LAB WORK 09. GRAPH ALGORITHMS IN COMPUTER NETWORKS.

1. LAB TARGET.

- The study of graph algorithms used in computer networks:
  - Graph representation and traversal methods (DFS and BFS);
  - STP (Spanning Tree Protocol) to eliminate network loops on switches;
  - MinST Prima & MinST Kruskal – finding minimal spanning tree in PIM (multicasting);
  - SPF Tree Dijkstra – finding the Shortest Path Tree in OSPF;
  - Max Flow – maximal flow search between two nodes of network in MPLS Traffic Engineering.
- Gain experience with graph mining software GraphOnline and GraphTea.

2. LAB ASSIGNMENT.

2.1. CREATE YOUR VARIANT OF THE TASKS.

1. Network topology variant = (N MOD 10)+1 , where N is the alphabetical number of the first letter of your name. Variants are listed at the end of the guidelines in section 5.

2. Vertices Names in the DiGraph (directed graph) are replaced by the first letters in your last name, first name and patronymic, written in the English alphabet with the rejection of repeated letters and non-alphabetic characters.

3. Edges Weight = to the modulus of the difference between the numbers of letters of adjacent vertices.

2.2. FORMULATION OF THE PROBLEM.

We have:

1. A network topology of several routers is set (by option).
2. A source s-router (start) and a target t-router (target) are specified.
3. GraphOnline or GraphTea software of your choice (hereinafter referred to as GraphApp).

Necessary:

1. Form a variant of an individual task.
2. Build an **Adjacency Matrix** (weight matrix) for your individual DiGraph.
3. Using the **GraphApp** program, graphically represent a DiGraph indicating: vertices, edges and weights in accordance with your variant. Save the Internet link for the resulting DiGraph.
4. Using **GraphApp**, build a **MinST** and report a graph with a selected MinST and its Cost.
5. Using the **Prima** and **Kruskal algorithms** to manually construct MinST, compare the results with item 4 and comment on the differences.
6. Using the **Dijkstra algorithm**, manually build an **SPF Tree** (Shortest Path First Tree) from the s-vertex of the DiGraph. Include in the report a table of the step-by-step operation of the algorithm and a list of the shortest paths from the s-vertex to the other vertices.
7. Using **GraphApp**, build an **SPF Tree** from an s-vertex to a t-vertex using the Dijkstra algorithm. Include in the report a DiGraph with highlighted SPF search paths. Compare results with 6 and comment differences.
8. Using the **GraphApp** program, determine the **MaxFlow** from the s-vertex to the t-vertex and report a DiGraph indicating the edge flow metrics.

2.3. Report and Graduation.

The Report is provided electronically. The report must include:

1. The process of forming a variant of the task and the original DiGraph.
2. The Adjacency Matrix for the resulting DiGraph.
3. Graphical representation of a DiGraph with marked vertices, edges and weights. Provide a link to the resulting graph on the Internet.
4. Cost and image of MinST received in GraphApp.
5. Step-by-step work for the Prima algorithm (Vertex Sequence and MinST cost) and for the Kruskal algorithm (Edge Sequence and MinST cost). Comparison of the 4 and 5 results.
6. Table of step-by-step work of the Dijkstra algorithm and a list of SPs from the s-vertex (3 points of grade).
7. SPF Tree image from s to t obtained by the Dijkstra algorithm in GraphApp. Comparison 6 and 7 results.
8. The value of the maximum MaxFlow flow from the s-router to the t-router and a DiGraph indicating the flow metrics through the communication lines, obtained by the MaxFlow algorithm in GraphApp.

Grade.

The maximum score is 10 points. Each item correctly completed is worth 1 point, and item 6 is worth 3 points.

An example of a report is provided in Report Blank Form.

3. BRIEF THEORETICAL INFORMATION.

3.1. ALGORITHMS ON GRAPHS.

Graph models are widely used in various applied areas, including computer networks:

- traversal of the graph in DFS (в глубину) and BFS (в ширину) (needed in many other algorithms);
- construction spanning tree ST (in the STP protocol for Ethernet switches);
- performing topological sorting of a directed acyclic graph (in various application tasks and for SP);
- construction of the minimum spanning tree MinST (in the PIM routing group protocol);
- finding the shortest paths between SP vertices (in the OSPF routing protocol);
- finding the maximum flow MaxFlow (for Traffic Engineering in MPLS)

Программное обеспечение для работы с графами и исследования алгоритмов на графах:

- Graph Online https://graphonline.ru/en/home?graph=weightedGraph
- Graph Tea http://graphtheorysoftware.com/
- Graph Editor https://csacademy.com/app/graph_editor/
- Graph Teory https://csacademy.com/lessons/

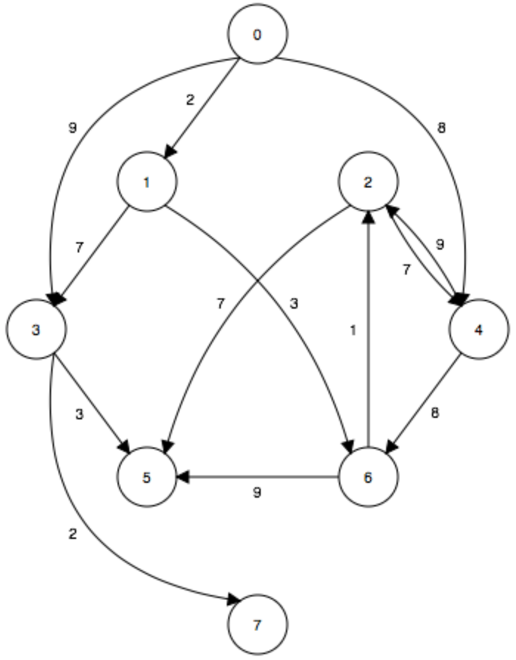
3.2. Graph presentation methods.

There are several ways to define graphs. To solve a specific problem, one or another method is chosen, depending on the convenience and efficiency of its application.

A demonstration and explanation of the various ways of describing graphs can be found at https://youtu.be/c8P9kB1eun4.

3.2.1. Graphical representation.

This method is best for figurative representation of the topology of connections for a human and bad for representation in a computer →



The graph G is a collection of sets of vertices V (or nodes) and edges E:

- G=(V,E).
- V={0,1,2,3,4,5,6,7}
- E={(0,1,2)(0,3,9)(0,4,8)(1,3,7)(1,6,3)(2,4,7)(2,5,7)(3,5,3)(3,7,2)(4,2,9)(4,6,8)(6,2,1)(6,5,9)}

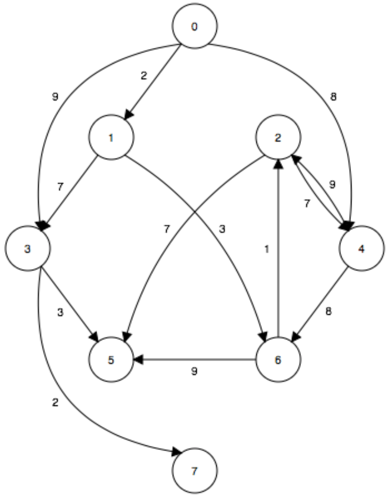
Graph types:

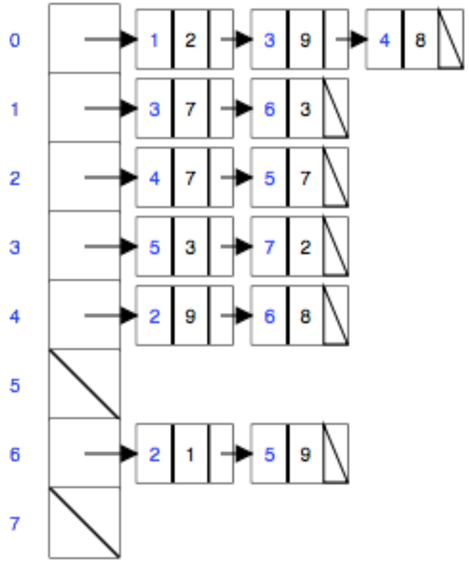
- connected, disconnected, null (V&E=0 or E=0 use in Mathematical induction);
- directed, undirected, mixed;
- weighted, unweighted;
- cyclic, acyclic.

