Московский авиационный институт
(национальный исследовательский университет)

Факультет информационных технологий и прикладной математики
Кафедра вычислительной математики и программирования

Курсовой проект по курсу «Дискретный анализ»

Аналог утилиты diff

Студент: А. Е. Аксенов
Преподаватель: С. А. Сорокин
Группа: М8О-306Б-19
Дата: 18.12.2021
Оценка:
Подпись:

Москва, 2021

1 Постановка задачи

Задание

Разработать на языке C++ программу, способную выводить наименьшее количество действий, позволяющих преобразовать один файл в другой (утилита diff).

Метод решения

Самым простом алгоритмом, выводящим доступный для человска формат, является алгоритм **Юджина Майерса**, поверхностно описать который можно так:

Проблему нахождения наименьшего изменения между двумя файлами A и B размеров N и M одновременно можно рассматривать как проблему нахождения пути на графе размером N x M между вершинами (0,0) и (N,M), причем на этом графе:

- Горизонтальная грань (x,y) $\rightarrow$ (x+1,y) означает удаление в файле A строки x (в файлах индексация строк с счипается единицы);
- Вертикальная грань (x,y) $\rightarrow$ (x,y+1) означает добавление из файла B строки y;
- Диагональная грань (x,y) $\rightarrow$ (x+1,y+1) означает, что строки x и y в файлах A и B соответственно равны.

Если считать, что у горизонтальных и вертикальных граней вес равен единице, а у диагональных - нулю, то задача решаема алгоритмом Дейкстры, но он не всегда будет выводить читабельный diff-вывод.

Алгоритм Майерса пользуется тем, что путь в графе всегда идет из верхнего левого угла в нижний правый угол.

Введем понятие уровня и D-пути:

- Уровень k на графе - это номер диагонали по сравнению с диагональю, на которой лежит точка (0,0). Соответственно, диагональ с точкой (0,0) имеет уровень k = 0, диагональ над ней - k = 1, диагональ под ней - k = -1 и т.д. Уровень можно высчитать по формуле k = x - y.
- D-путь - (D - 1)-путь, после которого идет или горизонтальная грань, или вертикальная грань, причем 0-путь - это путь, состоящий только из этой грани. После этой грани существует возможно пустая последовательность диагоналей.

Алгоритм имеет максимум N+M повторений, причем на d-ом повторении идет удлинение всех d-путей, лежащих на уровнях -d, -d+2,..., d-2, d. Удлинение на уровне k идет за удлинения пути либо на k-1 уровне или на k+1, в зависимости от того, какой длиннее. Приоритет бкдет отдан k-1 уровню когда это возможно, т.к. он удлиняется за счёт горизонтальной грани (т.е. удаления из A). Первый d-путь, достигший точки (N,M), считается оптимальным.

2 Общие сведения о программе

1. *main.cpp* - принимает из командной строки названия файлов и считывает их построчно в вектор, после чего вызывает функцию, строющую diff между этими векторами.

2. $diff.h$ - содержит шаблонную функцию. строющую diff в соотвествии с алгоритмом Юджина Майерса. Функция принимает любой тип, имеющий в себе метод $size$ и оператор квадратных скобок.

3 Исходный код

structs.h

```
#ifndef STRUCTS_H
#define STRUCTS_H

#include <cstdint>

struct TAction {
    enum {ADD, DEL, KEEP} type;
    int64_t x, y;
};

#endif
```

diff.h

```
#ifndef DIFF_H
#define DIFF_H

#include <cstdint>
#include <vector>

#include "structs.h"

std::vector<TAction> build_trace(
        const std::vector< std::vector<int64_t> >& trace,
        int64_t x, int64_t y, size_t total);

template<typename T>
std::vector<TAction>
find_diff(const T& data1, const T& data2) {
    const size_t len1 = data1.size();
    const size_t len2 = data2.size();
    const size_t total = len1 + len2;

    std::vector<int64_t> extensions(2 * total + 1);
    std::vector< std::vector<int64_t> > trace;

    extensions[1 + total] = 0;
    for (int64_t path = 0; path <= total; ++path) {

        trace.push_back(extensions);

        for (int64_t diag = -path; diag <= path; diag += 2) {
            int64_t x, y;
            bool go_down = (diag == -path
                            || (diag != path
                                && extensions[diag - 1 + total] < extensions[diag + 1 +
```

```
                                total]));
33
34              if (go_down) {
35                  x = extensions[diag + 1 + total];
36              } else {
37                  x = extensions[diag - 1 + total] + 1;
38              }
39
40              y = x - diag;
41
42              while (x < len1 && y < len2 && data1[x] == data2[y]) {
43                  ++x;
44                  ++y;
45              }
46
47              extensions[diag + total] = x;
48              if (x >= len1 && y >= len2) {
49                  return build_trace(trace, len1, len2, total);
50              }
51          }
52      }
53
54      return std::vector<TAction> ();
55  }
56
57  #endif
```

