МГТУ им. Н.Э. Баумана
ТЕХНОПАРК
Mail.Ru Group

Лекция 6.
Графы 2.

Алгоритмы и структуры данных

Мацкевич С.Е.

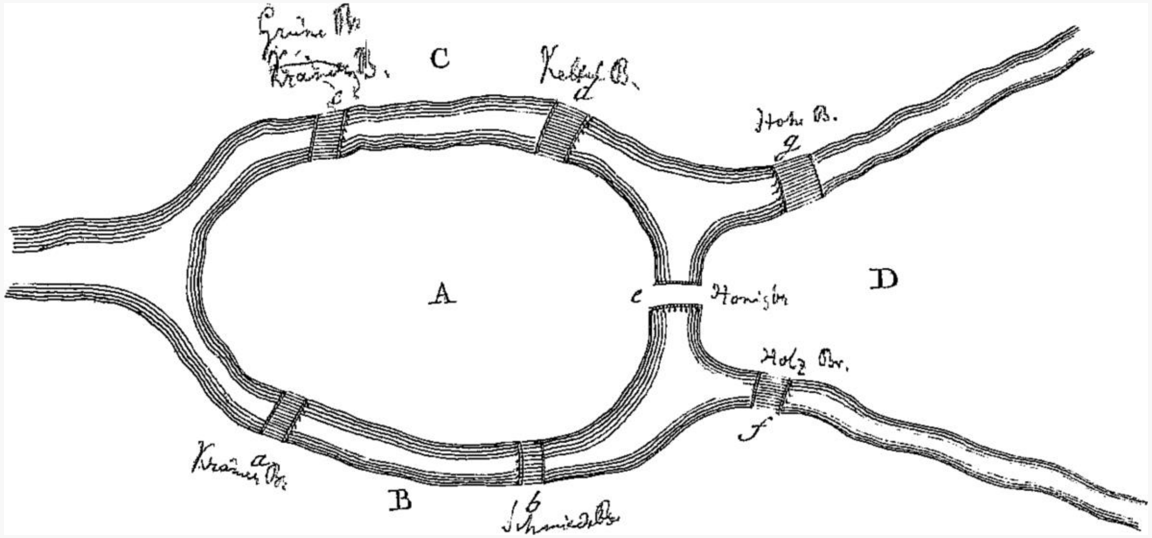
План лекции 6 «Графы 2»

1. Эйлеровы графы.
2. Задачи поиска кратчайших путей.
    1. Алгоритм Дейкстры.
    2. Алгоритм A*.
    3. Алгоритм Беллмана-Форда.
3. Минимальные остовные деревья.
    1. Алгоритм Прима.
    2. Алгоритм Крускала.

Эйлеровы графы

Задача Эйлера: Найти путь (цикл), проходящий по всем ребрам графа один раз.

Кёнигсбергский мосты:



Эйлеровы графы

Определение. Эйлеров путь – реберно-простой путь, проходящий через все ребра графа.

Эйлеров цикл определяется аналогично.

Утверждение. G – связный неориентированный граф.

- Эйлеров путь существует тогда и только тогда, когда в G не более двух нечетных вершин.
- Эйлеров цикл существует тогда и только тогда, когда в G все вершины четные.

Задачи поиска кратчайших путей

G – ориентированный взвешенный граф.

1. SPSP (Single Pair Shortest Path problem) – поиск кратчайшего пути между двумя вершинами.
   Алгоритмы Дейкстры, A, IdaStar.*
2. SSSP (Single Source Shortest Paths problem) – поиск кратчайших путей из выделенной вершины до всех остальных.
   Алгоритмы Дейкстры, Беллмана-Форда.
3. APSP (All Pairs Shortest Paths problem) – поиск кратчайших путей между всеми парами вершин.
   Алгоритмы Флойда, Джонсона.

Алгоритм Дейкстры

G – ориентированный взвешенный граф, s – стартовая вершина, $w(u, v) \geq 0$ – веса ребер неотрицательны.

