ФАКУЛТЕТ ЗА ИНФОРМАТИЧКИ НАУКИ И КОМПЈУТЕРСКО ИНЖЕНЕРСТВО

Графови - алгоритми

Алгоритми и податочни структури

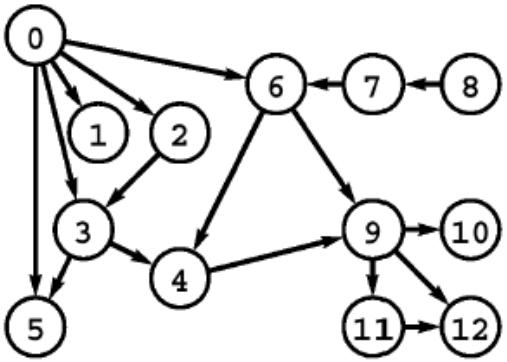
Аудиториска вежба 11

Графови - алгоритми

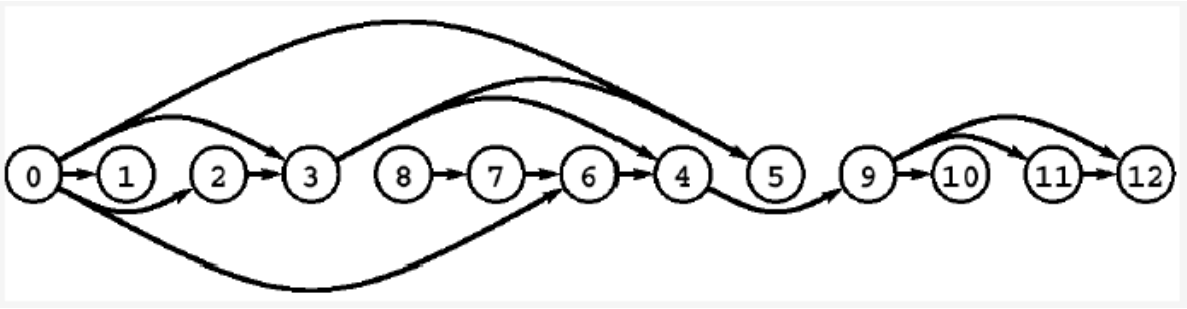
- Тополошко сортирање
- Алгоритам на Крускал
- Алгоритам на Прим
- Алгоритам на Дијкстра

Тополошко сортирање

- Directed acyclic graph (Насочен Ацикличен Граф)



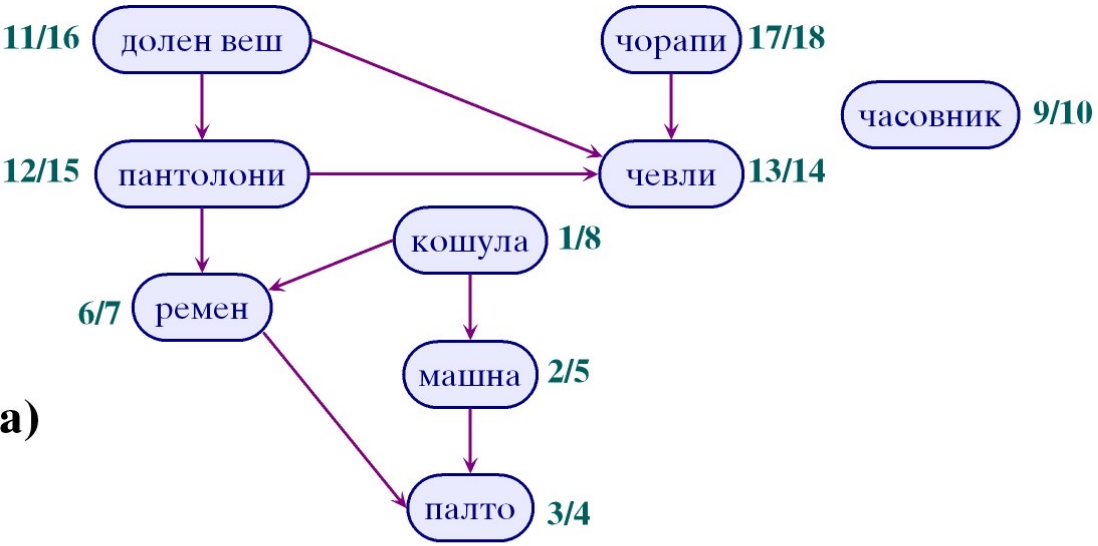
- Проблем на тополошко сортирање: да се подредат јазлите во графот така што сите ребра да покажуваат од лево кон десно



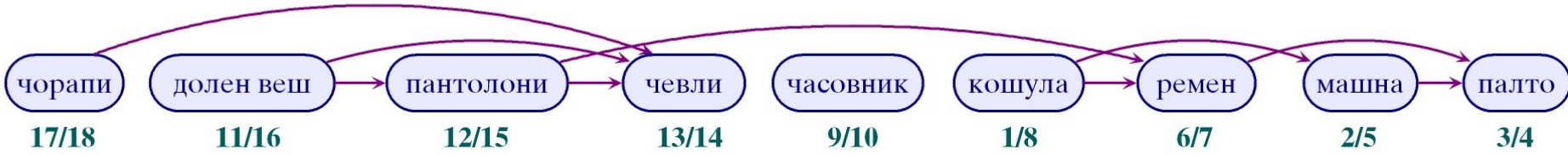
Тополошко сортирање - илустрација

- За дадено множество задачи за кои постојат услови за предност, по кој редослед треба да се извршуваат задачите?

а)



б)



Тополошко сортирање - Java

```java
private void topologicalSortUtil(T vertex, Set<T> visited, Stack<T> stack) {
    visited.add(vertex);
    for (T neighbor : getNeighbors(vertex)) {
        if (!visited.contains(neighbor)) {
            topologicalSortUtil(neighbor, visited, stack);
        }
    }
    stack.push(vertex);
}
```

Забелешка:
Имплементацијата се надоврзува на дефинираната структура за граф (насочен и нетежински) со листа на соседност

Тополошко сортирање - Java

```java
public List<T> topologicalSort() {
    Stack<T> stack = new Stack<>();
    Set<T> visited = new HashSet<>();

    for (T vertex : adjacencyList.keySet()) {
        if (!visited.contains(vertex)) {
            topologicalSortUtil(vertex, visited, stack);
        }
    }

    List<T> order = new ArrayList<>();
    while (!stack.isEmpty()) {
        order.add(stack.pop());
    }
    return order;
}
```

Алгоритми кај тежински графови

- Минимално распнувачко дрво (Minimum spanning tree - MST)
- Најкратко (со најмала тежина) поврзување на сите темиња во еден граф

Минимално распнувачко дрво

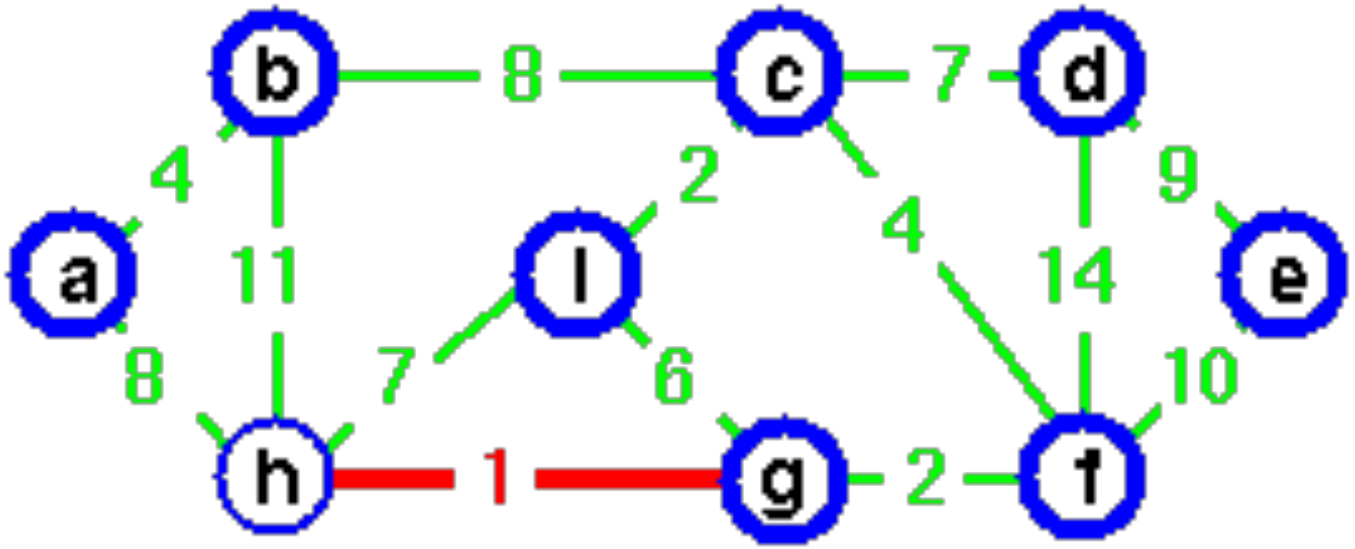
- **Дрво претставник на граф (распнувачко дрво)** е подмножество од множеството на сите ребра *E,* така што ребрата во тоа подмножество креираат дрво кое ги поврзува сите темиња графот. Во добиениот подграф нема кружни патеки (циклуси). Притоа, од посебен интерес е дрво – претставник на граф чиј што збир на тежини на ребрата е минимален.
- Пример: да се одреди која е должината на најмалиот вкупен пат кој ги поврзува градовите во некоја област.
- Бројот на врски кај дрвото е $|V| - 1$.
- Алгоритми за добивање на минимално распнувачко дрво во граф се алгоритмите на **Крускал** (Kruskal) и **Прим** (Prim). И двата алгоритми се претставници на архетипот алчни алгоритми.

