**Оглавление**

[Введение 6](#_Toc485381355)

[Глава 1. Аналитический обзор 7](#_Toc485381356)

[1.1. Метод ветвей и границ 9](#_Toc485381357)

[1.2. Комбинаторная оптимизация 12](#_Toc485381358)

[1.3. NP-полная задача  13](#_Toc485381359)

[1.4. Нормирование 22](#_Toc485381360)

[1.4.1. Нормировка показателей 22](#_Toc485381361)

[1.4.2. Типы показателей 23](#_Toc485381362)

[1.4.3.  Нормировка униполярного показателя 23](#_Toc485381363)

[1.4.4. Нормировка биполярного показателя 24](#_Toc485381364)

[1.4.5. Особенности балльных шкал 25](#_Toc485381365)

[1.5. Аналитические пакеты прикладных программ 27](#_Toc485381366)

[Глава 2. Выбор и обоснование алгоритма 31](#_Toc485381367)

[2.1. Содержательная постановка 31](#_Toc485381368)

[2.2. Формальная постановка задачи 33](#_Toc485381369)

[2.3. Метод эталонов 34](#_Toc485381370)

[2.4. Алгоритм 35](#_Toc485381371)

[2.5. Пример решения 36](#_Toc485381372)

[Глава 3. Выбор и обоснование программной реализации 44](#_Toc485381373)

[3.1. Язык программирования C# 44](#_Toc485381374)

[3.2. Интерфейс программы 45](#_Toc485381375)

[Глава 4. Экспериментальная часть 53](#_Toc485381376)

[Заключение 56](#_Toc485381377)

[Список использованной литературы 57](#_Toc485381378)

[Приложение. Листинг программы 58](#_Toc485381379)

# Оглавление

# Введение

# Глава 1. Аналитический обзор

# 1.1. Метод ветвей и границ

# 1.2. Комбинаторная оптимизация

# 1.3. NP-полная задача

# 1.4. Задача коммивояжёра

# 1.4.1. Пример задачи коммивояжёра

# 1.4.2. Асимметричная и симметричная задачи

# 1.4.3. Метрическая задача

# 1.4.4. Формулировка в виде задачи дискретной оптимизации

# 1.4.4.1. Алгоритмическая сложность

# 1.4.4.2 Замкнутые и незамкнутые варианты задачи

# 1.5. Задача о ранце

# 1.5.1. Варианты задачи о ранце

# Глава 2. Выбор и обоснование алгоритма

# 2.1. Метод типа ветвей и границ, осуществляющий фронтальный

# спуск по дереву ветвлений

# 2.2. Динамическое программирование.

# 2.3. Композитные алгоритмы, реализующие неявный перебор на частичных планах, и гарантирующие глобально оптимальное решение

# 2.3.1. Композитная реализация метода типа ветвей и границ

# 2.3.2. Композитная реализация динамического программирования

# Глава 3. Выбор и обоснование программной реализации

# 3.1. Язык программирования Delphi 7

## 3.2. Интерфейс программы

# Введение

Рост числа различных приложений систем искусственного интеллекта повышает требования к используемым в рамках технологии оптимизационным алгоритмам. Это особенно важно применительно к задачам, для которых отсутствуют эффективные процедуры, гарантирующие глобально оптимальное решение. К классическим алгоритмам поиска глобально оптимальных решений можно отнести методы: полный перебор, методы типа ветвей и границ и динамическое программирование. Ниже анализируется алгоритмы, ориентированных на решение задач с булевыми переменными. Под композитными алгоритмами ниже понимаются процедуры, в которых присутствуют компоненты не менее двух классических алгоритмов. Так, ниже процедура выбора направления спуска по дереву ветвлений в методах типа ветвей и границ дополняется технологией отсечения «плохих» планов, развитой в динамическом программировании, а процедуры отсечения, используемые в динамическом программировании, расширяются за счет применения оценок, присущих методам типа ветвей и границ.

Существуют разные способы сравнительной оценки алгоритмов, используемых для принятия решений с помощью математических моделей . Применительно к процедурам, гарантирующим глобально оптимальные решения, мерой может служить время поиска решения: если алгоритм А находит решение i-й задачи за время t(A, i), а алгоритм В находит решение той же задачи за время t(B, i), причем справедливо:



то интеллект алгоритма В выше, чем А.

Далее исследуется классические и композитные процедуры поиска глобально оптимальны решений экстремальных задач с булевыми переменными. Тяжела ты Шапка Мономаха, но короны Франкских королей ещё тяжелей. И хотя это не первое мое знакомство с литературой об этой любопытнейшей эпохе, все время было ощущение, что я читаю телефонную книгу и географический атлас. Имена, города и годы... ( спасибо рупору социалистического реализма Константину Федину за вовремя вспомненную фразу). Сначала была очаровательная дилогия Фетжена, чуть приподнявшая вуаль над лицами двух ярчайших королев- воительниц, потом восхитительный Огюстен Тьерри, донёсший до современного читателя труды Григория Турского, и вот титанический заключающий аккорд Брюно Дюмезиля. Это смутное далекое время обросло толстым слоем слухов, сплетен, необъективности, откровенной лжи и художественных отступлений. Просто захватывает дух как автор один за другим срывает эти слои и словно с помощью пинцета собирает по крупицам истинные события, обнажая действующие лица, их поступки и двигающие ими мотивы. На фоне всей истории выдающиеся женщины правительницы не очень часто появлялись на авансцене мировой политики, считающейся чисто мужским занятием, но имена их сияют ярчайшим светом, порою затмевая своих "коллег" мужчин.

# Глава 1. Аналитический обзор

# 1.1 Метод ветвей и границ

**Метод ветвей и границ** (branch and bound) - общий алгоритмический метод для нахождения оптимальных решений различных задач оптимизации, особенно дискретной и комбинаторной оптимизации. По существу, метод является вариацией полного перебора с отсевом подмножеств допустимых решений, заведомо не содержащих оптимальных решений.

Метод ветвей и границ впервые предложили в 1960 году Ленд и Дойгдля решения задач целочисленного программирования.

Общая идея метода может быть описана на примере поиска минимума функции {\displaystyle f(x)}f(x)  на множестве допустимых значений переменной {\displaystyle x}x. Функция {\displaystyle f}f  и переменная {\displaystyle x}x могут быть произвольной природы. Для метода ветвей и границ необходимы две процедуры: ветвление и нахождение оценок (границ).

Процедура ветвления состоит в разбиении множества допустимых значений переменной {\displaystyle x}x на подобласти (подмножества) меньших размеров. Процедуру можно рекурсивно применять к подобластям. Полученные подобласти образуют дерево, называемое деревом поиска или деревом ветвей и границ. Узлами этого дерева являются построенные подобласти (подмножества множества значений переменной {\displaystyle x}x).

Процедура нахождения оценок заключается в поиске верхних и нижних границ для решения задачи на подобласти допустимых значений переменной{\displaystyle x}.

В основе метода ветвей и границ лежит следующая идея: если нижняя граница значений функции на подобласти {\displaystyle A}A дерева поиска больше, чем верхняя граница на какой-либо ранее просмотренной подобласти {\displaystyle B}B, то {\displaystyle A}A может быть исключена из дальнейшего рассмотрения (правило отсева). Обычно минимальную из полученных верхних оценок записывают в глобальную переменную {\displaystyle m}m; любой узел дерева поиска, нижняя граница которого больше значения {\displaystyle m}m, может быть исключён из дальнейшего рассмотрения.

Если нижняя граница для узла дерева совпадает с верхней границей, то это значение является минимумом функции и достигается на соответствующей подобласти.

Метод используется для решения некоторыхNP-полных задач, в том числе задачи коммивояжёра и задачи о ранце.

**1.2 Комбинаторная оптимизация**

Комбинаторная оптимизация  — область теории оптимизации в прикладной математике. связанная с исследованием операции, теорией алгоритмов и теорией вычислительной сложности. Комбинаторная оптимизация заключается в поиске оптимального объекта в конечном множестве объектов, чем очень похожа на [дискретное программирование](https://ru.wikipedia.org/wiki/%D0%94%D0%B8%D1%81%D0%BA%D1%80%D0%B5%D1%82%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5). Некоторые источники  под дискретным программированием понимают целочисленное программирование, противопоставляя ему комбинаторную оптимизацию, имеющую дело с [графами](https://ru.wikipedia.org/wiki/%D0%93%D1%80%D0%B0%D1%84_(%D0%BC%D0%B0%D1%82%D0%B5%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)), [матроидами](https://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D1%82%D1%80%D0%BE%D0%B8%D0%B4) и похожими структурами. Однако оба термина очень близко связаны и в литературе часто переплетаются. Комбинаторная оптимизация часто сводится к определению эффективного распределения ресурсов, используемых для поиска оптимального решения.

Во многих задачах комбинаторной оптимизации [полный перебор](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D0%BB%D0%BD%D1%8B%D0%B9_%D0%BF%D0%B5%D1%80%D0%B5%D0%B1%D0%BE%D1%80) нереален. Комбинаторная оптимизация включает в себя задачи оптимизации, в которых множество допустимых решений дискретно или может быть сведено к дискретному множеству.

Комбинаторная оптимизация используется при:

Определении оптимальной сети аэрофлота;

Определении, какая машина из парка такси подберёт пассажиров;

Определении оптимального пути доставки посылок;

Определении правильных атрибутов перед [тестированием концепций](https://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D1%80%D0%BA%D0%B5%D1%82%D0%B8%D0%BD%D0%B3%D0%BE%D0%B2%D0%BE%D0%B5_%D0%B8%D1%81%D1%81%D0%BB%D0%B5%D0%B4%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5#%D1%82%D0%B5%D1%81%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5_%D0%BA%D0%BE%D0%BD%D1%86%D0%B5%D0%BF%D1%86%D0%B8%D0%B9).

Однако этими примерами приложение комбинаторной оптимизации не ограничивается.

Они встретились случайно.  
Великолепная Диана — дочь сокольничего и Марк — преуспевающий владелец ночных клубов в Москве.  
Диана - гордая и прекрасная! Она понимает язык животных и птиц — язык гармоничного мира, где древнее знание возрождает инстинкты и интуицию, дарит подлинное единение с природой.  
Марк, с друзьями решили испробовать что-то новое, дикое, первозданное. Они отправляются туда, где ходят легенды о соколиной охоте.  
Диана и Марк. Казалось бы, совершенно разные люди… Но любовь, вспыхнувшая как пламя, соединила их сердца.  
Но ведь между ними нет ничего общего! Или есть?  
Что же выберет мужчина, нечаянно и безгранично полюбивший прекрасную сокОлицу?

