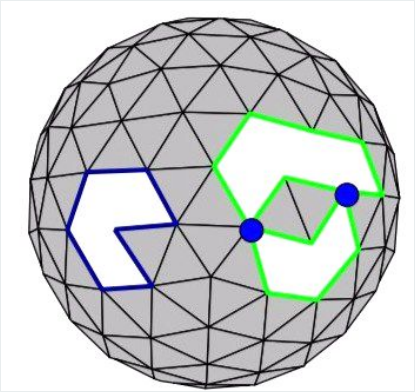
Представление триангуляции в памяти



Содержание

- Формула Эйлера для планарных графов и ее следствия
- Операции над объектами триангуляции
- Структуры для представления триангуляции
  - Узлы с соседями
  - Узлы, рёбра и треугольники
  - Узлы и треугольники
  - Легкие узлы и треугольники
  - Двойные рёбра
- Резюме
- Домашнее задание

Формула Эйлера

Для произвольного планарного графа G с V вершинами, E ребрами и F гранями справедливо следующее соотношение:

$$V - E + F = 2$$

Формула Эйлера

Для произвольного планарного графа G с V вершинами, E ребрами и F гранями справедливо следующее соотношение:

$$V - E + F = 2$$

Доказательство?

Формула Эйлера

Для произвольного планарного графа G с V вершинами, E ребрами и F гранями справедливо следующее соотношение:

$$V - E + F = 2$$

Доказательство:
Индукцией по количеству граней графа

Формула Эйлера

Следствия:

Пусть G связный планарный обыкновенный граф с V вершинами (V >=3), E ребрами и F гранями. Тогда:

- E <= 3V - 6
- F <= 2V - 4
- Средняя степень вершины в планарном графе равна 6, такое же количество и инцидентных треугольников для вершины

Операции над объектами

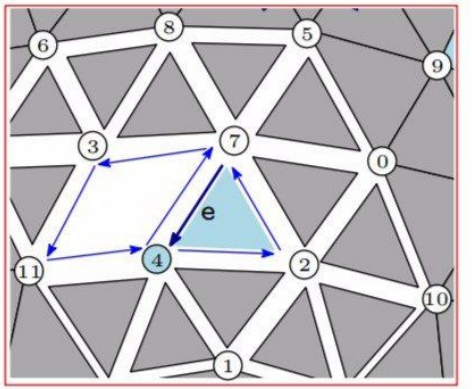
В триангуляции 3 основных вида объектов: узлы, рёбра и треугольники.

Часто требуются следующие операции с объектами триангуляции:

- Треугольник → узлы
- Треугольник → рёбра
- Треугольник → треугольники
- Ребро → узлы
- Ребро → треугольники
- Узел → рёбра
- Узел → треугольники

Структуры для представления триангуляции

Обозначения:
V - количество вершин
E - количество рёбер
F - количество треугольников



Узлы с соседями

Для каждого узла хранятся координаты и список указателей на соседние узлы (список номеров узлов), с которыми есть общие рёбра.

```
class NodeWithNeighbours:
    def __init__(self, idr: int, p: Point, neigh_nodes: List['NodeWithNeighbours']):
        self.p = p
        self.neigh_nodes = neigh_nodes
```

Узлы с соседями

Операция узел → треугольники

По узлу происходит выдача списка из списков узлов, составляющие соседние треугольники:

- рассмотрим всевозможные сочетания соседних узлов для узла self
- если в текущей взятой паре (i, j) выполняется условие, что j является соседом для i, то образуем треугольник из узлов [i, j, self]

Узлы с соседями

Для каждого узла хранятся координаты и список указателей на соседние узлы, с которыми есть общие рёбра.

```
class NodeWithNeighbours:
    def __init__(self, idr: int, p: Point, neigh_nodes: List['NodeWithNeighbours']):
        self.p = p
        self.neigh_nodes = neigh_nodes
```

Оценка памяти:
Каждое ребро учитывается дважды. На каждую вершину Point(x и y) и соседи.