Процедура релаксации ребра:

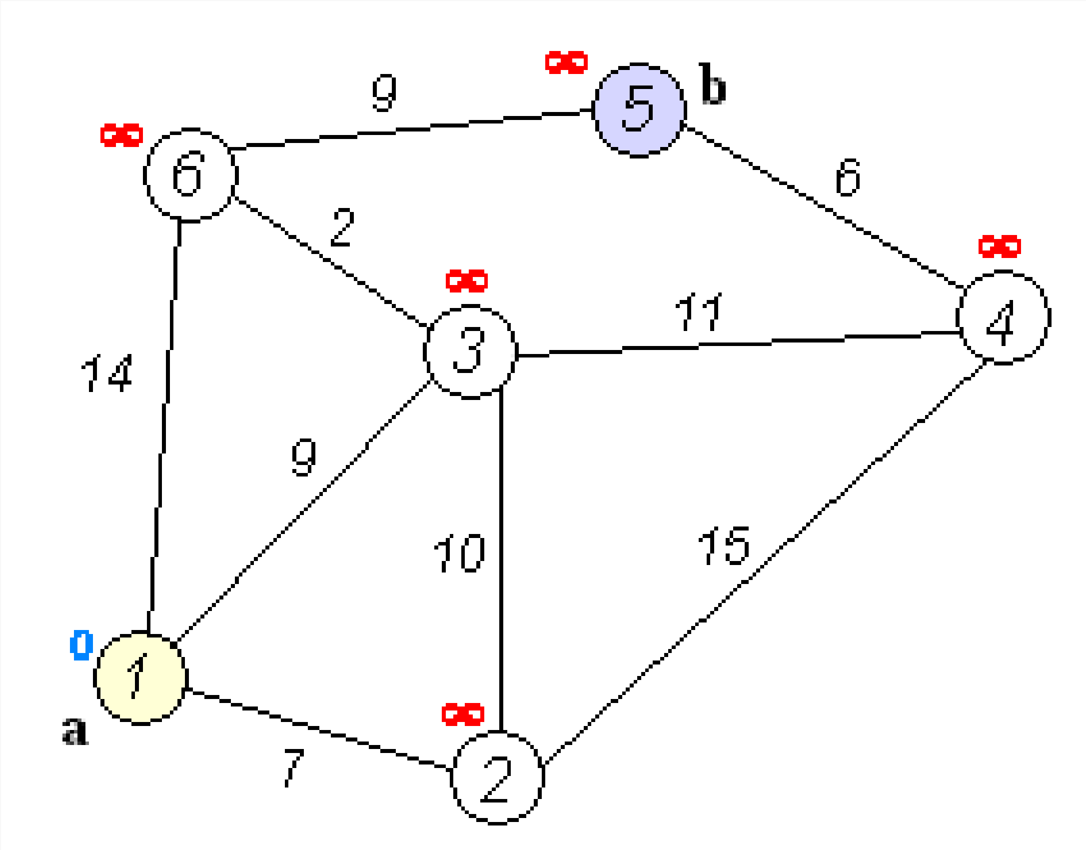
```
void Relax( u, v ) {
    if( d[v] > d[u] + w( u, v ) ) {
        d[v] = d[u] + w( u, v );
        pi[v] = u;
    }
}
```

Алгоритм Дейкстры

Алгоритм Дейкстры похож на BFS, но вместо обычной очереди используется очередь с приоритетом.

```
void Dijkstra( G, s ) {
    pi[V] = -1;
    d[V] = INT_MAX;
    d[s] = 0;
    priority_queue<int> q; q.push( s );
    while( !q.empty() ) {
        u = q.top(); q.pop();
        for( ( u, v ) : ребра из u ) {
            if( d[v] == INT_MAX ) {
                d[v] = d[u] + w(u, v);
                q.push( v );
            } else if(Relax( u, v )) {
                q.DecreaseKey( v, d[v] );
        }   }
    }
}
```

Алгоритм Дейкстры



Алгоритм Дейкстры

Оценка времени работы. Будем использовать в качестве реализации очереди с приоритетом двоичную кучу.

- Добавление узла: $O(\log V)$. Не более V операций.
- Уменьшение значения ключа: $O(\log V)$. Не более E операций.

Всего $T = O\big((E + V) \cdot logV\big)$.

Используемая память $M = O(V)$.

Алгоритм Дейкстры

Важная тонкость.

Как найти узел v в двоичной куче, чтобы вызывать

q.DecreaseKey(v, d[v]) ?

Решение 1. Хранить позицию i в куче для каждого узла v. В момент изменения позиции в куче обновлять эту позицию, хранящуюся в описании узла.

