Министерство образования и науки Российской Федерации федеральное государственное
автономное образовательное учреждение высшего образования
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Факультет «Программной инженерии и компьютерной техники.»

Алгоритмы и структуры данных

Лабораторная работа №4
Дополнительные задания.

Выполнил
Григорьев Давид Владимирович
Группа: P3215
Преподаватели
Косяков Михаил Сергеевич
Тараканов Денис Сергеевич

Санкт-Петербург 2025

Содержание

1 Дополнительное задание 1.

1.1 Рекуррентное соотношение

Обозначим x_n как количество бактерий после n-го дня. Процесс описывается следующим рекуррентным соотношением:

$$x_{n+1} = \begin{cases} \min(b \cdot x_n - c, d), & \text{если } b \cdot x_n - c \geq 0 \\ 0, & \text{иначе} \end{cases}$$

- **Умножение**: Каждая бактерия делится на b новых ($b \geq 1$).
- **Удаление**: c бактерий уничтожаются.
- **Ограничение**: Оставшиеся бактерии не могут превышать вместимость контейнера d.
- **Завершение**: Если $b \cdot x_n < c$, эксперимент заканчивается ($x_{n+1} = 0$).

1.2 Специальный случай: $b = 1$

Когда $b = 1$, умножение отсутствует. Соотношение упрощается:

$$x_{n+1} = x_n - c$$

Это модель линейного убывания:

- Если $c = 0$: Количество бактерий постоянно ($x_n = \min(a, d)$).
- Если $a < c$: Эксперимент завершается ($x_1 = 0$).
- Иначе: Население уменьшается на c ежедневно.

Максимальное количество дней до завершения:

$$\text{max\_days} = \left\lfloor \frac{a}{c} \right\rfloor$$

- Если $k \leq$ max_days: Результат $x_k = \max(0, a - k \cdot c)$, ограниченный d.
- Иначе: $x_k = 0$.

1.3 Общий случай: $b > 1$

Для $b > 1$ система демонстрирует экспоненциальный рост до стабилизации или завершения.

1.3.1 Условие стабилизации

Если популяция достигает d и выполняется условие:

$$b \cdot d - c \geq d \quad \Rightarrow \quad d(b-1) \geq c,$$

то популяция стабилизируется на уровне d:

$$x_{n+1} = \min(b \cdot d - c, d) = d.$$

Это означает, что система достигла **неподвижной точки**.

1.3.2 Сходимость к d

Если $d(b-1) \geq c$, популяция сходится к d за логарифмическое время:

$$\text{Число шагов} \sim \log_b \left(\frac{d}{a} \right).$$

1.3.3 Условие завершения

Если на любом шаге $b \cdot x_n < c$, популяция становится отрицательной, и процесс завершается ($x_{n+1} = 0$).

1.4 Почему аналитическое решение невозможно

Рекуррентное соотношение содержит:

1. **Нелинейность**: Умножение ($b \cdot x_n$) и вычитание ($-c$).
2. **Пороговые условия**: Ограничение ($\min(\cdot, d)$) вносит разрывы.
3. **Условное завершение**: Процесс останавливается, если $b \cdot x_n < c$.

1.5 Асимптотика

Итого сложность данного алгоритма $O(min(\texttt{days}, \log(\texttt{max\_capacity})))$

```cpp
#include <algorithm>
#include <iostream>

int main() {
  // Input values
  int64_t initial_bacteria = 0;
  int64_t multiplier = 0;
  int64_t removed_per_day = 0;
  int64_t max_capacity = 0;
  int64_t days = 0;

  // Read input
  std::cin >> initial_bacteria >> multiplier >> removed_per_day >> max_capacity >> days;

  // Special case: if bacteria don't multiply (multiplier = 1)
  if (multiplier == 1) {
    if (removed_per_day == 0) {
      // Bacteria count remains constant, just cap at max_capacity
      std::cout << std::min(initial_bacteria, max_capacity);
      return 0;
    }

    // If initial bacteria are not enough for the first day
    if (initial_bacteria < removed_per_day) {
      std::cout << 0;
      return 0;
    }

    // Calculate how many full days the bacteria can survive
    int64_t max_full_days = initial_bacteria / removed_per_day;

    if (days <= max_full_days) {
      // Bacteria still exist after 'days' days
      int64_t remaining = initial_bacteria - (days * removed_per_day);
      std::cout << std::min(remaining, max_capacity);
    } else {
      // All bacteria are gone before the end of 'days'
      std::cout << 0;
```

```cpp
    }
    return 0;
  }

  int64_t current_bacteria = initial_bacteria;
  bool has_stabilized = false;

  for (int64_t day = 0; day < days && !has_stabilized; ++day) {
    // Multiply bacteria and remove the used ones
    int64_t multiplied = current_bacteria * multiplier;
    int64_t after_removal = multiplied - removed_per_day;

    // If bacteria count drops below zero, experiment ends
    if (after_removal < 0) {
      std::cout << 0;
      return 0;
    }

    // Cap at max capacity
    int64_t next_bacteria = std::min(after_removal, max_capacity);

    // If no change occurs, further iterations won't change anything
    if (next_bacteria == current_bacteria) {
      break;
    }

    current_bacteria = next_bacteria;

    // Check if we've reached max capacity and the population is stable
    // This happens when d * (b - 1) >= c
    if (current_bacteria == max_capacity && max_capacity * (multiplier - 1) >= removed_p
      has_stabilized = true;
    }
  }

  // Output the final bacteria count after k days
  std::cout << current_bacteria;
  return 0;
}
```

