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 3.1, 3.2, 4.1, 4.2
  - Free copy online

Lecture outline

- Dijkstra's Algorithm
- Project scheduling
- Critical path
- Trees
- Spanning trees

Paths and spanning trees

- Dijkstra's Algorithm
- Project scheduling
- Critical path
- Trees
- Spanning trees

Shortest path problem

- A path is a sequence of vertices in which there is an edge between consecutive vertices and no vertex is repeated
- Weight: distance, cost, time, etc.
- Shortest path: the path of the least total weight
- A shortest path exists if the graph is connected
- Scenario: fastest route to travel from one location to another

Dijkstra's Algorithm

- Proposed in 1956 by Edsger W. Dijkstra
- Almost every GIS (Geographic Information System, or mapping software) uses a modification of Dijkstra' s algorithm to provide directions
- Numerous versions of Dijkstra' s Algorithm exist
- See original algorithm: DIJKSTRA, E. (1959). A Note on Two Problems in Connexion with Graphs. Numerische Mathematik, 1, 269-271.

Dijkstra's Algorithm – notations

- Each vertex is given a two-part label

$$L(v) = (x, \omega(v))$$

- x: the name of the vertex used to travel to v
- $\omega(v)$: the weight of the path that was used to get to v from the designated starting vertex
- F: a set of vertices that are not highlighted yet

Dijkstra's Algorithm – input and output

- **Input**: Weighted connected graph $G = (V, E)$ and vertices designated as *Start* and *End*
- **Output**: Highlighted path from *Start* to *End* and total weight $\omega\,(\textit{End})$

Dijkstra's Algorithm – steps

1. For each vertex v of G, assign a label $L(v)$:

$$L(v) = \begin{cases} (-, 0), & \text{if } v = \textit{Start} \\ (-, \infty), & \text{Otherwise} \end{cases}$$

   Highlight *Start*. $F = V - \{\textit{Start}\}$. Let *current vertex*$=$ *Start*.

2. Update the labels for each vertex v in F that is a neighbor of *current vertex*, say u:

$$L(v) = \begin{cases} (u, \omega(u) + \omega(uv)), & \text{if } \omega(u) + \omega(uv) < \omega(v) \\ L(v), & \text{Otherwise} \end{cases}$$

3. Highlight the vertex v in F with the lowest weight as well as the edge used to update the label. Remove v from F. Redefine *current vertex*$= v$.
4. Repeat steps 2 and 3 until the vertex *End* has been highlighted.

Dijkstra's Algorithm – steps

5. The shortest path from *Start* to *End* is found by tracing back from *End* using the first component of the labels. The total weight of the path is the weight for *End* given in the second component of its label.

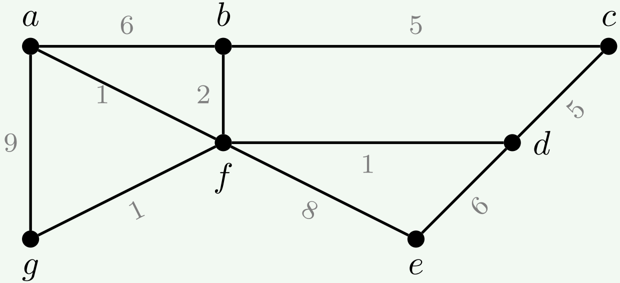
Note

- The set F of vertices consists of all un-highlighted vertices and all are under consideration for becoming the next highlighted vertex
- It is important that we do not only consider the neighbors of the last vertex highlighted, as a path from a previously chosen vertex may in fact lead to the shortest path

Dijkstra's Algorithm – example

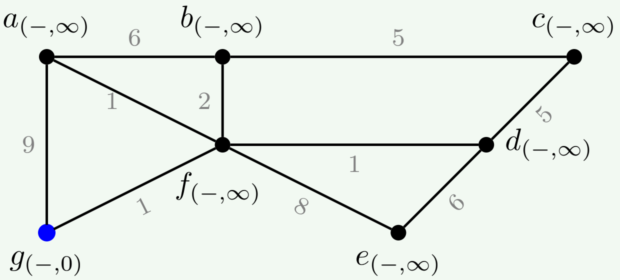
Example

Take *Start*$= g$, and *End*$= c$



Dijkstra's Algorithm – example

Example



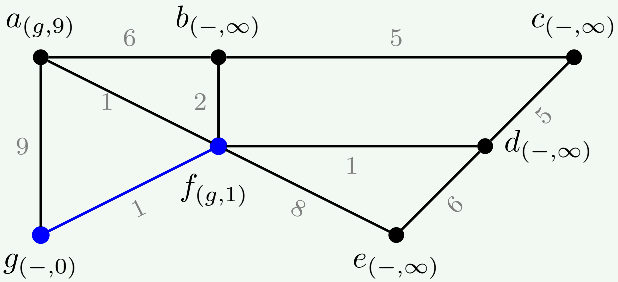
- Step 1. Highlight g. Label

$$L(v) = \begin{cases} (-,0) & v = g \\ (-,\infty) & \text{Otherwise} \end{cases}$$

$F = \{a, b, c, d, e, f\}$. *current vertex*$= g$.

Dijkstra's Algorithm – example

Example



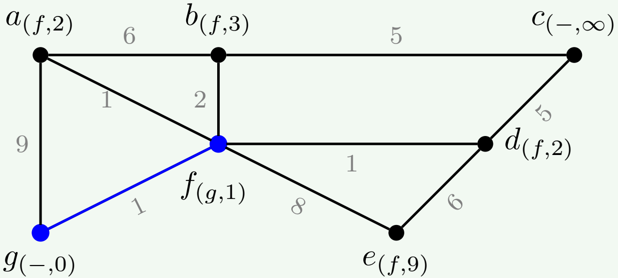
- Step 2. $F = \{a, b, c, d, e, f\}$. Neighbors of $g : a, f$

$$\begin{aligned} \omega(g) + \omega(ga) &= 0 + 9 = 9 < \infty = \omega(a) \Longrightarrow L(a) = (g, 9) \\ \omega(g) + \omega(gf) &= 0 + 1 = 1 < \infty = \omega(f) \Longrightarrow L(f) = (g, 1) \end{aligned}$$

- Step 3. minimum weight: f; highlight gf and f. $F = \{a, b, c, d, e\}$. *current vertex*$= f$.