Точная оценка: **2E + 2V**

Узлы с соседями

Для каждого узла хранятся координаты и список указателей на соседние узлы, с которыми есть общие рёбра.

```
class NodeWithNeighbours:
    def __init__(self, idr: int, p: Point, neigh_nodes: List['NodeWithNeighbours']):
        self.p = p
        self.neigh_nodes = neigh_nodes
```

Память: 2E + 2V

- Легко реализовать
- Компактно

Узлы с соседями

Для каждого узла хранятся координаты и список указателей на соседние узлы, с которыми есть общие рёбра.

```
class NodeWithNeighbours:
    def __init__(self, idr: int, p: Point, neigh_nodes: List['NodeWithNeighbours']):
        self.p = p
        self.neigh_nodes = neigh_nodes
```

Память: 2E + 2V

- Легко реализовать
- Компактно
- Представляются только узлы

Узлы, рёбра и треугольники

Для каждого ребра хранятся указатели на два концевых узла и два соседних треугольника. Для треугольников хранятся указатели на три образующих треугольник ребра. В каждом узле координаты, инцидентные ребра и инцидентные треугольники.

```
class NodesAndEdgesAndTriangles:
    class Node:
        def __init__(self, p: Point, edges: List['Edge'], triangles: List['Triangle']):
            self.p = p
            self.edges = edges
            self.triangles = triangles
    class Edge:
        def __init__(self, nodes: List['Node'], triangles: List['Triangle']):
            self.nodes = nodes
            self.triangles = triangles
    class Triangle:
        def __init__(self, edges: List['Triangle']):
            self.edges = edges
```

Узлы, рёбра и треугольники

Операция треугольник → треугольники

По треугольнику происходит выдача списка соседних треугольников:

- проитерируемся по инцидентным рёбрам для треугольника self для i-того ребра
- получаем треугольник из списка соседних треугольников, исключая self

Узлы, рёбра и треугольники

Для каждого ребра хранятся указатели на два концевых узла и два соседних треугольника. Для треугольников хранятся указатели на три образующих треугольник ребра. В каждом узле координаты, инцидентные ребра и инцидентные треугольники.

```
class NodesAndEdgesAndTriangles:
    class Node:
        def __init__(self, p: Point, edges: List['Edge'], triangles: List['Triangle']):
            self.p = p
            self.edges = edges
            self.triangles = triangles
    class Edge:
        def __init__(self, nodes: List['Node'], triangles: List['Triangle']):
            self.nodes = nodes
            self.triangles = triangles
    class Triangle:
        def __init__(self, edges: List['Triangle']):
            self.edges = edges
```

Оценка памяти:
Один узел требует 14 ссылок: 2 (Point) + 6 треугольников (среднее количество инцидентных треугольников) + 6 рёбер (среднее количество узлов).
Одно ребро требует 4 ссылки: 2 (концы) + 2 (у одного ребра два инцидентных треугольника).
Один треугольник требует 3 ссылки на ребра треугольника.
Получаем верхнюю оценку:
14V + 4E + 3F

Узлы, рёбра и треугольники

Для каждого ребра хранятся указатели на два концевых узла и два соседних треугольника. Для треугольников хранятся указатели на три образующих треугольник ребра. В каждом узле координаты, инцидентные ребра и инцидентные треугольники.

```
class NodesAndEdgesAndTriangles:
    class Node:
        def __init__(self, p: Point, edges: List['Edge'], triangles: List['Triangle']):
            self.p = p
            self.edges = edges
            self.triangles = triangles
    class Edge:
        def __init__(self, nodes: List['Node'], triangles: List['Triangle']):
            self.nodes = nodes
            self.triangles = triangles
    class Triangle:
        def __init__(self, edges: List['Triangle']):
            self.edges = edges
```

