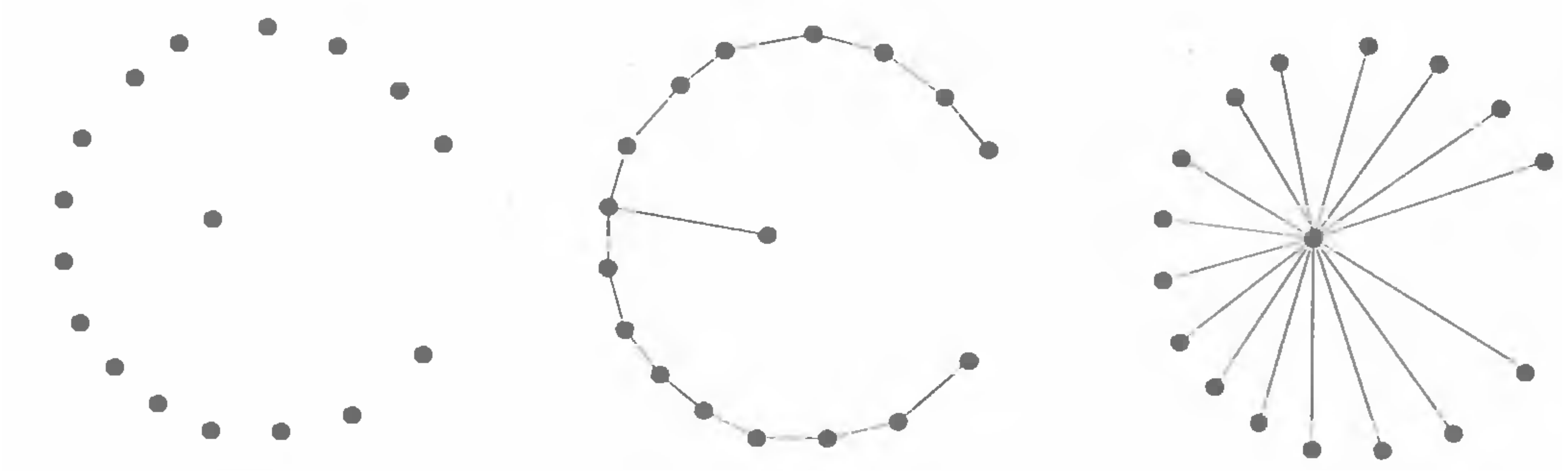
Алгоритмы для работы со взвешенными графами

Минимальные остовные деревья

$$G = (V,E)$$

Остовным деревом называется подмножество ребер E, которые создают дерево, содержащее все вершины V.



Алгоритм Прима
построения минимального остовного дерева

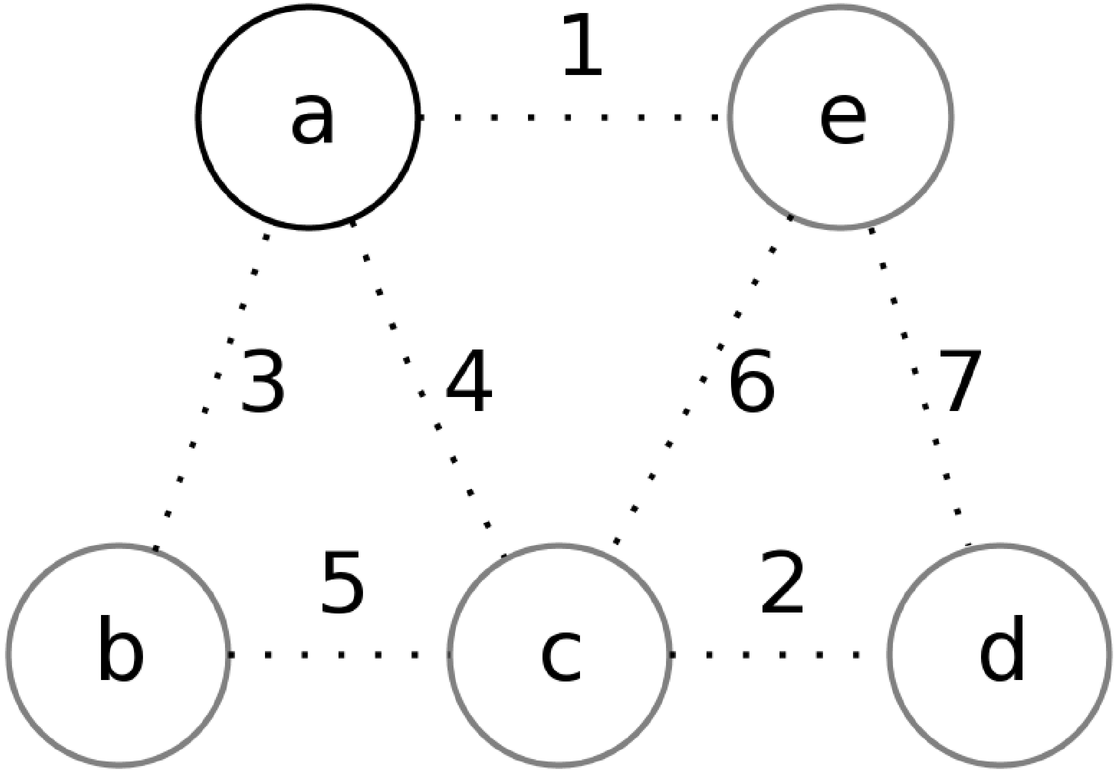
Алгоритм работает поэтапно

› Начинаем с указанной вершины

› Вставляем в остовное дерево одну новую вершину, исходя из «жадного» принципа (из множества рассмотренных ребер к дереву добавляется ребро с наименьшим весом)