linear_diff.h

```
1   #ifndef LINEAR_DIFF_H
2   #define LINEAR_DIFF_H
3
4   #include <vector>
5   #include <utility>
6   #include <cmath>
7
8   #include "structs.h"
9
10  namespace {
11      struct Box {
12          Box(int64_t left, int64_t top,
13                  int64_t right, int64_t bottom) {
14              this->left = left;
15              this->top = top;
16              this->right = right;
17              this->bottom = bottom;
18
19              width = right - left;
20              height = bottom - top;
21              size = width + height;
```

```
22            delta = width - height;
23        }
24
25        int64_t left;
26        int64_t top;
27        int64_t right;
28        int64_t bottom;
29        int64_t width;
30        int64_t height;
31        int64_t size;
32        int64_t delta;
33    };
34
35    using Point = std::pair<int64_t, int64_t>;
36    using Snake = std::pair<Point, Point>;
37
38    size_t wrap_index(int64_t i, int64_t size) {
39        while (i >= size) {
40            i -= size;
41        }
42        while (i < 0) {
43            i += size;
44        }
45        return i;
46    }
47
48    template<typename T>
49    std::pair<Snake, bool> forwards(
50                const Box& box, std::vector<int64_t>& forw_snakes,
51                std::vector<int64_t>& back_snakes, int64_t depth,
52                int64_t total, const T& a, const T& b) {
53        for (int64_t k = depth; k >= -depth; k -= 2) {
54            const int64_t size = 2 * total + 1;
55
56            int64_t c = k - box.delta;
57            int64_t x;
58            int64_t px;
59            int64_t y;
60            int64_t py;
61
62            bool go_down = ( k == -depth ||
63                   (k != -depth && forw_snakes[wrap_index(k - 1, size)] < forw_snakes[
                       wrap_index(k + 1, size)]) );
64            if (go_down) {
65                px = x = forw_snakes[wrap_index(k + 1, size)];
66            } else {
67                px = forw_snakes[wrap_index(k - 1, size)];
68                x = px + 1;
69            }
```

```
70
71          y = box.top + (x - box.left) - k;
72          py = (depth == 0 || x != px) ? y : y - 1;
73
74          while (x < box.right && y < box.bottom && a[x] == b[y]) {
75              ++x;
76              ++y;
77          }
78
79          forw_snakes[wrap_index(k, size)] = x;
80
81          bool delta_odd = box.delta % 2 == 1;
82          bool c_between = c >= -depth + 1 && c <= depth - 1;
83          if (delta_odd && c_between && y >= back_snakes[wrap_index(c, size)]){
84              if (depth == 1) {
85                  while (x > px && y > py) {
86                      --x;
87                      --y;
88                  }
89              }
90              Snake snake = std::make_pair(
91                              std::make_pair(px, py),
92                              std::make_pair(x, y)
93                          );
94              return std::make_pair(snake, true);
95          }
96      }
97
98      return std::make_pair(Snake(), false);
99  }
100
101 template<typename T>
102 std::pair<Snake, bool> backwards(
103             const Box& box, std::vector<int64_t>& forw_snakes,
104             std::vector<int64_t>& back_snakes, int64_t depth,
105             int64_t total, const T& a, const T& b) {
106     for (int64_t c = depth; c >= -depth; c -= 2) {
107         const int64_t size = 2 * total + 1;
108
109         int64_t k = c + box.delta;
110         int64_t x;
111         int64_t px;
112         int64_t y;
113         int64_t py;
114
115         bool go_up = ( c == -depth ||
116                 (c != depth && back_snakes[wrap_index(c - 1, size)] > back_snakes[
                        wrap_index(c + 1, size)]) );
117         if (go_up) {
```

```
            py = y = back_snakes[wrap_index(c + 1, size)];
        } else {
            py = back_snakes[wrap_index(c - 1, size)];
            y = py - 1;
        }

        x = box.left + (y - box.top) + k;
        px = (depth == 0 || y != py) ? x : x + 1;

        while (x > box.left && y > box.top && a[x - 1] == b[y - 1]) {
            --x;
            --y;
        }

        back_snakes[wrap_index(c, size)] = y;

        bool delta_even = box.delta % 2 == 0;
        bool k_between = k >= -depth && k <= depth;
        if (delta_even && k_between && x <= forw_snakes[wrap_index(k, size)]) {
            Snake snake = std::make_pair(
                            std::make_pair(x, y),
                            std::make_pair(px, py)
                          );
            return std::make_pair(snake, true);
        }
    }

    return std::make_pair(Snake(), false);
}

template<typename T>
std::pair<Snake, bool> midpoint(const Box& box, const T& a, const T& b) {
    if (box.size == 0) {
        return std::make_pair(Snake(), false);
    }

    int64_t max_d = ceil(box.size / 2.);
    std::vector<int64_t> forw_snakes(2 * max_d + 1);
    forw_snakes[1] = box.left;
    std::vector<int64_t> back_snakes(2 * max_d + 1);
    back_snakes[1] = box.bottom;

    std::pair<Snake, bool> snake;

    for (int64_t depth = 0; depth <= max_d; ++depth) {
        snake = forwards(box, forw_snakes, back_snakes, depth, max_d, a, b);
        if (snake.second) {
            return snake;
        }
```

```
167             snake = backwards(box, forw_snakes, back_snakes, depth, max_d, a, b);
168             if (snake.second) {
169                 return snake;
170             }
171         }
172 
173         snake.second = false;
174         return snake;
175     }
176 
177     template<typename T>
178     std::vector<Point> find_path(
179             int64_t left, int64_t top,
180             int64_t right, int64_t bottom,
181             const T& a, const T& b) {
182         Box box(left, top, right, bottom);
183         std::pair<Snake, bool> snake = midpoint(box, a, b);
184 
185         std::vector<Point> result;
186 
187         if (!snake.second) {
188             return result;
189         }
190 
191         Point start = snake.first.first;
192         Point end = snake.first.second;
193 
194         std::vector<Point> head = find_path(box.left, box.top, start.first, start.
                second, a, b);
195         std::vector<Point> tail = find_path(end.first, end.second, box.right, box.
                bottom, a, b);
196 
197         if (head.empty()) {
198             result.push_back(start);
199         } else {
200             result.insert(result.end(), head.begin(), head.end());
201         }
202 
203         if (tail.empty()) {
204             result.push_back(end);
205         } else {
206             result.insert(result.end(), tail.begin(), tail.end());
207         }
208 
209         return result;
210     }
211 
212     template<typename T>
213     std::vector<TAction> find_diff_linear(const T& data1, const T& data2) {
```

```
214         std::vector<Point> path = find_path(0, 0, data1.size(), data2.size(), data1,
                data2);
215         std::vector<TAction> diff_actions;
216         int64_t x = 0;
217         int64_t y = 0;
218 
219         for (size_t i = 0; i < path.size(); ++i) {
220             const Point& p = path[i];
221 
222             while (x < p.first && y < p.second) {
223                 diff_actions.push_back({TAction::KEEP, x, y});
224                 ++x;
225                 ++y;
226             }
227 
228             if (p.first - x < p.second - y) {
229                 diff_actions.push_back({TAction::ADD, x, y});
230                 ++y;
231             } else if (p.first - x > p.second - y) {
232                 diff_actions.push_back({TAction::DEL, x, y});
233                 ++x;
234             }
235         }
236 
237         return diff_actions;
238     }
239 }
240 
241 #endif
```