Память:
9V + 5E + 4F

- Легко реализовать
- Много памяти

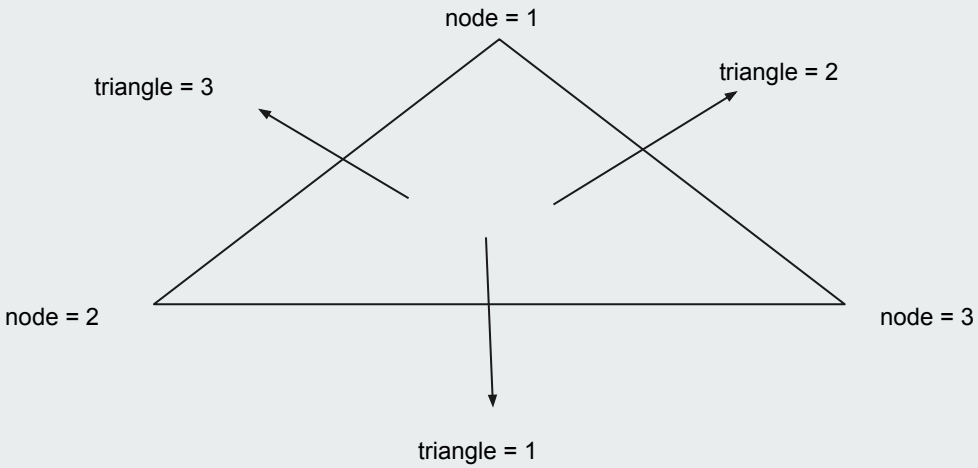
Узлы и треугольники

Для каждого треугольника хранятся три указателя на образующие его узлы и три указателя на смежные треугольники, а каждый узел хранит указатели на инцидентные ему треугольники.

```
class NodesAndTriangles:
    class Node:
        def __init__(self, p: Point, triangles: List['Triangle']):
            self.p = p
            self.triangles = triangles
    class Triangle:
        def __init__(self, nodes: List['Node'], triangles: List['Triangle']):
            self.nodes = nodes
            self.triangles = triangles
```

Узлы и треугольники

Нумерация узлов против часовой стрелки, напротив i узла находится i треугольник



Узлы и треугольники

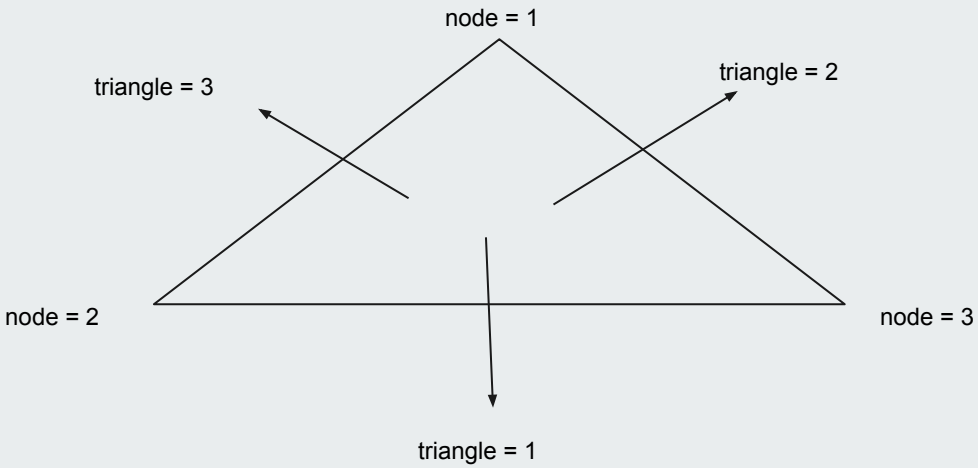
Операция узел → рёбра

По узлу происходит выдача списка из списков узлов, составляющие соседние рёбера:

- проитерируемся по инцидентным треугольникам для узла self
- для i-того треугольника образуем рёбра вида [self, el], где el — это один из соседов i, исключая self