Dijkstra's Algorithm – example

Example

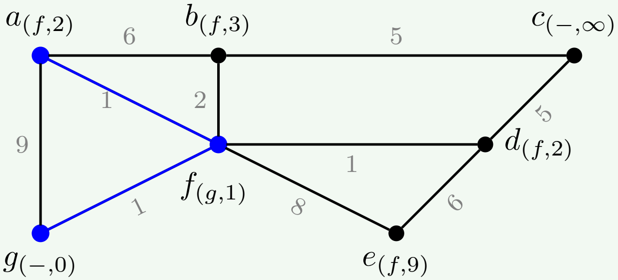


- Step 2. $F = \{a, b, c, d, e\}$. Neighbors of f: a, b, d, e

$$\begin{aligned}
\omega(f) + \omega(fa) &= 1 + 1 = 2 < 9 = \omega(a) \Longrightarrow L(a) = (f, 2) \\
\omega(f) + \omega(fb) &= 1 + 2 = 3 < \infty = \omega(b) \Longrightarrow L(b) = (f, 3) \\
\omega(f) + \omega(fd) &= 1 + 1 = 2 < \infty = \omega(d) \Longrightarrow L(d) = (f, 2) \\
\omega(f) + \omega(fe) &= 1 + 8 = 9 < \infty = \omega(e) \Longrightarrow L(e) = (f, 9)
\end{aligned}$$

Dijkstra's Algorithm – example

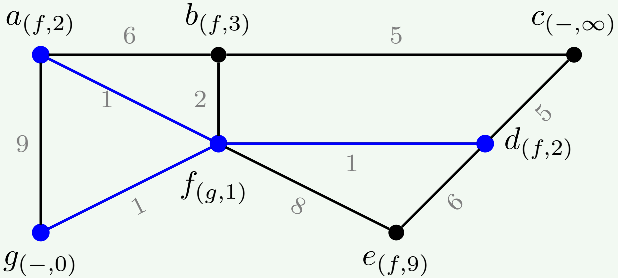
Example



- Step 3. $F = \{a, b, c, d, e\}$. The minimum weight for all vertices in F is that of a or d. Randomly choose one.
  - Let us highlight fa and a.
  - $F = \{b, c, d, e\}$.
  - *current vertex*$= a$.

Dijkstra's Algorithm – example

Example



- Step 2. $F = \{b, c, d, e\}$. Neighbor of a: b

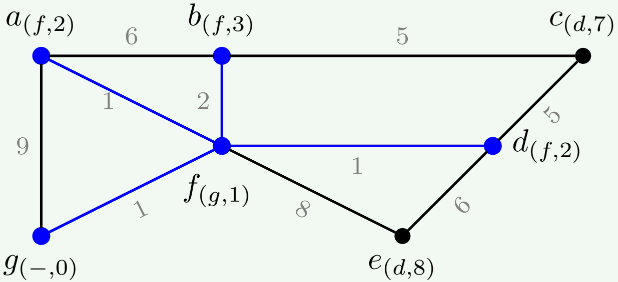
$$\omega(a) + \omega(ba) \quad = \quad 1 + 6 = 8 > 2 = \omega(b)$$

  We do not update label for b

- Step 3. Highlight fd and d. $F = \{b, c, e\}$. *current vertex*$= d$.

Dijkstra's Algorithm – example

Example



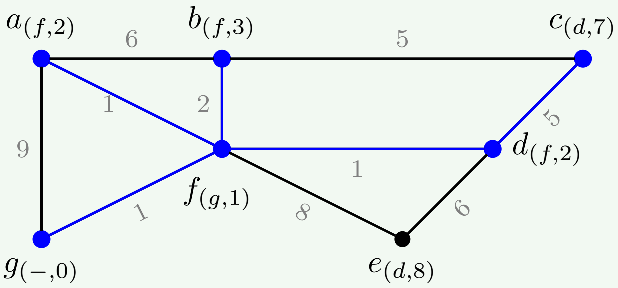
- Step 2. $F = \{b, c, e\}$. Neighbors of d: c, e

$$\begin{aligned} \omega(d) + \omega(dc) &= 2 + 5 = 7 < \infty = \omega(c) \Longrightarrow L(c) = (d, 7) \\ \omega(d) + \omega(de) &= 2 + 6 = 8 < 9 = \omega(e) \Longrightarrow L(e) = (d, 8) \end{aligned}$$

- Step 3. Highlight fb and b. $F = \{c, e\}$. *current vertex*$= b$.

Dijkstra's Algorithm – example

Example



- Step 2. $F = \{c, e\}$. Neighbor of b: $c\ \omega(b) + \omega(bc) = 3 + 5 = 8 > 7 = \omega(c)$
- Step 3. Highlight dc and c.
- Step 4. This terminates the iterations since we have reached *End*
- Step 5. The shortest path from g to c is $gfdc$, total weight 7

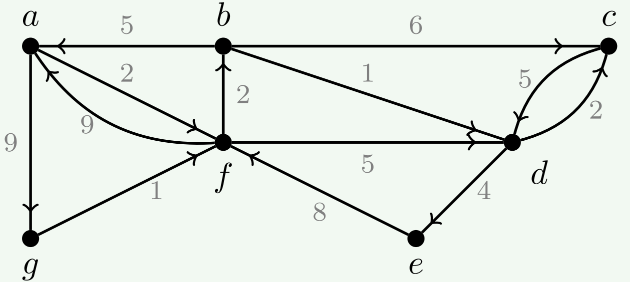
Dijkstra's Algorithm for digraphs

- A digraph is a graph in which the edges now have a direction associated to them, which could be used to model a one-way street.
- Arc yx, x is *head*, y is *tail*
- Instead of neighbors in Step 2, we consider the vertices that are heads for edges with the current vertex as a tail, called *out-neighbors*

Dijkstra's Algorithm for digraphs – example

Example

Start= g, *End*= c



We can record the changes with a table

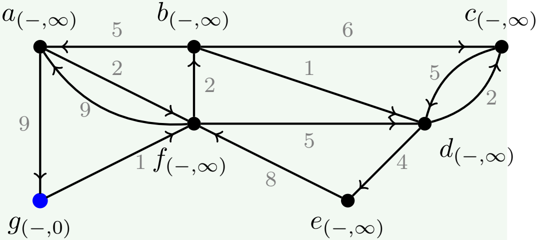
g	a	b	c	d	e	f

Dijkstra's Algorithm for digraphs – example

Example

- Step 1. *current vertex*$= g$,
  $F = \{a, b, c, d, e, f\}$

g	a	b	c	d	e	f
0	∞	∞	∞	∞	∞	∞



Dijkstra's Algorithm for digraphs – example

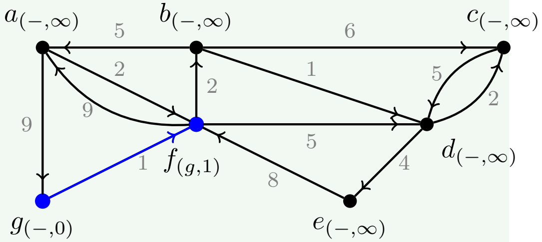
Example

- Step 2. *current vertex*$= g$,
  $F = \{a, b, c, d, e, f\}$,
  out-neighbors of g: f

$$\omega(g) + \omega(gf) = 0 + 1 < \infty$$

- Step 3. *current vertex*$= f$,
  $F = \{a, b, c, d, e\}$

	a	b	c	d	e	f
	∞	∞	∞	∞	∞	∞
$g(0)$	∞	∞	∞	∞	∞	$(g,1)$

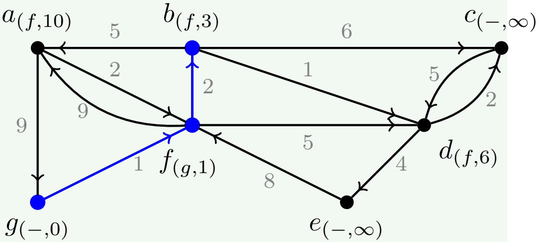


Dijkstra's Algorithm for digraphs – example

Example

- Step 2. *current vertex*$= f$,
  $F = \{a, b, c, d, e\}$,
  out-neighbors of f: a, b, d

$$\omega(f) + \omega(fa) = 1 + 9 = 10 < \infty,$$
$$\omega(f) + \omega(fb) = 1 + 2 = 3 < \infty,$$
$$\omega(f) + \omega(fd) = 1 + 5 = 6 < \infty.$$



