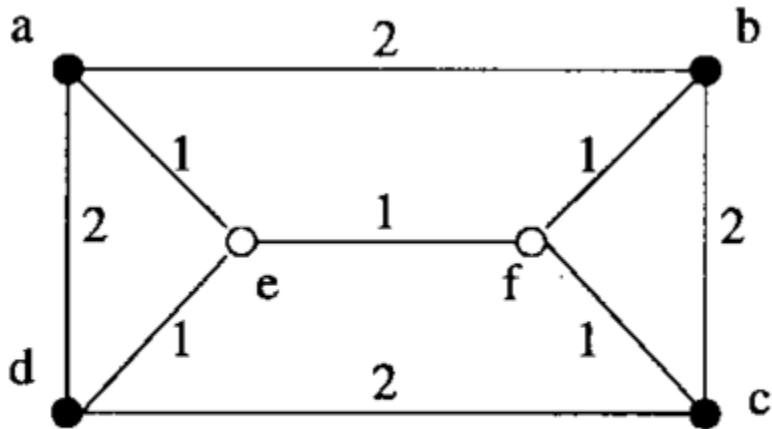
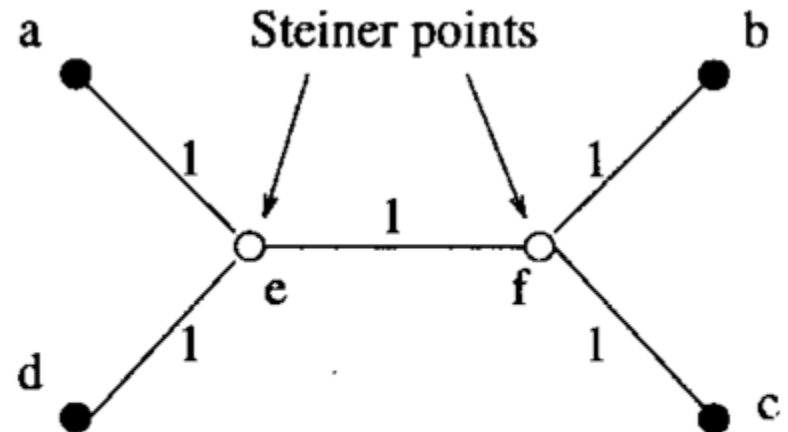


