Algebra and Discrete Mathematics
ADM

Bc. Xiaolu Hou, PhD.

FIIT, STU
xiaolu.hou @ stuba.sk

Course Outline

- Vectors and matrices
- System of linear equations
- Matrix inverse and determinants
- Vector spaces and matrix transformations
- Fundamental spaces and decompositions
- Eulerian tours
- Hamiltonian cycles
- Midterm
- Paths and spanning trees
- Trees and networks
- Matching

Recommended reading

- Saoub, K. R. (2017). A tour through graph theory. Chapman and Hall/CRC.
  - Sections 4.3, 4.4, 7.4, 7.5
  - Accessible online (free copy)
  - Alternative download link

Lecture outline

- Shortest networks
- Metric Traveling Salesman Problem
- Flow and capacity
- Rooted trees

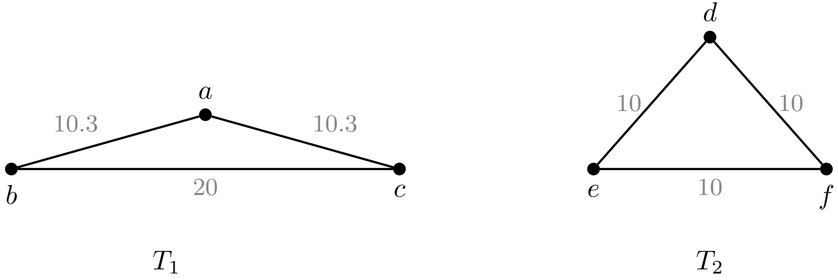
Trees and networks

- Shortest networks
- Metric Traveling Salesman Problem
- Flow and capacity
- Rooted trees

Shortest network

- Network: a connected graph
- Shortest network: least total weight

MST



- Edge lengths are drawn to scale since the angle created by the edges will play a large role in determining where to place shortcuts
- Minimum spanning tree: two edges of length 10.3 of T_1, any two edges of T_2
- Could we do better than a Minimum Spanning Tree?

Fermat point

- A similar question was posed by the 17th century French mathematician Pierre de Fermat in a letter to Evangelista Torricelli, an Italian physicist and mathematician
- In his letter, Fermat challenged Torricelli to find a point that minimizes the distance to each of the vertices in a triangle

Definition

A *Fermat point* for a triangle is the point p so that the total distance from p to the vertices of the triangle is minimized. Each of the three angles formed by these segments measures 120°.

Shortest network for a triangle

Theorem

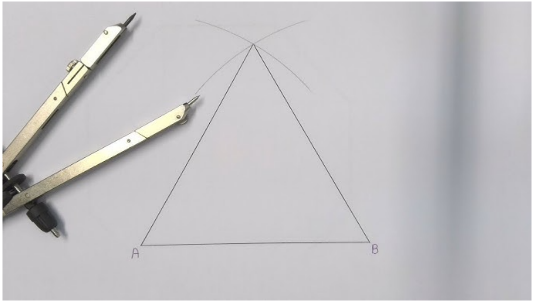
Given three points and a triangle T formed by these points, the Shortest Network connecting the three points will either be

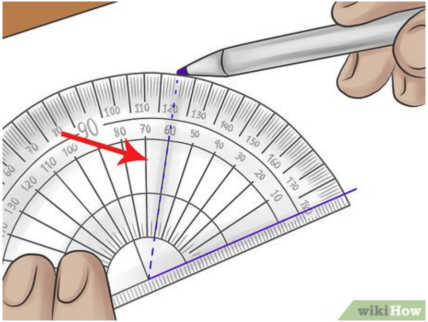
- *the two shortest sides of T provided T has one angle of at least 120°*
- *the three segments connecting the Fermat point for T to the original three vertices of T*

Torricelli's Construction

- Input: Triangle $T = \triangle abc$ where all angles measure less than 120°
- Steps
  1. Along edge ab of T construct an equilateral triangle using that edge and a new point x that is on the opposite side of the edge as c
  2. Repeat step 1 for the other two edges of T, introducing new points y and z across from b and a, respectively
  3. Join x and c, y and b, and z and a by a line segment
  4. The point of concurrency (intersection point of three lines) is the Fermat point p for T
  5. The shortest network is the line segments joining each of the original vertices, a, b, and c, with p
- Output: Fermat point p and shortest network connecting a, b, and c

Torricelli's Construction – tools



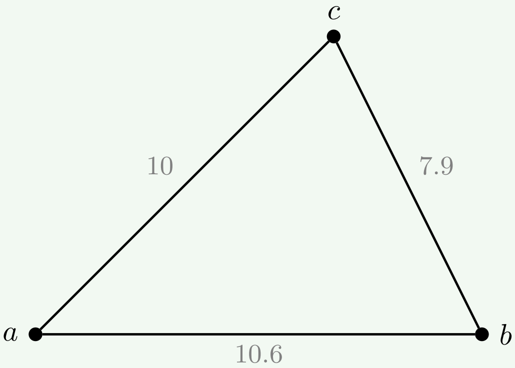


Note

Can be done using a ruler and a compass (or a protractor)

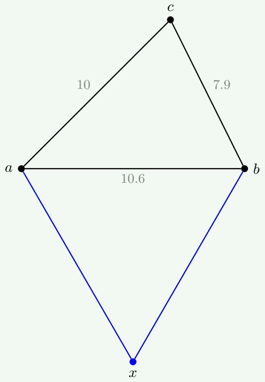
Torricelli's Construction – example

Example



Torricelli's Construction – example

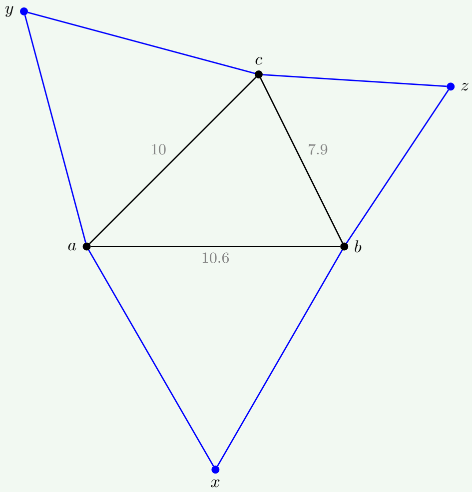
Example



- Step 1. form an equilateral triangle off edge ab with new point x on the opposite side of ab as c

Torricelli's Construction – example

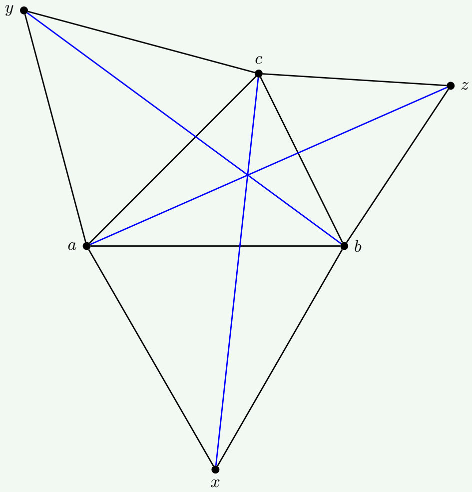
Example



- Step 2. repeat the same for edge ac and bc

Torricelli's Construction – example

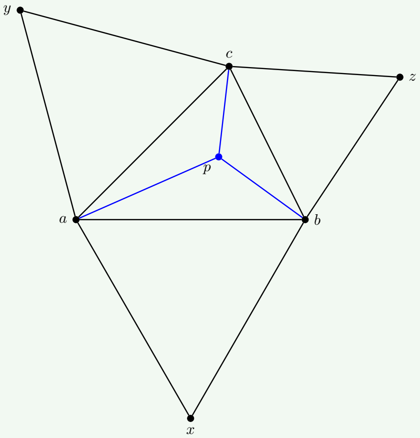
Example



- Step 3. join x and c, y and b, and z and a

Torricelli's Construction – example

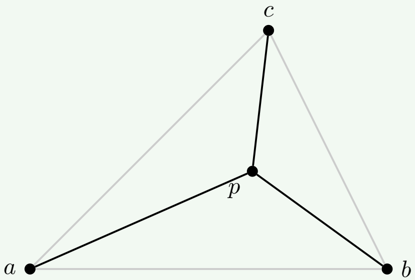
Example



- Step 4. find the Fermat point p
- Step 5. highlight the edges from p to each of the original vertices