- Step 3. *current vertex*$= b$,
  $F = \{a, c, d, e\}$

	a	b	c	d	e	f
	∞	∞	∞	∞	∞	∞
$g(0)$	∞	∞	∞	∞	∞	$(g,1)$
$f(1)$	$(f,10)$	$(f,3)$	∞	$(f,6)$	∞	$(g,1)$

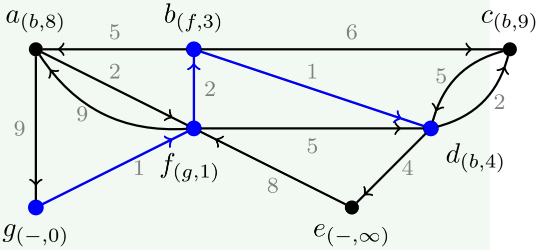
Dijkstra's Algorithm for digraphs – example

Example

- Step 2. *current vertex*$= b$, $F = \{a, c, d, e\}$, out-neighbors of b: a, c, d

$$\omega(b) + \omega(ba) = 3 + 5 = 8 < 10,$$
$$\omega(b) + \omega(bc) = 3 + 6 = 9 < \infty,$$
$$\omega(b) + \omega(bd) = 3 + 1 = 4 < 6.$$

- Step 3. *current vertex*$= d$, $F = \{a, c, e\}$



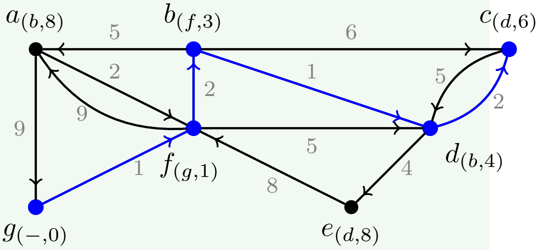
	a	b	c	d	e	f
	∞	∞	∞	∞	∞	∞
$g(0)$	∞	∞	∞	∞	∞	$(g, 1)$
$f(1)$	$(f, 10)$	$(f, 3)$	∞	$(f, 6)$	∞	$(g, 1)$
$b(3)$	$(b, 8)$	$(f, 3)$	$(b, 9)$	$(b, 4)$	∞	$(g, 1)$

Dijkstra's Algorithm for digraphs – example

Example

- Step 2. *current vertex*$= d$, $F = \{a, c, e\}$, out-neighbors of b: c, e

$$\omega(d) + \omega(dc) = 4 + 2 = 6 < 9,$$
$$\omega(d) + \omega(de) = 4 + 4 = 8 < \infty$$



- Step 3. *current vertex*$= c$
- Step 4. we have reached *End*
- Step 5. $g \to f \to b \to d \to c$, weight 6

	a	b	c	d	e	f
	∞	∞	∞	∞	∞	∞
$g(0)$	∞	∞	∞	∞	∞	$(g,1)$
$f(1)$	$(f,10)$	$(f,3)$	∞	$(f,6)$	∞	$(g,1)$
$b(3)$	$(b,8)$	$(f,3)$	$(b,9)$	$(b,4)$	∞	$(g,1)$
$d(4)$	$(b,8)$	$(f,3)$	$(d,6)$	$(b,4)$	$(d,8)$	$(g,1)$

Remark

- It is possible for a path not to exist from one vertex to another based upon the direction of the arcs
- e.g. if a is the head of all arcs, then no path originating at a could exist
- In such a case Dijkstra's Algorithm would halt and note that a shortest path could not be found

Paths and spanning trees

- Dijkstra's Algorithm
- Project scheduling
- Critical path
- Trees
- Spanning trees

Definitions

Definition

Consider a project containing multiple parts of steps.

- *Task*: a required step of a project that cannot be broken into smaller pieces. Labeled with lowercase letters
- *Processor*: the unit (such as a person) that completes a task. Labeled as P_1, P_2, etc. At any time a processor will either be idle or busy performing a task
- At any stage of a project, a task can be in one of four states:
  - *eligible*: the task can be performed
  - *ineligible*: the task cannot be performed
  - *in execution*: the task is currently being performed
  - *completed*: the task has been completed

  A task is eligible when all the tasks it relies upon are completed

Definitions

Definition

Consider a project containing multiple parts of steps.

- *Processing time* of a task: the time it takes to complete the task, denoted by $\mathrm{pt}(v)$ for task v
- *Precedence relationship*: task b relies on the completion of task a before it can be eligible, we call this a
- *Finishing time* of a schedule is the total time used in that schedule
- *Optimal time* of a project is the minimum finishing time among all possible schedules, denoted OPT.

Tasks and schedules – example

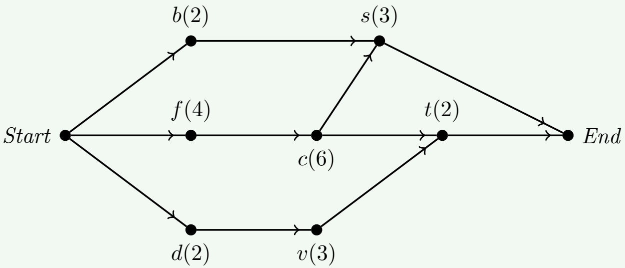
Example

- Party planning

Task	Vertex Name	Processing Time	Precedence Relationships
Buy Food	f	40	
Buy Drinks	b	20	
Dust	d	20	
Vacuum	v	30	d
Cook Food	c	60	f
Set Out Drinks	s	30	b, c
Set Table	t	20	v, c

Tasks and schedules – example

Example



- It is customary to include a vertex to represent the start and end of a project, as well as lay out vertices to avoid edge crossings whenever possible
- The processing times are shown in parentheses next to the vertex labels
- Edges: precedence relationships

Priority List Model

- Once a digraph has been created, the next step is to determine which processor (or person) should complete each task
- This may be easy in a project with only a few tasks, or if the interplay between tasks is not complex
- As complexity grows, we will need a procedure for assigning tasks
- *Priority List Model*: tasks must be assigned to processors according to their order in the *priority list* while precedence relationships, which are displayed in the digraph, are used to determine eligibility of a given task.

Priority List Model – example

Example

- Continuing from the previous example
- Take priority list: $b - d - t - v - s - f - c$
- Consider two processors
- Each step represents a moment in time where a decision must be made

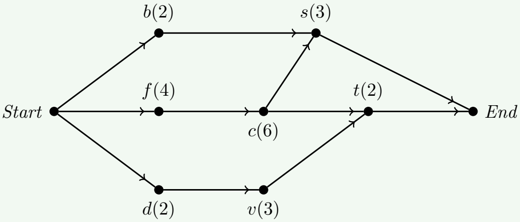
	10	20	30	40	50	60	70	80	90	100	110	120	130	140	150
P_1	b	b													
P_2	d	d													

- Step 1. $T = 0$ The first item in the priority list is b, since b does not rely on any other task, assign it to P_1. The next item d is also eligible. Assign d to P_2.

