**Вятский Государственный Гуманитарный Университет**

**Кафедра прикладной математики**

Курсовая работа по информатике

Тема: Разработка системы упражнений и задач (алгоритмы-программы) по дискретной математике.

Выполнил:Студент 4 курса

факультета информатики

Лепешкин Антон Геннадъевич

Проверила: Ашихмина Татьяна Викторовна

**Киров 2004**Содержание.

Содержание. 2

Введение. 3

Глава 1 Теоретический материал. 4

Перебор с возвратом. 4

Поиск данных. 5

Логарифмический(бинарный) поиск 5

Методы сортировки. 6

Сортировка слияниями. 6

Быстрая сортировка Хоара. 6

Графы. 6

Представление графа в памяти компьютера 6

Достижимость 7

Кратчайшие пути. 8

Алгоритм Дейкстры 8

Алгоритм Флойда (кратчайшие пути между всеми парами вершин). 9

Глава 2 Система задач и упражнений. 9

Классификация задач. 9

Комнаты музея. 12

Пират в подземелье. 13

Диспетчер и милиция. 14

Задача о футболистах. 15

Задача о семьях. 16

Метро. 16

Роботы. 17

Вожатый в лагере 20

Егерь. 21

Игра «Найди друга». 22

Приложение. 22

1 22

2 25

3 27

4 30

5 32

6 32

7 34

8 39

9 41

10 43

Заключение. 45

Литература 45

Введение.

Несмотря на то, что для решения задач в основном используются общие методы, все-таки мышление каждого конкретного человека немного отличается от мышления других людей, если он обладает достаточной базой знаний. Таким образом, при решении задач «начиная с нуля» можно зайти в тупик, если выбрать неверный путь решения задачи. В данном курсовом проекте мы разработаем собственную классификацию задач, позволяющую определить наиболее подходящий способ решения, чтобы облегчить процесс моделирования и составления алгоритма и предотвратить выбор неверного способа, также рассмотрим данную классификацию с точки зрения методики преподавания информатики. выбор неверного способа. В этом заключается актуальность данного курсового проекта.

**Цель:** Разработать собственную классификацию для задач по дискретной математике. Для достижения этой цели были поставлены следующие задачи:

1. Разработать собственную систему задач и упражнений по дискретной математике.
2. Определить способы решения данных задач, используя теоретический материал курса дискретной математики.
3. Составить алгоритм – программу для каждой задачи, реализующий выбранный способы решения.
4. Разработать систему критериев классификации данной системы задач.

Глава 1 Теоретический материал.

## Перебор с возвратом.

Общая схема

Даны N упорядоченных множеств U1 U2,..., Un (N - известно), и требуется построить вектор А=(а1 а2, ..., аn), где a1€U1, a2€U2, ..., an€Un, удовлетворяющий заданному множеству условий и ограничений.

В алгоритме перебора вектор А строится покомпонентно слева  
направо. Предположим, что уже найдены значения первых (к-1)  
компонент, A=(a1, a2, ..., a(k-1)), ?, ..., ?), тогда заданное множество  
условий ограничивает выбор следующей компоненты аk некоторым  
множеством SkCUk. Если Sk<>[ ] (пустое), мы вправе выбрать в  
качестве ак

**Начало**

**Выборы для А1**

**Выборы для А2 при А1**

**Выборы для А3 при данных А1 и А2**

наименьший элемент Sk и

перейти к выбору ^/^ "^выборы п«я»,

(к+1) компоненты и

так далее. Однако /[ ■ Д Jfcv при данном а,

если условия условия

а,, ^иаз

таковы, что Sk оказалось пустым, то мы возвращаемся к выбору

(к-1) компоненты, отбрасываем

а(k-1) и выбираем в качестве нового a(k-1) тот элемент S(k-i), который непосредственно следует за только что отброшенным. Может оказаться, что для нового a(k-1) условия задачи допускают непустое Sk, и тогда мы пытаемся снова выбрать элемент ак. Если невозможно выбрать a(k-1), мы возвращаемся еще на шаг назад и выбираем новый элемент а(к-2) и так далее.

Графическое изображение - дерево поиска. Корень дерева (0 уровень) есть пустой вектор. Его сыновья суть множество кандидатов для выбора а1 и, в общем случае, узлы k-го уровня являются кандидатами на выбор ак при условии, что а1, а2, ..., a(k-1) выбраны так, как указывают предки этих узлов. Вопрос о том, имеет ли задача решение, равносилен вопросу, являются ли какие-нибудь узлы дерева решениями. Разыскивая все решения, мы хотим получить все такие узлы.

Рекурсивная схема реализации алгоритма,

procedure Backtrack(Beктop,i);

begin

if <вектор является решением> then <записать его>

else begin <вычислить Si>;

for a€Si do Васкtrаск(вектор| | a,i+l); {| | - добавление к вектору компоненты}

end; end;

Оценка временной сложности алгоритма. Данная схема реализации перебора приводит к экспоненциальным алгоритмам. Действительно, Пусть все решения имеют длину N, тогда исследовать требуется порядка | Si| \*| S2| \*...\*| SN| узлов дерева. Если значение S; ограничено некоторой константой С, то получаем порядка CN узлов.

## Поиск данных.

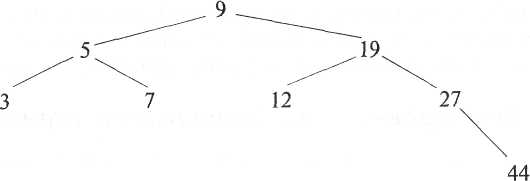
### Логарифмический(бинарный) поиск

Логарифмический (бинарный или метод деления пополам) по­иск данных применим к сортированному множеству элементов *а1 < а2 < ... < ап,* размещение которого выполнено на смежной па­мяти. Для большей эффективности поиска элементов надо, чтобы пути доступа к ним стали более короткими, чем просто последова­тельный перебор. Наиболее очевидный метод: начать поиск со среднего элемента, т.е. выполнить сравнение с элементом а. Результат сравнения позволит определить, в какой половине по­следовательности *а{, а2,..., а,* 1+„ ,,..., *ап* продолжить поиск,

применяя к ней ту же процедуру, и т.д. Основная идея бинарного поиска довольно проста, однако «для многих хороших програм­мистов не одна попытка написать правильную программу закон­чилась неудачей». Чтобы досконально разобраться в алгоритме, лучше всего представить данные *ах < а2 <* ... < *ап* в виде двоичного дерева сравнений, которое отвечает бинарному поиску.

Двоичное дерево называется *деревом сравнений* , если для лю­бой его вершины (корня дерева или корня поддерева) выполняет­ся условие:

***{Вершины левого поддерева}<Вершина корня<{Вершины правого поддерева* }.**



**Рис.** Пример дерева сравнений, отвечающего бинарному поиску среди сортированных элементов: 3,5,7,9,12,19,27,44

Т.о. бинарный поиск – это сравнение эталона х, которое осуществляется с элементом, расположенным в середине массива и в зависимости от результата сравнения (больше или меньше) дальнейший поиск проводится в левой или правой половине массива.