Узлы и треугольники

Нумерация узлов против часовой стрелки, напротив i узла находится i треугольник

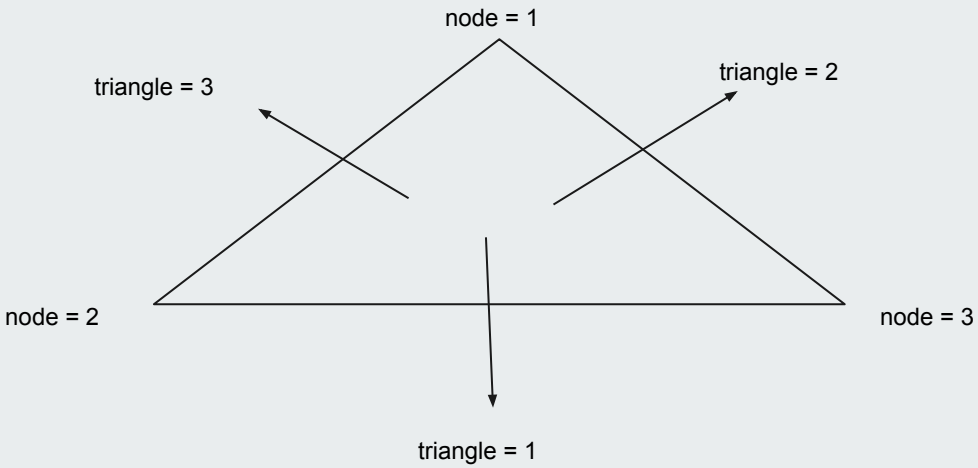


Оценка памяти:
Один узел требует 8 ссылок:
2 (Point) + 6 (среднее количество инцидентных треугольников).
Один треугольник требует 6 ссылок: 3 (вершины треугольника) + 3 (соседи).

Получаем верхнюю оценку:
8V + 6F

Узлы и треугольники

Нумерация узлов против часовой стрелки, напротив i узла находится i треугольник



Память:
8V + 6F

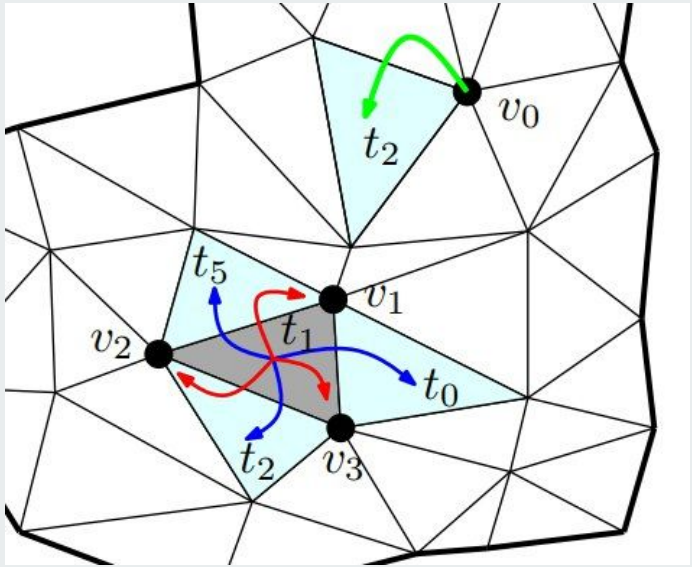
- Легко реализовать
- Компактно
- Удобно
- А можно лучше?

Легкие узлы и треугольники

Для каждого треугольника хранятся три указателя на образующие его узлы и три указателя на смежные треугольники, а каждый узел хранит только один указатель на инцидентный ему треугольник.

```
class LightNodesAndTriangles:
    class Node:
        def __init__(self, p: Point, triangle: 'Triangle'):
            self.p = p
            self.triangle = triangle
    class Triangle:
        def __init__(self, idr: int, nodes: List['Node'], triangles: List['Triangle']):
            self.nodes = nodes
            self.triangles = triangles
```