# ЗАДАЧА ШТЕЙНЕРА

## О МИНИМАЛЬНОМ ДЕРЕВЕ



Original graph



Optimal Steiner tree

# СОДЕРЖАНИЕ

## 1. ЗАДАЧА -----

1. ПРИМЕР УСЛОВИЯ
2. ОПИСАНИЕ
3. ОСОБЕННОСТИ

## 2. РЕШЕНИЕ -----

1. ТРИАНГУЛЯЦИЯ ДЕЛОНЕ
2. ТРИАНГУЛЯЦИЯ ДЕЛОНЕ: ПСЕВДО-КОД
3. ТОЧКИ ФЕРМА-ТОРРИЧЕЛЛИ
4. АЛГОРИТМ ПРИМА

## 3. РАСПАРАЛЛЕЛИВАНИЕ -----

1. ЭТАПЫ
2. ПРОГРАММА
3. ТРИАНГУЛЯЦИЯ
4. ТОЧКИ ФЕРМА
5. АЛГОРИТМ ПРИМА

## 4. РЕЗУЛЬТАТЫ -----

1. РАБОТА ПРОГРАММЫ
2. НАХОЖДЕНИЕ ТОЧЕК ФЕРМА
3. АЛГОРИТМ ПРИМА

## 5. ЗАКЛЮЧЕНИЕ -----

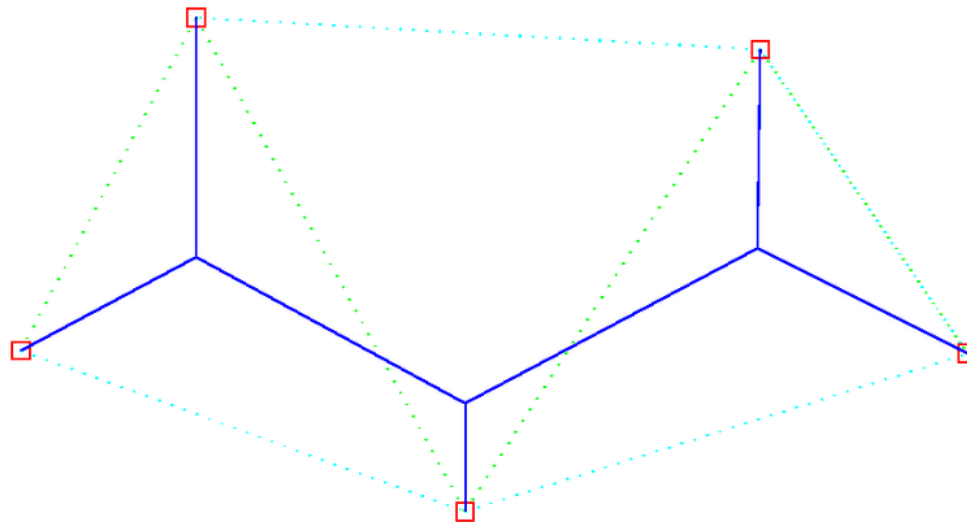
1. В ЧЕМ ПРОБЛЕМА

## 6. СПИСОК ИСТОЧНИКОВ -----

# ЗАДАЧА ПРИМЕР УСЛОВИЯ

Для общего понимания основной цели приведем **пример условия**, где требуется решить **задачу Штейнера о минимальном дереве**:

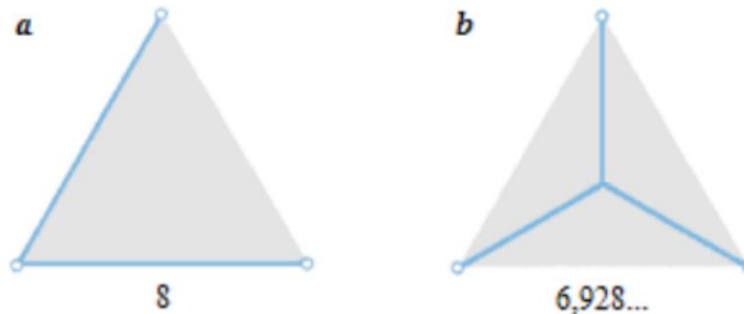
- «Пусть на некоторой плоскости расположено несколько объектов, между которыми необходимо проложить дороги. Нужно построить алгоритм, который принимает на вход расположение этих объектов и вычисляет такую карту дорог, в которой общая протяженность дорожного полотна минимальна»



# ЗАДАЧА ОПИСАНИЕ

То есть суть данной задачи состоит в поиске **кратчайшей сети**, соединяющей **все** данные в условии задачи точки (терминалы), используя **дополнительные** вершины – **точки Ферма-Торричелли** [1].

**Точки Ферма-Торричелли** – это точки плоскости, для каждой из которых сумма расстояний до вершин треугольников, для которых они были построены, является **минимальной**.



На рисунке выше видно, что метод нахождения минимального дерева с использованием точки Ферма-Торричелли (b) дает **более хороший** результат, чем метод без ее использования (a).

Полученное в результате решение является деревом, называемое **минимальным деревом Штейнера** (*Steiner Minimal Tree – SMT*).

# ЗАДАЧА ОСОБЕННОСТИ

У задачи Штейнера о минимальном дереве есть две значимые особенности:

- 1) Е. Гилберт и Х. Поллак показали, что дерево Штейнера получается **не более чем на 13,4%** короче минимального остовного дерева [2].
- 2) Эффективного (сложность выражается функцией, ограниченной сверху некоторым полиномом от параметра задачи, в данном случае число вершин графа) алгоритма, дающего точное решение проблемы Штейнера, **не существует** – задача является **NP-полной** [3].