Решение 2. Использовать вместо бинарной кучи множество set<pair<d, v>>.
Во время релаксации удалять старую пару <d[v], v>, добавлять новую.

Алгоритм Дейкстры

Утверждение. G – ориентированный граф с неотрицательными ребрами. Алгоритм Дейкстры на таком G работает корректно.

Доказательство. Докажем, что в момент извлечения узла v из очереди выполняется $d[v] = \rho(s, v)$.

Индукция по числу вершин.

Рассмотрим v – вершину из кучи с минимальным d[v] в некоторый момент, v извлекается алгоритмом Дейкстры.

Рассмотрим кратчайший путь $s \rightsquigarrow v$. Пусть x – последняя на этом пути вершина, которая была извлечена на пути в x. u – следующая после x

$$d[x] = \rho(s, x),$$
$$d[u] \geq d[v],$$
$$d[u] \leq d[x] + w(x, u) = \rho(s, x) + w(x, u) = \rho(s, u).$$

Итого:

$$d[v] \leq d[u] \leq \rho(s, u) \leq \rho(s, v).$$

Потенциал

Введем функцию на вершинах графа – **потенциал**:

$$\pi : V \to \mathbb{R}$$

Определим измененный вес ребра, добавив разность потенциалов:

$$w'(u,v) = w(u,v) + \pi(v) - \pi(u).$$

Как изменятся длины путей?

$$w'(p) = \sum w'(x_{i-1}, x_i) = \sum \big(w(x_{i-1}, x_i) + \pi(x_i) - \pi(x_{i-1})\big) =$$

$$= \sum w(x_{i-1}, x_i) + \pi(x_k) - \pi(x_0) =$$

$$= w(p) + \pi(x_k) - \pi(x_0)$$

Потенциал

То есть длина пути изменится на разность потенциалов в конечной и начальной точке.

<u>Следствие 1.</u>

Кратчайшие пути останутся кратчайшими в новой весовой функции w'.

И наоборот, кратчайшие пути относительно w' будут кратчайшими относительно w.

Можно применять алгоритмы поиска кратчайших путей для измененной весовой функции на разность потенциалов.

Потенциал

Пример 1. $\pi[v] = -\rho(s, v)$

Такой потенциал обнуляет ребра, лежащие на кратчайших путях из вершины s.

$$w'(u, v) = w(u, v) - \rho(s, v) + \rho(s, u) \geq 0$$

Кроме того, на обновленном графе можно применить алгоритм Дейкстры уже из любой вершины, достижимой из s.

Потенциал

Пример 2. $\pi[v] = \rho(v, t)$

Такой потенциал обнуляет ребра, лежащие на кратчайших путях к вершине t.

$$w'(u, v) = w(u, v) + \rho(v, t) - \rho(u, t) \geq 0$$

Кроме того, на обновленном графе можно применить алгоритм Дейкстры уже из любой вершины, достижимой из s.

Более того, алгоритм будет «сразу» находить кратчайший путь к вершине t.

Алгоритм A*

$$\pi[v] = \varepsilon(v, t)$$

– эвристика, оценивающая длину пути от v до t. $\varepsilon(t, t) = 0$.

Будем записывать $\varepsilon(u) = \varepsilon(u, t)$.

Чтобы алгоритм Дейкстры работал корректно, необходима неотрицательность ребер:

$$w'(u, v) = w(u, v) + \varepsilon(v, t) - \varepsilon(u, t) \geq 0.$$

Это условие называется «монотонностью».

<u>Определение.</u> Эвристика *монотонна*, если

$$\varepsilon(u) \leq w(u, v) + \varepsilon(v).$$

Похоже на неравенство треугольника.

Алгоритм A*

Определение. Эвристика ε *допустима*, если

$$\varepsilon(u) \leq \rho(u, t).$$

Похоже на неравенство треугольника.

Утверждение. Монотонная эвристика допустима.

Доказательство.

По индукции.

