Нижегородский Ордена Трудового Красного Знамени
Государственный Университет им. Н.И. Лобачевского

Экономический факультет

Кафедра информатики и вычислительной техники

**Курсовая работа
по программному обеспечению**

тема:

**Решение задачи о коммивояжере**

**Выполнили:**

Шапошников А.Д.

Яров Е.И.

**Научный руководитель:**

Громницкий В.С.

Нижний Новгород
1995 год

# Оглавление

Оглавление 2

Введение 3

Постановка задачи 4

Алгоритм решения 4

Математическая модель задачи 5

Описание программной реализации алгоритма 6

Описание программного интерфейса. 9

Описание программы 12

Литература 15

# Введение

В настоящее время быстро развивается научно-техническая революция. Появившись в 40-х годах нашего столетия компьютеры и компьютерные технологии прошли за это время путь от ламповых систем с прямым заданием кодов операций до сверхбыстрых персональных компьютеров на монокристальной технологии, использующих при работе многопользовательские операционные системы с графическим интерфейсом. Наиболее бурное развитие компьютерных технологий произошло за последние 10-15 лет, после того как была разработана технология производства СБИС, а на их основе микрочипов. Также в начале 80-х годов начала развиваться концепция персональной ЭВМ, которая со временем выразилась в существовании двух наиболее распространенных аппаратных платформ: Macintosh (производится фирмой Apple, процессоры фирмы Motorola) и IBM PC (производится фирмой IBM, процессоры фирмы Intel).

Каждая из этих платформ имеет свою направленность и особенности. Компьютеры Macintosh в основном ориентированы на работу в глобальных сетях и обработку информации, как бы внешнего восприятия, то есть графической и звуковой. Их главными особенностями являются встроенная в ПЗУ (ROM) графическая операционная система и несовместимость различных моделей этой фирмы, однако за счет этого достигнут очень быстрый рост производительности процессоров и системы в целом.

Фирма IBM, в отличие от Apple, придерживается концепции открытой системы, что выражается в том, что, во-первых, IBM не имеет эксклюзивного права на производство и продажу IBM-совместимых компьютеров (их производит множество фирм, таких как Hewlett Packard, AT&T, Compaq и другие), а также полной совместимости поздних моделей с более ранними, что позволяло IBM долгое время удерживать рынок сбыта, несмотря на то, что в начале 90-х годов Macintosh-и заметно превосходили PC по производительности (сейчас, с появлением Pentium и PowerPC, Macintosh-и потеряли безоговорочное лидерство по производительности).

Персональные компьютеры сейчас в основном используются в четырех областях:

1. обработка текстов и компьютерная верстка;
2. хранение баз данных с возможностью быстрой их обработки;
3. управление производственными процессами;
4. анализ сложных процессов;

Также сейчас интенсивно развиваются еще несколько областей применения ПЭВМ, таких как компьютерные игры (в 1994 году 43% рынка программных продуктов составляли игровые программы), гео-информационные системы, обучающие системы и системы мультимедиа.

Программа данной курсовой работы входит в раздел "анализ сложных процессов". Данная область начала развиваться в середине 80-х годов, когда производительность ПЭВМ достигла достаточного уровня для обработки необходимых объемов информации в реальном масштабе времени, что позволило начать применение компьютеров в областях, связанных с контролем сложных процессов и их анализом. Одной из таких проблем стала оптимизация систем со многими неизвестными, когда некоторый параметр было необходимо свести к какому-либо значению или просто максимизировать.

Наиболее ярким и характерным примером применения задачи "О коммивояжере" стала оптимизация операций на конвейере: в 1984 году, после того как был проведен анализ последовательности и временных затрат на операции на конвейерах заводов компании "General Motors" и приняты рекомендованные меры, удалось увеличить общую производительность почти на 13% при неизменном числе рабочих и том же оборудовании. Расчеты производились на компьютерах IBM 360gr, которые в то время являлись одними из самых производительных систем в мире. Просчет и оптимизация 200 основных и 87 вспомогательных операций занял около 230 часов. Считается, что это было первое коммерческое применение компьютерных технологий в области управления производством в целом.