# РЕШЕНИЕ ТРИАНГУЛЯЦИЯ ДЕЛОНЕ

Прежде чем начинать строить точки Ферма-Торричелли, нужно поделить данное в условии задачи множество точек плоскости на **множество треугольников**. Для этого был выбран **алгоритм Бойера–Уотсона** [4] **триангуляции Делоне** [5].

В вычислительной геометрии, алгоритм Бойера–Уотсона является алгоритмом, позволяющим выполнить триангуляцию Делоне конечного числа точек **для любого** количества измерений.

Алгоритм Бойера–Уотсона работает следующим образом:

- происходит добавление точек, по одной за раз, к триангуляции Делоне подмножества нужных точек.
- после каждой вставки любые треугольники, окружности которых содержат новую точку, удаляются, оставляя звездообразное многоугольное отверстие, которое затем **повторно триангулируется** с использованием новой точки.

# РЕШЕНИЕ ТРИАНГУЛЯЦИЯ ДЕЛОНЕ: ПСЕВДОКОД

Если попробовать изобразить алгоритм Бойера-Уотсона в виде псевдокода, то получится следующее:

```
function BowyerWatson (pointList)
    // pointList это массив координат, определяющий какие точки должны
    быть триангулированы
    triangulation := empty triangle mesh data structure
    add super-triangle to triangulation
    for each point in pointList do // добавляем все точки по одной для
    триангуляции
        badTriangles := empty set
        for each triangle in triangulation do // вначале находим все
        треугольники которые больше не подходят из-за вставки
            if point is inside circumcircle of triangle
                add triangle to badTriangles
        polygon := empty set
        for each triangle in badTriangles do // находим границу
        многоугольного отверстия
            for each edge in triangle do
                if edge is not shared by any other triangles in badTriangles
                    add edge to polygon
        for each triangle in badTriangles do // удаляем их из структуры
        данных
            remove triangle from triangulation
        for each edge in polygon do // ретриангулируем многоугольное
        отверстие
            newTri := form a triangle from edge to point
            add newTri to triangulation
        for each triangle in triangulation // вставка точек закончена,
        теперь чистка
            if triangle contains a vertex from original super-triangle
                remove triangle from triangulation
    return triangulation
```

# РЕШЕНИЕ ТОЧКИ ФЕРМА-ТОРРИЧЕЛЛИ

Нахождение **точек Ферма-Торричелли** происходит по следующему плану – для **каждого** треугольника выполняется следующее:

**1)** Находится вершина при **наибольшем угле**. В программе это реализовано посредством сравнения результатов решения следующих выражений (к примеру для угла  $\alpha$  при вершине A):

$$\alpha = \arccos \frac{b^2 + c^2 - a^2}{2bc}$$

где  $a, b, c$  – длины сторон соответственно напротив вершин A, B, C.



# РЕШЕНИЕ ТОЧКИ ФЕРМА-ТОРРИЧЕЛЛИ

2) Из **двух других** вершин строится **равносторонний** треугольник. В программе это реализовано решением следующей **системы уравнений** с целью нахождения координат оставшейся вершины нужного нам треугольника (к примеру если вершина *B* оказалась вершиной при **наибольшем угле**):

$$AC^2 = (X_X - X_A)^2 + (Y_X - Y_A)^2$$

$$AC^2 = (X_X - X_C)^2 + (Y_X - Y_C)^2$$

В данном случае координаты **искомой точки** обозначаются  $(X_X, Y_X)$ .

Из решений выбираются координаты той точки, что **наиболее удалена** от вершины при **наибольшем угле** изначального треугольника.

# РЕШЕНИЕ ТОЧКИ ФЕРМА-ТОРРИЧЕЛЛИ

3) Далее требуется найти координаты центра описанной вокруг полученного равностороннего треугольника окружности. Делается это аналогично пункту 2 – посредством решения **системы уравнений**:

$$AC^2 / 3 = (X_R - X_A)^2 + (Y_R - Y_A)^2$$

$$AC^2 / 3 = (X_R - X_C)^2 + (Y_R - Y_C)^2$$

В данном случае координаты **искомой точки** обозначаются  $(X_R, Y_R)$ .