Алгоритм A*

Взглянем на алгоритм Дейкстры с потенциалами.

```
void Dijkstra( G, s ) {
    pi[V] = -1;
    d[V] = INT_MAX;
    d[s] = 0;
    priority_queue<int> q; q.push( s );
    while( !q.empty() ) {
        u = q.top(); q.pop();
        for( ( u, v ) : E ) {
            if( d’[v] == INT_MAX )
                q.push( v );
            if( d’[v] > d’[u] + w’(u, v) ) {
                d’[v] = d’[u] + w’(u, v);
                pi[v] = u;
                q.DecreaseKey( v, d’[v] );
            }
        }
    }
}
```

Алгоритм A*

Модифицируем

```
void Dijkstra( G, s ) {
    pi[V] = -1;
    d[V] = INT_MAX;
    d[s] = 0;
    priority_queue<int> q; q.push( s );
    while( !q.empty() ) {
        u = q.top(); q.pop();
        for( ( u, v ) : E ) {
            if( d[v] == INT_MAX )
                q.push( v );
            if( d[v] + e[v] – e[s] > d[u] + w(u, v) + e[v] – e[s] ) {
                d[v] = d[u] + w(u, v);
                pi[v] = u;
                q.DecreaseKey( v, d[v] + e[v] – e[s] );
            }
        }
    }
}
```

Алгоритм A*

Модифицируем еще чуток. Получился A*

```
void Dijkstra( G, s ) {
    pi[V] = -1;
    d[V] = INT_MAX;
    d[s] = 0;
    priority_queue<int> q; q.push( s );
    while( !q.empty() ) {
        u = q.top(); q.pop();
        for( ( u, v ) : E ) {
            if( d[v] == INT_MAX )
                q.push( v );
            if( d[v] > d[u] + w(u, v) ) {
                d[v] = d[u] + w(u, v);
                pi[v] = u;
                q.DecreaseKey( v, d[v] + e[v] );
            }
        }
    }
}
```

Алгоритм Беллмана-Форда

SSSP

G – ориентированный взвешенный граф.

Если в графе нет циклов отрицательного веса, достижимых из s, то алгоритм Беллмана-Форда находит все кратчайшие пути из s до остальных вершин.

Если в графе есть достижимый из s цикл отрицательного веса, то алгоритм Беллмана-Форда сообщит о его наличии. (и может выдать его).

Алгоритм Беллмана-Форда

V-1 раз релаксируем все ребра.

```
bool BellmanFord( G, s ) {
    for( int i = 0; i < V; ++i ) {
        for( (u, v) : E ) Relax( u, v );
    }
    // Детектирование цикла.
    for( (u, v) : E ) {
        if( Relax( u, v ) )
            return false;
    }
    return true;
}
```

Алгоритм Беллмана-Форда

Утверждение. Если в G нет циклов отрицательного веса, достижимых из s, то алгоритм Беллмана-Форда вернет true и

$$d(v) = \rho(s, v).$$

Доказательство по индукции.

Утверждение. (Поиск циклов отрицательного веса) Если алгоритм вернул false, то цикл можно найти, пройдя по предкам от ребра, релаксировавшегося на V-ом шаге. Путь по предкам не дойдет до s, но зациклится на цикле отрицательного веса.

Остовные деревья

Пусть G – связные неориентированный граф.

Определение. $T \subset G$ – **остовное дерево**, если T содержит все вершины G и является деревом.