Имеется большое число литературы по [полиномиальным](https://ru.wikipedia.org/wiki/%D0%9A%D0%BB%D0%B0%D1%81%D1%81_P) по времени алгоритмам, работающим на некоторых классах задач дискретного программирования и существенная часть этих алгоритмов принадлежит теории [линейного программирования](https://ru.wikipedia.org/wiki/%D0%9B%D0%B8%D0%BD%D0%B5%D0%B9%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5). Некоторые примеры комбинаторной оптимизации, попадающие в эту область — это задача поиска [кратчайшего пути](https://ru.wikipedia.org/w/index.php?title=%D0%9F%D1%80%D0%BE%D0%B1%D0%BB%D0%B5%D0%BC%D0%B0_%D0%BA%D1%80%D0%B0%D1%82%D1%87%D0%B0%D0%B9%D1%88%D0%B5%D0%B3%D0%BE_%D0%BF%D1%83%D1%82%D0%B8&action=edit&redlink=1) и [дерева кратчайших путей](https://ru.wikipedia.org/w/index.php?title=%D0%94%D0%B5%D1%80%D0%B5%D0%B2%D0%BE_%D0%BA%D1%80%D0%B0%D1%82%D1%87%D0%B0%D0%B9%D1%88%D0%B8%D1%85_%D0%BF%D1%83%D1%82%D0%B5%D0%B9&action=edit&redlink=1), определение [максимального потока](https://ru.wikipedia.org/wiki/%D0%97%D0%B0%D0%B4%D0%B0%D1%87%D0%B0_%D0%BE_%D0%BC%D0%B0%D0%BA%D1%81%D0%B8%D0%BC%D0%B0%D0%BB%D1%8C%D0%BD%D0%BE%D0%BC_%D0%BF%D0%BE%D1%82%D0%BE%D0%BA%D0%B5), нахождение остовных деревьев, нахождение паросочитаний, задачи сматроидами.

Задачи комбинаторной оптимизации можно рассматривать как поиск лучшего элемента в некотором дискретном множестве, поэтому, в принципе, могут быть использованы любые [алгоритмы поиска](https://ru.wikipedia.org/w/index.php?title=%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC%D1%8B_%D0%BF%D0%BE%D0%B8%D1%81%D0%BA%D0%B0&action=edit&redlink=1) илиметаэвристические алгоритмы. Однако общие алгоритмы поиска не гарантируют ни оптимального решения, ни быстрого решения (за полиномиальное время). Поскольку некоторые задачи дискретной оптимизации [NP-полны](https://ru.wikipedia.org/wiki/NP-%D0%BF%D0%BE%D0%BB%D0%BD%D0%B0%D1%8F_%D0%B7%D0%B0%D0%B4%D0%B0%D1%87%D0%B0), как, например, задача о коммивояжёре, это же следует ожидать и для других задач (если не [P=NP](https://ru.wikipedia.org/wiki/%D0%A0%D0%B0%D0%B2%D0%B5%D0%BD%D1%81%D1%82%D0%B2%D0%BE_%D0%BA%D0%BB%D0%B0%D1%81%D1%81%D0%BE%D0%B2_P_%D0%B8_NP)).

**1.3 NP-полная задача**

**NP-полная задача** — в теории алгоритмов задача с ответом «да» или «нет» из класса NP, к которой можно свести любую другую задачу из этого класса за полиномиальное время (то есть при помощи операций, число которых не превышает некоторого полинома в зависимости от размера исходных данных). Таким образом, NP-полные задачи образуют в некотором смысле подмножество «типовых» задач в классе NP: если для какой-то из них найден «полиномиально быстрый» алгоритм решения, то и любая другая задача из класса NP может быть решена так же «быстро».

**Формальное определение**

Алфавитом называется всякое конечное множество символов (например, {{\displaystyle {0,1}}0,1} или {{\displaystyle {a,b,c}}a,b,c}). Множество всех возможных слов  (конечных строк, составленных из символов этого алфавита) над некоторым алфавитом {\displaystyle \Sigma } обозначается {\displaystyle \Sigma ^{\*}}. **Языком** *{\displaystyle L}L* над алфавитом {\displaystyle \Sigma } называется всякое подмножество множества {\displaystyle \Sigma ^{\*}}, то естьL {\displaystyle L\subset \Sigma ^{\*}}⊂.

**Задачей распознавания** для языка {\displaystyle L}L называется определение того, принадлежит ли данное слово языку {\displaystyle L}L.

Пусть {\displaystyle L\_{1}} и {\displaystyle L\_{2}} — два языка над алфавитом {\displaystyle \Sigma }. Язык {\displaystyle L\_{1}} называется сводимым (по Карпу) к языку {\displaystyle L\_{2}}, если существует функция, {\displaystyle f:\Sigma ^{\*}\to \Sigma ^{\*}}f : , вычислимая за полиномиальное время, обладающая следующим свойством:

{\displaystyle x\in L\_{1}}x∈ тогда и только тогда, когда {\displaystyle f(x)\in L\_{2}}f(x) ∈. Сводимость по Карпу обозначается как {\displaystyle L\_{1}{\leq }\_{p}L\_{2}}{\displaystyle L\_{1}}  {\displaystyle L\_{2}}  или {\displaystyle L\_{1}\varpropto L\_{2}}{\displaystyle L\_{1}} ∞ {\displaystyle L\_{2}}  .

Язык {\displaystyle L\_{2}} называется **NP-трудным**, если любой язык из класса NP сводится к нему. Язык называют **NP-полным**, если он NP-труден, и при этом сам лежит в классе NP.

Неформально говоря, то что задача {\displaystyle A}A сводится к задаче {\displaystyle B}B, означает, что задача {\displaystyle A}A «не сложнее» задачи {\displaystyle B}B (так как, если мы можем решить {\displaystyle B}B, то можем решить и {\displaystyle A}A). Таким образом, класс NP-трудных задач включает NP-полные задачи и задачи, которые «сложнее» их (то есть те задачи, к которым могут быть сведены NP-полные задачи). Класс NP включает NP-полные задачи и задачи, которые «легче» их (то есть те задачи, которые сводятся к NP-полным задачам).

Из определения следует, что, если будет найден алгоритм, решающий некоторую (любую) NP-полную задачу за полиномиальное время, то все NP-задачи окажутся в классе P, то есть будут решаться за полиномиальное время.

### NP-полнота в сильном смысле

Задача называется **NP-полной в сильном смысле**, если у неё существует подзадача, которая:

1. не является задачей с числовыми параметрами (то есть максимальное значение величин, встречающихся в этой задаче, ограничено сверху полиномом от длины входа),
2. принадлежит классу NP,
3. является NP-полной.

Класс таких задач называется **NPCS**. Если гипотеза P ≠ NP верна, то для NPCS-задачи не существует псевдополиномиального алгоритма

# 1.4Задача коммивояжёра

**Задача коммивояжёра** - одна из самых известных задач комбинаторной оптимизации, заключающаяся в поиске самого выгодного маршрута, проходящего через указанные города хотя бы по одному разу с последующим возвратом в исходный город. В условиях задачи указываются критерий выгодности маршрута (кратчайший, самый дешёвый, совокупный критерий и тому подобное) и соответствующие матрицы расстояний, стоимости и тому подобного. Как правило, указывается, что маршрут должен проходить через каждый город только один раз — в таком случае выбор осуществляется среди гамильтоновых циклов. Существует несколько частных случаев общей постановки задачи, в частности, геометрическая задача коммивояжёра (также называемая планарной или евклидовой, когда матрица расстояний отражает расстояния между точками на плоскости), метрическая задача коммивояжёра (когда на матрице стоимостей выполняется неравенство треугольника), симметричная и асимметричная задачи коммивояжёра. Также существует обобщение задачи, так называемая обобщённая задача коммивояжёра.

Оптимизационная постановка задачи относится к классу NP-трудных задач, впрочем, как и большинство её частных случаев. Версия «decisionproblem» (то есть такая, в которой ставится вопрос, существует ли маршрут не длиннее, чем заданное значение *k*) относится к классу NP-полных задач. Задача коммивояжёра относится к числу трансвычислительных: уже при относительно небольшом числе городов (66 и более) она не может быть решена методом перебора вариантов никакими теоретически мыслимыми компьютерами за время, меньшее нескольких миллиардов лет.

Автор романа Джеймс Э. Л. начинала книгу, как серию блогов. Впоследствии она переросла в полноценный роман, который и завоевал сердца многих читателей. Кстати большое количество желающих «50 оттенков серого» читать заставило обратить внимание на книгу и Голливуд. Благодаря этому уже в 2015 году на День влюбленных состоялась премьера романа.

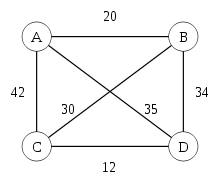
Популярности «50 оттенков серого» книге прибавила немалая скандальность произведения. Так некоторые библиотеки отказались добавлять ее в свои хранилища, указывая на ее провокационность и не соответствия жанра критериям отбора. В дальнейшем книга все-таки была добавлена в список книг практически всех библиотек мира. Кроме того многие критики указывали на недопустимую жестокость между отношениями, описанными в книге. В то же время многие указывали на достаточную реалистичность всего происходящего в романе, что посеяло массу противоречий о допустимости и этичности данного произведения.

Действие «50 оттенков серого» книги происходит в США. В книге описана история взаимоотношений, в том числе интимного характера, между преуспевающим бизнесменом Кристианом Греем и молодой журналисткой Анастейшей. Борьба желающего доминировать Кристиана с чистой и невинной Анастейшей, большое количество ролевых любовных игр, борьба любви и похоти все это есть в «50 оттенков серого» книге.

«50 оттенков серого» книга получилась настолько популярной, что благодаря ней появилось целое направление эротического романа. Так многие сравнивают серию книг «Я люблю тебя» с произведениями Эрики Джеймс. Хотя они не настолько откровенны и скандальны, а сама автор отрицает таковую связь. Кроме того на «50 оттенков серого» похожие книги вы можете найти в творчестве Анны Тодд и Ринки Кейт. А недавно вышедший русский ответ «50 оттенков серого» — книга «Два месяца и три дня» вообще пробудил немалый интерес.

# 1.4.1 Задача коммивояжёра

### Представление в виде графа

[](https://commons.wikimedia.org/wiki/File:Weighted_K4.svg?uselang=ru)

Симметричная задача для четырёх городов.

Для возможности применения математического аппарата для решения проблемы её следует представить в виде математической модели. Проблему коммивояжёра можно представить в виде модели на графе, то есть, используя вершины и ребра между ними. Таким образом, вершины графа соответствуют городам, а рёбра {\displaystyle \left(i,j\right)}(I,j) между вершинами {\displaystyle i}i и {\displaystyle j}j — пути сообщения между этими городами. Каждому ребру {\displaystyle \left(i,j\right)}(I,j) можно сопоставить критерий выгодности маршрута {\displaystyle c\_{ij}\geqslant 0} ≥ 0, который можно понимать как, например, расстояние между городами, время или стоимость поездки.

Гамильтоновым циклом называется маршрут, включающий ровно по одному разу каждую вершину графа.

В целях упрощения задачи и гарантии существования маршрута обычно считается, что модельный граф задачи является полностью связным, то есть, что между произвольной парой вершин существует ребро. В тех случаях, когда между отдельными городами не существует сообщения, этого можно достичь путём ввода рёбер с максимальной длиной. Из-за большой длины такое ребро никогда не попадет к оптимальному маршруту, если он существует.