CDWAG – connected, directed, weighted, acyclic graph.

3.2.2. Adjacency List (Списки смежности).

Each i-th list contains the numbers of vertices adjacent to the i-th vertex.





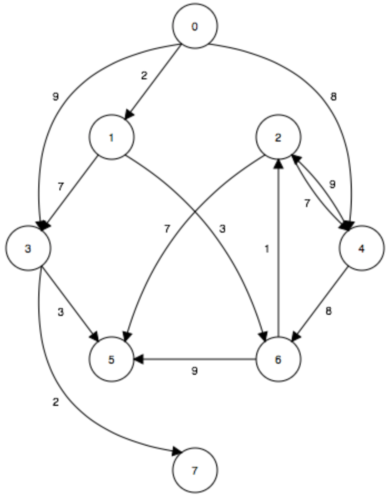
Adjacency lists are convenient for entering into a computer, save memory, but, in the case of a weighted graph, use is difficult, since you need to additionally store the weight of the edges.

For unweighted directed graph
0: 134
1: 36
2: 45
3: 57
4: 26
5: null
6: 25
7: null

3.2.3. Incidence Matrix (Матрица инцидентности).

Matrix A of size n*m (n is the number of vertices, m is the number of edges), the element aij of the matrix is equal to 1 if the i-th vertex is incident to the j-th edge and 0 otherwise.

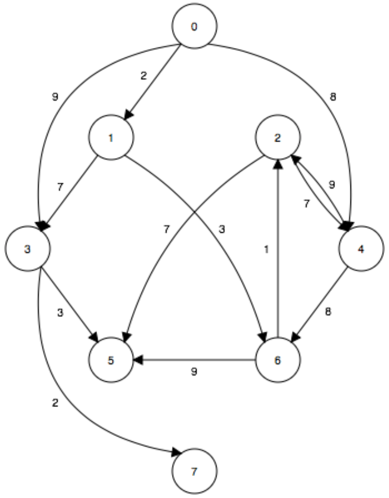
The incidence matrix does not carry direct information about the edges, and it is necessary to additionally set the weight vector of the edges and additionally set the direction of the connection (for example, as positive for the initial and as negative for the end node of the edge).



Edge j / Vertice i	e_0	e_1	e_2	e_3	e_4	e_5	e_6	e_7	e_8	e_9	e_{10}	e_{11}	e_{12}
0	1	1	1										
1	1			1	1								
2						1	1			1		1	
3		1		1				1	1				
4			1			1				1	1		
5							1	1					1
6					1						1	1	1
7									1				
Weight W_j	2	9	8	7	3	7	7	3	2	-9	8	-1	-9

3.2.4. Adjacency and Weight Matrices (Матрицы смежности и весов).

Matrix A of size n*n whose element Aij is equal to 1 if the i-th vertex is adjacent to the j-th one, and 0 otherwise. It is possible to store compactly as a bitmap.



	0	1	2	3	4	5	6	7
0		2		9	8			
1				7			3	
2					7	7		
3						3		2
4			9				8	
5								
6			1			9		
7								

The adjacency matrix of an undirected graph is symmetric about the main diagonal.

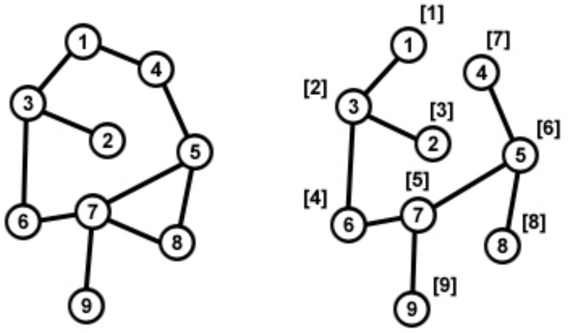
For a weighted graph, the value of the weight function is used and such a matrix is called the weight matrix.

Since the adjacency matrix has an estimate of size O(nˆ2), algorithms that use this way of representing graphs have no less complexity than O(nˆ2) (O -оценка сложности).

3.3. GRAPH TRAVERSAL.

The main graph traversal algorithms are depth traversal (DFS) or breadth traversal (BFS). Depth-first traversal sequentially scans the vertices of the graph by **building a chain** from the current vertex. Breadth-first searches the **vertices adjacent to the current** one first.

3.3.1. DFS - Depth First Search (поиск в глубину).



Detail.

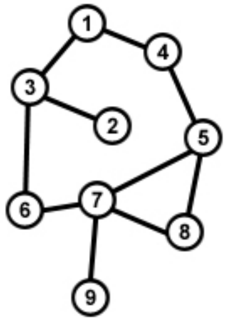
1. The search starts from some (any) vertex v.
2. We consider a new (from those not considered earlier) vertex u, adjacent to v, and mark it as scanned.
3. Step 2 is repeated with the vertex u.
4. If at the next step the vertex w was scanned and there are no vertices adjacent to w and not considered earlier, then we return from the vertex w to the before scanned vertex.
5. If all the vertices have been scanned or we have returned to the initial vertex v, but there are no new vertices adjacent to it, then the traversal process is over.

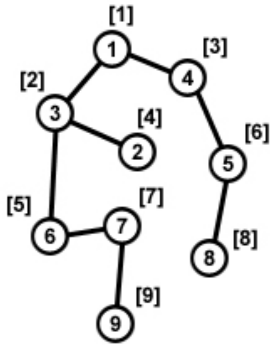
Complexity DFS=O(V+E). DFS is convenient for **recursive** or **stack** implementation. A demonstration and explanation of the DFS algorithm can be viewed at links https://csacademy.com/lesson/depth_first_search and https://youtu.be/ymlzHmRN4To. Algorithmic complexity (big O) is a measure of how long an algorithm would take to complete given some size input (n).

Example. A graph is given (see figure). The search starts from vertex 1. The left figure shows the original graph, and the right figure near the vertices in parentheses indicates the order in which the graph vertices were viewed during the DFS.

3.3.2. BFS – Breadth First Search (поиск в ширину).

Detail.





- Let a graph G=(V,E) and some initial vertex v be given.
- The breadth-first search algorithm enumerates all other vertices reachable from v in order of increasing distance from v.
- During the search, a part is extracted from the graph, called a breadth-first search tree rooted at v.
- BFS complexity = O(V+E).
- To implement this principle, the **queue** data structure is usually used.
- A demonstration and explanation of the operation of the BFS algorithm can be viewed at links https://csacademy.com/lesson/breadth_first_search and https://youtu.be/ymlzHmRN4To.