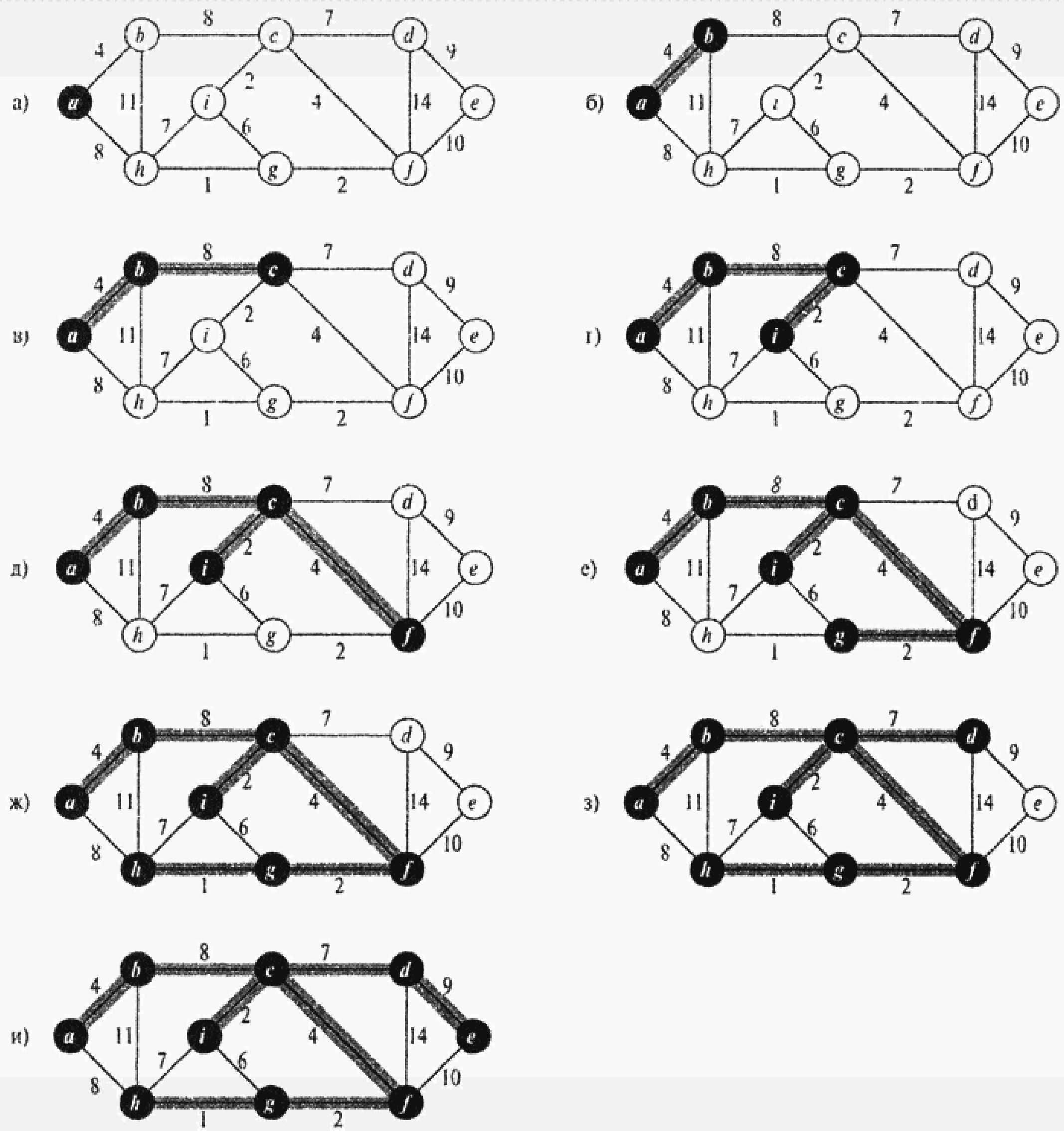
Определение. **Минимальным остовным деревом** во взвешенном неориентированном графе называется остовное дерево минимального веса.

Алгоритм Прима

Алгоритм Прима – алгоритм поиска минимального остовного дерева.

1) Выбираем произвольную стартовую вершину – начало строящегося дерева. Строим очередь с приоритетом вершин, до которых есть ребро от стартовой. Приоритет = длина ребра до стартовой.
2) Итеративно добавляем к дереву вершину из очереди с ребром до дерева, имеющим вес = приоритет вершины. Добавляем новые вершины, доступные из добавленной вершины по ребрам графа с соответствующими приоритетами. Либо обновляем приоритет вершины в очереди, если от добавленной вершины к ней ведет более легкое ребро.

Алгоритм Прима



Алгоритм Прима. Оценка

В качестве очереди с приоритетом для вершин можно использовать кучу или сбалансированное дерево поиска, как в алгоритме Дейкстры.

В этом случае одна итерация состоит из

1) Извлечение вершины из очереди с приоритетом – $O(\log V)$
2) Обработка ребер, инцидентных извлеченной вершине – $O(\log V)$ на каждое ребро.

Пункт 1) будет выполнять V раз.

Пункт 2) будет выполняться E раз.

Общая оценка времени работы: $T = O(E \log V)$.

