# Глава 2. Выбор и обоснование алгоритма оптимизации

## 2.1 Содержательная постановка задачи

Создать программную систему, выполняющую следующие операции: a) анализ исходного кода и локализация участков, на которых осуществляется последовательный перебор значений контейнеров std::list либо std::vector и сравнение каждого элемента с ключевым значением.

б)генерировать рекомендации по включению в исходный код вспомогательных индексированных контейнеров данных, обеспечивающих быстрый поиск элементов, причем суммарный размер этих контейнеров не должен превышать заданной верхней границы дополнительного объема ОП, выделенного на оптимизацию.

### 2.1.1Исходные данные к задаче.

1. Исходный код пользователя;
2. Верхняя граница объёма оперативной памяти, выделенной на оптимизацию.

## 2.2 Формальная постановка задачи

### 2.2.1 Обозначения.

М –количество контейнер.

n*i* – количество элементов i-го контейнера данных;

v*i*– размер одного элемента i-го контейнера данных;

P*i* – среднее количество операций поиска в i-м контейнере данных;

V– верхняя граница дополнительного объема ОП, выделенной на оптимизацию.

z*i* – булева переменная равная единице, если для оптимизации

b*i* – элемент массива булевых констант (входные данные). b*i* равна единице, если исходным контейнером хранения данных i-го массива являлся std::vector и нулю, в случае, когда исходным контейнером является std::list.

–вспомогательная функция, описывающая среднее время поиска элемента в i-м массиве с использованием вспомогательного контейнера std::map.

– скорость формирования контейнера std::map.

– средняя скорость сравнения двух произвольных элементов i-го массива.

–среднее время выборки и сравнения двух элементов i-го массива (равна );

–вспомогательная функция, описывающая среднее время поиска элемента в i-м массиве, реализованного в виде контейнера std::vector.

–вспомогательная функция, описывающая среднее время поиска элемента в i-м массиве, реализованного в виде контейнера std::list.

–коэффициент пропорциональности > 1, описывающий характер превышения времени выборки данных в контейнере std::list по сравнению с std::vector . Более высокие временные затраты на перемещение между элементами списка объясняются тем что обращение к соседним элементам осуществляется посредством вспомогательных указателей.

### 2.2.2 Формальная постановка задачи

Сформулируем оптимизационную задачу, минимизирующую суммарное время поиска в контейнерах std::vector и std::list с использованием вспомогательного индексированного контейнера std::map в виде модели (1):

(1)

Следует отметить, что выигрыш в скорости поиска дается не только ценой дополнительной памяти, но также дополнительным процессорным временем, который тратится на создание индексированного массива std::map — соответствует второму слагаемому вспомогательной функции :

(2)

Пропорциональный характер времени формирования контейнера std::map относительно размера массива подтвержден экспериментально:

Таблица 2.1. *Результаты тестовых замеров времени создания контейнера std::map*

|  |  |
| --- | --- |
| Количество элементов типа double | Время формирования std::map  (миллисекунды) |
| 1000000 | 609 |
| 2000000 | 1250 |
| 3000000 | 1890 |
| 4000000 | 2563 |
| 5000000 | 3219 |
| 6000000 | 3890 |
| 7000000 | 4562 |
| 8000000 | 5234 |
| 9000000 | 5922 |
| 10000000 | 6610 |

Рис.2.1 Зависимость времени формирования контейнера std::map от размера массива.

## 2.3 Алгоритм поиска решения

Для того чтобы обеспечить возможность поиска решения задачи (1) при больших размерностях входных данных мной был выбран метод Монте-Карло, приведенный ниже (Алгоритм 2.1):

*Алгоритм 2.1.*

1. Ввод числа итераций N.
2. i=0.
3. Рекорду **R** присваиваем значение «бесконечность».
4. С помощью генератора случайных чисел генерируются булевы значения (0 либо 1) для каждого элемента **.**
5. Если условие не выполняется, то переход к шагу 9, иначе к следующему шагу.
6. Вычисляем значение целевой функции **,** с помощью выражения, указанного в модели (1).
7. Если F>=R, то переход к шагу 9, иначе к следующему шагу
8. R = F, сохраняем элементы массива {z} в массив {}.
9. i = i+1.
10. Если i<=N, то переход к шагу 4, в противном случае к следующему шагу.
11. Решение найдено. R содержит текущее значение целевой функции, а массив {} — значения элементов zi, соответствующие рекорду.

## 2.4 Пример решения задачи вручную

Выполним поиск наилучшей стратегии оптимизации для следующего исходного кода пользователя (Листинг 2.1):

Листинг 2.1.