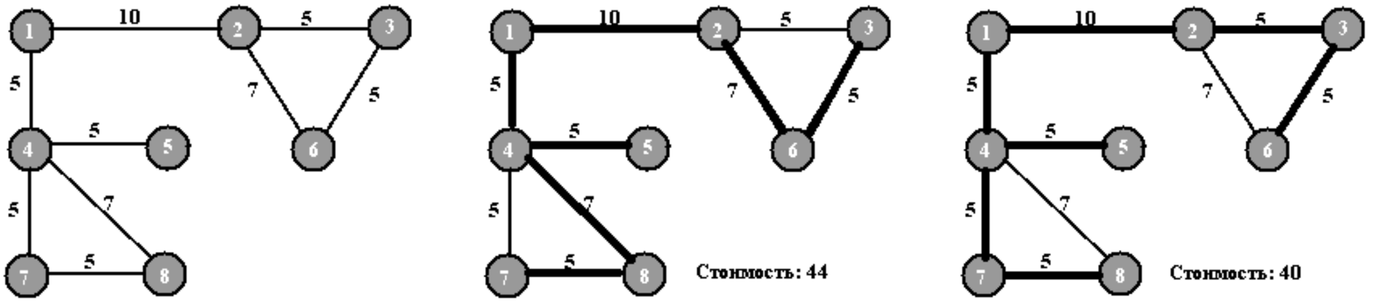
Example. A graph is given (see figure). The search starts from vertex 1. The left figure shows the original graph, and the right figure near the vertices in parentheses indicates the order in which the graph vertices were viewed during the BFS.

3.4. ST - Spanning Tree (остовое дерево).

A graph with a number assigned to each edge is called a weighted graph.

Definition of a Spanning Tree. For an arbitrary connected undirected weighted graph G=(V,E), a spanning tree (skeleton, carcass, frame) is a connected subgraph T=(V,E*), where E* ⊆ E, T containing all vertices V of the graph G, and not having cycles. The number of edges E* in the skeleton T is always one less than the number of vertices V of the graph G (E*=V-1).

A graph can have several skeletons from V to V^(V-2) depending on the completeness of graph connections (от полноты связей). For example, different skeletons (frameworks) can be built by starting a **depth-first search from different vertices** of the graph. The figure on the left shows a graph and on the right two of its frames.



The weight (or cost) of a skeleton is the sum of the weights of its edges. The cost spanning tree of 40 in the example is the minimal.

3.4. MinST - Minimal Spanning Tree.

For practical problems (transport, water pipes, networks, electrical, electronic circuit) it is often required to find a **frame of minimal weight**.

3.4.1. MinST Kruskal's algorithm (1956).

Joseph Bernard Kruskal, 1928-2010 – american mathematician.

Given: connected, weighted, undirected n-vertex graph G=(V,E).

Algorithm:

1. Start creating forest with null of Fo, containing only n vertices (without edges) of G.
2. Arrange the edges of the original graph G in non-decreasing order of their weights.
3. Starting from the first (minimal) edge, add the next edge to the forest Fi={T*} if its addition does not create a cycle in T*.
4. Repeat step 3 until the number of edges in F is equal to (n-1).

Result: From the forest we get one minST tree T=(V,E*), where $E^* \subseteq E$ (E* is subset E).

Time complexity of the algorithm: from O(E+V) - if E is sorted to O(ElogE) - sorting E will take O(ElogE).

Kruskal's algorithm is preferable when the graph is sparse (разреженный) (there are few edges in the graph): E<=(Vˆ2)/2 and the edges are already sorted or can be sorted in linear time.

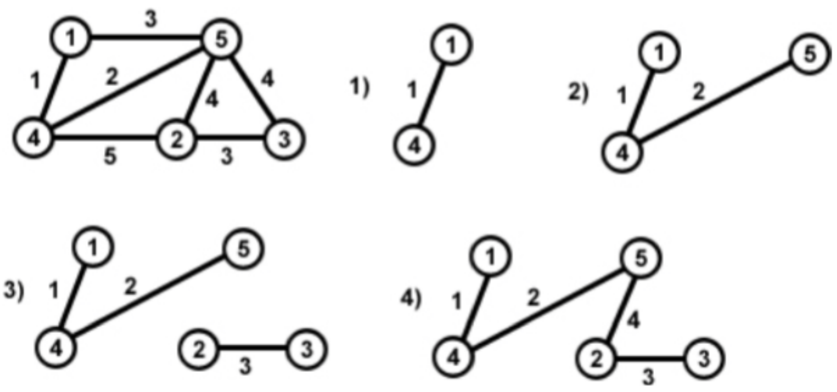
For G1(E=10,V=10)=>O1=20. For G2(E=Vˆ2,V=10)=>O2=100*log100=1000

Example 1.

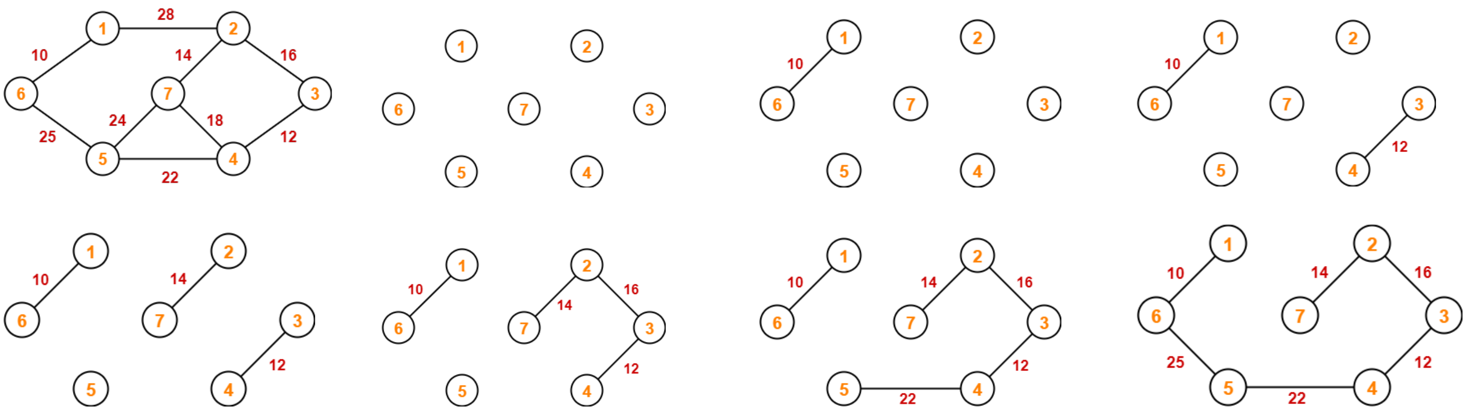
Graph and the process of constructing a MinST using the Kruskal algorithm →.

Edge Sequence: (1-4[1])(4-5[2])(2-3[3])(2-5[4]).

Cost of MinST = 10.



Example 2.



Edge Sequence = (1-6[10])(3-4[12])(2-7[14])(2-3[16])(4-5[22])(5-6[25])
Cost of MinST = 99 units.

3.4.2. MinST Prim's algorithm (1957).

Robert Clay Prim, born 1921 – american mathematician.

Given: connected, weighted, undirected n-vertex graph G=(V,E).

Algorithm: At each step, a finite tree T is completed (but not a forest of trees F as in Kruskal's method).

The tree starts from any root vertex r and grows until it spans (охватит) all vertices in V.

At each step, a vertex is added to the tree with **the least weight edge** among all the edges connecting the already considered tree vertices with vertices from the rest of the graph, if adding a vertex does not lead to a cycle in T*

Result: Minimal Spanning Tree T=(V,E*), where E* ⊆ E.

Time complexity of the algorithm: from O(E+VlogV) for Fibonacci heap (куча) to O(ElogV) for binary heap.

Prim's algorithm is preferable when the graph is dense (плотный): the graph has a large number of edges, for example (Vˆ2/2)<E<=(Vˆ2).