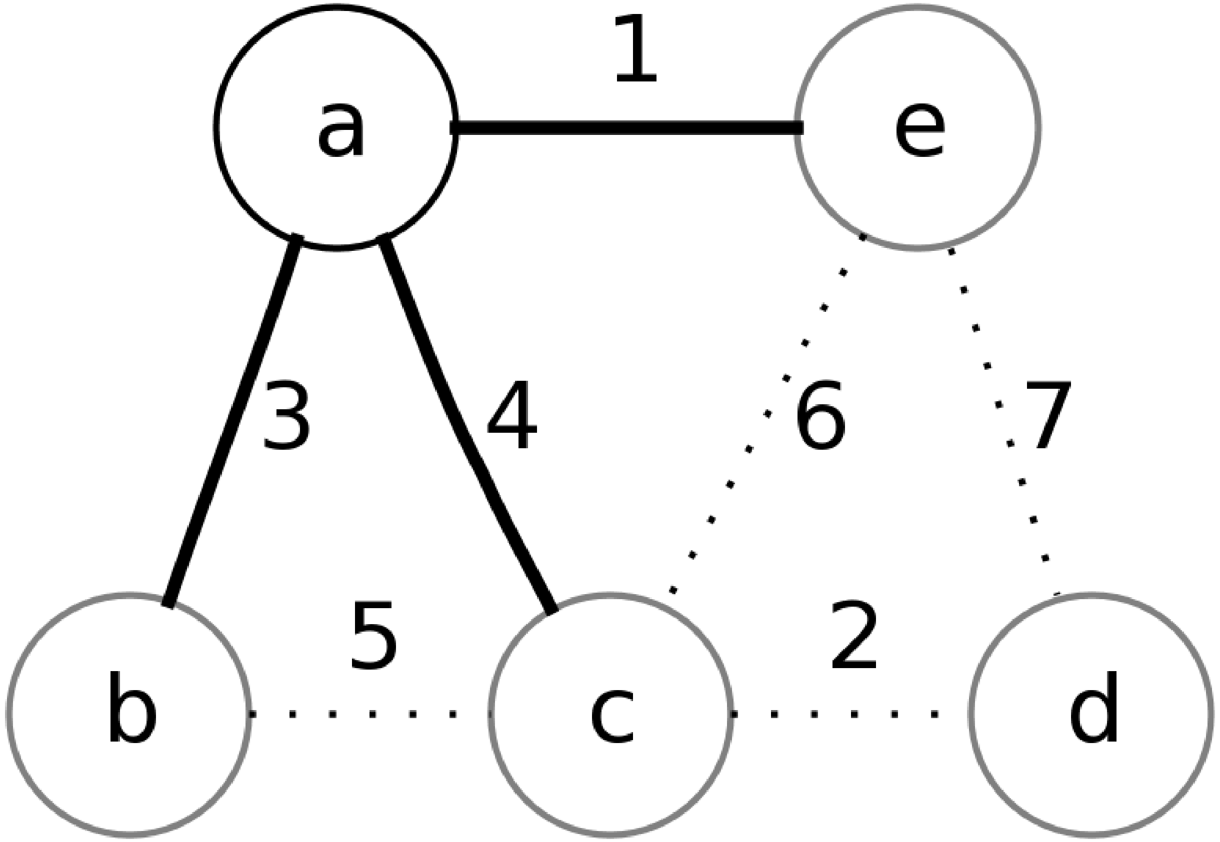
```
Prim-MST(G)
  Выбираем произвольную вершину s, с которой надо начинать
                                        построение дерева
   While (остаются вершины, не включенные в дерево)
    Выбираем ребро минимального веса между деревом и вершиной вне дерева
    Добавляем выбранное ребро и вершину в дерево T_prim
```