Torricelli's Construction – example

Example



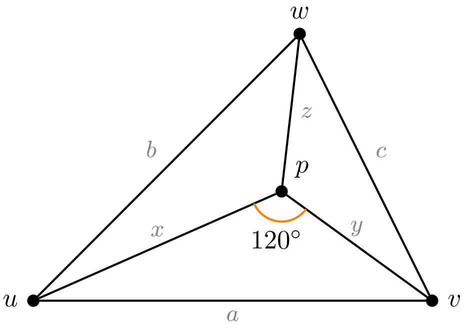
- Output: the shortest network consists of the edges from p back to each of the original vertices

Torricelli's Construction – length of the shortest network

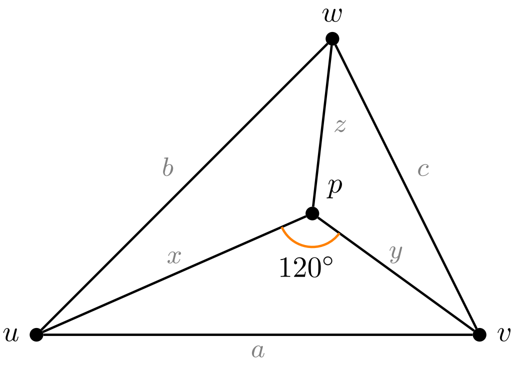
- Law of cosines

$$\begin{aligned} a^2 &= b^2 + c^2 - 2bc\cos(\angle uwv) \\ c^2 &= a^2 + b^2 - 2ab\cos(\angle wuv) \\ b^2 &= a^2 + c^2 - 2ac\cos(\angle wvu) \end{aligned}$$

- Solve for the angles
- Assign coordinates to vertices
- Solve for length x, y, z



Torricelli's Construction – length of the shortest network



- If ℓ denotes the length of the shortest network connecting vertices of a triangle having angles that measure less than $120°$ each and sides of lengths a, b, c

$$\ell = x + y + z = \sqrt{\frac{a^2 + b^2 + c^2}{2} + \frac{\sqrt{3}}{2}\sqrt{2a^2b^2 + 2a^2c^2 + 2b^2c^2 - (a^4 + b^4 + c^4)}}$$

Torricelli's Construction – length of the shortest network

Example

- Continuing from the previous example

$$a = 7.9, \quad b = 10, \quad c = 10.6$$

- With the formula we get the length of the network obtained

$$\ell = \sqrt{\frac{274.77}{2} + \frac{\sqrt{3}}{2}\sqrt{48978.7752 - 26519.7777}} \approx 16.345$$

- In comparison, the MST has a total length of 17.9
- The shortest network saves 1.555, or 8.69%

Trees and networks

- Shortest networks
- Metric Traveling Salesman Problem
- Flow and capacity
- Rooted trees

Traveling Salesman Problem

- How should a delivery service plan its route through a city to ensure each customer is reached?
- A traveling salesman has customers in numerous cities. He must visit each of them and return home, but wishes to do this with the least total cost
- Traveling Salesman Problem: shortest cycle that visits every city in a country $\rightarrow$ shortest Hamiltonian cycle (contains every vertex) of a weighted complete graph

Metric Traveling Salesman Problem

- In short mTSP
- Only considers the scenarios where the weights satisfy the *triangle inequality*
- For a weighted complete graph $K = (V, E, \omega)$, given any three vertices x, y, z

$$\omega\left(xy\right) + \omega\left(yz\right) \geq \omega\left(xz\right)$$

mTSP Algorithm

- mTSP Algorithm combines three ideas we have studied so far – Eulerian circuits, Hamiltonian cycles and MST
- A minimum spanning tree is modified by duplicating every edge, ensuring all vertices have even degree and allowing an Eulerian circuit to be obtained
- This circuit is then modified to create a Hamiltonian cycle
- Note that this procedure is guaranteed to work only when the underlying graph is complete
- It may still find a proper Hamiltonian cycle when the graph is not complete, but cannot be guaranteed to do so.

mTSP Algorithm

- Input: weighted complete graph K_n, where the weight function ω satisfies the triangle inequality
- Steps:
  1. Find a MST T for K_n
  2. Duplicate all the edges of T to obtain T^*
  3. Find an Eulerian circuit for T^*
  4. Convert the Eulerian circuit into a Hamiltonian cycle by skipping any previously visited vertex (except for the starting and ending vertex)
  5. Calculate the total weight
- Output: Hamiltonian cycle for K_n

mTSP Algorithm – example

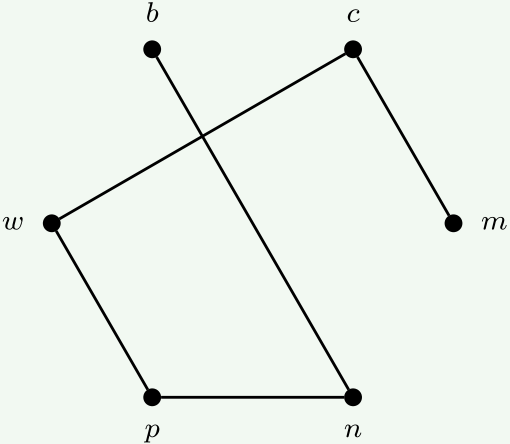
Example

Alice must visit six cities and needs to minimize the total distance

	Boston	Charlotte	Memphis	New York	Philadelphia	D.C.
Boston	.	840	1316	216	310	440
Charlotte	840	.	619	628	540	400
Memphis	1316	619	.	1096	1016	876
New York City	216	628	1096	.	97	228
Philadelphia	310	540	1016	97	.	140
Washington, D.C.	440	400	876	228	140	.

