Университет ИТМО

Факультет программной инженерии и компьютерной техники

Лабораторная работа №1

по «Алгоритмам и структурам данных»

Базовые задачи / Timus

Выполнил:

Студент группы P3219

Билобрам Д.А.

Преподаватели:

Косяков М.С.

Тараканов Д.С.

Санкт-Петербург

2023

Задача №1 «Агроном-любитель»

Пояснение к примененному алгоритму:

Алгоритм заключается в том, что, если я встречаю элемент, который повторяется 3 раза, я записываю индекс серединного его повторения (из этих трёх) а затем прохожусь по этим индексам как по границам отрезков и нахожу максимальный отрезок, он и будет решением.

Алгоритм будет корректным, так как я записываю индекс середины вхождения, а затем при подсчете длины отрезка я прибавляю единицу, поэтому подсчет длины будет учитывать и начало, и конец отрезка.

Оценка сложности по времени: O(n)

Оценка сложности по памяти: O(n)

```
int main()
{
    int n;
    cin >> n;
    vector<int> a(n);
    vector<int> borders;

    for (int i = 0; i < n; i++)
    {
        cin >> a[i];
    }

    int start = 0;
    int end = 0;

    int m = 0;
    int m_c = 0;

    if (n == 1)
    {
        cout << 1 << " " << 1;
    }
    else if (n == 2)
    {
        cout << 1 << " " << 2;
    }
    else
    {
        borders.insert(borders.begin(), 1);
        for (int i = 1; i < n - 1; i++)
        {
            if (a[i] == a[i - 1] && a[i] == a[i + 1])
            {
                borders.push_back(i + 1);
            }
        }
        borders.insert(borders.end(), n);
```

```
        for (int i = 0; i < borders.size() - 1; i++)
        {

            m_c = borders[i + 1] - borders[i] + 1;

            if (m_c > m)
            {
                m = m_c;
                start = borders[i];
                end = borders[i + 1];
            }
        }

        cout << start << " " << end;
    }
}
```

Задача №2 «Зоопарк Глеба»

Пояснение к примененному алгоритму:

Идея алгоритма заключается в том, что при раскрытии окружности каждый путь от животного к ловушке становится дугой, и чтобы эти дуги не пересекались, нужно чтобы после начала одной дуги не встречалась другая дуга, один конец которой лежит внутри дуги (предыдущей), а другой снаружи.

Этот алгоритм будет работать корректно, так как когда мы раскрываем круг зоопарка в линию, любой путь от животного к ловушке можно представить как дугу. Условие не пересечения путей преобразуется в условие, что если начало одной дуги находится между началом и концом другой дуги, то и конец первой дуги также должен находиться внутри второй дуги (или вне, если смотреть на ситуацию с другой стороны).

Оценка сложности по времени: O(n*logn)

Оценка сложности по памяти: O(n)

```
int main()
{
    string st;
    cin >> st;
    int n = st.size();

    stack<pair<char, int>> stck;
    vector<pair<int, int>> vec;

    int ind;
    if (islower(st[0]))
        ind = 1;
    else
```

```
    ind = 0;

stck.push(make_pair(st[0], ind));
for (int i = 1; i < n; i++)
{
    if (stck.empty())
    {
        stck.push(make_pair(st[i], i));
        continue;
    }

    if (isupper(st[i]))
    {
        if (tolower(st[i]) == stck.top().first)
        {
            vec.push_back(make_pair(i, stck.top().second));
            stck.pop();
        }
        else
        {
            stck.push(make_pair(st[i], i));
        }
    }
    else
    {
        ind++;
        if (toupper(st[i]) == stck.top().first)
        {
            vec.push_back(make_pair(stck.top().second, ind));
            stck.pop();
        }
        else
        {
            stck.push(make_pair(st[i], ind));
        }
    }
}

if (stck.empty())
{
    sort(vec.begin(), vec.end());

    cout << "Possible\n";
    for (pair<int, int> el : vec)
    {
        cout << el.second << " ";
    }
}
else
{
    cout << "Impossible";
```

```
    }
}
```

Задача №3 «Конфигурационный файл»

Пояснение к примененному алгоритму:

Алгоритм заключается в хранении значений переменной по уровню вложенности ({}) а также в хранении истории изменения переменных на каждом уровне вложенности. В итоге, когда мы встречаем “}” можно просто посмотреть какие переменные мы меняли на текущем уровне вложенности и откатить их значение до прошлого уровня. При этом мы запоминаем лишь последнее значение переменной на каждом уровне, так как предыдущие нас не интересуют.
Алгортм будет работать корректно так как он просто запоминает изменения переменных по уровням и в случае необходимости откатывает их значения до предыдущего уровня.