Таким образом, решение задачи коммивояжёра — это нахождение гамильтонова цикла минимального веса в полном взвешенном графе.

В зависимости от того, какой критерий выгодности маршрута сопоставляется величине ребер, различают различные варианты задачи, важнейшими из которых являются симметричная и метрическая задачи.

#### 1.4.2 Асимметричная и симметричная задачи

В общем случае, асимметричная задача коммивояжёра отличается тем, что она моделируется ориентированным графом. Таким образом, следует также учитывать, в каком направлении находятся ребра.

В случае симметричной задачи все пары ребер между одними и теми же вершинами имеют одинаковую длину, то есть, для ребра {\displaystyle \left(i,j\right)}(I,j) одинаковы длины {\displaystyle c\_{ij}=c\_{ji}} = . В симметричном случае количество возможных маршрутов вдвое меньше асимметричного случая. Симметричная задача моделируется неориентированным графом.

На самом деле, задача коммивояжёра в случае реальных городов может быть как симметричной, так и асимметричной в зависимости от длительности или длины маршрутов и от направления движения.

#### 1.4.3 Метрическая задача

Симметричную задачу коммивояжёра называют *метрической*, если относительно длин ребер выполняется неравенство треугольника. Условно говоря, в таких задачах обходные пути длиннее прямых, то есть, ребро от вершины {\displaystyle i}i до вершины {\displaystyle j}j никогда не бывает длиннее пути через промежуточную вершину {\displaystyle k}k :

{\displaystyle c\_{ij}\leqslant c\_{ik}+c\_{kj}} ≤ + .

Такое свойство длины ребер определяет измеримое пространство на множестве ребер и меру расстояния, удовлетворяющую интуитивному пониманию расстояния.

Распространенные на практике функции расстояния являются также метриками и удовлетворяют неравенству треугольника:

* *Евклидово расстояние* в евклидовой задаче коммивояжёра,
* *Манхэттенская метрика* (также квартальная метрика) прямоугольной задачи коммивояжёра, в которой расстояние между вершинами на решетке равно сумме расстояний по оси ординат и абсцисс,
* *Максимальная метрика*, определяющая расстояние между вершинами решетчатого графа как максимальное значение расстояния вдоль оси ординат и абсцисс.

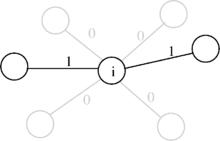
Две последние метрики находят применение, например, при сверлении отверстий в печатных платах, когда станок должен сделать больше отверстий за наименьшее время и может перемещать сверло в обоих направлениях для перехода от одного отверстия к следующему. Манхэттенская метрика соответствует случаю, когда передвижение в обоих направлениях происходит последовательно, а максимальная — случаю, когда передвижение в обоих направлениях происходит синхронно, а общее время равно максимальному времени передвижения вдоль оси ординат или абсцисс.

Неметрическая задача коммивояжёра может возникать, например, в случае минимизации длительности пребывания при наличии выбора транспортных средств в различных направлениях. В таком случае обходной путь самолетом может быть короче прямого сообщения автомобилем.

Если на практике в условиях задачи разрешается посещать города несколько раз, то симметричную задачу можно свести к метрической. Для этого задачу рассматривают на так называемом графе расстояний. Этот граф имеет такое же множество вершин, как и исходный, и является полностью связным. Длина рёбер {\displaystyle c\_{ij}} между вершинами {\displaystyle i}i и {\displaystyle j}j на графе расстояний соответствует длине кратчайшего расстояния между вершинами {\displaystyle i}i и {\displaystyle j}j в исходном графе. Для определенных таким образом длин {\displaystyle c\_{ij}} выполняется неравенство треугольника, и каждому маршруту на графе расстояний всегда соответствует маршрут с возможными повторениями вершин в исходном графе.

### 1.4.4 Формулировка в виде задачи дискретной оптимизации

Одним из подходов к решению задачи является формулировка её в виде задачи дискретной оптимизации, при этом решения представляются в виде переменных, а связи — в виде отношений неравенства между ними. Таким образом, возможно несколько вариантов. Например, симметричную задачу можно представить в виде множества ребер {\displaystyle V}V. Каждому ребру {\displaystyle \{i,j\}}{I,j} сопоставляется двоичная переменная {\displaystyle x\_{ij}\in \{0,1\}}∈ {0,1}, равная 1, если ребро принадлежит маршруту, и 0 — в противном случае. Произвольный маршрут можно представить в виде значений множества переменных принадлежности, но не каждое такое множество определяет маршрут. Условием того, что значения множества переменных определяют маршрут, являются описанные далее линейные неравенства.

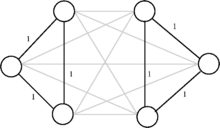
[](https://commons.wikimedia.org/wiki/File:TSP_degree_constraints.png?uselang=ru)

Условие кратности: каждая вершина должна иметь одно входное и одно выходное ребро маршрута.

Каждая вершина должна сообщаться через пару ребер с остальным вершинам, то есть, через входное и выходное ребро:

{\displaystyle \forall i\in V,\;\sum \_{j\in V\setminus \{i\}}x\_{ij}=2\qquad (1)}∈V, (1)

В сумме каждое слагаемое {\displaystyle x\_{ij}} равно или 1 (принадлежит маршруту) или 0 (не принадлежит). То есть, полученная сумма равна количеству ребер в маршруте, имеющих вершину {\displaystyle i}i на одном из концов. Она равна 2, так как каждая вершина имеет входное и выходное ребро. В приведенном рядом рисунке вершина {\displaystyle i}i показана с входным и выходными ребрами, а ребра маршрута обозначены толстыми линиями. Рядом с ребрами указаны длины {\displaystyle x\_{ij}}, прилагаемые к указанной выше сумме.

[](https://commons.wikimedia.org/wiki/File:TSP_short_cycles.png?uselang=ru)

Циклы: переменные удовлетворяют условию кратности, но не определяют маршрут.

Описанные ранее условия кратности выполняются не только маршрутами, но и значениями переменных, соответствующих отдельным циклам, где каждая вершина принадлежит лишь одному циклу (см. рисунок). Чтобы избежать подобных случаев, должны выполняться так называемые неравенства циклов (или условия устранения подмаршрутов), которые были определены Данцигом, Фалкерсоном и Джонсоном в 1954 году под названием *условия петель*. Этими неравенствами определялось дополнительное условие того, что каждое множество вершин {\displaystyle S\subset V}S⊂V является либо пустым, либо содержит все вершины, сочетающееся с остальным вершинам через минимум два ребра:

{\displaystyle \sum \_{i\in S,\;j\notin S}x\_{ij}\geqslant 2\qquad (2)} (2)

для всех множеств вершин {\displaystyle S}S, где {\displaystyle 1\leqslant |S|\leqslant |V|-1}1≤|S|≤|V| -1. Эта сумма равна сумме длин ребер маршрута между вершиной {\displaystyle i\in S}i∈S и вершиной {\displaystyle j\notin S}j∉S. Чтобы устранить лишние неравенства, можно ограничиться множествами вершин {\displaystyle S}S с минимум двумя и максимум {\displaystyle |V|-2}|V|-2 вершинами. На рисунке рядом ребра {\displaystyle \{i,j\}}{I,j} с длинами {\displaystyle x\_{ij}=1} = 1 обозначены толстыми линиями, остальные ребра имеют длину {\displaystyle x\_{ij}=0} = 0. Введение дополнительных условий (2) для множества вершин {\displaystyle S}S, состоящего из трех левых вершин, будет гарантировать, что {\displaystyle S}Sсочетается через минимум два ребра маршрута с тремя вершинами справа, чтобы устранить оба цикла. Количество неравенств устранения циклов согласно Данцигу, Фалкерсону и Джонсону равняется - 2(n-1){\displaystyle 2^{n}-2(n-1)}.

В 1960 году Миллер, Такер и Землин изобрели альтернативные условия устранения подмаршрутов путём введения {\displaystyle n}n новых переменных, определяющих порядок посещенных городов, требующих только {\displaystyle n^{2}-n+1} – n+1{\displaystyle 2^{n}-2(n-1)}. дополнительных неравенств. Более того, из-за двойственности {\displaystyle x\_{ij}} в формулировках Миллера, Такера и Землина задача коммивояжёра остается NP-сложной.

Так, каждый вектор с элементами, равными 0 и 1, удовлетворяющий всем неравенствам, определяет корректный маршрут, который является решением переформулированной задачи коммивояжёра: вычислить

{\displaystyle \min \;\left\{\sum \_{i\in V}\sum \_{j\in V\setminus \{i\}}c\_{ij}x\_{ij}\;|\;x{\mbox{ valid (1) (2),}}\;x\_{ij}\in \{0,1\}\right\}.\qquad (3)}min { } (3)

Поскольку переменные {\displaystyle x\_{ij}} имеют значения только 0 и 1, сумма равна общей длине {\displaystyle c\_{ij}} ребер {\displaystyle \{i,j\}}{I,j}, принадлежащих маршруту.

Количество неравенств типа (2) растет экспоненциально по мере увеличения количества городов, поскольку почти каждое из {\displaystyle 2^{|V|}}подмножеств узлов определяет одно неравенство. Эту проблему можно решить применением метода отсечения плоскостью, благодаря которому неравенства добавляются, только когда эти неравенства действительно необходимы. С геометрической точки зрения, линейные неравенства можно представить как гиперплоскости в пространстве переменных. Множество векторов, удовлетворяющих этим неравенствам, образуют в таком пространстве политоп (многомерный многогранник), или многомерный многоугольник, точная форма определяется длинами {\displaystyle c\_{ij}} и является в основном неизвестной. Однако, можно показать, что условия (1) и (2) определяют грани (фацет) политопа, то есть боковые поверхности политопа с наивысшей размерностью. Поэтому они относятся к самым сильным линейным неравенствам, которые могут описывать маршрут. Также существует много разных граней, определяемых неравенствами, известными лишь в немногих случаях. Хотя (1) и (2) вместе с ограничениями полностью моделируют проблему только для двоичных векторов, эти неравенства могут использоваться в методе ветвей и границ, чтобы отбросить решения методами линейной оптимизации с нецелыми координатами.

**1.4.4.1 Алгоритмическая сложность**

Поскольку коммивояжёр в каждом из городов встает перед выбором следующего города из тех, что он ещё не посетил, существует **{\displaystyle (n-1)!}(n - 1)!** маршрутов для асимметричной и ***{\displaystyle (n-1)!/2}(n -1)!/2*** маршрутов для симметричной задачи коммивояжёра. Таким образом, размер пространства поиска зависит экспоненциально от количества городов.

Различные варианты задачи коммивояжёра (метрическая, симметричная и асимметричная) NP-эквивалентны. Согласно распространенной, но недоказанной гипотезе о неравенстве классов сложности P и NP, не существует детерминированной машины Тьюринга, способной находить решения экземпляров задачи за полиномиальное время в зависимости от количества городов.