mTSP Algorithm – example

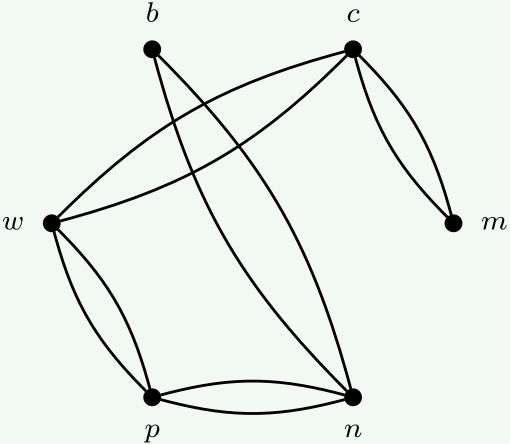
Example



- Step 1. MST

mTSP Algorithm – example

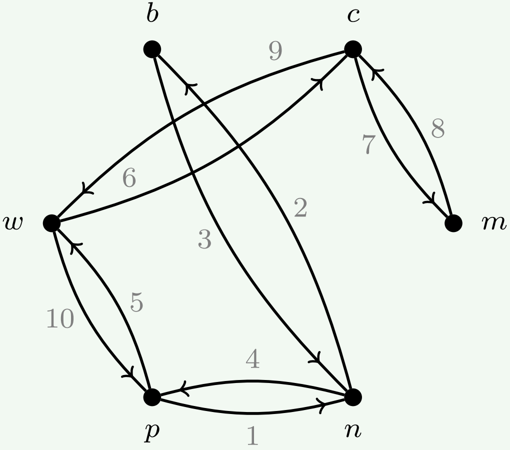
Example



- Step 2. Duplicate all edges of the MST

mTSP Algorithm – example

Example



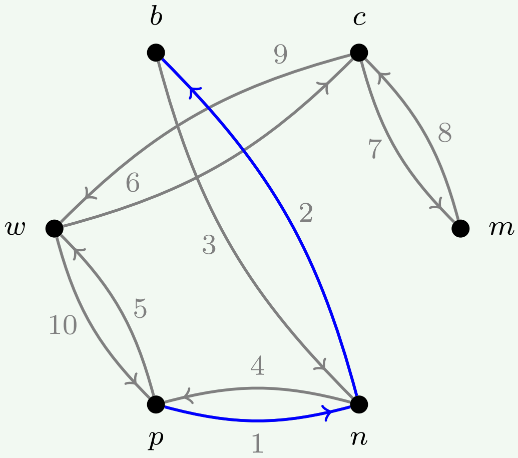
- Step 3. Find an Eulerian circuit $pnbnpwcmcwp$

mTSP Algorithm – example

Example

Step 4

- Follow the Eulerian circuit until we reach vertex b.
- Since we are looking for a Hamiltonian cycle, we cannot repeat vertices, so we cannot return to n
- The next vertex along the circuit that has not been previously visited is w

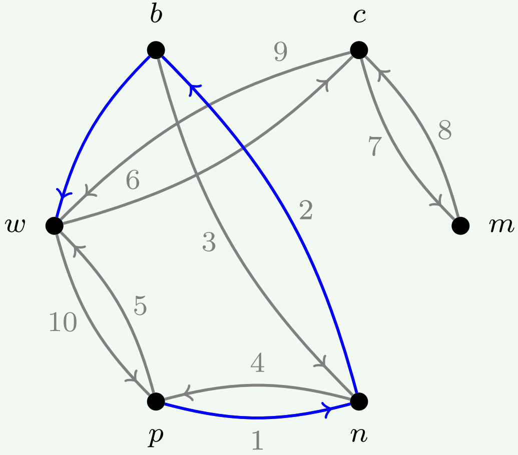


mTSP Algorithm – example

Example

Step 4

- Add the edge from b to w

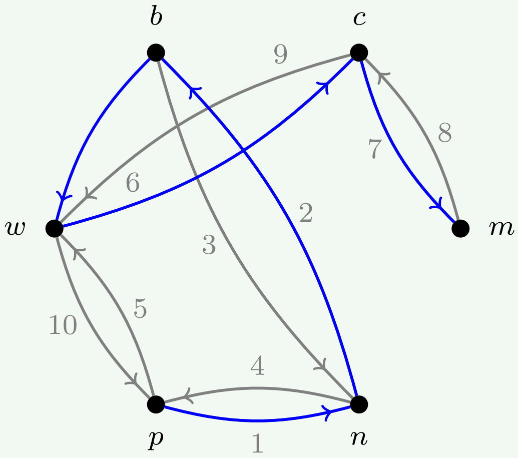


mTSP Algorithm – example

Example

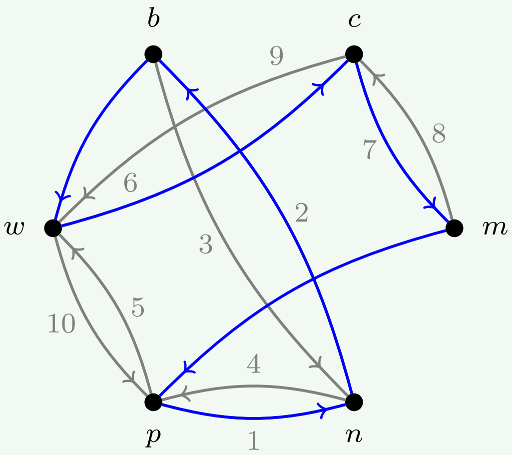
Step 4

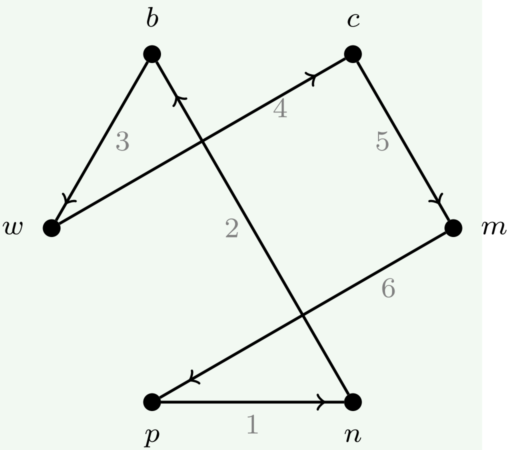
- We follow the circuit again until m is reached
- Again, we cannot return to c and at this point we must return to p since all other vertices have been visited



mTSP Algorithm – example

Example





mTSP Algorithm – example

Example

- Total weight of the MST: 1472 – the worst possible Hamiltonian cycle that can arise from it will have weight at most two times that 2944, due to the doubling of the edges
- Total weight of the Hamiltonian cycle: 2788
- Actual optimal cycle: 2781
- Our Hamiltonian cycle is off by a relative error of 0.25%

Trees and networks

- Shortest networks
- Metric Traveling Salesman Problem
- Flow and capacity
- Rooted trees

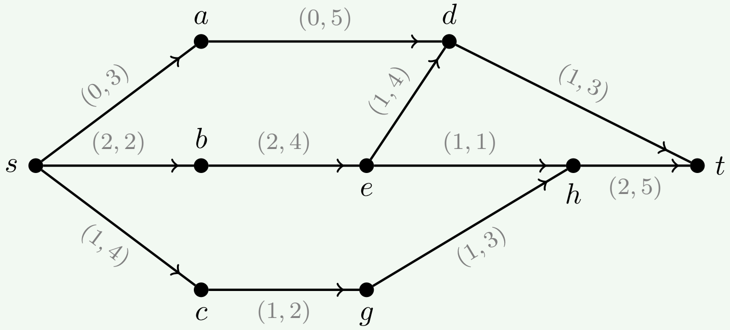
A different notion of a network

Definition

- A *network* is a connected digraph where each arc e has an associated nonnegative integer $c(e)$, called a *capacity*.
- The network has a designated starting vertex s called the *source* and a designated ending vertex t called the *sink*
- A *flow* f is a function that assigns a value $f(e)$ to each arc e of the network

Network – example

Example



- Each arc is given a two-part label, $(f(e), c(e))$
- The first component is the flow along the arc
- The second component is the capacity

Network

- Source, sink: reminiscent of a system of pipes with water coming from the source, traveling through some configuration of the piping to arrive at the sink
- Using this analogy further, flow should travel in the indicated direction of the arcs, no arc can carry more than its capacity, and the amount entering a junction point (a vertex) should equal the amount leaving.

