Университет ИТМО

Факультет программной инженерии и компьютерной техники

Лабораторная работа №1

по «Алгоритмам и структурам данных»

Базовые задачи

Выполнил:

Студент группы P3231

Савон Г.К.

Преподаватели:

Косяков М.С.

Тараканов Д.С.

Санкт-Петербург

2021

Задача А «Агроном-любитель»

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

int main() {
  int n;
  int max_size = 0;
  int left_max_line;
  int right_max_line;
  int now_left = 0;
  int now_right = -1;
  int now_flower = -1;
  int count_flower = 0;
  int prev_flower;
  cin >> n;
  for (int i = 0; i < n; i++) {
    now_right++;
    prev_flower = now_flower;
    cin >> now_flower;
    if (prev_flower == now_flower) {
      count_flower += 1;
      if (count_flower != 3) {
        if (now_right - now_left + 1 > max_size) {
          max_size = now_right - now_left + 1;
          left_max_line = now_left;
          right_max_line = now_right;
        }
      } else {
        now_left = now_right - 1;
        count_flower = 2;
      }
    } else {
      count_flower = 1;
      if (now_right - now_left + 1 > max_size) {
        max_size = now_right - now_left + 1;
        left_max_line = now_left;
        right_max_line = now_right;
      }
    }
  }
  cout << left_max_line + 1 << " " << right_max_line + 1;
  return 0;
}
```

Пояснение к примененному алгоритму:

O(n)

Считываем количество цветков. И создаем 2 указателя на правый и левый конец текущей последовательности цветов (изначально поставив их оба в самое начало).

Постепенно увеличиваем правый, считывая значения каждого цветка.

Также считаем количество подряд идущих одинаковых цветков на данный момент, и если их становится слишком много, то переставляем левый указатель на второй из идущих на данный момент одинаковых цветков.

И соответственно, если разница между правым и левым указателем на каком-либо шаге становится больше когда-либо имевшейся, то обновляем это значение, а также значения левого и правого края максимально длинной последовательности.

Задача B «Зоопарк Глеба»

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

int main() {
  vector<vector<int>> numbers; // 0 - само число, 1 - его фактический номер
  vector<int> ans;
  int count_big = 0;
  int count_little = 0;
  char let;
  string chars;
  cin >> chars;
  for (int i = 0; i < chars.size(); i++) {
    let = chars[i];
    if (static_cast<int>(let) > static_cast<int>('Z')) {
      count_little++;
      numbers.push_back({static_cast<int>(let), count_little});
    } else {
      numbers.push_back({static_cast<int>(let), count_big});
      count_big++;
      ans.push_back(0);
    }
    if (numbers.size() > 1 &&
        abs(numbers[numbers.size() - 2][0] - numbers[numbers.size() - 1][0]) ==
            32) {
      if (numbers[numbers.size() - 2][0] > static_cast<int>('Z')) {
        ans[numbers[numbers.size() - 1][1]] = numbers[numbers.size() - 2][1];
      } else
        ans[numbers[numbers.size() - 2][1]] = numbers[numbers.size() - 1][1];
      numbers.resize(numbers.size() - 2);
    }
  }
  if (numbers.size() == 0) {
    cout << "Possible" << endl;
    for (int i = 0; i < ans.size(); i++)
      cout << ans[i] << " ";
  } else
    cout << "Impossible";
  return 0;
}
```

Пояснение к примененному алгоритму:

O(n)

Небольшое пояснение: нельзя разогнать всех животных по клеткам только если две траектории пересекаются ..A…b..a..B.. (и при этом нет других возможностей куда-нибудь их отправить). Это чем-то похоже на задачи с последовательностей разных скобок, и каждая буква – вид скобки, но они не делятся на открывающие и закрывающие(тк можно и Aa и aA).

Для удобства будем запоминать не char-ы, а номера этих char-ов. (Между большой и маленькой буквой эти номера отличаются на 32, из-за чего будет удобно искать пары. А букв в алфавите меньше 32, так что даже никаких коллизий не возникнет)

Заводим вектор для считывания животных и клеток (с порядковым номером. Для клеток и для животных эти номера разные соответственно). Считываем поочередно буковки, и сравниваем значение новой буквы с предыдущим, если она равно 32, значит идет рядом стоящие зверек и клетка, соответственно запишем в ответ для этой клетки (не зря ее номер запоминали) номер зверя (его тоже запоминали), и удаляем общего вектора.

Если для каждого зверя нашлась клетка, то вектор будет пуст в конце. Иначе – в нем что-то останется и тогда мы выведем «Импоссибл».

