Университет ИТМО

Факультет программной инженерии и компьютерной техники

Лабораторная работа №1
по “Алгоритмам и структурам данных”
Базовые задачи

Выполнил:
Студент группы P3206
Сорокин А.Н.
Преподаватели:
Косяков М.С.
Тараканов Д.С.

Санкт-Петербург
2024

Задача №A “Агроном-любитель”

Решение за O(n).
Данный алгоритм является алгоритмом линейного поиска. Требуется найти подстроку с максимальной длиной, удовлетворяющую условию “цветок одного вида не повторяется более 2х раз подряд”. Имея левый и правый указатели, нарастить строку с помощью правого указателя, на каждой итерации проверяя условие. Как только условие не выполняется, сравнить полученную длину с длиной уже найденной подстроки, затем “подтянуть” левый указатель так, чтобы из 3х цветков одного вида, идущих подряд только второй и третий оказались в новой строке. Таким образом, мы сможем найти требуемую подстроку, обратившись к каждому элементу не более 1 раза.

Код решения:

```
#include <cstddef>
#include <ios>
#include <iostream>

using namespace std;

int main() {
  ios_base::sync_with_stdio(false);
  cin.tie(NULL);

  size_t n, a, start_pos = 1, end_pos = 0, min_start_pos = 1, max_end_pos =
1,
              count = 0, last = 0;
  cin >> n;

  for (size_t i = 0; i < n; ++i) {
    cin >> a;
    if (a == last) {
      ++count;
    } else {
      count = 1;
```

```
    }
    last = a;
    ++end_pos;
    if (count > 2) {
      if ((max_end_pos - min_start_pos) < (end_pos - start_pos - 1)) {
        min_start_pos = start_pos;
        max_end_pos = end_pos - 1;
      }
      start_pos = end_pos - 1;
      count = 2;
    }
  }
  if ((max_end_pos - min_start_pos) < (end_pos - start_pos)) {
    min_start_pos = start_pos;
    max_end_pos = end_pos;
  }

  cout << min_start_pos << " " << max_end_pos;
  return 0;
}
```

Задача №B “Зоопарк Глеба”

Решение за O(n).
Решением данной задачи является восходящий (bottom-up) парсер.

Идея в следующем. Подходящей парой является пара, в которой один сивмол является заглавным, второй строчным, но не учитывая case, они одинаковы. Программа последовательно просматривает каждый символ, начиная от первого символа (root), и спускается ниже до тех пор, пока не найдет подходящую пару (leaf), затрет ее и поднимется на уровень выше. Таким образом, для правильной последовательности вся строка будет затерта.

Главная задача алгоритма – превратить любую входную строку в пустую.

На примерах входных данных:

ABba → Ø

In	Out
A	A
AB	AB
ABb	A (Bb → “”)
Aa	Ø (Aa → “”) POSSIBLE

ABab → Ø

In	Out
A	A
AB	AB
ABa	ABa
ABab	ABab IMPOSSIBLE

Код решения:

```
#include <cctype>
#include <cstddef>
#include <ios>
#include <iostream>
#include <string>
#include <utility>
#include <vector>
#define START_UPPER 65
#define END_UPPER 90

using namespace std;

int main() {
  ios_base::sync_with_stdio(false);
  cin.tie(NULL);
```

```
vector<pair<char, size_t>> vp;
string s;
size_t j = 1, k = 1;
cin >> s;
size_t n = s.size();
for (size_t i = 0; i < s.size(); ++i) {
  char a = s.at(i);
  if ((a >= START_UPPER) && (a <= END_UPPER)) {
    vp.push_back(make_pair(a, j));
    ++j;
  } else {
    vp.push_back(make_pair(a, k));
    ++k;
  }
}
vector<pair<char, size_t>> stack;

size_t res[n / 2 + 1];
for (pair<char, size_t> p : vp) {
  stack.push_back(p);
  if (stack.size() < 2)
    continue;
  if (stack.back().first != stack.at(stack.size() - 2).first &&
      (tolower(stack.back().first) ==
       tolower(stack.at(stack.size() - 2).first))) {
    if (stack.back().first > stack.at(stack.size() - 2).first) {
      res[stack.at(stack.size() - 2).second] = stack.back().second;
    } else {
      res[stack.back().second] = stack.at(stack.size() - 2).second;
    }
    stack.pop_back();
    stack.pop_back();
  }
}
if (stack.empty()) {
```

```
    cout << "Possible\n";
    for (size_t l = 1; l <= n / 2; ++l) {
      cout << res[l] << " ";
    }
  } else {
    cout << "Impossible";
  }

  return 0;
}
```

Задача №C “Конфигурационный файл”