Используется, когда имеется какая-либо информация о массиве, например массив упорядочен по неубыванию. Общее количество сравнений имеет порядок О(N\*logN).

## Методы сортировки.

### Сортировка слияниями.

Используется, когда необходимо объединить упорядоченные фрагменты массивов: A[k],…,A[m] и B[m+1],…,B[q] в один C[k],…,C[q], тоже упорядоченный (k<=m<=q). Основная идея решения состоит в сравнении очередных элементов каждого фрагмента, выяснении, какой из элементов меньше, переносе его во вспомогательный массив С (для простоты) и продвижении по тому фрагменту массива, из которого взят элемент. При этом следует не забыть записать в С оставшуюся часть того фрагмента, который не успел себя «исчерпать».

Метод слияний – один из первых в теории алгоритмов сортировки. Он предложен Дж. Фон Нейманом в 1945 году. Эффективность алгоритма, по Д. Кнуту, составляет С=О(N\*logN).

### Быстрая сортировка Хоара.

Метод предложен Ч.Э.Р.Хоаром в 1962 году.

Идея метода. В исходном массиве А выбирается некоторый элемент Х (барьерный элемент). Основной целью алгоритма является запись Х «на свое место» в массиве, пусть это будет место k, такое, что слева от Х были элементы массива, меньшие или равные Х, а справа – элементы массива, большие Х, т.е. массив А будет иметь вид: (А[1],A[2],…,A[k-1]),A[k] (X), (A[k+1],…, A[n]).

В результате элемент A[k] находится на своем месте и исходный массив А разделен на две неупорядоченные части, барьером между которыми является элемент A[k]. Дальнейшие действия очевидны – независимо сортировать полученные части по той же логике до тех пор, пока не останутся части массива, состоящие из одного элемента, то есть пока не будет отсортирован весь массив.

## Графы.

### Представление графа в памяти компьютера

Определим граф как конечное множество вершин V и набор Е неупорядоченных и упорядоченных пар вершин и обозначим G=(V,E). Мощности множеств V и Е будем обозначать буквами N и М Неупорядоченная пара вершин называется ребром, а упорядоченная пара - дугой. Граф, содержащий только ребра, называется неориентированным; граф, содержащий только дуги, - ориентированным, или орграфом. Вершины, соединенные ребром, называются смежными. Ребра, имеющие общую вершину, также называются смежными. Ребро и любая из его двух вершин называются инцидентными. Говорят, что ребро (u, v) соединяет вершины и и v. Каждый граф можно представить на плоскости множеством точек, соответствующих вершинам, которые соединены линиями, соответствующими ребрам. В трехмерном пространстве любой граф можно представить таким образом, что линии (ребра) не будут пересекаться.

**Способы описания.** Выбор соответствующей структуры данных для представления графа имеет принципиальное значение при разработке эффективных алгоритмов. При решении задач используются следующие четыре основных способа описания графа: матрица инциденций; матрица смежности; списки связи и перечни ребер. Мы будем использовать только два: матрицу смежности и перечень ребер.

Матрица смежности - это двумерный массив размерности N\*N. 1, вершина с номером i смежна с вершиной с номером j, 0, вершина с номером i не смежна с вершиной с номером j

Для хранения перечня ребер необходим двумерный массив R размерности М\*2. Строка массива описывает ребро.

### Достижимость

Путем (или ориентированным маршрутом) ориентированного графа называется последовательность дуг, в которой конечная вершина всякой дуги, отличной от последней, является начальной вершиной следующей.

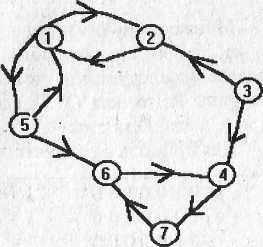
Простой путь - это путь, в котором каждая дуга используется не более одного раза.

Элементарный путь - это путь, в котором каждая вершина используется не более одного раза.

Если существует путь из вершины графа v в вершину i, то говорят, что i достижима из v.

Матрицу достижимости определим следующим образом:

1, если вершина i достижима из v и



R[v,u]=0, при недостижимости

Множество R(v) - это множество таких вершин графа G, каждая из которых может быть достигнута из вершины v. Обозначим через F(v) множество таких вершин графа G, которые достижимы из v с использованием путей длины 1. T2(v) - это Г(Г(у)), то есть с использованием путей длины 2 и так далее. В этом случае:

R(v)={v}UГ(v)UГ2(v)U...UГp(v).

При этом р - некоторое конечное значение, возможно, достаточно большое.

Пример (для рисунка). R(1)={1}U{2,5}U{1,6}U{2,5,4}U{1,6,7}={1,2,4,5,6,7}

Выполняя эти действия для каждой вершины графа, мы получаем матрицу достижимостей R.

## Кратчайшие пути.

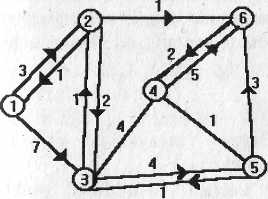
### Алгоритм Дейкстры

Дано. Ориентированный граф G=<V,E>, s - вершина источник; матрица смежности A (A:array[1..n,1..n] of integer); для любых u, v€V вес дуги неотрицательный (А[u,v]>=0). Результат. Массив кратчайших расстояний - D.

В данном алгоритме формируется множество вершин Т, для которых еще не вычислена оценка расстояние и, это главное, минимальное значение в D по множеству вершин, принадлежащих Т, считается окончательной оценкой для вершины, на которой достигается этот минимум. С точки зрения здравого смысла этот факт достаточно очевиден. Другой "заход" в эту вершину возможен по пути, содержащему большее количество дуг, а так как веса неотрицательны, то и оценка пути будет больше.

Пример

Его матрица смежности:



|  |
| --- |
| ∞ 3 7 ∞ ∞ ∞  1 ∞ 2 ∞ ∞ 1  А= ∞ 1 ∞ 4 4 ∞  ∞ ∞ ∞ ∞ 1 5  ∞ ∞ 1 ∞ ∞ 3  ∞ ∞ ∞ 2 ∞ ∞ |

В таблице приведена последовательность шагов (итераций) работы алгоритма. На первом шаге минимальное значение D достигается на второй вершине. Она исключается из множества Т, и улучшение оценки до оставшихся вершин (3,4,5,6) ищется не по всем вершинам, а только от второй.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| № итерации | **D[1]** | **D[2]** | **D[31** | **D[4]** | **D[5]** | **D[6]** | **T** |
| **1** | 0 | **3** | 7 | ∞ | ∞ | ∞ | [2,3,4,5,6] |
| **2** | 0 | 3 | 5 | ∞ | 11 | **4** | [3,4,5,6] |
| **3** | 0 | 3 | **5** | 6 | ∞ | 4 | [3,4,5] |
| **4** | 0 | 3 | 5 | **6** | 9 | 4 | [4,5] |
| **5** | 0 | 3 | 5 | 6 | **7** | 4 | [5] |

Время работы алгоритма пропорционально N2.