Priority List Model – example

Example

Priority list: $b - d - t - v - s - f - c$



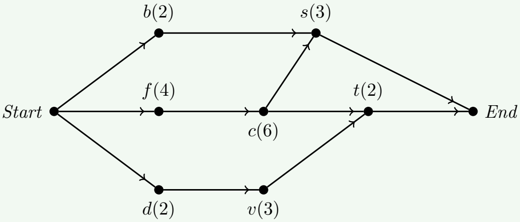
	10	20	30	40	50	60	70	80	90	100	110	120	130	140	150
P_1	b	b	v	v	v										
P_2	d	d	f	f	f	f									

- Step 2. $T = 20$. The next point at which a processor is free to pick up a task is at 20 minutes. Next task on the list is t, but ineligible. v is eligible, f is eligible

Priority List Model – example

Example

Priority list: $b - d - t - v - s - f - c$



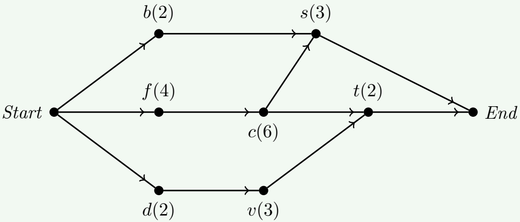
	10	20	30	40	50	60	70	80	90	100	110	120	130	140	150
P_1	b	b	v	v	v	$*$									
P_2	d	d	f	f	f	f									

- Step 3. $T = 60$. At 60 minutes, Processor 1 is ready for a new task. All tasks remaining require f to be complete. P_1 will remain idle until f is complete

Priority List Model – example

Example

Priority list: $b - d - t - v - s - f - c$



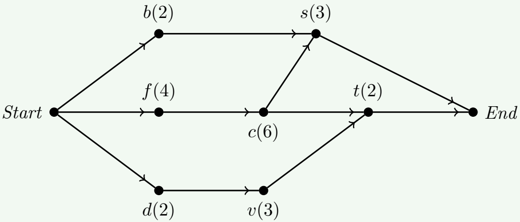
	10	20	30	40	50	60	70	80	90	100	110	120	130	140	150
P_1	b	b	v	v	v	$*$	c	c	c	c	c	c			
P_2	d	d	f	f	f	f	$*$	$*$	$*$	$*$	$*$	$*$			

- Step 4. $T = 70$. At 70 minutes, both processors are ready to take up a new task. Only eligible task is c. By convention, we assign the task to the lower indexed processor. The other processor remains idle

Priority List Model – example

Example

Priority list: $b - d - t - v - s - f - c$



	10	20	30	40	50	60	70	80	90	100	110	120	130	140	150
P_1	b	b	v	v	v	$*$	c	c	c	c	c	c	t	t	$*$
P_2	d	d	f	f	f	f	$*$	$*$	$*$	$*$	$*$	$*$	s	s	s

- Step 5. $T = 130$. Once c is complete, we can assign t to P_1 and s to P_2

The priority list $b - d - t - v - s - f - c$ yields a finishing time of 150 minutes using two processors.

Remarks

- The schedule we obtained contains a large amount of idle time, 8 hours in total
- Although some idle time may be unavoidable, its presence should indicate that more investigation is warranted.
- The priority list given did not seem to have any connection with the digraph (in fact, it was generated randomly)
- A better approach would be to use information from the digraph to obtain a good priority list.

Paths and spanning trees

- Dijkstra's Algorithm
- Project scheduling
- Critical path
- Trees
- Spanning trees

Critical path

- *Critical path*: the path with the highest total time out of all paths that begin at vertex *Start* and finish at *End*
- This path is of interest because it easily identifies restrictions on the completion time of a project
- In addition, it indicates which tasks should be prioritized.
- To find the critical path, we first need to find the *critical times* of all vertices in the graph.

Critical time

Definition

The *critical time* $\mathrm{ct}[x]$ of a vertex x is defined as the sum of the processing time of x and the maximum of the critical times for all vertices y for which xy is an arc

$$\mathrm{ct}[x] = \mathrm{pt}(x) + \max\{\, \mathrm{ct}[y] \;\mid\; xy \text{ is an arc} \,\}$$

Note

In the definition, y is an out-neighbor of x

Critical Path Algorithm

Input: Project digraph G with processing times given

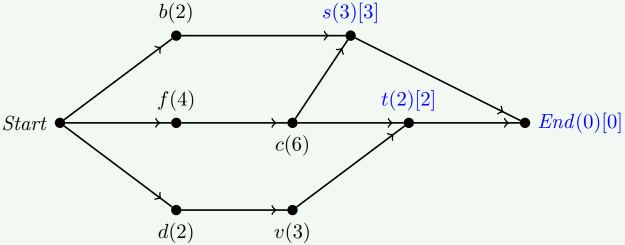
Steps:

1. Label the vertex *End* with $\mathrm{pt}(\textit{End}) = 0$ and $\mathrm{ct}[\textit{End}] = 0$. For any vertex x with an arc to *End*, define $\mathrm{ct}[x] = \mathrm{pt}(x)$.
2. Travel the arcs in reverse order. When a new vertex is encountered, calculate its critical time.
3. Once all critical times have been obtained, find the path from *Start* to *End* where if more than one arc exists out of a vertex, take the arc to the neighbor vertex of largest critical time.
4. Create a priority list by ordering vertices by decreasing critical time

Output: Critical path and critical path priority list

Example

- Continuing from the previous example
- We will use brackets for the critical times, distinguishing them from the processing times

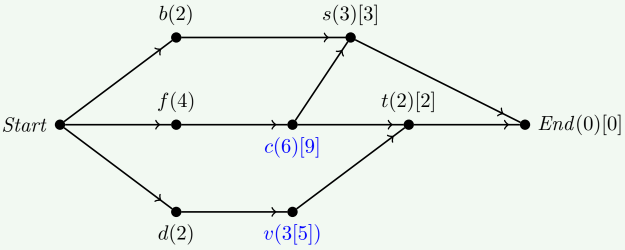


- Step 1. Label *End* with critical times 0. Since s and t have arcs to *End*, set

$$\mathrm{ct}[s] = \mathrm{pt}(s) = 3, \quad \mathrm{ct}[t] = \mathrm{pt}(t) = 2$$