2 Существование стоков и истоков в DAG

Топологическая сортировка принципиально опирается на свойства DAG. Докажем фундаментальную лемму о существовании начальных и конечных вершин:

1. **Предположим противное**: в DAG нет истоков, т.е. $\forall v \in V \ \deg^-(v) \geq 1$.
2. Выберем произвольную вершину v_1. По предположению $\exists v_2 \neq v_1$ такая, что $(v_2, v_1) \in E$.
3. Для v_2 аналогично $\exists v_3$ с $(v_3, v_2) \in E$, и т.д. Получаем бесконечную регрессию:

$$v_1 \leftarrow v_2 \leftarrow v_3 \leftarrow \dots$$

4. В конечном графе $\exists k < m : v_k = v_m$, что создаёт цикл:

$$v_k \rightarrow v_{k-1} \rightarrow \cdots \rightarrow v_m = v_k$$

Противоречие с ацикличностью DAG.

Аналогичное доказательство для стоков получается обращением ориентации рёбер.

Обобщение на несвязные графы:
Для DAG с несколькими компонентами связности доказательство применяется к каждой компоненте отдельно. Каждая компонента как подграф:

- Сохраняет ацикличность
- Содержит минимум один исток и один сток
- Гарантирует стартовые точки для алгоритмов (например, Кана)

Роль в данном коде

- Реализован в функции `ComputeTopologicalOrdering()`
- Позволяет корректно вычислить самый длинный путь:
  - Обработка вершин в топологическом порядке гарантирует, что все входящие пути учтены
  - Использует массив `longest_path_distances` для хранения промежуточных результатов

3 Зависимость сложности BFS/DFS от представления графа

3.1 Способы представления графов

3.1.1 Матрица смежности

- Хранение: матрица $A \in \{0,1\}^{V \times V}$, где

$$A[i][j] = \begin{cases} 1, & \text{есть ребро } (i \to j) \\ 0, & \text{иначе} \end{cases}$$

- Память: $O(V^2)$
- Поиск соседей для v: $O(V)$
- Сложность BFS/DFS: $O(V^2)$

3.1.2 Список смежности

- Хранение: массив списков $\mathrm{Adj}[V]$, где $\mathrm{Adj}[v]$ содержит соседей v
- Память: $O(V + E)$
- Поиск соседей для v: $O(1)$ на соседа
- Сложность BFS/DFS: $O(V + E)$

3.1.3 Список рёбер

- Хранение: список пар (u, v)
- Память: $O(E)$
- Поиск соседей для v: $O(E)$
- Сложность BFS/DFS: $O(V \cdot E)$

3.1.4 Матрица инцидентности

- Хранение: матрица $B \in \{0,1\}^{V \times E}$, где

$$B[i][j] = \begin{cases} 1, & \text{вершина } i \text{ инцидентна ребру } j \\ 0, & \text{иначе} \end{cases}$$

- Память: $O(V \cdot E)$
- Поиск соседей для v: $O(E)$
- Сложность BFS/DFS: $O(V \cdot E)$

3.2 Сравнение представлений

Представление	Время BFS/DFS	Память
Матрица смежности	$O(V^2)$	$O(V^2)$
Список смежности	$O(V + E)$	$O(V + E)$
Список рёбер	$O(V \cdot E)$	$O(E)$
Матрица инцидентности	$O(V \cdot E)$	$O(V \cdot E)$

4 Обзор алгоритмов поиска кратчайшего пути

4.1 Поиск в ширину (BFS)