### Алгоритм Флойда (кратчайшие пути между всеми парами вершин).

Дано. Ориентированный граф G=<V,E>, s - вершина источник; матрица смежности A (A:array[1..n,1..n] of integer); для любых u, v€V вес дуги неотрицательный (А[u,v]>=0). Результат. Матрица D кратчайших расстояний между всеми парами вершин графа и кратчайшие пути.

Идея алгоритма. Обозначим через D**m**[i,j] оценку кратчайшего пути из i в j с промежуточными вершинами из множества [1..m]. Тогда имеем: D**0**[i,j]:=A[i,j] и D(**m+1**)[i,j]=min{D**m**[i,j],D**m**[i,m+1]+D**m**[m+1,j]}. Второе равенство требует пояснения. Пусть мы находим кратчайший путь из i в j с промежуточными вершинами из множества [1..(m+1)]. Если этот путь не содержит вершину (m+1), то D**(m+1)**[i,j]=D**m**[i,j]. Если же он содержит эту вершину, то его можно разделить на две части от i до (m+1) до j. Время работы алгоритма пропорционально N3.

Глава 2 Система задач и упражнений.

## Классификация задач.

Набор задач, разработанный нами и изложенный ниже можно систематизировать по следующим критериям:

**По тематике.**

**Задачи с использованием алгоритма перебора с возвратом**

**Задачи с использованием**

**алгоритмов поиска данных**

**Задачи с использованием методов сортировки**

**Задачи с использова -нием алгоритмов на графах**

**Логарифмический поиск**

**Линейный поиск**

**Сортировка Хоара**

**Сортировка слияниями**

**Достижимость**

**Кратчайшие пути в графе**

**Алгоритм Дейкстры**

**Алгоритм Флойда**

**По уровню сложности задачи.**

**Задачи высокого уровня сложности**

**Задачи среднего уровня сложности**

**Задачи низкого уровня сложности**

**Задачи высокого уровня сложности:** это задачи олимпиадного уровня, требующие глубокого знания предмета, а также комплексного подхода к решению задачи (Пример для нашего набора задач, задача о роботах, задача о комнатах музея).

**Задачи среднего уровня сложности:** это задачи, требующие хороших знаний предмета и навыков применения знаний на практике, т.е в процессе решения задач (Пример: задача о семьях, задача о футболистах, задача про милицию и диспетчера).

**Задачи низкого уровня сложности:** это задачи, для решения которых необходимы общие знания предмета и не требующие особых навыков применения знаний на практике, т.к. данные задачи направлены на формирование данных навыков.

**По формулировке задачи.**

**Ситуативные задачи**

**Задачи со строгой формулировкой**

**Ситуативные задачи:** это задачи, формулировка которых представляет собой ситуацию из жизни. Это необходимо для более наглядного представления задачи, а также для того, чтобы сделать задачу более интересной для решения.

**Задачи со строгой формулировкой:** это задачи, в формулировке которой строго изложена суть задачи. Данные задачи являются задачами более низкого уровня, так как в них не требуется определения тематики задачи, а следовательно, и выбора способа решения, требуется лишь реализация алгоритма на языке программирования.

**Задачи с единственным способом решения**

**Задачи с несколькими способами решения**

**По количеству способов решения.**

**Задачи с единственным способом решения:** это задачи, решить которые можно лишь одним способом, т.е. задачу нельзя рассмотреть с точки зрения различных тематик, таким образом, отсутствует выбор способа решения задачи (Пример: задача о футболистах и т.д.).

**Задачи с несколькими способами решения:** это задачи, которые могут быть рассмотрены с точки зрения различных тематик и, таким образом, имеют более широкий спектр решений (Пример: задача о метрополитене и т.д.).

**По массовости решения задачи.**

**Задачи, имеющие решение применимое только к конкретным задачам.**

**Задачи, имеющие решение применимое к целому классу подобных задач.**

**Задачи, имеющие решение применимое только к конкретным задачам:** это задачи, которые в своей формулировке имеют достаточно много деталей, чтобы их решение было применимо только к конкретным задачам (Пример: задача о роботах).

**Задачи, имеющие решение применимое к целому классу подобных задач:** это задачи, в формулировке которых не содержится особых деталей, чтобы их решение было применимо к целому классу подобных задач (Пример: задача о метрополитене и т.д.).

**Задачи.**

Комнаты музея.Составьте алгоритм-программу определения числа комнат в музее и площади каждой комнаты в клетках. План музея показан ниже на рисунке.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |

1. 6 11 6 3 10 6

7 9 6 13 5 7 5

1 10 12 7 13 13 5

13 11 10 8 10 14 13

**Цифровая карта**

Площадь музея состоит из клеток: m рядов и p столбцов. В каждой клетке такой матрицы (цифровая карта) проставляется число, в котором кодируется наличие стен у данной клетки. Значение числа в каждой клетке является суммой чисел: 1 (клетка имеет стену на западе), 2 (клетка имеет стену на севере), 4 (клетка имеет стену на востоке), 8 (клетка имеет стену на юге). Например, если в клетке стоит число 11 (11=8 + 2 + 1), то клетка имеет стену с южной стороны, с северной и с западной.

Исходные данные представляются в текстовом файле со следующей структурой. Первая строка: m, p – размерность сетки. Вторая строка, третья и следующие строки содержат описание матрицы цифровой карты по строкам. Расчетные данные вывести на экран в следующем порядке: первая строка – площадь каждой комнаты музея, вторая строка – количество комнат в музее.

**Пример файла исходных данных:**

1. 7

11 6 11 6 3 10 6

7 9 6 13 5 7 5

1 10 12 7 13 13 5

13 11 10 8 10 14 13

**Пример выходных данных:**

9 3 8 2 6

5

**Идея решения:**

Данную задачу можно решить используя метод **перебора с возвратом.** Используя массив координат перемещения, смотрим, где отсутствуют стены, для каждой клетки, и последовательно двигаемся в ту клетку, в которую возможно, предварительно помечая клетку, в которой уже были. Если мы зашли в тупик, то возвращаемся в клетку, из которой вышли. Одновременно считаем количество клеток в каждой комнате. Когда происходит возврат в начальную точку движения, делаем всю комнату просмотренной (при помощи массива логического типа). Затем ищем клетку, в которой ещё не были и делаем её начальной точкой движения.

