дороги в список
                                        //смежности i-й вершины
   }
  else if (m < b->weight) b->weight = m;
}

 if(j<N)
            return 0; //есть изолированные города

 else     return 1; //введен связный оргграф
}

//функция поиска кратчайших путей от вершины k

int search_ver ( int k, int N, listinc *adj [] )
{
 int i,v,u,flag=1;
 float d[n];                //массив расстояний от вершины k до остальных
 line *Q;              //определение очереди (указателя на начало очереди)
```

```
 listinc *use ;

 Q = NULL;                 //создание пустой очереди
 for(i=0;i<N;i++)       //установка начальных (максимальных) расстояний
 d[i]=1000;
 d[k]=0;                   //установка расстояния до вершины k
for(i=0; i<N; i++)   //кладем в очередь все вершины и расстояния до них
  in_line (&Q, i, d);

while(Q&&flag)
{
 u=out_line_min (&Q);                    //берем из очереди вершину с
                                         //минимальным расстоянием
 if (d [u] > 100 ) flag = 0;      //если расстояние до u больше 100, то поиск
                                  //кратчайшего расстояния прекращаем
else
 {use=adj[u];
   if  ( use )
  {
   while( use )                      //просмотр списка вершин смежных с u
     {
           v=use->top;
      if(d[v] > (d[u]+use->weight ))          //релаксация расстояния до
                                              //вершины v
           d[v] = d[u] + use->weight;
           use=use->next;
     }
    change (Q,  d );          //изменение расстояний до вершин в очереди Q
  }
}
}
if(flag)                     //если просмотрены все вершины, начиная от k-й
for(i=0;i<N&&flag;i++)              //и расстояния до некоторых вершин
                                    //больше 100,
  if(d[i]>100) flag=0;              //то вершина k не является результатом

return flag;
}
void main()
```

```
{
 FILE *p;
 listinc *adj [n];
 int N,r,i,flag,result;

 p=fopen("system.in","r");
 if (p!= NULL)
 { fscanf ( p,"%d",&N );                    //ввод числа городов
 if(N>1)
 {
        for(i=0;i<N;i++)
            adj [i]=NULL;          //создание пустого списка смежности

        r=input(adj, N, p);                    //ввод системы дорог

        if(r==1)
        {
         if(N==2)
         {
            for(i=0,flag=0;i<2&&!flag;i++)  //поиск искомого города, если
                                            //в системе
                for(;adj [i]!=NULL;adj [i]=adj [i]->next)       //дорог всего
                                                                //два города
                    if(adj [i]->weight<=100) flag=1;

                    result=flag;
         }
         else
         {      for(i=0,flag=0;i<N&&!flag;i++)              //поочередный
                                                  //просмотр городов и
                    flag = Search_ver (i, N, adj);    //поиск искомого
                                                      //города

                    result=flag;
         }
        }
        if(r==0) result=0;                     //изолированная вершина

        if(result) { printf("\nв системе дорог есть такой город\n");
```

```
                printf("\nэто город     %s\n", z [i-1] ); }
          else       printf("\nв системе дорог нет такого города\n");
          }
          else printf("\n система дорог пуста\n");
 fclose(p);
}
 else printf("\n файл не открыт \n");
}
```

MODYL_2.CPP

```
#include <stdio.h>
extern struct line                  //определение типа элемента очереди
{int town;
  float pr;
  line *next, *prev;};

//функция  добавления вершины и расстояния до нее в очередь
void in_line (line **s, int v, float d [ ] )
{
  line *p;
  if ( *s )                         //включение элемента, если очередь не пуста
    { p=(*s)->prev;
      p->next = new line;
      p->next->prev = p;
      p = p->next;
      p-> town = v;
      p->pr = d [v];
      (*s)->prev = p;
     }
else                              //включение элемента, если очередь пуста
   { (*s) = new line;
     (*s)->next = (*s);
     (*s)->prev = (*s);
     (*s)-> town = v;
     (*s)-> pr = d [v];
    }
//функция изменения приоритета (расстояния) у элемента очереди
void change (line *s, int N, float d [ ] )
{
```