Из решений выбираются координаты той точки, что **наиболее удалена** от вершины изначального треугольника при наибольшем угле.

# РЕШЕНИЕ ТОЧКИ ФЕРМА-ТОРРИЧЕЛЛИ

4) Требуется найти координаты пересечения окружности с центром в точке  $(X_R, Y_R)$  с прямой, проведенной через точки, полученные в пунктах 1 и 2. Делается это опять же посредством решения **системы уравнений**:

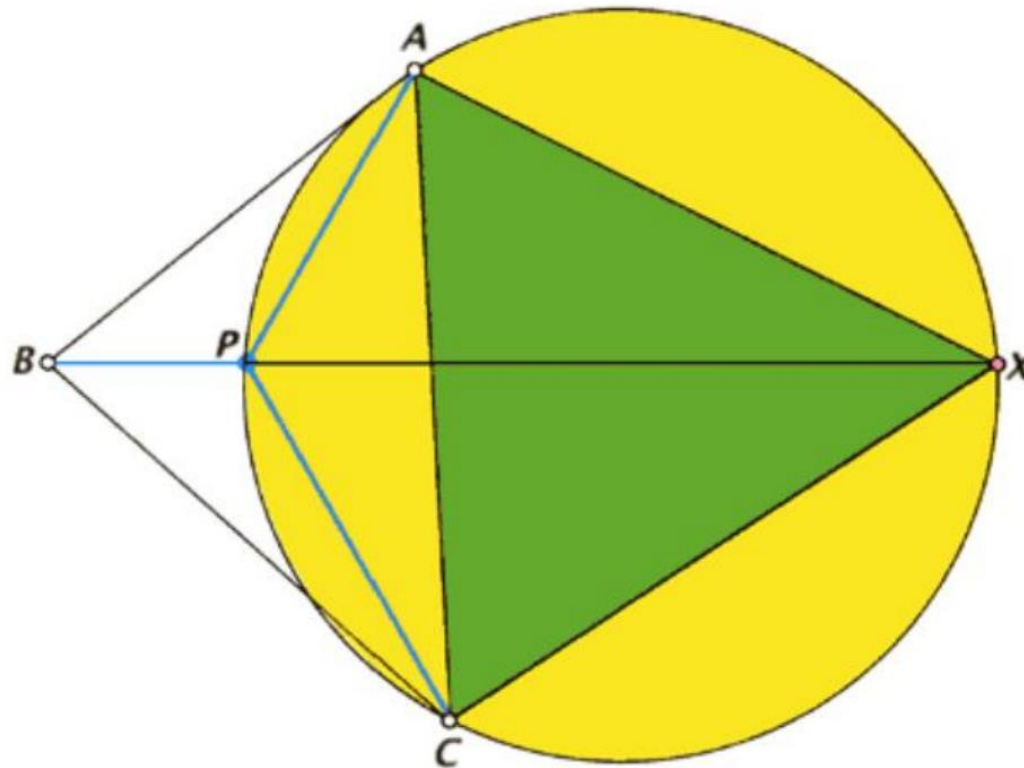
$$(X_P - X_X) / (X_B - Y_X) = (Y_P - Y_X) / (Y_B - Y_X)$$

$$(X_P - X_R)^2 + (Y_P - Y_R)^2 = AC^2 / 3$$

В данном случае координаты **искомой точки** обозначаются  $(X_P, Y_P)$ .

# РЕШЕНИЕ ТОЧКИ ФЕРМА-ТОРРИЧЕЛЛИ

Пример результата (*точка P*) всех этих построений можно увидеть на рисунке ниже:



# РЕШЕНИЕ АЛГОРИТМ ПРИМА

Для построения кратчайшего пути (*используя точки данные в условии и точки Ферма-Торричелли*) было решено воспользоваться **алгоритмом Прима** [6], служащим для нахождения минимального остовного дерева взвешенного неориентированного графа.