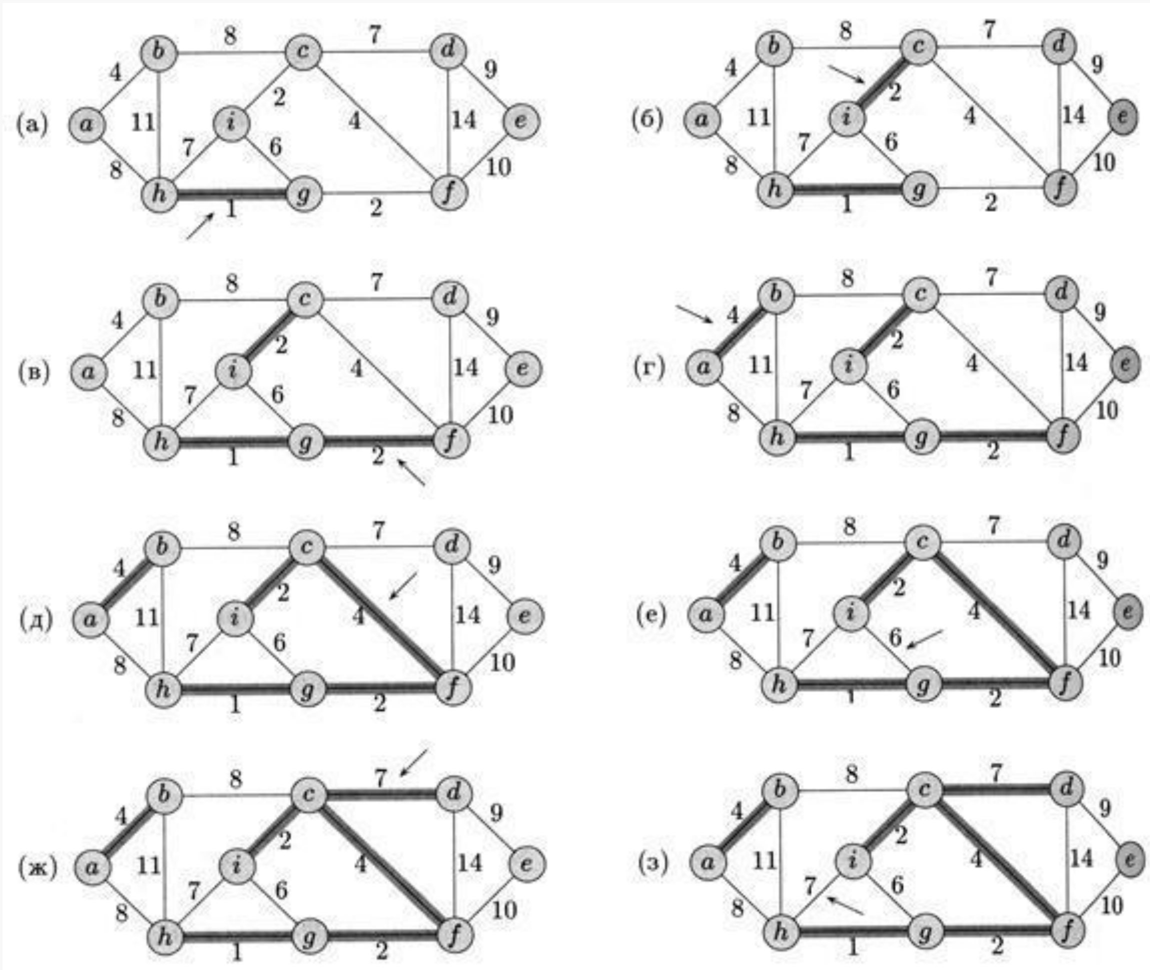
Оценка доп.памяти: $M = O(\log V)$.