Оценка сложности по времени: O(n)

Оценка сложности по памяти: O(n)

```
int main()
{
    string st
    size_t pos;
    string name;
    string val;
    int res;
    int lvl = 0;

    unordered_map<string, vector<pair<int, int>>> variables; // значение/уровень
    unordered_map<int, unordered_set<string>> changes;
    vector<string> vec;

    while (true)
    {
        getline(cin, st);
        if (st.empty())
        {
            break;
        }
        vec.push_back(st);
    }

    for (string st : vec)
    {

        if (st == "{")
        {
            lvl++;
        }
```

```
else if (st == "}")
{
    for (string changed : changes[lvl])
    {
        variables[changed].pop_back();
        if (variables[changed].empty())
        {
            variables.erase(changed);
        }
    }
    changes[lvl].clear();
    lvl--;
}
else
{
    pos = st.find("=");
    name = st.substr(0, pos);
    val = st.substr(pos + 1);

    res = 0;
    if (isNumber(val))
    {
        res = stoi(val);
    }
    else
    {

        if (variables.count(val))
        {
            res = variables[val].back().first;
        }

        cout << res << "\n";
    }
    if (variables.count(name))
    {
        if (variables[name].back().second == lvl)
        {
            variables[name].back().first = res;
        }
        else
        {
            variables[name].push_back(make_pair(res, lvl));
        }
    }
    else
    {
        variables[name] = vector<pair<int, int>>();
        variables[name].push_back(make_pair(res, lvl));
    }
    changes[lvl].insert(name);
}
```

```
    }
}
```

Задача №4 «Доктор Хаос»

Пояснение к примененному алгоритму:

Алгоритм заключается в эмуляции процессов, описанных в задаче. Ключевой является проверка на то, что количество бактерий за два дня не изменилось, из этого следует что количество бактерий и далее не будет меняться, а значит можно прекратить эмуляцию, не доходя до k-того дня.

Алгоритм будет работать верно, ведь несмотря на возможные огромные значения параметра k, есть достаточно небольшие ограничения на кол-во бактерий c и d, а за каждый день количество бактерий изменится как минимум на 1, следовательно мы быстро достигнем ограничений, и максимальная продолжительность эмуляции это 1000 итераций.

Оценка сложности по времени: O(k)

Оценка сложности по памяти: O(1)

```
int main()
{

    int a, b, c, d, k;
    cin >> a >> b >> c >> d >> k;

    int now = a;
    int last;
    for (int i = 1; i <= k; i++)
    {
        last = now;
        now = now * b - c;
        if (now <= 0)
        {
            cout << 0;
            exit(0);
        }
        else if (now > d)
        {
            now = d;
        }

        if (now == last)
        {
            cout << now;
            exit(0);
        }
    }
```

```
    cout << now;
}
```

Задача №5 «Куча камней»

Пояснение к примененному алгоритму:

Алгоритм заключается в переборе всех возможных комбинаций двух куч камней. Для перебора используется битовая маска, которая позволяет удобно перебирать комбинации через инкремент. В маске 0 - камень лежит в первой куче, 1 - камень взят во вторую кучу. Таким образом можно посчитать разницу между всеми возможными кучами и выбрать минимальную.

Алгоритм работает корректно так как перебирает все возможные разницы между кучами, а также не превышает временной лимит ведь максимальное количество камней всего лишь 20.