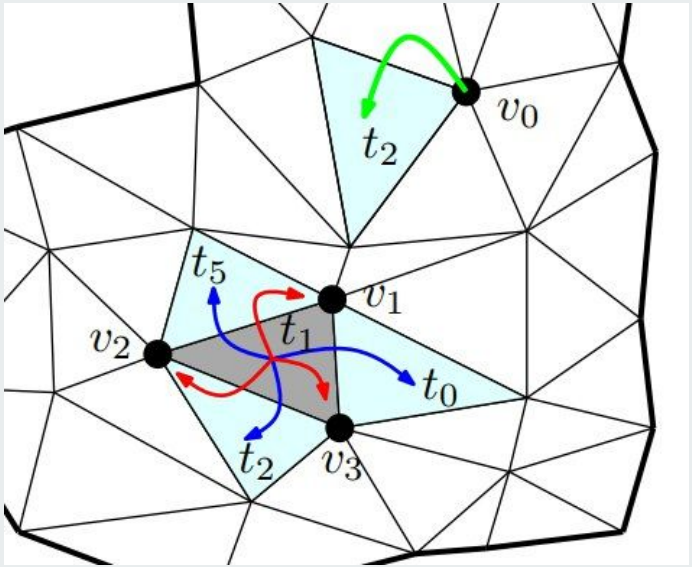
Легкие узлы и треугольники



Операция узел → треугольники
По узлу происходит выдача всех соседних треугольников:

- получим по узлу self инцидентный узлу треугольник (current_triangle, start_triangle) добавим его в итоговый список
- найдем индекс по которому находится self в current_triangle.node и берем следующий индекс (index), берём current_triangle.triangles[index], добавляем этот треугольник в итоговый список. Повторяем, пока current_trinagle != start_triangle
- если же в какой-то момент треугольника напротив узла не окажется (None), то прервать проход против часовой и сделать аналогичный цикл, в котором уже брать предыдущий индекс,

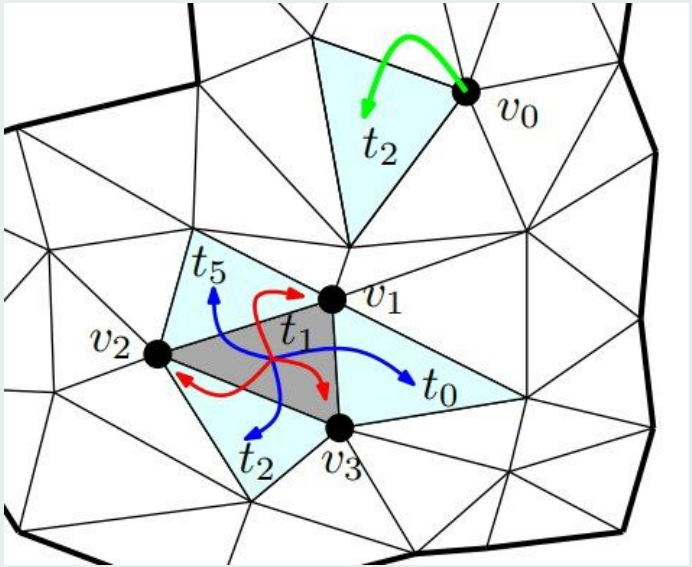
Легкие узлы и треугольники



Оценка памяти:
Один узел требует 3 ссылки:
2 (Point) + 1 (любой инцидентный треугольник).
Один треугольник требует 6 ссылки:
3 (составляющие треугольник вершины) +
3 (соседние треугольники).

Получаем верхнюю оценку: **3V + 6F**

Легкие узлы и треугольники



Память: 3V + 6F

- Довольно компактно
- Используется на практике (CGAL)

Что еще используется на практике?

Двойные ребра

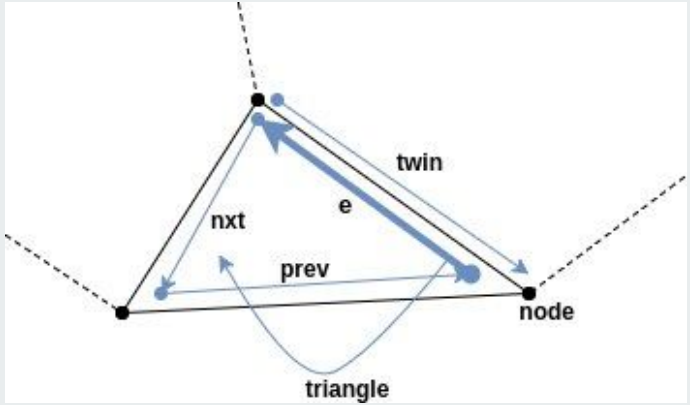
```
class DoubleEdges:
    class Node:
        def __init__(self, p: Point, he: 'HalfEdge'):
            self.p = p
            self.he = he
    class HalfEdge:
        def __init__(self, node: 'HalfEdge', prev: 'HalfEdge', nxt:
'HalfEdge', twin: 'HalfEdge', triangle: 'Triangle'):
            self.node = node
            self.prev = prev
            self.nxt = nxt
            self.twin = twin
            self.triangle = triangle
    class Triangle:
        def __init__(self, he: 'HalfEdge'):
            self.he = he
```

Вершина хранит в себе координаты узла и ссылку на любое инцидентное полуребро, выходящее из этого узла.