diff.cpp

```
1  #include "diff.h"
2  
3  std::vector<TAction> build_trace(
4         const std::vector< std::vector<int64_t> >& trace,
5         int64_t x, int64_t y, size_t total) {
6     std::vector<TAction> diff_actions;
7  
8     for (int64_t d = trace.size() - 1; d >= 0; --d) {
9         const std::vector<int64_t>& layer = trace[d];
10 
11        int64_t k = x - y;
12        int64_t prev_k;
13 
14        bool went_down = (k == -d || (k != d && layer[k - 1 + total] < layer[k + 1 +
              total]));
15        if (went_down) {
16            prev_k = k + 1;
17        } else {
```

```
            prev_k = k - 1;
        }

        int64_t prev_x = layer[prev_k + total];
        int64_t prev_y = prev_x - prev_k;

        while (x > prev_x && y > prev_y) {
            --x;
            --y;
            diff_actions.push_back({TAction::KEEP, x, y});
        }

        if (d == 0) {
            continue;
        }

        if (x == prev_x) {
            y = prev_y;
            diff_actions.push_back({TAction::ADD, x, y});
        } else if (y == prev_y) {
            x = prev_x;
            diff_actions.push_back({TAction::DEL, x, y});
        }
    }

    return std::vector<TAction>(diff_actions.rbegin(), diff_actions.rend());
}
```

