Университет ИТМО

Факультет программной инженерии и компьютерной техники

**Лабораторная работа №1**

по «Алгоритмам и структурам данных»

Базовые задачи

Выполнил:

Студент группы P32092

Голиков А.С.

Преподаватели:

Косяков М.С.

Тараканов Д.С.

Санкт-Петербург

2023

Задача A «Агроном-любитель»

Пояснение к примененному алгоритму:

Сложно что-то доказывать в этой задаче..

Динамически ищем нужную подпоследовательность, если встретили 3 подряд одинаковых - начинаем считать новую, начиная со второй буквы из тех 3 одинаковых.

Так, полученная последовательность является искомой, так как она не может быть продлена ни в какую сторону (начинается/кончается либо в начале/конце строки, либо на втором повторе тройки одинаковых).

Асимптотика - O(n)

Код решения:

#include <iostream>  
  
int main() {  
 int n;  
 std::cin >> n;  
 int count\_rep = 1;  
 int prev;  
 int start = 1;  
 std::cin >> prev;  
 int ans = 1, curr = 1, first = 1, second = 1;  
 for (size\_t i = 1; i < n; ++i) {  
 int x;  
 std::cin >> x;  
 if (x == prev) {  
 count\_rep += 1;  
 } else {  
 count\_rep = 1;  
 }  
  
 if (count\_rep < 3) {  
 curr += 1;  
 if (curr > ans) {  
 ans = curr;  
 first = start;  
 second = i + 1;  
 }  
 } else {  
 curr = 2;  
 count\_rep = 2;  
 start = i;  
 }  
 prev = x;  
 }  
 std::cout << first << " " << second;  
 return 0;  
}

Задача B «Зоопарк Глеба»

Пояснение к примененному алгоритму:

Проведем очевидную биекцию между этой задачей и проведением цветных непересекающихся хорд окружности.

Если мы провели хорду - закроем на нее глаза, будто ее нет (провели и провели чо на нее смотреть)

Будем идти по окружности и проводить хорду только назад и только в том случае, если она не отрежет никаких точек, то есть либо с соседней точкой, либо с такой, что там где-то уже все хорды проведены (но мы же уже договорились, что на них не смотрим - так что хоры можно проводить только назад и только в соседние точки)

Так как мы пройдем по всем точкам, мы рассмотрим каждый потенциальный второй конец хорды, а значит, ничего не пропустим.

Если ничего не осталось - мы успешно провели все хорды (ура салют овации ну победа кароче)

Если остались точки, то мы проиграли, так как в правильной конфигурации всегда есть крайняя хорда, а тут нет.

Асимптотика - O(n)

Код решения:

#include <iostream>  
#include <string>  
#include <stack>  
#include <vector>  
  
  
bool is\_lower(char c) {  
 return c <= 'z' && c >= 'a';  
}  
  
bool is\_upper(char c) {  
 return c <= 'Z' && c >= 'A';  
}  
  
bool is\_pair(char a, char b) {  
 return is\_lower(a) == is\_upper(b) && toupper(a) == toupper(b);  
}  
  
using namespace std;  
  
int main() {  
  
 std::string input;  
 std::cin >> input;  
  
 std::stack<std::pair<char, int>> st;  
 size\_t i = 0;  
 size\_t n = input.size();  
 int count\_low = 0;  
 int count\_upp = 0;  
 std::vector<int> nums(n / 2);  
 while (i < n) {  
 int this\_char;  
 if (is\_upper(input[i])) {  
 this\_char = ++count\_upp;  
 } else {  
 this\_char = ++count\_low;  
 }  
 if (!st.empty() && is\_pair(input[i], st.top().first)) {  
 if (is\_lower(input[i])) {  
 nums[st.top().second - 1] = count\_low;  
 } else {  
 nums[count\_upp - 1] = st.top().second;  
 }  
 st.pop();  
 } else {  
 st.emplace(input[i], this\_char);  
 }  
 i += 1;  
 }  
  
 if (st.empty()) {  
 std::cout << "Possible\n";  
 for (auto t: nums) {  
 std::cout << t << " ";  
 }  
  
 } else {  
 std::cout << "Impossible";  
 }  
  
 return 0;  
}

Задача C «Конфигурационный файл»

Пояснение к примененному алгоритму:

Тут никакой идеи нет, так что и доказывать нечего. Нам необходимо иметь массив ассоциированных пар - мапу.

При переходе на новый уровень вложенности блоков надо запоминать старую мапу чтобы при выходе ее восстановить

Чтобы это реализовать я храню по каждому имени стэк из пар значение-уровень вложенности и при выходе из блока попаю из всех обновленных стеков значения до нужного уровня вложенности.

