NUMERICAL METHODS

$$\frac{\partial v}{\partial t} + V \cdot \nabla v = \nabla \cdot (k \nabla v) + g(v)$$

$$(\lambda + \mu)\nabla(\nabla \cdot u) + \mu \nabla^2 u = \alpha(3\lambda + 2\mu)\nabla T - \rho b$$

$$U^{n+1} = U^n + \Delta t\, f(U^n)$$

Lecture 9

$$\rho\left(\frac{\partial u}{\partial t} + V \cdot \nabla u\right) = -\frac{\partial p}{\partial x} + \frac{\partial \tau_{xx}}{\partial x} + \frac{\partial \tau_{yx}}{\partial y} + \frac{\partial \tau_{zx}}{\partial z} + \rho f_x$$

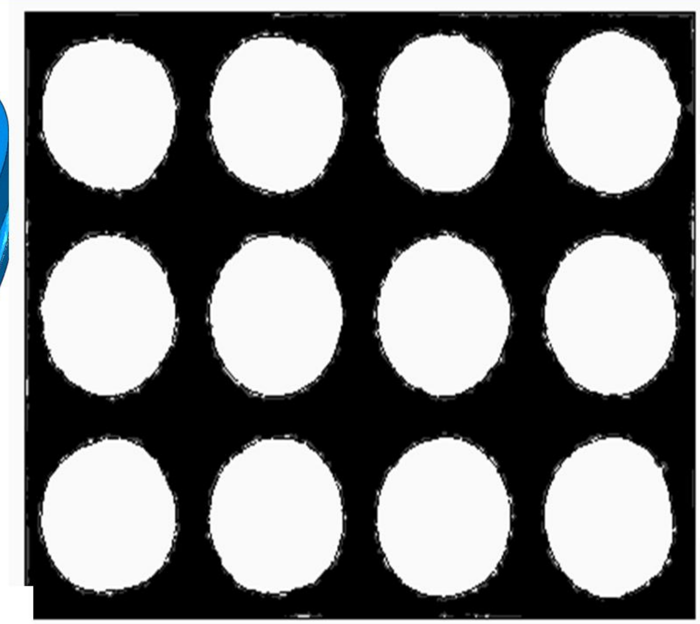
$$\nabla^2 u = f$$

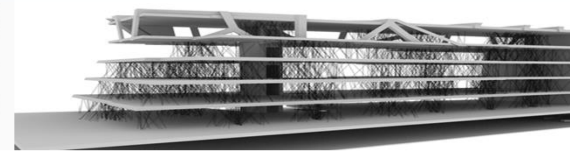
Dr. P V Ramana

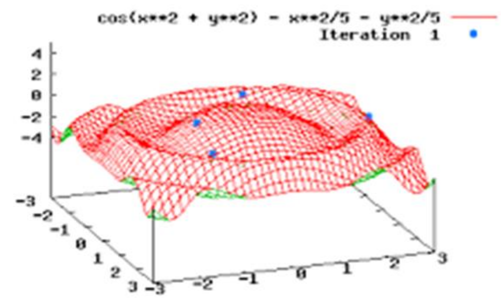
Optimization

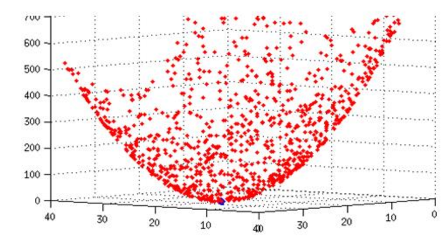
The stationary points of a 1-D function (Maxima & Minima)

- **Consider a function of one variable f(x). Maxima (maximum point) or minima, f ’(x) = 0 *i.e.,* gradient is zero, hence these are called stationary points.**









A schematic view of modeling/optimization process

Real-world problem

assumptions, abstraction,data, simplifications

Mathematical model

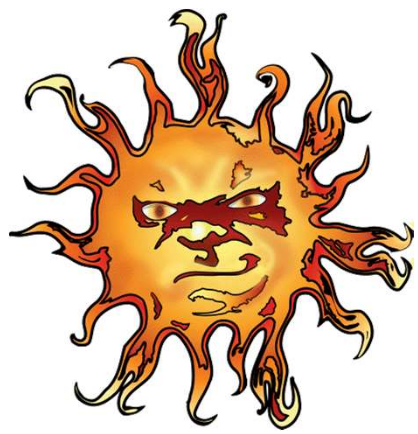
optimization algorithm

Solution to model

interpretation

Solution to real-world problem

makes sense? change the model, assumptions?

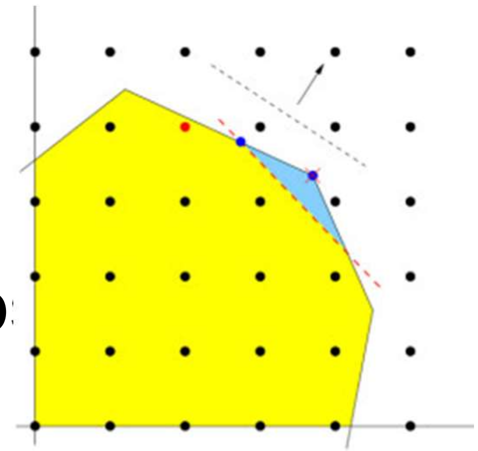


dizziness, headache and fainting. It can usually be treated with rest, a cool environm

What is a model?

Model: **A schematic description of a system, theory, or phenomenon that accounts for its known or inferred properties and maybe used for further study of its characteristics.**

- Mathematical models
  - are abstract models
  - describe the mathematical relationships among elements in a system
- In NM, mathematical models dealing with discrete optimization



Mathematical models in Optimization

- The general form of an *optimization model*:

  min or *max* $f(x_1,\ldots,x_n)$ (objective function)

  subject to $g_i(x_1,\ldots,x_n) \geq 0$ (functional constrains)

  $x_1,\ldots,x_n \in S$ (set constraints)

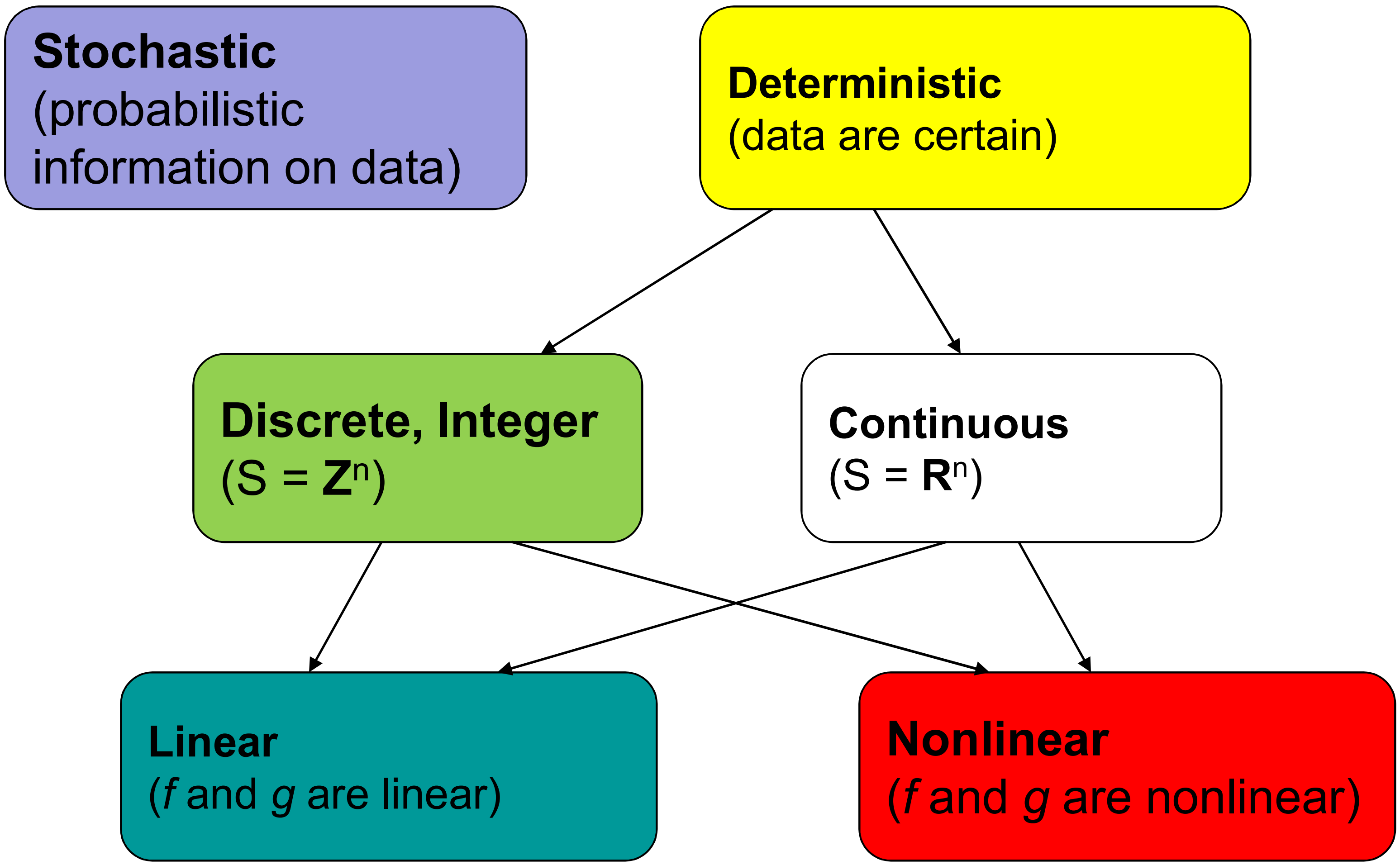
- $x_1,\ldots,x_n$ are called decision variables