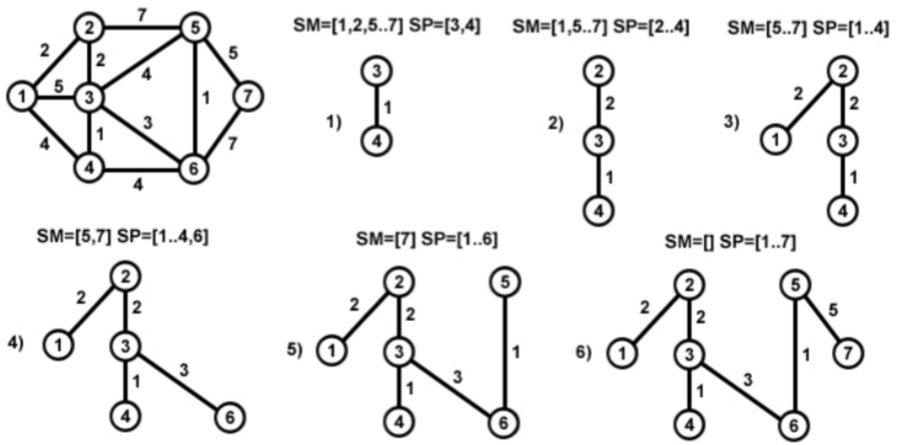
Example 1.

Graph and the process of constructing a minimum spanning tree (MinST) using Prim's algorithm →

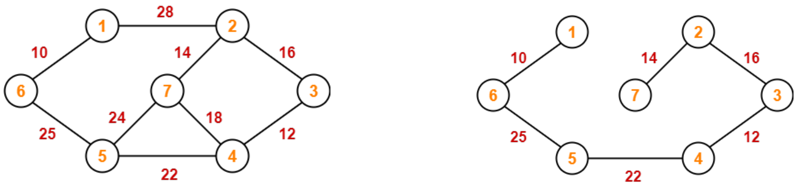
SM is the remaining set of vertices in the graph, and SP is the set of vertices included in the constructed MinST tree.

Vertex Sequence = 3,4,2,1,6,5,7.

Cost of MinST = 14.



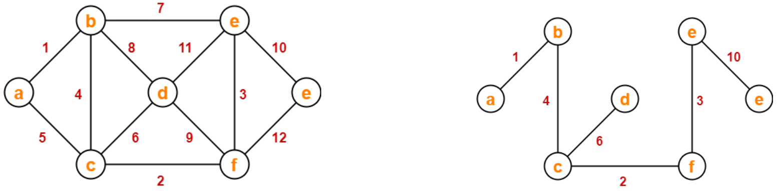
Example 2.



Vertex Sequence = 1,6,5,4,3,2,7.

Cost of MinST = 99 units.

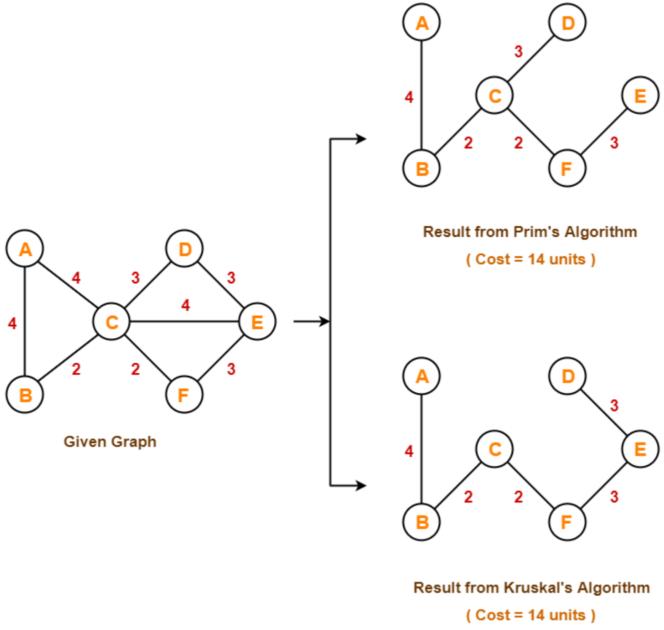
Пример 3.



Vertex Sequence = a,b,c,f,e,d,g.

Cost of MinST = 99 units.

3.4.3. Prim’s Algorithm vs Kruskal’s Algorithm.



3.5. Search for the Shortest Path between two graph nodes.

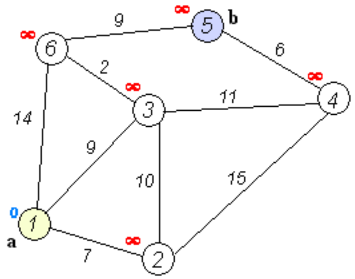
The problem of finding the shortest path from a given vertex of a graph to another given vertex has no simpler solution than the **simultaneous search for all shortest paths** (**одновременный поиск всех наикратчайших путей)** from some given vertex to every other vertex, so the SP problem is reduced to the SP Tree problem.

3.5.1. SPT Dijkstra's algorithm (1959).

Edsger Wybe Dijkstra, 1930-2002 – holland mathematician.

Dijkstra's algorithm is designed to find the shortest paths from one vertex of a weighted directed graph to all other vertices.

For the algorithm to work correctly, it is necessary that there are no edges with negative weights in the graph.



The algorithm uses the fact that any part of the shortest path is itself the shortest path between the corresponding vertices. At each step, the algorithm "visits" one vertex and tries to decrease the path cost labels. (https://net.academy.lv/labwork/net_LW-09EN_Dijkstra.gif)

A demonstration and explanation of the SP Tree Dijkstra algorithm can be viewed at https://youtu.be/pVfj6mxhdMw.

Given: connected, weighted, cyclic n-vertex graph G=(V,E).

Initialization.

All vertices of graph V are marked as **unvisited**. The **cost label** of the initial vertex to itself is assumed to be 0, the cost labels to the remaining vertices are infinity (∞), this reflects that the distances (sumE') to the other vertices of V are not yet known.

Algorithm.

1. The algorithm terminates when all vertices have been visited.
2. From the vertices not yet visited, the vertex **u with the minimal cost label** is selected. If the labels are equal, then choose the oldest one.
3. The vertices connected to the vertex u by edges are called the **neighbors** of this vertex. For each neighbor, a **new cost label** of the path is considered, if the resulting length is less than the current neighbor label, then replace the label with the new value.
4. Having considered all the neighbors, mark the vertex u **as visited** and repeat the steps of the algorithm.

The result is a list of shortest paths connecting the given vertex to each vertex.

Dijkstra's algorithm time complexity is **O(V*logV+E).**

3.5.2. SPT Floyd algorithm (1962).

Floyd's algorithm is designed to find the **SPTs for all vertices** of a weighted graph simultaneously, which takes more time than Dijkstra's algorithm. Floyd's algorithm time complexity is **O(V^3).**

3.5.3. SPT Bellman-Ford algorithm (1956).

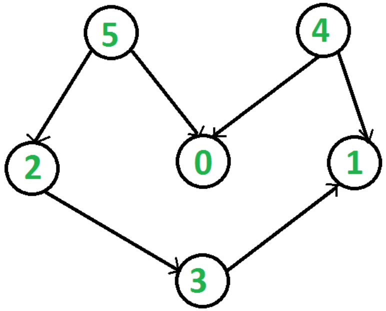
The Bellman-Ford algorithm is simpler than Dijkstra's and works well for distributed systems, but its time complexity is **O=(V*E),** which **is larger than Dijkstra's**. Dijkstra does not work for graphs with negative edge weights, while Bellman-Ford does work.

Negative weights occur in various applications of graphs. For example, instead of increasing the cost of a path, we **can benefit** (выгоду) by following a particular path.

3.5.4. SPT for CDWAG (Connected Directed Weighted Acyclic Graph).

Для CDWAG есть алгоритм со сложностью порядка O(V+E), что меньше или равно чем у Дейкстры.

3.5.4.1. Топологическая сортировка для DWAG.



