Дискретная оптимизация (П.А. Алисейчик, А.С. Строгалов)

1. Введение

1.1. Цели практикума

1. Использование ООП при проектировании архитектуры программы.
2. Анализ существующих алгоритмов.
3. Разработка своего алгоритма.
4. Оптимизация алгоритма.
5. Анализ сложности алгоритма.
6. Эксперимент и тестирование.
7. Общее представление о графических библотеках (на примере MFC).
8. Архитектура программ с графическим интерфейсом.

1.2. Указания по планированию архтектуры системы

Результатом работы должны быть 4 главных класса:

1. Класс, хранящий данные и решающий задачу ПО ШАГАМ.
2. Класс консольного интерфейса пользователя, позволяющий решать задачу целиком или по шагам.
3. Тестирующий класс.
4. Класс графического интерфейса пользователя, позволяющий решать задачу целиком или по шагам.

Консольное приложение должно использовать классы 1,2,3, MFC-приложение — 1,3,4. Таким образом, классы 1 и 3 должны быть написаны таким образом, чтобы уметь взаимодействовать как с консольным, так и с графическим интерфейсом пользователя. Взаимодействие между классами осуществляется с помощью public-методов.

1.3. Этапы работы

Работа делится на этапы, каждый из которых заканчивается сдачей работающей программы:

1. План архитектуры системы: определения 2-х основных классов (алгоритм решения задачи по шагам и интерфейс пользователя), большинство методов пока без реализации.

2. Консольное приложение без оптимизации: реализована 1-я версия решения задачи по шагам и консольный интерфейс пользователя, файловый ввод-вывод.

3. Тестирование: проверка корректности работы алгоритма на различных входных данных.

4. Оптимизация: оптимизированная версия решения задачи, измерение сложности каждого шага и подсчет общей сложности решения задачи.

5. Анализ сложности и эксперимент: набор тестов, позволяющих обнаружить зависимость общей сложности решения от параметров задачи.

6. Графический интерфейс: MFC-приложение для визуализации хода решения по шагам.

2. Условия задач

2.1. Упаковка предметов в контейнеры

Дано:

N предметов , каждому сопоставлен размер $p(i)$ (вектор действительных чисел длины N.)

S ящиков, в которые можно класть предметы (все ящики имеют вместимость $Vm > p(i)$).

Числа N, S фиксированы.

Программа из входного файла должна получать вектор размеров предметов и значение вместительности ящиков.

Надо:

Реализовать алгоритм оптимальной (по количеству использованных ящиков) упаковки в предметов в ящики, или используя такой алгоритм:

1) отсортировать предметы по убыванию размера

2) первый предмет положить в 1 ящик, второй — туда же, если влезает. если не влезает — во второй. и т.д.

Вывести ответ в выходной файл.

2.2. Задача о рюкзаке

По данному набору из n предметов стоимостями $v_1, v_2, \ldots, v_n$ и весами $w_1, w_2, \ldots, w_n$ (действительные числа, получаемые из входного файла) найти поднабор (с учетом того, что нельзя брать один предмет несколько раз) такой, что его стоимость будет максимальна, среди всех поднаборов веса не более W.

Вывести ответ в выходной файл.

2.3. Покрытие таблицы из 0 и 1

Во входном файле задана матрица $N \times M$ из 0 и 1. Необходимо построить покрытие матрицы, используя следующий алгоритм:

1) Взять столбец веса k. Удалить его и все строки, которые его покрывают из рассмотрения.

2) В полученной матрице повторить процедуру.

(Строка покрывает столбец, если на их пересечении находится 1. таким образом, матрица покрывается набором строк)

Вывести ответ в выходной файл.

2.4. Гамильтонов цикл

В гиперкубе, заданном как векторе размерности 2^N, построить гамильтонов цикл. Вывести ответ в выходной файл.

2.5. Обход графа методом ближайшего соседа

Дан граф, как набор вершин (общее число — F) и ребер (пар вершин) в файле. Используя алгоритм "ближайшего соседа"построить его обход по вершинам. Структура хранения графа — на усмотрение решающего.

Вывести ответ в выходной файл.

2.6. Гамильтонов обход графа

Дан полный граф, для которого выполнено неравенство треугольника, как набор вершин (общее число — F) и ребер (пар вершин) в файле. Построить гамильтонов обход графа. Найти его вес. Структура хранения графа — на усмотрение решающего.