(Текст программы см. Приложение 1)

Пират в подземелье.В поисках драгоценных камней пират проваливается в подземелье. План подземелья – матрица N\*M комнат с драгоценными камнями. Камни из одной комнаты имеют одинаковую стоимость. Пирату в каждой комнате разрешается взять всего лишь один камень с собой и следовать в любую другую соседнюю с ней комнату. Каждую из комнат пират может посещать всего лишь один раз. Требуется составить алгоритм-программу определения маршрута посещения пиратом К комнат лабиринта таким образом, чтобы он набрал камней на максимально возможную сумму. Входные и выходные данные: В первой строке входного файла содержатся числа N,M,K. В следующих N строках располагается матрица N\*M лабиринта. Каждый элемент матрицы представляется стоимостью камня соответствующей комнаты. Маршрут начинается с левой верхней угловой комнаты лабиринта. Выходные данные: содержат единственное число, равное общей стоимости взятых с собой камней.

**Пример файла исходных данных:**

3 4 7

1 1 1 1

1 1 2 1

1 1 2 3

**Выходные данные для данного примера:**

12

**Идея решения:** Данную задачу можно решить используя метод **перебора с возвратом.** Двигаясь последовательно по комнатам считаем общую стоимость камней и выбирая наибольшую перебираем все возможные варианты передвижения пирата по комнатам.

**(**Текст программы см. Приложение 2**)**

Диспетчер и милиция.У диспетчера имеется схема города, на которой изображены районы и дороги, связывающие данные районы. На схеме указаны расстояния от одного пункта к другому и направление движения, которое разрешено. Схема выглядит следующим образом:

**1**

**6**

**2**

**2**

**3**

**4**

**4**

**1**

**3**

**5**

**1**

**1**

**1**

**2**

**7**

**5**

**4**

**3**

**1**

Диспетчеру поступают запросы из патрульных машин милиции, патрульные сообщают район, где они находятся и район, в который им необходимо попасть на вызов. Требуется составить алгоритм – программу, которая бы помогла диспетчеру найти минимальное расстояние, которое предстоит покрыть патрульной машине. Необходимо учесть направление движения, которое разрешено на данном участке пути.

**Решение.** Входные и выходные данные:

Первая строка входного файла содержит количество районов города. Затем идет матрица смежности, где занесены все пути из одной вершины в другую с расстоянием:

6

0 3 7 0 0 0

1 0 2 0 0 1

0 1 0 4 4 0

0 0 0 0 1 5

0 0 1 0 0 3

0 0 0 2 0 0

Номер района, из которого выехала милицейская машина и в который ей необходимо попасть вводятся с клавиатуры.

Выходные данные:

Единственное число, которое представляет собой минимальный путь, который предстоит покрыть милицейской машине.

**Идея решения:** данную задачу можно решить с помощью **алгоритма поиска кратчайших путей в графе (алгоритм Дейкстры).**

(Текст программы см. Приложение 3)

Задача о футболистах.Футбольная команда поехала на выездную игру, так как команда большая, то все игроки залезли в два автобуса, в произвольном порядке и в разных количествах. В автобусах игроки по привычке построились по возрастанию номеров и сели. Необходимо составить алгоритм – программу, помогающую игрокам, на выходе из двух автобусов, сразу же вставать по возрастанию номеров.

**Исходные и выходные данные:**

Входной файл содержит три строки. В первой строке находятся два числа – количество игроков в первом и втором автобусах. Вторая строка содержит номера игроков, находящихся в первом автобусе. Третья строка содержит номера игроков, находящихсяво втором автобусе:

5 8

4 7 9 15 23

1 2 3 5 6 8 10 17

Выходные данные: номера футболистов, вышедших из автобусов в порядке возрастания. Выходные данные для данного примера:

1 2 3 5 6 7 8 9 10 15 17 23

**Идея решения:**

Оптимального решения данной задачи можно добиться, используя **метод сортировки слияниями.**

**(**Текст программы см. Приложение 4**)**

Задача о семьях.На сельской улице живут Ивановы и Петровы. Необходимо, используя минимальное число обменов, расселить их так, чтобы Ивановы жили с одного конца улицы, а Петровы – с другого.

Исходные и выходные данные. С клавиатуры вводится n - количество человек, проживающих на данной улице. Затем вводится массив А[1..n], состоящий из 0 и 1, где 0 – Петров, 1 – Иванов. Выходными данными является число обменов.

**Идея решения:**

Задача по **методам сортировки**. Один из способов её решения заключается в следующем. Пусть Ивановы должны жить в начале улицы, а Петровы – в конце. По индексу i (i<j) ищем первого Петрова, i увеличивается с шагом 1. Если нашли, то ищем Иванова с конца улицы – индекс j, он уменьшается. Если пара составлена, то совершаем обмен, и так до тех пор, пока i будет меньше j.

(Текст программы см. Приложение 5)

Метро.Дана схема метрополитена, с направлениями движения поездов до других станций. Станции пронумерованы. Необходимо составить алгоритм – программу, которая выводит номера станций, в которые можно попасть из станции с номером k (k вводится с клавиатуры). Схема метрополитена имеет следующий вид:

**1**

**2**

**3**

**4**

**5**

**6**

**Решение:**

Если входные данные представить в виде матрицы смежности путей метрополитена, то при помощи **алгоритма нахождения матрицы достижимости** можно решить данную задачу. Выходные данные: это индексы столбцов матрицы достижимости k – той строки, которые в значении имеют 1.

Исходные данные для данной задачи будут иметь вид:

6 {первая строка – это количество станций метро}

0 1 1 1 0 0

0 0 1 0 1 0

0 0 0 0 1 1

0 0 0 0 0 1

0 0 0 0 0 1

0 0 0 1 1 0

Пример выходных данных для данного примера:

Введите пункт отправки – 4

5 6

(Текст программы см. Приложение 6)

Роботы.Пункты с номерами 1,2,…,N (N<=50) связаны сетью дорог, длины которых равны 1. Дороги проходят на разной высоте и пересекаются только в пунктах. В начальный момент времени в некоторых пунктах находятся M роботов. Все роботы начинают двигаться с постоянной скоростью 1. Останавливаться или менять направление они могут только в пунктах.

1. Требуется найти минимальное время Т1, через которое все роботы могут встретиться в одном пункте, указать этот пункт или сообщить, что такая встреча невозможна.
2. Если встреча возможна, то найти время Т2<=T1, через которое встреча может произойти и вне пунктов.
3. Пусть роботам запрещена какая – либо остановка, и скорость равна 1 или 2. При этих условиях найти минимальное время Т, через которое произойдет их встреча, или сообщить, что встреча невозможна.

Примечания:

* + Для задачи (в) можно указать, что М равно 2 или 3.
  + При решении задач (а) и (б) данные о скоростях игнорируются.

**Решение.**

Идея решения основана на свойстве достижимости одной вершины из другой на графе.

Данные о связях между пунктами будем хранить в массиве Alink[1..n,1..n], элементы которого равны 0 или 1. Значение Alink[i,j]=1 говорит о том, что между пунктами i и j есть дорога.

