Каркасы графа

Лекция 6

Определения

G(V,E) - связный неориентированный граф с заданной функцией стоимости, отображающей ребра в вещественные числа.

Остовное дерево или **каркас (скелет)** графа – это подграф, который :

1) содержит все вершины графа,
2) является деревом.

Нас интересуют алгоритмы построения минимального каркаса.

Минимальным каркасом является такой каркас, сумма весов ребер которого минимальна.

Алгоритм Краскала (Джозеф Крускал, 1956 год)

1. Сортируем ребра графа по возрастанию весов.
2. Полагаем, что каждая вершина относится к своей компоненте связности.
3. Проходим ребра в "отсортированном" порядке. Для каждого ребра выполняем:
   a) если вершины, соединяемые данным ребром, лежат в разных компонентах связности, то объединяем эти компоненты в одну, а рассматриваемое ребро добавляем к минимальному остовному дереву;
   b) если вершины, соединяемые данным ребром лежат в одной компоненте связности, то исключаем ребро из рассмотрения.
4. Если есть еще нерассмотренные ребра и не все компоненты связности объединены в одну, то переходим к шагу 3, иначе выход.

Время работы:

Сортировка рёбер - $O(|E| \times log|E|)$

Компоненты связности удобно хранить в виде системы непересекающихся множеств.

Все операции в таком случае займут $O(E)$

Алгоритм Краскала

Вход. Неориентированный граф *G*= (*V, E*) с функцией стоимости *c,* заданной на его ребрах.

Выход. Остовное дерево *S*= (*V, T*) наименьшей стоимости для G.

Метод:

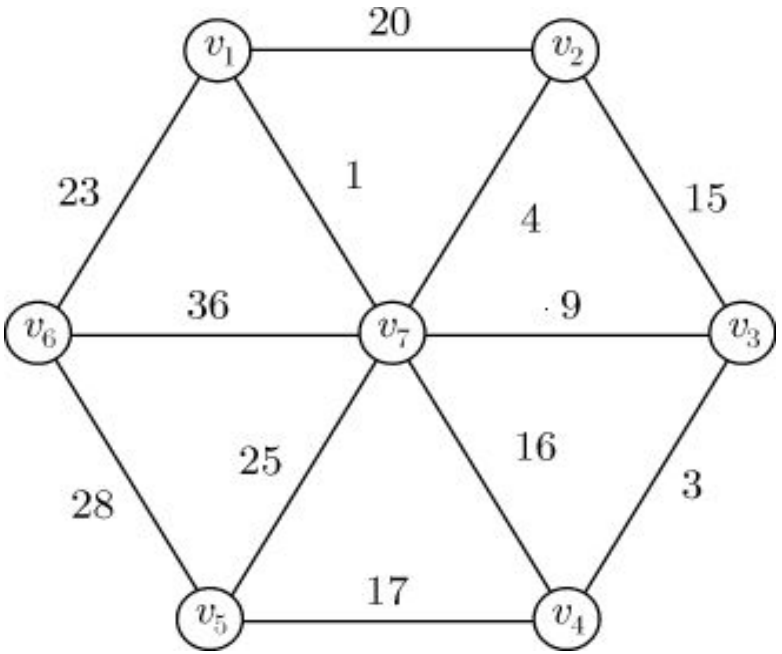
1. *T*← ∅*;* *// остовное дерево (каркас)*
2. *VS*← ∅ *;* *// набор непересекающихся множеств вершин*
3. построить очередь с приоритетами *Q,* содержащую все ребра из *E;*
4. Для ∀ v∈V ВЫПОЛНИТЬ: добавить {*v*} к *VS;*

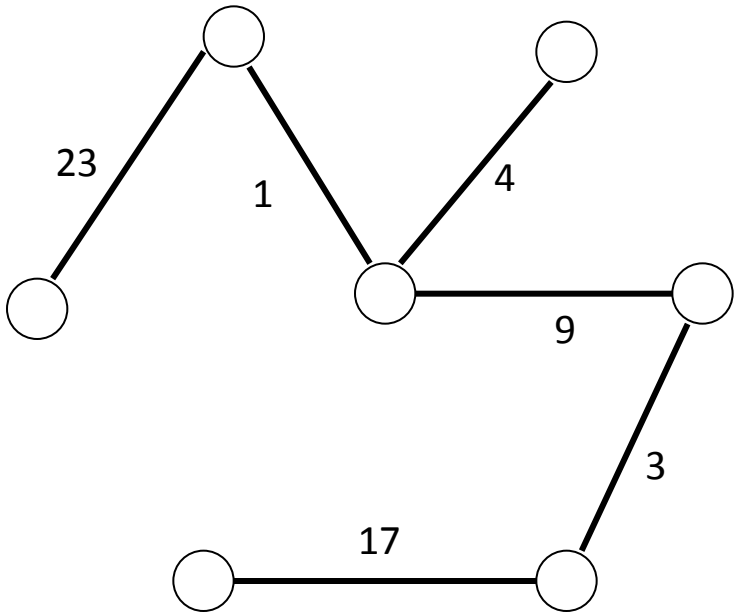
   Пока |*VS*|> 1 выполнить:
6. { выбрать в *Q* ребро (*v, w*) наименьшей стоимости;
7. удалить (*v, w*) из Q;
8. если *v* и *w* принадлежат различным множествам *W1 и W2* из *VS* то
9. { заменить *W1 и W2* на *W1* ∪ *W2* в *VS;*
10. добавить (*v, w*) к *T;*