Также известно, что при условии {\displaystyle P\neq NP}P≠NP не существует алгоритма, который для некоторого полинома **{\displaystyle p}p** вычислял бы такие решения задачи коммивояжёра, которые отличались бы от оптимального максимум на коэффициент {\displaystyle 2^{p(n)}}.

На практике поиск строго оптимального маршрута требуется не всегда. Существуют алгоритмы поиска приближенных решений для метрической задачи за полиномиальное время и нахождения маршрута максимум вдвое длиннее оптимального. До сих пор не известен ни один алгоритм с полиномиальным временем, который бы гарантировал точность лучшую, чем 1,5 от оптимальной. По предположению {\displaystyle P\neq NP}P≠NP , существует (неизвестная) константа {\displaystyle c>0}c> 0, такая, что ни один алгоритм с полиномиальным временем не может гарантировать точность {\displaystyle 1+c}1 + c. Как было показано Арора, для евклидовой задачи коммивояжёра существует схема полиномиального времени PTAS для поиска приближённого решения.

Кроме того данные могут иметь особенности, которые могут помочь решить задачу. Например, города расположены не случайно, а по рельефу местности, либо даже вдоль уже давно найденного оптимального торгового маршрута. Дополнительная информация и эвристики позволяют за приемлемое время находить приемлемые решения.

**1.4.4.2 Замкнутые и незамкнутые варианты задачи**

В замкнутом варианте задачи коммивояжёра требуется посетить все вершины графа, после чего вернуться в исходную вершину. Незамкнутый вариант отличается от замкнутого тем, что в нём не требуется возвращаться в стартовую вершину.

Незамкнутый вариант задачи сводится к замкнутому путём замены весов дуг, входящих в стартовую вершину, на число 0. Оптимальный замкнутый маршрут коммивояжёра в таком графе соответствует оптимальному незамкнутому маршруту в исходном графе.

Чтобы свести замкнутый вариант к незамкнутому, нужно определить число {\displaystyle K}K, строго превосходящее вес любого маршрута коммивояжёра в заданном графе (например, в качестве {\displaystyle K}K можно взять сумму максимальных по весу дуг, выходящих из каждой вершины, увеличенную на 1). Затем нужно добавить к графу новую вершину{\displaystyle v\_{n}} (предполагаем, что вершины исходного графа пронумерованы числами от 0 до {\displaystyle n-1}n - 1, при этом стартовая вершина имеет номер 0). Стоимости дуг, выходящих и входящих в вершину {\displaystyle v\_{n}}, определяются следующим образом:

* {\displaystyle c\_{n,i}=3K} = 3K
* {\displaystyle c\_{0,n}=3K} = 3K
* {\displaystyle c\_{i,n}=c\_{i,0}+2K} = 3K при {\displaystyle i}i от {\displaystyle 1}1 до {\displaystyle n-1}n-1

Оптимальный незамкнутый маршрут коммивояжёра в таком графе соответствует оптимальному замкнутому маршруту коммивояжёра в исходном графе и имеет стоимость на {\displaystyle 2K}2K больше.

**1.5 Задача о ранце**

**Задача о ранце** (или **задача о рюкзаке**) — NP-полная задача комбинаторной оптимизации. Своё название получила от конечной цели: уложить как можно большее число ценных вещей в рюкзак при условии, что вместимость рюкзака ограничена. С различными вариациями задачи о ранце можно столкнуться в экономике, прикладной математике, криптографии и логистике.

В общем виде задачу можно сформулировать так: из заданного множества предметов со свойствами «стоимость» и «вес» требуется отобрать подмножество с максимальной полной стоимостью, соблюдая при этом ограничение на суммарный вес.

Классическая постановка задачи

Пусть имеется набор предметов, каждый из которых имеет два параметра - вес и ценность. Также имеется рюкзак определённой вместимости. Задача заключается в том, чтобы собрать рюкзак с максимальной ценностью предметов внутри, соблюдая при этом ограничение рюкзака на суммарный вес.

Математически задача формулируется следующим образом: имеется {\displaystyle n}n грузов. Для каждого {\displaystyle i}i-го груза определён его **вес** {\displaystyle w\_{i}>0} > 0 и **ценность** {\displaystyle v\_{i}>0}>0,i=1,2,…,n {\displaystyle i=1,2,...,n}. Ограничение суммарного веса предметов в рюкзаке задаётся **грузоподъёмностью** {\displaystyle W}W. Необходимо

максимизировать {\displaystyle \sum \_{i=1}^{n}v\_{i}x\_{i}}

с ограничениями {\displaystyle \sum \_{i=1}^{n}w\_{i}x\_{i}\leq W}≤W и {\displaystyle x\_{i}\in \{0,1\}}∈ {0,1}.

### 1.5.1 Варианты задачи о ранце

Постановка задачи допускает большое количество обобщений, в зависимости от условий, наложенных на рюкзак, предметы или их выбор. Наиболее популярными разновидностями являются следующие:

1. Рюкзак 0-1 - не более одного экземпляра каждого предмета.
2. Ограниченный рюкзак - не более заданного числа экземпляров каждого предмета.
3. Неограниченный рюкзак - произвольное количество экземпляров каждого предмета.
4. Рюкзак с мультивыбором - предметы разделены на группы, и из каждой группы требуется выбрать только один предмет.
5. Множественный рюкзак - есть несколько рюкзаков, каждый со своим максимальным весом. Каждый предмет можно положить в любой рюкзак или оставить.
6. Многомерный рюкзак - вместо веса дано несколько разных ресурсов (например, вес, объём и время укладки). Каждый предмет тратит заданное количество каждого ресурса. Надо выбрать подмножество предметов так, чтобы общие затраты каждого ресурса не превышали максимума по этому ресурсу, и при этом общая ценность предметов была максимальна.
7. Квадратичная задача о рюкзаке - суммарная ценность задается неотрицательно определённой квадратичной формой.

# Глава 2. Выбор и обоснование алгоритма

1. **Обозначения, допущения и определения**

Ниже анализируется сравнительный интеллект различных алгоритмов поиска решения однокритериальных задач с булевыми переменными типа:



где:  Сi и *aj*- константы,*bi,j*– коэффициенты, причем предполагаем, что  Сi ≥0. При этом все численные примеры, иллюстрирующие эффективность предлагаемых подходов, ниже приводятся для случая, когда F → max, j = 1, что отвечает задаче о ранце вида:

F = 7z1 +3z2 + 5z3 + 9z4 + 2z5 → max;

3z1 + 4z2 + 2z3 + 7z4 + 6z5 ≤ 10; (3)

zi = 1, 0; *i* = 1, 2, …,5.

Далее, применительно к методам неявного перебора, делаем поиск решения на множестве частичных планов, используются определения:

1. «Висячей» вершиной на любой итерации считается такая вершина построенной части дерева ветвлений, из которой не исходят дуги.
2. Корневая вершина дерева ветвлений считается «висячей» вершиной построенной части дерева на первой итерации.
3. «Кустом» с высотой h и с корнем в вершине  является ориентированный подграф, обладающий следующими свойствами:

* в каждую вершину куста, за исключением корневой, входит только одна дуга;
* из каждой вершины куста, за исключением множества висячих вершин, исходят две дуги;
* из вершины  в каждую висячую вершину куста существует только один путь, число дуг которого равно h;
* число вершин куста равно 2h+1-1, число висячих вершин равно 2h, число дуг куста определяется выражением: 2(2h - 1);
* все рассмотренные ниже примеры в разделах 3 – 6 используют случаи, когда h = 1, только в примере, рассмотренном в седьмом разделе, h = 2.

1. Ветвлением из заданной вершины  которой соответствует подмножество введенных в базис переменных I(xk), является построение «куста» с корнем в этой вершине, причем каждой из 2h висячих вершин этого куста соответствует │I(xk)│+ h введенных в базис переменных.
2. Верхняя оценка Δ(xk) величины F, отвечающей вершине  соответствующей вектору переменных задачи (3), в базис которого введены I1(xk) переменных, далее определяется следующим образом:

Δ(xk)=  (4)

где I – множество индексов всех переменных.

1. Аналогично определяется нижняя оценка δ(I1) величины F:

δ(xk) =  (5)

В то время как нижняя граница, определяемая на основании (5) в силу сделанных выше допущений не зависит от специфики задачи, величина Δ(xk) непосредственно связана с этой спецификой: выражение (4) справедливо только для задачи о ранце. Примеры способов вычисления верхней оценкиΔ(xk) применительно к ряду других задач, приведены в [11]. Далее полагаем, что время счета в первом приближении линейно зависит от числа вершин построенного в ходе поиска решения дерева ветвлений G(X,U), где Х – множество вершин, U – множество дуг.

Общей чертой приводимых ниже графических иллюстраций поиска решения системы (3) является кодировка вершин дерева ветвлений: «серые» вершины дерева отвечают значениям соответствующих переменных, равным единице, «белые» – равным нулю. Вершине последнего яруса с «жирным» контуром соответствует оптимальный вектор переменных Z.

1. **Методы неявного перебора на частичных планах, гарантирующие глобально оптимальное решение**

Ниже рассмотрено два метода такого рода: динамическое программирование и методы типа ветвей и границ.

* 1. **Метод типа ветвей и границ, осуществляющий фронтальный спуск по дереву ветвлений**

Содержательно работа такого алгоритма может быть представлена последовательным построением дерева ветвлений G(X,U):

Алгоритм

1. На множестве висячих вершинпостроенной части дерева ветвлений G(X,U) выбирается вершина xj с наилучшей оценкой. Если это осуществляется на первой итерации, то такой вершиной *apriori* считается корневая вершина дерева.
2. Если выбранной вершине отвечает равенство I1 = I, то перейти к шагу 5, в противном случае – к следующему шагу.
3. Ветвление осуществляется из выбранной на шаге 1 последней итерации вершины xj. Новое множество висячих вершин дерева вновь обозначаем Х1.
4. Вычисляются оценки, отвечающие висячим вершинам построенного на предыдущем шаге «куста». Для этого можно воспользоваться (4), если целевая функция системы (2) является максимизируемой и (5) – если целевая функция минимизируется. Перейти к шагу 1.
5. Алгоритм закончен. Вектор переменных, соответствующий выбранной вершине, является оптимальным.

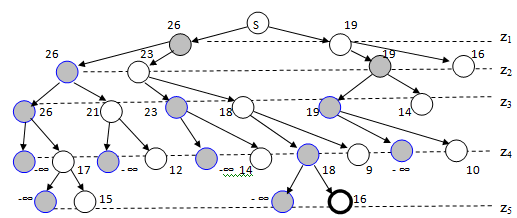


Рис. 1. Дерево ветвлений G1(X1,U1), построенное алгоритмом 1 применительно к задаче (3).

Оптимальный вектор переменных задачи (3), Z= {1,0,01,0}, соответствующее значение целевой функции F=16, число вершин дерева G1(X1, U1), │X1│=27.