**1**

**2**

**3**

**4**

**5**

**6**

**7**

**8**

**1**

В двумерном массиве Aplace[1..n,1..m] для каждого робота значениями, равными единице, будем указывать те населенные пункты, в которых данный робот может находиться в данный момент времени. Поясним логику решения на примере. Четыре робота находятся в пунктах 1,2,7,8.

Alink Aplace

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |  | 1 | 2 | 3 | 4 |
| 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 |
| 2 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 2 | 0 | 1 | 0 | 0 |
| 3 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 3 | 0 | 0 | 0 | 0 |
| 4 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 4 | 0 | 0 | 0 | 0 |
| 5 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 5 | 0 | 0 | 0 | 0 |
| 6 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 6 | 0 | 0 | 0 | 0 |
| 7 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 7 | 0 | 0 | 1 | 0 |
| 8 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 8 | 0 | 0 | 0 | 1 |

Первый робот может остаться в первом пункте или пойти во второй или третий пункты, в соответствии со связями в матрице Alink. Таким образом, в первом столбце матрицы Aplace во второй и третьей строках вместо 0 должны появиться 1. Изменения матрицы Aplace для роботов с номерами 2, 3 и 4 выполняются аналогичным образом.

Aplace через 1 момент времени Aplace в следующий момент времени

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 1 | 2 | 3 | 4 |
| 1 | 1 | 1 | 0 | 0 |
| 2 | 1 | 1 | 0 | 0 |
| 3 | 1 | 1 | 0 | 0 |
| 4 | 0 | 0 | 0 | 0 |
| 5 | 0 | 0 | 0 | 0 |
| 6 | 0 | 0 | 1 | 1 |
| 7 | 0 | 0 | 1 | 0 |
| 8 | 0 | 0 | 0 | 1 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 1 | 2 | 3 | 4 |
| 1 | 1 | 1 | 0 | 0 |
| 2 | 1 | 1 | 0 | 0 |
| 3 | 1 | 1 | 1 | 1 |
| 4 | 1 | 1 | 1 | 1 |
| 5 | 0 | 0 | 1 | 1 |
| 6 | 0 | 0 | 1 | 1 |
| 7 | 0 | 0 | 1 | 1 |
| 8 | 0 | 0 | 1 | 1 |

Итак, появилась строка или строки матрицы Aplace, состоящие из одних единиц. Эта строка будет соответствовать населенному пункту, в котором возможна встреча роботов.

Однако для пунктов

**1**

**2**

**3**

**4**

**5**

**6**

И при начальном расположении двух роботов в пунктах 1 и 6 встреча роботов никогда не произойдет, и строки Aplace, состоящей из одних единиц, не появится. Это требует введения второй матрицы (Aold), в которой должны фиксироваться достижимые пункты для каждого робота в предыдущий момент времени. Итак, если Aplace и Aold совпадают и нет ни одной строки, состоящей из одних единиц, то встреча роботов невозможна. Это схема решения первого задания. Решение второго задания отличается от первого тем, что требуется найти время Т2=Т1 – 1 (Т1 – время встречи роботов в одном населенном пункте), в которое все роботы находятся в одном из двух (произвольных) населенных пунктов, соединенных дорогой. В этом случае возможна их встреча и вне населенного пункта. Другими словами, в каждый момент времени необходимо проверять (находить) две строки матрицы Aplace, поэлементная логическая сумма которых дает строку, состоящую из одних единиц. При решении задания три матрицу Aplace следует не дополнять элементами, равными единице, а обновлять в соответствии со связями из матрицы Alink. Причем обновление выполнять не для всех роботов одновременно: в нечетные моменты времени 1,3,… для роботов, имеющих скорость 2, а в четные – 2, 4, …для всех роботов.

(Текст программы см. Приложение 7)

Вожатый в лагере**.** У вожатого в отряде дети разных возрастов от 10 до 17. каждое утро дети выходят на линейку, где они должны построится по старшинству (сначала старшие, затем младшие), но на первой линейке дети этого не знали и построились в произвольном порядке. Вожатый составил список возрастов построившихся. Необходимо составить алгоритм – программу, которая бы помогла вожатому как можно быстрее выстроить детей по старшинству.

**Решение.** Входные данные представляют собой список возрастов, который считывается из файла. Пример:

13 10 15 17 14 16 12 11

Выходные данные для данного примера:

17 16 15 14 13 12 11 10

Идея решения: задача решается с использованием методов сортировки. Так как в задаче указано, что необходимо выстроить детей как можно быстрее, то необходимо применить один из методов быстрой сортировки, например **метод Хоара,** эффективность данного алгоритма, по Д. Кнуту, составляет С=О(N\*logN). Для некоторых исходных данных время сортировки пропорционально О(N2). (Текст программы см. Приложение 8)

Егерь. У егеря в лесном хозяйстве 4 станции, уезжая в командировку, он оставил своему молодому напарнику, подробную карту, на которой изображены все дороги из одной станции в другую. В качестве приложения он оставил таблицу, в которую занес время, которое понадобиться напарнику, чтобы добраться из одной станции в другую, таблица имеет следующий вид:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **1** | **2** | **3** | **4** |
| **1** | 0 | 60 | 5 | 5 |
| **2** | 2 | 0 | 7 | 60 |
| **3** | 6 | 5 | 0 | 2 |
| **4** | 3 | 60 | 5 | 0 |

Где номер строки, это номер станции из которой напарник должен выйти, а номер столбца – это номер станции, в которую он должен попасть. Необходимо написать алгоритм-программу, которая укажет станции, через которые напарнику придется пройти, чтобы очутиться в нужной станции за минимальное время. Начальная и конечная станции вводятся с клавиатуры.

**Решение.** Данную таблицу можно рассматривать как матрицу смежности и построить по ней граф, который наглядно отобразит схему дорог из одной станции в другую. Таким образом можно применить один из алгоритмов поиска кратчайших путей, в данном случае наиболее удобно использовать **алгоритм Флойда**, который позволит не только найти минимальное время, которое потребуется напарнику, но и сам путь. Временная сложность данного алгоритма пропорциональна О(N3).

(Текст программы см. Приложение 9)

Игра «Найди друга». Всем ребятам выдаются карточки с номерами, они выстраиваются в ряд, по возрастанию номеров. Ребенку, который водит, также выдается карточка с номером. Считается, что ребенок нашел друга, если номер на его карточке совпадает с номером человека, к которому он подходит. Написать алгоритм – программу, которая позволит ребенку найти друга так, чтобы ребенок подходил к минимальному количеству участников. В случае если невозможно найти друга программа выводит результат «No», если же это возможно, то программа должна выводить количество детей, к которым подходил «вожа».

Примечание: номера детей определяются с помощью датчика случайных чисел, а номер ребенка, который водит, вводится с клавиатуры.