```
  int i;
  line *p;
  p = s;
  do
     {  i =  p-> town;
        p->pr = d [i];
        p = p->next;
       }
  while ( p!=s );
  }
//функция взятия из очереди элемента с наименьшим приоритетом
//(расстоянием)
 int out_line_min (line **s )
{
  line *p, *q;
  int x;
  float w;
  if ( (*s) = = (*s)->next )          //случай, когда в очереди один элемент
       { x =(*s)-> town; delete (*s); (*s) = NULL; }
  else
     {                          //случай, когда в очереди больше одного элемента
       w = (*s)->pr;
       q = *s;
       p = (*s)->next;
      while ( p != *s )           //поиск элемента с наименьшим приоритетом
         {
           if ( p->pr < w ) {w = p->pr;  q = p;}
           p = p->next;
         }
     x = q-> town;                    //удаление элемента из очереди
     q->prev->next = q->next;
     q->next->prev = q->prev;
     if ( q = = (*s) ) (*s) = q->next;
     delete (q);
   }
   return x;
}
```

7. Набор тестов

Замечание: Приведенный набор тестов не полный, он является только примером того, как надо оформлять тесты. В студенческих отчетах набор тестов должен быть полным. Он может быть оформлен в виде таблицы, но обязательно содержать все указанные здесь пункты.

Тест 1

Назначение: Система дорог пуста

Входные данные:

1

Результат:

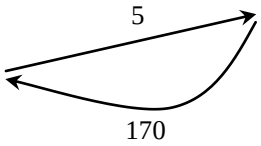
Система дорог пуста

Тест 2

Назначение: Тривиальный вариант непустой системы дорог и искомый город есть

Входные данные:

2 novosib sydney 5 sydney novosib 170



Результат:

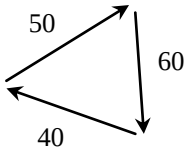
В системе дорог есть такой город это novosib

Тест 3

Назначение: В простой системе дорог существуют кратчайшие пути от одной вершины ко всем остальным, но длина превышает 100

Входные данные:

3 novosib sydney 50 sydney colo 60 colo novosib 40

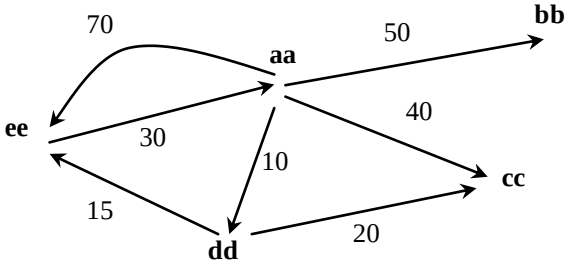


Результат:

В системе дорог нет такого города

Тест 4
Назначение: В системе дорог существует две вершины удовлетворяющие условию задачи
Входные данные:

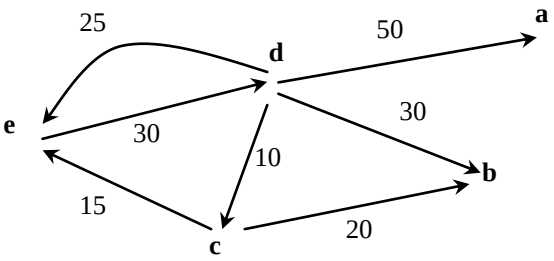
4 aa bb 50 aa cc 40 aa dd 10 aa ee 70 ee aa 30 dd ee 15 dd cc 20



Результат:
В системе дорог есть такой город это aa.

Тест 5
Назначение: В системе дорог существует две вершины удовлетворяющие условию задачи и более одного кратчайшего пути между вершинами
Входные данные:

5 c b 20 c e 15 e d 30 d e 25 d a 50 d c 10 d b 30



Результат:
В системе дорог есть такой город это e.

ПРИМЕРЫ ВАРИАНТОВ ЗАДАНИЙ

Тема. **Графы, деревья, системы дорог**

1. Найти все вершины орграфа, от которых существует путь заданной длины к выделенной вершине.

2. *Источником* орграфа назовем вершину, от которой существует путь ко всем другим вершинам; *стоком* – вершину, достижимую от всех других вершин. Найти все источники и стоки данного орграфа.

3. Известно, что заданный граф – не дерево. Проверить, можно ли удалить из него одну вершину (вместе с инцидентными ей ребрами), чтобы в результате получилось дерево.

4. Подсчитать количество компонент связности в дополнении заданного графа.

5. Найти такую вершину заданного графа, которая принадлежит каждому пути между двумя выделенными (различными) вершинами и отлична от каждой из них.

6. По графу G построить граф $K(G)$ с тем же множеством вершин, что и у G; вершины в $K(G)$ смежны тогда и только тогда, когда расстояние между ними в G не превышает 2. Проверить, совпадают ли степени всех вершин в $K(G)$, и если нет, то нельзя ли удалить из него одну вершину так, чтобы полученный граф удовлетворял этому требованию.