Kdfhbvlkxj sdfkljvblsjkvb skjdvblksbv ksbgv skdvbsk vbnkb sbdcvhdsbckhdvlh dcbsdhkbcsbd chbv kdbcskb dchvlshdfcvbnks kdhvcshdv chvhdcb c.skbdckdsvbvhvdhakb jdghfbs;fuig;cbsabc a.kugfckb.chkvlauihcb .kabcuiahfjb ckb; ajbc;uoahcajb c kabcikhbalj c

* 1. **Динамическое программирование**

Содержательно решение задачи дискретной оптимизации с помощью динамического программирования может быть проиллюстрировано последовательным, ярус за ярусом, построением дерева ветвлений G2(X2,U2), стратегия которого существенно отличается от стратегии, описанной в приведенном выше алгоритме 1. Ниже приводится агрегированное описание процедуры такого рода при условии, что все коэффициенты системы (2) неотрицательны.

Алгоритм

1. На множестве висячих и не помеченных вершин построенной части дерева ветвлений G2(X2,U2) выбирается вершина xj. Если таковой вершины нет, то перейти к шагу 4. Если выбор осуществляется на первой итерации, то такой вершиной *apriori* считается корневая вершина дерева. 2. Осуществляется ветвление из выбранной на шаге 1 последней итерации вершины xj и вычисляются компоненты каждого вектора R(xk), отвечающего всем висячим вершинам построенного «куста» (величина h=1). Для определения первой компоненты этого вектора используется (5), остальные компоненты вычисляются по формуле:

rj(xk) = (6)

гдеxk – вершина, принадлежащая множеству висячих вершин куста, в которую заходит дуга из выбранной на шаге 1 последней итерации вершины xj. 3. Все вершины куста, построенного на шаге 2 последней итерации, помечаются, перейти к шагу 1. 4. Убираются все пометки подмножества висячих вершин дерева ветвлений. 5. Если на множестве висячих вершин существует вершина xk, для которой справедливо хотя бы одно из условий: 

а) I1(xk) = I1(xq);

б) δ(xk) «хуже» чем δ(xq);

в) rj(xq) ≥ rj(xk),

то эта вершина вычеркивается.

6. Если число введенных в базис переменных висячих вершин дерева ветвлений равно числу переменных решаемой задачи, то перейти к следующему шагу, в противном случае перейти к шагу 1. 7. На множестве не вычеркнутых висячих вершин выбирается вершина xk с наилучшей первой компонентой соответствующего ей вектора R(xk). 8. Алгоритм закончен. Вектор переменных Z, соответствующий выбранной на шаге 7 вершине xk, является оптимальным.

|  |
| --- |
| Ниже на рис. 2 изображено дерево G2(X2, U2), построенное с помощью алгоритма 2 применительно к задаче (3), векторы у вершин определены на шагах 3 каждой итерации    Рис. 2. Дерево ветвлений G2(X2,U2), построенное c помощью алгоритма 2 применительно к задаче (3). |

На рис. 2 перечеркнуты вершины, вычеркнутые на шаге 6 алгоритма 2, и отсутствуют векторы перечеркнутых вершин, для которых справедливо условие в) шага 6, причем число вершин построенного дерева G2(X2, U2), │X2│=35.

1. **Композитные алгоритмы, реализующие неявный перебор на частичных планах, и гарантирующие глобально оптимальное решение**

Ниже приводятся модификации методов типа ветвей и границ, методика отсечения в которых включает технологии, используемые в динамическом программировании, а также модификации алгоритма, реализующего динамическое программирование с привлечением оценок, используемых методами типа ветвей и границ.

* 1. **Композитная реализация метода типа ветвей и границ**

В рамках реализации этого подхода каждой вершине xj строящегося дерева ветвлений G3(X3,U3) ставится в соответствие вектор V(xj), первой компонентой которого является оценка, используемая в традиционной реализации этого алгоритма, а остальные компоненты совпадают с компонентами вектора R(xj), описание которого приводится в предыдущем разделе, посвященном динамическому программированию. Ниже приводится содержательное описание такой процедуры.

Алгоритм

1. На множестве не вычеркнутых висячих вершинпостроенной части дерева ветвлений G(X,U) выбирается вершина xj с «наилучшей» первой компонентой вектора V(xj). Если это осуществляется на первой итерации, то такой вершиной *apriori* считается корневая вершина дерева.

2. Если выбранной вершине отвечает равенство I1 = I, то перейти к шагу 8, в противном случае – к следующему шагу.

3. Ветвление осуществляется из выбранной на шаге 1 последней итерации вершины xj. Новое множество висячих вершин дерева вновь обозначаем 

4. Для каждой висячей вершины построенного на предыдущем шаге «куста» xj формируется вектор V(xj). Для формирования первой компоненты этого вектора используется процедура, применявшаяся для вычисления соответствующей оценки в методе типа ветвей и границ, для определения второй компоненты вектора V(xj) используется (5), остальные компоненты вычисляются по формуле (6).

1. Если на множестве висячих вершин существует вершина xk, для которой справедливо хотя бы одно из условий: 

а) I1(xk) = I1(xq);

б) δ(xk) «хуже» чем δ(xq);

в) rj(xq) ≥ rj(xk),

то эта вершина вычеркивается.

1. Если существуют вершины и , для которых справедливо:

а) I1(xj) = I1(xq);

б) δ(xj) «лучше» чем Δ(xq),

то вершина xqвычеркивается.

1. Перейти к шагу 1.

8. Алгоритм закончен. Вектор переменных, соответствующий выбранной вершине, является оптимальным.

Ниже на рис. 3 изображено дерево G3(X3, U3), построенное с помощью алгоритма 3 применительно к задаче (3). Число вершин построенного дерева ветвлений G3(X3, U3), │X3│=23, причем однократно перечеркнуты вершины, отвечающие применению шага 5, а дважды перечеркнутые вершины отображают реализацию шага 6.

В соответствии с принятыми выше допущениями применительно к задаче (3) относительное улучшение интеллекта метода типа ветвей и границ, осуществляющего фронтальный спуск по дереву ветвлений, в результате привлечения технологий отсечения, присущих динамическому программированию, определяется выражением:

 (7)

|  |
| --- |
| Рис. 3. Дерево ветвлений G3(X3,U3), построенное алгоритмом 3 применительно к задаче (3). |

* 1. **Композитная реализация динамического программирования**

Предлагаемый далее подход отличается от приведенного выше алгоритма 2 следующими параметрами:

1. Каждой вершине xj дерева ветвлений ставится в соответствие вектор V(xj), аналогичный тому, который использовался в алгоритме 3.
2. Если на i-й итерации i-у ярусу построенной части дерева ветвлений принадлежат две такие вершины xj и xq , у которых вторая компонента вектора V(xj) «лучше» первой компоненты вектора V(xq), то вершина xq вычеркивается. Практически эта операция совпадает с шагом 6 алгоритма 3.

Ниже приводится содержательное описание композитной реализации динамического программирования, включающей алгоритм 2 и приведенные выше параметры.

кружит над пирсом, пытаясь убедиться, что зрение его не обманывает, и лишь после этого связывается с диспетчерами Шорема.   
  
Пирс теперь окончательно раскололся пополам – изломанные края обеих половин с изумлением смотрят друг на друга, а между ними вода, и в ней сорок пять тонн деревянных обломков и скрученного в узлы металла. Группка людей, застрявших на дальней части пирса, повисшей над морем, в отчаянии застыла на самом краю в надежде, что их увидит или услышит кто-нибудь, способный прийти на помощь. Остальные все же стараются держаться подальше от края, пытаясь оценить, насколько прочна уцелевшая часть пирса. Три пары угодили в ловушку, застряв в аттракционе «Поезд-призрак», и теперь вынуждены лишь прислушиваться к голосам и шумам, доносящимся снаружи; выбираться самостоятельно они боятся – им

Алгоритм

1. На множестве висячих, не вычеркнутых и не помеченных вершин построенной части дерева ветвлений G4(X4,U4) выбирается вершина, которую обозначаем, как xj. Если таковой вершины нет, то перейти к шагу 4. Если выбор осуществляется на первой итерации, то такой вершиной *apriori* считается корневая вершина дерева G4(X4,U4).
2. Осуществляется ветвление из выбранной на шаге 1 последней итерации вершины xj и вычисляются компоненты каждого вектора V(xk), отвечающего всем висячим вершинам построенного «куста». Процедура вычисления компонент этого вектора совпадает с процедурой, описанной на шаге 4 алгоритма 3.
3. Пометить все полученные на шаге 2 последней итерации висячие вершины куста и перейти к шагу 1.
4. Убрать пометки всех висячих вершин построенной части дерева ветвлений G4(X4,U4).
5. Если на множестве висячих вершин дерева G4(X4,U4) существует вершина xk, для которой справедливо хотя бы одно из условий: 

а) I1(xk) = I1(xq);

б) δ(xk) «хуже» чем δ(xq);

в) rj(xq) ≥ rj(xk),

то вершина xk вычеркивается.

1. Если на множестве висячих вершин построенной части дерева G4(X4,U4) существуют две такие вершины xj и xq , у которых вторая компонента вектора V(xj) «лучше» первой компоненты вектора V(xq), то вершина xq вычеркивается.
2. На множестве не вычеркнутых висячих вершин выбирается вершина xk с наилучшей первой компонентой соответствующего ей вектора V(xk).
3. Если выбранной вершине отвечает число переменных, введенных в базис, равное числу переменных решаемой задачи, то перейти к шагу 9, в противном случае перейти к шагу 1.
4. Алгоритм закончен. Вектор переменных Z, соответствующий выбранной на шаге 9 вершине xk, является оптимальным.

Ниже на рис. 4 изображено дерево G4(X4, U4), построенное с помощью алгоритма 4 применительно к задаче (3), векторы у вершин определены на шагах 3 каждой итерации. В соответствии с принятыми выше допущениями применительно к задаче (3) относительное улучшение интеллекта динамического программирования, в результате привлечения технологий отсечения, присущих методам типа ветвей и границ, определяется выражением:

 (8)

|  |
| --- |
| где Х2 – множество вершин дерева G2(X2, U2) (см. выше рис. 2), Х4 - множество вершин дерева G4(X4, U4) (см. ниже рис.4).  Рис. 4. Дерево ветвлений, иллюстрирующее поиск решения задачи (3) алгоритмом 4. |

Знаком « - ∞» помечены вершины, отвечающие векторам переменных, нарушающим ограничения системы (3), однократно перечеркнуты вершины, отвечающие применению шага 6, а дважды перечеркнутые вершины отображают реализацию шага 7.

В соответствии с принятыми выше допущениями применительно к задаче (3) относительное улучшение интеллекта динамического программирования, в результате привлечения технологий отсечения, присущих методам типа ветвей и границ, определяется выражением:

 (8)

# Глава 3. Выбор и обоснование программной реализации

## 3.1. Язык программирования Delphi 7

## Delphi — императивный, структурированный, объектно-ориентированный язык программирования, диалект Object Pascal. Начиная со среды разработки Delphi 7.0, в официальных документах Borland стала использовать название Delphi для обозначения языка Object Pascal. Начиная с2007 года уже язык Delphi (производный от Object Pascal) начал жить своей самостоятельной жизнью и претерпевал различные изменения, связанные с современными тенденциями (например, с развитием платформы .NET) развития языков программирования: появились class helpers, перегрузки операторов и другое.