Алгоритам на Крускал

- **Алгоритмот на Крускал** започнува да го гради минималното распнувачко дрво така што на почеток графот го гледа како шума од $|V|$ дрва (секој јазел е посебно дрво).
- Во секоја итерација од алгоритмот се зема реброто од графот кое има најмала тежина од сите ребра кои сеуште не се вметнати во минималното дрво. Ако тоа ребро поврзува два јазли кои спаѓаат во иста поврзана компонента (дрво), тогаш не се става во минималното распнувачко дрво, бидејќи ќе создаде циклус кој ќе ја уништи карактеристиката на дрво. Ако не, тоа се додава во него и на тој начин две дрва се поврзуваат во едно поголемо дрво. Алгоритмот завршува кога минималното распнувачко дрво ќе се пополни со точно $|V| - 1$ врска.
- Алгоритмот има алчен пристап бидејќи во секоја итерација ја разгледува врската со најмала тежина.

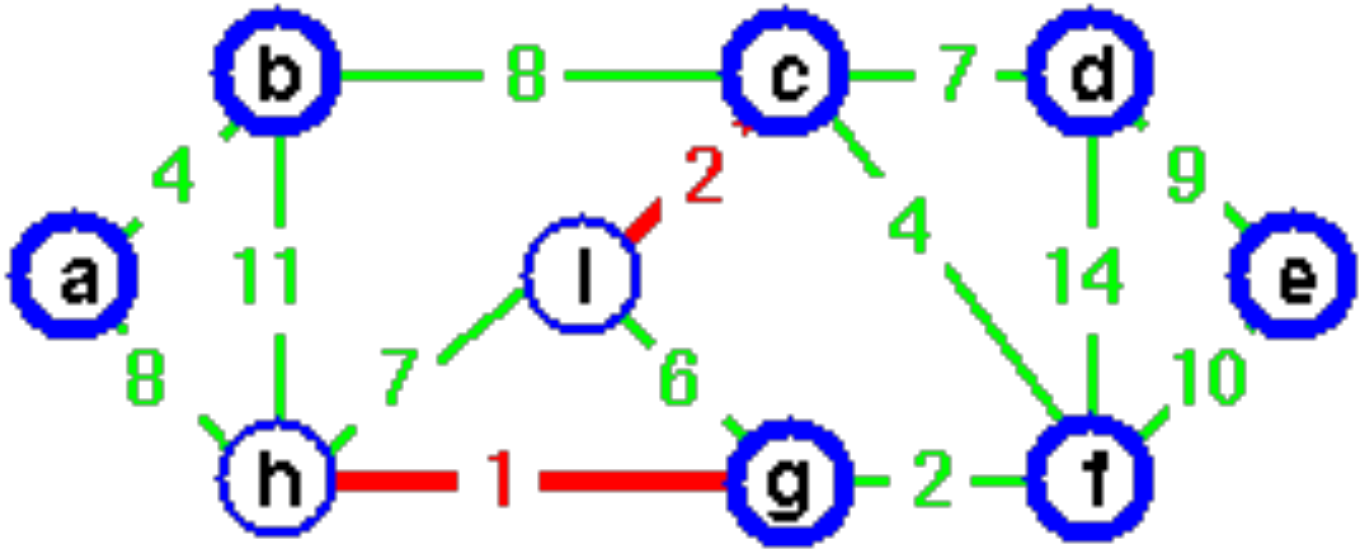
Алгоритам на Крускал - илустрација

- Чекор 1. Во графот се избира реброто со најмала тежина. Тоа е реброто (g, h).



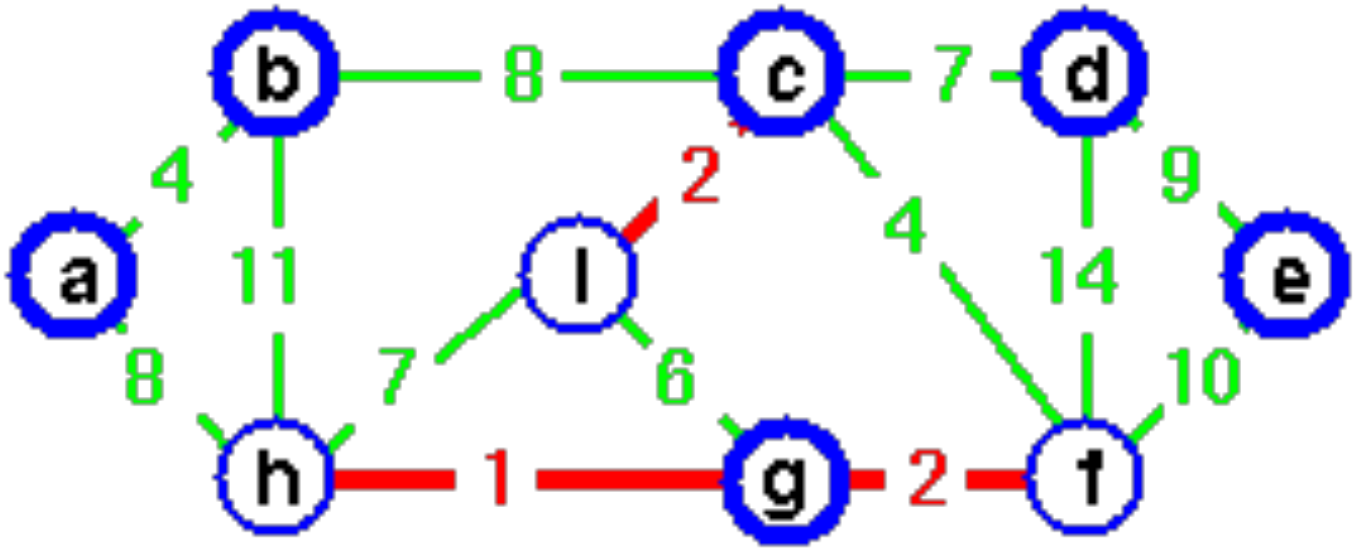
Алгоритам на Крускал - илустрација

- Чекор 2. Следното ребро кое има најмала тежина е (c, i). Реброто (c, i) креира второ дрво.



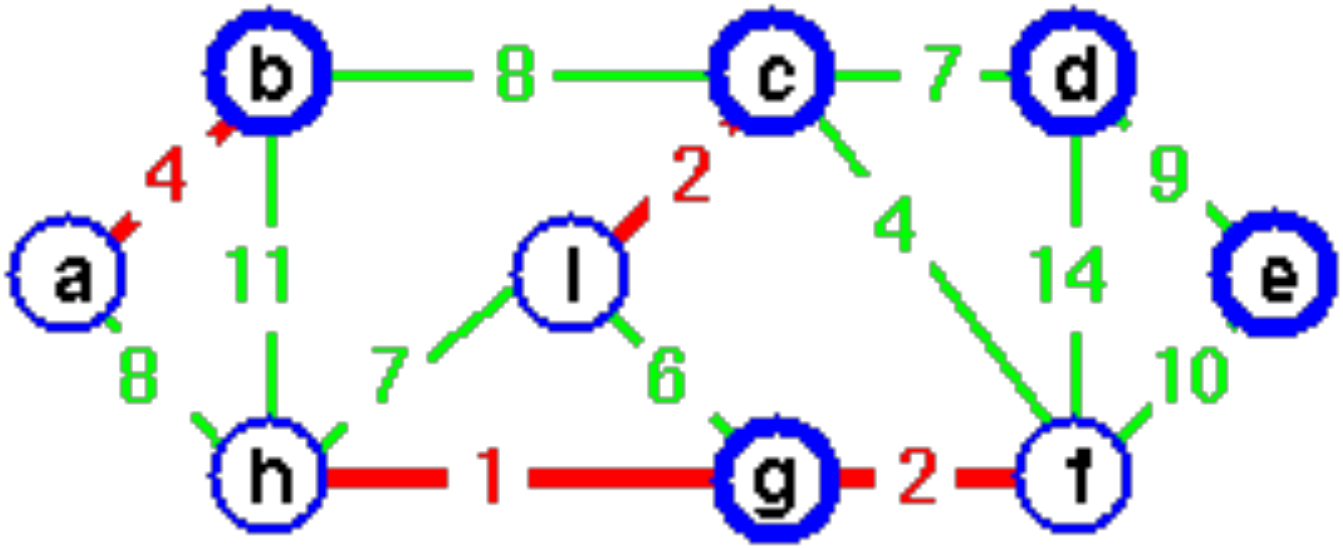
Алгоритам на Крускал - илустрација

- Чекор 3. Реброто (g, f) има најмала тежина. Ова ребро се додава кон првото дрво.