Сейчас решение данной задачи необходимо во многих областях связанных с замкнутыми и при этом жестко связанными по времени системами, такими как: конвейерное производство, многооперационные обрабатывающие комплексы, судовые и железнодорожные погрузочные системы, перевозки грузов по замкнутому маршруту, расчет авиационных линий.

Поэтому данная проблема на современном этапе развития общества имеет не самое последнее по значимости место.

# Постановка задачи

Классическая постановка задачи о коммивояжере выглядит следующим образом:

Имеется N городов, которые должен обойти коммивояжер с минимальными затратами. При этом на его маршрут накладывается два ограничения:

1. маршрут должен быть замкнутым, то есть коммивояжер должен вернуться в тот город, из которого он начал движение;
2. в каждом из городов коммивояжер должен побывать точно один раз, то есть надо обязательно обойти все города, при этом не побывав ни в одном городе дважды.

Для расчета затрат существует матрица условий, содержащая затраты на переход из каждого города в каждый, при этом считается, что можно перейти из любого города в любой, кроме того же самого (в матрице как бы вычеркивается диагональ). Целью решения является нахождения маршрута, удовлетворяющего всем условиям и при этом имеющего минимальную сумму затрат.

# Алгоритм решения

В курсовой работе для решения задачи о коммивояжере применяется метод ветвей и границ, относящийся к методам дискретной оптимизации. В сущности, это полный перебор решений, который оптимизируется за счет того, что при переборе вариантов по определенным признакам отсекаются неоптимальные множества перебора. Так как количество вершин от уровня к уровню возрастает в факториальной прогрессии, то отсечение вершин верхних уровней значительно сокращает общее число перебираемых вариантов.

Общая схема метода такова (данная программная реализация):

1. Все множество разбивается на N-1 подмножеств, каждое из которых оценивается верхней и нижней оценками.
2. Производится отсев неоптимальных множеств по следующему критерию: если нижняя оценка (решение релаксированной задачи) больше минимальной из верхних оценок (решений нерелаксированной задачи), то множество считается очевидно неоптимальным и отсекается, иначе остается до следующей итерации.
3. Находится множество с лучшей нижней оценкой (прогнозом) и дробится на количество равное размерность исходной матрицы минус количество уже пройденных (фиксированных для данного множества) городов.
4. Находятся минимальные верхняя и нижняя оценка. Если они равны и достигнуты на одном и том же множестве, то это значит, что получено оптимальное решение и алгоритм заканчивает работу, иначе возврат к шагу 2.

# Математическая модель задачи

N - число городов.

Ci j , i, j=1..N - матрица затрат, где Ci j - затраты на переход из i-го города в j-й.

Xi j - матрица переходов с компонентами:

Xi j = 1, если коммивояжер совершает переход из i-го города в j-й,

Xi j = 0, если не совершает перехода,

где i, j = 1..N и i≠j.

Критерий:

 (1)

Ограничения:

, i = 1..N (2)

, j = 1..N (3)

Ui - Uj + N ⋅ Xi j ≤ N-1, i, j = 1..N, i ≠ j. (4)

Доказательство, что модель (1-4) описывает задачу о коммивояжере:

Условие (2) означает, что коммивояжер из каждого города выезжает только один раз; условие (3) - въезжает в каждый город только один раз; условие (4) - обеспечивает замкнутость маршрута, содержащего N городов, и не содержащего замкнутых внутренних петель.

Рассмотрим условие (4). Применим метод доказательства от противного, то есть предположим, что условие (4) выполняется для некоторого подцикла T из R городов, где R<N. Сложив все неравенства (4) вдоль этого подцикла получим

.

Так как

,

то N ⋅ R ≤ (N -1), где R<N, R ≠ 0.

Следовательно, не существует замкнутого подцикла с числом городов меньшим, чем N.