main.cpp

```
#include <iostream>
#include <fstream>
#include <string>
#include <vector>

#include "diff.h"

std::vector<std::string> read_file(char* filename) {
    std::ifstream is(filename);
    std::string line;
    std::vector<std::string> text;

    while(std::getline(is, line)) {
        text.push_back(line);
    }

    return text;
}

int main(int argc, char* argv[]) {
```

```
21      if (argc < 3) {
22          std::cout
23              << "Usage: "
24              << argv[0]
25              << " FILE1 FILE2"
26              << std::endl;
27          return -1;
28      }
29
30      std::vector<std::string> text1 = read_file(argv[1]);
31      std::vector<std::string> text2 = read_file(argv[2]);
32
33      std::vector<TAction> actions( find_diff(text1, text2) );
34
35      for (const auto& act : actions) {
36          switch (act.type) {
37            case TAction::ADD: {
38              std::cout << "+ ";
39              std::cout << text2[act.y] << std::endl;
40              break;
41            }
42            case TAction::DEL: {
43              std::cout << "- ";
44              std::cout << text1[act.x] << std::endl;
45              break;
46            }
47            case TAction::KEEP: {
48              std::cout << " ";
49              std::cout << text1[act.x] << std::endl;
50              break;
51            }
52          }
53      }
54  }
```

4 Пример работы и тесты

Пусть имеется 2 файла:

f1.txt

```
Эта часть документа
оставалась неизменной
от версии к версии. Если
в ней нет изменений,она
не должна отображаться.
Иначе это не способствует
выводу оптимального
объёма произведённых
изменений.

Этот абзац содержит
устаревший текст.
Он будет удалён
в ближайшем будущем.

В этом документе
необходима провести
проверку правописания.
С другой стороны,ошибка
в слове -не конец света.
Остальная часть абзаца
не требует изменений.
Новый текст можно
добавлять в конец документа.
```

f2.txt

```
Это важное замечание!
Поэтому оно должно
быть расположено
в начале этого
документа!

Эта часть документа
оставалась неизменной
от версии к версии. Если
в ней нет изменений,она
```

```
не должна отображаться.
Иначе это не способствует
выводу оптимального
объёма информации.

В этом документе
необходимо провести
проверку правописания.
С другой стороны,ошибка
в слове -не конец света.
Остальная часть абзаца
не требует изменений.
Новый текст можно
добавлять в конец документа.

Этот абзац содержит
важные дополнения
для данного документа.
```

Запустим для них утилиту:

```
fallfire13@DESKTOP-M7F3IHA:~/DA_kp$ time ./a.out f1.txt f2.txt
+ Это важное замечание!
+ Поэтому оно должно
+ быть расположено
+ в начале этого
+ документа!
+
Эта часть документа
оставалась неизменной
от версии к версии. Если
в ней нет изменений,она
не должна отображаться.
Иначе это не способствует
выводу оптимального
-объёма произведённых
-изменений.
+ объёма информации.

-Этот абзац содержит
-устаревший текст.
```

```
-Он будет удалён
-в ближайшем будущем.
-
В этом документе
-необходима провести
+ необходимо провести
проверку правописания.
С другой стороны,ошибка
в слове -не конец света.
Остальная часть абзаца
не требует изменений.
Новый текст можно
добавлять в конец документа.
+
+ Этот абзац содержит
+ важные дополнения
+ для данного документа.

real    0m0.027s
user    0m0.000s
sys     0m0.000s
```

Проверим работу, запустив стандартную утилиту diff:

```
fallfire13@DESKTOP-M7F3IHA:~/DA_kp$ time diff f1.txt f2.txt
0a1,6
>Это важное замечание!
>Поэтому оно должно
>быть расположено
>в начале этого
>документа!
>
8,14c14
<объёма произведённых
<изменений.
<
<Этот абзац содержит
<устаревший текст.
<Он будет удалён
<в ближайшем будущем.
---
```

```
>объёма информации.
17c17
<необходима провести
---
>необходимо провести
24a25,28
>
>Этот абзац содержит
>важные дополнения
>для данного документа.

real    0m0.019s
user    0m0.000s
sys     0m0.000s
```

Как видим, результаты работ совпадают.