Critical Path Algorithm – example

Example



- Step 2. As v has a single arc to t

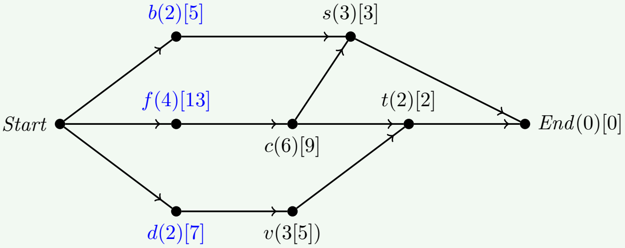
$$\mathrm{ct}[v] = \mathrm{pt}(v) + \mathrm{ct}[t] = 3 + 2 = 5$$

c has an arc to both s and t. $\mathrm{ct}[s] > \mathrm{ct}[t]$

$$\mathrm{ct}[c] = \mathrm{pt}(c) + \mathrm{ct}[s] = 6 + 3 = 9$$

Critical Path Algorithm – example

Example

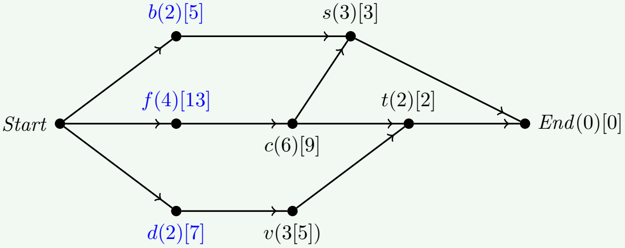


- Step 3. The remaining vertices each have a single arc to previously considered vertices

$$\text{ct}[b] = \text{pt}(b) + \text{ct}[s] = 2 + 3 = 5$$
$$\text{ct}[f] = \text{pt}(f) + \text{ct}[c] = 4 + 9 = 13$$
$$\text{ct}[d] = \text{pt}(d) + \text{ct}[v] = 2 + 5 = 7$$

Critical Path Algorithm – example

Example

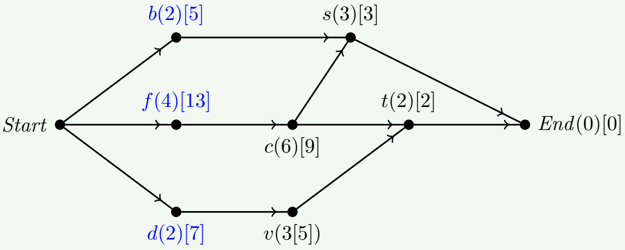


- Step 4. Label the processing time of *Start* as 0. f is the out-neighbor with the largest critical time

$$ct(\textit{Start}) = 0 + 13 = 13$$

Critical Path Algorithm – example

Example



- Step 5. Follow the path from *Start* to *End* where the vertices are chosen based on the largest critical times. This gives the path

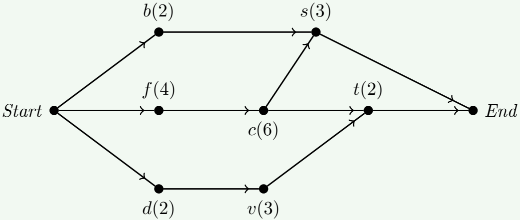
$$\textit{Start} \to f \to c \to s \to \textit{End}$$

of total time 130 Ordering the vertices in decreasing order of critical times gives the critical path priority list

Priority List Model for project scheduling – example

Example

Now we use the critical path priority list $f - c - d - b - v - s - t$ to find a new schedule for the tasks



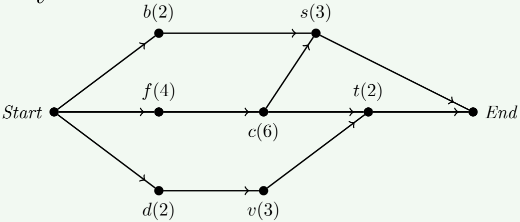
	10	20	30	40	50	60	70	80	90	100	110	120	130
P_1	f	f	f	f									
P_2	d	d											

- Step 1. $T = 0$. Since f is the first item in the list, assigne it to P_1. c is not eligible. Assign d to P_2

Priority List Model for project scheduling – example

Example

$f - c - d - b - v - s - t$



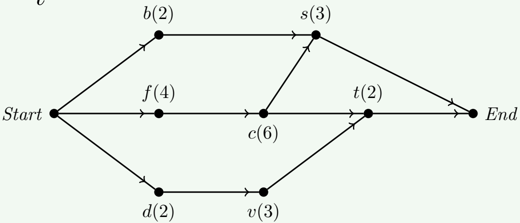
	10	20	30	40	50	60	70	80	90	100	110	120	130
P_1	f	f	f	f									
P_2	d	d	b	b									

- Step 2. $T = 20$. P_2 can be assigned a new task. Assign b, the next eligible task to P_2

Priority List Model for project scheduling – example

Example

$f - c - d - b - v - s - t$



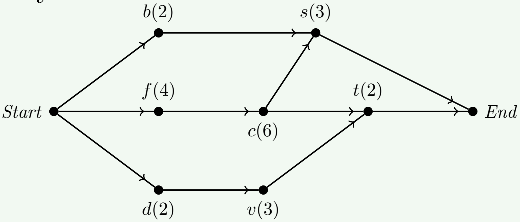
	10	20	30	40	50	60	70	80	90	100	110	120	130
P_1	f	f	f	f	c	c	c	c	c	c			
P_2	d	d	b	b	v	v	v						

- Step 3. $T = 40$. c is eligible. The next eligible task is v

Priority List Model for project scheduling – example

Example

$f - c - d - b - v - s - t$



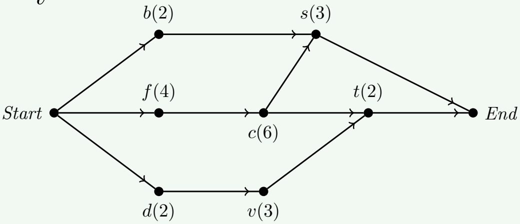
	10	20	30	40	50	60	70	80	90	100	110	120	130
P_1	f	f	f	f	c	c	c	c	c	c			
P_2	d	d	b	b	v	v	v	$*$	$*$	$*$			

- Step 4. $T = 70$. All remaining taks are ineligible since they reply on the completion of c. P_2 remians idle.

Priority List Model for project scheduling – example

Example

$f - c - d - b - v - s - t$



	10	20	30	40	50	60	70	80	90	100	110	120	130
P_1	f	f	f	f	c	c	c	c	c	c	s	s	s
P_2	d	d	b	b	v	v	v	$*$	$*$	$*$	t	t	$*$

- Step 5. $T = 100$. s and t are now eligible
- Finishing time: 130 minutes, 4 hours of idle time

Remarks

- Both schedules contained some idle time, though the one utilizing the critical path priority list had half that of the initial example – This is in part because items on the critical path were prioritized over less important tasks
- The schedule above must be optimal since its finishing time is equal to the critical time of Start
- In general, the critical path priority list results in a very good, though not always optimal, schedule

Optimal schedule

- The optimal time of a schedule is no less than the critical time of *Start*

$$OPT \geq \mathrm{ct}[\textit{Start}]$$

- Calculate the sum of all processing times of all tasks. The optimal time is no less than this sum divided by the total number of processors used

$$OPT \geq \frac{\sum_v \mathrm{pt}(v)}{\text{number of processors}}$$

Optimal schedule – example

Example

With our running example, sum of all processing times is 220 minutes

- Using two processors $OPT \geq \frac{220}{2} = 110$
- Using three processors $OPT \geq \frac{220}{3} \approx 73$
- $ct(\mathit{Start}) = 130$. Neither of the above two calculations provides additional insight into the optimal schedule.