Покажем, что существует Ui, которое для замкнутого цикла, начинающегося в некотором начальном пункте, удовлетворяют условию (4). При всех Xi j (j-й город не посещается после i-го) в (4) имеем Ui - Uj ≤ N - 1, что допустимо в силу произвольных Ui и Uj.

Пусть на некотором R-ом шаге i-й город посещается перед j-м, то есть Xi j = 1. В силу произвольности значений Ui и Uj положим Ui = R, а Uj = R + 1, тогда из (4) имеем:

Ui - Uj + N ⋅ Xi j ≤ R - (R - 1) + N = N - 1

Итак, существуют такие конечные значения для Ui и Uj, что для маршрута, содержащего N городов, условие (4) удовлетворяется как неравенство или строгое равенство. А следовательно, модель (1-4) описывает задачу о коммивояжере.

# Описание программной реализации алгоритма

В программе реализован классический вариант метода ветвей и границ, то есть алгоритм решения следующий:

1. Все множество разбивается на N-1 подмножеств, каждое из которых оценивается верхней и нижней оценками.
2. Производится отсев неоптимальных множеств по следующему критерию: если нижняя оценка (решение релаксированной задачи) больше минимальной из верхних оценок (решений нерелаксированной задачи), то множество считается очевидно неоптимальным и отсекается, иначе остается до следующей итерации.
3. Находится множество с лучшей нижней оценкой (прогнозом) и дробится на количество равное размерность исходной матрицы минус количество уже пройденных (фиксированных для данного множества) городов.
4. Находятся минимальные верхняя и нижняя оценка. Если они равны и достигнуты на одном и том же множестве, то это значит, что получено оптимальное решение и алгоритм заканчивает работу, иначе возврат к шагу 2.

При этом предусмотрена возможность смены направления критерия, то есть возможность решения задачи на максимум.

Наибольший выбор программа предоставляет в шаге 3, где реализовано четыре метода расчета нижней оценки множества и три метода расчета верхней оценки множества. Сначала рассмотрим методы нахождения нижней оценки, то есть решения релаксированной задачи:

**Метод 1: "Из каждого города".**

Рассчитывается цена маршрута по зафиксированным для данной вершины городам. Затем в каждой строке матрицы, в которой нет фиксированных элементов, находится минимальный элемент (в программе реализовано функцией Min\_Col) и прибавляется к общей сумме, то есть берутся ближайшие города для выхода из всех еще не пройденных городов. Программная реализация ‑ VHCOUNT. H\_1

**Метод 2: "В каждый город".**

Рассчитывается цена маршрута по зафиксированным для данной вершины городам. Затем в каждом столбце матрицы, в котором нет фиксированных элементов находится минимальный элемент (в программе реализовано функцией Min\_Str) и прибавляется к общей сумме, то есть берутся ближайшие города для входа во все еще не пройденные города. Программная реализация ‑ VHCOUNT. H\_2

**Метод 3: "Комбинированный".**

Рассчитываются оценки по предыдущим двум методам и из них выбирается максимальная, то есть худший прогноз для данной вершины. Программная реализация - VHCOUNT. H\_12

**Метод 4: "Венгерский метод".**

Программная реализация - SOLUTION. DOWNLEV

Рассчитывается цена маршрута по зафиксированным для данной вершины городам. Затем формируется матрица из элементов незанятых строк и столбцов размерностью M×M, где M = N ‑ количество пройденных городов. Эта матрица передается венгерскому алгоритму, который описан ниже (Программная реализация - VENGRSOL. CENTRAL\_CONTROL):

*Предварительный этап.*(В программе реализован процедурой Begin\_Set)

Разыскивают минимальный элемент в столбце и из всех элементов этого столбца последовательно вычитают минимальный. Эту операцию проделывают над всеми столбцами матрицы. В результате образуется матрица, в каждом столбце которой имеется по крайней мере один нуль.