Для данной задачи он реализован по следующему принципу:

- 1) Создается булевый вектор **mstSet**, который следит за тем, какие вершины уже включены в **MST**.
- 2) Поставим значения всем вершинам во входном графе. Все значения инициализируются как **INFINITE**. Ставится значение **0** для первой вершины, чтобы она была выбрана первой.
- 3) Пока **mstSet** не включает в себя все вершины-города (могут быть случаи, когда не обязательно включать некоторые точки Ферма-Торричелли):
  - 1) Выбираем вершину **u**, которая не в **mstSet** и имеет минимальное значение.
  - 2) Включаем **u** в **mstSet**.
  - 3) Обновляем значение всех смежных вершин **u**. Чтобы обновить значение, пройдемся по всем смежным вершинам. Для каждой смежной вершины **v**, если вес ребра **u-v** меньше предыдущего значения **v**, обновим значение как вес **u-v**.

# РАСПАРАЛЛЕЛИВАНИЕ **ЭТАПЫ**

В итоге имеем, что предложенный способ решения задачи состоит из **трех этапов**:

- 1) Триангуляция Делоне алгоритмом Бойера-Уотсона
- 2) Нахождение точек Ферма-Торричелли при помощи геометрических построений
- 3) Использование алгоритма Прима для нахождения MST

Каждый из алгоритмов, участвующий в этапах решения задачи, может быть **распараллелен**.

# РАСПАРАЛЛЕЛИВАНИЕ ПРОГРАММА

Программа была написана на **C++** и реализована следующим образом:

- Для **алгоритма триангуляции**, реализованного в **delaunay.h**, были созданы дополнительные классы **Edge**, **Vector2** и **Triangle**, соответственно реализованные в **edge.h**, **vector2.h** и **triangle.h**.
  - **Edge** является классом, который представляет собой ребро графа и инициализируется двумя объектами класса **Vector2**.
  - Объект класса **Vector2** представляет собой точку, храня два значения – координаты  $x$  и  $y$  на плоскости.
  - **Triangle** – треугольник, который может инициализироваться объектами класса **Vector2** и **Edge**.
- **Алгоритм вычисления координат точек Ферма-Торричелли** был реализован в файле **steiner.h**, где на вход подается вектор объектов класса **Triangle**.
- **Алгоритм Прима** реализован в **prim.h**.

Для распараллеливания был использован **OpenMP** [7].

Github: <https://github.com/nechaevolegiu8/Steiner-Minimal-Tree>

# РАСПАРАЛЛЕЛИВАНИЕ ТРИАНГ.

Напомним, что этот алгоритм основан на инкрементной вставке с ретриангуляцией полости. **Параллельный** алгоритм, предложенный **Никосом Крисокоидесом [8]**, начинается с последовательного построения грубой триангуляции подмножества точек последовательным алгоритмом.

Созданные симплексы разбиваются на  $k$  непрерывных областей и распределяются по  $k$  процессорам. Границы между областями образуются некоторыми гранями симплексов и могут изменяться в процессе.

После распределения работы процессоры одновременно вставляют точки. На каждом шаге определяется полость, которую необходимо **повторно триангулировать**. Если полость пересекает границы, необходимо использовать некоторую синхронизацию процессоров, разделяющих симплексы в этой полости, прежде чем можно будет выполнить ретриангуляцию.

Новые симплексы перераспределяются по участникам, чтобы сбалансировать нагрузку на процессоры и минимизировать длину границы.



# РАСПАРАЛЛЕЛИВАНИЕ ТОЧКИ ФЕРМА

Алгоритм нахождения точек Ферма-Торричелли при помощи геометрических построений достаточно распараллелить так, чтобы поиск точки Ферма-Торричелли производился **сразу в нескольких** треугольниках **одновременно**.