**Решение**. Так как мы знаем, что ребята расположены по возрастанию номеров на карточке, то наиболее быстрый способ найти друга можно реализовать с помощью **бинарного поиска.**

(Текст программы см. Приложение 10)

## Приложение.

1 **Комнаты музея.**

Uses crt;

Const n=100;

X:array[0..3]of -1..1=(0,-1,0,1); {массив координат перемещения по

Y:array[0..3]of -1..1=(-1,0,1,0); клеткам. Индекс элемента массива

Type Mas=array[0..n,0..n]of Integer; соответствует степени двойки}

var A:mas;

B:array[0..n,0..n]of Boolean;

m,p,col,rooms,indexX,indexY:integer;

procedure Init(Z:string); {заполнение из входного файла массива, представляющего цифровую карту музея}

Var f:text;

i,j:integer;

Begin

Assign(f,z);

Reset(f);

ReadLn(f,m,p);

For i:=1 to m do

begin

For j:=1 to p do

Read(f,A[i,j]);

ReadLn(f);

end;

FillChar(B,SizeOf(B),true);

For i:=1 to m do

For j:=1 to p do

B[i,j]:=false;

Close(f);

end;

function Degree2(i:integer):integer; {функция, вычисляющая i–ую степень двойки}

var j,t:integer;

begin

t:=1;

For j:=1 to i do

t:=t\*2;

Degree2:=t;

end;

Procedure Solve(i,j:integer);

Var k:integer;

begin

k:=3;

While k>=0 do

begin

If A[i,j]<Degree2(k)then {смотрим имеет ли клетка стену в заданном направлении}

begin

If not B[i+X[k],j+Y[k]] then {определяем, заходили ли мы в клетку ранее}

begin

Inc(col); {учитываем клетку в общей площади комнаты}

B[i,j]:=true; {отмечаем, что в текущей клетке мы уже были}

Solve(i+X[k],j+Y[k]); {переходим в следующую клетку}

B[i,j]:=False; {делаем клетку, в которой последний раз были не просмотренной, чтобы рассмотреть другие варианты хода из неё в другую клетку}

end;

end

Else A[i,j]:=A[i,j]-Degree2(k);

Dec(k);

end;

end;

procedure Prosmotr; {данная процедура отмечает уже просмотренную комнату}

var i,j:integer;

begin

For i:=1 to m do

For j:=1 to p do

If A[i,j]=0 then B[i,j]:=True;

end;

begin

clrscr;

Init('A:museum.txt');

rooms:=0;

For indexX:=1 to m do {ищем ранее не просмотренную клетку}

For indexY:=1 to p do

If not B[indexX,indexY] Then

begin

col:=1;

Inc(rooms);

Solve(indexX,indexY);

Write(Col,' '); {вывод площади только что просмотренной комнаты}

Prosmotr;

end;

WriteLn;

WriteLn(rooms); {вывод количества комнат}

readkey;

end.

2 **Пират в подземелье.**

uses crt;

Const k=100;

dx:array[1..4] of Integer=(1,0,-1,0); {массив координат перемещения пирата}

dy:array[1..4] of Integer=(0,1,0,-1);

Type mas=array[0..k,0..k]of Integer;

mas2=array[0..k,0..k]of boolean; {массив логического типа для пометки комнат, в которых пират уже побывал}

var n,m,sum1,sum,col:integer;

A:mas;

B:mas2;

Procedure Init(z:string); {инициализация входных данных}

Var f:text;

i,j:integer;

Begin

Assign(f,z);

Reset(f);

FillChar(A,SizeOf(A),0);

FillChar(B,SizeOf(B),true);

ReadLn(f,n,m,col);

for i:=1 to n do

begin

for j:=1 to m do

Read(f,A[i,j]);

ReadLn(f);

end;

Close(f);

End;

Procedure Solve(x,y,p:integer);

var i,j:integer;

begin

If p=0 then begin

If sum>sum1 then {сравниваем текущую стоимость набранных камней со стоимотью набранных ранее, с целью увеличения стоимости}

sum1:=sum;

end

Else begin

For i:=1 to 4 do

If (A[x+dx[i],y+dy[i]]>0)and B[x+dx[i],y+dy[i]] then {просматриваем варианты перехода пирата в другую комнату, проверяя не был ли пират в ней до этого}

begin

sum:=sum+A[x+dx[i],y+dy[i]]; {прибавляем стоимость камня, находящегося в данной комнате к суммарной стоимости}

B[x+dx[i],y+dy[i]]:=false; {отмечаем, что в данной комнате мы уже были}

Solve(x+dx[i],y+dy[i],p-1);

sum:=sum-A[x+dx[i],y+dy[i]];

B[x+dx[i],y+dy[i]]:=true;

end;

end;

end;

begin

clrscr;

Init('A:241.txt');

sum1:=0; sum:=A[1,1];

Solve(1,1,col);

WriteLn('Result= ',sum1);

readkey;

end.

3 **Диспетчер и милиция.**

Uses crt;

Const n=100;

Type mas=array[1..n,1..n]of Integer;

mas1=array[1..n]of Integer;

mn=Set of 1..n;

Var m,first,last:integer;

D:mas1;

A:mas;

procedure Init(z:string); {инициализация входных данных}

Var i,j:integer;

f:text;

begin

Assign(f,z);

Reset(f);

ReadLn(f,m);

For i:=1 to m do

begin

For j:=1 to m do

Read(f,A[i,j]);

ReadLn(f);

end;

Close(f);

end;

function MinZn(R:mn):integer; {вычисляет номер района, путь до которого из района отправления минимален}

var i,minn:integer;

Begin

minn:=MaxInt;

For i:=1 to m do

If (D[i]<minn)and(D[i]>0)and(i in R) then

begin

MinZn:=i;

minn:=D[i];

end;

End;

Function Min(i,j:integer):integer;{возвращает минимальное значение из двух возможных}

Begin

If i<>0 then

begin

If j<>0 then

begin

If j<i then Min:=j else Min:=i;

end Else Min:=i;

end Else Min:=j;

End;

procedure Milicia(s:integer);

var v,u:integer;

T:mn;

Begin

for v:=1 to m do D[v]:=A[s,v];

D[s]:=0; T:=[1..m]-[s];

While T<>[] do

Begin

u:=MinZn(T);

T:=T-[u];

For v:=1 to m do

If v in T then

If A[u,v]<>0 Then

D[v]:=Min(D[v],D[u]+A[u,v]);

end;

End;

Begin

clrscr;

Init('A:milicia.txt');

WriteLn('Введите пункт отправления и пункт назначения');

ReadLn(first,last);

Milicia(first);

WriteLn(D[last]);

readkey;

End.