Feasible flow

Definition

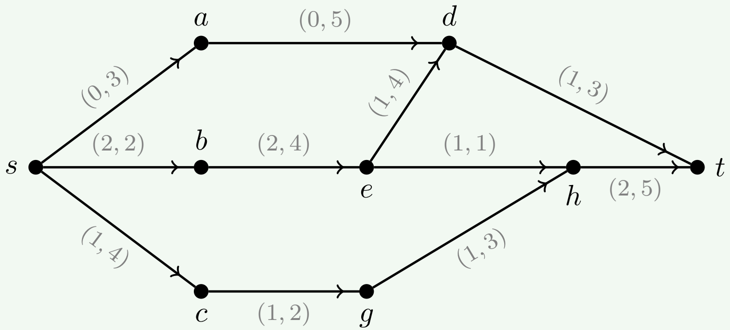
For a vertex v, let $f^-(v)$ represent the total flow entering v and $f^+(v)$ represent the total flow exiting v. A flow is *feasible* if it satisfies the following conditions

- $f(e) \geq 0$ for all arcs e
- $c(e) \geq f(e)$ for all arcs e
- $f^+(v) = f^-(v)$ for all vertices other than s and t
- $f^-(s) = f^+(t) = 0$

- The notation for in-flow and out-flow mirrors that for in-degree and out- degree of a vertex, though here we are adding the flow value for the arcs entering or exiting a vertex
- The requirement that a flow is non-negative indicates the flow must travel in the direction of the arc, as a negative flow would indicate items going in the reverse direction
- The final condition is not necessary in theory, but more logical in practice and simplifies our analysis of flow problems

Network – example

Example



- $f^+(s) = 0 + 2 + 1 = 3,\ f^-(t) = 1 + 2 = 3$
- $c(ad) = 5,\ f(ad) = 0,\ c(ad) > f(ad)$
- $f^+(b) = f^-(b) = 2$

Maximum flow

Definition

- The *value* of a flow is defined as

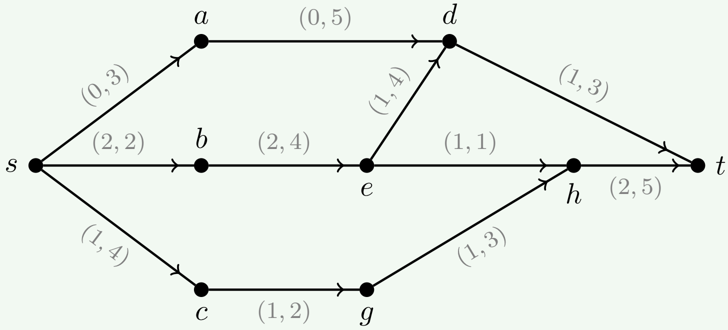
$$|f| = f^+(s) = f^-(t),$$

  i.e. the amount exiting the source which must also equal the flow entering the sink
- A *maximum flow* is a feasible flow of largest value

- In practice, we use integer values for the capacity and flow, though this is not required.
- *How to find the maximum flow?*

Network – example

Example



- Feasible flow, value 3
- Finding maximum flow is not as simple as putting every arc at capacity – e.g. if we had a flow of 5 along arc ad, flow along dt would be 6, more than its capacity

Slack

Definition

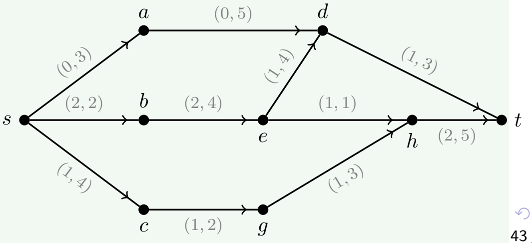
Let f be a flow along a network. The *slack* k of an arc is the difference between its capacity and flow

$$k(e) = c(e) - f(e).$$

Slack edge: $k(e) > 0$

Example

- $k(sa) = 3$, $k(sc) = 3$, $k(sb) = 0$
- We may want to increase flow along sa and sc but not sb

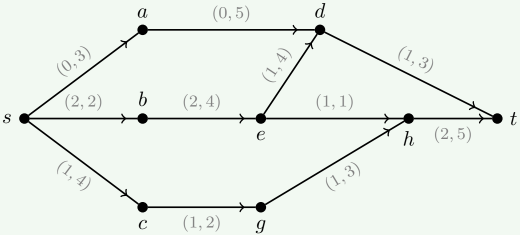


Chain

Definition

A *chain* K is a path in a digraph where the direction of the arcs are ignored

Example



- Both $sadt$ and $sadeht$ are chains
- $sadeht$ is not a directed path

Augmenting Flow Algorithm (Ford-Fulkerson Algorithm)

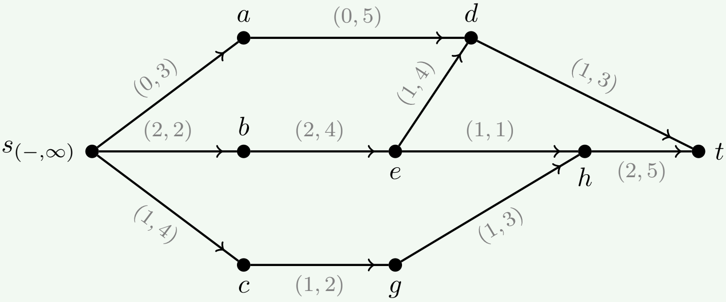
- Similar to Dijkstra's Algorithm which found the shortest path in a graph (or digraph)
- Vertices will be assigned two-part labels that aid in the creation of a chain on which the flow can be increased
- The label consists of two parts, the second component is denoted by $\sigma(v)$
- Input: Network $G = (V, E, c)$, where each arc is given a capacity c, and a designated source s and sink t
- Output: Maximum flow f

Augmenting Flow Algorithm – steps

1. Label s with $(-, \infty)$, set $\sigma(v) = \infty$ for other vertices
2. Choose a labeled vertex x, scan x:
   a. For any arc yx, if $f(yx) > 0$ and y is unlabeled, then label y with $(x^-, \sigma(y))$, where $\sigma(y) = \min\{\sigma(x), f(yx)\}$
   b. For any arc xy, if $k(xy) > 0$ and y is unlabeled, then label y with $(x^+, \sigma(y))$, where $\sigma(y) = \min\{\sigma(x), k(xy)\}$
3. If t has been labeled, go to Step 4. Otherwise, choose a different labeled vertex that has not been scanned and go to Step 2. If all labeled vertices haven been scanned, then f is a maximum flow.
4. Find an $s - t$ chain K of slack edges by backtracking from t to s. Along the edges of K, increase the flow by $\sigma(t)$ units if they are in the forward direction and decrease by $\sigma(t)$ units in they are in the backward direction. Remove all vertex labels except that of s and return to Step 2

Augmenting Flow Algorithm – example

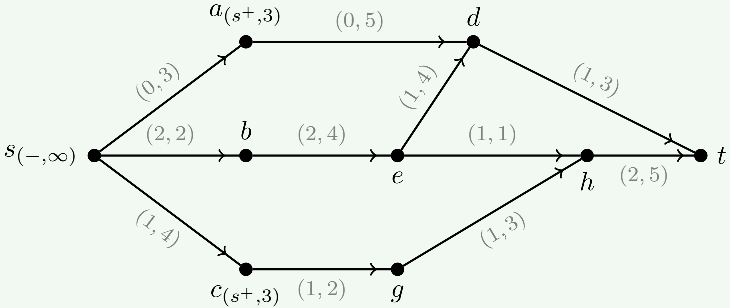
Example



- Step 1. Label s as $(-, \infty)$

Augmenting Flow Algorithm – example

Example

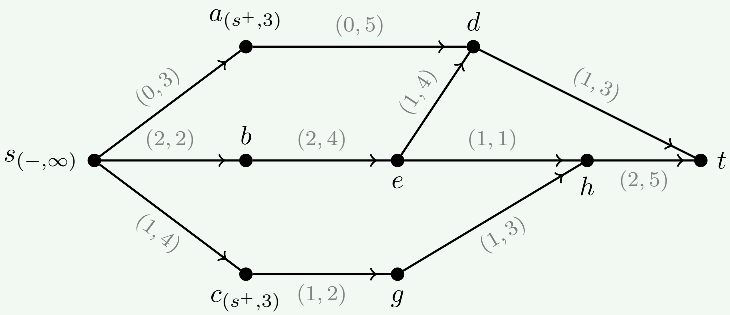


Step 2.