Исследование времени выполнения

```
fallfire13@DESKTOP-M7F3IHA:~/DA_kp$ time ./a.out f1.txt f1.txt
Эта часть документа
оставалась неизменной
от версии к версии. Если
в ней нет изменений,она
не должна отображаться.
Иначе это не способствует
выводу оптимального
объёма произведённых
изменений.

Этот абзац содержит
устаревший текст.
Он будет удалён
в ближайшем будущем.

В этом документе
необходима провести
проверку правописания.
С другой стороны,ошибка
в слове -не конец света.
Остальная часть абзаца
не требует изменений.
```

```
Новый текст можно
добавлять в конец документа.

real    0m0.017s
user    0m0.000s
sys     0m0.000s
```

Построение diff на одинаковых файлах быстрое, т.к. алгоритм останавливается за один шаг.

```
fallfire13@DESKTOP-M7F3IHA:~/DA_kp$ tac f1.txt >f3.txt
fallfire13@DESKTOP-M7F3IHA:~/DA_kp$ time ./a.out f1.txt f3.txt
-Эта часть документа
-оставалась неизменной
-от версии к версии. Если
-в ней нет изменений,она
-не должна отображаться.
-Иначе это не способствует
-выводу оптимального
-объёма произведённых
-изменений.
+ добавлять в конец документа.
+ Новый текст можно
+ не требует изменений.
+ Остальная часть абзаца
+ в слове -не конец света.
+ С другой стороны,ошибка
+ проверку правописания.
+ необходима провести
+ В этом документе

-Этот абзац содержит
-устаревший текст.
-Он будет удалён
в ближайшем будущем.
+ Он будет удалён
+ устаревший текст.
+ Этот абзац содержит

-В этом документе
-необходима провести
```

```
-проверку правописания.
-С другой стороны,ошибка
-в слове -не конец света.
-Остальная часть абзаца
-не требует изменений.
-Новый текст можно
-добавлять в конец документа.
+ изменений.
+ объёма произведённых
+ выводу оптимального
+ Иначе это не способствует
+ не должна отображаться.
+ в ней нет изменений,она
+ от версии к версии. Если
+ оставалась неизменной
+ Эта часть документа

real    0m0.076s
user    0m0.000s
sys     0m0.000s
```

Сложность алгоритма очень схожа с $O((N+M)D)$. Особенно это видно на примере с *tac*, где из-за того, что строки файла в обратном порядке, требуется заменить все строки, и глубина $D=N+M$.

5 Выводы

Кроме своего стандартного использования (сравнения файлов), можно diff ещё можно использовать для:

- Построение patch-файлов, Они могут понадобиться для изменения конкретных частей файла без изменения его всего (в отличии от передачи всего файла).
- Минимальная передача данных при синхронизации бинарных файлов за счет проверки их на одинаковость и отправки только отличий (**rsync**).
- Противу интуитивным понятиям, diff можно обобщить до сравнения всего, что можно редактировать, в том числе двух бинарных файлов.
- Более специфично: обнаружение и сравнение мутации ДНК.

Стоит заметить, для большинства вводов существует более чем одна минимальная разность файлов, но одна разность может иметь раскинутые действия по всему файлу, а другая будет иметь сгруппированные в одном месте действия. Далее рассматриваются четыре алгоритма, встроенные в **git diff**:

1. Алгоритм **Юджина Майерса** в линейном пространстве - стандартный применяемый алгоритм, используемый при вызове утилиты. Чаще всего выводит хорошие результаты за быстрое время и малую память, но на пекоторых вводах запинается и выводит сильно смешанный вывод. Другой флаг *minimal* работает на том же алгоритме, но рассматривает больше промежуточных вариантов для вывода чего-то более читабельного. Можно применять, когда нет причин использовать другие алгоритмы.
2. Алгоритм **patience** - алторитм делит файл на секции, используя общие строки, которые не повторяются нигде в самих документах. Деление файлов на общие секции и поиск изменений в самих секциях выводит сгруппированную разницу чаще, чем у Майерса. Пространственная сложность линенная.
3. Алгоритм **histogram** - является модификацией patience, превосходящий по скорости и результатам и Майерса и patience. В отличие от patience, ищет не уникальные элементы, а наименее повторяющиеся. Следует применять при сравнении исходного кода.

Один из неупомянутых алгоритмов (который был введён столько раз,что уже неизвестно, кто первым его придумал): алгоритм Вагнера-Фишера, который считает рстояния между каждым префиксом обоих массивов, что имеет предположительно пространственную и алгоритмическую сложность $O(NM)$.