Рассматриваем строку полученной матрицы и из каждого ее элемента вычитаем минимальный элемент этой строки. Проведя эту операцию с каждой строкой, получаем матрицу с неотрицательными элементы, в каждом столбце и строке которой имеется по крайней мере один нуль. Отметим произвольный нуль в первом столбце звездочкой (\*). Далее просматриваем второй столбец, и если в нем есть нуль, расположенный в строке, где нет нуля со звездочкой, то отмечаем звездочкой. Аналогично просматриваем все столбцы и на этом предварительный этап заканчивается. (В программе реализовано процедурой Make\_Label\_Zero).

*(К+1)-я итерация.*

Допустим, что К-я итерация уже проведена и в результате получена некоторая матрица. Если в матрице N нулей со звездочкой (\*), то процесс решения заканчивается. Если же в матрице нулей со звездочкой меньше N, то переходим к (К+1)-й итерации. (В программе проверяется процедурой Central\_Control)

Каждая итерация начинается первым и заканчивается вторым этапом. Между ними может несколько раз проводиться третий этап. Перед началом итерации значком (+) выделяют столбцы матрицы, содержащие нули со звездочкой (0\*). (В программе реализовано процедурой Make\_Label\_Col)

*ПЕРВЫЙ ЭТАП.*

Просматривают невыделенные столбцы (если первый этап проводится после третьего, то также отсекают и выделенные строки). (В программе реализовано процедурой Check\_Zero). Если среди них не окажется нулевых элементов, то переходят к третьему этапу.

Если же невыделенный нуль в матрице обнаружен, то возможен один из двух случаев :

1. в строке, содержащей нуль, имеется также нуль со звездочкой (0\*) ;
2. эта строка не содержит нуль со звездочкой.

(Проверка случая производится в процедуре Central\_Control)

В первом случае невыделенный нуль отмечают штрихом ( ' ) и выделяют строку, в которой он содержится, постановкой справа от нее значком (+). Затем уничтожают знак (+) над столбцом, содержащим нуль со звездочкой (0\*).

Далее просматривают этот столбец, отыскивают в нем невыделенный нуль (поиск идет только среди непомеченных строк), непомеченный звездочкой, отмечают его штрихом и выделяют строку, содержащую такой нуль. Затем просматривают эту строку, отыскивая в ней нуль со звездочкой (0\*). Этот процесс за конечное число шагов заканчивается одним из следующих исходов :

1. все нули матрицы выделены, то есть находятся в выделенных строках и столбцах. При этом переходят к третьему этапу;
2. имеется невыделенный нуль в строке, где нет нуля со звездочкой. В этом случае переходят ко второму этапу, отметив последний по порядку нуль штрихом ( ' ).

(В программе реализовано процедурой Find\_Zero и подпроцедурами Find\_Zero\_in\_Col и Find\_Zero\_in\_Line; выбор случая производится процедурой Central\_Control)

*ВТОРОЙ ЭТАП.*

Строят следующую цепочку из элементов матрицы : исходный нуль со штрихом (0' ), нуль со звездочкой (0\*), расположенный в одном столбце с первым, нуль со штрихом (0' ), расположенный в одной строке с предшествующим нулем со звездочкой (0\*), и так далее. Итак, цепочка образуется передвижением от 0' к 0\* по столбцу, от 0\* к 0' по строке и так далее. (В программе реализовано процедурой Chain подпроцедурами Find\_Link\_in\_Col и Find\_Link\_in\_Line, а также внутренней подпроцедурой процедуры Chain процедурой NewLink)

Далее над элементами цепочки, стоящими на нечетных местах (0' ), ставим звездочки, уничтожая их над четными элементами (0\*). (В программе реализовано процедурой Change\_Chain). Затем уничтожаем все штрихи над элементами матрицы и знаки +. (В программе реализовано процедурой Erase\_Label) При этом количество независимых нулей будет увеличено на единицу. (К+1)-я итерация закончена.