    }

   }

Пример



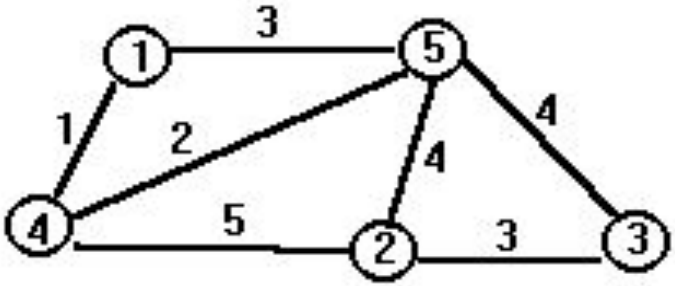


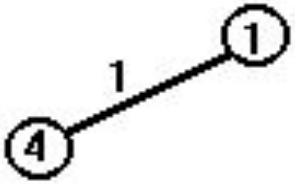
Получившееся дерево является каркасом минимального веса.

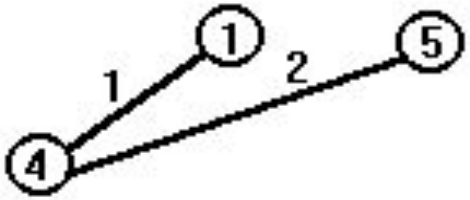
Введем массив меток вершин графа *Mark*.
Начальные значения элементов массива равны номерам
соответствующих вершин (*Mark*[*i*] = *i*; $i \in 1..N$).

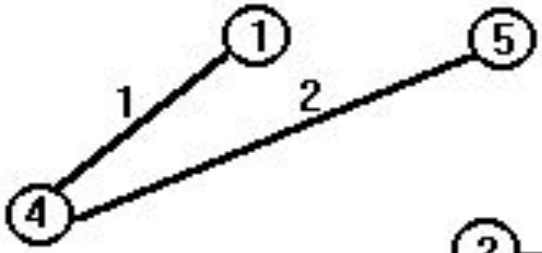
Ребро выбирается в каркас в том случае, если вершины, соединяемые им, имеют разные значения меток.

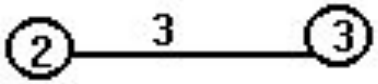
Пример приведен на следующем слайде,
изменения *Mark* показаны в таблице.

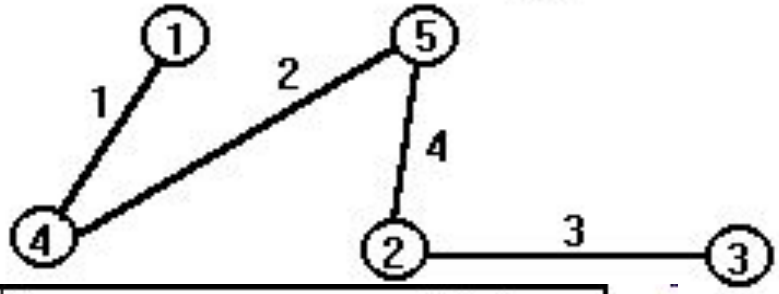












Номер итерации	Ребро	Значения элементов Mark
начальное значение	-	[1,2,3,4,5]
1	<1,4>	[1,2,3,1,5]
2	<4,5>	[1,2,3,1,1]
3	<2,3>	[1,2,2,1,1]
4	<2,5>	[1,1,1,1,1]

Алгоритм Прима

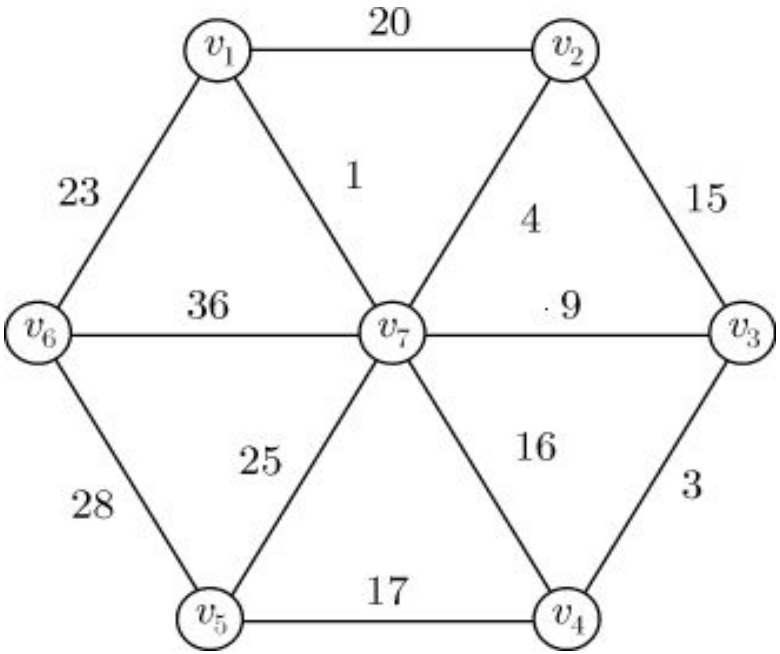
На каждом шаге вычеркиваем из графа дугу максимальной стоимости с тем условием, что она не разрывает граф на две или более компоненты связности, т.е. после удаления дуги граф должен оставаться связным.