Вывести ответ в выходной файл.

2.7. Минимальное остовное дерево

Дан граф, как набор вершин (общее число — F) и ребер (пар вершин) в файле.

Построить минимальное остовное дерево графа. Структура хранения графа — на усмотрение решающего.

Вывести ответ в выходной файл.

2.8. Эволюция слов

В файле задано слово W длины M из алфавита $\{0, 1, \ldots, N\}.N < M$. Также в файле задана функция $F : \{0, 1, \ldots, N\}^M \to \{0, 1, \ldots, N\}^M$, в виде слова (это слово – результат применения функции к набору $0, 1, ldots, N$). Для произвольного j построить $F^j(W)$. Проверить, нет ли в получившемся слове подряд идущих букв алфавита. Вывести ответ в выходной файл.

2.9. X-O на доске $N \times N$

Задано число $N > 10$. Реализовать игру "крестики-нолики". Предполагается, что пользователь играет крестиками и может подавать на вход программы ходы с помощью задания двух целых чисел — координат крестика в матрице $N \times N$. Выигравшим считается тот, кто первым построил ряд из 5 крестиков (или ноликов).

Консольный интерфейс: вывод на экран текущей позиции с виде

```
   |   a   |   b   |   c   |   d   |   e   |   f   |
-  +  -  +  -  +  -  +  -  +  -  +  -  +  -  |
1  |     |     |     |     |     |     |     |     |
-  +  -  +  -  +  -  +  -  +  -  +  -  +  -  |
2  |     |     |  X  |  O  |  O  |     |     |     |
-  +  -  +  -  +  -  +  -  +  -  +  -  +  -  |
3  |     |     |     |  X  |  X  |  O  |     |     |
-  -  -  -  -  -  -  -  -  -  -  -  -  -  -,
```

запрос с клавиатуры следующего хода человека.

Оптимизация алгоритма: функция оценки позиции, сокращение перебора таким образом, чтобы обеспечить оценку позиции на 3 хода вперед (2 хода программы и 1 ход человека).

Усложнение задачи: реализовать игру на бесконечной доске (структуру данных для хранения позиции, обеспечивающую быструю оценку позиции).

2.10. Задача о раскройке

Дан прямоугольник $P \times S$ (длины сторон — P и S задаются на входе программы пользователем). Задано 20 фиксированного одинакового размера многоугольников вида "Г". Построить рациональную раскройку прямоугольника на многоугольники так, чтобы коэффициент

v=(Площадь остатка прямоугольника после вырезания)/$(P \times S)$ был минимальным из всех возможных случаев.

Вывести ответ в выходной файл.

2.11. Нахождение подслов

Даны два слова S и Q, причем длина слова S намного превосходит длину слова Q. Требуется найти все вхождения Q в S. Программа должна считать из входного файла слова S и Q и напечатать ответ в выходной файл.

2.12. Быстрое умножение двоичных чисел

Даны два n-битовых числа x и y. Если n не степень двойки, то надо эти числа доопределить слева нулями. Требуется реализовать алгоритм быстрого умножения двоичных чисел.

Алгоритм быстрого умножения состоит в следующем:

1. представляем каждое число x и y в виде $x = x_1x_2$ и $y = y_1y_2$, где x_1, x_2, y_1, y_2 — $n/2$-битовые числа;

2. вычисляем x_1y_1, x_2y_2,$(x_1 + x_2)(y_1 + y_2)$;

3. вычисляем xy по формуле: $xy = (x_1 \times 2^{(n-1)} + x_2)(y_1 \times 2^{(n-1)} + y_2) = x_1y_1 \times 2^n + (x_1y_2 + x_2y_1) \times 2^{(n-1)} + x_2y_2$, где $x_1y_2 + x_2y_1 = (x_1 + x_2)(y_1 + y_2) - x_1y_1 - x_2y_2$.

Программа должна считать из входного файла два n-битовым числа и напечатать в выходной файл результат их умножения и количество операций.

2.13. Быстрое умножение матриц

Даны две матрицы A и B размера $n \times n$. Если n не степень двойки, то надо доопределить их нулями, например, начиная с верхнего левого угла. Требуется реализовать алгоритм быстрого умножения матриц.