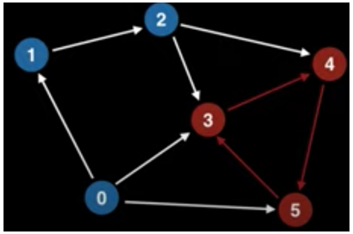
Топологическая сортировка - это линейное упорядочивание вершин таким образом, что для каждого направленного ребра uv вершина u находится перед v в порядке. Топологическая сортировка невозможна, если граф не является DWAG.

Для графа может быть более одной топологической сортировки. Первой вершиной в топологической сортировке всегда является вершина с входящей степенью, равной 0 (вершина без входящих ребер).

Например, топологическая сортировка графа показанного на рисунке - «5 4 2 3 1 0» или «4 5 2 3 1 0».

Алгоритм **топологической сортировки** - это просто DFS (Deep First Search) с дополнительным стеком. Таким образом, временная сложность такая же, как у DFS = O(V+E).

Graph contain a cycle cannot have a valid ordering →



Использование: Топологическая сортировка используется для планирования заданий по заданным зависимостям между заданиями. В информатике приложения этого типа возникают при планировании инструкций, упорядочивании вычислений ячеек формулы при пересчете значений формул в электронных таблицах, логическом синтезе, определении порядка задач компиляции, выполняемых в файлах make, сериализации данных и разрешении символьных зависимостей в компоновщиках.
Демонстрацию и объяснение работы алгоритма Topological Sort of DWAG можно посмотреть на https://youtu.be/eL-KzMXSXXI.

3.5.4.2. SPT for DWAG.

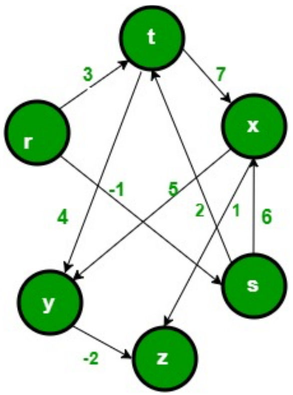
Алгоритм.

1. топологически отсортировать DWAG;
2. устанавить расстояние до источника равным 0 и бесконечность до всех остальных вершин;
3. для каждой вершины из списка проходим всех ее соседей и проверяем кратчайший путь;

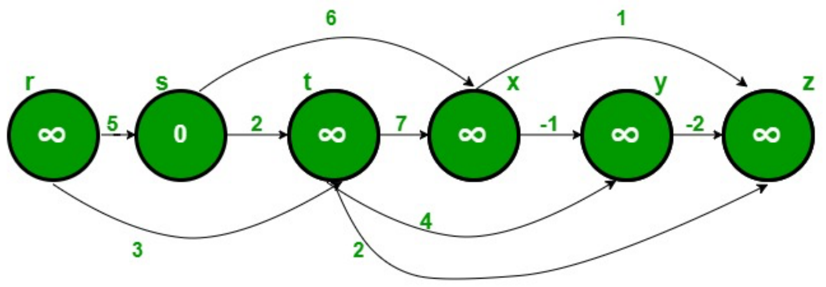
Алгоритм очень похож на алгоритм Дейкстры с той разницей, что тогда мы использовали приоритетную очередь, а на этот раз мы использовали список из топологической сортировки.

Пример.

Ниже, рисунок (b) - это линейное представление рисунка (a).

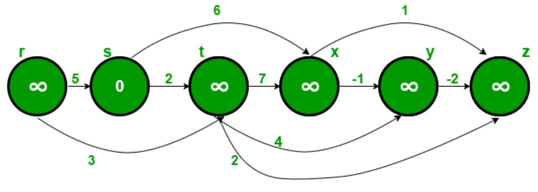


(a)

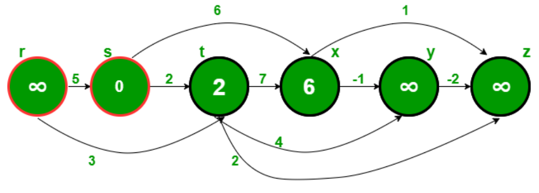


(b)

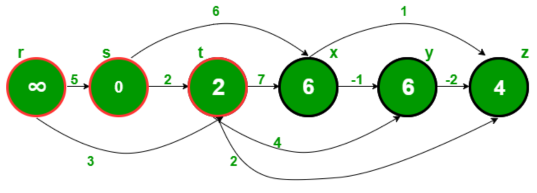
Когда у нас есть топологический порядок (или линейное представление), мы последовательно обрабатываем все вершины в топологическом порядке. Для каждой обрабатываемой вершины мы обновляем расстояния до соседней вершины, используя расстояние до текущей вершины.



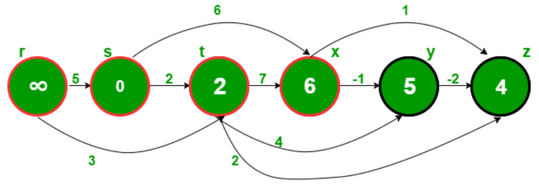
(c)



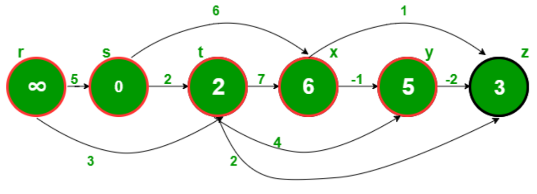
(d)



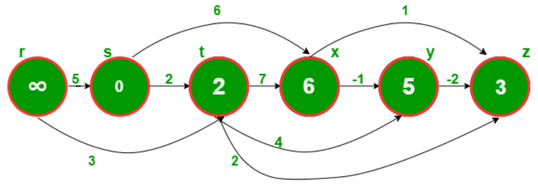
(e)



(f)



(g)



(h)

Выход: Кратчайшие расстояния от источника 1. 0 2 6 5 3

3.5.5. Сложность алгоритмов SPT.

3.5.5.1. Сложность алгоритма Дейкстры зависит от способа нахождения ближайшей вершины, от способа хранения множества не посещённых вершин и способа обновления меток. Обозначим через V количество вершин, а через E — количество рёбер в графе. Размещать в памяти требуется порядка O(V^2) данных, основная сложность – это количество сравнений и релаксаций.

1. В простейшем случае при реализации на матрице смежности, когда для поиска вершины с минимальным количеством пробных путей просматривается всё множество вершин, время работы алгоритма имеет порядок **O(V^2+E*V)**. Основной цикл выполняется порядка V раз, в каждом из них на нахождение минимума тратится порядка V операций, плюс количество релаксаций (смен меток), которое не превосходит количества ребер E в G.
2. Для разрежённых графов (то есть таких, для которых E много меньше V^2) не посещённые вершины можно хранить в двоичной куче (heap) H, а в качестве ключа использовать значения d[i], тогда время извлечения вершины из H станет logV, при том, что время модификации d[i] возрастет до logV. Так как цикл выполняется порядка V раз, а количество релаксаций не больше E, скорость работы такой реализации **O(V*logV + E*logV)**.
3. Если для хранения не посещённых вершин использовать фибоначчиеву кучу, для которой удаление происходит в среднем за O(logV), а уменьшение значения в среднем за O(1), то время работы алгоритма составит **O(V*logV+E)**.

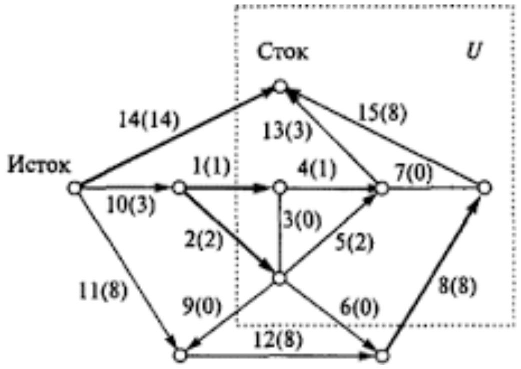
3.5.5.2. Сложность алгоритма Флойда - требует **O(V^3)** времени для работы.