## Целевая платформа

## Изначально среда разработки была предназначена исключительно для разработки приложений MicrosoftWindows, затем был реализован также для платформ Linux (как Kylix), однако после выпуска в 2002 году Kylix3 его разработка была прекращена, и, вскоре после этого, было объявлено о поддержке Microsoft .NET.

## Реализация среды разработки проектом Lazarus (Free Pascal, компиляция в режиме совместимости с Delphi) позволяет использовать его для создания приложений на Delphi для таких платформ, как Linux, Mac OS X иWindows CE.

## Также предпринимались попытки использования языка в проектах GNU (например, Notepad GNU) и написания компилятора.

## История языка

## Object Pascal — результат развития языка Турбо Паскаль, который, в свою очередь, развился из языка Паскаль. Паскаль был полностью процедурным языком, Турбо Паскаль, начиная с версии 5.5, добавил в Паскаль объектно-ориентированные свойства, а в Object Pascal — динамическую идентификацию типа данных с возможностью доступа к метаданным классов (то есть к описанию классов и их членов) в компилируемом коде, также называемом интроспекцией — данная технология получила обозначение RTTI. Так как все классы наследуют функции базового класса TObject, то любой указатель на объект можно преобразовать к нему, после чего воспользоваться методом ClassType и функцией TypeInfo, которые и обеспечат интроспекцию.

## Также отличительным свойством Object Pascal от С++ является то, что объекты по умолчанию располагаются в динамической памяти. Однако можно переопределить виртуальные методы NewInstance и FreeInstanceкласса TObject. Таким образом, абсолютно любой класс может осуществить «желание» «где хочу — там и буду лежать». Соответственно организуется и «много кучность».

## Object Pascal (Delphi) является результатом функционального расширения Turbo Pascal.

## Delphi оказал огромное влияние на создание концепции языка C# для платформы .NET. Многие его элементы и концептуальные решения вошли в состав С#. Одной из причин называют переход Андерса Хейлсберга, одного из ведущих разработчиков Дельфи, из компании Borland Ltd. в Microsoft Corp.

## • Версия 8 способна генерировать байт-код исключительно для платформы .NET. Это первая среда, ориентированная на разработку мультиязычных приложений (лишь для платформы .NET);

## • Последующие версии (обозначаемые годами выхода, а не порядковыми номерами, как это было ранее) могут создавать как приложения Win32, так и байт-код для платформы .NET.

## Delphi for .NET — среда разработки Delphi, а также язык Delphi (Object Pascal), ориентированные на разработку приложений для .NET.

## Первая версия полноценной среды разработки Delphi для .NET — Delphi 8. Она позволяла писать приложения только для .NET. Delphi 2006 поддерживает технологию MDA с помощью ECO (Enterprise CoreObjects) версии 3.0.

## В марте 2006 года компания Borland приняла решение о прекращении дальнейшего совершенствования интегрированных сред разработки JBuilder, Delphi и C++ Builder по причине убыточности этого направления. Планировалась продажа IDE-сектора компании. Группа сторонников свободного программного обеспечения организовала сбор средств для покупки у Borland прав на среду разработки и компилятор.

## Однако в ноябре того же года было принято решение отказаться от продажи IDE бизнеса. Тем не менее, разработкой IDE продуктов теперь будет заниматься новая компания — CodeGear, которая будет финансово полностью подконтрольна Borland.

## В августе 2006 года Borland выпустил облегченную версию RAD Studio под именем Turbo: Turbo Delphi (дляWin32 и .NET), Turbo C#, Turbo C++.

## В марте 2008 года было объявлено о прекращении развития этой линейки продуктов.

## В марте 2007 года CodeGear порадовала пользователей обновленной линейкой продуктов Delphi 2007 forWin32 и выходом совершенно нового продукта Delphi 2007 for PHP.

## В июне 2007 года CodeGear представила свои планы на будущее, то есть опубликовала так называемыйro admap.

## Embarcadero RAD Studio 2010

## 25 августа 2008 года компания Embarcadero, новый хозяин CodeGear, опубликовала пресс-релиз на Delphifor Win32 2009. Версия привнесла множество нововведений в язык, как то[8]:

## • По умолчанию полная поддержка Юникода во всех частях языка, VCL и RTL; замена обращений ко всем функциям Windows API на юникодные аналоги (то есть MessageBox вызывает MessageBoxW, а неMessageBoxA).

## • Обобщённые типы, они же generics.

## • Анонимные методы.

## • Новая директива компилятора $POINTERMATH [ON|OFF].

## • Функция Exit теперь может принимать параметры в соответствии с типом функции.

## Вышедшая в 2011 году версия Delphi XE2 добавила компилятор Win64 и кросс-компиляцию для операционных систем фирмы Apple.

## Компиляторы

## • Embarcadero Delphi (ранее наз. CodeGear Delphi и Borland Delphi) — наверное, самый известный компилятор, который является последователем Borland Pascal и Turbo Pascal. Используется Win16 (Delphi 1), Win32 (Delphi 2 и позже), Win64 (Delphi 16 (XE2) и позже), а также .NET 1.x, 2.0 (Delphi 8, Delphi 2005-Delphi2007). Поддержка .NET, впоследствии выделена в отдельный продукт, известный как Oxygene.

## • Free Pascal (FPC) — свободный компилятор Оbject Pascal, который поддерживает различные диалекты Паскаля, включая Turbo Pascal, Delphi и собственные диалекты. На текущий момент, FPC может генерировать код для x86, x86-64, PowerPC, SPARC и процессоров ARM, а также для различных операционных систем, в том числе для Microsoft Windows, Linux, FreeBSD, Mac OS. Существует несколько сред разработки программного обеспечения для FPC (один из самых известных представителей — Lazarus).

## • GNU Pascal (отдельно разработанная версия из GCC). Не ставит целью продолжение серии диалектов Delphi, как составляющей Паскаля, но тем не менее содержит режим совместимости Borland Pascal, и очень медленно приспосабливает компоненты языка Delphi. Не подходит для компиляции больших проектов, содержащих код Delphi, но стоит отметить, что его поддерживают большинство операционных систем и архитектур.

## • Oxygene (ранее известен как Chrome) — компилятор Object Pascal, который интегрирован в Microsoft VisualStudio. Также доступный в виде компилятора с вольной командной строкой CLI. Использует .NET и моноплатформы. В настоящий момент продаётся под маркой Embarcadero Delphi Prism.

## • MIDletPascal — язык программирования с Delphi-подобным синтаксисом, и одноименный компилятор, который преобразует исходный код в компактный и быстрый байт-код Java.

## • PocketStudio — основанная на Паскале, IDE для Palm OS.

## • Virtual Pascal — Бесплатный компилятор и текстовая IDE для Win32, OS/2 и Линукса. На тот момент очень быстрый и весьма совместимый (частично поддерживаются конструкции Delphi 5). Внешне очень похож на текстовую среду Borland Pascal 7, хотя отсутствует совместимая с ним графика, например. Однако разработка окончилась в 2004 году, а исходники открыты не были. С тех пор FPC ушёл намного вперед и в целом для программирования лучше он. Тем не менее, VP остаётся очень неплохим вариантом быстрой замены ещё более устаревших версий Borland Pascal для школы/института, учитывая родную работу в Win32без проблем с русскими кодировками.

## Объектно-ориентированные особенности языка

## *Инкапсуляция*

## Объединение и скрытие объектных данных, а также обрабатывающих их методов внутри конкретного класса от пользователя называется инкапсуляцией.

## *Наследование*

## При создании новых объектов получить все свойства и методы от своих предков называют наследованием. Такие объекты наследуют после своего создания все поля, свойства, события, методы и прочее от своих предков. Наследование часто избавляет разработчиков от рутинной работы и позволяет не мешкая приступить к разработке чего-то нового.

## *Полиморфизм*

## Это методы различных объектов, которые могут иметь одинаковые имена, но по внутреннему содержимому отличаются друг от друга.

## Расширения файлов

## • .pas — исходный код модуля (pascal)

## • .dpr — исходный код проекта (pascal)

## • .dproj — исходный код проекта (xml)

## • .dproj.local — исходный код проекта (xml)

## • .dfm — исходный код формы

## • .dpk — скомпилированный пакет

## • .dcu — скомпилированный модуль

## • .exe — скомпилированное приложение

## • .res — ресурсы

## • .dsk — привязки к файлам

## • .identcache — кэшированные привязки к файлам

## Известное программное обеспечение, созданное на Delphi

## Среди многих распространённых программных продуктов, написанных на Delphi, можно найти[9]:

## • Продукция Embarcadero: Embarcadero Delphi, Embarcadero C++ Builder, Borland JBuilder 1 и 2 версии.

## • Администрирование и разработка баз данных: MySQL Tools, IBExpert.

## • Инженерное программное обеспечение: Altium Designer.

## • Файловые менеджеры: Total Commander, Frigate.

## • Просмотрщики графики: FastStone Image Viewer, FuturixImager, drComRead.

## • Видео- и аудиопроигрыватели: Light Alloy, The KMPlayer, AIMP, X-Amp.

## • Программы мгновенного обмена сообщениями: QIP, R&Q, графический интерфейс Skype, The Bat!, PopTray, FeedDemon.

## • Создание музыки: FL Studio, Guitar Pro (до версии 6.0).

## • Разработка программного обеспечения: Dev-C++, DUnit, Game Maker, Inno Setup, PyScripter.

## • Веб-разработка: Macromedia HomeSite.

## • Текстовые редакторы: BirEdit, Notepad GNU, Bred.

## • Бухучёт и налогообложение: «ПАРУС», AVARDA (до версии 6.x включительно).

## • Программы для создания анимаций: Pivot Stickfigure Animator.

## • Программы для сжатия данных: ALZip, PowerArchiver, PeaZip.

## • Компьютерные игры: Age of wonders, «Космические рейнджеры», Venom. Codename: Outbreak, SpaceEmpires V, «Правда о девятой роте».

## • Графические редакторы: Real Paint.

## 3.2. Интерфейс программы

## 1 шаг: Вводится количество переменных и предел к которому стремиться целевая функция.

## 

## 2 шаг: Вводятся коэффициенты целевой функции.

## 

## 3 шаг: Задается количество ограничений.

## 

## 4 шаг: Вводятся коэффициенты ограничения ее знак и само ограничение.

## 

## 5 шаг: Выбирается метод решения задачи.

## 

## 6 шаг: Производится расчет.

## 

## Результат выводится в виде висячих вершин на каждой итерации.

# Глава 4. Экспериментальная часть

**Целью эксперимента является следующее:**

Целью эксперимента является сравнение времени работы алгоритма классического метода ветвей и границ и композитного алгоритма.