In words,

the goal is to find $x_1,\ldots,x_n$ that satisfy the constraints;

achieve min (max) objective function value.

Types of Optimization Models



What is Discrete Optimization?

Discrete Optimization

is a field of applied mathematics,

combining techniques from

- combinatorics and graph theory,
- linear programming,
- theory of algorithms, to solve optimization problems over discrete structures.

1. **TRAVELING SALESMAN PROBLEM (TSP)**
2. **JOB SCHEDULING (JS)**
3. **THE SHORTEST PATH PROBLEM (SPP)**

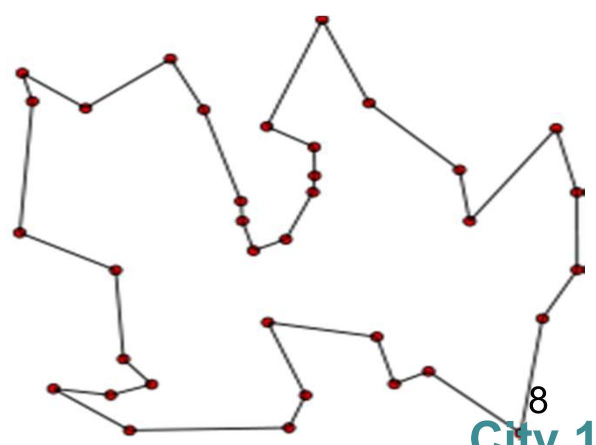
Examples of Discrete Optimization Models:

Traveling Salesman Problem (TSP)

- There are n cities. The salesman
  - starts his tour from City 1,
  - visits each of the cities exactly once,
  - and returns to City 1.

For each pair of cities i,j there is a cost c_{ij} associated with traveling from City i to City j .

- Goal: Find a minimum-cost tour.

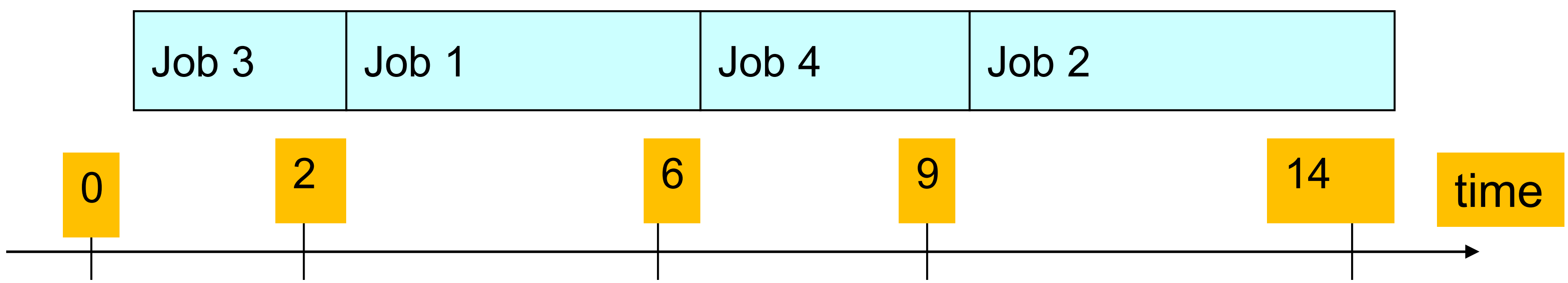


Examples of Discrete Optimization Models: Job Scheduling

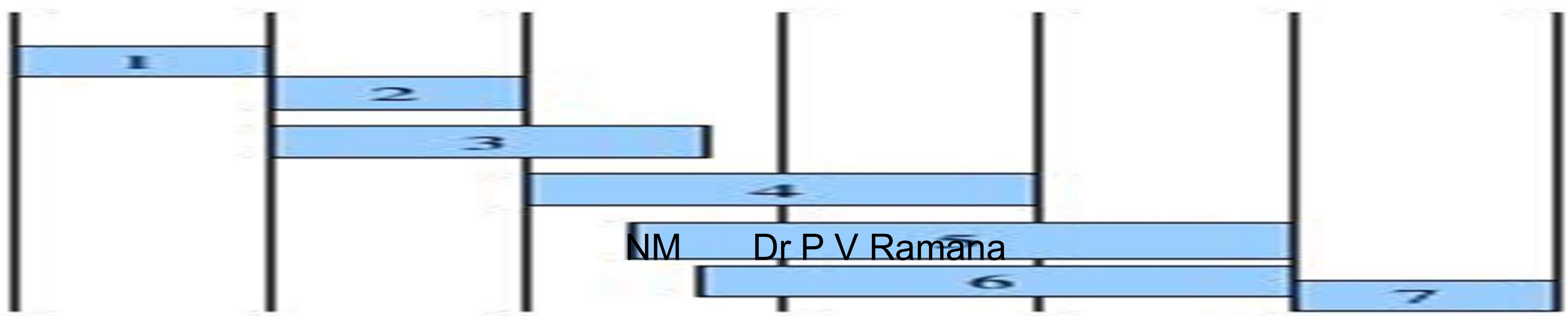
- There are 4 jobs that should be processed on the same machine. (*Can't be processed simultaneously*).

  Job *k* has processing time p_k .

  Here is an example of a possible schedule:

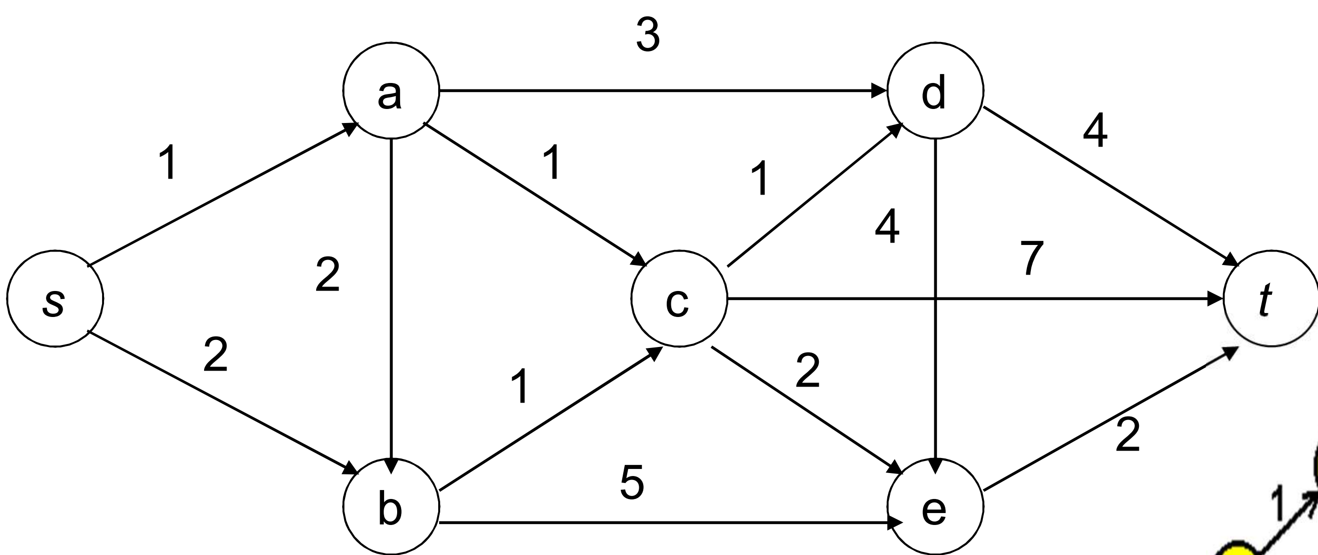


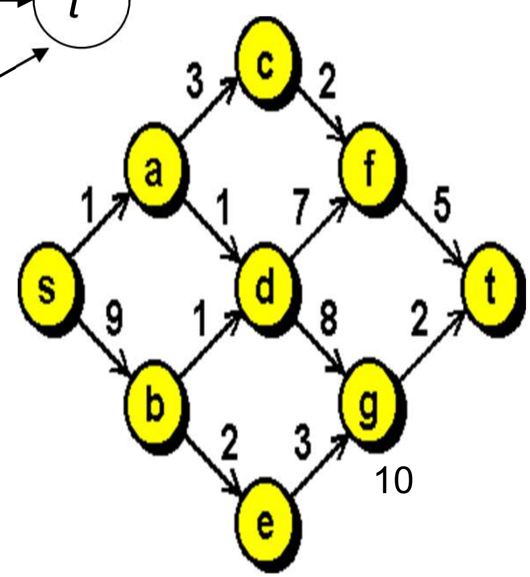
- Goal: Find a schedule which minimizes, average completion time of the jobs.



Examples of Discrete Optimization Models: Shortest Path Problem

- In a network, have distances on arcs; source node *s* and sink node *t* .





- Goal: Find a shortest path from the source to the sink.

Optimization Algorithms

- **Many real-world problems involve maximizing or minimizing a value:**

✓**How can a car manufacturer get the most parts out of a piece of sheet metal?**
✓**How can a moving company fit the most furniture into a truck of a certain size?**
✓**How can the phone company route calls to get the best use of its lines and connections?**
✓**How can a university schedule its classes to make the best use of classrooms without conflicts?**

Optimal vs. Approximate Solutions

Often, one can make a choice:

- **Do want the guaranteed optimal solution to the problem, even though it might take a lot of work/time to find?**

Do want to spend less time/work and find an approximate solution (near optimal)?

Approximate Solutions
Greedy Algorithms

- **Approximates optimal solution**
- **May or may not find optimal solution**
- **Provides “quick and dirty” estimates**
- **A greedy algorithm makes a series of “short-sighted” decisions and does not “look ahead”**
- **Spends less time**

A Greedy Algorithm

Consider the weight and value of some foreign coins:

foo	**$6.00**	**500 grams**
bar	**$4.00**	**450 grams**
baz	**$3.00**	**410 grams**
qux	**$0.50**	**300 grams**

If one can only fit 860 grams in our pockets...

A greedy algorithm would choose:

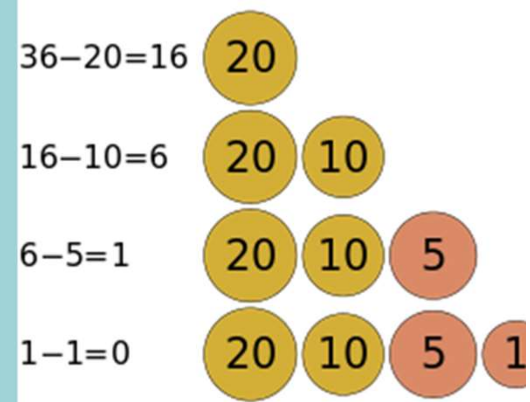
1 foo	**500 grams**	**= $6.00**
1 qux	**300 grams**	**= $0.50**

Total of $6.50

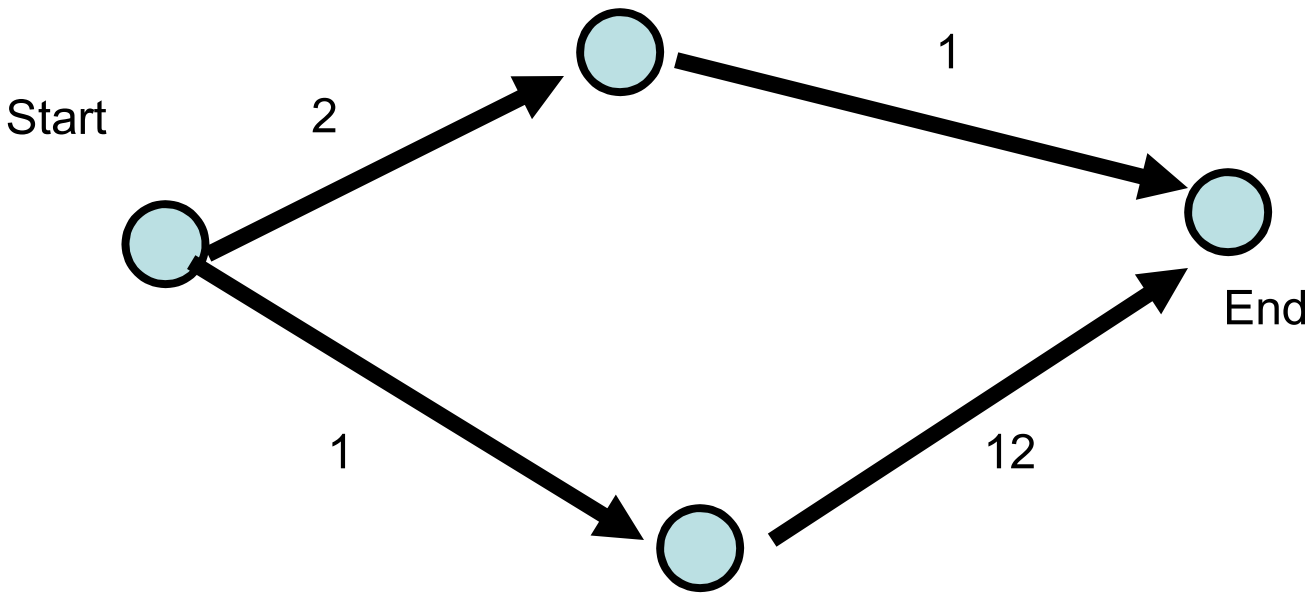
Optimal solution is:

1 bar	**450 grams = $4.00**
1 baz	**410 grams = $3.00**

Total of $7.00

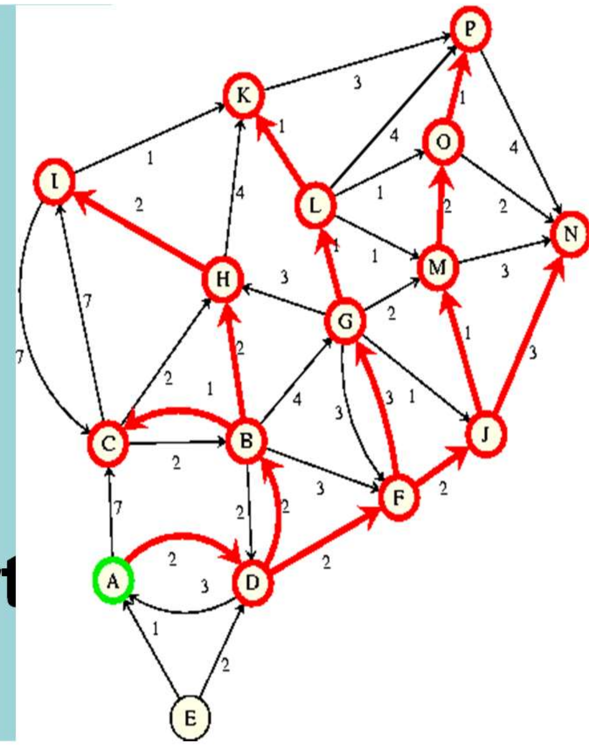


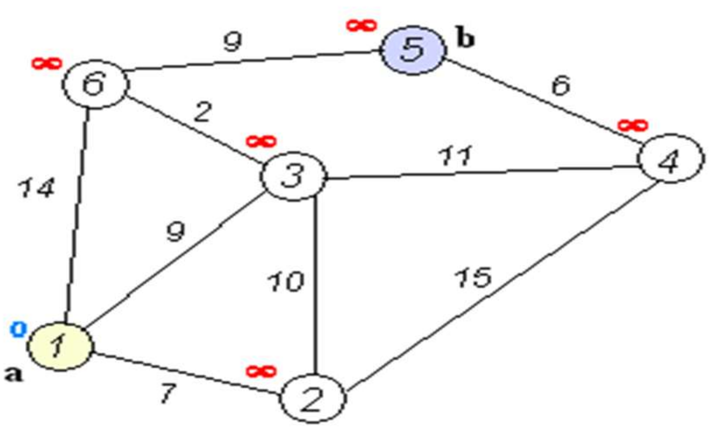
Short-Sighted Decisions



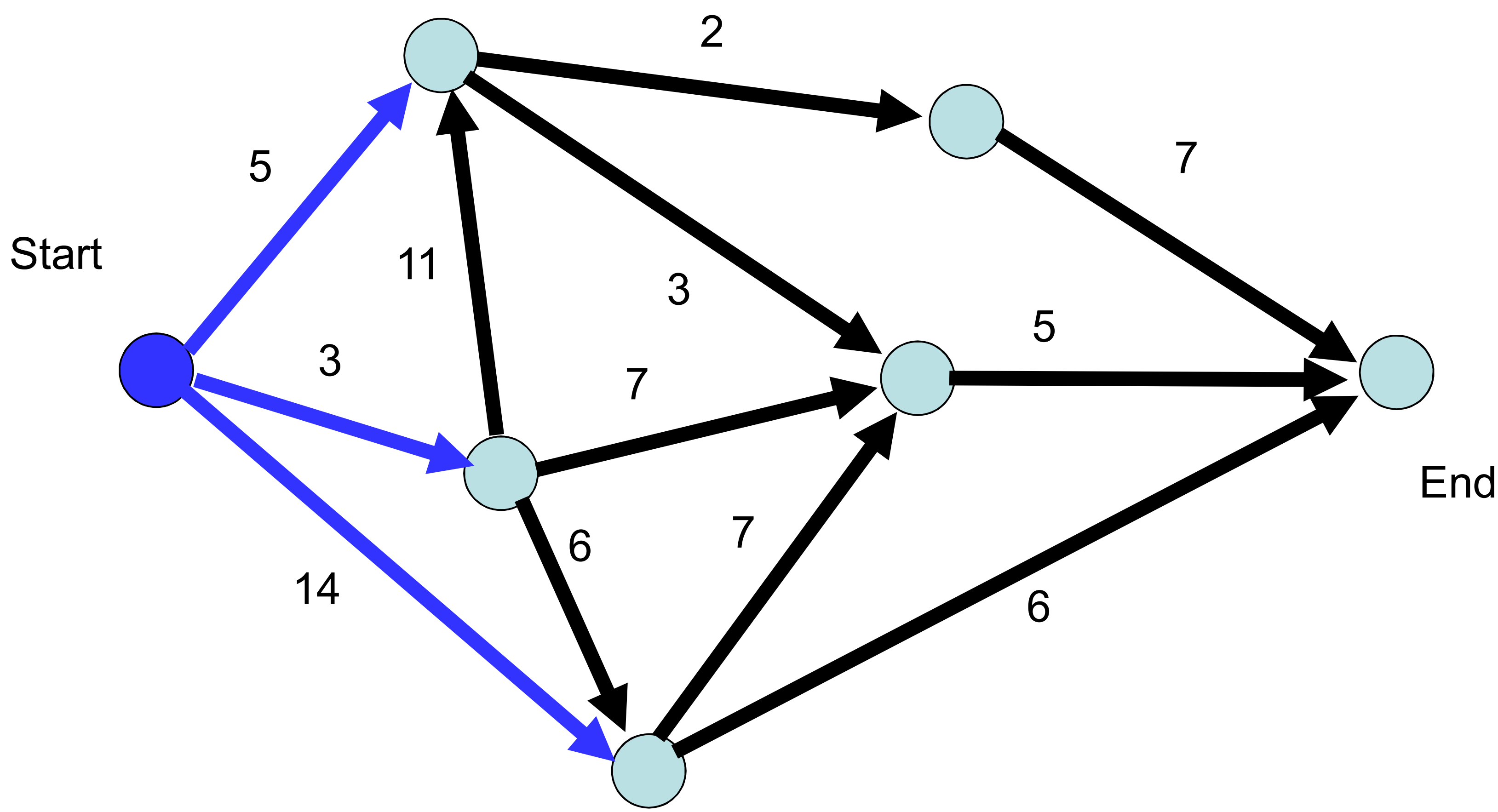
The Shortest Path Problem

- **Given a directed, acyclic, weighted graph…**
- **Start at some vertex A**
- **What is the shortest path from start vertex A to some end vertex B?**