Полуребро хранит в себе ссылку на выходящий узел, ссылки на обратное, следующее и предыдущее ребра в порядке обхода треугольника, а также ссылку на инцидентный треугольник.

Треугольник хранит в себе любое полуребро, составляющее его границу.

Двойные ребра

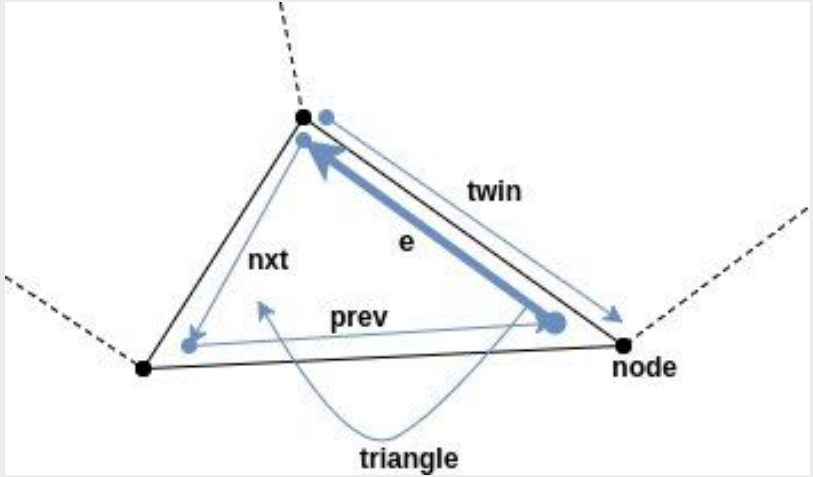


Операция узел → треугольники

По узлу происходит выдача соседних треугольников:

- зафиксируем инцидентное полуребро (current_he), выходящее из self, и инцидентный треугольник для этого полуребра (current_triangle, start_triangle)
- будем двигаться по соседним треугольнкам против часовой стрелки: добавим current_triangle в итоговый список, затем возьмем current_he.prev, если у него существует близнец, то он теперь и current_he, а треугольник от него станет current_triangle. Повторяем, пока current_trinagle != start_triangle
- если же в какой-то момент близнеца не окажется, то прервать проход против часовой и сделать аналогичный цикл по часовой стрелке для полуребра изначально данной вершины

Двойные ребра



Оценка памяти:
Один узел требует 3 ссылки:
2 (Point) + 1 (полурёбро).

Одно полуребро требует 5 ссылок:
1 (выходящая вершина) + 1 (треугольник) + 3 (prev, next, twin полурёбра), а на каждое ребро есть два полуребра, отсюда x2.

Один треугольник требует 1 ссылку любого инцидентного полуребра.

Получаем верхнюю оценку: **3V + 10E + F**

Резюме

Название структуры данных	Узлы	Рёбра	Треугольники	Оценка памяти	Оценка памяти в V
"Узлы с соседями"	+	-	-	2V + 2E	8V
"Узлы и треугольники"	+	-	+	8V + 6F	20V
"Узлы, рёбра и треугольники"	+	+	+	14V + 4E + 3F	32V
"Двойные рёбра"	+	+	+	3V + 10E + F	35V
"Лёгкие узлы и треугольники"	+	-	+	3V + 6F	15V

Домашнее задание

- Прочитать про структуры в конспекте
- Ознакомиться с реализацией задачи “Walking in triangulation” для рассмотренных структур
- Реализовать алгоритм “Walking in triangulation” для структуры “Легкие узлы и треугольники”