Пример реализации алгоритма Прима



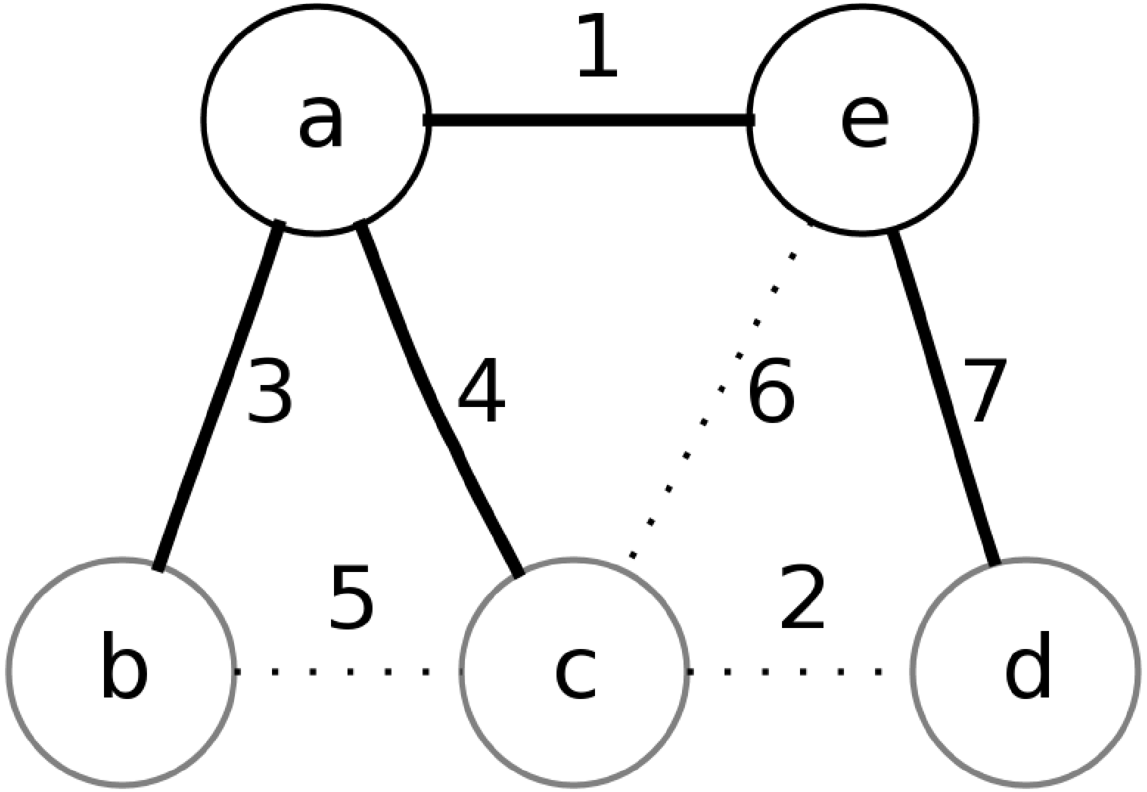
a	b	c	d	e
0	∞	∞	∞	∞

Пример реализации алгоритма Прима



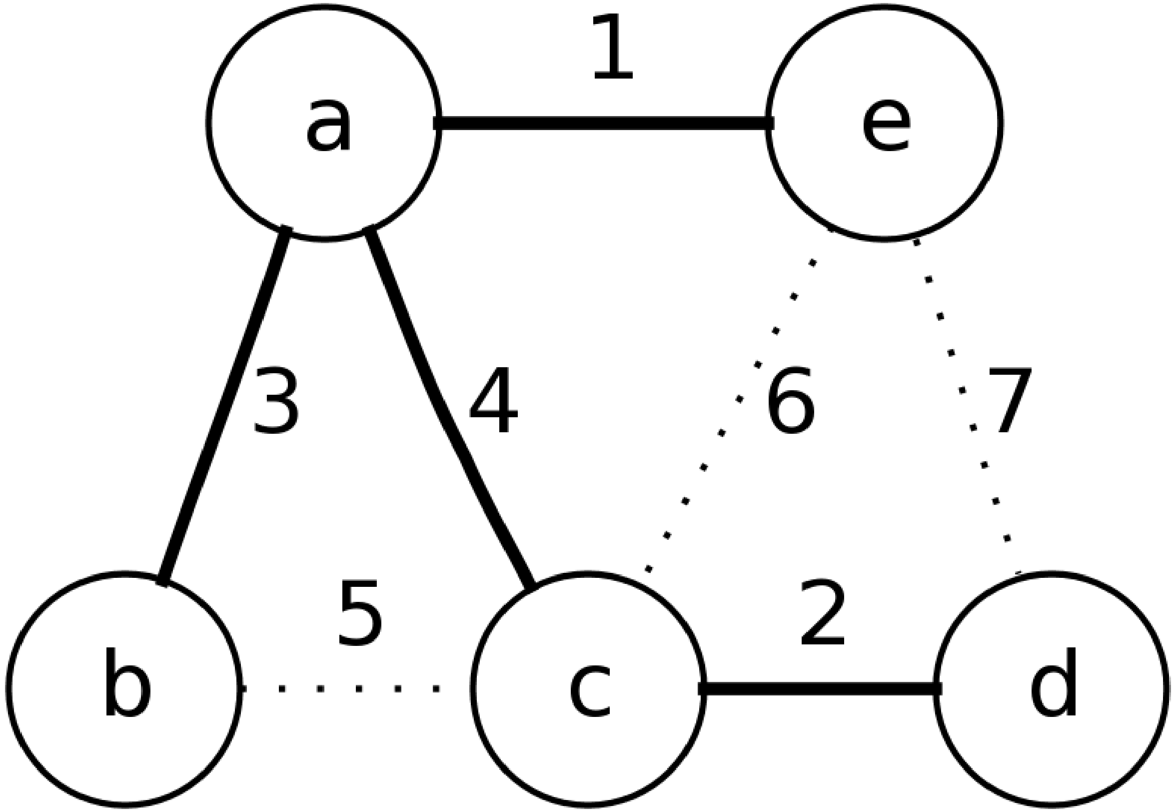
a	b	c	d	e
0	3	4	∞	1

Пример реализации алгоритма Прима



a	b	c	d	e
0	3	4	7	1

Пример реализации алгоритма Прима



a	b	c	**d**	e
0	3	4	2	1

Анализ эффективности алгоритма Прима

Зависит от используемых структур данных

Если *n* – исполняемых циклов; *m* – просматриваемых ребер в каждом цикле

$\boldsymbol{O(n^2)}$ – «наивная» реализация

$\boldsymbol{O(m + n \lg n)}$ – применение структуры данных в виде кучи с приоритетами (двоичная куча, Фибоначчиева куча)