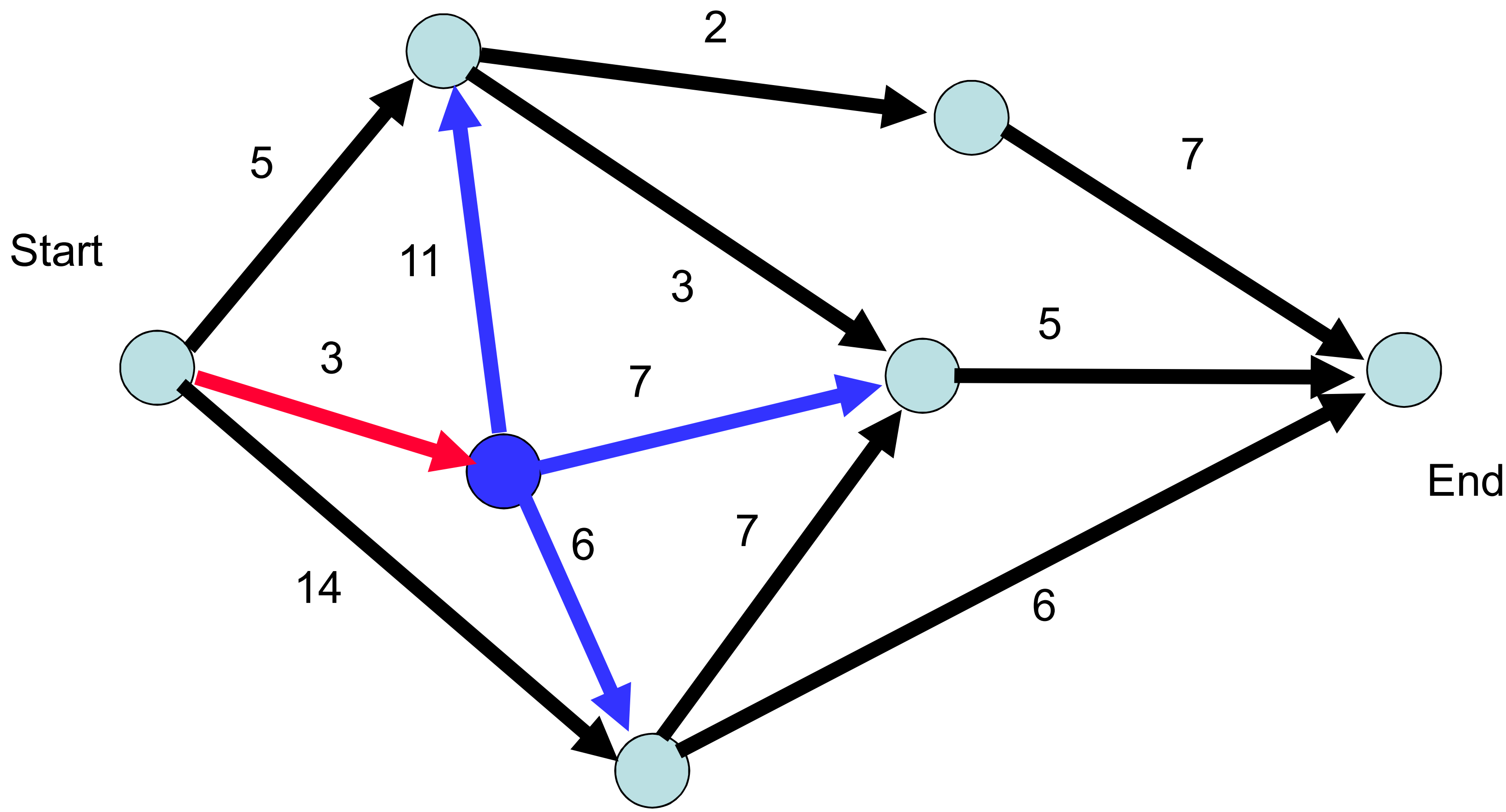




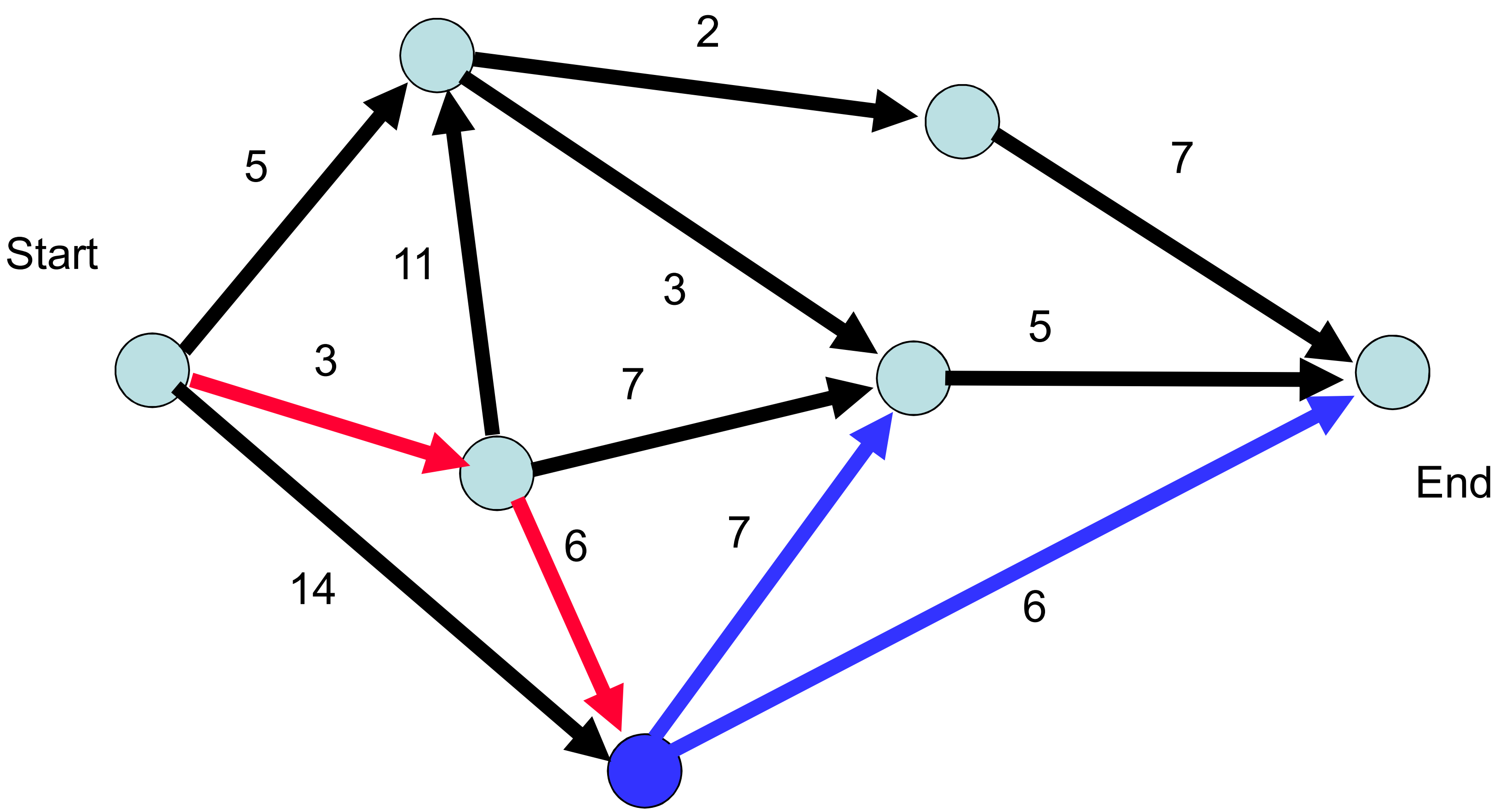
A Greedy Algorithm



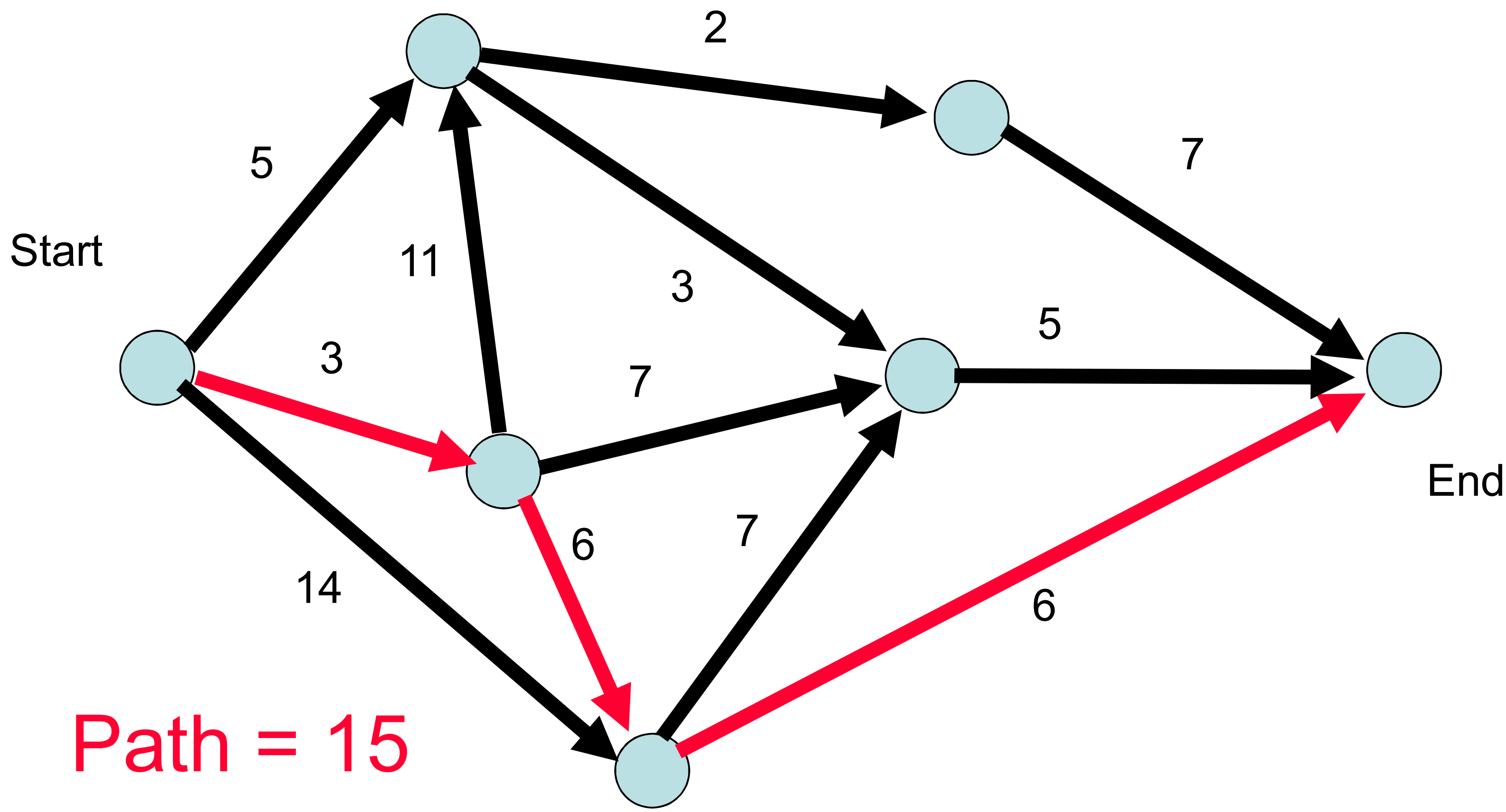
A Greedy Algorithm



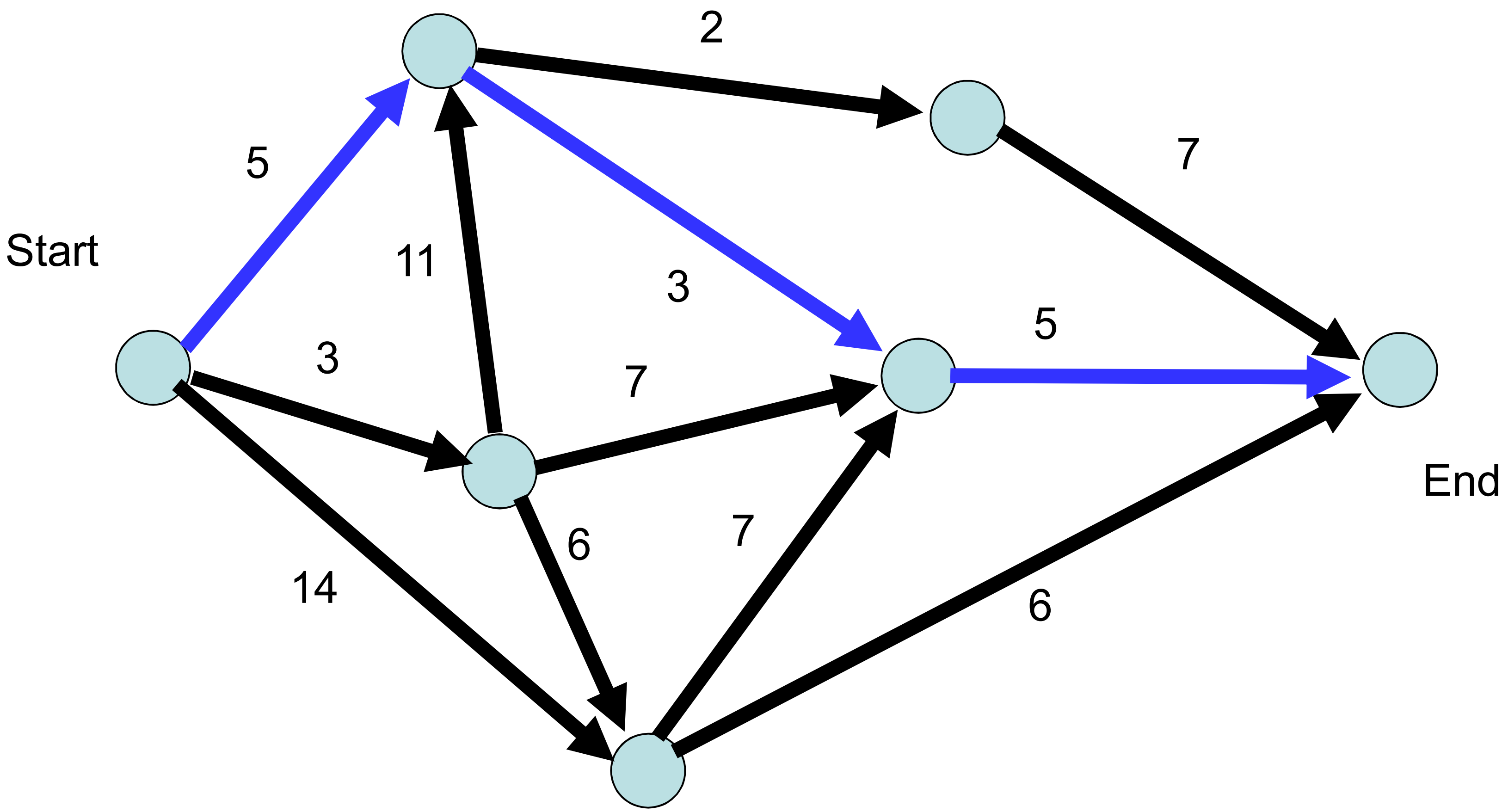
A Greedy Algorithm



A Greedy Algorithm



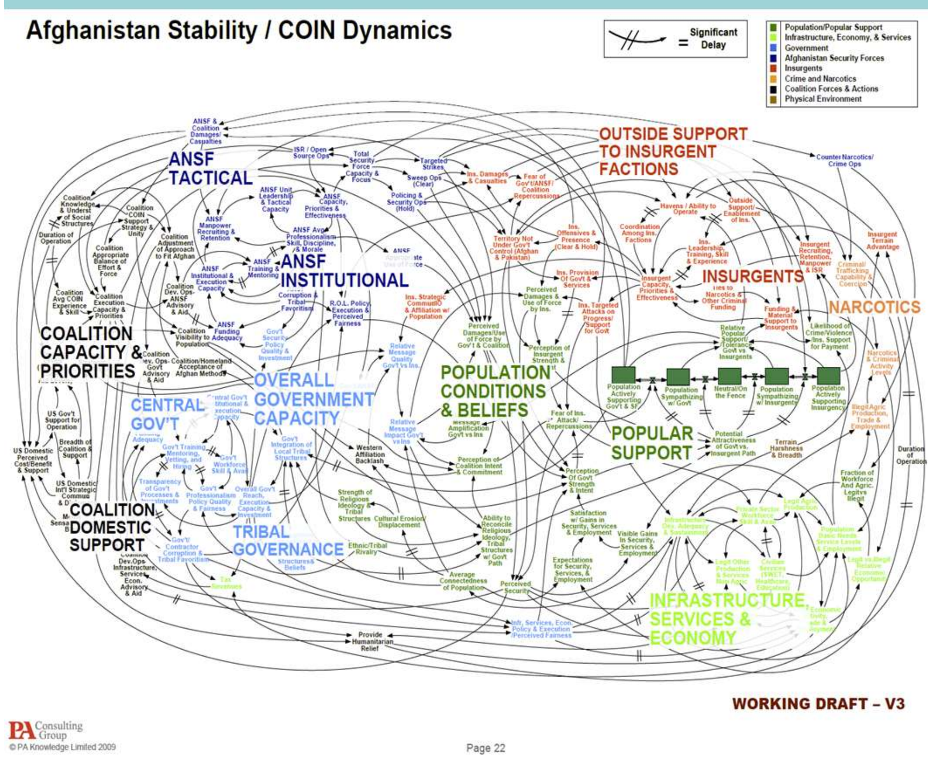