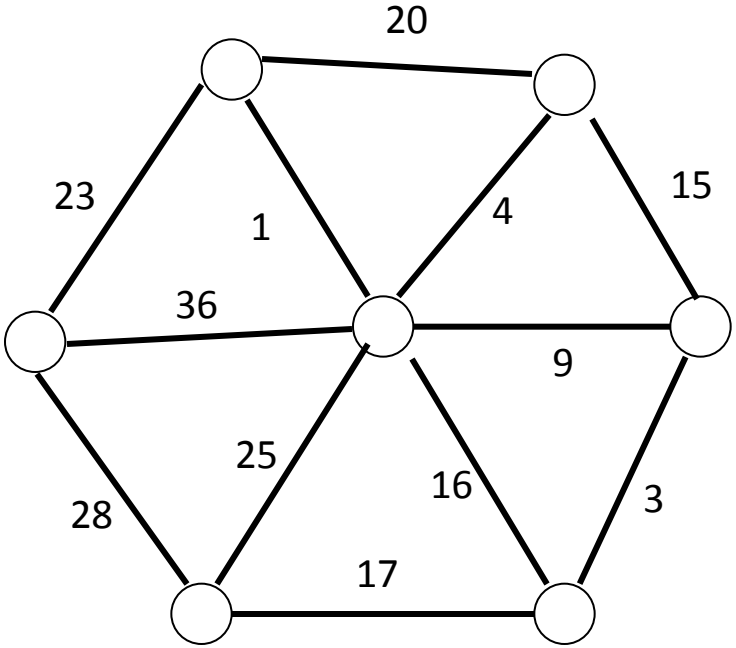
Для того, чтобы определить, остался ли граф связным, достаточно запустить поиск в глубину от одной из вершин, связанных с удаленной дугой.

Условие окончания алгоритма?

Например, пока количество ребер больше либо равно количеству вершин, нужно продолжать, иначе – остановиться.

Пример


v_1
v_2
v_3
v_4
v_5
v_6
v_7
20
23
1
4
15
36
9
25
16
28
3
17


20
23
1
4
15
36
9
25
16
28
3
17

Алгоритм Прима (Ярника, Дейкстры)

Алгоритм впервые был открыт в 1930 году чешским математиком Войцехом Ярником, позже переоткрыт Робертом Примом в 1957 году, и, независимо от них, Э. Дейкстрой в 1959 году.

Время работы алгоритма – $O(V * E)$

Можно улучшить – $O(E \log V + V^2)$

При использовании двоичной кучи – $O(E \log V)$

При использовании фибоначчиевой кучи – $O(E + V \log V)$

1) Выбирается произвольная вершина - она будет корнем остовного дерева;

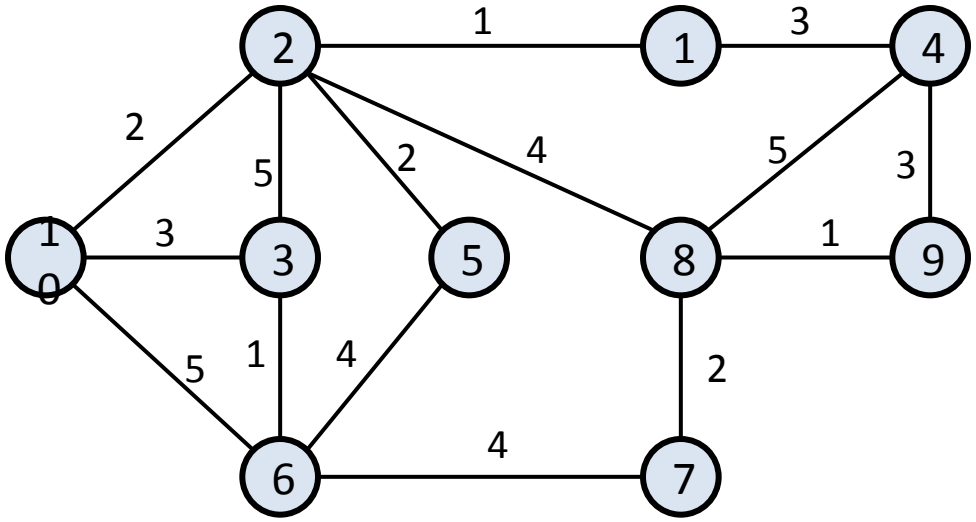
2) Измеряется расстояние от нее до всех других вершин, т.е. находится минимальное расстояние *s* от дерева до вершин, которые не включены в дерево;

3) До тех пор, пока в дерево не добавлены все вершины делать:

- найти вершину *u*, расстояние от дерева до которой минимально;
- добавить ее к дереву;
- пересчитать расстояния от невключенных вершин до дерева следующим образом:

если расстояние до какой-либо вершины от *u* меньше текущего расстояния *s* от дерева, то в *s* записывается новое расстояние.