Алгоритм быстрого умножения состоит в следующем:

1. представляем матрицы A и B в виде: $A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}$ $B = \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}$ где A_{ij}, B_{ij} — матрицы размера $(n/2) \times (n/2)$;

2. вычисляем
   $p_1 = (A_{11} + A_{22})(B_{11} + B_{22})$
   $p_2 = (A_{21} + A_{22})B_{11}$
   $p_3 = A_{11}(B_{12} - B_{22})$
   $p_4 = A_{22}(B_{21} - B_{11})$
   $p_5 = (A_{11} + A_{12})B_{22}$
   $p_6 = (A_{21} - A_{11})(B_{11} + B_{12})$
   $p_7 = (A_{12} - A_{22})(B_{21} + B_{22})$

3. вычисляем AB по формуле $AB = \begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix}$, где
   $C_{11} = p_1 + p_4 - p_5 + p_7$,
   $C_{12} = p_3 + p_5$,
   $C_{21} = p_2 + p_4$,
   $C_{22} = p_1 + p_3 - p_2 + p_6$.

Программа должна считать из входного файла две матрицам размера nxn и напечатать в выходной файл результат их умножения и количество операций.

2.14. Проверка однозначности декодирования

Каждой букве a_i из алфавита $A = \{a_1, \ldots, a_r\}$ ставится в соответствие некоторое слово B_i из алфавита $B = \{b_1, \ldots, b_q\}$. Для каждого кода B_i рассмотрим все его нетривиальные разложения $B_i = q_1 B_{j_1} \ldots B_{j_k} q_2$.

Обозначим через V множество, содержащее пустое слово E и все слова q, встречающиеся в нетривиальном разложениях как в виде начал, так и в виде окончаний. Построим помеченный ориентированный граф G по следующим правилам. Множеством вершин графа G является V. Проводим дугу из вершины q_1 в вершину q_2, если и только если в некотором нетривиальном разложении q_1 является началом, а q_2 — концом. При этом дуга (q_1, q_2) помечается словом $B_{j_1} \ldots B_{j_k}$.

Схема кодирования не обладает свойством однозначности декодирования тогда и только тогда, когда граф G содержит контур, проходящий через вершину E.

Программа должна считать из входного файла кодовые слова B_i и напечатать в выходной файл либо 0, если граф G содержит контур, проходящий через вершину E (схема не обладает свойством однозначности декодирования), либо 1 в противном случае (схема обладает свойством однозначности декодирования).

2.15. Задача о куче камней

Задано множество положительных чисел w_i, $i = 1, \ldots, N$, и l. Разбиваем множество $M = \{i | i = 1, \ldots, N\}$ на k подмножеств M_i, $i = 1, \ldots, k$, где $k < N$. Вес подмножества $W(M_i)$ определяется как сумма всех w_j из M_i. Вводится ограничение $\max(W(M_i)) < l$.

Требуется найти разбиение, для которого k минимально.

Программа должна считать из входного файла сначала число l, а затем последовательно числа $w_1, \ldots, w_N$, и напечатать в выходной файл для каждого $i = 1, \ldots, k$ множество $\{w_j | j \in M_i\}$, начиная каждое множество с новой строки.

2.16. Нахождение ориентированного цикла

Дан ориентированный граф. Требуется построить в нем ориентированный цикл наибольшей длины, если он существует.

Программа должна считать из входного файла таблицу смежности графа и напечатать в выходной файл либо последовательность вершин, образующих цикл, либо -1, если ориентированных циклов нет.

2.17. Раскраска графа

Дан граф без петель и кратных ребер и натуральное число l. Требуется раскрасить вершины этого графа в l красок. Можно использовать любой алгоритм, например, ближайшего соседа.

Программа должна считать из входного файла таблицу смежности графа и напечатать в выходной файл либо последовательность вершин, окрашенных в каждый из цветов, либо написать -1, если данный граф в l цветов не раскрашивается.

2.18. Нахождение кратчайшего пути

Дан граф без петель и кратных ребер и выделенная вершина s. Ребрам этого графа приписаны положительные числа. Требуется найти кратчайшие расстояния

от данной вершины s до остальных вершин этого графа. Можно использовать любой алгоритм, например, алгоритм Дейкстры.