Достаточно распараллелить работу следующего цикла:

//steiner.h

```
...
public:
    using TriangleType = Triangle<T>;
    using VertexType = Vector2<T>;

    const std::vector<VertexType>& additionalVertices(std::vector<TriangleType> &triangles)
    {
        std::vector<VertexType> additionalvertices;

        omp_set_num_threads(2); //OpenMP
        #pragma omp parallel for //OpenMP
        for (int e1 = 0; e1 < triangles.size(); e1++) //<-----
        {
            VertexType Vertex, RadiusVertex, SteinerVertex;
            int n = largestAngle(triangles[e1].p1, triangles[e1].p2, triangles[e1].p3);

            if (n == 1)
            {
                Vertex = findThirdVertex(triangles[e1].p2, triangles[e1].p1, triangles[e1].p3);
                RadiusVertex = findCenterVertex(triangles[e1].p2, triangles[e1].p1, triangles[e1].p3);
            }
        }
    }
...

```

# РАСПАРАЛЛЕЛИВАНИЕ АЛГ. ПРИМА

Итерации **алгоритма Прима** должны выполняться последовательно и, тем самым, **не могут** быть распараллелены. С другой стороны, выполняемые на каждой итерации алгоритма действия являются независимыми и **могут** реализовываться одновременно.

Так, например, **определение ребра наименьшего веса**, исходящего из оптимального каркаса в вершину, которая еще не принадлежит каркасу, может осуществляться для каждой вершины **в отдельности**.

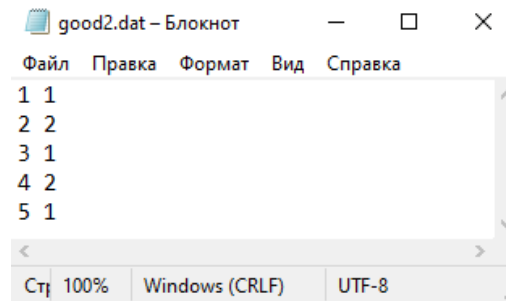
Достаточно распараллелить работу следующего цикла:

//prim.h

```
...
// Function to find the vertex with minimum key value, from the set
// of vertices not yet included in MST
int minKey(std::vector<float> &key, std::vector<bool> &mstSet, std::vector<std::vector<float>> &graph)
{
    // Initialize min value
    float min = FLT_MAX;
    int min_index;
    omp_set_num_threads(2); // OpenMP
    #pragma omp parallel for // OpenMP
    for (int v = 0; v < graph.size(); v++) { //<-----
        if (mstSet[v] == false && key[v] < min)
            min = key[v], min_index = v;
    }
    return min_index;
}
...
```

# РЕЗУЛЬТАТЫ РАБОТА ПРОГРАММЫ

Программа принимает на вход файлы, содержащие x и y координаты городов в следующем формате: на каждой строке должно быть по два числа, разделенных пробелом, которые собственно будут представлять собой координаты.



```
good2.dat - Блокнот
Файл  Правка  Формат  Вид  Справка
1 1
2 2
3 1
4 2
5 1
Стр 100%  Windows (CRLF)  UTF-8
```