Алгоритм Крускала

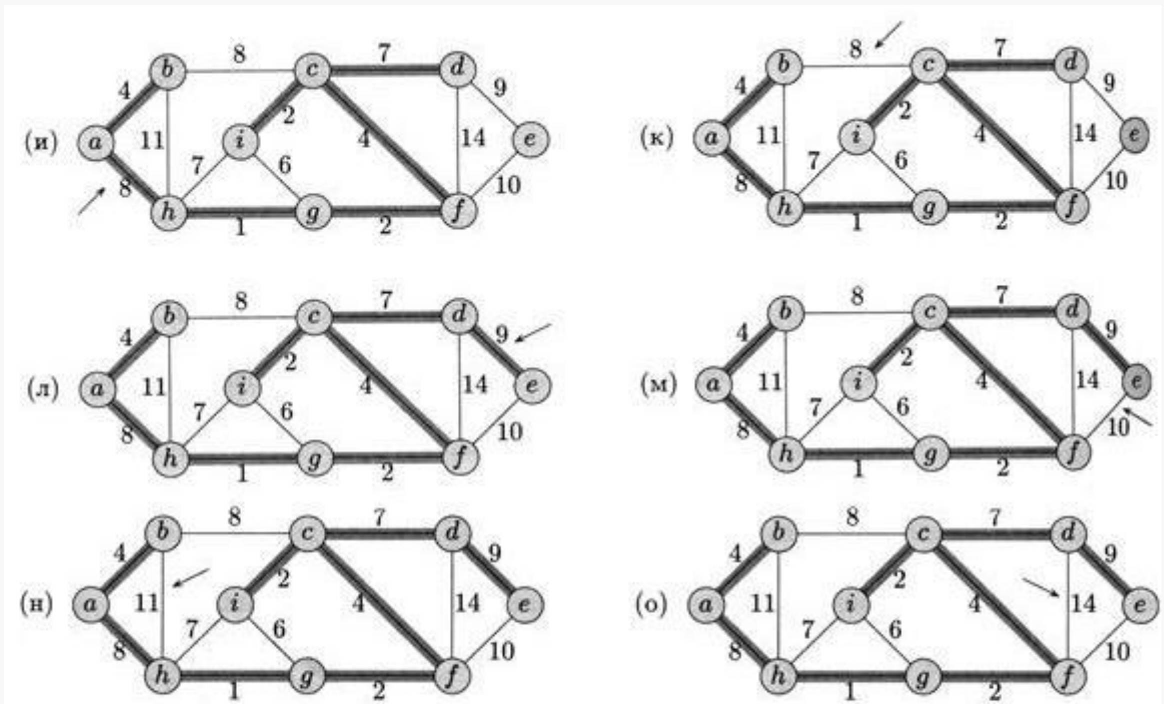
Алгоритм Крускала – еще один алгоритм поиска минимального остовного дерева.

1) Сортируем все ребра графа по весу.
2) Инициализируем лес деревьев. Изначально каждая вершина – маленькое дерево (пень).
3) Последовательно рассматриваем ребра графа в порядке возрастания веса.
   Если очередное ребро соединяет два разных дерева из леса, то объединяем эти два дерева этим ребром в одно дерево.
   Если очередное ребро соединяет две вершины одного дерева из леса, то пропускаем такое ребро.
   Повторяем 3), пока в лесу не останется одно дерево.

Алгоритм Крускала



Алгоритм Крускала



Алгоритм Крускала. СНМ

Требуется СД «Система Непересекающихся Множеств» (СНМ=DSU) с операциями

1) Create(u) – создать множество с одним элементом u.
2) Find(u) – найти множество по элементу u, чтобы можно было сравнить их (Find(u) == Find(v)).
3) Union(u, v) – объединить два множества, одно из которых содержит элемент u, а другое – элемент v.