Асимптотика - O(n) (с большой константой)

Код решения:

#include <iostream>  
#include <unordered\_map>  
#include <stack>  
#include <string>  
#include <vector>  
#include <set>  
  
using namespace std;  
  
vector<string> split(const string &stringToBeSplitted, const string &delimeter) {  
 vector<string> splittedString;  
 unsigned long startIndex = 0;  
 unsigned long endIndex;  
 while ((endIndex = stringToBeSplitted.find(delimeter, startIndex)) < stringToBeSplitted.size()) {  
 string val = stringToBeSplitted.substr(startIndex, endIndex - startIndex);  
 splittedString.push\_back(val);  
 startIndex = endIndex + delimeter.size();  
 }  
 if (startIndex < stringToBeSplitted.size()) {  
 string val = stringToBeSplitted.substr(startIndex);  
 splittedString.push\_back(val);  
 }  
 return splittedString;  
}  
  
int parse\_int(string s) {  
 int is\_negative = 1;  
 int ans = 0;  
 if (s[0] == '-') {  
 is\_negative = -1;  
 s = s.substr(1);  
 }  
 for (char i: s) {  
 ans \*= 10;  
 ans += i - '0';  
 }  
 return is\_negative \* ans;  
}  
  
void handle\_init(unordered\_map<string, stack<pair<int, int>>> \*map, const string &name, int value, int level\_counter,  
 set<string> \*set) {  
 if ((\*map).find(name) != (\*map).end()) {  
 while (!(\*map)[name].empty() && (\*map)[name].top().second >= level\_counter) {  
 (\*map)[name].pop();  
 }  
 }  
 (\*map)[name].emplace(value, level\_counter);  
 (\*set).insert(name);  
}  
  
void  
handle\_ref(unordered\_map<string, stack<pair<int, int>>> \*map, const string &dest, const string &reference,  
 int level\_counter, set<string> \*set) {  
 int val = 0;  
 if ((\*map).find(reference) != (\*map).end()) {  
 while (!(\*map)[reference].empty() && (\*map)[reference].top().second > level\_counter) {  
 (\*map)[reference].pop();  
 }  
 if (!(\*map)[reference].empty()) {  
 val = (\*map)[reference].top().first;  
 }  
 }  
 handle\_init(map, dest, val, level\_counter, set);  
 cout << val << "\n";  
}  
  
inline void handle\_start\_block(int \*level\_counter, set<string> \*vars, stack<set<string>> \*st) {  
 (\*level\_counter)++;  
 (\*st).push(\*vars);  
 (\*vars).clear();  
}  
  
inline void handle\_end\_block(unordered\_map<string, stack<pair<int, int>>> \*map, int \*level\_counter, set<string> \*vars,  
 stack<set<string>> \*st) {  
 (\*level\_counter)--;  
 int c = \*level\_counter;  
  
 for (const auto &t: (\*vars)) {  
 if (!(\*map)[t].empty()) {  
 if ((\*map)[t].top().second > c) {  
 (\*map)[t].pop();  
 }  
 }  
 }  
 (\*vars) = (\*st).top();  
 (\*st).pop();  
  
}  
  
int main() {  
 int level\_counter = 0;  
 unordered\_map<string, stack<pair<int, int>>> map;  
 set<string> vars;  
 stack<set<string>> st;  
 string curr;  
 while (getline(cin, curr)) {  
 if (curr == "{") {  
 handle\_start\_block(&level\_counter, &vars, &st);  
 } else if (curr == "}") {  
 handle\_end\_block(&map, &level\_counter, &vars, &st);  
 } else {  
 vector<string> curr\_v = split(curr, "=");  
 string name = curr\_v[0];  
 if (curr\_v[1][0] - '0' <= 9) {  
 int value = parse\_int(curr\_v[1]);  
 handle\_init(&map, name, value, level\_counter, &vars);  
 } else {  
 string ref = curr\_v[1];  
 handle\_ref(&map, name, ref, level\_counter, &vars);  
 }  
 }  
 }  
 return 0;  
}

Задача D «Профессор Хаос»

Пояснение к примененному алгоритму:

Дана последовательность чисел (количества бактерий каждый день, без учета онуления или обрезания по d), t-тый член которой выглядит как  
a\_t = (ab-c)b-c... = ab^t - c(b^{t-1} + b^{t-2} + ... 1) = ab^t - c(b^t-1)/(b-1) = (a-c/(b-1))b^t + c/(b-1).  
Очевидно, что монотонность последовательности зависит от знака <?> в выражении a <?> c/(b-1)  
  
  
>) Мы имеем возрастающую последовательность, имеем доминирующий член b^t, который при минимально возможном b = 2 при t > 10 уже дает >1024, а макс d = 1000, значит перебор и считать не нужно  
<) Аналогично имеем убываюшую последовательность, где мы прийдем к 0 при t > 10  
=) тут все понятно  
  
Итого, нам достаточно посчитать руками при k <= 10 и вывести d или 0 зависимости от <?>  
  
Асимптотика - O(1)

Код решения:

#include <iostream>  
  
int main() {  
 int a, b, c, d;  
 unsigned long long k;  
 std::cin >> a >> b >> c >> d >> k;  
 int ans = a;  
 if (k < 10) {  
 for (size\_t i = 0; i < k; ++i) {  
 ans = ans \* b - c;  
 }  
 if (ans <= 0) {  
 std::cout << 0;  
 } else if (ans >= d) {  
 std::cout << d;  
 } else {  
 std::cout << ans;  
 }  
 return 0;  
 }  
 if (b == 1) {  
 if (a > c \* k) {  
 std::cout << a - c \* k;  
 } else {  
 std::cout << 0;  
 }  
 } else {  
 double diff = (double) c / (b - 1);  
 if ((double) a == diff) {  
 std::cout << a;  
 } else if ((double) a < diff) {  
 std::cout << 0;  
 } else {  
 std::cout << d;  
 }  
 }  
 return 0;  
}