*ТРЕТИЙ ЭТАП.*

К этому этапу переходят после первого, если все нули матрицы выделены, то есть находятся в выделенных строках и столбцах. В таком случае среди невыделенных элементов матрицы выбирают минимальный элемент и обозначают его H (H>0). Далее вычитают H из всех элементов матрицы, стоящих в невыделенных строках и прибавляют ко всем элементам матрицам, расположенным в выделенных столбцах. Получают новую матрицу, эквивалентную исходной. (В программе реализовано процедурами Find\_Min\_No\_Label (нахождение минимального элемента) и Plus\_Minus (вычитание и прибавление)).

Поскольку среди невыделенных элементов появятся новые нули (согласно определению), переходят к первому этапу, рассматривая преобразованную матрицу. Завершив первый этап, либо переходят ко второму, либо вновь к третьему этапу, если все нули матрицы оказываются выделенными. (В программе выбор реализован в процедуре Central\_Control ).

В первом случае после проведения второго этапа итерация заканчивается, а во втором - после проведения третьего этапа получают новую матрицу, в которой будут невыделенные нули, и всю последовательность операций, начиная с первого этапа, необходимо повторить. После конечного числа повторений очередной первый этап обязательно закончится переходом на второй этап и количество независимых нулей увеличится на единицу. (К+1)-я итерация закончена.

Теперь можно перейти к рассмотрению методов получения верхней оценки:

**Метод 1: "По возрастанию номеров".**

Рассчитывается цена маршрута по зафиксированным для данной вершины городам. Затем работает следующий алгоритм нахождения маршрута:

1. Рассчитывается число пройденных городов.
2. Если пройдены все города, то выход, иначе из последнего города незаконченного маршрута добавляется переход в еще непройденный город с минимальным номером и возврат к шагу 1.

Программная реализация ‑ VHCOUNT. V\_1

**Метод 2: "С оптимизацией".**

Рассчитывается цена маршрута по зафиксированным для данной вершины городам. Затем работает следующий алгоритм нахождения маршрута:

1. Рассчитывается число пройденных городов.
2. Если пройдены все города, то выход, иначе из последнего города незаконченного маршрута добавляется переход в город, переход в который имеет минимальные затраты и возврат к шагу 1.

Программная реализация - VHCOUNT. V\_2

**Метод 3: "Венгерский метод".**

Данный метод основан на нижней оценке, получаемой венгерским методом. При установке расчета верхней оценки данным методом расчет нижней оценки автоматически устанавливается на венгерский метод, так как необходимым условием является то, что в каждой строке и каждом столбце только одна пометка. Алгоритм работы метода следующий:

1. Получаем решение релаксированной задачи венгерским методом.
2. Проверяем сколько городов пройдено.
3. Если пройдены все города, то значит получено решение нерелаксированной задачи то переход к пункту 5, иначе переход к пункту 4.
4. В строке города, из которого был возврат в первый, находим минимальное значение среди столбцов еще не пройденных городов и первого города.
5. Если в новом маршруте пройдено N городов, то из последнего города назначаем переход в первый.
6. Если маршрут замкнут, то выход из алгоритма, иначе переход к пункту 2.

Метод схож с методом "С оптимизацией", но отличается тем, что отсутствуют дополнительные проверки, так как исходное решение уже подчиняется вышеуказанным условиям. Программная реализация - SOLUTION. LEVELS

# Описание программного интерфейса.

Интерфейс программы построен по структуре, аналогичной Turbo-средам, но не содержит объекто-ориентированного внутреннего содержания. Главное меню имеет следующую структуру:

Данные

Редактирование

Генерация

Работа с диском

Сохранить матрицу

Открыть матрицу

Решение

Начать решение

Последний результат

Режим решения

Минимум

Максимум

Главное меню

Новая задача

Размерность задачи

Автоматический

Обучающий

Расчет оценок

По возрастанию

С оптимизацией

Венгерский метод

Из каждого города

В каждый город

Комбинированный

Венгерский метод

Опции

Часы

Звук

Ввод

Screen Saver

Сохранить опции

Выход

Рассмотрим меню по пунктам:

*Данные. Новая задача.*

Этот пункт меню вызывает сначала процедуру ввода размерности задачи, затем процедуру ее редактирования. При вызове производится обнуление всех текущих значений матрицы.