3.5.5.3. Сложность алгоритма Беллмана-Форда - требует **O(V*E)** времени для работы.

3.5.5.4. Сложность SPT for DWAG. Сложность топологической сортировки по времени составляет O(V+E). После определения топологического порядка алгоритм обрабатывает все вершины и для каждой вершины выполняет цикл для всех смежных вершин. Общее количество смежных вершин в графе равно O(E). Таким образом, внутренний цикл выполняется O(V+E) раз. Следовательно, общая временная сложность этого алгоритма составляет **O(V+E)**. Проблема только в том, что мы должны быть уверены, что на графе нет циклов.

Вывод. Так как на практике обычно V<<E и возможны циклы, то алгоритм Дейкстры имеет лучшую или сравнимую оценку времени работы по сравнению с остальными алгоритмами, поэтому в OSPF кратчайший путь ищется распределённо в каждой вершине с применением алгоритма Дейкстры.

3.6. Maximal Flow finding.



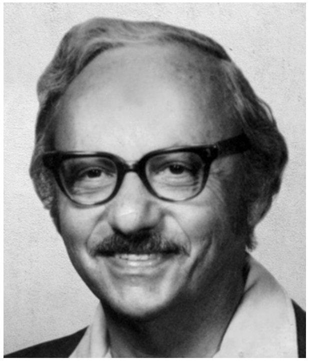
The Maximal Flow Problem (MaxFlow) in the network is to find such a flow through the transport network that the sum of flows from the source, or, which is the same, the sum of flows to the sink, is maximum.

3.6.1. MinCut algorithm of MaxFlow.

1. Cut off the source vertex from the original graph G, get the area U, calculate the total input flow F* through the section U (the weight of the input edges).
2. Sequentially cut off the next vertex and re-calculate the potential flow F*
3. Stop if only the stock top remains in U.
4. The minimum of the found sum F* will be the value of Max Flow.

Example. For the graph in the figure, we get MinCut=MaxFlow=25:
14+10+11=35; 14+10+0+12=27; 14+1+2+0+12=29; 14+1+2+0+0+8=25; 14+1+3+5+8=31; 14+4+5+8=31; 14+13+7+8=42; 14+13+15=42.

3.6.2. Maximal Flow Problem solution history.

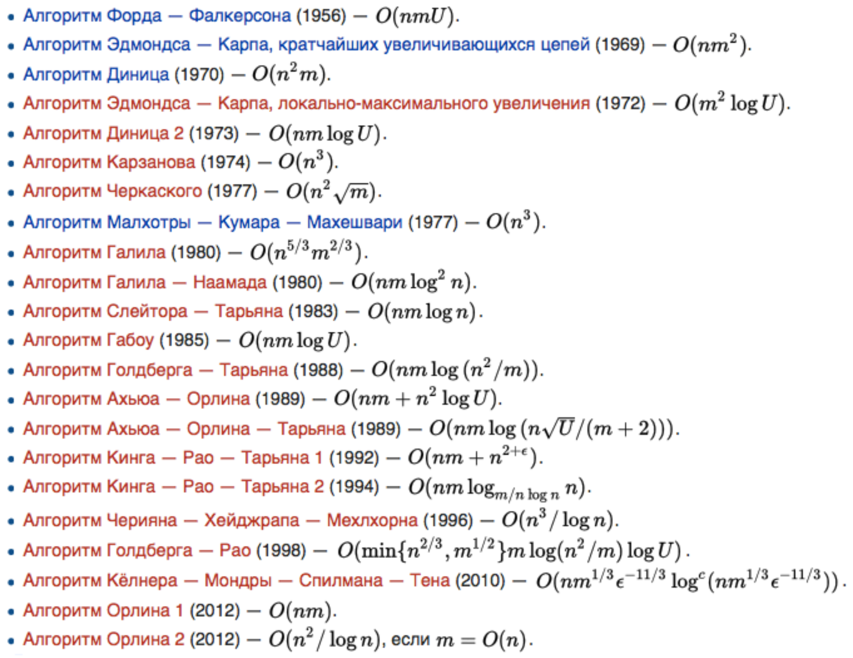


- In 1951, George Bernard Dantzig (1914-2005) first formulated the problem in general terms. Danzig is an american mathematician, known as the developer of the algorithm used in solving problems using the simplex method. Considered the founder of linear programming, along with Leonid Kantorovich and von Neumann.
- In 1956, Lester Ford and Delbert Ray Fulkerson first built an algorithm specifically designed to solve this problem. Their algorithm is called the Ford-Fulkerson algorithm.
- In the future, the solution of the problem was improved many times.
- In 2010 Jonathan Kelner, Aleksander Mądry, Daniel Spielman and Shang-Hua Teng found another algorithm improvement for the first time in 12 years.

3.6.3. Comparison of the efficiency of algorithms for finding the MaxFlow.

https://en.wikipedia.org/wiki/Maximum_flow_problem

- n - number of vertices,
- m - number of edges,
- U - the largest value of the maximal network throughput.



- Алгоритм Форда — Фалкерсона (1956) — $O(nmU)$.
- Алгоритм Эдмондса — Карпа, кратчайших увеличивающихся цепей (1969) — $O(nm^2)$.
- Алгоритм Диница (1970) — $O(n^2m)$.
- Алгоритм Эдмондса — Карпа, локально-максимального увеличения (1972) — $O(m^2 \log U)$.
- Алгоритм Диница 2 (1973) — $O(nm \log U)$.
- Алгоритм Карзанова (1974) — $O(n^3)$.
- Алгоритм Черкаского (1977) — $O(n^2 \sqrt{m})$.
- Алгоритм Малхотры — Кумара — Махешвари (1977) — $O(n^3)$.
- Алгоритм Галила (1980) — $O(n^{5/3}m^{2/3})$.
- Алгоритм Галила — Наамада (1980) — $O(nm \log^2 n)$.
- Алгоритм Слейтора — Тарьяна (1983) — $O(nm \log n)$.
- Алгоритм Габоу (1985) — $O(nm \log U)$.
- Алгоритм Голдберга — Тарьяна (1988) — $O(nm \log (n^2/m))$.
- Алгоритм Ахьюа — Орлина (1989) — $O(nm + n^2 \log U)$.
- Алгоритм Ахьюа — Орлина — Тарьяна (1989) — $O(nm \log (n\sqrt{U}/(m+2)))$.
- Алгоритм Кинга — Рао — Тарьяна 1 (1992) — $O(nm + n^{2+\epsilon})$.
- Алгоритм Кинга — Рао — Тарьяна 2 (1994) — $O(nm \log_{m/n \log n} n)$.
- Алгоритм Черияна — Хейджрапа — Мехлхорна (1996) — $O(n^3 / \log n)$.
- Алгоритм Голдберга — Рао (1998) — $O(\min\{n^{2/3}, m^{1/2}\} m \log(n^2/m) \log U)$.
- Алгоритм Кёлнера — Мондры — Спилмана — Тена (2010) — $O(nm^{1/3}\epsilon^{-11/3} \log^c (nm^{1/3}\epsilon^{-11/3}))$.
- Алгоритм Орлина 1 (2012) — $O(nm)$.
- Алгоритм Орлина 2 (2012) — $O(n^2/\log n)$, если $m = O(n)$.

4. Example of a possible task in the SPF Tree building for exam.

4.1. Given.

- • The graph (network topology and link costs) is defined by an adjacency matrix.
- • Initial vertex s=A (or B,.., or J).