Алгоритм Крускала. СНМ

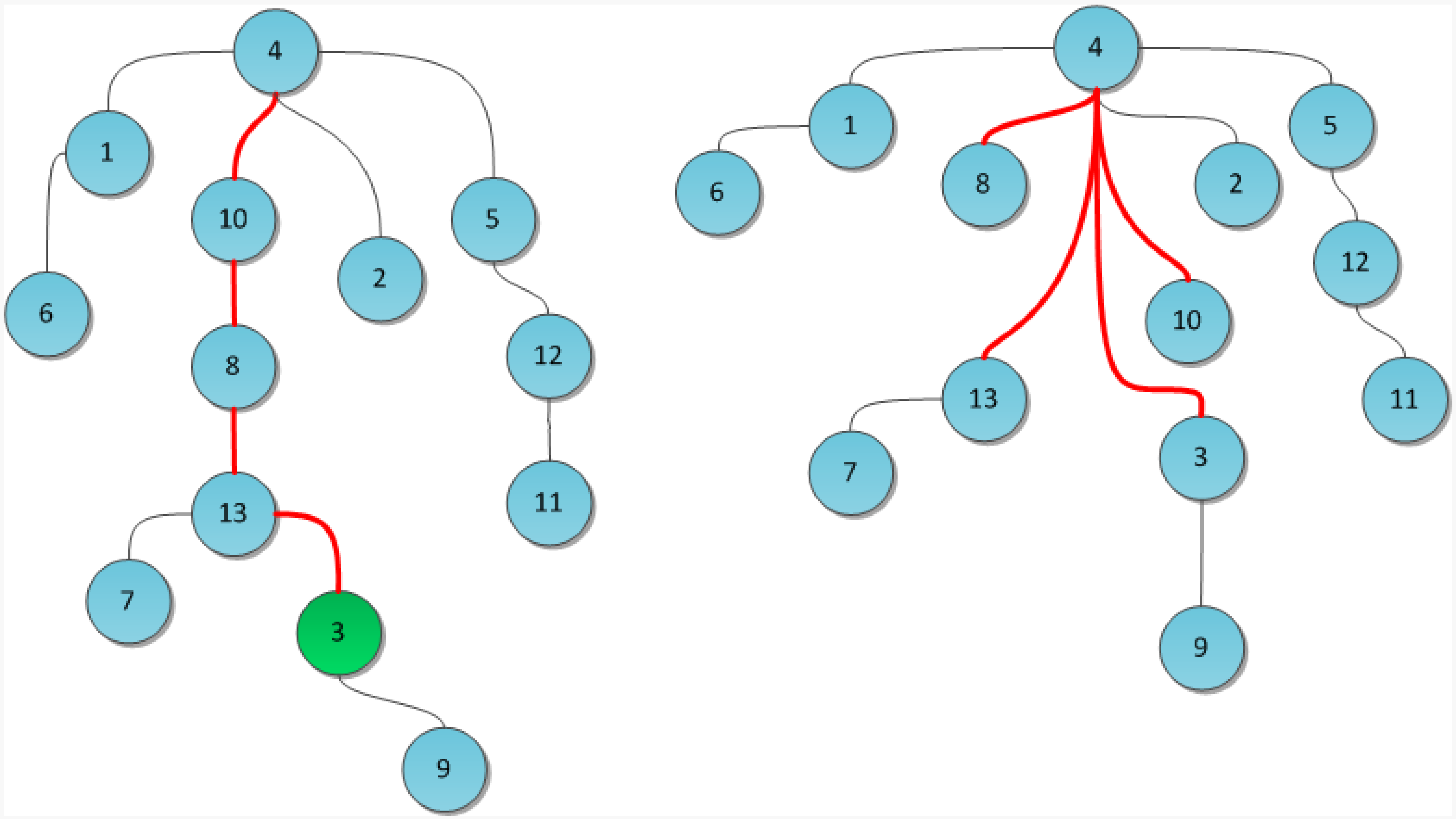
Предлагается хранить элементы каждого множества в виде дерева – каждый элемент имеет указатель на родителя. Корень такого дерева называется представителем.

Представитель возвращается методом Find. Сравнение деревьев – сравнение представителей.

Объединение двух деревьев – одному представителю назначается родитель = другой представитель.

На следующем слайде показана работа метода Find, которая кроме возвращения представителья выполняет «сжатие пути». Сжатие пути – в каждой вершине на пути к представителю обновляется родитель = представитель. Это существенно ускоряет работу СНМ.

Алгоритм Крускала. СНМ



Алгоритм Крускала. СНМ

```
// Сжатие пути
Node* Find(Node* node) {
  if (node->parent == 0) return node;
  return node->parent = Find(node->parent);
}
```

Алгоритм Крускала. Оценка

Время работы алгоритма складывается из сортировки ребер – $O(E \log V)$ и из цикла, перебирающего ребра.

Работа методов Find и Union в СНМ требует $O\left(A^{-1}(V)\right) \approx O(1)$ времени для эффективно реализованной СНМ со сжатием пути и правильным выбором «какое дерево к какому подключать» в методе Union.

Общее время работы = $O(E \log V)$.

Замечание про алгоритмы Прима и Крускала

Корректность алгоритмов доказывается с помощью «Леммы о Безопасном ребре».

http://neerc.ifmo.ru/wiki/index.php?title=Остовные_деревья:_определения,_лемма_о_безопасном_ребре

МГТУ
им. Н.Э. Баумана
ТЕХНОПАРК
Mail.Ru Group

???

Спасибо за внимание!