*Данные. Размерность задачи.*

Данный пункт меню активизирует процедуру изменения размерности задачи. При этом если матрица уменьшается, то значения в отсеченной части не зануляются и при последующем увеличении размерности могут быть снова использованы. В случае увеличения размера на большее значение крайние элементы оказываются нулевыми.

*Данные. Редактирование.*

Пункт активизирует процедуру по вводу начальной матрицы условий. В процедуре реализован вертикальный и горизонтальный скроллинг матрицы, а также скроллинг внутри вводимой строки. Кроме того возможна настройка ширины строки ввода, которая описана в пункте меню "Опции. Ввод".

*Данные. Генерация.*

Пункт активизирует процедуру, генерирующую матрицу случайным образом в заданном диапазоне значений, который задается перед генерацией.

*Данные. Работа с диском. Сохранить матрицу.*

Данный пункт позволяет сохранить текущую исходную матрицу в файл на диск. Может быть активизирован в любой момент работы меню клавишей F2.

*Данные. Работа с диском. Открыть матрицу.*

Данный пункт позволяет считать с диска исходную матрицу. Может быть активизирован в любой момент работы меню клавишей F3.

*Решение. Начать решение.*

Данный пункт запускает работу алгоритма решения, используя текущие настройки.

*Решение. Последний результат.*

Данный пункт позволяет вывести последний полученный результат решения.

*Режим решения. Блок: Минимум - Максимум.*

Этот блок позволяет выбрать направление решения задачи на минимум или на максимум. Значение по умолчанию - Минимум.

*Режим решения. Блок: Автоматический - Обучающий.*

Этот блок позволяет выбрать между автоматическим и обучающим режимами решения задачи. Значение по умолчанию - Автоматический.

*Расчет оценок. Блок "Верхняя оценка".*

Этот блок позволяет выбрать метод расчета верхней оценки. При выборе третьего метода расчета ("Венгерский метод"), при решении автоматически устанавливается четвертый метод расчета нижней оценки ("Венгерский метод"). Значение по умолчанию - Венгерский метод.

*Расчет оценок. Блок "Нижняя оценка".*

Этот блок позволяет выбрать метод расчета нижней оценки. Какого-либо влияния на метод расчета верхней оценки выбор не оказывает. Значение по умолчанию - Венгерский метод.

*Опции. Часы.*

Данный пункт позволяет включить и выключить часы.

*Опции. Звук.*

Данный пункт позволяет включить и выключить звук.

*Опции. Ввод.*

Данный пункт позволяет задать ширину строки ввода при редактировании начальной матрицы задачи. Значение по умолчанию - 6.

*Опции. Screen Saver.*

Данный пункт позволяет задать время задержки срабатывания Screen Saver-а. Время указывается в минутах. Значение по умолчанию - 5 минут.

*Опции. Сохранить опции.*

Данный пункт позволяет запомнить состояние часов и звука в файле (Shadow.dsk).

*Выход.*

Данный пункт осуществляет выход из программы.

# Описание программы

Структурно программа состоит из девяти модулей:

1. DESCRIPT.PAS - описание всех глобальных переменных программы. Исполняемых процедур не содержит.
2. IOMENU.PAS - модуль для обработки меню. Авторы - Илья Осипов, Андрей Шапошников.
3. IOCRT.PAS - модуль экранных функций вывода. Автор - Илья Осипов.
4. INOUT.PAS - модуль ввода-вывода. Автор - Андрей Шапошников.
5. SERVICE.PAS - модуль системных процедур программы. Автор - Андрей Шапошников.
6. VHCOUNT.PAS - модуль расчета оценок. Автор - Игорь Яров.
7. SOLUTION.PAS - модуль общего управления решением. Авторы - Игорь Яров, Андрей Шапошников.
8. VENGRSOL.PAS - модуль расчета оценок венгерским методом. Автор - Андрей Шапошников.
9. SHADOW.PAS - модуль общего управления программой. Автор - Андрей Шапошников.