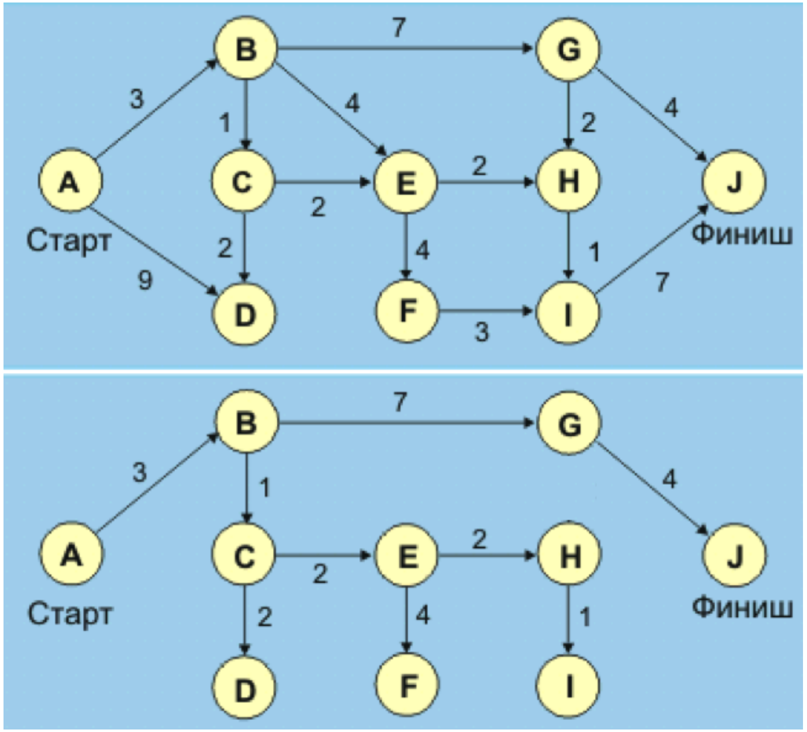
	A	B	C	D	E	F	G	H	I	J
A	0	3		9						
B	3	0	1		4		7			
C		1	0	2	2					
D	9		2	0						
E		4	2		0	2		4		
F					2	0			3	
G		7					0	2		4
H					4		2	0	1	
I						3		1	0	7
J							4		7	0

4.2. Exercise.

1. Build the initial network graph.
2. Construct an SPF Tree using the Dijkstra algorithm for node s (any vertex).
3. Draw a table showing the step-by-step work of the Dijkstra algorithm.
4. List the shortest paths from vertex s to the rest of the vertices.

4.3. Solution.

1. The original graph is shown in the figure above.
2. SP Tree from node s=A built by Dijkstra's algorithm is shown at the bottom of the figure.



3. Table of step-by-step work of the SPT Dijkstra algorithm.

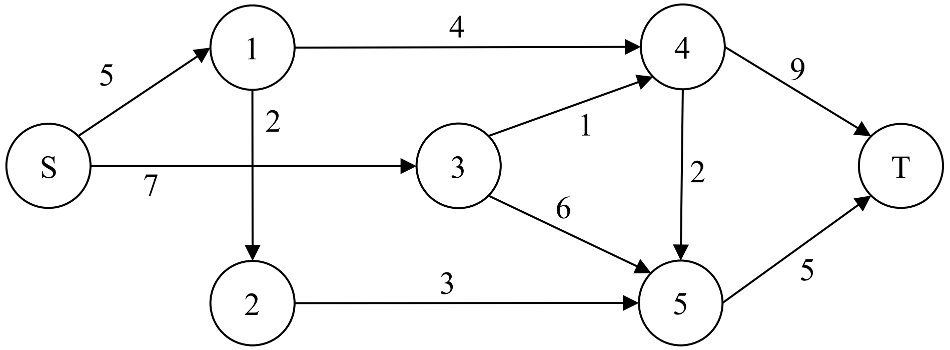
Step i	Set of visited vertices	Cost labels and the path of connection of vertex s=A with vertices								
		B	**C**	**D**	**E**	**F**	**G**	**H**	**I**	**J**
0	{A}	3	-	9	-	-	-	-	-	-
1	{A,B}	(3)	4	9	7	-	10	-	-	-
2	{A,B,C}	3	(4)	6	6	-	10	-	-	-
3	{A,B,C,D}	3	4	(6)	6	-	10	-	-	-
4	{A,B,C,D,E}	3	4	6	(6)	10	10	8	-	-
5	{A,B,C,D,E,H}	3	4	6	6	10	10	(8)	9	-
6	{A,B,C,D,E,H,I}	3	4	6	6	10	10	8	(9)	18
7	{A,B,C,D,E,H,I,G}	3	4	6	6	10	(10)	8	9	14
8	{A,B,C,D,E,H,I,G,F}	3	4	6	6	(10)	10	8	9	14
9	{A,B,C,D,E,H,I,G,F,J}	3	4	6	6	10	10	8	9	(14)

4. List of shortest paths.

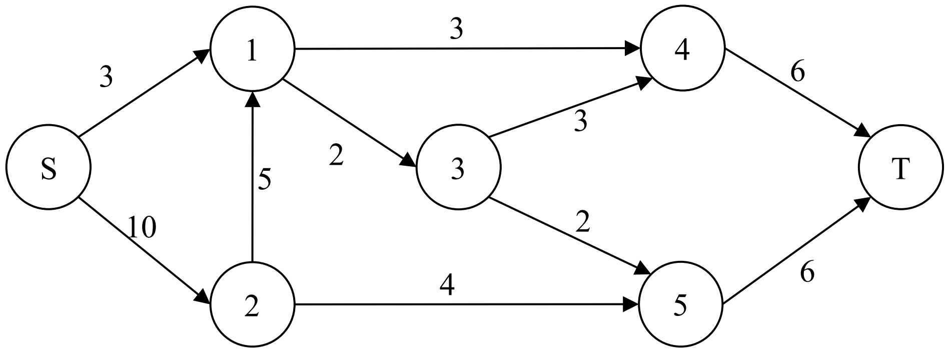
A→A	=A	=0
A→B	=AB	=3
A→C	=ABC	=4
A→D	=ABCD	=6
A→E	=ABCE	=6
A→H	=ABCEH	=8
A→I	=ABCEHI	=9
A→F	=ABCEF	=10
A→G	=ABG	=10
A→J	=ABGJ	=14

5. Network Topology Variants.

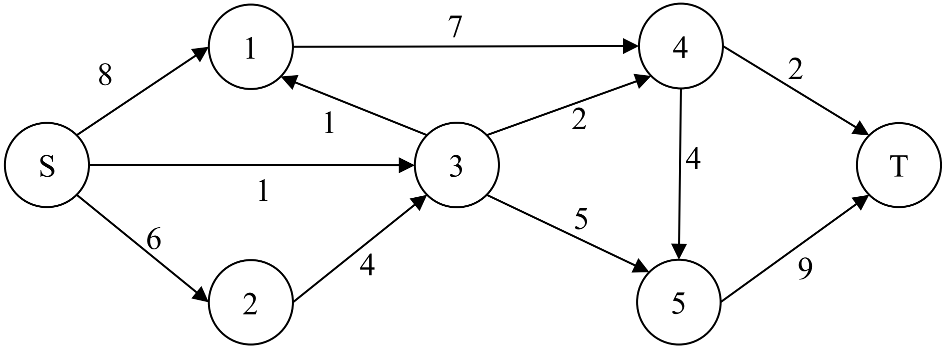
Variant 1.