Оценка сложности по времени: O(2^n)

Оценка сложности по памяти: O(n)

```
int main()
{

    int n;
    cin >> n;
    vector<int> weights(n);
    int total = 0;
    for (int i = 0; i < n; i++)
    {
        cin >> weights[i];
        total += weights[i];
    }

    int min_diff = total;
    for (int bit_mask = 0; bit_mask < (1 << n); bit_mask++)
    {
        int current = 0;
        for (int i = 0; i < n; i++)
        {
            if (bit_mask & (1 << i))
            {
                current += weights[i];
            }
        }
        int diff = abs(total - current * 2);
        min_diff = min(min_diff, diff);
    }
    cout << min_diff;
}
```

Задача №6 «Дуоны»

Пояснение к примененному алгоритму:

Мой алгоритм заключается в том, чтобы сначала пройти по всем рёбрам куба уничтожая как можно больше пар частиц у смежных вершин ребра, тогда частицы могут остаться только на несмежных вершинах, поэтому я рассматриваю диагонали куба, и в случае, если на диагонали в обоих вершинах есть частицы, я уничтожаю эти частицы через промежуточное создание частиц на ребре смежном обеим вершинам этой диагонали пока в одной из вершин не закончатся частицы. Затем я прохожусь по вершинам и если в одной из них остались частицы, то уничтожить их уже невозможно.

Алгоритм работает корректно так как он последовательно сокращает количество частиц при этом рассматривая все возможные сценарии взаимоположения частиц для уничтожения.

Оценка сложности по времени: O(n)

Оценка сложности по памяти: O(1)

```
int main()
{
    string result = "";
    map<int, vector<int>> neighbors = {
        {0, {1, 3, 4}},
        {1, {0, 2, 5}},
        {2, {1, 3, 6}},
        {3, {0, 2, 7}},
        {4, {0, 5, 7}},
        {5, {1, 4, 6}},
        {6, {2, 5, 7}},
        {7, {3, 4, 6}}};
    vector<pair<int, int>> borders = {{0, 1}, {1, 2}, {2, 3}, {3, 0}, {4, 5}, {5, 6},
{6, 7}, {7, 4}, {0, 4}, {1, 5}, {2, 6}, {3, 7}};
    vector<pair<int, int>> diagonals = {{0, 6}, {1, 7}, {2, 4}, {3, 5}};
    vector<string> LETTERS = {"A", "B", "C", "D", "E", "F", "G", "H"};

    int peaks[8];
    int sum = 0;

    for (int i = 0; i < 8; i++)
    {
        cin >> peaks[i];
        sum += peaks[i];
    }

    if (sum % 2 != 0)
    {
        cout << "IMPOSSIBLE";
        exit(0);
    }
```

```
    for (auto border : borders)
    {
        while (peaks[border.first] != 0 && peaks[border.second] != 0)
        {
            peaks[border.first]--;
            peaks[border.second]--;
            result += LETTERS[border.first] + LETTERS[border.second] + "-\n";
        }
    }
    for (auto diag : diagonals)
    {
        pair<int, int> tmp_pair;
        for (auto border : borders)
        {
            if (find(neighbors[diag.first].begin(), neighbors[diag.first].end(),
border.first) != neighbors[diag.first].end() &&
                find(neighbors[diag.second].begin(), neighbors[diag.second].end(),
border.second) != neighbors[diag.second].end())
            {
                tmp_pair = {border.first, border.second};
                break;
            }
            else if (find(neighbors[diag.first].begin(), neighbors[diag.first].end(),
border.second) != neighbors[diag.first].end() &&
                     find(neighbors[diag.second].begin(), neighbors[diag.second].end(),
border.first) != neighbors[diag.second].end())
            {
                tmp_pair = {border.second, border.first};
                break;
            }
        }
        while (peaks[diag.first] != 0 && peaks[diag.second] != 0)
        {

            result += LETTERS[tmp_pair.first] + LETTERS[tmp_pair.second] + "+\n";
            result += LETTERS[diag.first] + LETTERS[tmp_pair.first] + "-\n";
            result += LETTERS[diag.second] + LETTERS[tmp_pair.second] + "-\n";
            peaks[diag.first]--;
            peaks[diag.second]--;
        }
    }
    for (int i = 0; i < 8; i++)
    {
        if (peaks[i] != 0)
        {
            cout << "IMPOSSIBLE";
            exit(0);
        }
    }
    cout << result;
}
```