Процедуры, которые используются при решении задачи описаны ранее. Можно лишь добавить, что общее управление алгоритмом ветвей и границ осуществляется процедурой OverDrive в автоматическом режиме решения и процедурой StudyMode в обучающем режиме решения.

Процедуры интерфейса программы и глобальные переменные описаны ниже.

***Модуль INOUT.PAS***

**Procedure MatrIn(var a : workmatr ; var sz : byte ; msize, diag : boolean);**

Процедура ввода начальной матрицы условий. Передаваемые параметры:

var a : workmatr - указатель на матрицу (описана глобальной переменной).

var sz : byte - текущая размерность задачи (описана глобальной переменной).

msize : boolean - возможность изменения размеров матрицы (True - могут быть изменены).

 diag : boolean - доступность главной диагонали (False - ввод на главной диагонали запрещен).

**Procedure Inp (x, y, l : byte ; gg : char ; var qq : char ; var a : real ; var s : string ; st\_r : boolean; scroll : boolean ; attrib : byte);**

Процедура ввода строки с внутренним скроллингом. Передаваемые параметры:

x, y : byte - координаты начала строки ввода.

l : byte - ширина строки ввода.

gg : char - последний введенный символ до начала редактирования.

var qq : char - последний введенный символ при редактировании.

var a : real - возвращаемое real-число.

var s : string - возвращаемая строка.

st\_r : boolean - переключатель ввода real / string (True - ввод real-числа).

scroll : boolean - выключатель скроллинга внутри строки (True - скроллинг включен).

attrib : byte - текущие цветовые настройки.

**Procedure M\_Size (var qq : char);**

Процедура изменения размера матрицы. Передаваемые параметры:

var qq : char - последний введенный символ при редактировании.

**Procedure New\_Task (var b : workmatr);**

Процедура генерации новой задачи. Передаваемые параметры:

var b : workmatr - указатель на матрицу (описана глобальной переменной).

**Procedure Matr\_Rnd (var a : workmatr);**

Процедура случайной генерации матрицы. Передаваемые параметры:

var a : workmatr - указатель на матрицу (описана глобальной переменной).

**Procedure ShowSolve (Solve : vertex);**

Процедура вывода последнего полученного решения. Передаваемые параметры:

Solve : vertex - запись, содержащая все параметры последнего решения.

**Function ChooseFile (Title : string ; var qq : char) : string;**

Функция ввода имени файла. Передаваемые параметры:

Title : string - заголовок окна.

var qq : char - последний введенный символ при редактировании.

ChooseFile : string - строка, содержащая имя файла.

**Procedure FileOpen (var b : workmatr ; var NN : byte);**

Процедура чтения с диска матрицы задачи. Передаваемые параметры:

var b : workmatr - указатель на матрицу (описана глобальной переменной).

var NN : byte - текущая размерность задачи (описана глобальной переменной).

**Procedure FileSave (var b : workmatr ; var NN : byte);**

Процедура записи на диск матрицы задачи. Передаваемые параметры:

var b : workmatr - указатель на матрицу (описана глобальной переменной).

var NN : byte - текущая размерность задачи (описана глобальной переменной).

**Procedure InpWidht;**

Процедура задания ширины строки ввода при редактировании начальной матрицы задачи.

**Procedure InpSaver;**

Процедура задания времени задержки срабатывания Screen saver-а.

**Function ChooseVertex (var N : Point ; var Count : integer ; var Act : char) : Point;**

Процедура выбора вершины для обработки при обучающем режиме решения. Передаваемые параметры:

var N : Point - указатель на начало списка вершин.

var Count : integer - общее количество конечных вершин.

var Act : char - код клавиши, определяющий действия, производимые над вершиной.

ChooseVertex : Point - указатель на вершину, над которой совершаются действия.

***Модуль SERVICE.PAS***

**Function GetKey : char;**

Функция, полностью эквивалентная функции Readkey (имеет некоторые отличия по обработке прерываний клавиатуры).