- There are no arcs to s.
- Arcs out of s: sa, sb, sc, with slack $3, 0, 3$
- Label a with $(s^+, 3)$, c with $(s^+, 3)$. Leave b unlabeled since sb has slack 0

Augmenting Flow Algorithm – example

Example

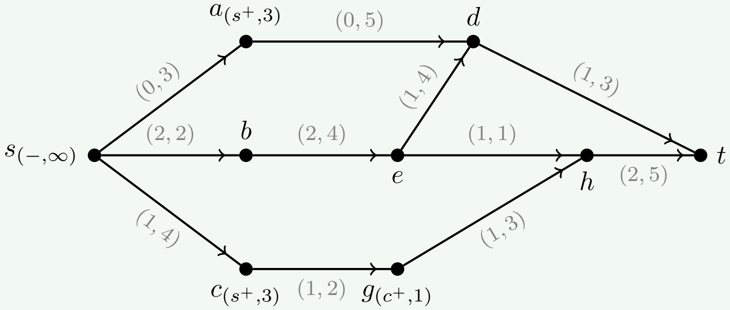


Step 3.

- As t is not labeled, we will choose a different labeled vertex that has not been scanned– a or c
- Choose c.

Augmenting Flow Algorithm – example

Example

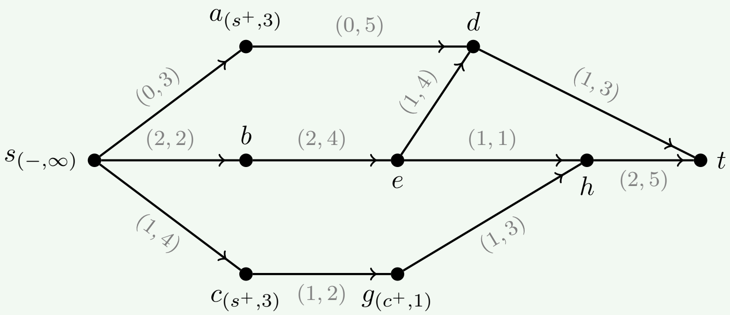


Step 2.

- The only arc going into c is from a labeled vertex
- Consider the arcs out of c – there is only cg with slack of 1
- Label g as $(c^+, 1)$, where $\sigma(g) = \min\{\sigma(c), k(cg)\} = \min\{3, 1\} = 1$

Augmenting Flow Algorithm – example

Example

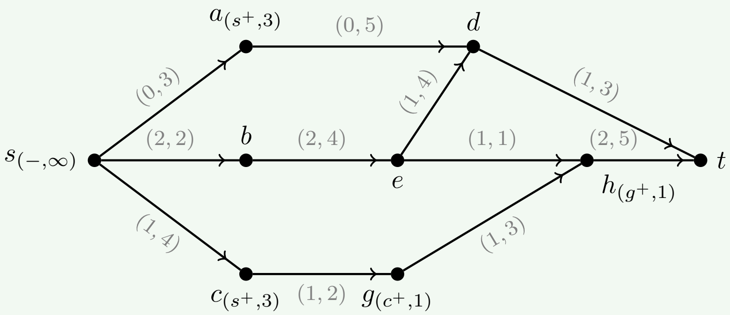


Step 3.

- t is not labeled
- We can scan either a or g
- Choose g

Augmenting Flow Algorithm – example

Example

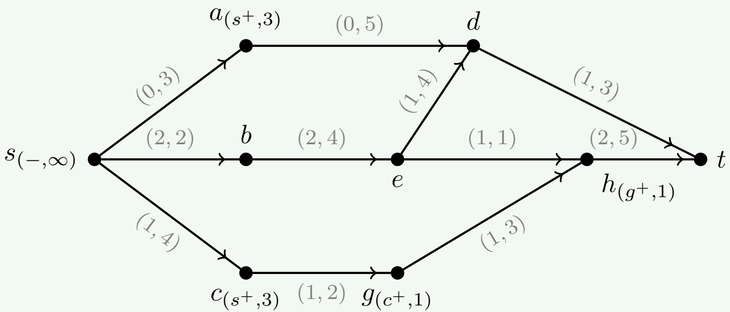


Step 2.

- The only arc going into g is from a labeled vertex
- The only arc out of g is gh
- Label h as $(g^+,1)$, where $\sigma(h) = \min\{\sigma(g), k(gh)\} = \min\{1,2\} = 1$

Augmenting Flow Algorithm – example

Example

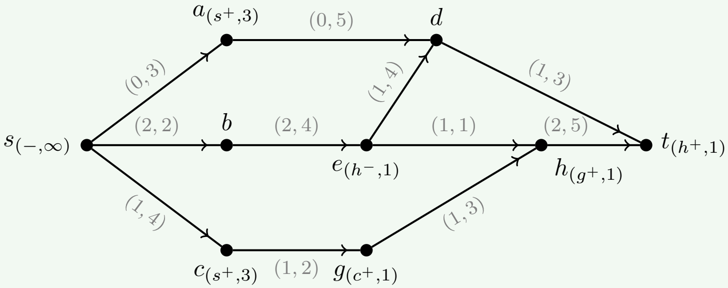


Step 3.

- t is not labeled, we can choose a or h
- Choose h

Augmenting Flow Algorithm – example

Example

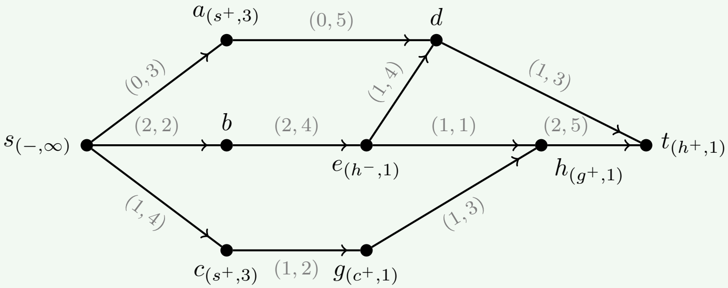


Step 2.

