Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

НАПРАВЛЕНИЕ ПОДГОТОВКИ «09.03.04 Программная инженерия»

ОТЧЕТ

ПО ЛАБОРАТОРНОЙ РАБОТЕ №4

Название: Распараллеливание алгоритма DBSAN

Дисциплина: Анализ алгоритмов

Студент	ИУ7-56Б		Ковель А.Д.
	Группа	Подпись, дата	И. О. Фамилия
Преподаватель			Волкова Л.Л.
Преподаватель			Строганов Ю.В.
		Подпись, дата	И. О. Фамилия

Москва, 2022 г.

Оглавление

1 Выполнение задания

В этом разделе будут представлены код реализации алгоритма и его графовые представления.

1.1 Средства реализации

В данной работе для реализации был выбран язык программирования [1].

1.2 Программный код

На листинге 1.1 демонстрируется реализация простого алгоритма DBSCAN.

Листинг 1.1 – Реализация простого алгоритма DBSCAN

```
pub fn dbscan(points: &Vec<Vec<bool>>, min_ptx: usize, eps: f64,
    mut imgbuf: RgbImage) -> u32 {

  let mut cluster_count = 0;                              // 1
  let mut current_point = points.clone();                 // 2
  for i in 0..points.len() {                              // 3
    for j in 0..points[i].len() {                         // 4
      if current_point[i][j] {                            // 5

        let mut v = Vec::<[usize; 2]>::new();             // 6
        v.push([i, j]);                                   // 7
        let mut neighbor_count_check = 0;                 // 8
        while !v.is_empty() {                             // 9
          let p = v.pop().unwrap();                       // 10
          if !current_point[p[0]][p[1]] {                 // 11
            continue;                                     // 12
          }
          current_point[p[0]][p[1]] = false;              // 13

          let mut neighbor_count = 0;                     // 14

          regionquery(points, min_ptx, &mut v, &mut
             neighbor_count, p, eps);                     // 15

          if neighbor_count >= min_ptx {                  // 16
            neighbor_count_check += 1;                    // 17
          }
        }
        if neighbor_count_check > 0 {                     // 18
          cluster_count += 1;                             // 19
        }
        current_point[i][j] = false;                      // 20
      }
    }
  }

  cluster_count
}
```

1.3 Графовые представления

На рисунках 1.1, 1.2, 1.3, 1.4 представлен операционный граф, информационный граф, граф операционной истории, граф информационной истории.