**Procedure Knock;**

Процедура, производящая щелчок.

**Procedure Clock\_on;**

Процедура включения внутреннего таймера программы.

**Procedure Clock\_off;**

Процедура выключения внутреннего таймера программы.

**Procedure SaveIt;**

Процедура сохранения текущих установок программы в файле shadow.dsk.

**Procedure RestoreIt;**

Процедура восстановления установок программы. При отсутствии файла shadow.dsk делает установки по умолчанию.

**Function Stop : boolean;**

Процедура обработки клавиши Escape в фоновом режиме.

**Procedure GetDaTi (var Time : Stime);**

Процедура взятия времени и даты во внутренний формат программы. Передаваемые параметры:

var Time : Stime - текущие время и дата во внутреннем формате.

**Procedure TIME (var Time1, Time2 : Stime);**

Процедура расчета времени работы алгоритма. Передаваемые параметры:

var Time1 : Stime - время начала работы алгоритма.

var Time2 : Stime - время работы алгоритма.

**Function ProcExit : word;**

Процедура подтверждения выхода из программы.

***Модуль DESCRIPT.PAS***

*Переменные, управляющие работой интерфейса и настройкой решения.*

|  |  |
| --- | --- |
| M : Pmenu ; | Указатель на основное меню программы |
| Sel : Word ; | Текущий выбранный пункт меню |
| ch, sk, gg, qq : char ; | Переменные для работы с клавиатурой |
| MethodH, MethodV, Tip, Direc : word ; | Переменные определяющие режим решения |
| TimeSScr : Longint ; | Время задержки срабатывания Screen Saver-а |
| w : boolean ; | Временная булевская переменная |
| SScrAct, | Активность Screen Saver-а |
| ClockAct, | Активность часов |
| SoundAct, | Активность звука |
| SoluAct : boolean ; | Активность решения |
| TimeN, TimeE : Stime ; | Время начала и завершения решения |
| TempStr : string ; | Временная string-переменная |
| TempReal : real ; | Временная real-переменная |
| Len, | Длина элемента матрицы |
| Step : byte ; | Интервал вывода элементов матрицы |

*Типы, используемые при работе алгоритма решения.*

|  |  |
| --- | --- |
| WorkMatr = array [ 1 .. Nmax+1, 1..Nmax+1 ] of real ; | Тип рабочей матрицы |
| Solu = array [ 1..Nmax ] of byte ; | Вектор решения |
| Labels = recordgor, ver : Solu ;end ; | Запись, содержащая вектора фиксированных городов |
| Lab = array [ 1..Nmax ] of boolean ; | Массив меток |
| Point = ^Vertex ; | Указатель на вершину |
| Vertex = recordHi, Lo : real ;Go : Solu ;Res : Solu ;Attr : Char ;Prev, Next : Point ;end ; | Запись, содержащая все свойства единичной вершины |

*Переменные, используемые при работе алгоритма решения.*

|  |  |
| --- | --- |
| b,c : workmatr ; | Исходная матрица задачи иматрица, используемая алгоритмом венгерского метода |
| x : Solu ; | Вектор решения |
| i, j, | Индексные переменные |
| NN : byte ; | Текущая размерность задачи |
| MaxR, MinR : real ; | Переменные, определяющие диапазон генерации матрицы |
| LastSolve : Vertex ; | Запись, содержащая параметры последнего решения |

# Литература

1. Мамиконов А.Г. "Основы построения АСУ", Москва, "Высшая школа" - 1981.
2. Схрейвер А. "Теория линейного и целочисленного программирования", Москва, "Мир" - 1981.
3. Таха Х. "Введение в исследование операций", Москва, "Мир" - 1985.
4. Волчков Б.А., Лифшиц И.И. "Автоматизированные системы в планировании", Москва, "Экономика" - 1980.
5. Касаткин А.И. "Управление ресурсами", Минск, "Высшая школа" -1992.
6. Журнал "PC Magazine" (№3 - 1994), стр. 45 - 48.