Федеральное государственное автономное образовательное учреждение высшего образования «Национальный исследовательский университет ИТМО»

Факультет программной инженерии и компьютерной техники

Отчет по блоку №1 (яндекс контест)

По Алгоритмам и структурам данных

Выполнил:

Ступин Тимур Русланович

Группа № P3208

Поток № 1.3

Преподаватель:

Тараканов Денис Сергеевич

Санкт-Петербург 2025

Содержание

Задача 1 Агроном-любитель

Решение

Воспользуемся методом двух указателей. Заведём указатели `l=0` и `r=2`. При этом будем считать что левый указатель является началом отрезка, а правый указывает на элемент после его конца. Таким образом изначально имеем отрезок из первых двух элементов. Также будем поддерживать координата лучшего на данный момент отрезка его длину (`max_len`, `best_l`, `best_r`). Далее пока правая граница не выйдет за пределы массива будем проверять, не совпадает ли текущий правый элемент с двумя предыдущими. Если да, то значит дальше формировать отрезок нельзя. В таком случае обновляем максимум и устанавливаем левую границу в предыдущую ячейку. В конце в любом случае сдвигаем правую границу.

Сложность алгоритма

1. **По времени:** выполняется один проход по массиву, что даёт сложность O(n)
2. **По памяти:** помимо хранения входных данных выделяется только место под локальные переменные, поэтому сложность по памяти O(1), или O(n) в случае учета хранения входных данных.

Итого:

- **По времени:** O(n)
- **По памяти:** O(1)

Код

```cpp
#include <bits/stdc++.h>
using namespace std;
int main() {
  int n;
  cin >> n;
  vector<int> a(n);
  for (int i = 0; i < n; i++) {
    cin >> a[i];
  }
  if (n < 3) {
    cout << 1 << " " << n << endl;
    return 0;
  }
  int l = 0;
  int r = 2;
  int max_len = r - l;
  int best_l = l + 1;
  int best_r = r;
  while (r < n) {
    if (a[r] == a[r - 1] && a[r] == a[r - 2]) {
      if (r - l > max_len) {
        max_len = r - l;
        best_l = l + 1;
        best_r = r;
      }
      l = r - 1;
    }
    r++;
  }
  if (r - l > max_len) {
    best_l = l + 1;
    best_r = r;
  }
  cout << best_l << " " << best_r << endl;
  return 0;
}
```

Задача 2 Зоопарк Глеба

Решение

Заметим, что данную задачу можно свести к задаче, похожей на проверку правильности скобочной последовательности с несколькими типами скобок (в данном случае 26). Заведём три стека:

- Для животных (`animals`), в котором будем хранить номера животных в порядке обхода,
- Для ловушек (`cells`), в котором будем хранить индексы ловушек
- Общий стек (`st`) который и будет использоваться для проверки правильности последовательности.

Также заведём массив ответов, в который будем по индексам ловушек вписывать номера пойманных туда животных (просто чтобы потом не сортировать) и счётчик для животных (`cnt`). Далее, проходясь по строке, будем выполнять следующее:

- В зависимости от типа текущего элемента добавляем его индекс\номер в стек ловушек\животных.
- Если общий стек не пуст и при этом текущий элемент и верхний в стеке образуют пару ловушка-животное, то фиксируем пару в массиве ответов и удаляем верху верхние элементы со всех стеков (мы их использовали)
- Иначе кладём текущий элемент на вершину общего стека.

В конце достаточно проверить что общий стек пуст и если это не так то ответ `Impossible`. Для формирования самого ответа достаточно пройтись по массиву `ans` и вывести все ненулевые элементы.

Сложность алгоритма

1. **По времени:** выполняется один проход по массиву, что даёт сложность O(n)
2. **По памяти:** выделяется место под три стека, максимальный возможный размер которых равен 2n. Также выделяется массив для формирования ответов размера 2n. получаем суммарно O(8n) что эквивалентно O(n).

Итого:

- **По времени:** O(n)
- **По памяти:** O(n)

Код

```
#include <bits/stdc++.h>
using namespace std;
int main() {
  string s;
  cin >> s;
  stack<int> animals;
  stack<int> cells;
  stack<char> st;
  vector ans(s.size(), 0);
  int cnt = 1;
  for (int i = 0; i < static_cast<int>(s.size()); i++) {
    s[i] >= 'a' && s[i] <= 'z' ? animals.push(cnt++) : ceils.push(i);
    if (!st.empty() && s[i] != st.top() &&
       tolower(s[i]) == tolower(st.top())) {
      ans[cells.top()] = animals.top();
      st.pop();
      cells.pop();
      animals.pop();
    } else {
      st.push(s[i]);
    }
  }
  if (st.empty()) {
    cout << "Possible" << endl;
    for (auto a : ans) {
      if (a != 0)
        cout << a << " ";
    }
    cout << endl;
  } else {
    cout << "Impossible" << endl;
  }
  return 0;
}
```

Задача 3 Конфигурационный файл

Решение

Для решения данной задачи я использовал рекурсию. Определим функцию parse, которая принимает массив строк для парсинга, текущий указатель в нем, а также хеш-таблицу, в виде stl контейнера `unordered_map` для хранения информации о значении переменных. Сама функция возвращает индекс элемента до которого был произведён парсинг, или -1 если все данные были обработаны.