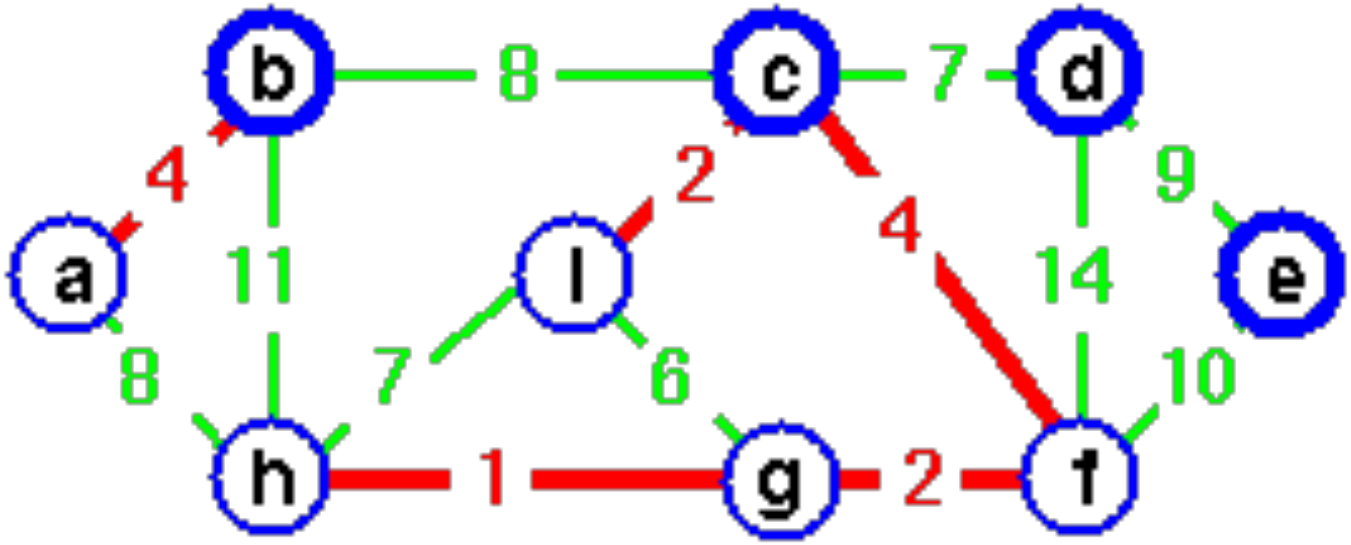
Алгоритам на Крускал - илустрација

- Чекор 4. Реброто (a, b) креира трето дрво.



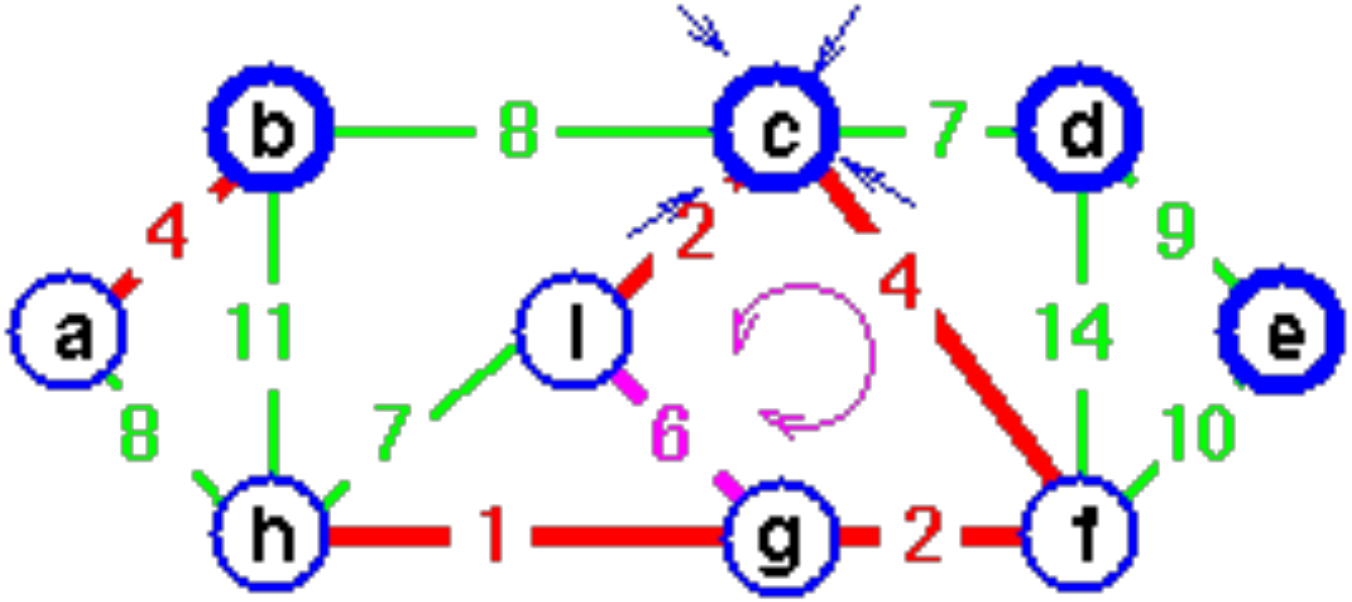
Алгоритам на Крускал - илустрација

- Чекор 5. Се додава реброто (c, f) и се спојуваат првите две дрва.



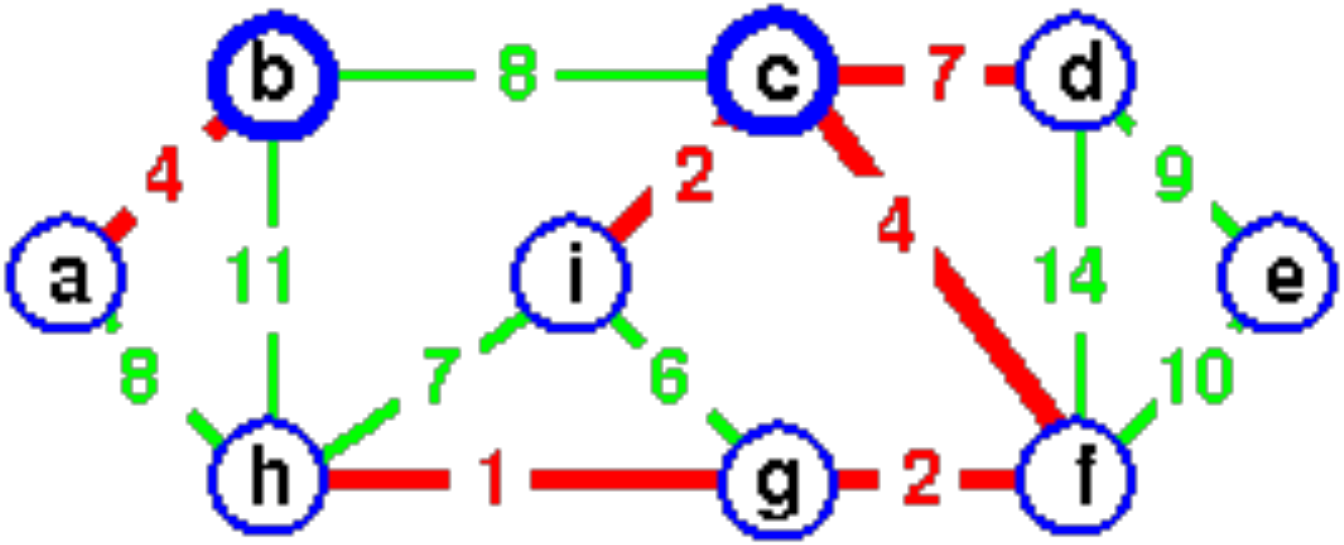
Алгоритам на Крускал - илустрација

- Чекор 6. Следното ребро кое има најмала тежина е (g, i), но ако се додаде ќе се создаде циклус (јамка).