Полученный в итоге работы программы результат выглядит следующим образом:



```
Консоль отладки Microsoft Visual Studio
...Points, included in SMT:
#1| x: 1 | y: 1 |
#2| x: 2 | y: 2 |
#3| x: 3 | y: 1 |
#4| x: 4 | y: 2 |
#5| x: 5 | y: 1 |
#6| x: 1.99944 | y: 1.57735 |
#7| x: 3.9992 | y: 1.57735 |

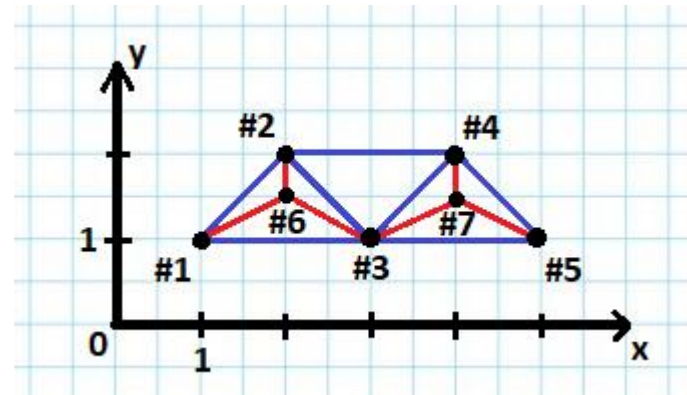
Solution:
Path      Length
#6 <-> #2  0.42
#6 <-> #3  1.16
#7 <-> #4  0.42
#7 <-> #5  1.16
#1 <-> #6  1.15
#3 <-> #7  1.15
Summary: 5.46
```

# РЕЗУЛЬТАТЫ РАБОТА ПРОГРАММЫ

Покажем наглядно построенное программой минимальное дерево Штейнера (из *красных* ребер):

```
...Points, included in SMT:
#1| x: 1 | y: 1 |
#2| x: 2 | y: 2 |
#3| x: 3 | y: 1 |
#4| x: 4 | y: 2 |
#5| x: 5 | y: 1 |
#6| x: 1.99944 | y: 1.57735 |
#7| x: 3.9992 | y: 1.57735 |

Solution:
Path      Length
#6 <-> #2  0.42
#6 <-> #3  1.16
#7 <-> #4  0.42
#7 <-> #5  1.16
#1 <-> #6  1.15
#3 <-> #7  1.15
Summary: 5.46
```



Где:

- точки #1 - #5 – заданные в **условии** координаты объектов, которые нужно обязательно **соединить все**;
- точки #6 - #7 – **вспомогательные** точки;

**Пример того, что не все вспомогательные точки нужны:**

- Вспомогательная точка #8 треугольника 234 была построена, но **отброшена** по мере работы программы как **неоптимальная** для минимального дерева Штейнера

# РЕЗУЛЬТАТЫ НАХОЖДЕНИЕ ТОЧЕК ФЕРМА

Сравнение времени работы алгоритма		
N	Время работы (миллисекунды)	
	Последовательный	Параллельный (2 потока)
2000	178	132
3000	279	219
5000	478	383
6000	723	625

# РЕЗУЛЬТАТЫ АЛГОРИТМ ПРИМА

Сравнение времени работы алгоритма		
N	Время работы (миллисекунды)	
	Последовательный	Параллельный (2 потока)
2000	7986	7669
3000	17785	17214
4000	32394	31359
5000	53270	51992
6000	81544	76111

# ЗАКЛЮЧЕНИЕ В ЧЕМ ПРОБЛЕМА

Напомним, что задача Штейнера о минимальном дереве **не имеет эффективного решения**. Данная проблема, по крайней мере в реализованном способе решения, становится видна на этапе применения алгоритма Прима.

Дело в том, что к этому моменту, после предыдущих двух этапов алгоритма, имеется **два набора** координат:

- точек изначально данных в условии объектов;
- вспомогательные точек.