Запускаем алгоритм обхода графа, начиная с произвольной вершины.
В качестве контейнера выбираем очередь с приоритетами. Приоритет – текущая величина найденного расстояния до уже построенной части остовного дерева.
Релаксации подвергаются прямые и обратные ребра.



n	1	2	3	4	5	6	7	8	9	10
π		1	10	1	2	3	8	9	4	2
d	0	1	3	3	2	1	2	1	3	2

В результате работы получаем список ребер остовного дерева вместе с весами

Реализация за $O(M \log N + N^2)$

Отсортируем все рёбра в списках смежности каждой вершины по увеличению
веса – $O(M \log N)$.
Для каждой вершины заведем указатель, указывающий на первое доступное
ребро в её списке смежности. Изначально все указатели указывают на начала
списков.
На i-ой итерации перебираем все вершины, и выбираем наименьшее по весу
ребро среди доступных. Поскольку все рёбра уже отсортированы по весу, а
указатели указывают на первые доступные рёбра, то выбор наименьшего
ребра осуществится за $O(N)$.
После этого обновляем указатели (сдвигаем вправо), т.к. некоторые из них
указывают на ставшие недоступными рёбра (оба конца которых оказались
внутри дерева).

Система непересекающихся множеств (СНМ)

Система непересекающихся множеств — это структура данных, которая реализует разбиение множества.

Каждое подмножество, входящее в разбиение, характеризуется своим представителем.

Это понятие было введено Тарьяном (Tarjan) в 1975 году.

СНМ поддерживает следующие операции:

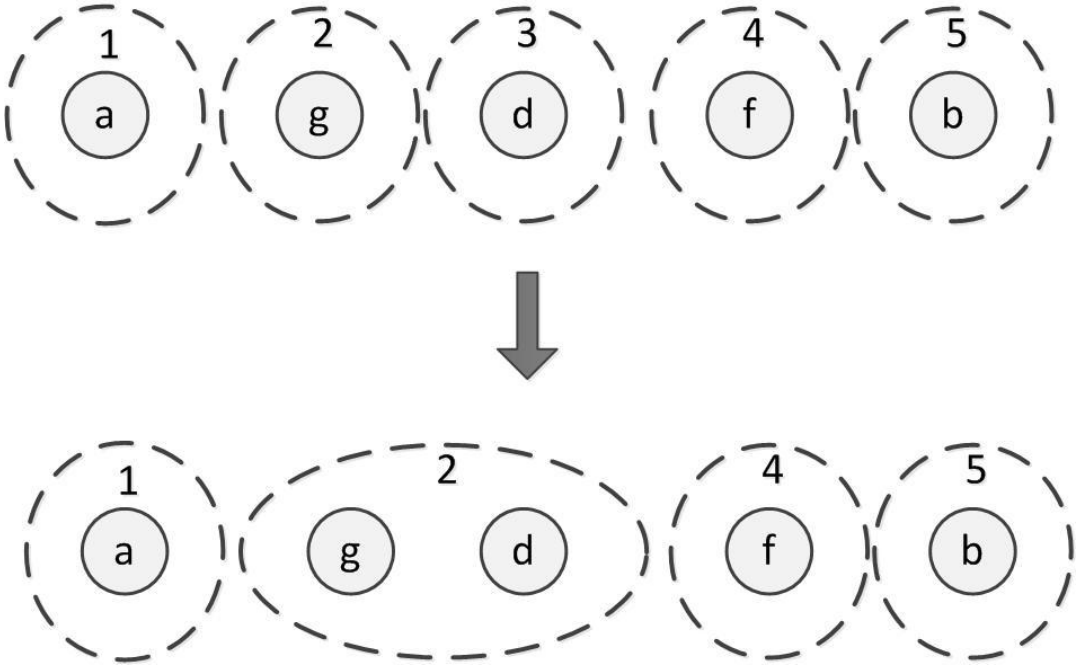
MakeSet (*x*) — добавляет в СНМ новый элемент *x*, который заносится в новое подмножество, представителем которого становится *x*.

FindSet (*x*) — осуществляет поиск подмножества, которому принадлежит элемент *x*, и возвращает его представителя.

Union (*x, y*) — объединяет в одно множество два подмножества, к которым принадлежат элементы *x* и *y*. Возвращает элемент, который становится представителем этого множества.

Простая реализация

В этой реализации для каждого элемента множества хранится номер или, другими словами, цвет подмножества, к которому этот элемент принадлежит.



Если хранить множества, например, в виде массива, то при таком подходе операция *FindSet* (*x*) выполняется за *O*(1), а операция *Union* (*x, y*) — за *O*(|*V*|). Последняя оценка не удовлетворяет требованию СНМ.