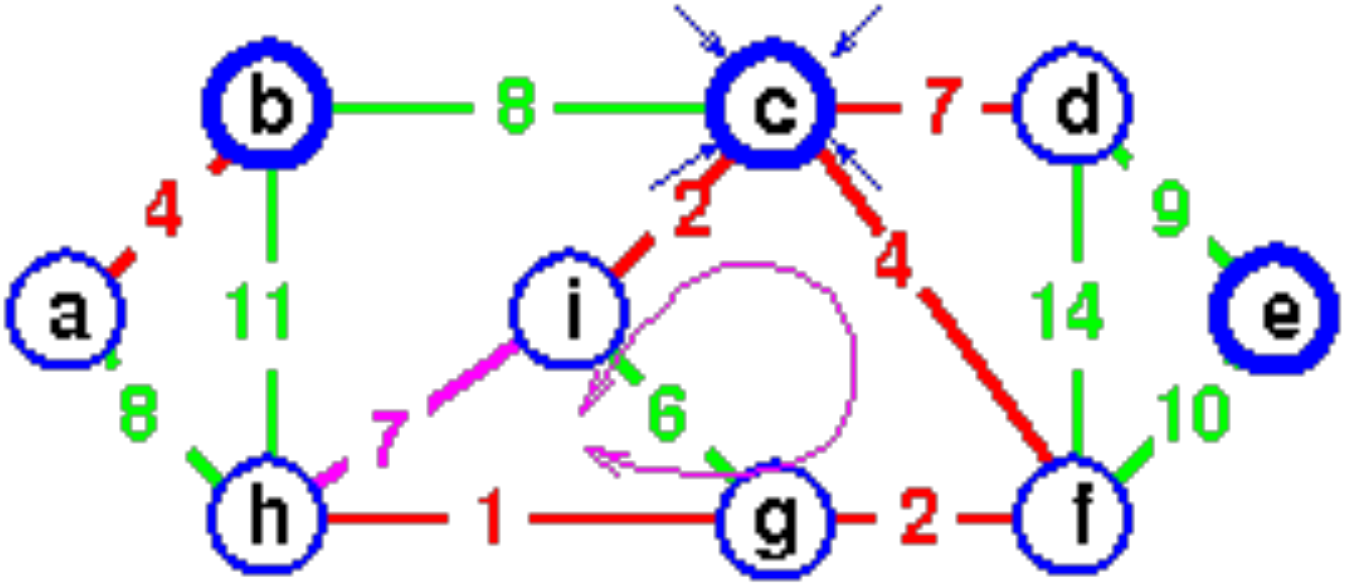
Алгоритам на Крускал - илустрација

- Чекор 7. Наместо реброто (g, i), се додава реброто (c, d).



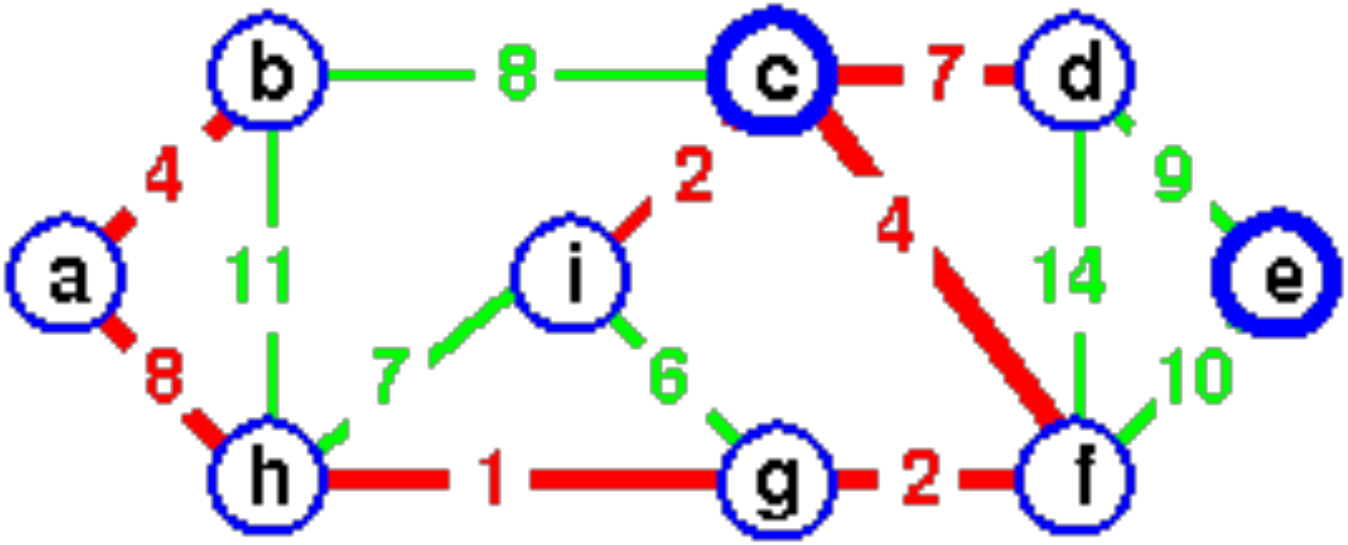
Алгоритам на Крускал - илустрација

- Чекор 8. Ако се додаде реброто (h, i) ќе се создаде циклус (јамка).



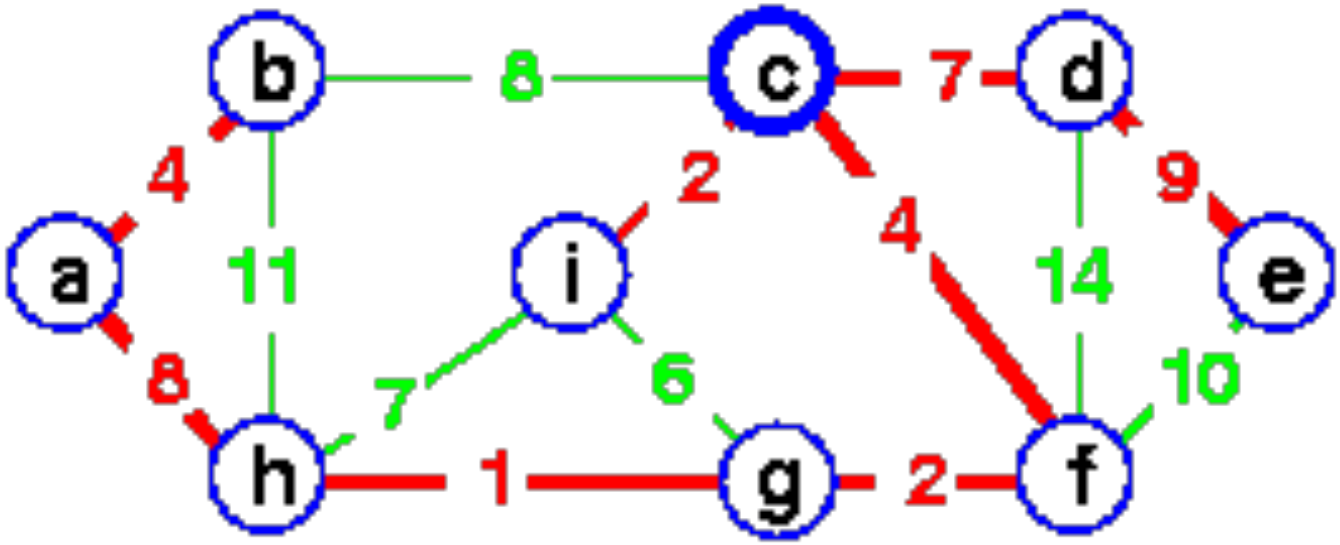
Алгоритам на Крускал - илустрација

- Чекор 9. Наместо реброто (h, i) се додава реброто (a, h).



Алгоритам на Крускал - илустрација

- Чекор 10. Повторно, ако се додаде реброто (b, c), ќе се создаде циклус (јамка). Се додава реброто (d, e).