7. Задан орграф с циклами. Проверить, можно ли удалить одну вершину так, чтобы в полученном орграфе не было циклов.

8. По графу G построить граф G' следующим образом: в качестве вершин в G' берутся ребра графа G: две вершины в G' смежны тогда и только тогда, когда смежны соответствующие ребра в G. В G' найти все вершины, расстояние от которых до некоторой выделенной равно 2.

9. Граф назовем *асимметричным,* если у него имеется единственный автоморфизм – тождественный. Определить, является ли асимметричным заданный граф.

10. Определить, является ли заданный граф *двудольным,* т. е. можно ли разбить множество его вершин на два подмножества так, чтобы каждое ребро соединяло вершины из разных подмножеств.

11. Задана система односторонних дорог. Найти путь, соединяющей города A и B и не проходящий через заданное множество городов.

12. Система двусторонних дорог называется *трисвязной,* если для любой четверки разных городов A, B, C, D существует два различных

пути из *A* в *D*, причем один из них проходит через *B*, а другой – через *C*. Определить, является ли трисвязной данная система двусторонних дорог.

13. В системе двусторонних дорог для каждой пары городов указать длину кратчайшего пути между ними.

14. Задана система двусторонних дорог. Найти два города и соединяющий их путь, который проходит через каждую из дорог системы ровно один раз.

15. Задана система двусторонних дорог, причем для любой пары городов можно указать соединяющий их путь. Найти такой город, для которого сумма расстояний до остальных городов минимальна.

16. Имеется *N* городов. Для каждой пары городов (*I, J*) можно построить дорогу, соединяющую эти два города и не заходящую в другие города. Стоимость такой дороги из пункта *I* в пункт *J* известна. Вне городов дороги не пересекаются. Написать программу для нахождения самой дешевой системы дорог, позволяющей попасть из любого города в любой другой.

17. В системе двусторонних дорог за проезд по каждой из них взимается некоторая пошлина. Найти путь из города *A* в город *B* с минимальной величиной *S* + *P*, где *S* –сумма длин дорог, а *P* – сумма пошлин за проезд по дорогам.

18. Заданы две системы двусторонних дорог с одним и тем же множеством городов (железные и шоссейные дороги). Найти минимальный по длине путь из города *A* в город *B* (который может проходить как по железным, так и по шоссейным дорогам) и места пересадок с одного вида транспорта на другой на этом пути.

19. Построить дерево двоичного поиска для заданного множества целых чисел и занумеровать его вершины в соответствии с порядком прямого обхода этого дерева.

20. На рис. 4 приведен пример запутанной сети дорог района города. Представьте, что мусорная машина должна пройти по всем дорогам хотя бы один раз, чтобы собрать мусор. Число на каждой стороне показывает время, которое мусорная машина должна потратить на преодоление этого интервала. Кроме того, проходя перекресток, машина тратит дополнительное время, которое для каждого перекрестка определяется количеством пересекающихся дорог. Для заданной сети дорог и времени проезда по ним определите путь, который машина для сбора мусора должна пройти за кратчайшее время. В заданной сети перекрестком будем считать вершину, в которой сходятся не менее четырех дорог.

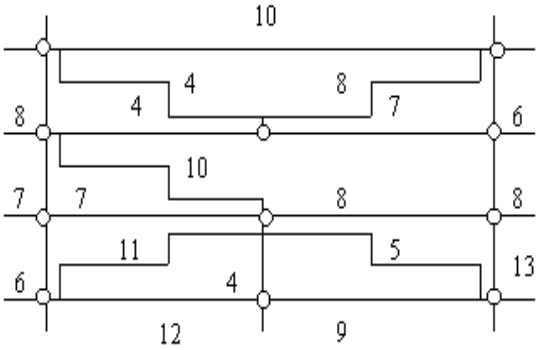


Рис. 4

21. Множество целых чисел представить в виде дерева двоичного поиска и на основе этого представления упорядочить это множество.