4 **Задача о футболистах.**

uses crt;

Const k=100;

Type mas=array[1..k]of Integer;

Var m,q:integer;

A,B:mas;

procedure Init(z:string); {инициализация исходных данных}

var i:integer;

f:text;

begin

Assign(f,z);

Reset(f);

ReadLn(f,m,q);

For i:=1 to m do

Read(f,A[i]);

ReadLn(f);

For i:=1 to q do

Read(f,B[i]);

Close(f);

end;

procedure Solve;

var i,j,t:integer;

D:mas;

begin

i:=1; j:=1; t:=1;

While (i<=m)and(j<=q)do {пока не вышли футболисты хотя бы из одного автобуса}

Begin

{сравниваем номера футболистов в разных автобусах, выходит в строй футболист с наименьшим номером}

If A[i]<=B[j] Then begin D[t]:=A[i]; Inc(i); end

Else begin D[t]:=B[j]; Inc(j); end;

Inc(t);

end;

{из одного автобуса вышли все футболисты, осталось выйти остальным}

While i<=m do begin D[t]:=A[i]; Inc(i); Inc(t); end;

While j<=q do begin D[t]:=B[j]; Inc(j); Inc(t); end;

For i:=1 to t-1 do Write(D[i],' ');

end;

begin

clrscr;

Init('A:socker.txt');

Solve;

readkey;

end.

5 **Задача о семьях.**

Uses crt;

Const MaxN=1000;

Var A:array[1..maxN]of byte;

N, cnt,i,j:integer;

Procedure Swap(var a,b:byte);

Var c:byte;

Begin

c:=a; a:=b; b:=c;

End;

Begin

Write(‘введите N’); readln(N);

Write(‘введите массив через пробел(0 – Петров, 1 - Иванов)’);

For i:=1 to N do read(A[i]);

i:=1; j:=N; cnt:=0;

While i<j do

If A[i]=1 then Inc(i) else

If A[j]=0 then Dec(j) else begin

Swap(A[i],A[j]);

Inc(i); dec(j);

Inc(cnt);

End;

writeLn(‘Число обменов - ’, cnt);

End.

6 **Метро.**

uses crt;

const p=100;

Type mas=array[1..p,1..p]of 0..1;

var k,n:integer;

A:mas;

procedure Init(z:string); {инициализация данных}

var f:text;

i,j:integer;

begin

Assign(f,z);

Reset(f);

ReadLn(f,n);

For i:=1 to n do

begin

For j:=1 to n do

Read(f,A[i,j]);

ReadLn(f);

end;

Close(f);

end;

procedure Get(i:integer); {i – номер станции, из которой необходимо отправится}

var S,T:Set of 1..p;

j,l:integer;

begin

T:=[i];

Repeat

S:=T;

For l:=1 to n do

If l in S then {по строкам матрицы смежности А, принадлежащим множеству S}

For j:=1 to n do

If A[l,j]=1 Then T:=T+[j]; {смотрим если есть путь из данного пункта в пункт j, то добавляем номер пункта j в множество Т}

Until S=T;

For j:=1 to n do

If (j in T)and(i<>j) then Write(j,' '); {просматриваем содержится ли номер пункта j в множестве имеющих путь из пункта i}

end;

begin

clrscr;

Init('A:metro.txt');

readLn(k);

Get(k);

readkey;

end.

7 **Роботы.**

Program Robots;

Const max=50;

Type Sset=Set of 1..max;

Mas=array[1..max]of Sset;

Var A,B:Mas;

{A – матрица достижимостей, B[i] – какие роботы могут быть в i пункте}

SOne, STwo: SSet; {SOne – роботы, которые едут со скоростью 1, STwo – роботы, которые едут со скоростью 2}

N, M:integer; {N – число пунктов, M – число роботов}

Procedure Init; {инициализация входных данных}

Var K, i, FrP, ToP:integer;

Begin

FillChar(A,SizeOf(A),0);

Write(‘Число пунктов:’); ReadLn(N);

Write(‘Число дорог:’); ReadLn(K);

For i:=1 to K do begin

writeLn(‘Введите пункты, которые соединяет дорога №’, i);

ReadLn(FrP, ToP);

Include(A[FrP],ToP);

Include(A[ToP],FrP);

End;

Write(‘Число роботов:’); ReadLn(M);

For i:=1 to M do Begin

Write(‘Пункт, где находится робот №’,i,’:’); ReadLn(K);

Include(B[k],i);

Write(‘скорость робота №’,i,’:’);

ReadLn(k);

If K=1 then Include(SOne,i) Else Include(STwo,i);

End;

End;

Function ProvCanMet: Boolean;

Var i:integer;

Begin

i:=1;

While (i<=N)and(B[i]<>[1..M])do Inc(i);

ProvCanMet:=i<=N;

End;

Function InTwoNear: Boolean;

Var i,j:integer;

Begin

i:=1; j:=N+1;

while (i<N)and(j>N)do begin

j:=i+1;

while(j<=N)and Not((j in A[i])and(B[i]+B[j]=[1..M]))do Inc(j);

Inc(i);

End;

InTwoNear:=j<=N;

End;

Function AddIfCan(mode:integer; S:Sset):Boolean;

Var i,j:integer;

C:mas;

Begin

AddIfCan:=false; {S – множество роботов, которые едут}

If mode=0 then

For i:=1 to N do C[i]:=B[i]-S

Else C:=B;

For i:=1 to N do

For j:=1 to N do

If (i<>j)and(j in A[i])and(C[i]\*B[j]\*S<>B[j]\*S) Then Begin

AddIfCan:=true;

C[i]:=C[i]+B[j]\*S;

End;

B:=C;

End;

Function InTwoForC: byte;

Var i,j:integer;

Begin

i:=1; j:=N+1;

while (i<N)and(j>N)do begin

j:=i+1;

While (j<=N)and (not(j in A[i])or(B[i]+B[j]<>[1..m])or Not((SOne=[])or(STwo=[])or((B[i]\*SOne=SOne)and(B[j]\*STwo=STwo))or (B[j]\*SOne=SOne)and(B[i]\*STwo=STwo)))do Inc(j);

Inc(i);

End;

If j>N Then InTwoForC:=0 Else

If STwo=[] Then InTwoForC:=1 Else

If SOne=[] Then InTwoForC:=2 Else

InTwoForC:=3;

End;

Procedure SolveC;

Var time:integer;

FindS, IncS: Boolean;

ForMet: integer;

Begin

Time:=0;

IncS:=true;

ForMet:=InTwoForC;

FindS:=ProvCanMet;

While IncS and Not FindS and(time<=N\*2)and(ForMet=0)do begin

Inc(time);

If Time Mod 2=0 then IncS:=AddIfCan(0,[1..m])

Else incS:=AddIfCan(0,STwo);

ForMet:=InTwoForC;

FindS:=ProvCanMet and(time mod 2=1);

End;

If Time>N\*2 then WriteLn(‘Пункт В: Роботы не встретятся’)

Else begin

Write(‘Пункт В: Роботы встретятся через’);

If FindS Then Write(Time/2:0:3)

Else Case ForMet of

1: write((time+1)/2:0:3);

2: write(time/2+1/4:0:3);