- **Принцип работы**: Обход графа послойно от начальной вершины. Исследование всех соседей на каждом шаге. Гарантирует кратчайший путь в невзвешенных графах. Реализация через очередь (FIFO).
- **Примеры**: Поиск пути в лабиринте, анализ социальных сетей.
- **Сложность**:
  - Время: $O(V + E)$ (списки смежности), $O(V^2)$ (матрица смежности)
  - Память: $O(V)$

4.2 Поиск в глубину (DFS)

- **Принцип работы**: Исследование пути до конца с последующим возвратом. Использует стек (LIFO) или рекурсию. Не гарантирует кратчайший путь.
- **Примеры**: Проверка связности, поиск циклов.
- **Сложность**:
  - Время: $O(V + E)$ (списки смежности), $O(V^2)$ (матрица смежности)
  - Память: $O(V)$

4.3 Алгоритм Дейкстры

- **Принцип работы**: Жадный алгоритм для графов с неотрицательными весами. Выбор вершины с минимальным расстоянием и обновление соседей.
- **Примеры**: Маршрутизация в GPS.
- **Сложность**:
  - Наивная реализация: $O(V^2)$
  - С приоритетной очередью: $O((E + V) \log V)$
  - С фибоначчиевой кучей: $O(E + V \log V)$

4.4 Алгоритм A*

- **Принцип работы**: Модификация Дейкстры с эвристической функцией $h(x)$. Требует допустимой эвристики.
- **Примеры**: Навигация в играх, планирование пути роботов.
- **Сложность**:
  - Время: $O(E + V \log V)$ (при оптимальной эвристике)
  - Память: $O(V)$

4.5 Алгоритм Беллмана-Форда

- **Принцип работы**: Обрабатывает графы с отрицательными весами (без циклов отрицательной стоимости). Выполняет $V - 1$ релаксаций рёбер.
- **Примеры**: Финансовые модели с учётом долгов.
- **Сложность**:
  - Время: $O(V \cdot E)$

4.6 Алгоритм Флойда-Уоршелла

- **Принцип работы**: Находит кратчайшие пути между всеми парами вершин. Использует динамическое программирование.
- **Примеры**: Анализ транспортных сетей.
- **Сложность**:
  - Время: $O(V^3)$
  - Память: $O(V^2)$

Сравнение алгоритмов

Алгоритм	Тип графа	Время	Применение
BFS	Невзвешенный	$O(V + E)$	Кратчайший путь, связность
Дейкстра	Неотрицательные веса	$O((E+V)\log V)$	GPS, сети
A*	Неотрицательные + эвристика	$O(E + V \log V)$	Игры, робототехника
Беллман-Форд	Отрицательные веса	$O(V \cdot E)$	Финансы, обнаружение циклов
Флойд-Уоршелл	Все пары вершин	$O(V^3)$	Плотные графы, сетевой анализ

minted

5 Примеры реализации алгоритмов

5.1 Алгоритм Дейкстры

```cpp
#include <vector>
#include <queue>
#include <limits>

using namespace std;

vector<int> dijkstra(int start, vector<vector<pair<int, int>>> adj) {
    int n = adj.size();
    vector<int> dist(n, numeric_limits<int>::max());
    dist[start] = 0;

    priority_queue<pair<int, int>, vector<pair<int, int>>,
                   greater<pair<int, int>>> pq;
    pq.push({0, start});

    while (!pq.empty()) {
        int u = pq.top().second;
        int d = pq.top().first;
        pq.pop();

        if (d > dist[u]) continue; // Устаревший путь

        for (auto edge : adj[u]) {
            int v = edge.first;
            int weight = edge.second;
```

```
            if (dist[u] + weight < dist[v]) {
                dist[v] = dist[u] + weight;
                pq.push({dist[v], v});
            }
        }
    }
    return dist;
}
```

5.2 Поиск в ширину (BFS)

```
#include <vector>
#include <queue>

using namespace std;

vector<int> bfs(int start, vector<vector<int>> adj) {
    int n = adj.size();
    vector<int> dist(n, -1);
    queue<int> q;

    dist[start] = 0;
    q.push(start);

    while (!q.empty()) {
        int u = q.front(); q.pop();
        for (int v : adj[u]) {
            if (dist[v] == -1) {
                dist[v] = dist[u] + 1;
                q.push(v);
            }
        }
    }
    return dist;
}
```

5.3 Алгоритм Беллмана-Форда

```
#include <vector>
#include <limits>

using namespace std;

vector<int> bellman_ford(int start, int n, vector<vector<int>> edges) {
    vector<int> dist(n, numeric_limits<int>::max());
    dist[start] = 0;

    for (int i = 0; i < n-1; ++i) {
        for (auto e : edges) {
            int u = e[0], v = e[1], w = e[2];
            if (dist[u] != numeric_limits<int>::max() &&
                dist[u] + w < dist[v]) {
                dist[v] = dist[u] + w;
            }
        }
    }
```

```
    return dist;
}
```