22. Проверить, является ли заданное дерево двоичного поиска АВЛ-деревом.

23. К множеству целых чисел *S*, представленному в виде АВЛ-дерева, добавить число *n* так, чтобы множество $S \bigcup \{n\}$ также оказалось представленным в виде АВЛ-дерева.

Тема. **Грамматики, языки, автоматные диаграммы**

24. Слово называется *слабо симметричным,* если его можно преобразовать в симметричное некоторой последовательностью операций обращения, применяемых к произвольным подсловам получающихся слов. Определить, является ли слабо симметричным заданное слово.

25. Задана грамматика *G,* каждое правило которой имеет вид $A \to BC$ или $A \to a$, где *A, B, C* – нетерминалы, а *a* – терминал. Определить, выводимо ли заданное слово, составленное из терминалов, в грамматике *G*.

26. Определить, пусто ли *L*(*G*) для заданной грамматики *G*.

27. Определить, является ли бесконечным *L*(*G*) для заданной грамматики *G*.

28. По заданной грамматике построить такую, которая эквивалентна исходной, а каждое правило имеет вид $A \to BC$ или $A \to a$, где *A, B, C* – нетерминалы, *a* – терминал.

29. Две вершины в автоматной диаграмме называются *близнецами,* если выходящие из них дуги с одинаковыми пометками ведут к одной

и той же вершине. По диаграмме *D* построить эквивалентную ей диаграмму *D'* так, чтобы в *D'* не было ни одной пары вершин-близнецов.

30. По двум заданным автоматным диаграммам над алфавитом *X* определить, эквивалентны ли они.

Указание. Удалить вершины, недостижимые от начальной, и склеить все склеиваемые вершины в каждой из диаграмм. Если полученные диаграммы совпадают, то исходные были эквивалентны, иначе – нет.

31. По заданной автоматной диаграмме *D* определить, пуст ли ее язык *L*(*D*).

32. По заданной автоматной диаграмме *D* определить, является ли ее язык *L*(*D*) бесконечным.

Тема. **Логические формулы и фрагменты**

33. По заданной логической формуле построить эквивалентную логическую формулу, в которой знак отрицания встречается только перед переменными.

Указание. Воспользоваться эквивалентными преобразованиями

NOT NOT *переменная => переменная*

NOT(формула-1) OR *(формула-2) =>*

NOT (формула-1) AND NOT *(формула-2)*

NOT(конъюнкция-1 AND *конъюнкция-2) =>*

NOT (конъюнкция-1) OR NOT *(конъюнкция-2)*

34. Реализовать упрощение логических формул относительно правил преобразования, заданных следующими схемами правил:

TRUE OR *формула => TRUE*

TRUE AND *конъюнкция => конъюнкция*

формула OR TRUE => TRUE

конъюнкция AND TRUE => *конъюнкция*

NOT FALSE => TRUE

NOT TRUE => FALSE

35. Проверить, является ли заданная ДНФ совершенной.

36. Вычислить значение заданной логической формулы, не содержащей вхождений переменных.

37. Вычислить значение заданной ДНФ для заданных значений встречающихся в ней переменных.

38. Определить, эквивалентны ли две заданные расширенные формулы.

39. Определить, эквивалентны ли заданные ДНФ и расширенная формула.

Тема. **Выражения, программы и полиномы**

40. Построить синтаксический анализатор для понятия *описание-переменных,* которое определяется следующим образом:

описание-переменных ::= VAR *пробел список-описаний пробелы*

пробелы ::= *{пробел | пробел пробелы}*

$$\textit{список описаний} ::= \left\{ \begin{array}{l} \textit{описание;} \\ \textit{описание; список - описаний} \end{array} \right\}$$

описание ::= *список-идентификаторов : идентификатор*

список-идентификаторов ::=