Программа должна считать из входного файла таблицу смежности графа, в которой вместо единиц стоят веса, приписанные ребрам, и напечатать в выходной файл вектор кратчайших расстояний от вершины s до остальных вершин графа, а также с новой строки вектора предшественников на кратчайших путях, т.е номера тех вершин, которые встречаются в этих путях.

2.19. Распределение последовательностей букв в тексте

Во входном файле записан текст. Пусть A — множество всех непробельных символов этого текста. Словом текста будем называть всякую последовательность символов из алфавита A (сам текст, при этом, разбивается пробельными символами на слова). Пусть L — длина длиннейшего слова текста. В выходной файл последовательно для каждого натурального n не превосходящего L следует записать все слова длины n (в алфавите A), упорядоченные в порядке убывания частоты встречаемости в качестве подслова в каком-то слове текста.

2.20. Наибольшее общее подслово

Входной файл содержит два слова, разделённых пробелом. В выходной файл следует записать их общее подслово максимальной длины (в частности, ничего не записать, если единственное общее подслово — пустое слово).

2.21. Задача о растекании жидкости

Имеется клетчатая область размера $n \times k$. В начальный момент в каждой клетке имеется некоторое количество "жидкости". Процесс "растекания жидкости" происходит в дискретном времени в соответствии с правилом: для каждой ячейки Я области

1. если ячейка Я — правая нижняя ячейка области, то жидкость, находившаяся в ней, остаётся на месте;

2. иначе если ячейка Я — крайняя правая ячейка области, то вся находившаяся в ней жидкость перетекает в соседнюю ячейку снизу;

3. иначе если ячейка Я — крайняя нижняя ячейка области, то вся находившаяся в ней жидкость перетекает в соседнюю ячейку справа;

4. иначе q частей жидкости из ячейки перетекает в соседнюю ячейку справа, а оставшиеся $(1 - q)$ частей — в соседнюю ячейку снизу.

Требуется смоделировать этот процесс.

Входной файл содержит матрицу, задающую исходное распределение жидкости (n строк, в каждой строке k чисел, разделённых пробелом) и число q. В выходной файл следует последовательно записать все состояния системы, возникающие в ходе процесса растекания жидкости, начиная с начального распределения вплоть до состояния стабилизации (когда вся жидкость перетечёт в правый нижний угол).

2.22. Выпуклая оболочка

Дано конечное множество $M = \{(x_1, y_1), \ldots, (x_n, y_n)\}$ точек на плоскости. Требуется построить их выпуклую оболочку. Можно воспользоваться, например, алгоритмом Грехема:

1. Пусть p_0 — точка с наименьшей ординатой среди точек из M, а $p_1, \ldots, p_n$ — остальные точки множества M, отсортированные в порядке возрастания полярного угла относительно p_0.

2. Заносим в стек точки p_0, p_1, p_2.

3. Последовательно перебирая точки $p_3, \ldots, p_n$, в зависимости от направления движения к текущей точке (левый или правый поворот) либо добавляем в стек текущую вершину, либо вначале выталкиваем из стека одну вершину, а затем добавляем текущую.

Входной файл состоит из множества строк вида $< x_k, y_k >$, разделённых пробелами. В выходной файл следует вывести координаты точек выпуклой оболочки, начиная с p_0 в порядке обхода против часовой стрелки.

2.23. Компоненты связности ориентированного графа

Дан ориентированный граф (без кратных рёбер). Множество M вершин графа называется компонентой связности, если для любых v_1, v_2 из M существует ориентированный путь в графе из v_1 в v_2, проходящий через вершины из множества M. Компонента связности M называется максимальной, если для любой вершины v графа, не лежащей в M, множество M, объединенное с v, не является компонентой связности. Требуется найти все максимальные компоненты связности графа.

Матрица инцидентности графа, содержащего n вершин, — это матрица размера $n \times n$, в клетке (i, j) которой стоит 1, если граф содержит ориентированное ребро из i-ой вершины в j-ую, и 0 в противном случае.

Входной файл содержит матрицу инцидентности графа (n строк, в каждой строке n чисел из множества $\{0, 1\}$, разделённых пробелом).В выходной файл

следует вывести матрицы инцидентности максимальных компонент связности (рассматриваемых как граф с n вершинами, некоторые из которых, возможно, не соединены ребром).