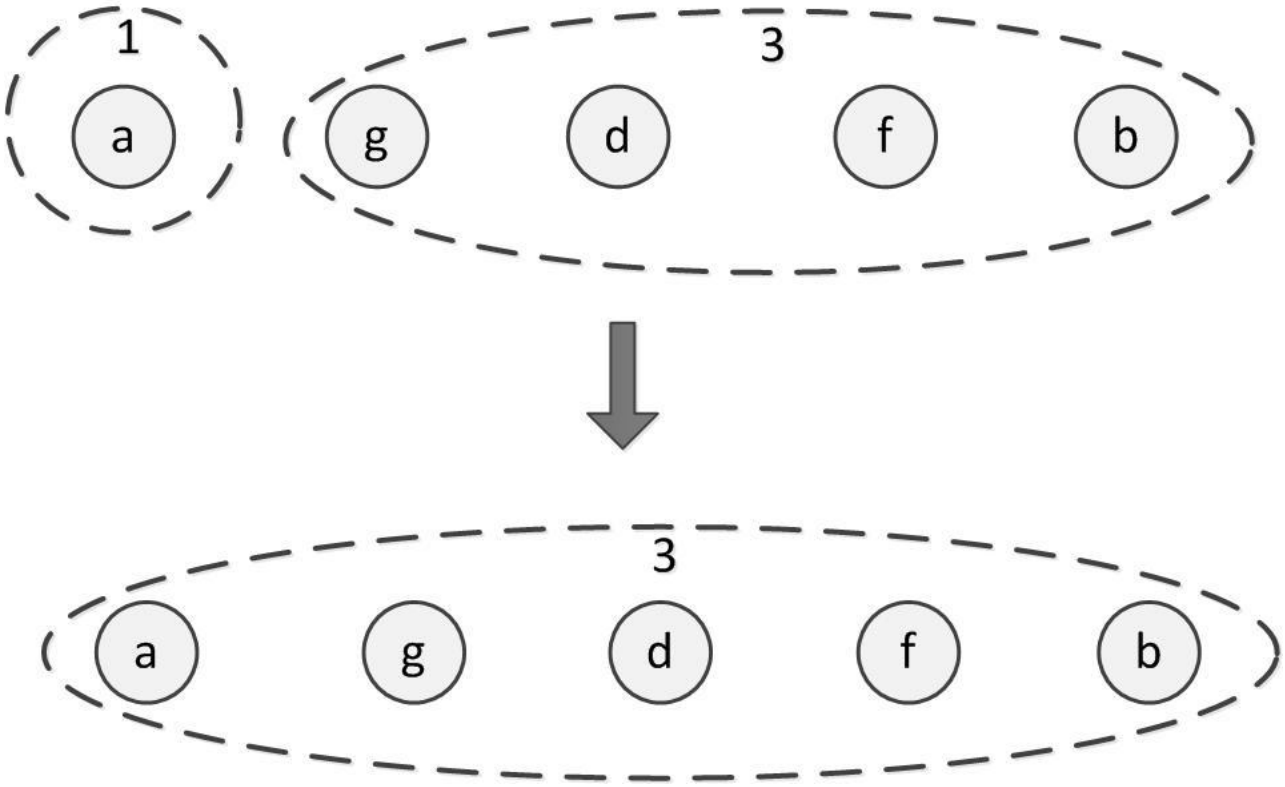
Реализация с помощью списков

- 1 *способ*. Если хранить множества в виде линейных списков с указателями на начало и конец списка, и в качестве представителя множества возвращать голову списка, то операция *Union* (*x, y*), слияние двух списков, выполняется за *O*(1), а *FindSet* (*x*), поиск элемента в списке, — за *O*(|*V*|).

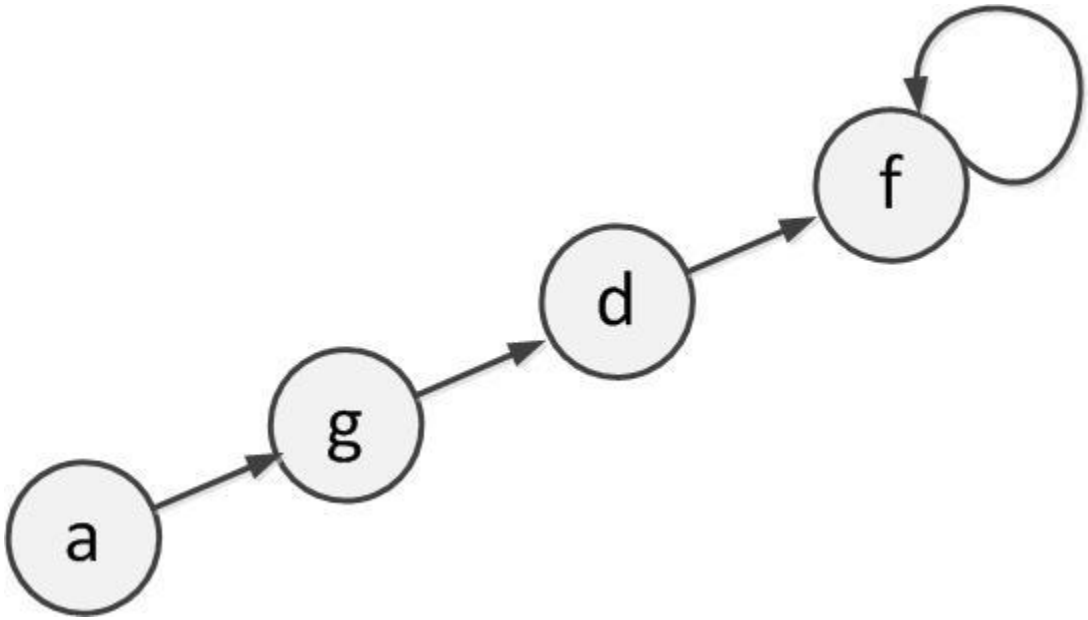
- 2 *способ*. Каждый элемент списка может содержать ссылки на следующий элемент и на первый элемент списка. Кроме того, для каждого списка хранятся указатели на его первый и последний элементы. При такой реализации операция *FindSet*(*x*) требует времени *O*(1). При выполнении операции *Union* (*x, y*) список, содержащий элемент *y*, добавляется к концу списка, содержащего элемент *x*. При этом требуется установить правильные указатели на начало списка для всех бывших элементов множества, содержащего *y*. Время на выполнение операции *Union* (*x, y*) линейно зависит от размера множества, которому принадлежит *y,* т.е. составляет *O*(|*V*|).