A Greedy Algorithm

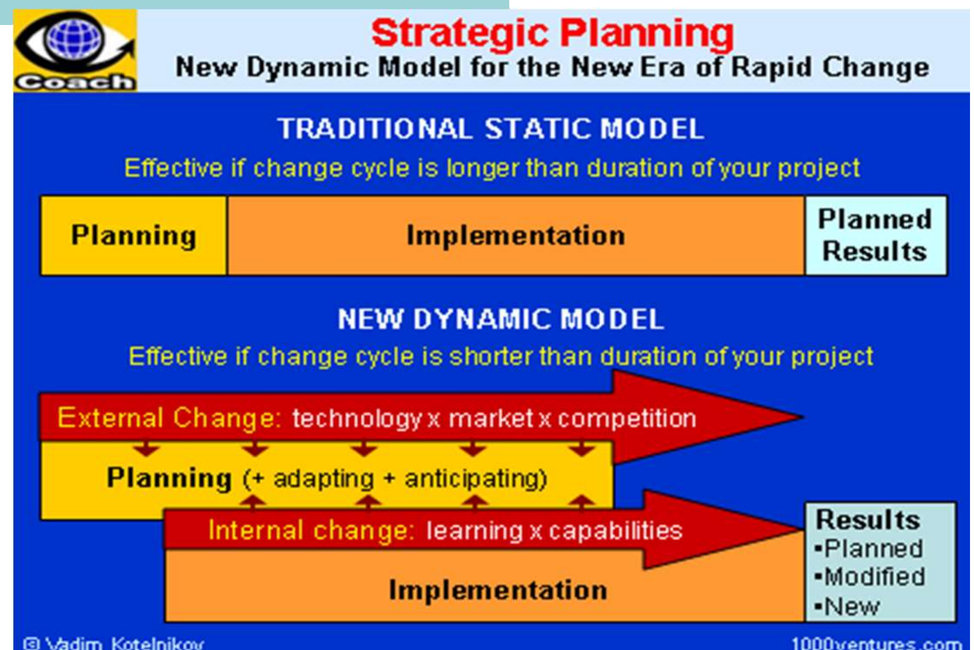


Shortest Path = 13

Dynamic Planning

- **Calculates all of the possible solution options, then chooses the best one.**
- **Implemented recursively.**
- **Produces an optimal solution.**
- **Spends more time.**