#include <iostream>

#include <algorithm>

#include <vector>

#include <iterator>

int main()

{

int n1 = 3;

int n2 = 5;

[std::vector](http://en.cppreference.com/w/cpp/container/vector)<int>x1;

[std::vector](http://en.cppreference.com/w/cpp/container/vector)<double>x2;

[std::list](http://en.cppreference.com/w/cpp/container/vector)<int>x3;

**//созданиеизаполнениемассивов**

for (int i=0; i<**1000**; i++) x1.push\_back(i);

for (int i=0; i<**700**; i++) x2.push\_back((double)i);

for (int i=0; i<**500**; i++) x3.push\_back(i);

 //поисквмассивах

for (int i=0; i<**3000**; i++)

{

auto result1 = std::find([std::begin](http://en.cppreference.com/w/cpp/iterator/begin)(x1), [std::end](http://en.cppreference.com/w/cpp/iterator/end)(x1),

(int)(1000.0\*(double)rand()/(double)RAND\_MAX) );

std::cout<< (result1!=x1.end()?"Значение найдено!":

"Значение не найдено!";

}

for (int i=0; i<**200**; i++)

{

auto result2 = std::find([std::begin](http://en.cppreference.com/w/cpp/iterator/begin)(x2), [std::end](http://en.cppreference.com/w/cpp/iterator/end)(x2),

700.0\*(double)rand()/(double)RAND\_MAX );

std::cout<< (result2=x2.end()?"Значение найдено!":

"Значение не найдено!";

auto result3 = std::find([std::begin](http://en.cppreference.com/w/cpp/iterator/begin)(x3), [std::end](http://en.cppreference.com/w/cpp/iterator/end)(x3),

(int)(500.0\*(double)rand()/(double)RAND\_MAX) );

std::cout<< (result3=x3.end()?"Значение найдено!":

"Значение ненайдено!";

}

}

На основании Листинга1 выделим часть исходных данных и поместим их в Таблицу 2.2.

Таблица 2.2. Входные данные, полученные на основании анализа исходного кода.

|  |  |  |  |
| --- | --- | --- | --- |
| **M** | 3 | | |
| **{n}** | 1000 | 700 | 500 |
| **{v}** | 4 | 8 | 4 |
| **{P}** | 3000 | 200 | 200 |
| **{b}** | 1 | 1 | 0 |

Примем в качестве верхней границы доступного для оптимизации объема ОП значение **7000** байт, а значения скоростей сравнения и создания контейнера 1.00E+13 байт/сек и 1.00E+13 байт/сек соответственно. Рассчитаем вспомогательные переменные ti по формуле .

Таблица 2.3. Дополнительные входные данные.

|  |  |  |  |
| --- | --- | --- | --- |
| **s\_compare** | 1.00E+13 байт/сек | | |
| **s\_map** | 1.00E+10 байт/сек | | |
| **k** | 15 | | |
| **{t}** | 4E-13 | 8E-13 | 4E-13 |
| **V** | **7000 байт** | | |

Выполним расчет значений целевой функции для всех вариантов стратегий (Таблица 2.4).

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **z1** | **z2** | **z3** | **Tmap** | **Tvector** | **Tlist** | **Ограничение** | **F** |
| 0 | 0 | 0 | 0 | 0.000000656 | 0.0000003 | 0 | 0.000000956 |
| 0 | 0 | 1 | 2.00717E-07 | 0.000000656 | 0 | 2000 | 8.56717E-07 |
| 0 | 1 | 0 | 5.61512E-07 | 0.0000006 | 0.0000003 | 5600 | 1.46151E-06 |
| 0 | 1 | 1 | 7.62229E-07 | 0.0000006 | 0 | 7600 | +Infiniy |
| 1 | 0 | 0 | 4.11959E-07 | 0.000000056 | 0.0000003 | 4000 | 7.67959E-07 |
| 1 | 0 | 1 | 6.12676E-07 | 0.000000056 | 0 | 6000 | 6.68676E-07 |
| 1 | 1 | 0 | 9.73471E-07 | 0 | 0.0000003 | 9600 | +Infiniy |
| 1 | 1 | 1 | 1.17419E-06 | 0 | 0 | 11600 | +Infiniy |
| **F min** | | | | | | | **6.68676E-07** |

Наилучшей стратегией является использование вспомогательных контейнеров std::map для 1-го и 3-го массивов, что обеспечит минимальное суммарное время поиска элементов в массивах пользовательского алгоритма, приведенного в Листинге 2.1.

Решение данного примера было получено с помощью программы Excel. Документ, в котором было получено решение, находится в облаке и получить доступ к нему можно по следующей ссылке: <https://cloud.mail.ru/public/vBsp/csP9mA4WT>.

В следующей главе приведено описание программной реализации описанных подходов.