- There is one unlabeled vertex with an arc going into h, namely e, label e with $(h^-, 1)$ since $\sigma(e) = \min\{\sigma(h), f(he)\} = \min\{1, 1\} = 1$
- Label t as $(h^+, 1)$, where $\sigma(t) = \min\{\sigma(h), k(ht)\} = \min\{1, 3\} = 1$

Augmenting Flow Algorithm – example

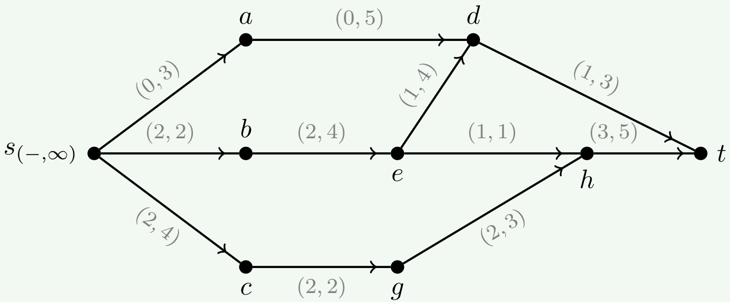
Example



- Step 3. t is labeled, go to Step 4
- Step 4. we find an $s-t$ chain K of slack edges. Backtracking from t to s gives the chain $scght$

Augmenting Flow Algorithm – example

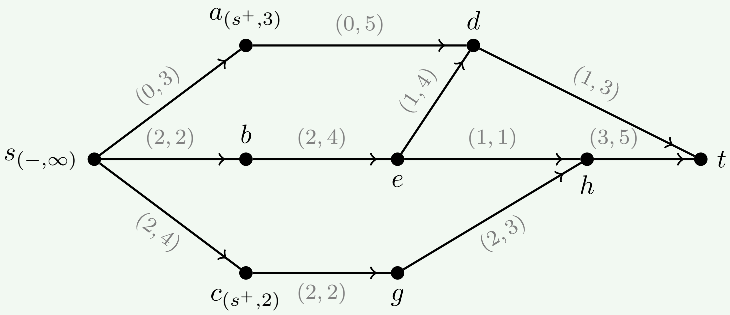
Example



- Step 4. we find an $s-t$ chain K of slack edges. Backtracking from t to s gives the chain $scght$
  - Increase the flow by $\sigma(t)=1$ units along each of these edges since all are in the forward direction.
  - Update the network flow and remove all labels except for s

Augmenting Flow Algorithm – example

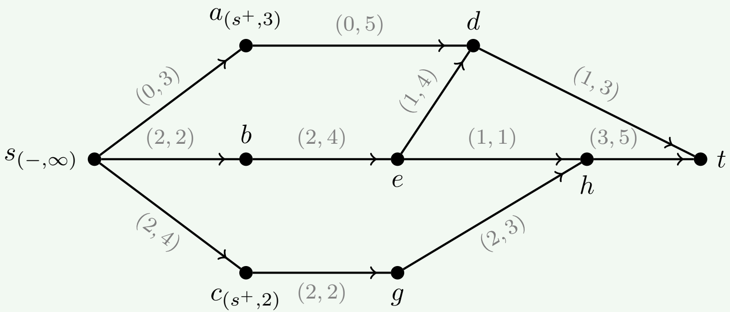
Example



- Step 2. Label a with $(s^+, 3)$ and c with $(s^+, 2)$
- Step 3. Choose c to scan

Augmenting Flow Algorithm – example

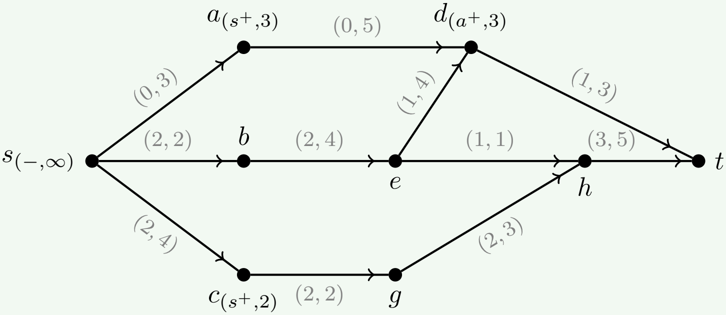
Example



- Step 2. No vertex to label, $k(cg) = 0$
- Step 3. Choose a to scan

Augmenting Flow Algorithm – example

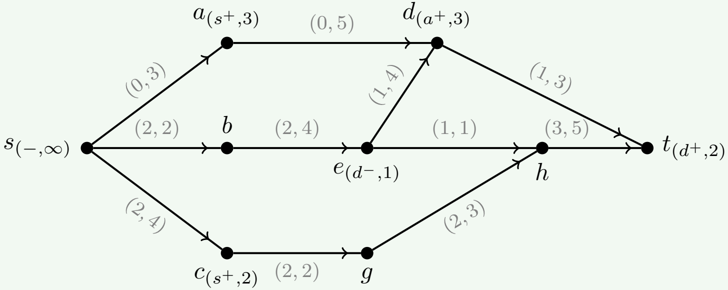
Example



- Step 2. Label d with $(a^+, 3)$, where $\sigma(d) = \min\{\sigma(a), k(ad)\} = \min\{3, 5\} = 3$
- Step 3. Only unscanned labeled vertex is d

Augmenting Flow Algorithm – example

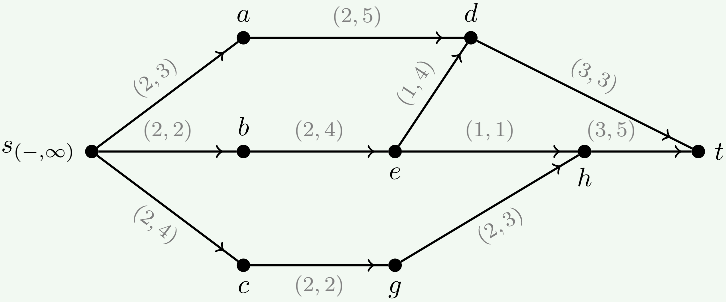
Example



- Step 2. Label e with $(d^-, 1)$, where $\sigma(e) = \min\{\sigma(d), f(ed)\} = \min\{3, 1\} = 1$. Label t with $(d^+, 2)$, $\sigma(t) = \min\{\sigma(d), k(dt)\} = \min\{3, 2\} = 2$
- Step 3. t is labeled, go to step 4
- Step 4. Find chain $sadt$

Augmenting Flow Algorithm – example

Example

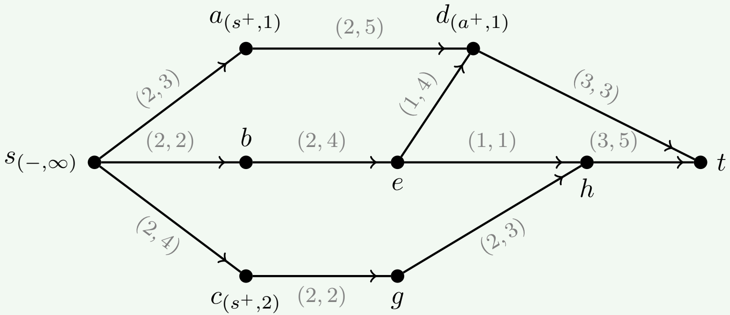


Step 4.