Remark

The calculations also show that 2 processors are sufficient, no need more processors.

Paths and spanning trees

- Dijkstra's Algorithm
- Project scheduling
- Critical path
- Trees
- Spanning trees

Definition

Definition

A graph G is

- *acyclic* if there are no cycles or circuits in the graph
- a *network* if it is connected
- a *tree* if it is an acyclic network, i.e. acyclic and connected
- a *forest* if it is an acyclic graph

A vertex of degree 1 is called a *leaf*.

Trees – example

Example

Game

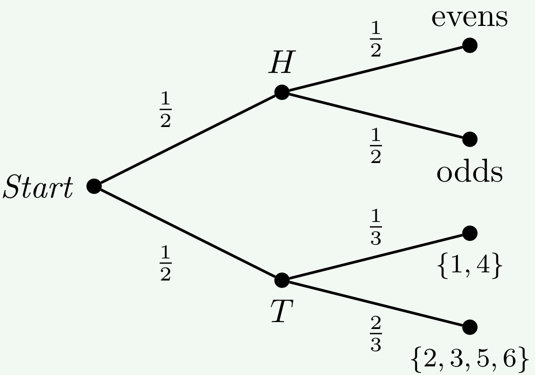
- Adam flips a coin, Jano rolls a die
- Adam gets heads, Jano rolls an even number $\rightarrow$ Jano wins 2 Euros
- Adam gets heads, Jano rolls an odd number $\rightarrow$ Adam wins 3 Euros
- Adam gets tails, Jano rolls 1 or 4 $\rightarrow$ Jano wins 5 Euros
- Adam gets heads, Jano rolls $2, 3, 5$, or 6 $\rightarrow$ Adam wins 2 Euros

A probability tree – vertices representing possible outcomes, edges labeled with the probability of the outcome

Trees – example

Example

- Adam gets heads, Jano rolls an even number $\rightarrow$ Jano wins 2 Euros
- Adam gets heads, Jano rolls an odd number $\rightarrow$ Adam wins 3 Euros
- Adam gets tails, Jano rolls 1 or 4 $\rightarrow$ Jano wins 5 Euros
- Adam gets heads, Jano rolls $2, 3, 5,$ or 6 $\rightarrow$ Adam wins 2 Euros



$$\frac{1}{2} \cdot \frac{1}{2} = \frac{1}{4}$$

$$\frac{1}{2} \cdot \frac{1}{2} = \frac{1}{4}$$

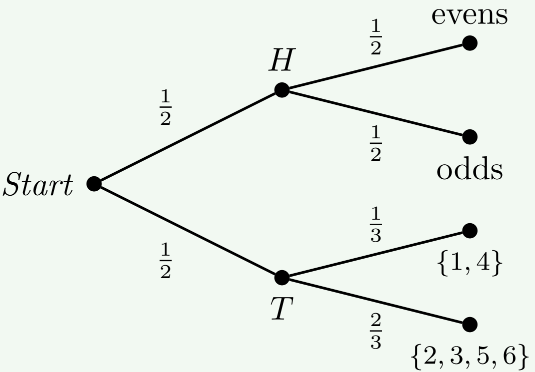
$$\frac{1}{2} \cdot \frac{1}{3} = \frac{1}{6}$$

$$\frac{1}{2} \cdot \frac{2}{3} = \frac{1}{3}$$

Trees – example

Example

- The probability Jano wins 5 Euros (tails and 1 or 4) is $\frac{\S}{6}$
- The probability that Jano wins any money is $\frac{1}{6} + \frac{1}{4} = \frac{5}{12}$



$\frac{1}{2} \cdot \frac{1}{2} = \frac{1}{4}$

$\frac{1}{2} \cdot \frac{1}{2} = \frac{1}{4}$

$\frac{1}{2} \cdot \frac{1}{3} = \frac{1}{6}$

$\frac{1}{2} \cdot \frac{2}{3} = \frac{1}{3}$

Trees – example

Example

- Trees can be used to store information for quick access
- Consider the following sequence of numbers

$$4,\ 2,\ 7,\ 10,\ 1,\ 3, 5$$

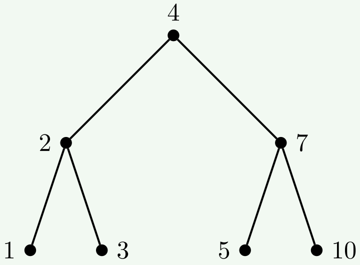
- We can form a tree by creating a vertex for each number in the list
- As we move from one entry in the list to the next, we place an item below and to the left if it is less than the previously viewed vertex and below and to the right if it is greater
- If we add the restriction that no vertex can have more than two edges coming down from it, then we are forming a binary tree.

Trees – example

Example

- As we move from one entry in the list to the next, we place an item below and to the left if it is less than the previously viewed vertex and below and to the right if it is greater
- Restriction: no vertex can have more than two edges coming down from it

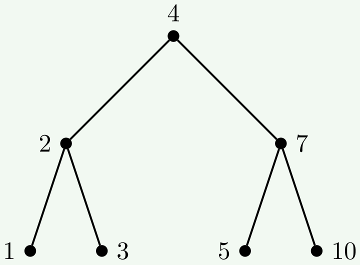
$$4,\ 2,\ 7,\ 10,\ 1,\ 3,\ 5$$



Trees – example

Example

- If we want to search for an item, then we only need to make comparisons with at most half of the items in the list
- Find item 5, first compare it to the vertex at the top of the tree $\rightarrow$ move along the edge to the right of 4 $\rightarrow$ compare to 7 $\rightarrow$ move along the edge to the left of 7
- Only two comparisons



Properties of trees

1. For every $n \geq 1$, any tree with n vertices has $n - 1$ edges
2. For any tree with $n \geq 1$ vertices, the sum of the degrees is $2n - 2$
3. Every tree with at least two vertices contains at least two leaves
4. Any network on n vertices with $n - 1$ edges must be a tree
5. For any two vertices in a tree, there is a unique path between them
6. The removal of any edge of a tree will disconnect the graph

Note

- $1 \Rightarrow 2$
- By counting the degrees of vertices, $2 \Rightarrow 3$

Paths and spanning trees

- Dijkstra's Algorithm
- Project scheduling
- Critical path
- Trees
- Spanning trees

Spanning Tree

Definition

- A *subgraph* H of a graph G is a graph s.t. $V(H) \subseteq V(G)$, $E(H) \subseteq E(G)$
- H is a *spanning subgraph* if $V(H) = V(G)$
- *Spanning tree*: is a spanning subgraph that is also a tree

Note

- If an edge appears in a subgraph, then both endpoints must also be included in the subgraph
- If a vertex appears in a subgraph, any number of its incident edges may be included