Класа Edge

```
public class Edge {
    private int fromVertex, toVertex;
    private int weight;
    public Edge(int from, int to, int weight) {
        this.fromVertex = from;
        this.toVertex = to;
        this.weight = weight;
    }

    public int getFrom() {
        return this.fromVertex;
    }
    public int getTo() {
        return this.toVertex;
    }
    public int getWeight() {
        return this.weight;
    }
}
```

Алгоритам на Крускал - Java

```
private List<Edge> getAllEdges() {
    List<Edge> edges = new ArrayList<>();

    for(int i=0;i<numVertices;i++) {
        for(int j=0;j<numVertices;j++) {
            if(isEdge(i,j)) {
                edges.add(new Edge(i, j, matrix[i][j]));
            }
        }
    }

    return edges;
}
```

Забелешка:
Имплементацијата се надоврзува на дефинираните структури за тежински графови со матрица на соседност

Алгоритам на Крускал - Java

```
private void union(int u, int v, int[] trees) {
    int findWhat, replaceWith;
    if(u<v) {
        findWhat = trees[v];
        replaceWith = trees[u];
    } else {
        findWhat = trees[u];
        replaceWith = trees[v];
    }

    for(int i=0;i<trees.length;i++) {
        if(trees[i] == findWhat) {
            trees[i] = replaceWith;
        }
    }
}
```

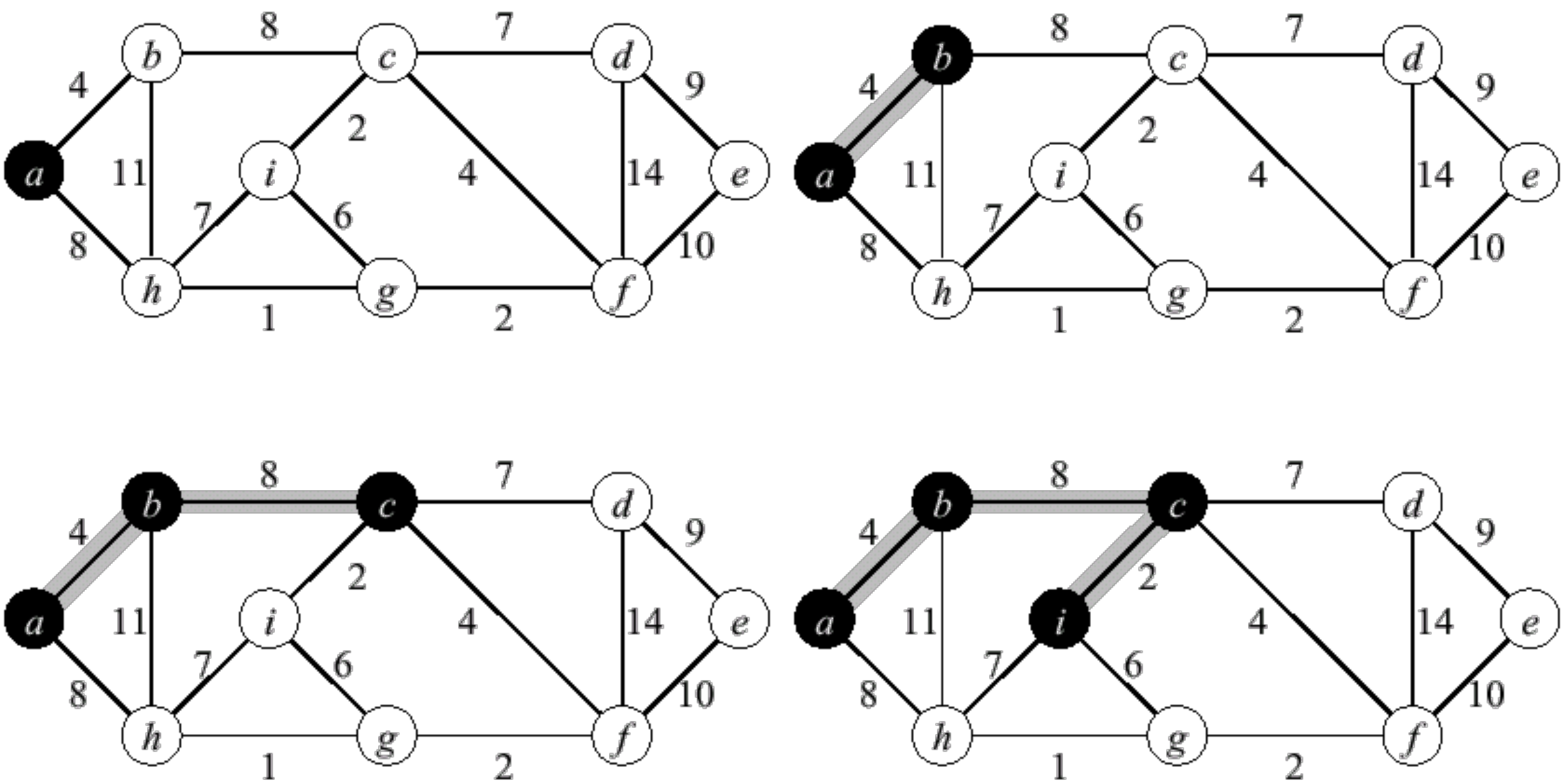
Алгоритам на Крускал - Java

```java
public List<Edge> kruskal() {
    List<Edge> mstEdges = new ArrayList<>();
    List<Edge> allEdges = getAllEdges();

    allEdges.sort(Comparator.comparingInt(Edge::getWeight));

    int trees[] = new int[numVertices];

    for(int i=0;i<numVertices;i++)
        trees[i] = i;

    for(Edge e: allEdges) {
        if(trees[e.getFrom()] != trees[e.getTo()]) {
            mstEdges.add(e);

            union(e.getFrom(), e.getTo(), trees);
        }
    }

    return mstEdges;
}
```