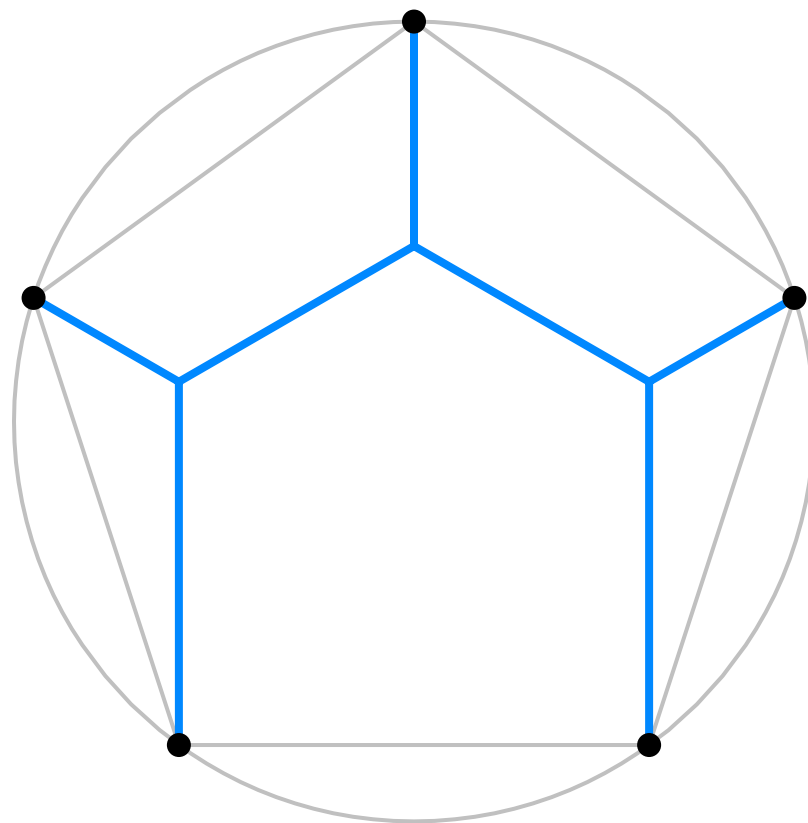
Но, как было показано на примере, для оптимального решения из последних нужны **не все**. Поэтому складывается ситуация, что просто так всех их **нельзя** задать как входные данные для алгоритма Прима (*и аналогичных ему*) – он соединит их все.

Остается только применение метода **полного перебора** для того чтобы решить – нужно ли включать ту или иную вспомогательную точку в итоговое решение.

# СПИСОК ИСТОЧНИКОВ

1. Точка Ферма-Торричелли // Википедия. URL:  
[https://ru.wikipedia.org/wiki/Точка\\_Ферма](https://ru.wikipedia.org/wiki/Точка_Ферма) (Дата обращения 09.12.20)
2. Steiner Minimal Trees // Semantic Scholar. URL:  
<https://pdfs.semanticscholar.org/46e5/7939baff2ec484b2e0f5c4b78f9129e710e9.pdf>  
(Дата обращения 09.12.20)
3. Доказательство NP-полноты задачи Штейнера // Profs.Sienze. URL:  
<http://profs.sci.univr.it/~rrizzi/classes/Complexity/provette/Santuari/steiner.pdf>  
(Дата обращения 09.12.20)
4. Триангуляция Делоне // Википедия. URL:  
[https://ru.wikipedia.org/wiki/Триангуляция\\_Делоне](https://ru.wikipedia.org/wiki/Триангуляция_Делоне) (Дата обращения 09.12.20)
5. Описание алгоритма Бойера-Уотсона // Википедия. URL:  
[https://en.wikipedia.org/wiki/Bowyer-Watson\\_algorithm](https://en.wikipedia.org/wiki/Bowyer-Watson_algorithm) (Дата обращения 09.12.20)
6. Алгоритм Прима // Википедия. URL:  
[https://ru.wikipedia.org/wiki/Алгоритм\\_Прима](https://ru.wikipedia.org/wiki/Алгоритм_Прима) (Дата обращения 09.12.20)
7. OpenMP // Википедия. URL: <https://ru.wikipedia.org/wiki/OpenMP> (Дата обращения 09.12.20)
8. Параллельная реализация алгоритма Бойера-Уотсона // Research Gate. URL:  
[https://www.researchgate.net/publication/2607476\\_Task\\_Parallel\\_Implementation\\_of\\_the\\_Bowyer-Watson\\_Algorithm](https://www.researchgate.net/publication/2607476_Task_Parallel_Implementation_of_the_Bowyer-Watson_Algorithm) (Дата обращения 09.12.20)





**СПАСИБО ЗА ВНИМАНИЕ**