Bellman's Principle of Optimality

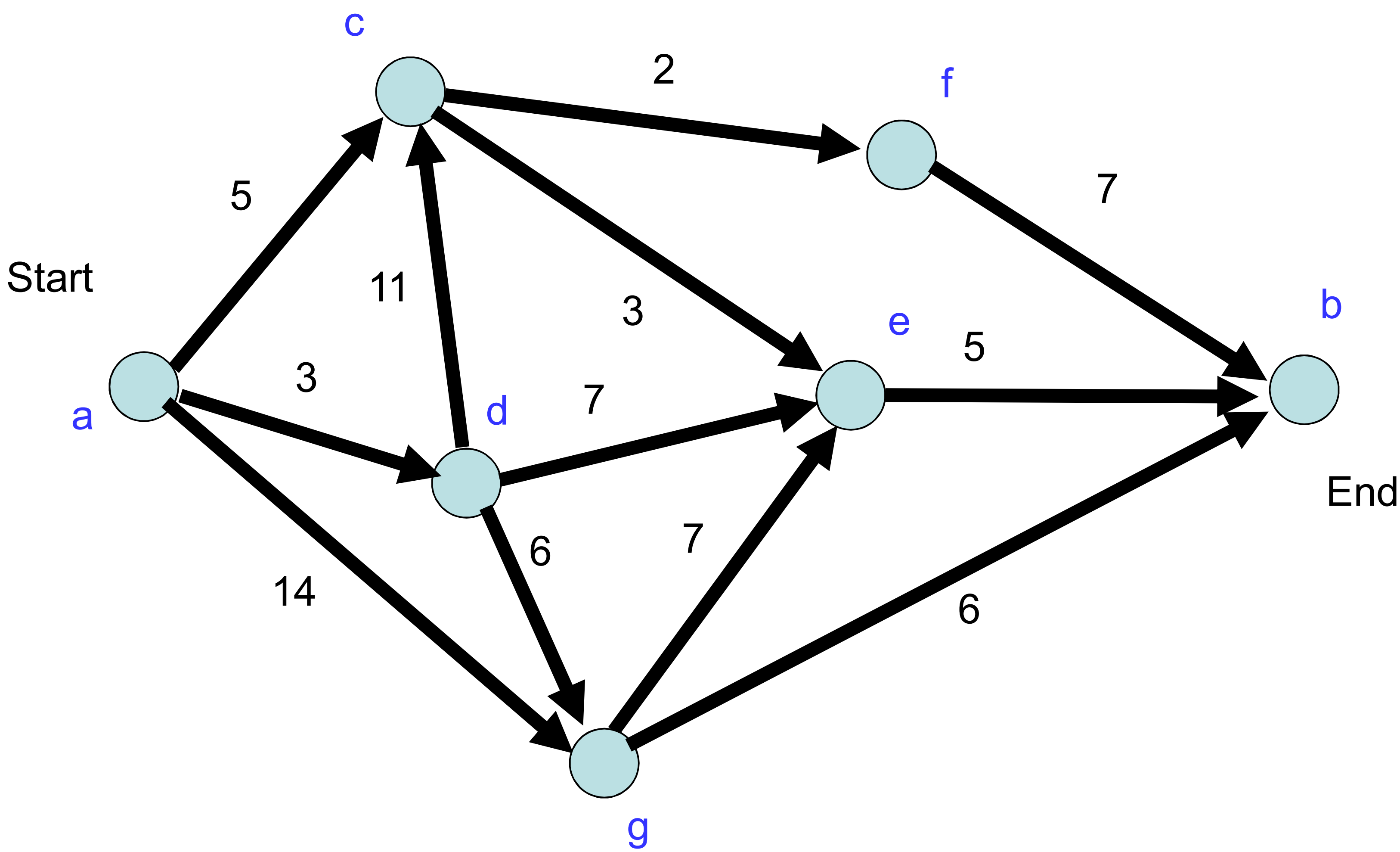
- **Regardless of how you reach a particular state (graph node), the optimal strategy for reaching the goal state is always the same.**

- **This greatly simplifies the strategy for searching for an optimal solution.**

The Shortest Path Problem

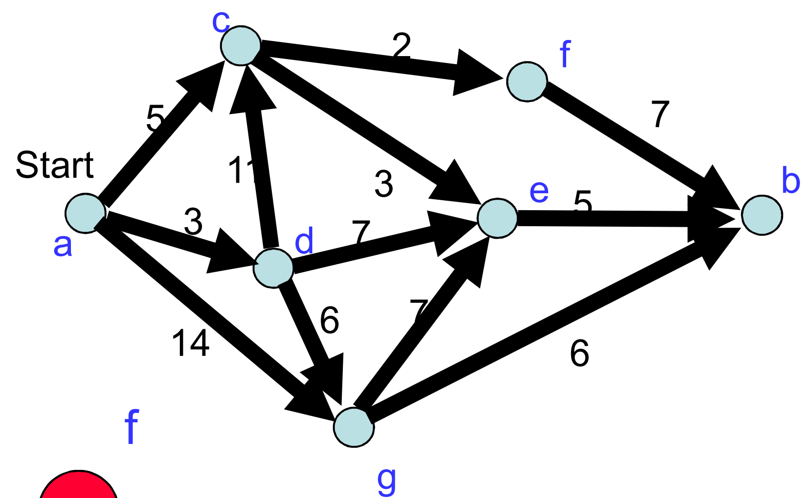
- **Given a directed, acyclic, weighted graph**
- **What is the shortest path from the start vertex to some end vertex?**
- **Minimize the sum of the edge weights**