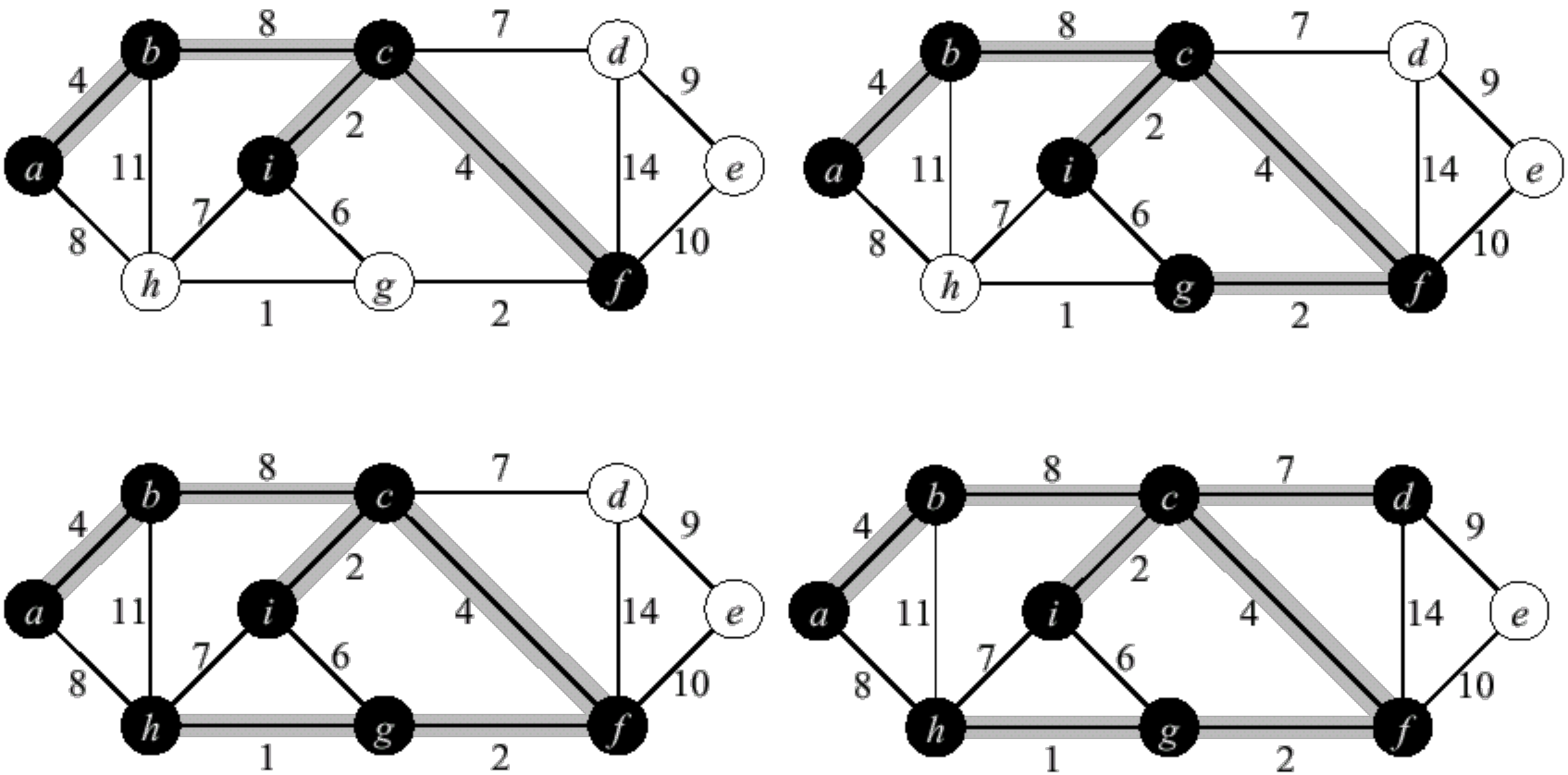
Алгоритам на Прим

- **Алгоритмот на Прим** се разликува од алгоритмот на Крускал по тоа што дрвото започнува да се гради од еден јазел и во секоја итерација се додаваат јазли на едно дрво кое постојано се доградува. Откако ќе се избере произволно почетно теме, кон него се додаваат ребра од множеството на ребра на темињата кои се споени со него. Во следните чекори се разгледуваат сите ребра кои не припаѓаат на веќе изграденото дрво, но излегуваат од темиња што припаѓаат на него. Реброто (односно темето) што се додава кон минималното распнувачко дрво во графот е она ребро што го задоволува условот да не се спојува со теме што веќе припаѓа на минималното дрво и има најмала тежина (според алчниот алгоритам).

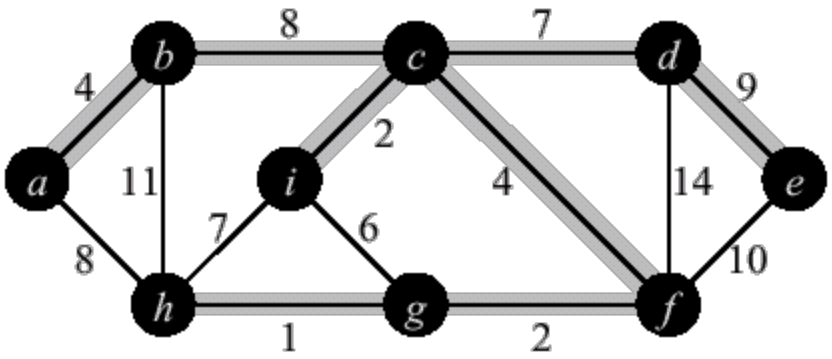
Алгоритам на Прим - илустрација



Алгоритам на Прим - илустрација



Алгоритам на Прим - илустрација



Алгоритам на Прим - Java

```
public List<Edge> prim(int startVertexIndex) {
    List<Edge> mstEdges = new ArrayList<>();
    Queue<Edge> q = new PriorityQueue<>(Comparator.comparingInt(Edge::getWeight));

    boolean included[] = new boolean[numVertices];

    for(int i=0;i<numVertices;i++) {
        included[i] = false;
    }

    included[startVertexIndex] = true;

    for(int i=0;i<numVertices;i++) {
        if(isEdge(startVertexIndex,i)) {
            q.add(new Edge(startVertexIndex, i, matrix[startVertexIndex][i]));
        }
    }

    while(!q.isEmpty()) {
        Edge e = q.poll();

        if(!included[e.getTo()]) {
            included[e.getTo()] = true;
            mstEdges.add(e);
            for(int i=0;i<numVertices;i++) {
                if(!included[i] && isEdge(e.getTo(),i)) {
                    q.add(new Edge(e.getTo(), i, matrix[e.getTo()][i]));
                }
            }
        }
    }

    return mstEdges;
}
```

Забелешка: Имплементацијата се надоврзува на дефинираните структури за тежински графови со матрица на соседност

Најкратка патека меѓу јазли

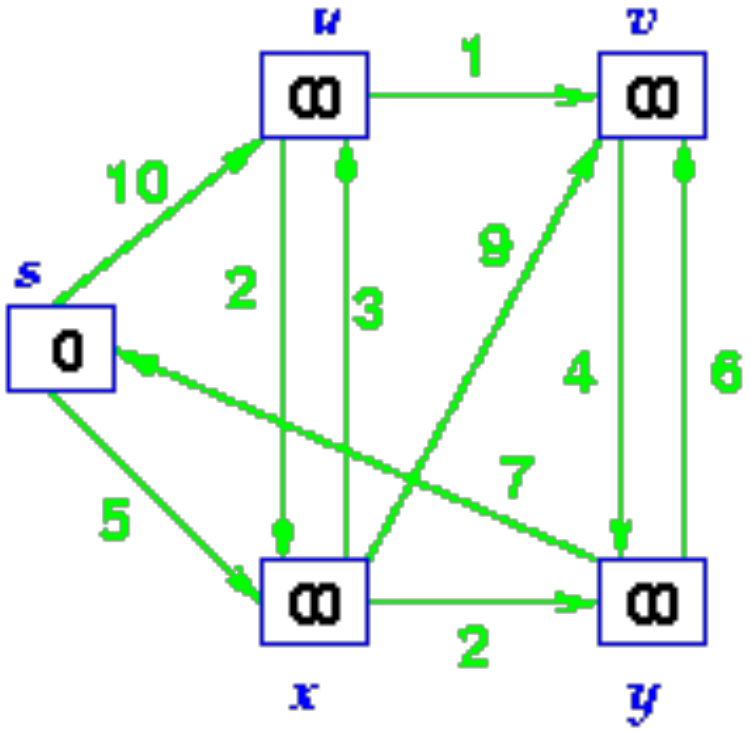
- Проблем:
  - Најкратка патека од еден до друг јазел
  - Најкратки патеки од еден јазел до сите останати
  - Најкратки патеки од сите јазли до сите останати
  - Најкратки патеки од сите јазли до еден јазел
- Кај нетежински графови – изминување по широчина.
- Кај тежински графови – алгоритам на Дијкстра (Dijkstra)

Алгоритам на Дијкстра

- **Алгоритмот на Дијкстра** претпоставува дека сите тежини на ребрата се ненегативни. Тој ги пронаоѓа најкратките патеки од еден јазел во даден тежински граф до сите други јазли, преку првенство на тежините со претпоставка дека секоја тежина е позитивен број. Идејата на овој алгоритам се базира на фактот дека секоја минимална патека од јазел X до Z е всушност проширување на друга минимална патека од X до Y со додавање на реброто помеѓу Y и Z.
- Алгоритамот на Дијкстра е типичен претставник на алгоритмите кои користат динамичко програмирање.

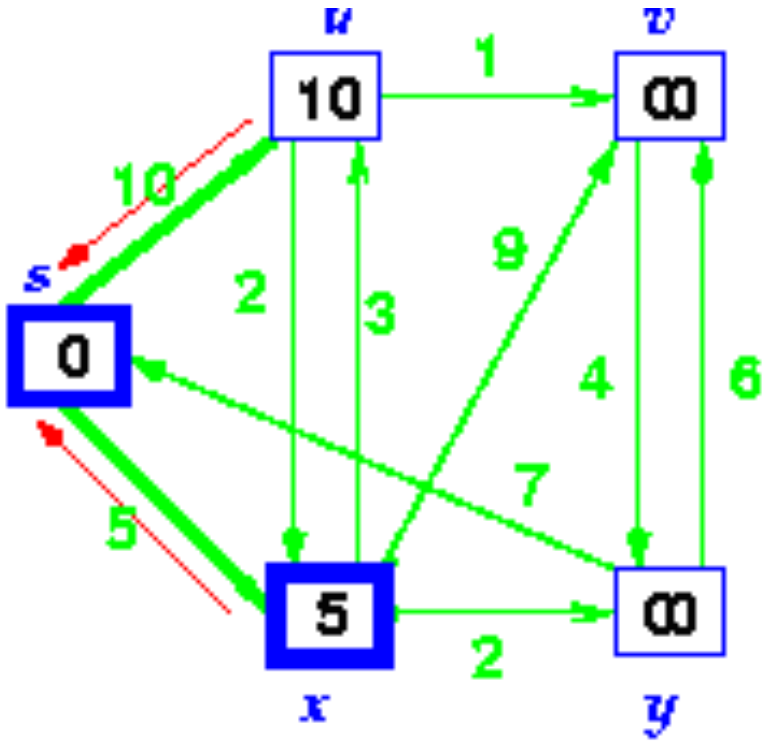
Алгоритам на Дијкстра - илустрација

- Чекор 1. За даден почетен граф G=(V, E), сите јазли немаат конечна цена на патеката од почетниот јазел s, освен почетниот јазел кој има цена 0.