Весовая эвристика

Весовая эвристика — это улучшение простой реализации, в которой следует перекрашивать элементы из множества меньшей мощности. В этой реализации для каждого множества из СНМ необходимо хранить его мощность.

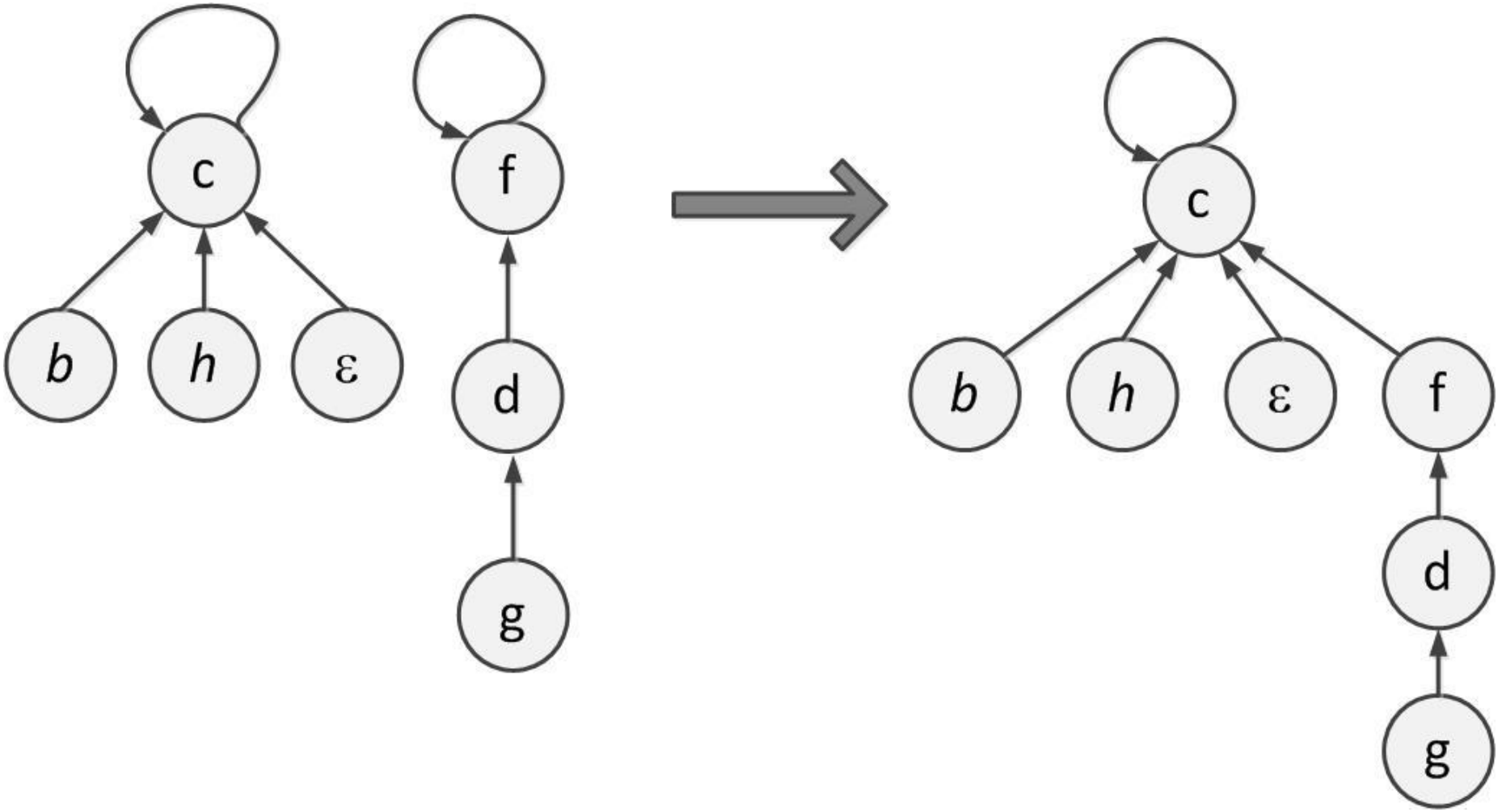


Реализация с использованием дерева



Применение весовой эвристики

(вес вершины – количество узлов в ее поддереве)



Если размер дерева равен k, то его высота не более $\lfloor \log k \rfloor$.