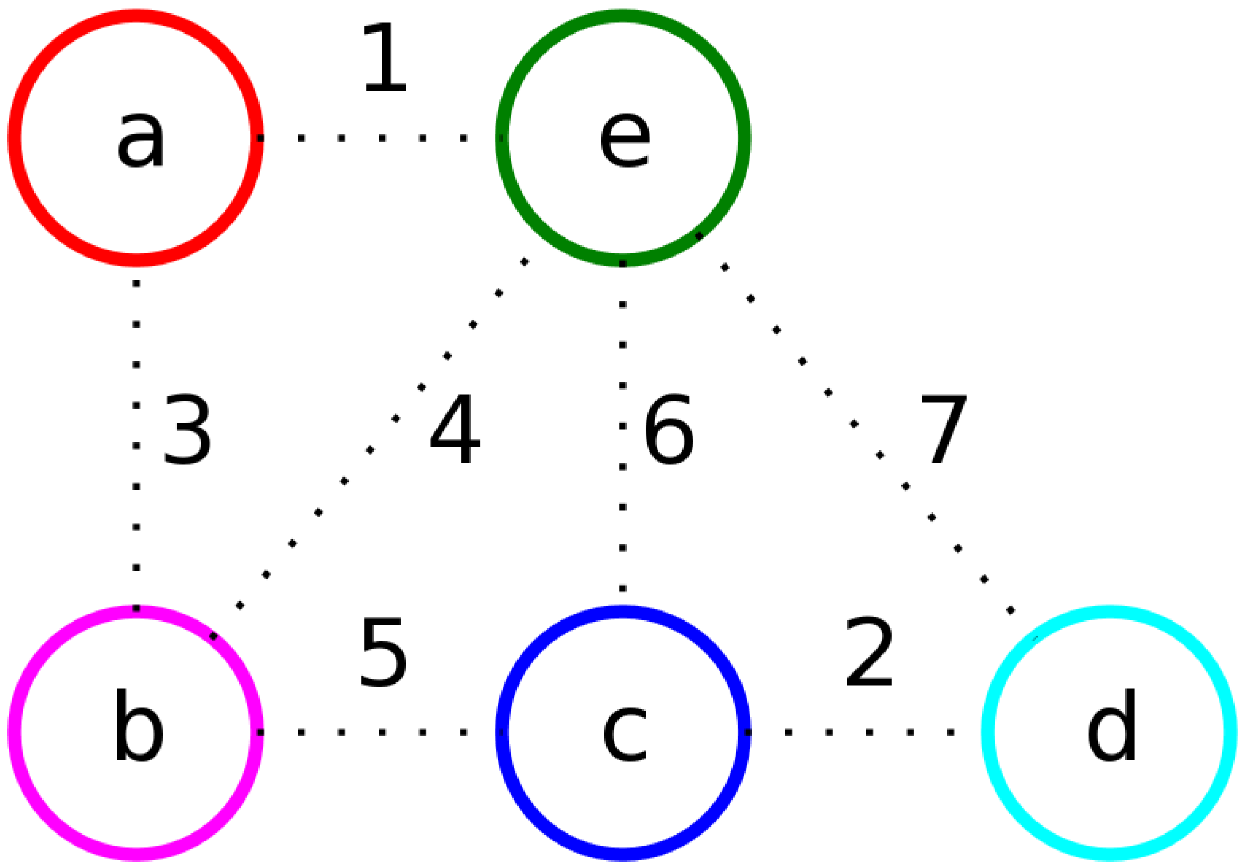
Алгоритм Крускала

построения минимального остовного дерева

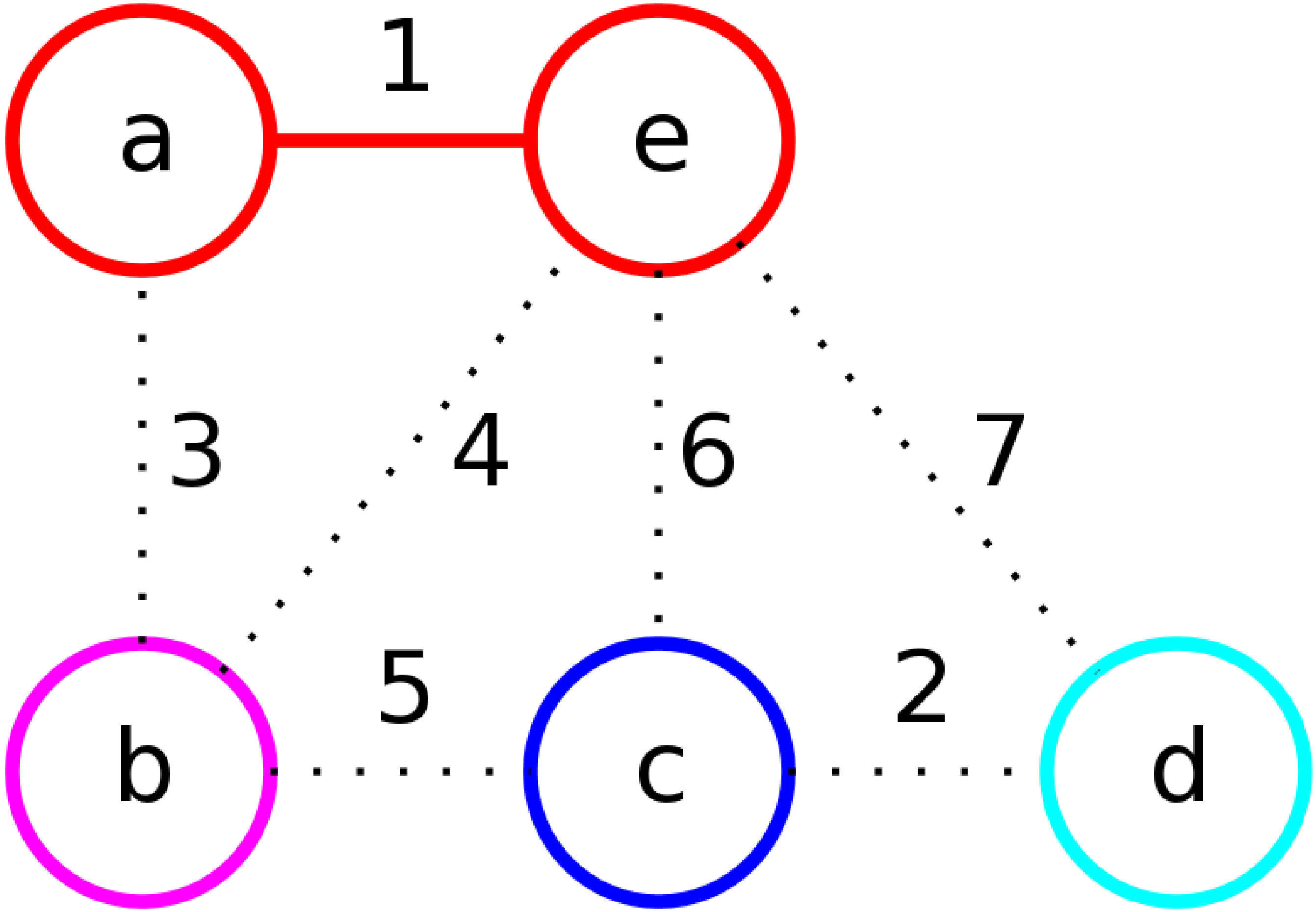
- Первоначально каждая вершина – отдельный компонент будущего дерева
- Последовательно ищем ребро для добавления в расширяющийся лес путем поиска самого легкого среди соединяющих два дерева в лесу
- Выполняется проверка на нахождение обеих конечных точек ребра-кандидата в одной и той же связанной компоненте

```
Kruskal-MST(G)
   Помещаем ребра в очередь с приоритетами, упорядоченную по весу
   count = 0
   while (count < n - 1) do
      рассматриваем следующее ребро (v, w)
      if (component (v) ≠ component(w))
         добавляем в дерево Tkruskal
         объединяем component(v) и component(w)
```

