ДВФУ
ДАЛЬНЕВОСТОЧНЫЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ

Планарная укладка графа.

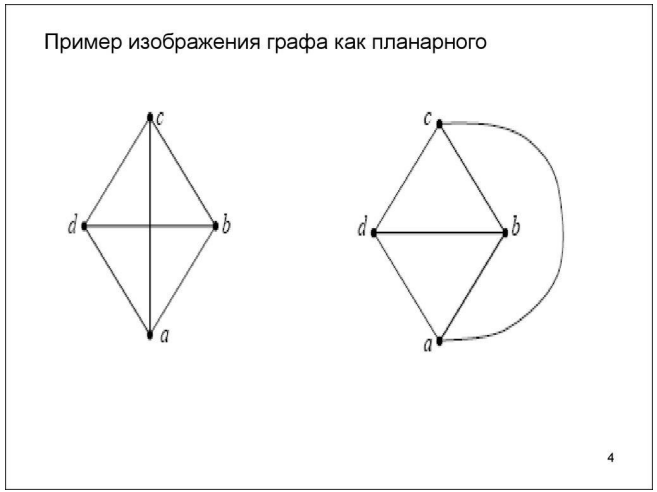
Гамма-алгоритм

Снетков Данила

Б8118-01.03.02 систпро

Планарный граф.

Это граф который можно изобразить на плоскости так, чтобы ребра графа не пересекались с друг другом.



Гамма-алгоритм. Входные данные.

На вход алгоритму подаются графы со следующими свойствами:

1. Граф связный.
2. Граф содержит хотя бы один цикл.
3. Граф не имеет мостов.

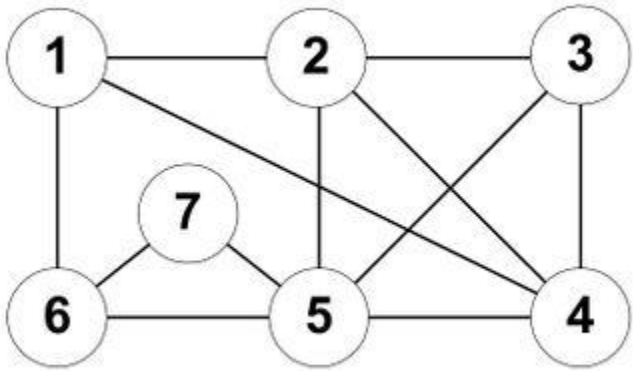
Если нарушено свойство 1, то граф нужно укладывать отдельно по компонентам связности. Если нарушено свойство 2, то граф — дерево и нарисовать его плоскую укладку тривиально.

Гамма-алгоритм.Описание алгоритма.

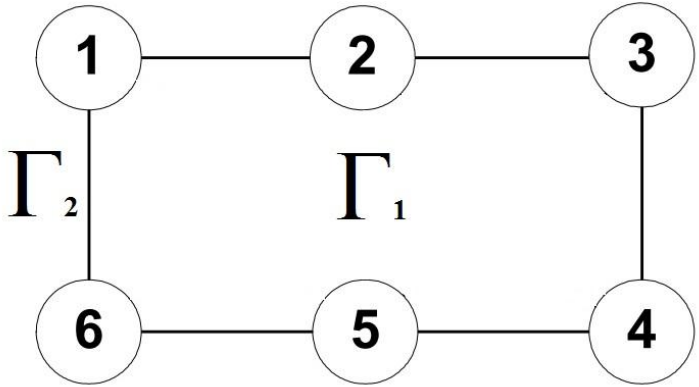
1. Находим в графе любой простой цикл и производим его укладку на плоскость.После его укладки получаем две грани.
2. Далее находим все **сегменты** графа. **Сегмент** графа - это путь с двумя **контактными** вершинами на его концах. **Контактные** вершины принадлежат либо циклу либо ранее найденным **сегментам.**
3. Затем, на каждой шаге выбираем сегмент с минимальным числом общих граней для **контактных** вершин **сегмента**. Если это число равно нулю, то граф не планарен, иначе производим укладку цепи в грань, это грань разделится на две.
4. По укладке всех **сегментов** будет получена плоская укладка графа, или граф окажется не планарен.