$$::= \left\{ \begin{array}{l} \textit{идентификатор} \\ \textit{идентификатор, список - идентификаторов} \end{array} \right\}$$

идентификатор ::= *буква {буква | цифра}* пробелы*

41. Построить синтаксический анализатор для определяемого в словаре понятия *линейная-программа.*

42. Построить синтаксический анализатор для определяемого в словаре понятия *простая-программа.*

43. Построить синтаксический анализатор для определяемого в словаре понятия *простое-выражение.*

44. По заданному простому выражению, не содержащему вхождений идентификаторов, вычислить его значение.

45. Восстановить простое выражение по заданной его обратной польской записи.

46. Восстановить простое выражение по заданной его прямой польской записи.

47. Выполнить заданную линейную программу для заданных начальных значений всех встречающихся в ней переменных. Определить значения этих переменных после выполнения линейной программы (если не возникнет ошибки переполнения).

48. По заданному простому выражению построить линейную программу, которая вычисляет и засылает в переменную RESULT значение этого выражения.

49. По заданной линейной программе, в которой последний оператор содержит в левой части переменную RESULT, построить простое выражение, значение которого совпадает с тем значением, которое получает переменная RESULT после исполнения линейной программы.

50. Упростить заданную правдоподобную простую программу, заменяя всякий оператор

IF *константа-1 знак-отношения константа-2* THEN *оператор*

на *оператор* в случае истинности условия или удаляя этот условный оператор в случае ложности условия.

51. Из-за недосмотра программиста был утерян раздел описаний переменных правдоподобной простой программы. Восстановить его.

52. Проверить, является ли приведенным заданный полином.

53. По заданному полиному и (целым) значениям всех входящих в него переменных вычислить значение полинома.

54. Заданный полином преобразовать в приведенный.

55. Заданное простое выражение преобразовать в полином. Если этого нельзя сделать, как, например, в случае простого выражения *x*x*x*x* ... *x* (20 раз множитель *x*), то программа должна печатать сообщение об этом.

56. Задан приведенный полином P от трех переменных x, y, z. Определить, имеет ли уравнение $P = 0$ хотя бы одно такое решение x, y, z в целых числах, что $x \le 50$, $y \le 50$ и $z \le 50$.

Тема. **Геометрия**

57. На плоскости задано множество n произвольным образом пересекающихся отрезков прямых линий. Перечислить множество всех треугольников, образованных указанными отрезками.

58. Найти минимальное множество прямых, на которых можно разместить все точки заданного на плоскости множества точек.

59. Найти положение на плоскости прямоугольника с заданными длинами сторон при условии, что его вершины должны иметь целочисленные координаты и что внутри него должно находиться максимальное число точек заданного множества.

60. Найти минимальное множество окружностей, на которых можно разместить все точки заданного на плоскости множества точек.

61. В трехмерном пространстве задано множество материальных точек. Найти разбиение этого множества на два таких непустых и непересекающихся множества, чтобы их центры тяжести находились наиболее близко друг к другу.

Тема. **Игры**

62. Имеется N прямоугольных конвертов и N прямоугольных открыток различных размеров. Можно ли разложить все открытки по конвертам так, чтобы в каждом конверте было по одной открытке.