Доказательство по индукции:

для k = 1 утверждение верно.

Пусть теперь объединяются два дерева размеров k_1 и k_2 ; тогда по предположению индукции их высоты меньше либо равны, соответственно, $\lfloor \log k_1 \rfloor$ и $\lfloor \log k_2 \rfloor$. Не теряя общности, считаем, что первое дерево — большее ($k_1 \geq k_2$), поэтому после объединения глубина получившегося дерева из $k_1 + k_2$ вершин станет равна:

$h = \max(\lfloor \log k_1 \rfloor , 1 + \lfloor \log k_2 \rfloor)$.

Чтобы завершить доказательство, надо показать, что:

$$h \leq \lfloor \log (k_1 + k_2) \rfloor$$

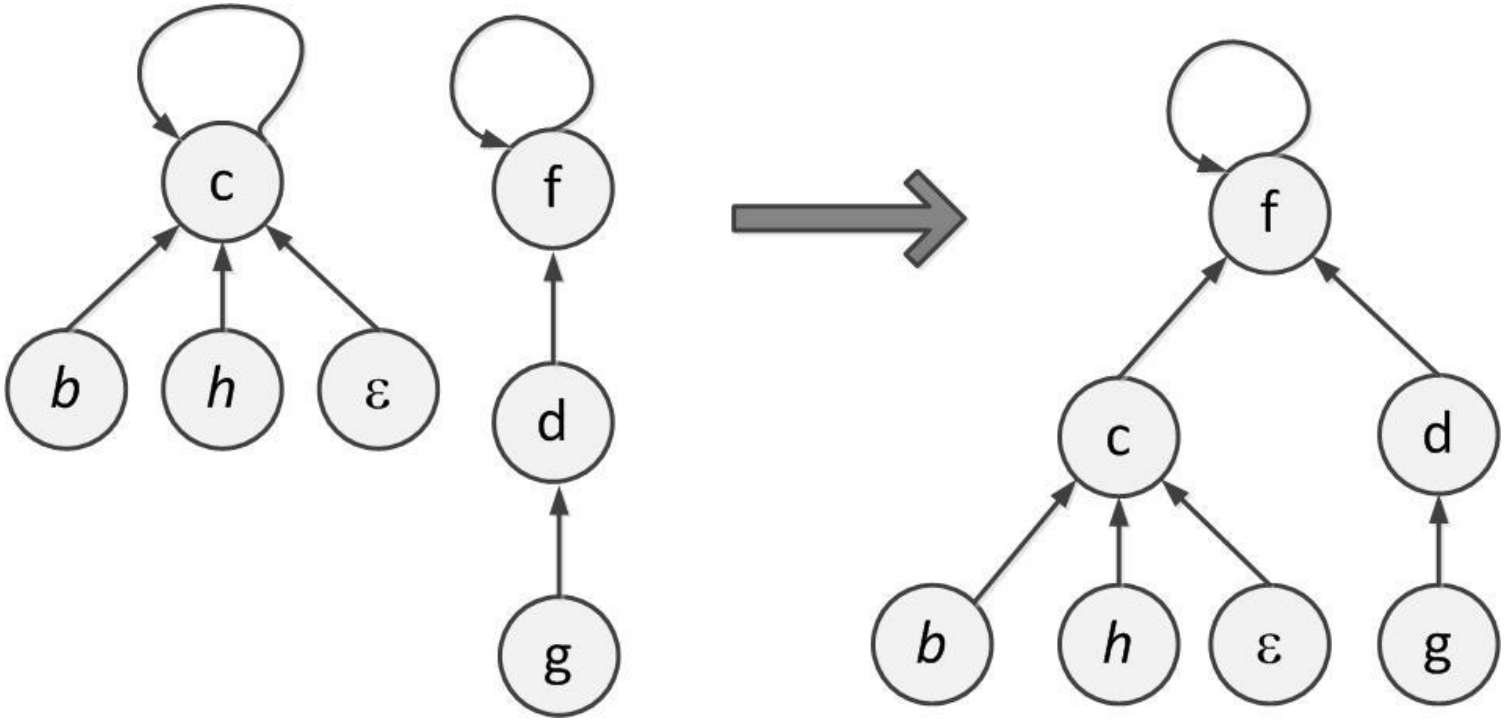
$2^h = \max (2^{\lfloor \log k1 \rfloor} , 2^{\lfloor \log k2 \rfloor}) \leq 2^{\lfloor \log (k1 + k2) \rfloor}$,

что есть почти очевидное неравенство, поскольку

$$k_1 \leq k_1 + k_2 \text{ и } 2k_2 \leq k_1 + k_2$$

Эвристика объединением по рангу

(ранг вершины – высота ее поддерева)