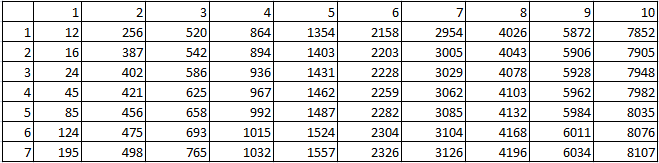
Эксперимент был поставлен на компьютере со следующими характеристиками:

* + Процессор Intel (R) Pentium 4 (R) 2.1 GHz;
  + 3 ГБ ОЗУ;
  + Microsoft Windows 7.

**Эксперимент**

Для сравнения использовались 3 критерия: время (в мс.), количество коэффициентов, количество ограничений.

Экспериментальные значения классического метода:



Экспериментальные значения композитного метода:

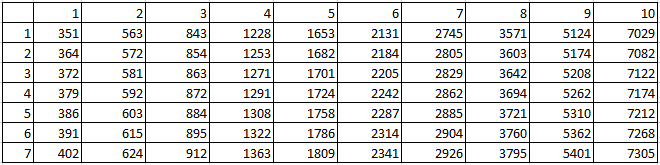


График классического метода:

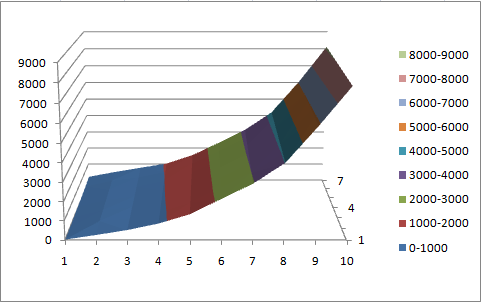
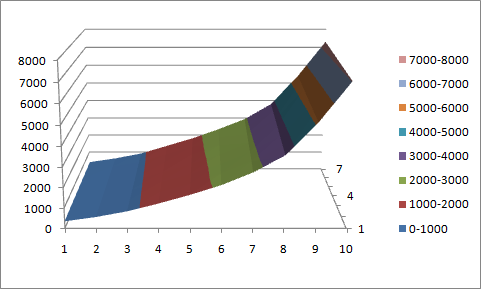
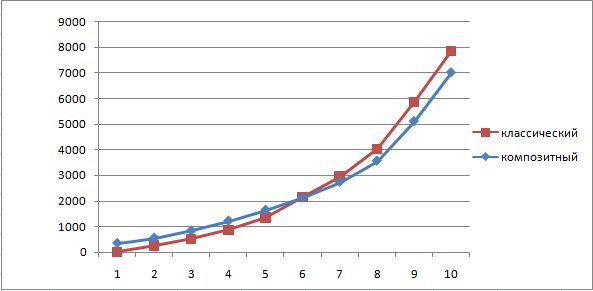


График композитного метода:



Сравнительный график двух методов:



Из графиков видно что при малом количестве переменных композитный метод не эффективен. Значительное преимущество композитный метод показывает при больших размерностях задачи.

Таким образом, рассмотренный в Главе 4 эксперимент позволил визуально выявить зависимость количества переменных и времени выполнения времени.

# Заключение

В результате выполнения выпускной квалификационной работы была разработана программная реализация классического метода ветвей и границ и композитного метода. Также был осуществлен сравнительный анализ обоих методов. Экспериментально была доказана эффективность композитного алгоритма над классическим при больших размерностях задачи.

Полученный программный продукт можно использовать для демонстрации на лабораторных работах по курсу «Теории принятия решения»

# Список использованной литературы

1. Лорьер Ж. Л. Системы искусственного интеллекта, М.: Мир, 1991. 568 с.

2. G. F. Luger. Artificial Intelligence: Structures and Strategies for Complex Problem Solving, 4-th edition, Williams, 2005, pp. 864 с. ISBN 5-8459-0437-4.

3. Handbook of Operations Research, v.2: Models and Applications. Edited by Joseph J. Moder and Salah E. Elmaghraby, North Carolina State University, 1978, 677p.

4. Кормен Т. Х., Лейзерсон Ч. И., Ривест Р. Л., Штайн, К. Алгоритмы: построение и анализ, 2-e издание. М.: «Вильямс», 2005. Стр. 833 – 839.

5. A. H. Land and A. G. Doig. An automatic method of solving discrete programming problems, Econometrica, Vol. 28, No. 3. (Jul., 1960), pp. 497-520.

6. Беллман Р. Динамическое программирование. — М.: Изд-во иностранной литературы, 1960.

7. V. Chvátal et al. Selected combinatorial research problems. Technical Report STAN-CS-72-292. Computer Science Department, Stanford University, 1972.

8. Корбут А.А., Финкельштейн Ю.Ю. Дискретное программирование. М.: Наука, 1969. 368 с.

9. Гроппен В.О. Повышение эффективности методов типа ветвей и границ для комбинаторных задач с булевыми переменными. ‘’Автоматика и телемеханика’’ N 5, 1978г., с. 105-112.

10. Гроппен В.О. Эффективная организация комбинаторных алгоритмов на однородных вычислительных средствах. Труды 5-го Международного семинара «Прикладные аспекты теории автоматов», Варна, 1979г., с. 358-364.

11. Гроппен В.О. Экстремальные задачи на взвешенных графах. Изд. Фламинго, Владикавказ, 2012г., 90 с.

12. Кербер М. Л., Полимерные композиционные материалы. Структура. Свойства. Технологии. — СПб.: Профессия, 2008, 560 с.

13. Перепелкин К.Е. Армирующие волокна и волокнистые полимерные композиты, СПб.: Научные основы и технологии, 2009 г., 380 с.

14. Мелешко А.И. Половников С.П.Углерод, углеродные волокна, углеродные композиты, 2007, ISBN 5-88070-119-0, стр. 192.

15. Васильев В. В., Механика конструкций из композиционных материалов, М.: Машиностроение, 1988, 272 с.

16. Федунов Б. Е. Методика экспресс - оценки реализуемости графа решений оператора антропоцентрического объекта на этапе разработки спецификаций алгоритмов бортового интеллекта. М.: Известия РАН, Теория и системы управления (ТиСУ), №1, 2002.

17. А. А. Разборов. О сложности вычислений //МЦНМО, Математическое просвещение, № 3, 1999, стр. 127-141.

# Приложение. Листинг программы

unit Unit1;

interface

uses

Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,

Dialogs, StdCtrls, Grids, ExtCtrls;

type

TForm1 = class(TForm)

Button1: TButton;

Button2: TButton;

Edit1: TEdit;

Memo1: TMemo;

Memo2: TMemo;

Memo3: TMemo;

Memo4: TMemo;

Memo5: TMemo;

Label1: TLabel;

Label2: TLabel;

StringGrid1: TStringGrid;

Button3: TButton;

ComboBox1: TComboBox;

Label3: TLabel;

LabeledEdit1: TLabeledEdit;

Button4: TButton;

StringGrid2: TStringGrid;

Button5: TButton;

StringGrid3: TStringGrid;

StringGrid4: TStringGrid;

Label4: TLabel;

Label5: TLabel;

Label6: TLabel;

ComboBox2: TComboBox;

Label7: TLabel;

Label8: TLabel;

Label9: TLabel;

Label10: TLabel;

procedure Button1Click(Sender: TObject);

procedure Button2Click(Sender: TObject);

procedure Button3Click(Sender: TObject);

procedure Button4Click(Sender: TObject);

procedure Button5Click(Sender: TObject);

private

{ Private declarations }

public

{ Public declarations }

end;

var

Form1: TForm1;

var

i,j,g,l,ii,jj:integer;

s,st,str,kmin:string;

masskomb:array[1..100002,1..100]of integer; //ìàññèâ áèíàðíîãî ñ÷åò÷èêà

mainf:array[1..100]of integer; //êîýô öåëåâîé ôóíêöèè

border:array[1..100]of integer; //îãðàíè÷àþùåå ÷èñëî

znak:array[1..100]of string; //çíàê â îãðàíè÷åíèè

ogr:array[1..100,1..100]of integer; // ñïèñîê îãðàíè÷åíèé[êîë îãð],[êîýô â îãð]

vektor:array[1..100]of integer; // âåêòîð çíà÷åíèé ïåðåìåííûõ

helpmain:array[1..100]of integer;

helpkpzmain:array[1..100]of integer;

helpogr:array[1..100,1..100]of integer;

helpkpzsravn:array[1..100]of string;

vverkom:array[1..100002]of string; //äëÿ ðåãèñòðàöèè íå íóæíûõ âåðøèí

mtree:array[1..100002,1..100]of integer;// ìàññèâ äåðåâî [êîë âåòâåé],[òåêóùåå ÷èñëî íà âåòâè, ñòóïåíü âåòâè, ïîïàäàíèå â îãðàíè÷åíèÿ...]

kolkof:byte; // êîëè÷åñòâî êîýôèöåíòîâ

kolvektor:byte;// êîëè÷åñòâî çíà÷åíèé âåêòîðà

kologr:byte; // êîëè÷åñòâî îãðàíìè÷åíèé

KolKom:integer; // êîëè÷åñòâî êîìáèíàöèé

MaxElVektor:integer; //ìàêñèìàëüíûé åëåìåíò âåêòîðà

MinElVektor:integer; //ìèíèìàëüíûé åëåìåíò âåêòîðà

predel:integer; //ê ÷åìó ñòðåìèòñÿ ôóíêöèÿ (ìèí,ìàõ)

extr:integer;

kolkpzsravn:byte;

tkextr:integer;// òåêóùèé ýêñòðåìóì

isextr:integer;// èñêîìûé ýêñòðåìóì

mainsum,mainkpzsum,ogrsum:integer;

hlp,hlpit,hlprzr,max:integer;

konec,kpzdel,kompozresh:boolean;

Sravn:string;

iter,iterkpz:integer;

ttime:Longint;

implementation

{$R \*.dfm}

procedure TForm1.Button1Click(Sender: TObject);

begin

ttime:=0;

//kolkof:=5;

kolkof:=strtoint(Edit1.Text);

StringGrid1.ColCount:=kolkof;

predel:=1;

if ComboBox1.Text = 'min' then

predel:=2;

if ComboBox1.Text = 'max' then

predel:=1;

StringGrid2.ColCount:=kolkof;

StringGrid1.Enabled:=true;

Button3.Enabled:=true;

//-

kolvektor:=2;

vektor[1]:=0;

vektor[2]:=1;

//vektor[3]:=2;

end;

procedure TForm1.Button3Click(Sender: TObject);

begin

{

mainf[1]:=4;

mainf[2]:=7;

mainf[3]:=8;

mainf[4]:=5;

mainf[5]:=6;

}

for i:=1 to kolkof do

begin

mainf[i]:=strtoint(StringGrid1.Cells[i-1,0]);

Label2.Caption:=Label2.Caption+inttostr(mainf[i])+'-';

end;

LabeledEdit1.Enabled:=true;

Button4.Enabled:=true;

end;

procedure TForm1.Button4Click(Sender: TObject);

begin

//kologr:=2;

kologr:=strtoint(LabeledEdit1.text);