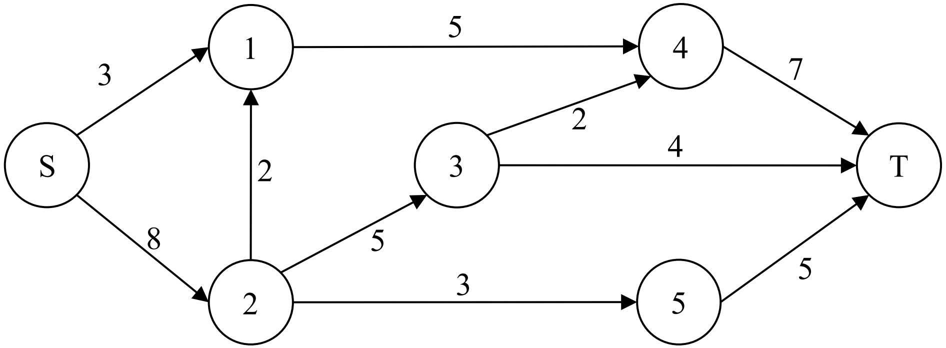
Variant 2.



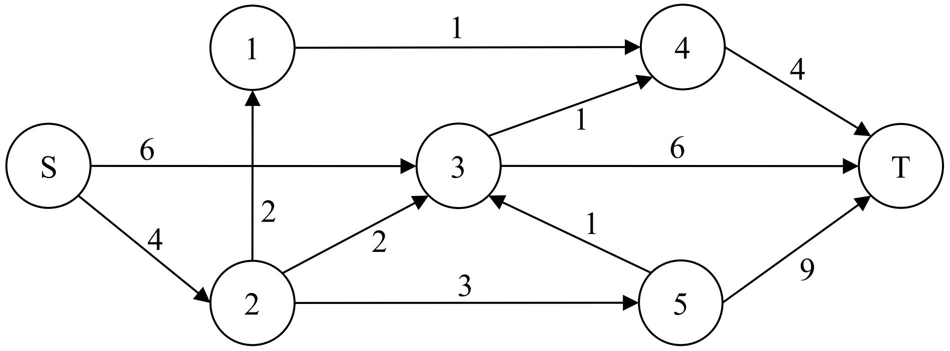
Variant 3.



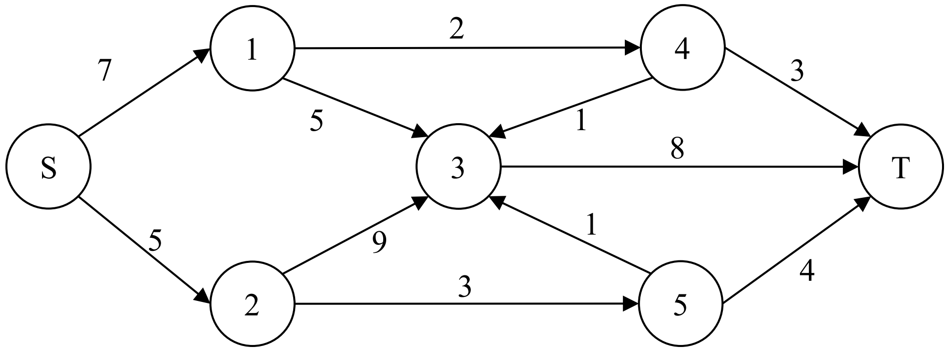
Variant 4.



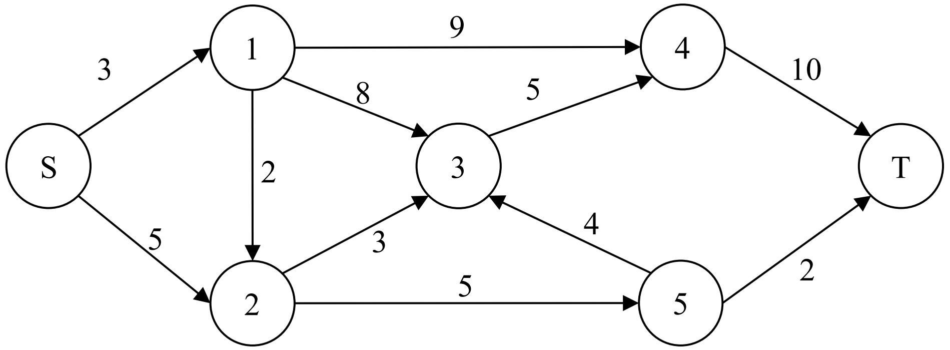
Variant 5.



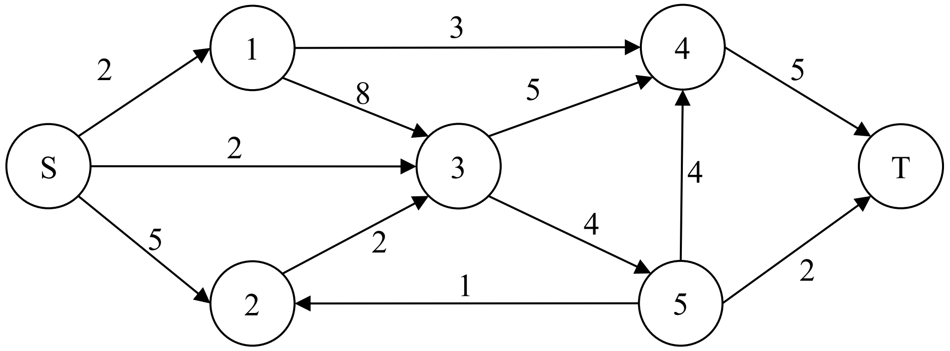
Variant 6.



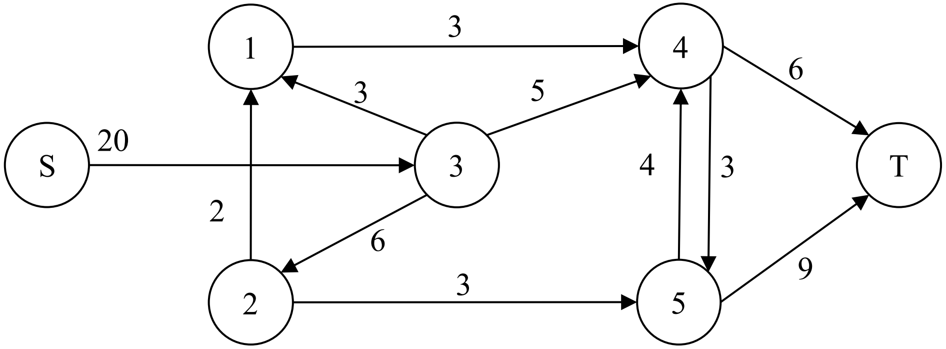
Variant 7.



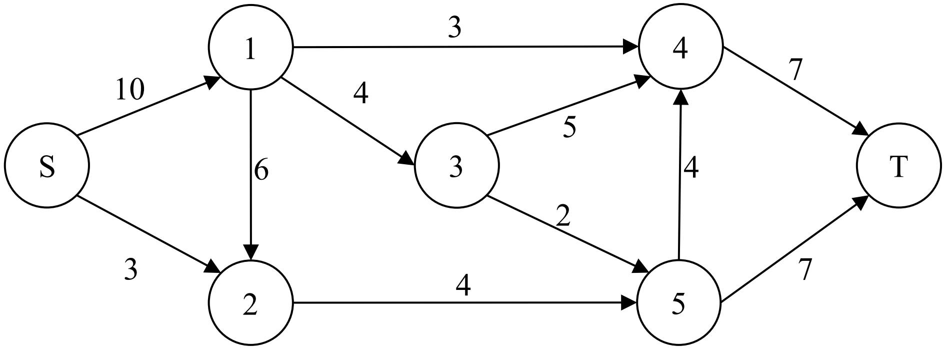
Variant 8.



Variant 9.



Variant 10.


S
1
2
3
4
5
T
10
3
3
4
6
5
2
4
4
7
7