- Find chain $sadt$, increase the flow by $\sigma(t) = 2$ units along each of these edges since all are in the forward direction
- Update the network flow and remove all labels except for s

Augmenting Flow Algorithm – example

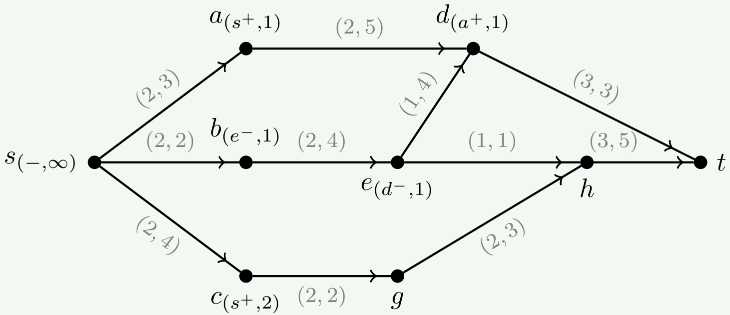
Example



- Step 2. Label a, c
- Step 3. Choose a
- Step 2. Label d
- Step 3. Choose d

Augmenting Flow Algorithm – example

Example



- Step 2. Label e as $(d^-, 1)$, t is not given a label since $k(dt) = 0$
- Step 3. Choose e
- Step 2. Label b
- Step 3. No vertices to be labeled, we get a maximum flow

Augmenting Flow Algorithm

- When the Augmenting Flow Algorithm halts, a maximum flow has been achieved, though understanding why this flow is indeed maximum requires additional terminology and results

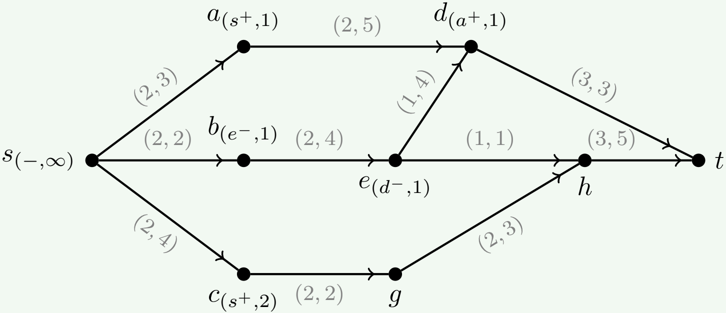
Cut

Definition

- Let P be a set of vertices and $\overline{P} = V - P$.
- A *cut* $(P, \overline{P})$ is the set of all arcs xy where $x \in P$ and $y \in \overline{P}$
- An $s - t$ *cut* is a cut in which $s \in P$, $t \in \overline{P}$

Cut – example

Example



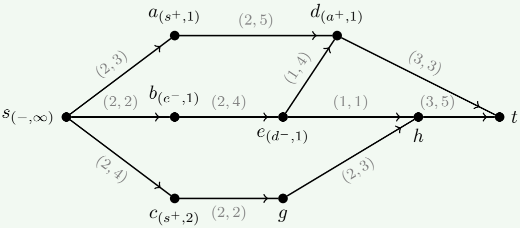
- $P = \{s, a, e, g\}$, $\overline{P} = \{b, c, d, h, t\}$
- $(P, \overline{P}) = \{sb, sc, ad, ed, eh, gh\}$
- Note: $be \notin (P, \overline{P})$ because $b \in \overline{P}$, $e \in P$

Capacity

Definition

The *capacity* of a cut, $c(P, \overline{P})$, is defined as the sum of the capacities of the arcs that comprise the cut.

Example



- $P = \{s, a, e, g\}$, $\overline{P} = \{b, c, d, h, t\}$, $(P, \overline{P}) = \{sb, sc, ad, ed, eh, gh\}$, $c(P, \overline{P}) = 2 + 4 + 5 + 4 + 1 + 3 = 19$
- $P = \{s\}$, $c(P, \overline{P}) = 9$

Max Flow-Min Cut

Theorem

In any directed network, the value of a maximum $s - t$ flow equals the capacity of a minimum $s - t$ cut.

- The difficulty in using this result to prove a flow is maximum is in finding the minimum cut
- We can use the vertex labeling procedure to obtain our minimum cut

Min-Cut Method

Steps

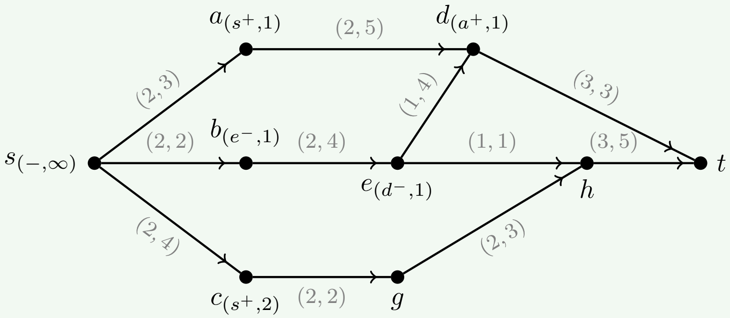
1. Let $G = (V, A, c)$ be a network with a designated source s and sink t and each arc is given a capacity c
2. Apply the Augmenting Flow Algorithm
3. Define an $s - t$ cut $(P, \overline{P})$ where P is the set of labeled vertices from the final implementation of the algorithm
4. $(P, \overline{P})$ is a minimum $s - t$ cut for G

Note

In practice, we can perform the Augmenting Flow Algorithm and the Min-Cut Method simultaneously, thus finding a maximum flow and providing a proof that it is maximum (through the use of a minimum cut) in one complete procedure.

Min-Cut Method – example

Example



- $P = \{s, a, b, c, d, e\}$, $\overline{P} = \{g, h, t\}$
- $(P, \overline{P}) = \{dt, eh, cg\}$
- $c(P, \overline{P}) = 3 + 1 + 2 = 6$

Trees and networks

- Shortest networks
- Metric Traveling Salesman Problem
- Flow and capacity
- Rooted trees

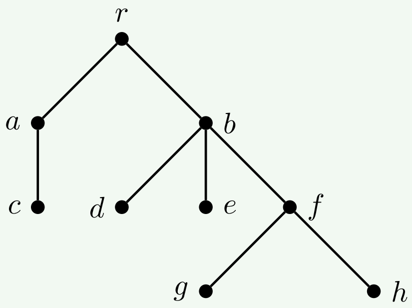
Definition

Definition

- A *rooted tree* is a tree T with a special designated vertex r, called the *root*
- The *level* of any vertex in T is defined as the length of its shortest path to r
- The *height* of a rooted tree is the largest level for any vertex in T

Example

- root r: level 0
- a, b: level 1
- c, d, e, f: level 2
- g, h: level 3
- height of the tree: 3



Terminologies

Definition

Let T be a tree with root r. Then for any vertices x and y

- y is on the unique path from x to r: x is a *descendant* of y; y is an *ancestor* of x
- x is a descendant of y and exactly one level below y: x is a *child* of y, y is a *parent* of x
- x is a *sibling* of y if x and y has the same parent

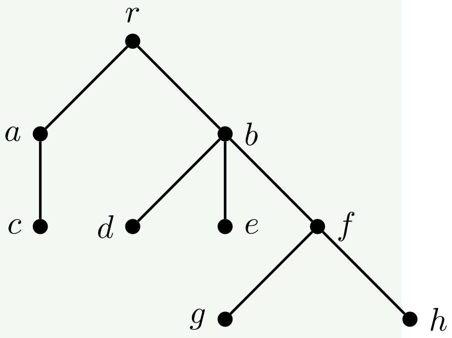
Note

Analogy: family tree

Terminologies – example

Example

- parent of a: r
- child of a: c
- parent of e: b
- e has no children
- ancestors of g: f, b, r – unique path from g to r is $gfbr$
- descendants of b: d, e, f, g, h
- siblings of d: e, f