Укладка цикла на плоскость.

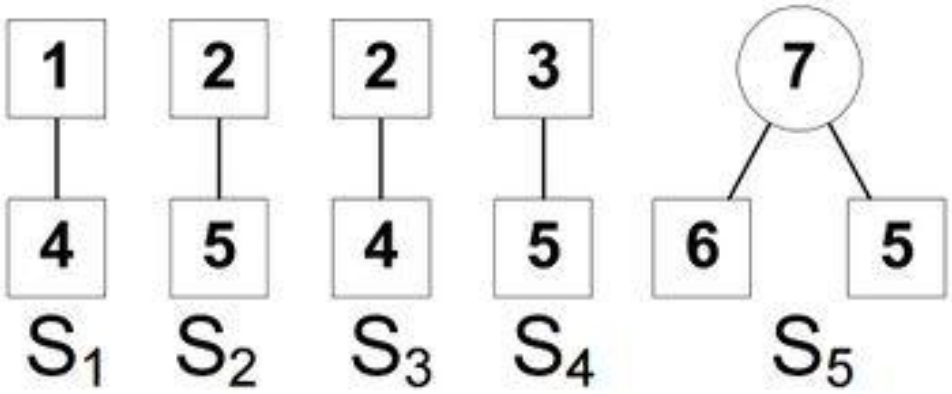
Исходный граф:



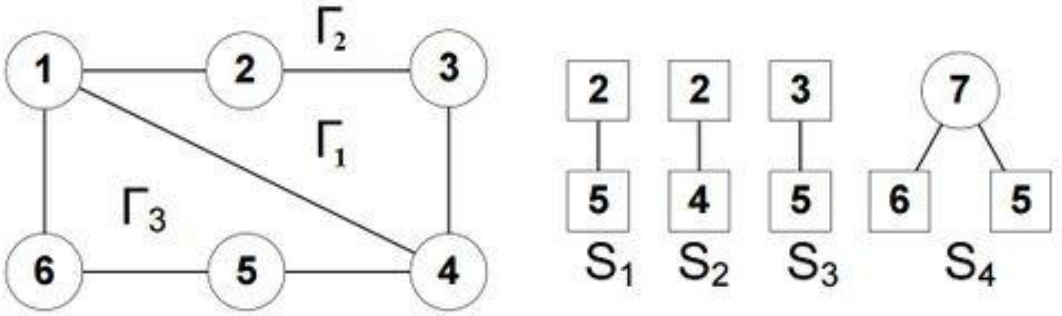
Пример укладки цикла



Пример выделения сегментов



Пример шага алгоритма.



Детали реализации. Структуры данных.

```
struct point {
  double x, y;
  std::set<int> faces = {0, 1};
  Color color = white;
  std::vector<int> edges_ind;
  std::vector<int> drawn_edges_ind;
  bool is_in_cycle = false;
  Point_type point_type = standard;
struct section {
  //using vertex_iter = std::vector<point>::iterator;
  //using edge_iter = std::vector<edge>::iterator;

  std::vector<int> vertexes;
  std::vector<int> edges;
  int com_faces_num = 2;
  std::set<int> com_faces{0, 1};
```

```
class vertexes_in_face: public std::vector<int> {
      public:
         int &operator [](int i) {
             i = i < 0 ? i * (-1) - 1 : i;
             i = i >= this->size() ? i - (int)this->size() : i;
             return std::vector<int>::operator[](i);
         }
};
struct face {
  vertexes_in_face vertexes;
  std::unordered_set<int> edges;
```

```
struct edge {
  int faces_ind[2] = {0, 1};
  int leads[2];
  bool is_in_cycle = false;
```

Детали реализации. Использованные алгоритмы.

- Для поиска цикла, а также для поиска **сегментов** использовался dfs.
- Если при очередной вставке **сегмента** между контактными вершинами отсутствовал прямой путь (прямая пересекает рёбра), то для поиска пути использовалось следующие консистентное свойство: все вершины грани уложены против часовой стрелки, путь между двумя контактными вершинами также ведется против часовой стрелки “вдоль” пути, по вершинам грани (новый путь располагается близко к существующем вершинам грани).

Примеры работы алгоритма.