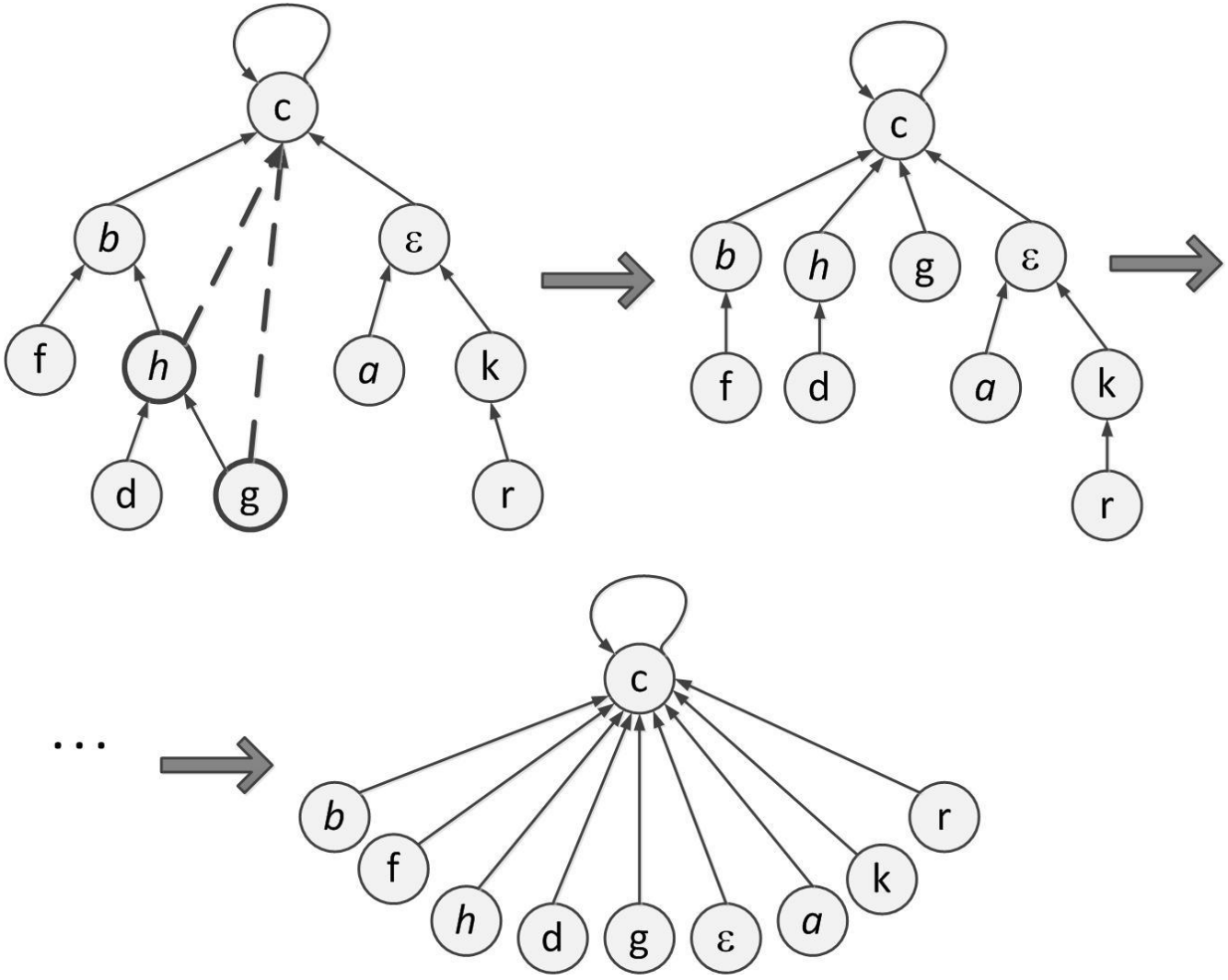
При применении ранговой эвристики получаем дерево высоты O(*log n*)

Покажем, что если ранг дерева равен *k,* то это дерево содержит как минимум 2^k вершин (отсюда будет автоматически следовать, что ранг, а, значит, и глубина дерева, есть величина O(log n)).

Доказательство по индукции:

- для k = 0 это очевидно.
- Ранг дерева увеличивается с k–1 до k, когда к нему присоединяется дерево ранга k–1; применяя к этим двум деревьям размера k–1 предположение индукции, получаем, что новое дерево ранга *k* действительно будет иметь как минимум 2^k вершин, что и требовалось доказать.

Эвристика сжатия путей



Пример реализации СНМ

```
const int MAXN = 1000;
int p[MAXN], rank[MAXN];
void makeset (int x)
{
   p[x] = x; rank[x] = 0;
}
int find_set (int x)
{
   if (x == p[x]) return x;
   return p[x] = find_set (p[x]);
}
void union (int x, int y)
{
   x = find_set (x);
   y = find_set (y);
   if (x == y) return;
   if (rank[x] > rank[y])
    p[y] = x;
   else
   {
    p[x] = y;
    if (rank[x] == rank[y])
        ++rank[y];
   }
}
```

Итог

При совместном применении эвристик сжатия пути и объединения по рангу время работы на один запрос получается в среднем $O(\alpha(n))$, где $\alpha(n)$ — обратная функция Аккермана, которая растёт очень медленно, настолько медленно, что для всех разумных ограничений она **не превосходит 4** (примерно для $n \leq 10^{600}$).

Именно поэтому про асимптотику работы системы непересекающихся множеств уместно говорить "почти константное время работы".