3: write(time/2:0:3,’+1/’,(time mod 2+1)\*3);

End;

WriteLn(‘Момент(а,ов) времени’);

End;

End;

Procedure SolveAB;

Var time:integer;

ForB, FindS, IncS: Boolean;

Old:mas;

Begin

Old:=B;

Time:=0;

IncS:=true; FindS:=ProvCanMet;

While IncS and Not FindS do begin

ForB:=InTwoNear;

Inc(time);

incS:=AddIfCan(1,[1..m]);

FindS:=ProvCanMet;

End;

If FindS Then begin

WriteLn(‘Пункт А:’,time,’момент(а,ов) времени’);

WriteLn(‘Пункт Б:’,time – Byte(ForB)\*0.5:0:1,’момент(а,ов) времени’);

SolveC;

End

Else begin

WriteLn(‘Пункт А: Роботы не встретятся’);

writeLn(‘Пункт Б: Роботы не встретятся’);

writeLn(‘Пункт В: Роботы не встретятся’);

end;

B:=Old;

End;

Begin

Init;

SolveAB;

End.

8 **Вожатый в лагере.**

uses crt;

Const k=50;

Type mas=array[1..k]of integer;

var col:integer;

A:mas; {массив представляющий собой список возрастов детей}

procedure Init(z:string); {инициализация данных}

var i:integer;

f:text;

begin

Assign(f,z);

Reset(f);

i:=0;

While not EoLn(f) do

begin

Inc(i);

Read(f,A[i]);

end;

col:=i;

Close(f);

end;

procedure Print; {вывод списка на экран}

var i:integer;

begin

For i:=1 to col do

Write(A[i],' ');

end;

procedure Solve(m,t:integer);

var i,j,w,x:integer;

begin

If m>=t then exit;

i:=m; j:=t; x:=A[(m+t)div 2]; {x- барьерный элемент, т.е. возраст, относительно которого будет сортироваться список, i,j – нижний и верхний номер, рассматриваемой части списка}

While i<j do

If A[i]>x then Inc(i)else {смотрим элементы списка относительно

If A[j]<x then Dec(j)else барьерного элемента, пока не найдем из правой и

Begin левой части по элементу, которые стоят не на

w:=A[i]; A[i]:=A[j]; A[j]:=w; своем месте. Меняем их местами}

end;

Solve(m,j-1); Solve(i+1,t); {ищем далее барьерный элемент, сначала в правой

end; части списка, затем в левой}

begin

clrscr;

Init('A:alfa.txt');

Print;

WriteLn;

Solve(1,col);

Print;

readkey;

end.

9 **Егерь.**

Program Eger;

uses crt;

Const n=4;

var A,P,D:array[1..n,1..n]of Integer; {A – матрица смежности; D – массив кратчайших путей, где D[i,j] – минимальное время, которое потребуется, чтобы добраться из станции i до станции j; P – массив, элементами которого являются номера станций, которые будут составлять путь с минимальным временем}

k,m:integer; {начальная и конечная станции движения}

procedure Init(z:string); {инициализация данных}

var i,j:integer;

f:text;

begin

Assign(f,z);

Reset(f);

For i:=1 to n do

begin

For j:=1 to n do

Read(f,A[i,j]);

ReadLn(f);

end;

Close(f);

end;

Procedure Solve;

var i,j,k:integer;

begin

For i:=1 to n do

For j:=1 to n do

begin

D[i,j]:=A[i,j];

P[i,j]:=i;

end;

for k:=1 to n do begin

for i:=1 to n do

for j:=1 to n do

If D[i,j]>D[i,k]+D[k,j] then begin {определение пути с минимальным

D[i,j]:=D[i,k]+D[k,j]; временем}

P[i,j]:=k; {заносим номер станции, которая будет

end; предпоследней, посещенной напарником}

end;

end;

procedure Way(i,j:integer); {рекурсивная процедура, выводит

begin последовательность станций, которые посетит

If P[i,j]<>i then begin напарник, отталкиваясь от данных,

Way(i,P[i,j]); занесенных в массив P}

Write (P[i,j]:2,'->');

Way(P[i,j],j);

end

end;

begin

clrscr;

Init('A:eger.txt');

Solve;

Writeln('Введите из какой станции и в какую будем искать путь:');

Readln(k,m);

Write(k:2,'->');

Way(k,m);

WriteLn(m:2);

WriteLn(‘Время пути= ‘,D[k,m]);

readkey;

end.

10 **Игра «Найди друга».**

uses crt;

Const n=20;

type mas=array[1..n]of Integer;

var A:mas;

X,b:integer;

procedure Init(z:string);

var i:integer;

f:text;

begin

Assign(f,z);

Reset(f);

For i:=1 to n do

Read(f,A[i]);

Close(f)

end;

procedure Print;

var i:integer;

begin

For i:=1 to n do

Write(A[i],' ');

end;

procedure Solve(i,j:integer;Var t:integer);

var m:integer;

begin

If i>j then Writeln('No')

else begin m:=(i+j)div 2;

Inc(b);

If A[m]<X then Solve(m+1,j,t)

else If A[m]>X then Solve(i,m-1,t)

else Write(b);

end;

end;

begin

clrscr;

Init('A:game.txt');

Print;

WriteLn;

ReadLn(x);

Solve(1,n,b);

readkey;

end.

Заключение.

В данном курсовом проекте мы разработали свой набор задач и критерии, по которым данный набор можно классифицировать. Несмотря на то, что разрабатывая критерии классификации, мы оперировали с конкретным набором задач, данная классификация может быть применима ко многим наборам задач. Единственное несоответствие, которое может произойти, это несоответствие по тематике. Таким образом, данная классификация достаточно универсальна и может иметь широкое практическое применение. При выполнении данного курсового проекта основные трудности пришлись на выбор литературы, так как по данной теме литературы немного и ее необходимо рассматривать с точки зрения методики преподавания информатики. В сборниках задач большое место отведено задачам, имеющим строгую формулировку, которую изменить на ситуативную достаточно сложно, так как задачи имеют маленькую практическую значимость в жизни.

Таким образом, цели поставленные при выполнении данного курсового проекта достигнуты.

Литература**:**

1. **Б.Н. Иванов Дискретная математика. Алгоритмы и программы. Москва 2001г.**
2. **С.М. Окулов Программирование в алгоритмах. Москва 2002г.**
3. **Н.Вирт Алгоритмы и структуры данных. Москва «Мир» 1989г.**
4. **В.М. Кирюхин, А.В. Лапунов, С.М. Окулов Задачи по информатике. Международные олимпиады 1989-1996гг. Москва ABF 1996г.**
5. **С.М. Окулов, А.А. Пестов, О.А. Пестов Информатика в задачах. Киров 1998г.**
6. **Н.Вирт Систематическое программирование. Под ред. Ю.М. Баяковского. Москва «Мир» 1977г.**