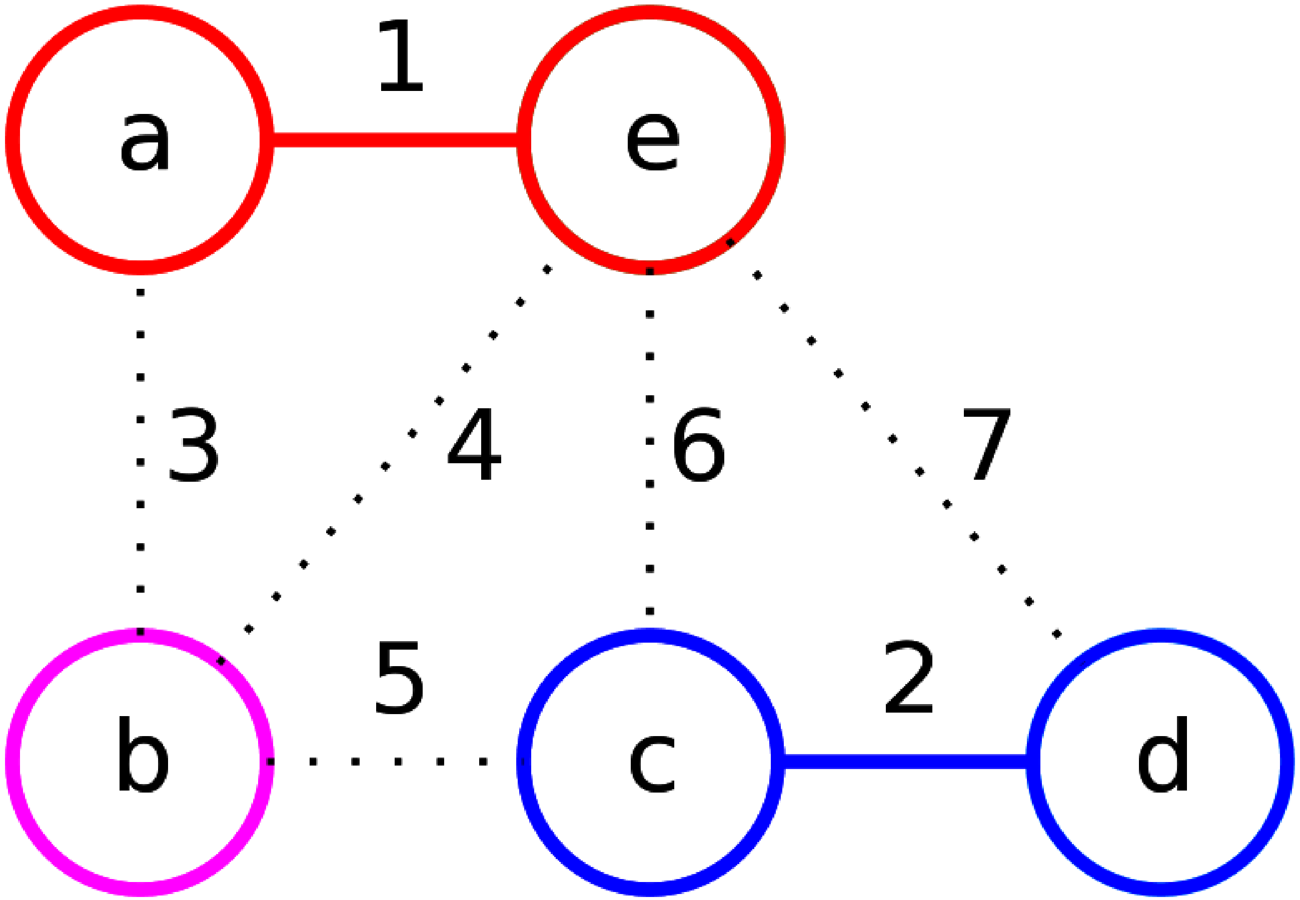
Пример реализации алгоритма Крускала



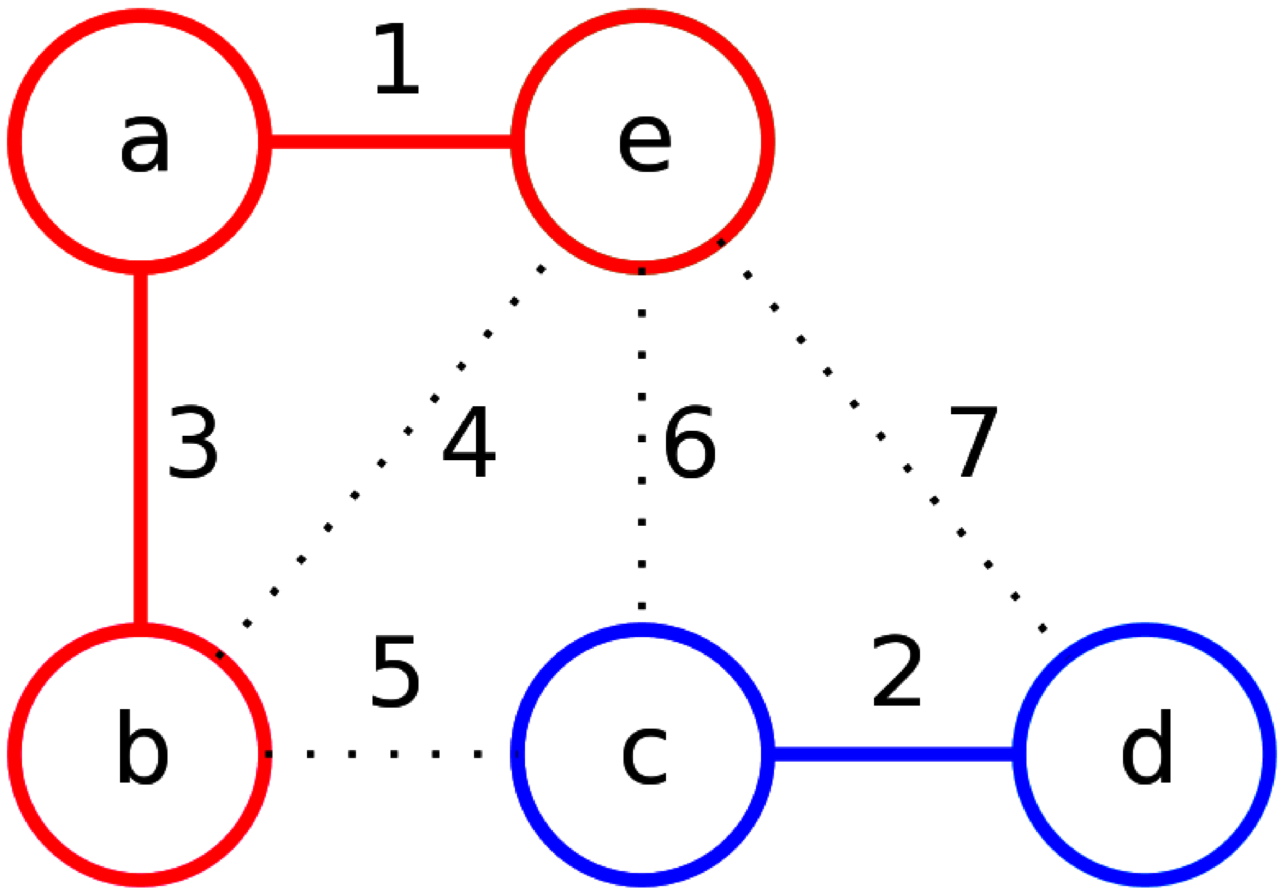
Пример реализации алгоритма Крускала



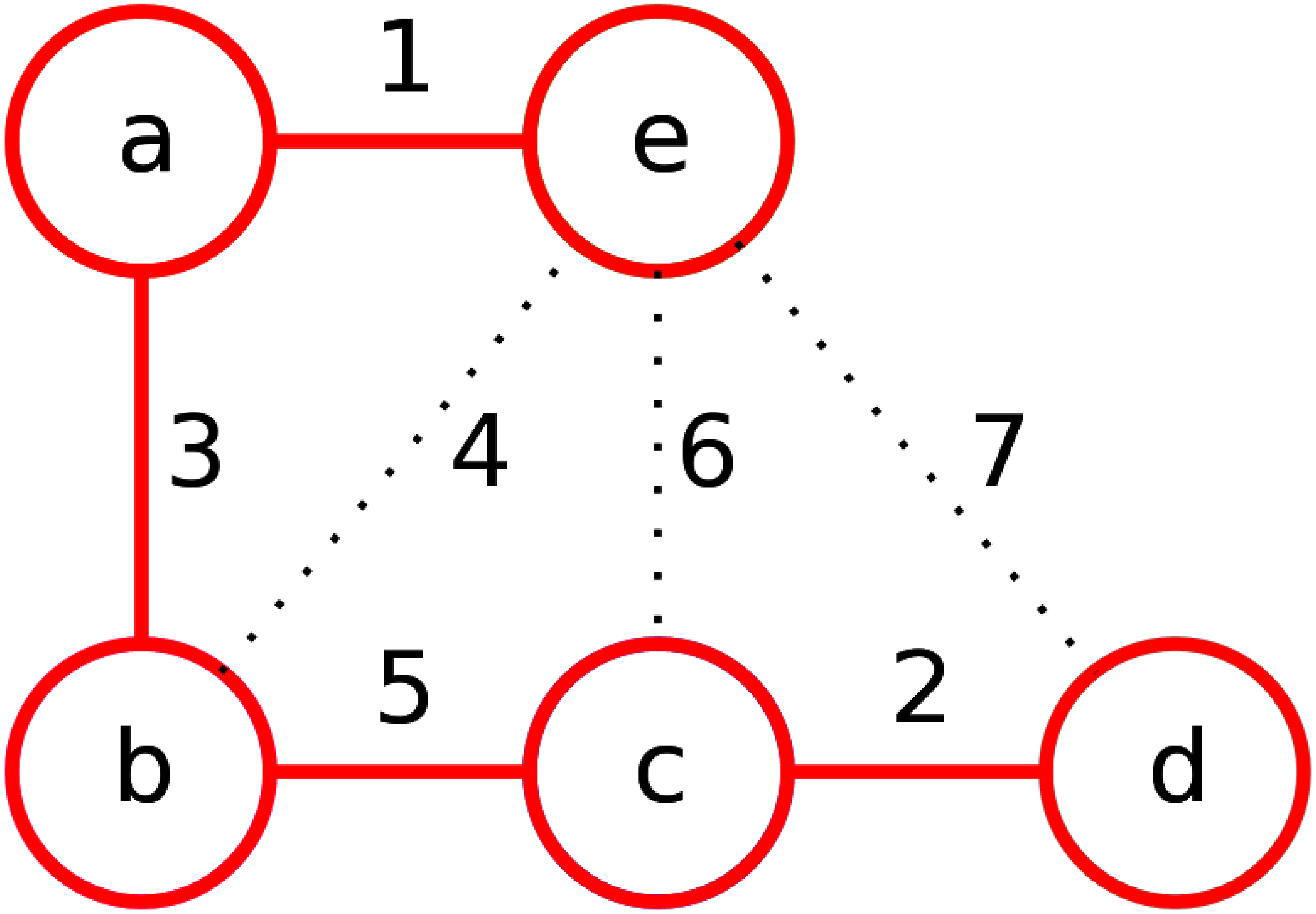
Пример реализации алгоритма Крускала



Пример реализации алгоритма Крускала



Пример реализации алгоритма Крускала



Анализ эффективности алгоритма Крускала

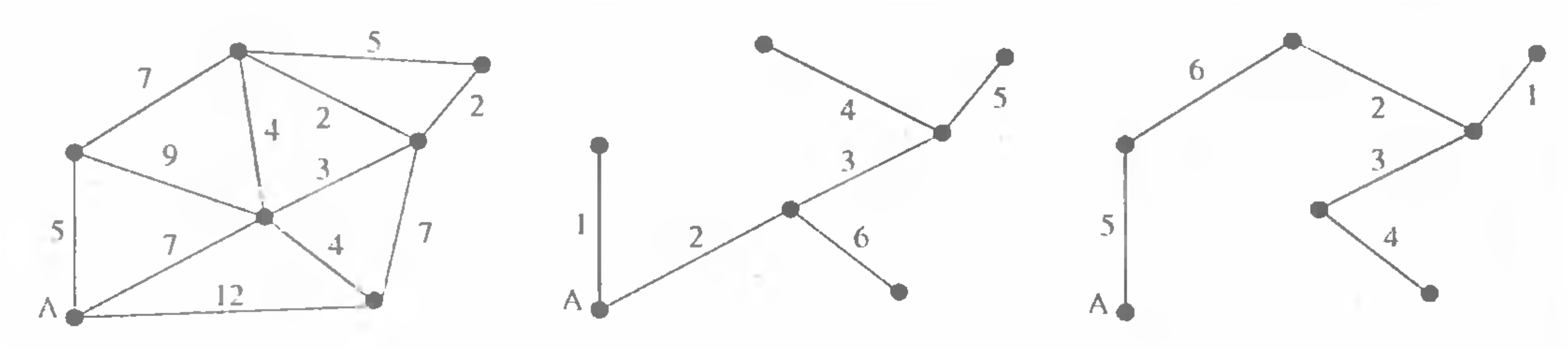
Зависит от используемых структур данных

Если n – вершин; m – ребер

$\boldsymbol{O(m\,\lg m)}$ – время упорядочивания ребер

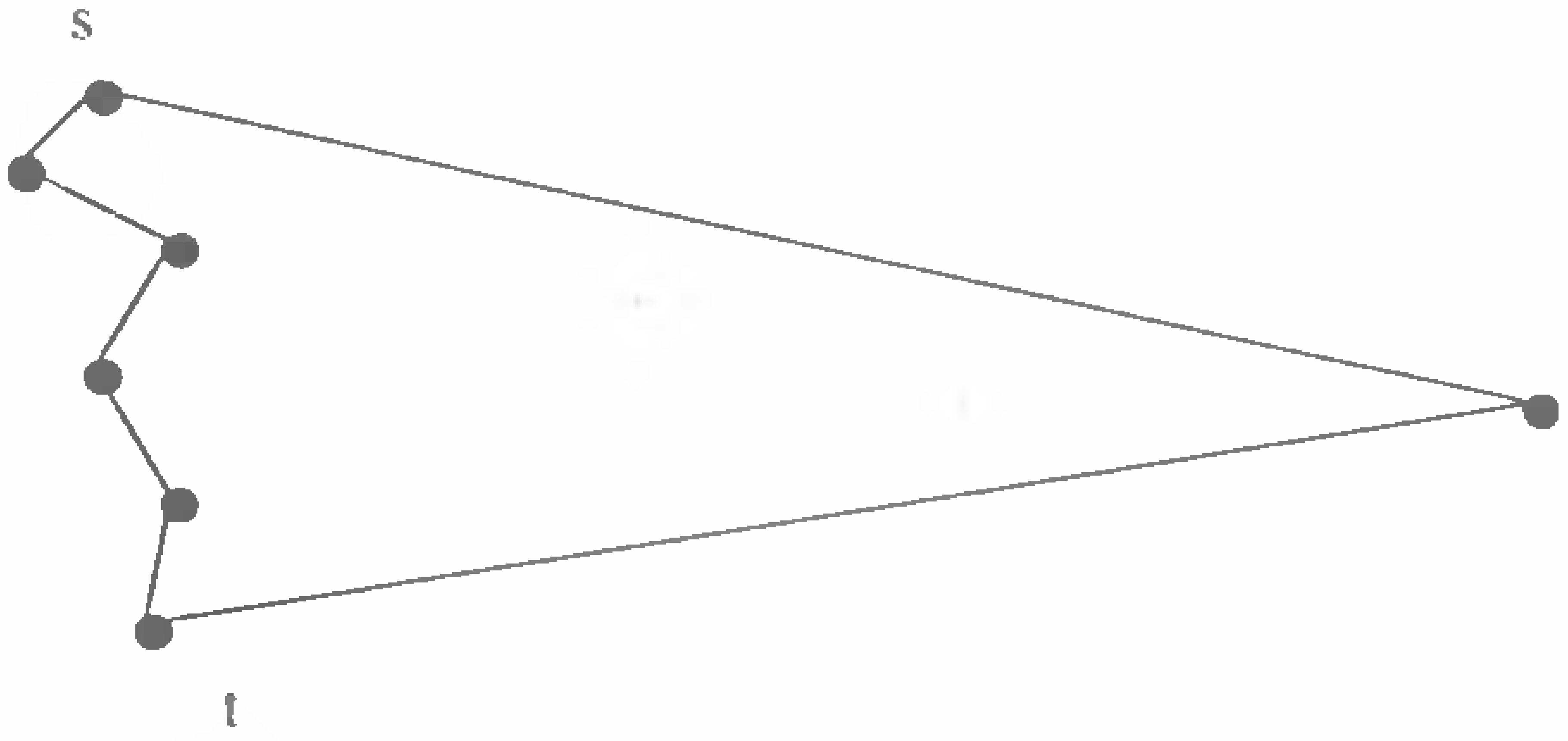
$\boldsymbol{O(m\,n)}$ – время исполнения при реализации поиска в ширину или глубину

Сравнение МОД по алгоритмам Прима и Крускала



Поиск кратчайшего пути