6 Причины неприменимости алгоритма Дейкстры в 1450

1. **Целевая задача**:
    - Алгоритм Дейкстры предназначен для поиска **кратчайших путей** в графах с **неотрицательными весами**
    - Для поиска **самого длинного пути** требуется принципиально иной подход
2. **Проблема жадного выбора**:
    - Дейкстра предполагает, что найденный кратчайший путь до вершины уже оптимален
    - Для длинных путей это неверно: добавление новых рёбер может увеличить длину пути

7 Обход дерева в 1329

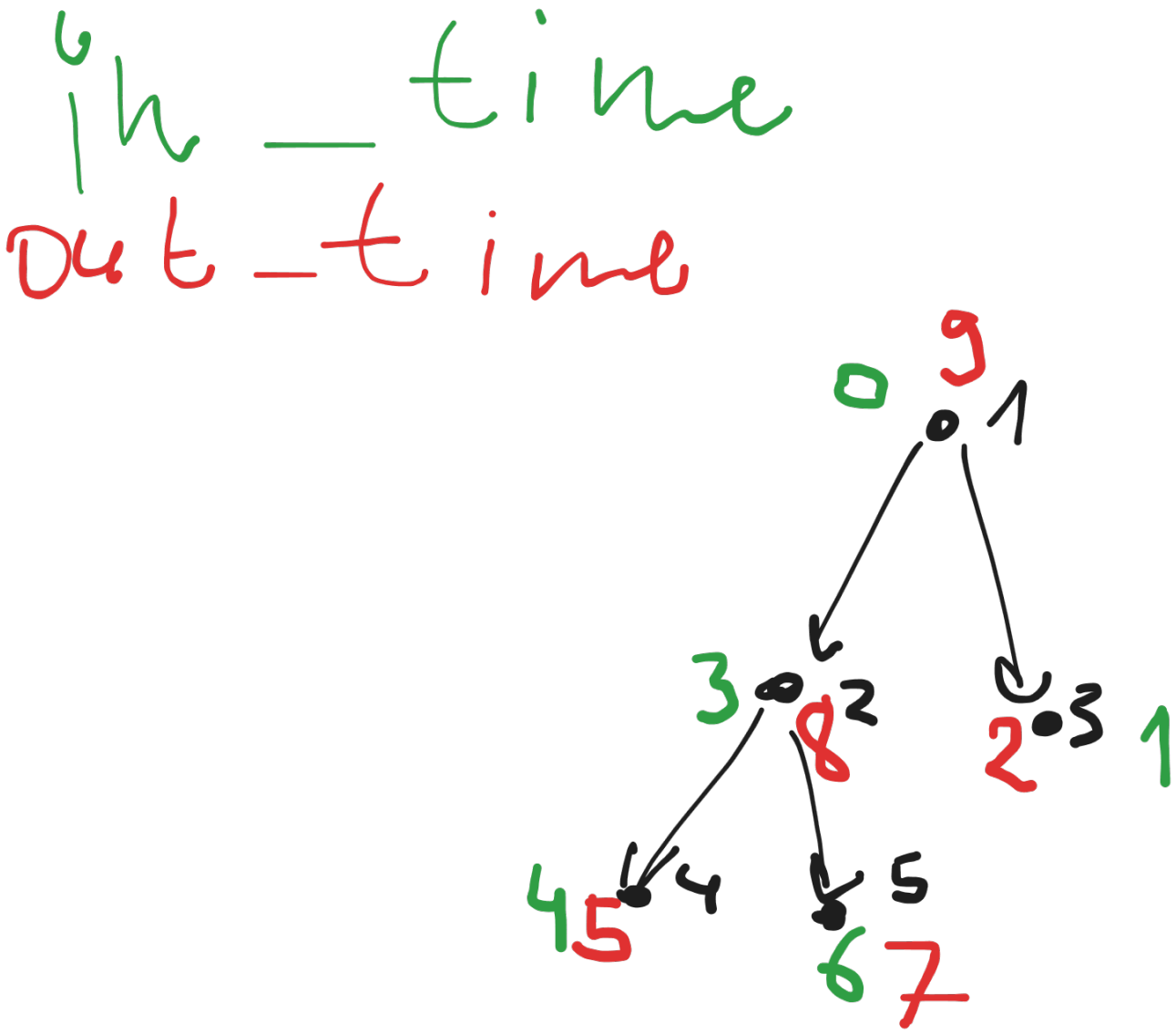

Рис. 1: Обход дерева в 1329