Общая идея в том, что перед тем как менять значение переменной, мы сохраняем предыдущее значение (или 0 если это первое появление), меняем значение переменной, производим рекурсивный вызов парсинга, и после этого возвращаем переменно исходное значение. Это позволяет решить проблему восстановления значения переменных после выхода из блока, так как вся история изменения значений хранится в стеке вызовов.

Также важно отметить, что возвращаемое значение используется для определения конца блоков. Вот поясняющий пример:

```
a=2
{
b=2
a=b
}
s=a
```

Мы встретим символ `{` при `ptr=1`. Мы производим рекурсивный вызов, который последовательно обработает всё содержимое блока, и в конце концов дойдет до закрывающей скобки `}` при `ptr=4`, откуда вернется значение `ptr`, которое пройдет по стеку вызовов до момента `ptr=1`, после чего парсинг продолжится со следующего за возвращенным элемента, а именно `ptr=5`.

Сложность алгоритма

1. **По времени:** выполняется один проход по массиву, что даёт сложность O(n)
2. **По памяти:** Для хранения данных о соответствиях используется хэш-таблица, которая в худшем случае занимает O(n) по памяти.

Итого:

- **По времени:** O(n)

- **По памяти:** O(n)

Код

```cpp
#include <bits/stdc++.h>
using namespace std;

int parse(vector<string>& v, int ptr, unordered_map<string, int>& m) {
  if (ptr == static_cast<int>(v.size()))
    return -1;
  string s = v[ptr];
  if (s == "}")
    return ptr;
  if (s == "{") {
    const int new_ptr = parse(v, ptr + 1, m);
    if (new_ptr == -1)
      return -1;
    return parse(v, new_ptr + 1, m);
  }
  const int eq = static_cast<int>(s.find('='));
  const string var = s.substr(0, eq);
  const string val = s.substr(eq + 1);
  int buf = 0;
  if ((val[0] >= '0' && val[0] <= '9') ||
       val[0] == '-' || val[0] == '+') {
    buf = m[var];
    m[var] = stoi(val);
  } else {
    buf = m[var];
    m[var] = m[val];
    cout << m[val] << endl;
  }
  const int res = parse(v, ptr + 1, m);
  m[var] = buf;
  return res;
}

int main() {
  unordered_map<string, int> m;
  vector<string> v;
  string s;
  while (cin >> s) {
    v.push_back(s);
  }
  parse(v, 0, m);
  return 0;
}
```

Задача 4 Профессор Хаос

Решение

Идея заключается в том, что в какой-то момент времени количество бактерий начнет циклически повторяться до бесконечности, либо же они будут полностью уничтожены. Причём с учётом ограничений для входных данных процесс перехода к данному моменту можно полностью смоделировать и запомнить все значения в цикле. После этого останется только правильно пересчитать номер запрашиваемого дня, приведя его к номеру в списке цикла.

Для хранения истории моделирования вводится массив `history` а для определения момента цикла массив `used` на 1001 элемент. Данное ограничение размера достаточно, так как к концу любого дня у нас не может быть больше 1000 бактерии в связи с ограничением на `d`. В случае если мы смогли промоделировать все запрашиваемые дни то просто выводится текущее количество бактерий `cur`. Если же нет, то вычисляется день начала цикла (это просто `used[cur]`) а также количество дней в цикле `delt`, после чего производится перерасчет. Для пояснения формулы пересчета приведу пример:

```
1 3 1 5 2
```

Цикл моделирования завершится на шаге `day=3`, когда значение `cur=5`, так как такое уже было в конце `day=2 (used[5]=2)`. Вот вид массива `history`:

```
history = [1, 2, 5]
```

Очевидно что мы не завершили рассчёт, а значит будет применения формула пересчета. Значение `l = used[cur] = used[5] = 2`. Значение `delt = day - l = 3 - 2 = 1`. Это действительно так, ведь из 5 бактерий на конец 2-го дня мы получили 5 бактерий на конец 3-го дня, а значит так будет продолжать до бесконечности. Теперь применим саму формулу и получим что нас интересует день `l=2`.

Сложность алгоритма

1. **По времени:** максимальное количество шагов моделирования заранее неизвестно, но оно зависит от `k`, поэтому сложность O(k)
2. **По памяти:** Используются массивы `used` и `history`, что даёт сложность O(k)

Итого:

- **По времени:** O(k)
- **По памяти:** O(k)

Код

```
#include <bits/stdc++.h>
using namespace std;

int main() {
  int a, b, c, d, k;
  cin >> a >> b >> c >> d >> k;

  vector used(1001, -1);
  vector<int> history;

  used[a] = 0;
  history.push_back(a);

  int cur = a;
  int day = 1;
  for (; day <= k; day++) {
    cur *= b;
    if (cur <= c) {
      cout << 0 << endl;
      return 0;
    }
    cur -= c;
    cur = min(cur, d);
    if (used[cur] != -1)
      break;
    used[cur] = day;
    history.push_back(cur);
  }
  if (day == k + 1) {
    cout << cur << endl;
    return 0;
  }
  const int l = used[cur];
  const int delt = day - l;
  cout << history[l + (k - day) % delt] << endl;
  return 0;
}
```