Решение за O(n).
Сложность реализации заключается не во времени выполнения, а в ограничении по памяти. Обозначим каждый блок входных данных как слой. Тогда необходимо иметь 3 структуры данных:
- структура, хранящая название переменной и все слои, на которых она была инициализована/переназначена - appeared
- структура, хранящая слой и все названия переменных, которые были изменены на этом слое - changed
- структура, позволяющая по номеру слоя и названию переменной однозначно определить значение переменной - buffer
При входе в блок создается пустая карта и кладется в buffer, номер слоя увеличивается. При выходе из блока из buffer удалется верхняя карта, все изменения, сделанные в этом блоке, откатываются (так как любые изменения, сделанные в блоке, из которого мы только что вышли, нам больше не нужны). Любые создания/изменения переменных на конкретном слое отражаются в appeared и changed соответственно.

Код решения:

```
#include <cstdlib>
#include <fstream>
#include <iostream>
#include <unordered_map>
```

```
#include <vector>
using namespace std;

bool checknum(const string s) {
  if (s == "0") {
    return true;
  }
  long long num = atoll(s.c_str());
  return num != 0;
}
int main() {

  ifstream infile("input.txt");

  string s;
  vector<unordered_map<string, long long>> buffer;

  unordered_map<string, vector<size_t>> appeared;
  unordered_map<size_t, vector<string>> changed;

  size_t layer = 0;
  buffer.emplace_back();

  while (infile >> s) {
    if (s == "{") {
      // going to next layer
      ++layer;
      buffer.emplace_back();
      continue;
    }
    if (s == "}") {
      // going to previous layer, cleaning the last layer based on changed
      // values
      buffer.pop_back();
      for (const auto &i : changed[layer]) {
        appeared[i].pop_back();
```

```
  }
  changed[layer].clear();
  --layer;
  continue;
 }

 int eq = s.find("=");
 string s1 = s.substr(0, eq);
 string s2 = s.substr(eq + 1, s.size());

 if (checknum(s2)) {
  // <var1>=number
  buffer[layer][s1] = atoll(s2.c_str());
  // if empty var1 for this layer, adding it to layers
  if (appeared[s1].empty() ||
      appeared[s1][appeared[s1].size() - 1] != layer) {
    appeared[s1].push_back(layer);
    changed[layer].push_back(s1);
  }
 } else {
  // <var1>=<var2>
  // if empty var2 for this layer, adding it to layers
  if (appeared[s2].empty()) {
    appeared[s2].push_back(layer);
    changed[layer].push_back(s2);
  }
  // searching for a value of var2 in this and previous layers
  size_t s2_layer = appeared[s2][appeared[s2].size() - 1];
  buffer[layer][s1] = buffer[s2_layer][s2];

  // if empty var1 for this layer, adding it to layers
  if (appeared[s1].empty() ||
      appeared[s1][appeared[s1].size() - 1] != layer) {
    appeared[s1].push_back(layer);
    changed[layer].push_back(s1);
  }
```

```
      cout << buffer[layer][s1] << "\n";
    }
  }
  return 0;
}
```

Задача №D “Доктор Хаос”

Решение за O(d) (по сравнению с входным количеством дней можно считать за O(1)).
Пусть a – начальное количество бактерий, f(x) = x * b – c – функция изменения количества бактерий. Заметим, что данная функция монотонна, то есть если f(a) > a, то f(f(a)) > f(a) и наоборот. Далее заметим, что в контейнере в любой момент времени может находится от 0 (очевидно) до d бактерий, то есть в конце дня в контейнер помещается min(f(a), d) бактерий. Исходя из факта о монотонности функции и ограничении вместимости контейнера делаем вывод, что при f(a) > d или f(a) < 0 дальнейшие вычисления бессмысленны. Дополнительный случай: при f(a) = a дальнейшие вычисления так же бессмысленны, так как значение функции изо дня в день изменяться не будет. Следовательно, количество бактерий можно находить в течение максимум d итераций, что гораздо меньше возможного входного количества дней.

Код решения:
```
#include <iostream>

using namespace std;

int main() {
  ios_base::sync_with_stdio(false);
  cin.tie(NULL);

  long long a, b, c, d, k;
  cin >> a >> b >> c >> d >> k;
  long long last = a;
```

```
  for (long long i = 1; i <= k; ++i) {
    long long m = a * b;
    if ((a * b) <= c) {
      cout << 0;
      return 0;
    }
    long long n = m - c;
    if (n >= d) {
      cout << d;
      return 0;
    }
    a = min(n, d);
    if (a == last) {
      cout << last;
      return 0;
    }
    last = a;
  }
  cout << a;
  return 0;
}
```