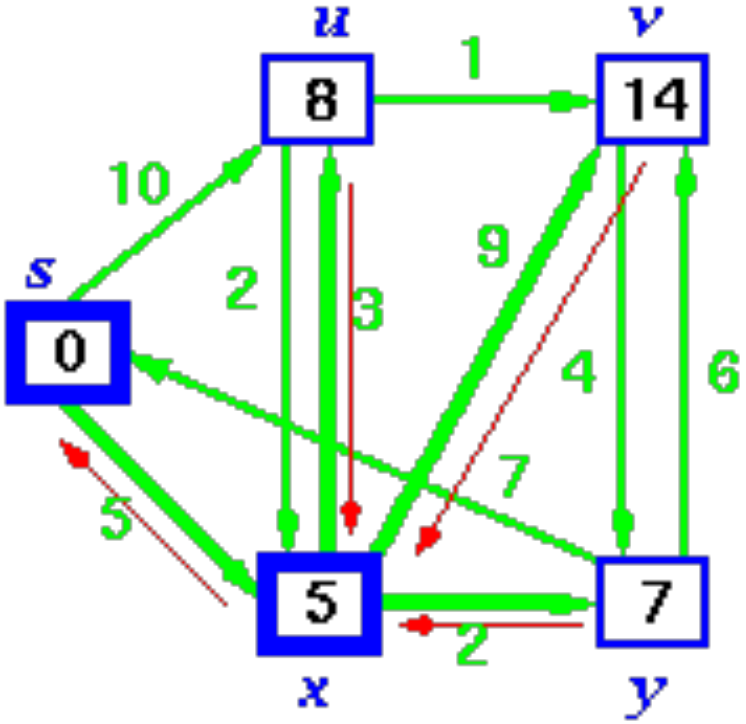
Алгоритам на Дијкстра - илустрација

- Чекор 2. Се определуваат цените на патеките кон сите соседни јазли, тоа се x и u, а нивните цени се 5 и 10 соодветно. Се избира јазел кој е најблиску до јазелот s, а тоа е јазелот x. И сега неговата цена е конечна и изнесува 5.



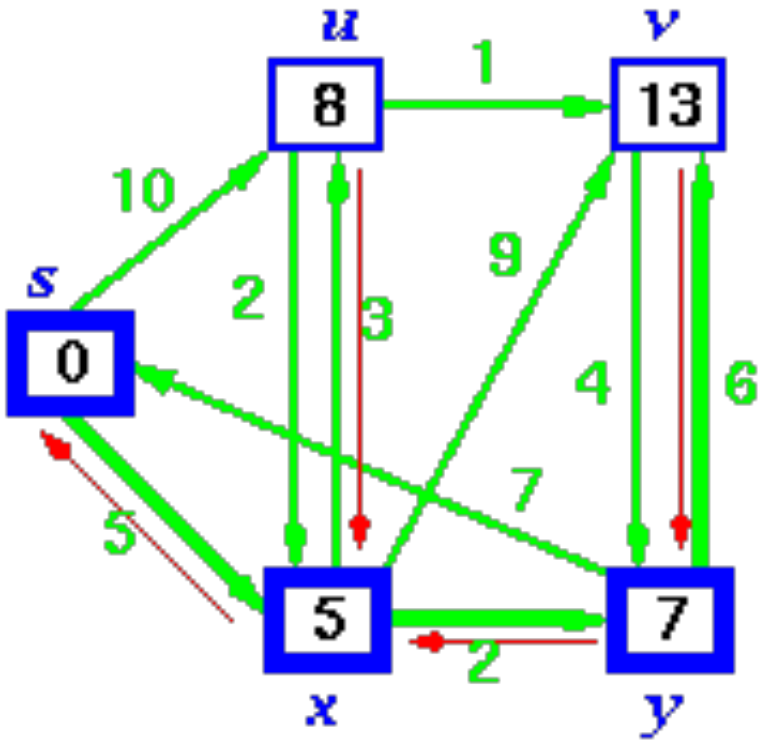
Алгоритам на Дијкстра - илустрација

- Чекор 3. Се определуваат цените на патеките кон сите соседни јазли, но сега на јазелот кој добил конечна цена во претходниот чекор, x. Соседни јазли на јазелот x се u, v и y, а нивните цени се 8, 14 и 7 соодветно.



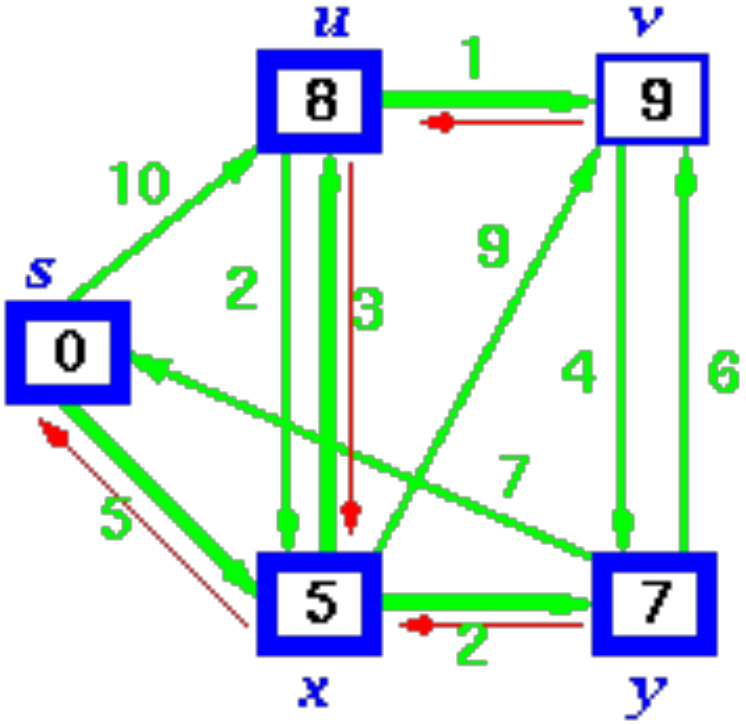
Алгоритам на Дијкстра - илустрација

- Чекор 4. Се избира јазел кој е најблиску до јазелот s, а тоа е јазелот y, неговата цена 7 е минимална. Во овој чекор јазелот y добива конечна цена.



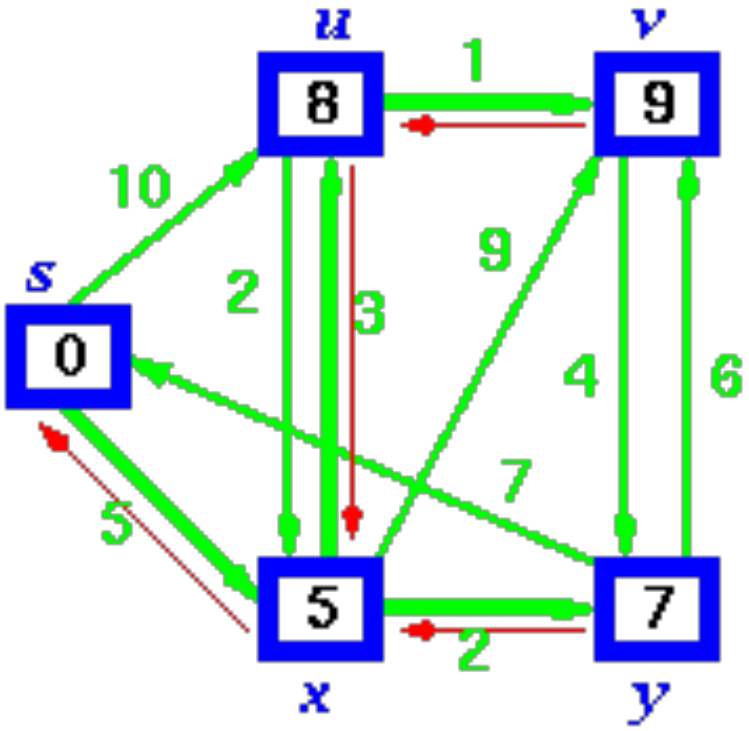
Алгоритам на Дијкстра - илустрација

- Чекор 5. Повторно се бараат цените на патеките кон сите соседни јазли на јазелот кој добил конечна цена во претходниот чекор, y. Соседен јазел на јазелот y е v со цена 13. Сега јазелот u има минимална цена, 8 и неговата цена станува конечна.



Алгоритам на Дијкстра - илустрација

- Чекор 6. На крај јазелот v добива конечна цена. Неговата цена е збир од цената на јазелот u и цената на реброто (u, v).