Замечание. Открытки нельзя складывать, сгибать и т. п., но можно помещать в конверт под углом. Например, открытка с размерами сторон 5:1 помещается в конверты с размерами 5:1, 6:3, 4.3:4.3, но не входит в конверты с размерами 4:1, 10:0.5, 4.2:4.2.

63. N колец сцеплены между собой. Удалить минимальное количество колец так, чтобы получилась цепочка.

64. На шашечной доске размера $n \times n$ ее верхний правый угол имеет номер (1,1). В позиции (i, j) стоит фишка, которая может двигаться по правилу, показанному на рис. 5 (допустимые ходы обозначены x, шашка – O). Фишка не может дважды ставиться на одно и то же поле. Играют двое, по очереди двигая фишку. Выигрывает, поставивший фишку в позицию (1,1). Для введенных i, j, определить, может ли выиграть первый игрок при наилучших ходах второго игрока. Написать программу, где первый игрок – компьютер.

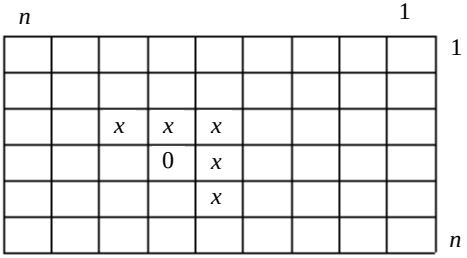


Рис. 5

65. Пусть слово – это последовательность от 1 до 8 символов, не включающая пробелов. Вводится n слов $A_1,...,A_n$. Можно ли их переупорядочить так, чтобы получилась «цепочка», т. е. для каждого слова A_j его первая буква должна совпадать с последней буквой предыдущего слова, а последняя буква в A_j – с первой буквой последующего слова; соответственно последняя буква последнего слова должна совпадать с первой буквой первого слова. В цепочку входят все n слов без повторений. Если такое упорядочение возможно, то вывести какую-нибудь цепочку слов. Слова при выводе разделяются пробелами.

66. Игровой автомат состоит из нескольких изогнутых трубок, по форме похожих на перевернутую вверх ногами букву Y. Сверху у трубы находится входное отверстие, а снизу – два выходных отверстия. Труба может находиться в одном из двух возможных состояний: шарики кладутся внутрь один за другим через входное отверстие. В состоянии *1* (рис. 6) шарик покидает трубу через незаблокированный правый выход, при этом состояние *1* автоматически изменяется на состояние *0* (рис. 7, правый вход заблокирован, левый – нет). В состоянии *0* все происходит наоборот.

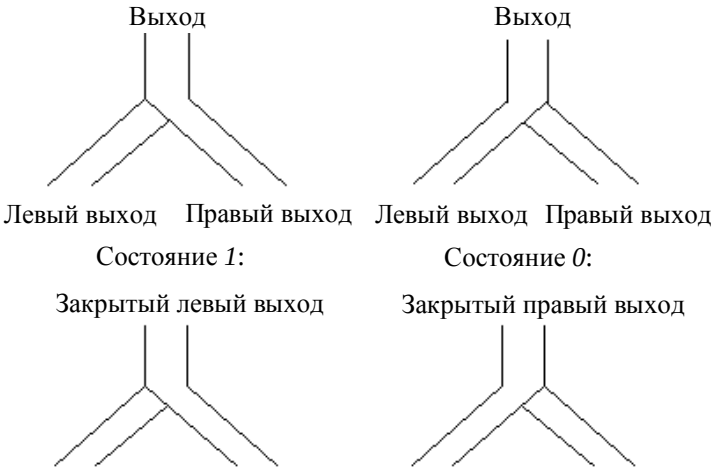


Рис.6 *Рис.7*

Пусть игральный автомат состоит из 8 Y-образных трубок, взаимное расположение которых показано на рис. 8. Вы можете забрасывать шарики через входы *A*, *B* или *C*.

Если мы забрасываем шарик через вход *A*, то это действие будем обозначать буквой *A*, если бросаем шарик через вход *B*, то это действие обозначаем буквой *B*, а если забрасываем шарик через вход *C*, то соответственно – буквой *C*. Например, первый шарик опускается через вход *A*. Он покидает трубу G_1 через левый выход, изменяя ее состояние с *0* на *1*, поступает в G_6 и покидает автомат через левый выход G_6 (изменяя состояние G_6 с *0* на *1*). Автомат изменил свое состояние в результате выполнения действия *A*.

Мы снова забрасываем шарик через вход *A* (вход *A* находится в состоянии *1*). Шарик покидает G_1 через правый выход, изменяя состояние *1* на состояние *0*, попадает в G_4, покидает G_4 через правый выход, изменяя состояние *1* на состояние *0*, попадает в G_7, покидает автомат через левый выход (переводя G_7 из состояния *0* в состояние *1*). Два вышеизложенных действия мы запишем в виде: *AA*

Если мы опустим третий шарик через вход *B*, четвертый – через вход *C*, то все действия запишутся следующим образом: *AABC*

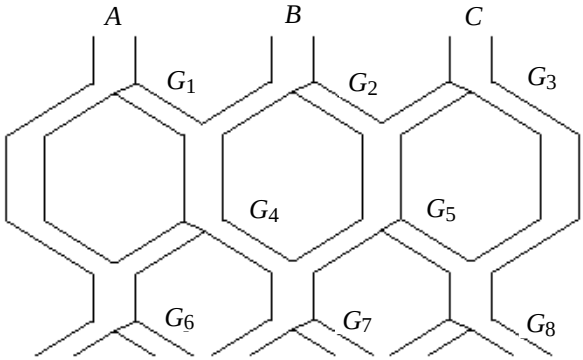


Рис. 8

Необходимо:

1) ввести с клавиатуры последовательность из 8 двоичных цифр (бит), показывающих соответственно начальное состояние G_1–G_8:

бит	7	6	5	4	3	2	1	0