Depth-First Search Tree

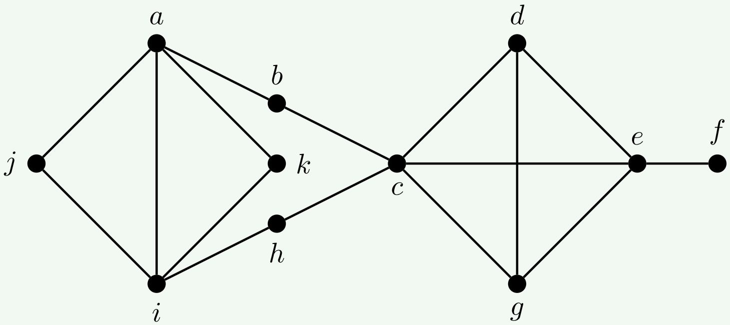
- Main idea is to travel along a path as far as possible from the root of a given graph
- If this path does not encompass the entire graph, then branches are built off this central path to create a tree
- The formal description of this algorithm relies on an ordered listing of the neighbors of each vertex and uses this order when adding new vertices to the tree
- For simplicity, we will always use an alphabetical order when considering neighbor lists
- Input: Simple (no multi-edges or loops) connected graph $G = (V, E)$ and a designated root vertex r
- Output: Depth-first search tree T

Depth-First Search Tree – steps

1. Initialize the DFS tree $T = (V', E')$ with $V' = \{r\}$ and $E' = \emptyset$. Set r as the current vertex v.
2. Select the first unvisited neighbor x of the current vertex v. Add vertex x and edge vx to T, and recursively repeat Step 2 with x as the new current vertex until no unvisited neighbors remain.
3. If all vertices of G are now in T, then T is the depth-first search tree. Otherwise, backtrack the path from the last visited vertex x to the root in T to find a vertex v that has unvisited neighbor. Use v as the current vertex and repeat Step 2.

Depth-First Search Tree – example

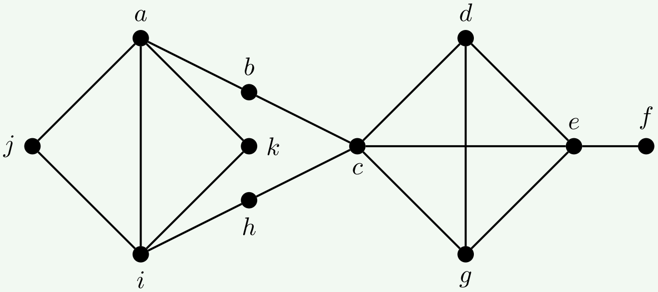
Example

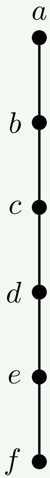


- Suppose a is the root
- Step 1. current vertex is a
- Step 2. add b, c, d, e, f, this stops with f since f has no further neighbors in G

Depth-First Search Tree – example

Example

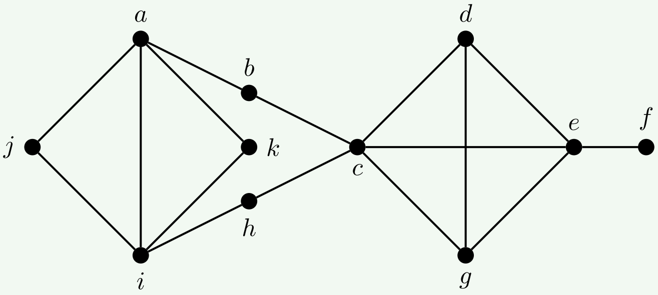


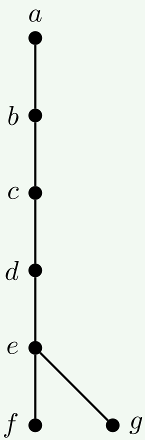


- Step 3. backtracking along the path, the first vertex with an unvisited neighbor is e
- Step 2. add edge eg and vertex g to T

Depth-First Search Tree – example

Example

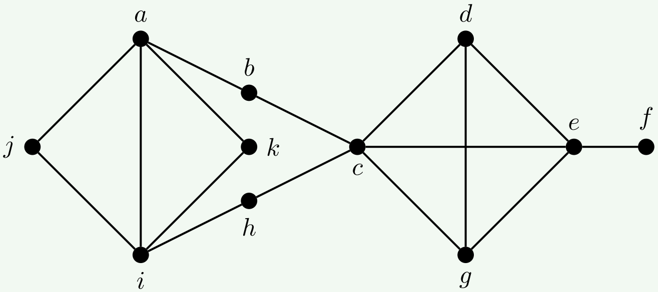


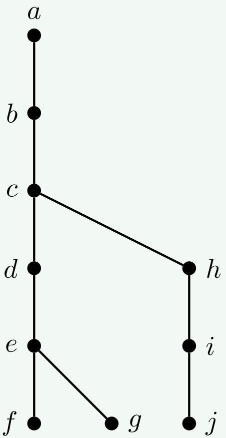


- Step 3. backtracking along the path from g to a, the first vertex with an unvisited neighbor is c
- Step 2. add h, i, j

Depth-First Search Tree – example

Example

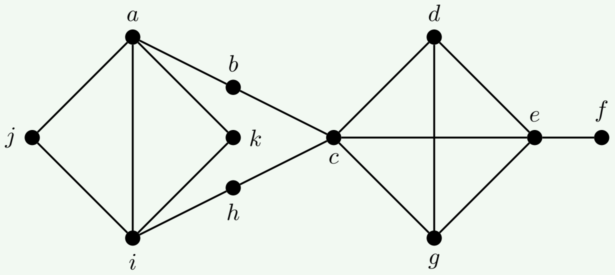


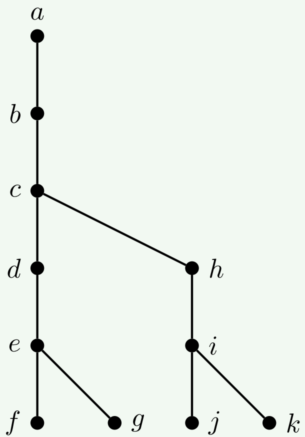


- Step 3. backtracking along the path from j to a, the first vertex with an unvisited neighbor is i
- Step 2. add k

Depth-First Search Tree – example

Example





- T contains all the vertices of G
- The resulting tree is the depth-first search tree
- Height 5, one vertex each at level 1 and 2, two vertices each at levels 3 and 4, four vertices at level 5

Remark

- If the graph is not connected, we can slightly adjust the algorithm and get a forest as the output
- Search for unvisited vertices when no vertex with unvisited neighbors can be found

Homework

- **Submission Deadline:** T8-T11, Wednesday at 23:59; T12 11th Dec 23:59
- **Submission Method:** Submissions may be made either through AIS or in person. T8-T11 Viki, T12 me
- **Requirements:** All answers must be written clearly and include detailed solution steps.
- **Late Submissions:** A penalty of 2 marks will be applied for each missed or late submission.
- **Satisfactory Work:** A submission is considered satisfactory if it includes detailed, well-presented solutions for all questions and demonstrates clear effort.
- **Unsatisfactory Work:** If a submission is deemed unsatisfactory, you have one more chance to refine your solutions-second submission should be within one week from the deadline. Failure to do so will result in a deduction of 2 marks.

Homework

- Next week homework will be Tutorial 10
  - Deadline: 3rd Dec, 23:59
- **Extra credits**: submit homework to Tutorials 1-7. Each satisfactory submission earns 1 mark.
  - Submission through AIS or in person to me
  - Deadline: 31st, Dec, 23:59

Second Midterm

- **Location:** Room 1.37 (same room as the lecture and tutorial)
- **Calculators:** Calculators are permitted; however, the use of phone calculators is strictly prohibited
- Toilet breaks are **not** allowed
- **Time:** Week 11, 5th Dec, from $9{:}00$ to $12{:}00$
- **Content:** The exam will cover the same questions as last time
- Come before 9am