Kruskal's Algorithm

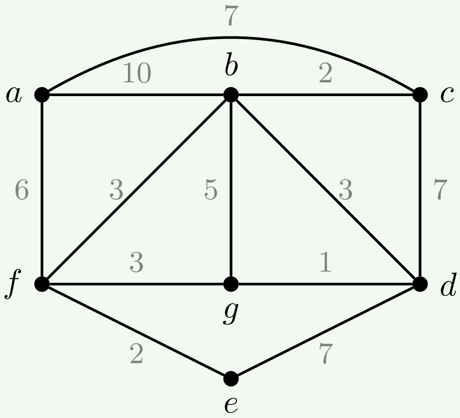
- First published in 1956 by Joseph Kruskal, an American mathematician
- Input: weighted connected graph $G = (V, E, \omega)$
- Steps
  1. Choose the edge of least weight from not highlighted edges. Highlight it and add it to $T = (V, E', \omega')$
  2. Repeat step 1 as long as no circuit is created.
- Output: minimum spanning tree (MST) T of G

Note

- If there are more than two edges with the least weight, randomly choose one
- At each step of the algorithm we are building a forest subgraph that will eventually result in a spanning tree

Kruskal's Algorithm – example

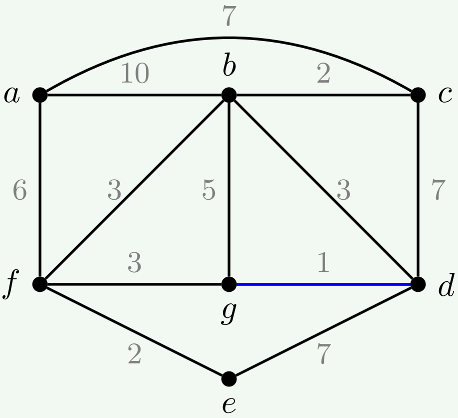
Example



- Step 1. edge with the least weight: gd

Kruskal's Algorithm – example

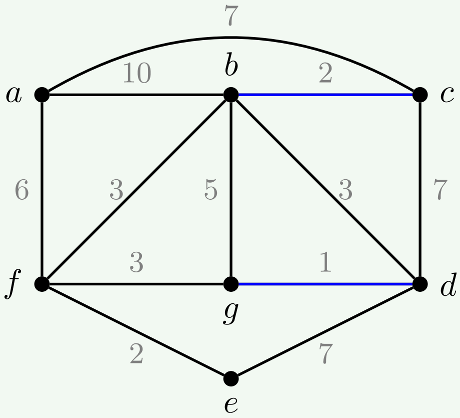
Example



- Step 1. two choices: ef and bc, let's choose bc

Kruskal's Algorithm – example

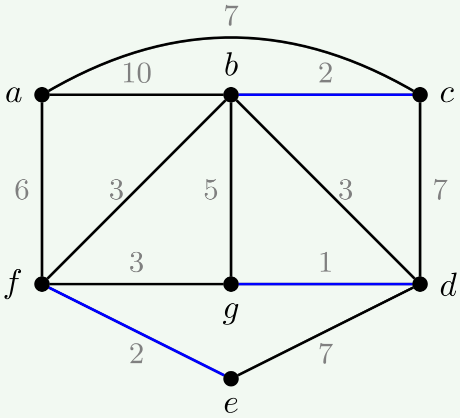
Example



- Step 1. the other edge with weight 2 is still a valid choice. Highlight ef

Kruskal's Algorithm – example

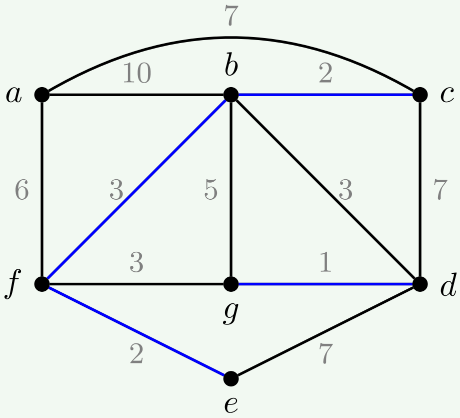
Example



- Step 1. the next smallest edge weight is 3. We randomly pick bf

Kruskal's Algorithm – example

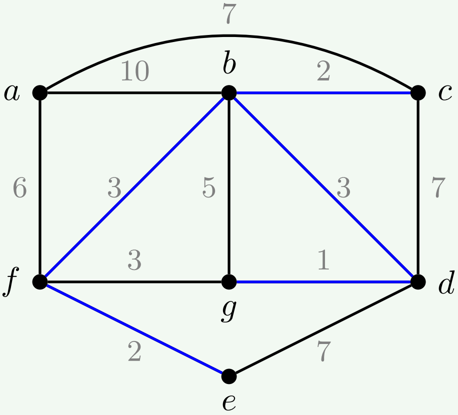
Example



- Step 1. both of the other edges of weight 3 are still available. We choose bd

Kruskal's Algorithm – example

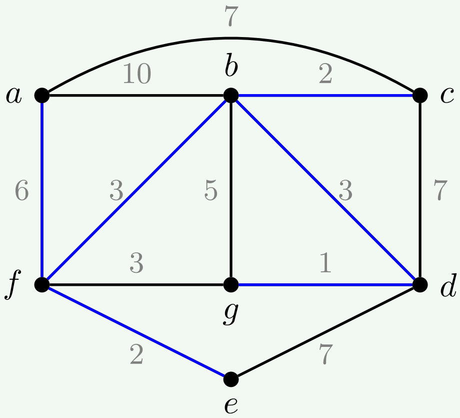
Example



- Step 1. cannot choose fg. The next smallest edge weight is 5. But we cannot choose bg. Next available edge is af of weight 6

Kruskal's Algorithm – example

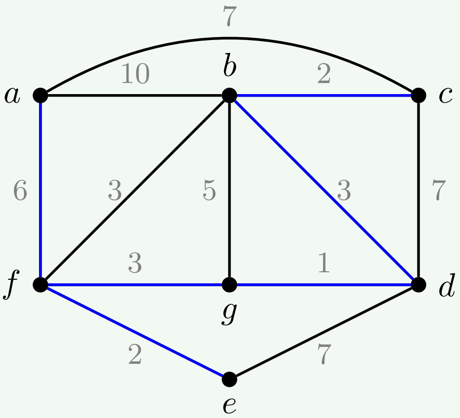
Example

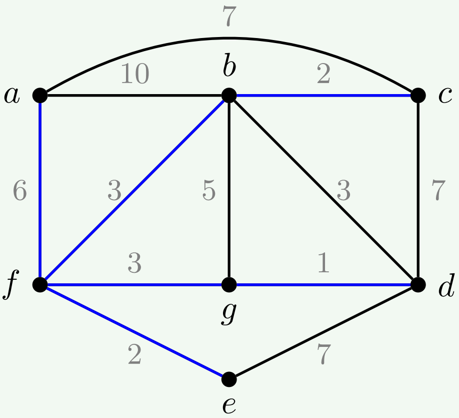


- Step 2. Now we have a tree containing all vertices, we can stop. With 7 vertices and 6 edges, we know we have a tree. MST weight 17

Kruskal's Algorithm – example

Example





- When we were choosing edge with weight 3, we chose ef
- Two other spanning trees exist