вход	G_8	G_7	G_6	G_5	G_4	G_3	G_2	G_1

Например, последовательность 11001010 означает, что G_8, G_7, G_4 и G_2 находятся в состоянии *1* (левый вход заблокирован), тогда как G_6, G_5, G_3 и G_1 находятся в состоянии *0* (правый вход заблокирован);

2) ввести с клавиатуры вторую последовательность из 8 бит, показывающих конечные состояния G'_1–G'_8;

3) напечатать последовательность ходов *A*, *B* или *C*, указывающую, каким образом можно получить конечную позицию, забрасывая шарики через входы *A*, *B* и *C*.

67. Данные *N* косточек домино по правилам игры выкладываются в прямую цепочку начиная с косточки, выбранной произвольно, в оба конца до тех пор, пока это возможно. Определить такой вариант выкладывания заданных косточек, при котором к моменту, когда цепочка не может быть продолжена, «на руках» останется максимальное число очков.

68. Одинокий король долго ходил по бесконечной шахматной доске. Известна последовательность из *N* его ходов (вверх, вниз, влево, вправо, вверх-влево и т. п.). Написать программу, определяющую, побывал ли король дважды на одном и том же поле за минимально возможное при заданном *N* числе вычислений.

69. Из листа клетчатой бумаги размером *M*N* клеток удалили некоторые клетки. На сколько кусков распадется оставшаяся часть листа?

70. Решить предыдущую задачу при условии, что перед удалением клеток лист склеили в цилиндр высотой *N*.

71. Матрица размером *N*M* определяет некоторый лабиринт. В матице элемент *1* обозначает стену, а *0* определяет свободное место. В первой строке матрицы определяются входы $x(i)$, а в последней выходы $y(i)$, $i = 1,\ldots, k$, которые должны быть нулевыми элементами. Необходимо определить, можно ли провести k человек от входа $x(i)$ до выхода $y(i)$ (соответственно $i = 1,\ldots, k$), таким образом, чтобы каждое свободное место посещалось не более одного раза. То же, но выводить можно через любой из выходов.

Примечание. Движение в лабиринте осуществляется только по вертикали или горизонтали.

72. В городе *N* активно развивается общественный транспорт. В ближайшее время городские власти планируют отремонтировать около 20 км трамвайных путей по новой технологии с применением специальной плитки. Внедрение нового способа ремонта межрельсового пространства началось минувшим летом, и первые результаты видны уже сейчас.

Так, бригада Равшана и Джамшуда вымостила плиткой два перекрестка в центре города. Поразительная скорость работы объясняется новой хитростью, придуманной их начальником. Перед началом ремонтных работ он сам размещает между рельсами несколько плиток так, чтобы можно было продолжить выкладывать их только одним способом. В результате экономится время на проектирование плиточных узоров, которое занимает большую часть рабочего дня у других бригад.

Каждая плитка имеет размер 1×2. Требуется замостить плиткой прямоугольную область размером $n \times m$. Определить минимальное число плиток, которое должен выложить начальник, чтобы итоговый узор (т. е. полное замощение прямоугольника плитками) определялся однозначно. Два узора считаются одинаковыми, если расположение всех плиток в них совпадает.

73. Всем хорошо известна «Игра в 15», представляющая собой 15 квадратных фишек, пронумерованных числами от 1 до 15. Фишки уложены в квадрат со стороной в 4 стороны фишки, одна позиция для фишки свободна. Если обозначить свободную позицию за $*$, то головоломка состоит в том, чтобы получить из произвольной начальной позиции позицию следующего вида:

```
1  2  3  4
5  6  7  8
9  10 11 12
13 14 15 *
```

Единственной разрешенной операцией является обмен $*$ с одной из соседних по ребру фишек. Операции будем кодировать буквами:

r поменять $*$ с фишкой, которая стоит справа от $*$
l поменять $*$ с фишкой, которая стоит слева от $*$
u поменять $*$ с фишкой, которая стоит сверху от $*$
d поменять $*$ с фишкой, которая стоит снизу от $*$

Например, решением головоломки

```
1  2  3  4
5  6  7  8
9  10 12 *
13 14 11 15
```

является последовательность ldr.

От вас требуется решить более простую головоломку «Игра в 8», в которой требуется расположить 8 фишек в виде

```
1  2  3
4  5  6
7  8  *
```

74. Фирма Kel-Morian Productions разрабатывает искусственный интеллект для новой автоматизированной модели промышленного робота SCV-2. На данном этапе создается робот для строительства и ремонта стен, составленных из стандартных строительных блоков.

Для начала было принято решение сделать упрощенную модель робота, который будет работать со стенами, состоящими из блоков одинакового размера. Стена представляет собой последовательность

столбиков, составленных из блоков. Пример такой стены приведен на рис. 9.

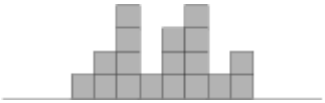


Рис. 9

Первая модель робота может выполнять ровно одно элементарное действие – взять верхний блок в некотором столбике и положить его на соседний столбик. При этом создавать новый столбик, ставя блок рядом с краем стены, не разрешается.

В качестве тестового задания для искусственного интеллекта была поставлена задача *выравнивания стены*. Стена называется *ровной*, если высота двух любых столбиков различается не более чем на один. В процессе выравнивания робот должен с использованием элементарных действий превратить заданную стену в произвольную ровную. При этом количество выполненных элементарных действий должно быть минимальными, а количество столбиков в стене не должно измениться.

Например, стена, приведенная на рис. 9, может быть превращена в ровную стену, приведенную на рис. 10, за четыре элементарных действия, и это число действий минимально.

Рис. 10

Помогите разработчикам искусственного интеллекта проверить разработанный ими алгоритм, найдите минимальное количество действий, которое придется совершить роботу для выравнивания заданной стены.

75. На стандартной шахматной доске(8×8) живут два шахматных коня: Красный и Зеленый. Обычно они беззаботно скачут по просторам доски, пощипывая шахматную травку, но сегодня особенный день: у Зеленого коня день рождения. Зеленый конь решил отпраздновать это событие вместе с Красным. Но для осуществления этого прекрасного плана им нужно оказаться на одной клетке. Красный и Зеленый шахматные кони сильно отличаются от Черного и Белого тем, что они ходят не по очереди, а одновременно, и если оказываются на одной

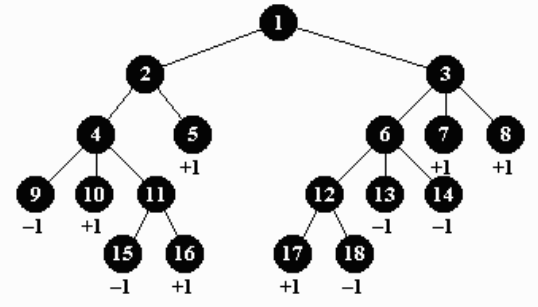
клетке, никто никого не съедает. Сколько ходов им потребуется, чтобы насладиться праздником?

76. На прогулке в детском саду Никифор играет в классики. Поле для игры представляет собой прямоугольник размером M на N м, разбитый на клетки размером 1 на 1 м. Никифор прыгает из клетки в клетку (не обязательно в соседнюю). При этом каждая клетка окрашена в черный или в белый цвет. Каждый раз, когда Никифор попадает на клетку, во всех клетках, центры которых удалены от центра клетки с Никифором на целое число метров, цвет меняется на противоположный. Известно, сколько раз Никифор побывал на каждой из клеток, а также цвета всех клеток в конце игры. Требуется восстановить начальную раскраску клеток поля.

77. На кухне живет мышка. Также на кухне живет кошка и лежит сыр. Координаты сыра и мышки известны, а кошка спит. Наконец, на кухне стоит мебель. Мебель – это набор выпуклых многоугольников. Мышка хочет добраться до сыра как можно более незаметно. Точка пути называется опасной, если расстояние до ближайшего предмета мебели больше 10 см. Необходимо найти для мышки наименее опасный маршрут, т. е. такой, в котором сумма длин опасных участков будет минимальной.

78. Игра для двух игроков определяется ее деревом. Соперники делают ходы по очереди. Первый игрок начинает игру. Игра кончается или вничью, или победой одного из игроков. Листья дерева этой игры могут иметь значения, равные одному из трех чисел: +1 – победа первого игрока, −1 – победа второго игрока, 0 – ничья. Ваша задача – по заданному дереву игры определить, кто выиграет, если оба противника следуют правильной стратегии. Ниже приведен пример дерева-игры.

В данном примере выигрывает первый игрок.



Литература

1. *Андерсон Дж*. Дискретная математика и комбинаторика. – М.; СПб.; Киев: Вильямс, 2004.

2. *Касьянов В.Н., Сабельфельд В.К*. Сборник заданий по практикуму на ЭВМ. – М.: Наука, 1986.

3. *Хиценко В.П., Шапошникова Т.А*. Практикум на ЭВМ. Алгоритмы: учеб. пособие. – Новосибирск: НГТУ, 2004.

СТРУКТУРЫ ДАННЫХ И АЛГОРИТМЫ

Методические указания

Редактор *Н.В. Городник*
Выпускающий редактор *И.П. Брованова*
Компьютерная верстка *С.И. Ткачева*

Подписано в печать 10.02.2009. Формат 60 × 84 1/16. Бумага офсетная. Тираж 180 экз.
Уч.-изд. л. 3,25. Печ. л. 3,5. Изд. № 238. Заказ № . Цена договорная

Отпечатано в типографии
Новосибирского государственного технического университета
630092, г. Новосибирск, пр. К. Маркса, 20