Алгоритам на Дијкстра - Java

```java
public Map<T, Integer> shortestPath(T startVertex) {
    Map<T, Integer> distances = new HashMap<>();
    PriorityQueue<T> queue = new PriorityQueue<>(Comparator.comparingInt(distances::get));
    Set<T> explored = new HashSet<>();

    // Initialize distances
    for (T vertex : adjacencyList.keySet()) {
        distances.put(vertex, Integer.MAX_VALUE);
    }
    distances.put(startVertex, 0);

    queue.add(startVertex);

    while (!queue.isEmpty()) {
        T current = queue.poll();
        explored.add(current);

        for (Map.Entry<T, Integer> neighborEntry : adjacencyList.get(current).entrySet()) {
            T neighbor = neighborEntry.getKey();
            int newDist = distances.get(current) + neighborEntry.getValue();

            if (newDist < distances.get(neighbor)) {
                distances.put(neighbor, newDist);

                // Update priority queue
                if (!explored.contains(neighbor)) {
                    queue.add(neighbor);
                }
            }
        }
    }

    return distances;
}
```

Забелешка: Имплементацијата се надоврзува на дефинираните структури за тежински графови со листа на соседност

Алгоритам на Дијкстра

- За дома:
  - Најкратка патека од едно до друго теме
  - Најкратка патека од секое до секое теме
  - Негативни тежини на ребрата?

Задача 1 - Поплава

При обилни врнежи настануваат поплави со кои се оштетуваат патиштата. Оштетувањата можат да бидат одрони, поплавени патишта или срушени мостови. Во таков случај патната мрежа во една држава станува дисконектирана и задача на соодветните служби е што е можно побрзо да обезбедат поврзаност на сите градови помеѓу себе. Оштетувањата на патиштата не се од иста категорија на сериозност, т.е. за секое од оштетувањата е потребно различно време за патот да се санира и да се направи прооден. Поради обемот на штетите и итноста на поправка, ве´ке е започнато воспоставување на поврзаноста и дел од патиштата се санирани. Но, со цел оптимизација на процесот, на нас ни е доделена задачата да најдеме приоритетни патишта кои треба да се санираат со цел сите градови да бидат поврзани поме´гу себе со најмалку еден пат.

Задача 1 - Поплава

Влез: Во првиот ред е даден бројот на градови, M. Во вториот ред е даден бројот на патишта ме´гу градовите, N. Во третиот ред е даден бројот на ве´ке санирани патишта. P. Во наредните M реда се дадени имињата на градовите. Во следните N реда се дадени парови на имиња на градови, проследени со цел број што претставува време кое е проценето дека е потребно за да се расчисти- /санира делницата ме´гу тие два града. Во последните P реда се дадени парови од градови каде е завршена санацијата на патиштата (пред да ни го дадат проблемот на нас) и тие се ве´ке проодни.

Излез: Во првиот ред се печатат два броја: бројот на патишта кои се останати да се санираат, и времето потребно за санација на тие патишта. Потоа се печатат сите парови на градови поме´гу кои треба да бидат поправени патиштата, секој во посебен ред.

Задача 1 - решение

```
public List<Edge> adaptedKruskal(int trees[]) {
    List<Edge> mstEdges = new ArrayList<>();
    List<Edge> allEdges = getAllEdges();

    allEdges.sort(Comparator.comparingInt(Edge::getWeight));

    for(Edge e: allEdges) {
        if(trees[e.getFrom()] != trees[e.getTo()]) {
            mstEdges.add(e);

            union(e.getFrom(), e.getTo(), trees);
        }
    }

    return mstEdges;
}
```

Задача 1 - решение

```java
import java.util.List;
import java.util.Scanner;
import java.util.HashMap;

public class Flood {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        int M = sc.nextInt();
        int N = sc.nextInt();
        int P = sc.nextInt();
        sc.nextLine();

        HashMap<String, Integer> mapping = new HashMap<>();

        AdjacencyMatrixGraph<String> cityNetwork = new AdjacencyMatrixGraph<>(M);

        int trees[] = new int[M];

        for(int i=0;i<M;i++) {
            String city = sc.nextLine();
            mapping.put(city, i);
            cityNetwork.addVertex(i, city);
            trees[i] = i+1;
        }

        for(int i=0;i<N;i++) {
            String line = sc.nextLine();
            String parts[] = line.split( regex: " ");
            cityNetwork.addEdge(mapping.get(parts[0]), mapping.get(parts[1]), Integer.parseInt(parts[2]));
        }
```

Задача 1 - решение

```
        for(int i=0;i<P;i++) {
            String line = sc.nextLine();
            String parts[] = line.split( regex: " ");
            trees[mapping.get(parts[0])] = 0;
            trees[mapping.get(parts[1])] = 0;
        }

        sc.close();

        List<Edge> resultEdges = cityNetwork.adaptedKruskal(trees);

        float suma = 0;

        for(Edge e : resultEdges) {
            suma+=e.getWeight();
        }

        System.out.println(resultEdges.size() + " " + suma);

        for(Edge e : resultEdges) {
            System.out.println(cityNetwork.getVertex(e.getFrom()) + " " + cityNetwork.getVertex(e.getTo()));
        }
    }
}
```

Задача 2 - Водовод

Дадена е водоводна мрежа во даден град. Во мрежата постои еден извор на вода и поголем број на крајни корисници. Постојат и неколку дистрибуциски јазли низ кои поминува водата и кои може да се поврзани преку повеќе патеки. Во самата мрежа можно е да има повеќе патеки низ кои водата стигнува до крајните корисници.

Мрежата е дадена преку граф од јазли (кои можат да бидат изворот, дистрибуциските јазли или крајните корисници). Јазлите се поврзани со цевки. Притоа кај секоја од цевките ја има дадено и нејзина карактеристика- пад на притисок кој се случува доколку водата помине низ неа.

За сите крајни корисници дадена е и надморската висина на која се наоѓаат. Корисниците добиваат вода доколку разликата од почетниот притисок и вкупната сума на падот на притисокот по патот до нив е позитивен број. Пример: почетен притисок е 100, водата поминува низ цевки со пад на притисок од 30, 50 и 10. Корисникот ќе добие вода само доколку се наоѓа на надморска висина помала од 10 (висина <(100 – (30+50+10))).

Во иницијалната конфигурација на мрежата, сите корисници добиваат вода. Но, во мрежата се случуваат неколку дефекти, кои го зголемуваат падот на притисокот кај дадени цевки за неколку пати.

За дадена иницијална мрежа, како и локација и сериозност на дефекти, ваша задача е да ги испечатите крајните корисници кои нема да добијат вода во таква конфигурација на системот.

Задача 2 - Водовод

Влез: На влез прво е даден бројот на јазли во мрежата N Во наредните N линии се даени бројот на јазол, па неговата надморска висина. Изворот е секогаш прв со реден број 0. Бројот до изворот е притисокот во мрежата на ниво на изворот. Дистрибуциските јазли се на надморска висина 0 На крај следат крајните корисници, до кои после редниот број е дадена и нивната надморска висина Потоа следи бројот M, кој го означува бројот на цевки во мрежата. Во наредните M редови се дадени три броеви: почетно теме на цевката, крајно теме на цевката и пад на притисок на таа цевка На крај следи бројот P, кој го означува бројот на дефекти Во наредните P редици се дадени цевките кај кои има дефекти, на следен начин: почетно теме на цевката, крајно теме на цевката и уште еден природен број, кој кажува колку пати сега е зголемен падот на притисокот од иницијалниот.

Излез: На излез се печатат броевите на јазлите на крајните корисници кај кои нема да има вода

Задача 2 - решение

```
import java.util.Map;
import java.util.Scanner;

public class Plumbing {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        int n = sc.nextInt();

        AdjacencyListGraph<Integer> waterNetwork = new AdjacencyListGraph<>();
        int values[] = new int[n];

        for(int i=0;i<n;i++) {
            int ind = sc.nextInt();
            int val = sc.nextInt();

            waterNetwork.addVertex(ind);
            values[ind] = val;
        }

        int m = sc.nextInt();

        for(int i=0;i<m;i++) {
            waterNetwork.addEdge(sc.nextInt(), sc.nextInt(), sc.nextInt());
        }

        int p = sc.nextInt();

        for(int i=0;i<p;i++) {
            int startVertex = sc.nextInt();
            int endVertex = sc.nextInt();
            int increaseFactor = sc.nextInt();
            waterNetwork.addEdge(startVertex, endVertex,  weight: waterNetwork.getNeighbors(startVertex).get(endVertex)*increaseFactor);
        }

        sc.close();
```

Задача 2 - решение

```
        Map<Integer, Integer> shortestPaths = waterNetwork.shortestPath( startVertex: 0);

        for(int i=1;i<n;i++) {
            if(values[i] != 0) {
                if(values[0] - (shortestPaths.get(i) + values[i]) <= 0) {
                    System.out.print(i + " ");
                }
            }
        }

    }
}
```