Kruskal's Algorithm – remarks

- When we skipped over an edge, e.g. bg of weight 5, we did so because including it would create a cycle
- This means a path between the endpoints of that edge, e.g. b and g, must already exist and the other edges along that path must each be of weight no greater than the edge we skip over
- In a way, if you think of finding a spanning tree as breaking cycles, then the largest edge on that cycle should never be chosen

Prim's Algorithm

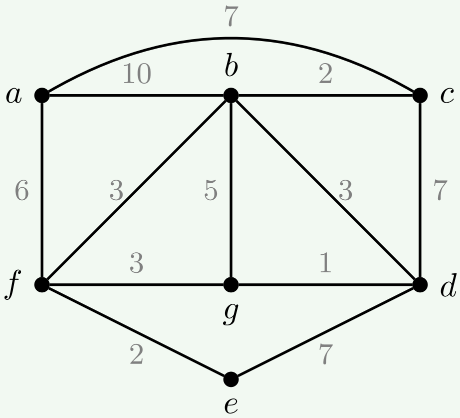
- Named after Robert C. Prim, American mathematician and computer scientist, published in 1957
- Originally discovered by Vojtěch Jarnik, Czech mathematician, in 1930
- Root: a clear starting point for the tree

Prim's Algorithm

- Input: Weighted connected graph $G = (V, E)$
- Steps
  1. Let v be the root. If no root is specified, choose a vertex at random. Highlight it and add it to $T = (V', E')$
  2. Among all edges incident to v, choose the one of minimum weight. Highlight it. Add the edge and its other endpoint to T.
  3. Let S be the set of all edges with exactly one endpoint from $V(T)$. Choose the edge of minimum weight from S. Add it and its other endpoint to T
  4. Repeat step 3 until T contains all vertices of G
- Output: rooted MST T of G

Prim's Algorithm – example

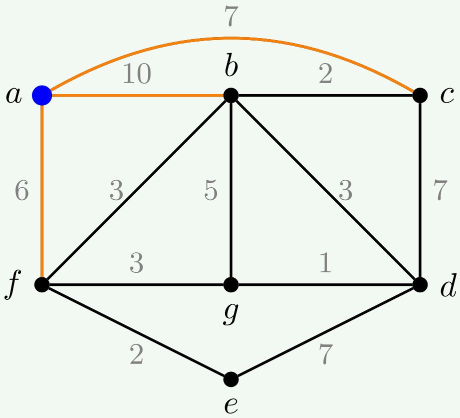
Example



- Same example as before
- Step 1. Choose a as the starting vertex

Prim's Algorithm – example

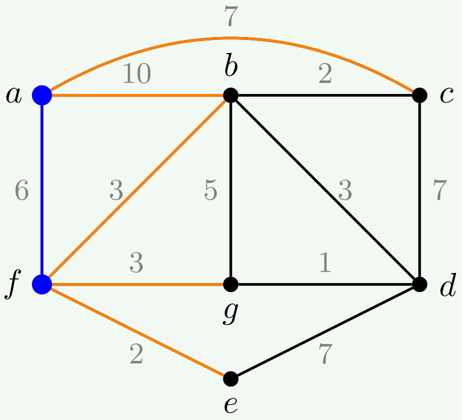
Example



- Step 2. among the edges incident to a, af, ab, ac, the edge of the least weight is af

Prim's Algorithm – example

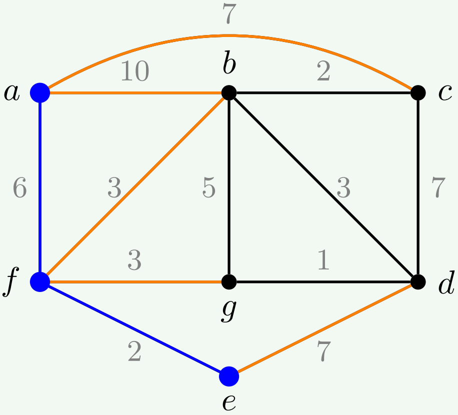
Example



- Step 3. S consists edges with one endpoint a or f. The edge with the minimum weight is ef

Prim's Algorithm – example

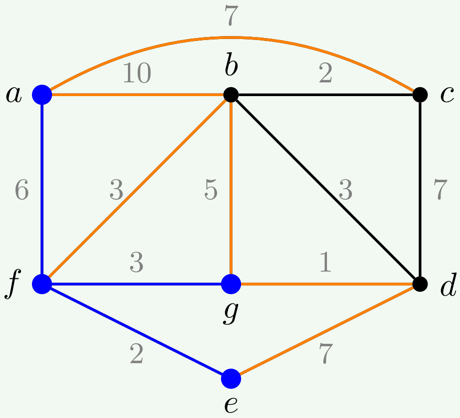
Example



- Step 3. S consists edges with one endpoint a, e or f. The edges with the minimum weight fb or fg. Let us choose fg

Prim's Algorithm – example

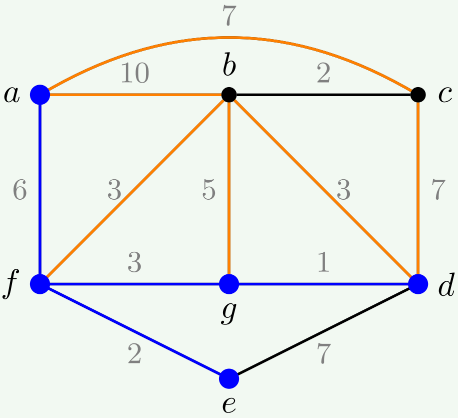
Example



- Step 3. The next edge to add to the tree is dg

Prim's Algorithm – example

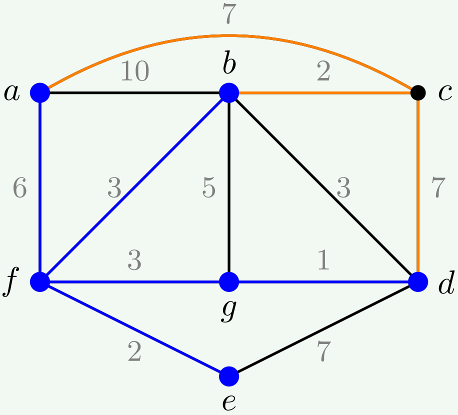
Example



- Step 3. There are two possible minimum weight edges, bf or bd. We choose bf

Prim's Algorithm – example

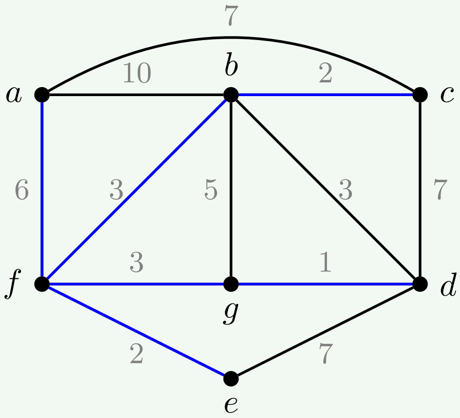
Example



- Step 3. Now the only edges to consider are those with one endpoint of c since this is the only vertex not part of our tree. The edge of minimum weight is bc

Prim's Algorithm – example

Example



- We have obtained a MST of weight 17