Путь – последовательность ребер ,соединяющих две вершины



Алгоритм Дейкстры

Задача:

Для заданного взвешенного графа *G = (V, E)* найти кратчайшие пути из заданной вершины *s* до всех остальных вершин. Веса всех рёбер неотрицательны.

Алгоритм:

В ориентированном взвешенном графе, вес рёбер которого неотрицателен w: $E \to \mathbb{R}$, алгоритм Дейкстры находит длины кратчайших путей из заданной вершины *s* до всех остальных.

В алгоритме поддерживается множество вершин U, для которых уже вычислены длины кратчайших путей до них из *s*. На каждой итерации выбирается вершина $u \notin U$, которой не соответствует минимальная оценка кратчайшего пути. Вершина u добавляется в множество U и производится релаксация всех исходящих из неё рёбер.

Псевдокод алгоритма Дейкстры

```
func dijkstra(s):
    for v ∈ V
        d[v] = ∞
        used[v] = false
    d[s] = 0
    for i ∈ V
        v = null
        for j ∈ V                        // найдём вершину с минимальным расстоянием
            if !used[j] and (v == null or d[j] < d[v])
                v = j
        if d[v] == ∞
            break
        used[v] = true
        for e : исходящие из v рёбра     // произведём релаксацию по всем рёбрам, исходящим из v
            if d[v] + e.len < d[e.to]
                d[e.to] = d[v] + e.len
```

Эффективность алгоритма Дейкстры

Пусть n - количество вершин, m - количество рёбер.

	Поиск минимума	**Релаксация**	**Общее**	**Описание**
Наивная реализация	$O(n)$	$O(1)$	$O(n^2+m)$	n раз осуществляем поиск вершины с минимальной величиной d среди $O(n)$ непомеченных вершин и m раз проводим релаксацию за $O(1)$. Для плотных графов ($m \approx n^2$) данная асимптотика является оптимальной.
Двоичная куча	$O(\log n)$	$O(\log n)$	$O(m \log n)$	Можно выполнять операции извлечения минимума и обновления элемента за $O(\log n)$. Тогда время работы алгоритма Дейкстры составит $O(n \log n + m \log n) = O(m \log n)$.
Фибоначчиева куча	$O(\log n)$	$O(1)$	$O(n \log n+m)$	Можно выполнять операции извлечения минимума за $O(\log n)$ и обновления элемента за $O(1)$. Таким образом, время работы алгоритма составит $O(n \log n +m)$.

Визуализация алгоритма Дейкстры