Задача C «Конфигурационный файл»

```cpp
#include <iostream>
#include <vector>
#include <map>
#include <set>
#include <stdlib.h>
#include <string>

using namespace std;

map<string, vector<int>> all_var; //<имя переменной: значения >
map<int, set<string>> used_this_level; // level, names
int level;

bool ifNumber(string word) {
  if (word[0] != '-' && (word[0] > '9' || word[0] < '0'))
    return false;
  for (int i = 1; i < word.size(); i++)
    if (word[i] > '9' || word[i] < '0')
      return false;
  return true;
}

void remade_vec(string name, int last_num, set<string> &used_set) {
  vector<int> replace_vec;
  replace_vec.clear();
  if (all_var.find(name) != all_var.end()) {
    replace_vec = all_var[name];
    if (used_set.find(name) != used_set.end())
      replace_vec.pop_back();
  }
  replace_vec.push_back(last_num);
  all_var[name] = replace_vec;
}
```

```cpp
int main() {
  string str;
  string str1;
  string str2;
  vector<int> vec_of_str2;
  set<string> used_set;
  level = 0;
  map<int, set<string>>::iterator it_level;
  while (getline(cin, str)) {
    //начало блока
    if (str[0] == '{') {
      level++;
      used_this_level[level - 1] = used_set;
      used_set.clear();
    } else {
      //конец блока
      if (str[0] == '}') {
        for (set<string>::iterator i = used_set.begin(); i != used_set.end();
             i++) {
          string s = *i;
          all_var[s].pop_back();
          if (all_var[s].size() < 1)
            all_var.erase(s);
        }
        used_set = used_this_level[level - 1];
        if (used_this_level.find(level) != used_this_level.end()) {
          it_level = used_this_level.find(level);
          used_this_level.erase(it_level);
        }
        level--;
      } else {
        //парсинг
        str1 = "";
        str2 = "";
        int i = 0;
        while (str[i] != '=') {
          str1 += str[i];
          i++;
        }
        str2 = str.substr(i + 1, str.size() - 1);
        //если var=number
        if (ifNumber(str2)) {
          remade_vec(str1, stoi(str2), used_set);
        }
        // если var=var
        else {
          int now_number;
          if (all_var.find(str2) == all_var.end()) {
            now_number = 0;
          } else {
            vec_of_str2 = all_var[str2];
            now_number = vec_of_str2[vec_of_str2.size() - 1];
          }
          remade_vec(str1, now_number, used_set);

          cout << now_number << endl;
        }
        used_set.insert(str1);
      }
    }
  }
  return 0;
}
```

Пояснение к примененному алгоритму:

O(nlogn)

Создадим несколько штук:

-мапу с ключем – именем переменной и значением – вектором ее еще актуальных (в открытых блоках) значениями

-мапу с ключами – значениям еще открытых уровней* и значением – сетом переменных, которые изменялись на данном уровне

*(уровень – это номер вложенного блока, при входе в новый блок уровень увеличивается, при выходе из блока – уменьшается, и соответственно удаляются

все последние значения переменных, менявшихся на данном уровне из первой мапы (для того, чтобы помнить эти значения еще вектор строк на каждом уровне заполняем, а если открывается новый блок – закидываем этот вектор в мапу, чтобы при возвращении в него можно было вспомнить, что же мы вообще меняли уже)

Читаем поочередно строки и парсим их, соответственно уменьшая и увеличивая уровень (если это скобки) и добавляя новое или изменяя (если она уже изменялась в текущем блоке) значение первой переменной на данном уровне и записывая в сет изменявшихся переменных на этом уровне.

Ну и конечно же выводим присвоенное значение для строк вида <variable1>=<variable2>

Задача D «Профессор Хаос»

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

int main() {
  int a, b, c, d, k;
  cin >> a >> b >> c >> d >> k;
  for (int i = 0; i < k; i++) {
    a = a * b - c;
    if (a <= 0)
      break;
    if (a * b - c == a)
      break;
    if (a > d) {
      a = d;
      if (a * b - c > a)
        break;
    }
  }
  if (a <= 0)
    cout << 0;
  else
    cout << a;
  return 0;
}
```

Пояснение к примененному алгоритму:

O(k)

Считываем значения всех параметров. И по количеству дней пересчитываем количество бактерий, отправленных в контейнер на конец каждого конкретного дня. Пожалуй, это не самое эффективное решение при большом количестве дней.

Но если в какой-то момент количество бактерий становится 0, то очевидно, больше их уже не образуется. А также, если на какой-то момент количество бактерий в начале дня равно количеству в конце дня (или больше, чем влезает в контейнер), то итоговое значение в конце каждого дня будет одинаковым и дальше. Во всех этих случаях следующие дни не смотрим уже.