StringGrid2.RowCount:=kologr;

StringGrid3.RowCount:=kologr;

StringGrid4.RowCount:=kologr;

StringGrid2.Enabled:=true;

StringGrid3.Enabled:=true;

StringGrid4.Enabled:=true;

Button5.Enabled:=true;

end;

procedure TForm1.Button5Click(Sender: TObject);

begin

{

ogr[1,1]:=5;

ogr[1,2]:=2;

ogr[1,3]:=3;

ogr[1,4]:=8;

ogr[1,5]:=1;

ogr[2,1]:=2;

ogr[2,2]:=4;

ogr[2,3]:=3;

ogr[2,4]:=9;

ogr[2,5]:=6;

znak[1]:='>=';

znak[2]:='<=';

border[1]:=12;

border[2]:=17; }

//Label2.Caption+inttostr(mainf[i])+'-';

Label2.Caption:='';

for i:=1 to kologr do

begin

for j:=1 to kolkof do

begin

ogr[i,j]:=strtoint(StringGrid2.Cells[j-1,i-1]);

Label2.Caption:=Label2.Caption+inttostr(ogr[i,j])+'-';

end;

znak[i]:=StringGrid3.Cells[0,i-1];

border[i]:=strtoint(StringGrid4.Cells[0,i-1]);

Label2.Caption:=Label2.Caption+' '+znak[i]+inttostr(border[i])+'|';

end;

ComboBox2.Enabled:=true;

Button2.Enabled:=true;

end;

procedure TForm1.Button2Click(Sender: TObject);

begin

ttime:=0;

iter:=0;

iterkpz:=0;

kompozresh:=false;

if ComboBox2.Text = 'êîìïîçèòíàÿ âåðñèÿ ìåòîäà âåòâåé è ãðàíèö' then

kompozresh:=true;

//==================================================================================

kolkom:=1;

for i:=1 to kolkof do

begin

masskomb[kolkom,i]:=1;

end;

hlp:=0;

while hlp<(kolkof\*(kolvektor)) do

begin

inc(ttime);

for i:=kolkof downto 1 do

begin

if masskomb[kolkom,i]=kolvektor+1 then begin inc(masskomb[kolkom,i-1]); masskomb[kolkom,i]:=1; end;

end;

str:='';

for i:=1 to kolkof do

str:=str+inttostr(masskomb[kolkom,i]);

Memo1.Lines.Add(str);

hlp:=0;

for i:=1 to kolkof do

hlp:=hlp+masskomb[kolkom,i];

inc(kolkom);

for i:=1 to kolkof do

masskomb[kolkom,i]:=masskomb[kolkom-1,i];

inc(masskomb[kolkom,kolkof]);

end;

kolkom:=kolkom-1;

Label2.Caption:=inttostr(kolkom)+'-'+inttostr(kolkof);

//==========================================================================

for i:=1 to KolKom do

begin

MTree[i,2]:=0;

vverkom[i]:='p';

end;

MaxElVektor:=Vektor[1];

MinElVektor:=Vektor[1];

for i:=1 to KolVektor do

begin

if MaxElVektor<Vektor[i] then

MaxElVektor:=Vektor[i];

if MinElVektor>Vektor[i] then

MinElVektor:=Vektor[i];

end;

Label2.Caption:=inttostr(MaxElVektor)+'-'+inttostr(MinElVektor);

//==========================================================================

begin

if predel=1 then

extr:=0;

if predel=2 then

extr:=10000;

for i:=1 to kolkom do

begin

mtree[i,1]:=extr;

end;

//-----------------

konec:=false;Sravn:='';

while konec=false do

begin

Memo2.Clear;Memo3.Clear;Memo4.Clear;

for i:=1 to kolkom do

begin

if (mtree[i,1]=extr) and (vverkom[i]='p') then

inc(mtree[i,2]);

end;

//-------------------------------

st:='';hlpit:=-1;hlprzr:=0;

for i:=1 to kolkom do

begin

for j:=1 to kolkof do

begin

if predel=1 then

helpmain[j]:=MaxElVektor;

if predel=2 then

helpmain[j]:=MinElVektor;

helpkpzmain[j]:=MinElVektor;

end;

for j:=1 to kologr do

for g:=1 to kolkof do

begin

if (znak[j]='>=') or (znak[j]='>') then

helpogr[j,g]:=MaxElVektor;

if (znak[j]='<=') or (znak[j]='<') then

helpogr[j,g]:=MinElVektor;

end;

for j:=1 to mtree[i,2] do

begin

helpmain[j]:=vektor[masskomb[i,j]];

helpkpzmain[j]:=vektor[masskomb[i,j]];

end;

for j:=1 to kologr do

for g:=1 to mtree[i,2] do

begin

helpogr[j,g]:=vektor[masskomb[i,g]];

end;

mainsum:=0;mainkpzsum:=0;

for j:=1 to kolkof do

begin

mainsum:=mainsum+helpmain[j]\*mainf[j];

mainkpzsum:=mainkpzsum+helpkpzmain[j]\*mainf[j];

end;

mtree[i,1]:=mainsum;

mtree[i,3]:=mainkpzsum;

for j:=1 to kologr do

begin

ogrsum:=0;

for g:=1 to kolkof do

begin

ogrsum:=ogrsum+helpogr[j,g]\*ogr[j,g];

end;

mtree[i,j+3]:=ogrsum;

if znak[j]='>=' then

if mtree[i,j+3] >= border[j] then

else

begin

vverkom[i]:='n';

mtree[i,1]:=0;

end;

if znak[j]='>' then

if mtree[i,j+3] > border[j] then

else

begin

vverkom[i]:='n';

mtree[i,1]:=0;

end;

if znak[j]='<=' then

if mtree[i,j+3] <= border[j] then

else

begin

vverkom[i]:='n';

mtree[i,1]:=0;

end;

if znak[j]='<' then

if mtree[i,j+3] < border[j] then

else

begin

vverkom[i]:='n';

mtree[i,1]:=0;

end;

end;

end; //áîëüøîé öèêë

//-------------------------------------------

for i:=1 to kolkom do

begin

str:='';

for l:=1 to kologr do

str:=str+'\_'+(inttostr(mtree[i,l+3]));

Memo2.Lines.add((inttostr(mtree[i,1]))+'\_'+(inttostr(mtree[i,3]))+str);

Memo3.Lines.add((inttostr(mtree[i,2])));

Memo4.Lines.add(vverkom[i]);

if ((hlpit<>vektor[masskomb[i,mtree[i,2]]]) or (hlprzr<>mtree[i,2])) and (vverkom[i]<>'n') then

begin

hlpit:=vektor[masskomb[i,mtree[i,2]]];

hlprzr:=mtree[i,2];

st:=st+inttostr(mtree[i,1])+'\_'+inttostr(mtree[i,3])+str+' || ';

end;

//êîìïîçèòíàÿ ÷àñòü

if kompozresh=true then

begin

kolkpzsravn:=kologr+3;

hlp:=1;

for j:=1 to kolkom do

begin

if (mtree[j,2]=mtree[i,2]) then

begin

helpkpzsravn[2]:='+';

if predel=1 then

begin

if (mtree[j,1]<mtree[i,1]) then

helpkpzsravn[1]:='+';

if (mtree[j,1]=mtree[i,1]) then

helpkpzsravn[1]:='=';

if (mtree[j,1]>mtree[i,1]) then

helpkpzsravn[1]:='-';

if (mtree[j,3]<mtree[i,3]) then

helpkpzsravn[3]:='+';

if (mtree[j,3]=mtree[i,3]) then

helpkpzsravn[3]:='=';

if (mtree[j,3]>mtree[i,3]) then

helpkpzsravn[3]:='-';

end;

if predel=2 then

begin

if (mtree[j,1]>mtree[i,1]) then

helpkpzsravn[1]:='+';

if (mtree[j,1]=mtree[i,1]) then

helpkpzsravn[1]:='=';

if (mtree[j,1]<mtree[i,1]) then

helpkpzsravn[1]:='-';

if (mtree[j,3]>mtree[i,3]) then

helpkpzsravn[3]:='+';

if (mtree[j,3]=mtree[i,3]) then

helpkpzsravn[3]:='=';

if (mtree[j,3]<mtree[i,3]) then

helpkpzsravn[3]:='-';

end;

for g:=1 to kologr do

begin

if (znak[g]='>=') or (znak[g]='>') then

begin

if (mtree[j,g+3]<mtree[i,g+3]) then

helpkpzsravn[g+3]:='+';

if (mtree[j,g+3]=mtree[i,g+3]) then

helpkpzsravn[g+3]:='=';

if (mtree[j,g+3]>mtree[i,g+3]) then

helpkpzsravn[g+3]:='-';

end;

if (znak[g]='<=') or (znak[g]='<') then

begin

if (mtree[j,g+3]>mtree[i,g+3]) then

helpkpzsravn[g+3]:='+';

if (mtree[j,g+3]=mtree[i,g+3]) then

helpkpzsravn[g+3]:='=';

if (mtree[j,g+3]<mtree[i,g+3]) then

helpkpzsravn[g+3]:='-';

end;

end;

kpzdel:=true;

for g:=1 to kologr+3 do

if (helpkpzsravn[g]='-') or (helpkpzsravn[g]='=') then

kpzdel:=false;

if (kpzdel=true) then

begin

mtree[j,1]:=0;

vverkom[j]:='n';

if hlp=1 then

begin

inc(iterkpz);

hlp:=0;

end;

end;

end;

end;

end;

end;

Memo5.Lines.Add(st);

//----------------

if predel=1 then

begin

extr:=0;

for i:=1 to kolkom do

if (mtree[i,1]>extr) and (vverkom[i]='p') then

begin

inc(iter);

extr:=mtree[i,1];

tkextr:=i;

end;

end;

if predel=2 then

begin

extr:=1000;

for i:=1 to kolkom do

if (mtree[i,1]<=extr) and (vverkom[i]='p') then

begin

inc(iter);

extr:=mtree[i,1];

tkextr:=i;

end;

end;

//---------------------------

if st=Sravn then

konec:=true

else

Sravn:=st;

end; //êîíåö whila

end; //êîíåö ïðåäåëà ìîæíî óäàëèòü

Label9.Caption:='Îòâåò: ';

str:='';

for i:=1 to kolkom do

begin

if mtree[i,1]=extr then

begin

for l:=1 to kologr do

str:=str+'\_'+(inttostr(mtree[i,l+3]));

Label9.Caption:=Label9.Caption+(inttostr(mtree[i,1]))+'\_'+(inttostr(mtree[i,3]))+str;

break;

end;

end;

Label10.Caption:='Âðåìÿ ó.å.: ';

//Label10.Caption:=Label10.Caption+inttostr(ttime)+' - '+inttostr(iter)+' - '+inttostr(iterkpz)+' - '+inttostr(kologr);

Label10.Caption:=Label10.Caption+floattostr((ttime\*iter)+(ttime\*kologr));

end;

end.