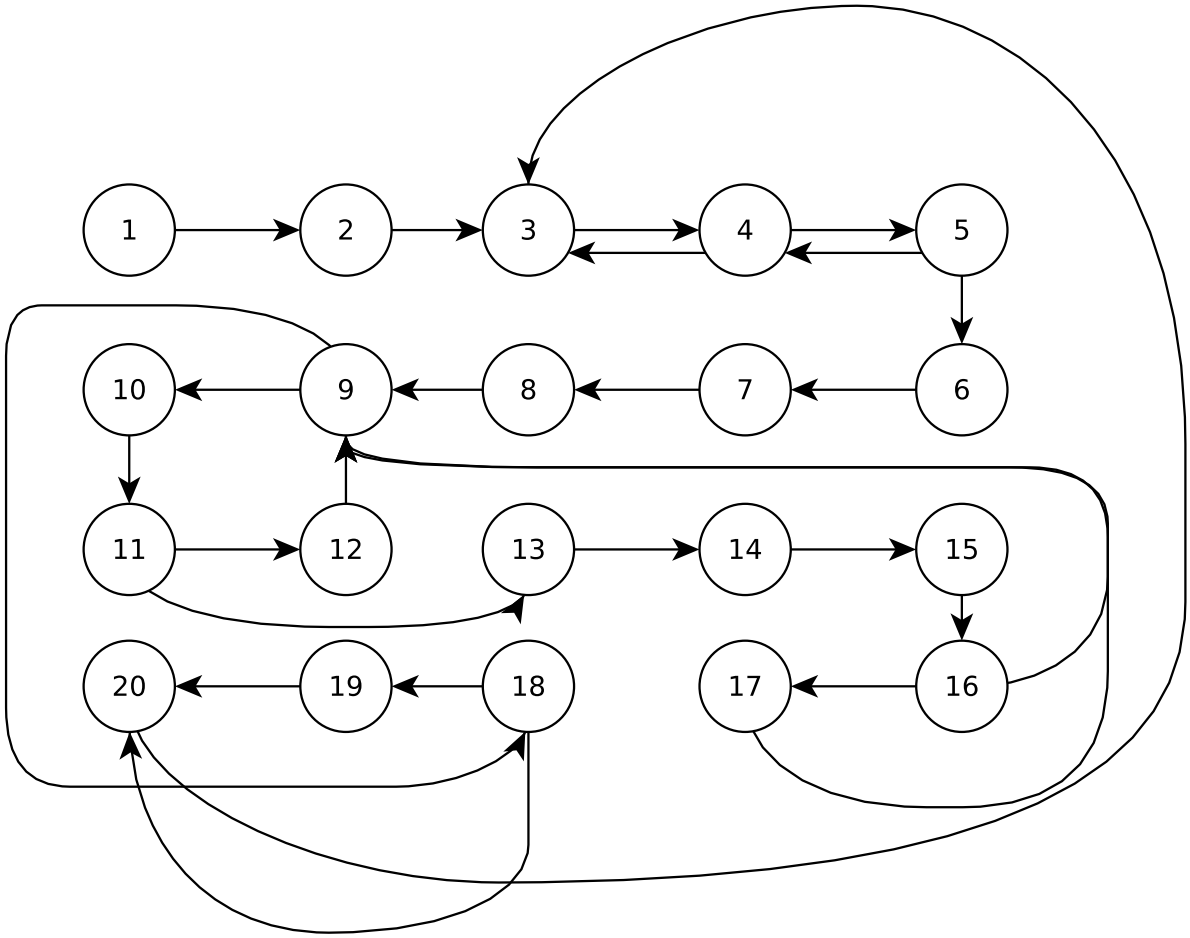

Рисунок 1.1 – Операционный граф

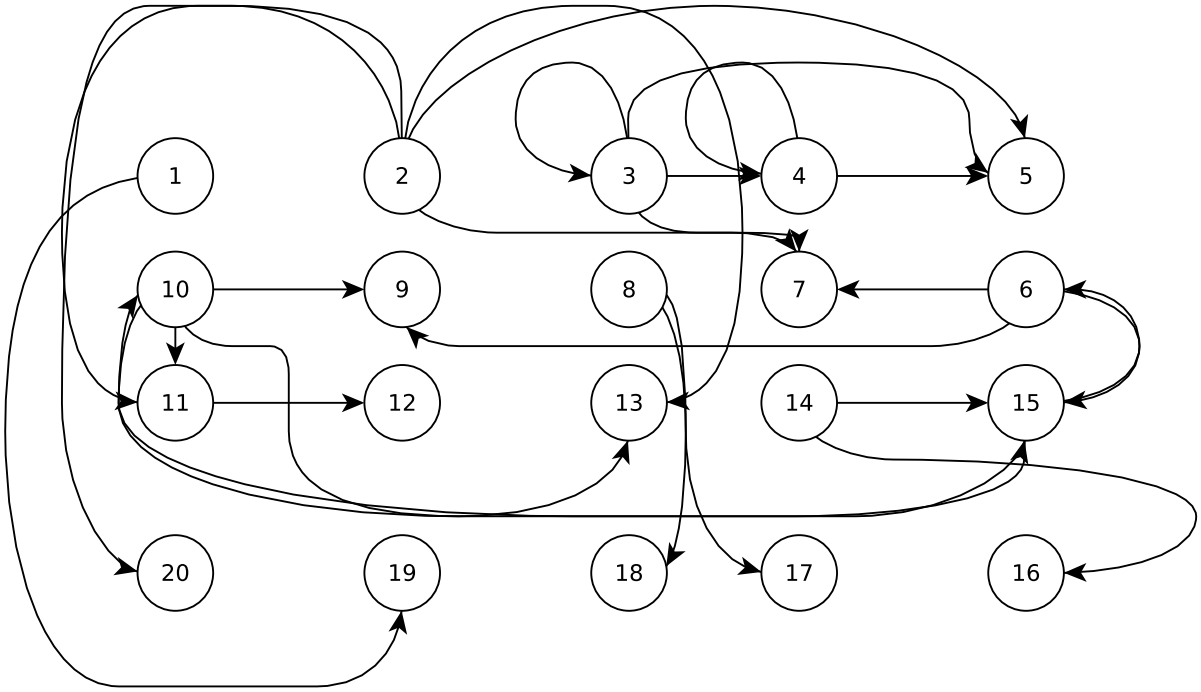

Рисунок 1.2 – Информационный граф

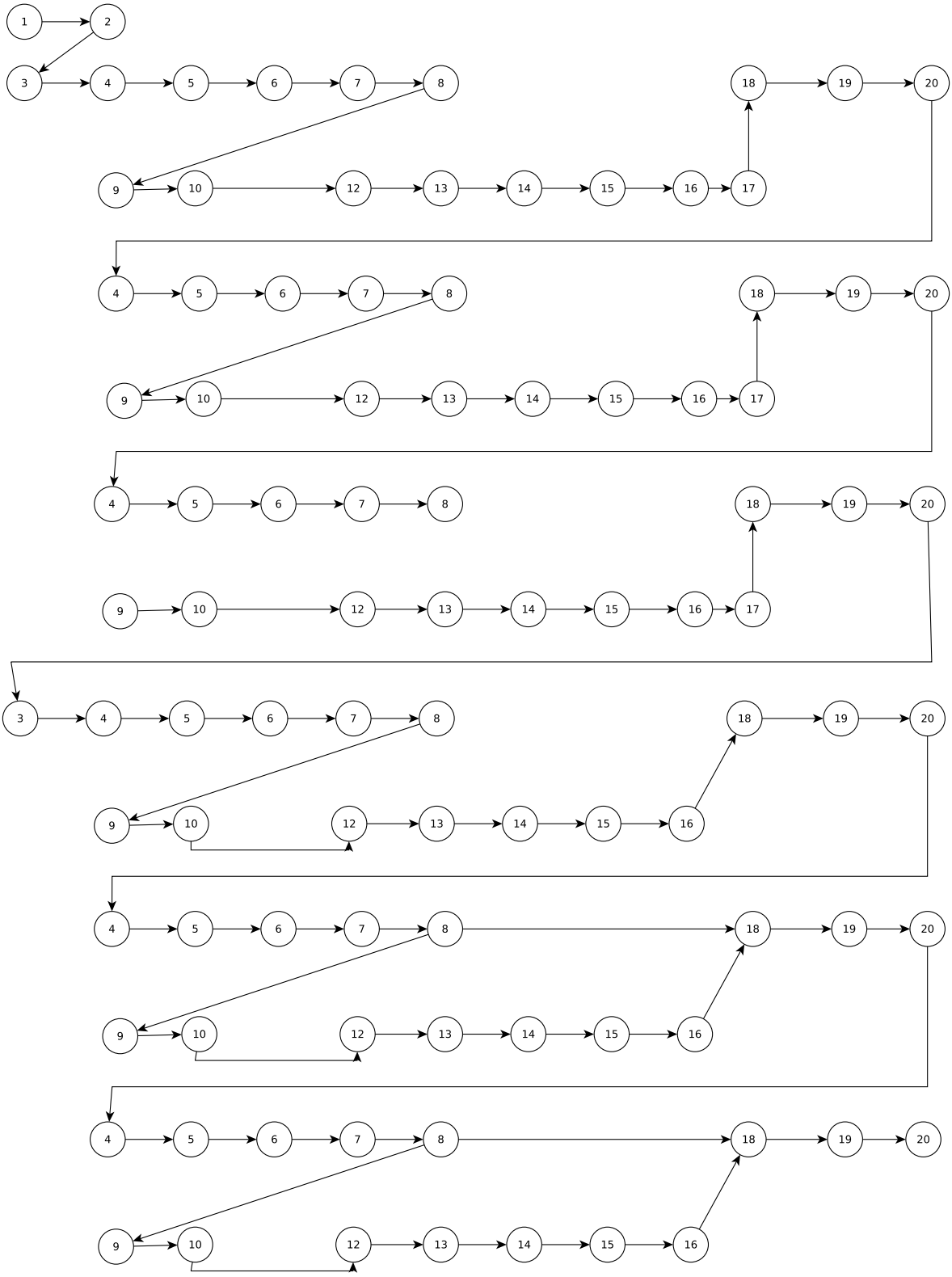

Рисунок 1.3 – Граф операционной истории

Рисунок 1.4 – Граф информационной истории

Список использованных источников

[1] Rust Документация [Электронный ресурс]. Режим доступа: `https://www.rust-lang.org/learn` (дата обращения: 28.11.2022).