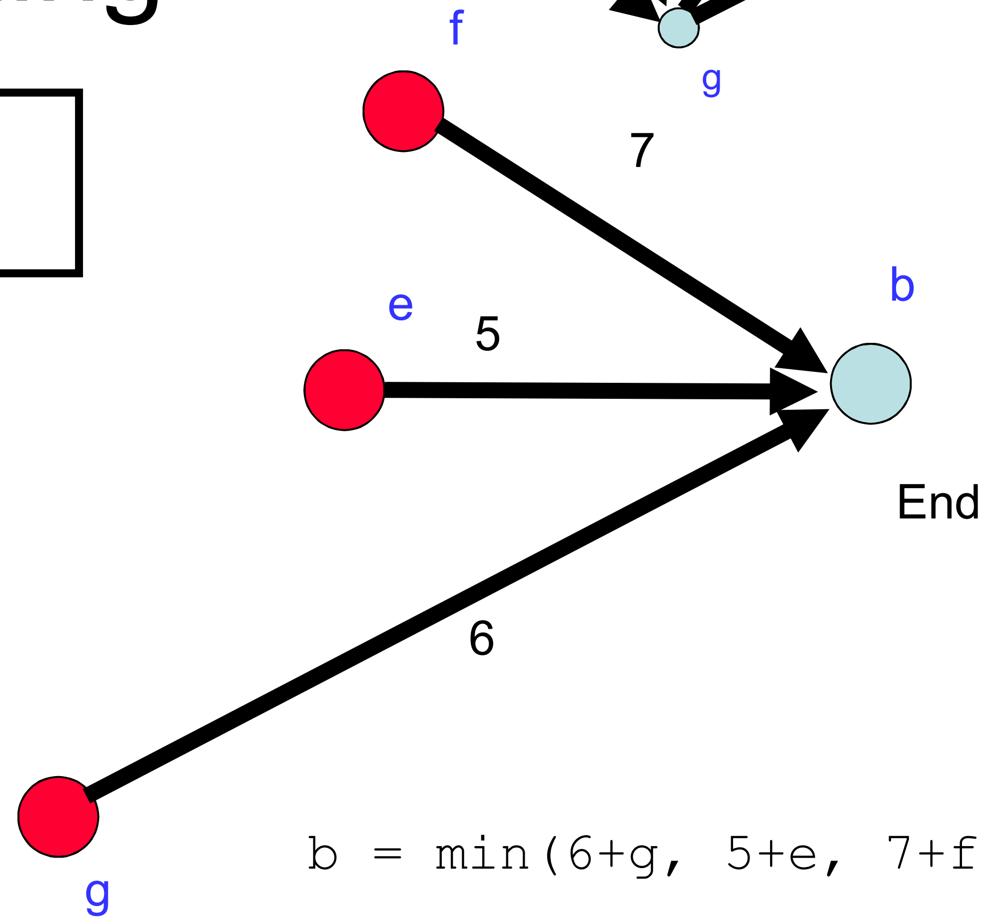
Dynamic Planning



Dynamic Planning

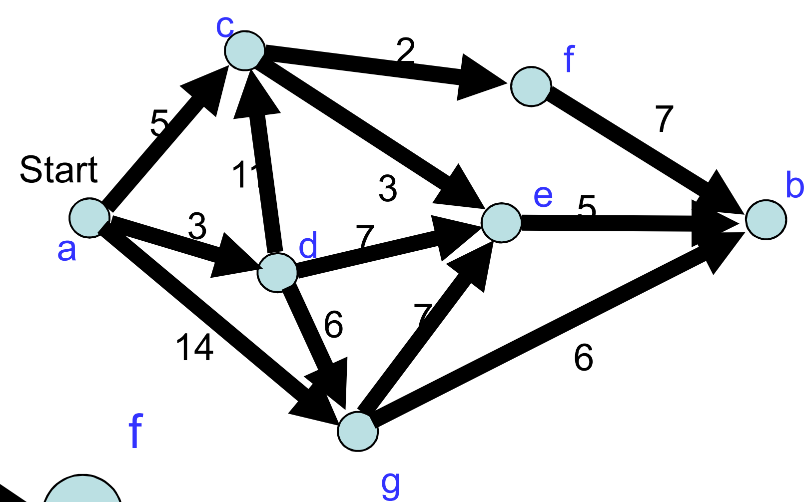
Notation: ‘x’ means
“shortest path to x”

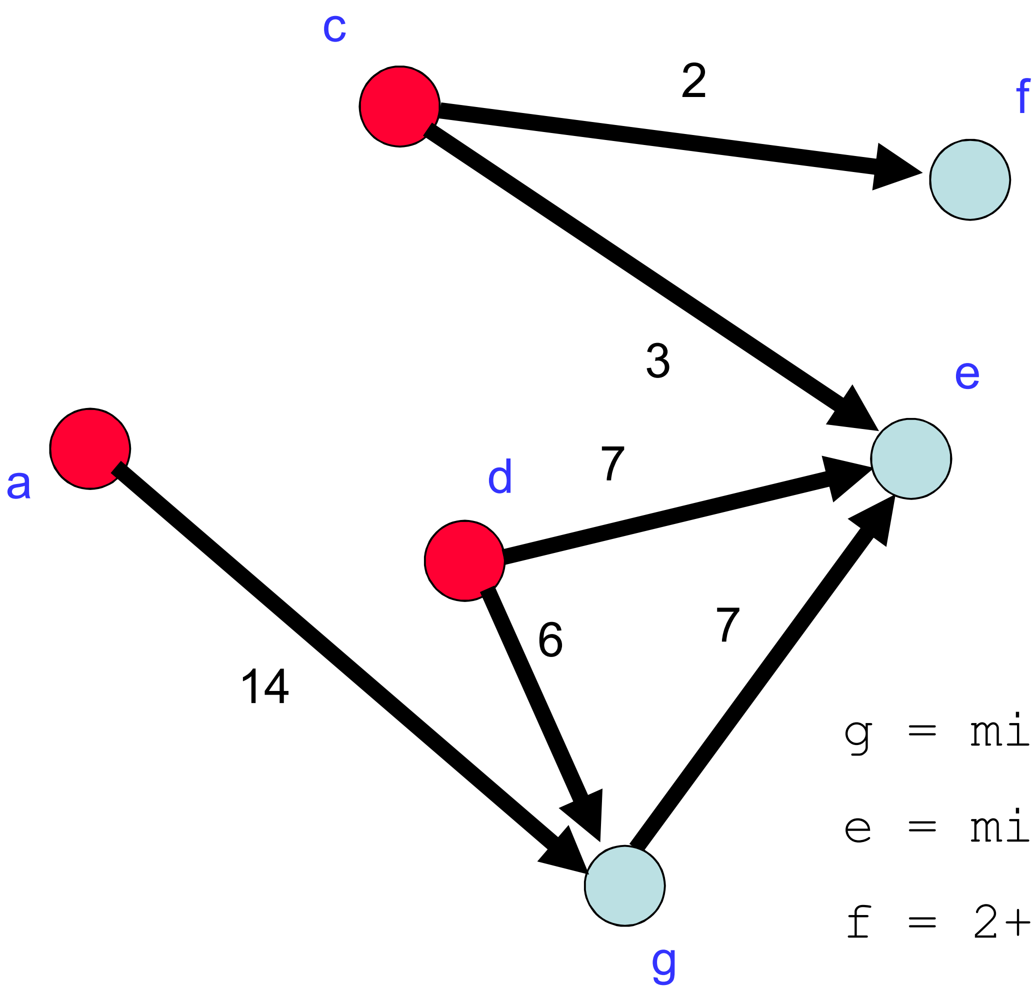





```
b = min(6+g, 5+e, 7+f)
```

Dynamic Planning



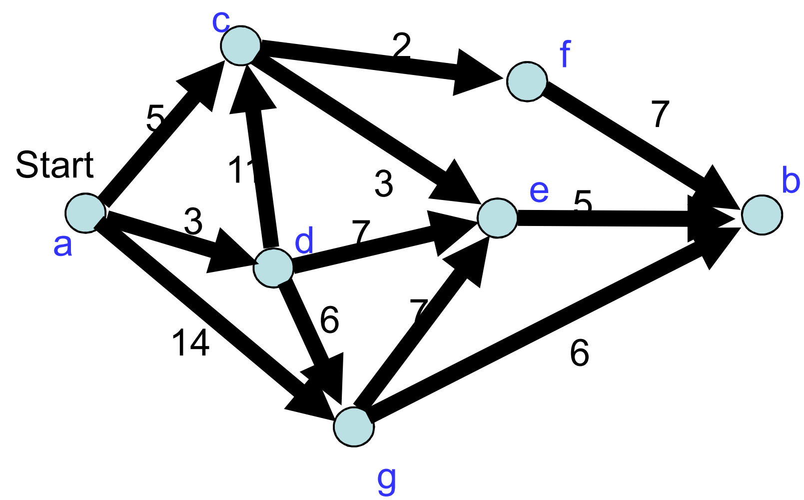


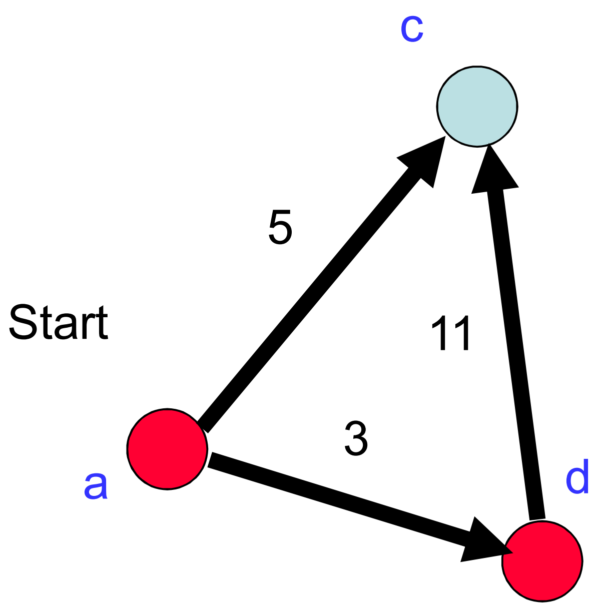
g = min(6+d, 14+a)

e = min(3+c, 7+d, 7+g)

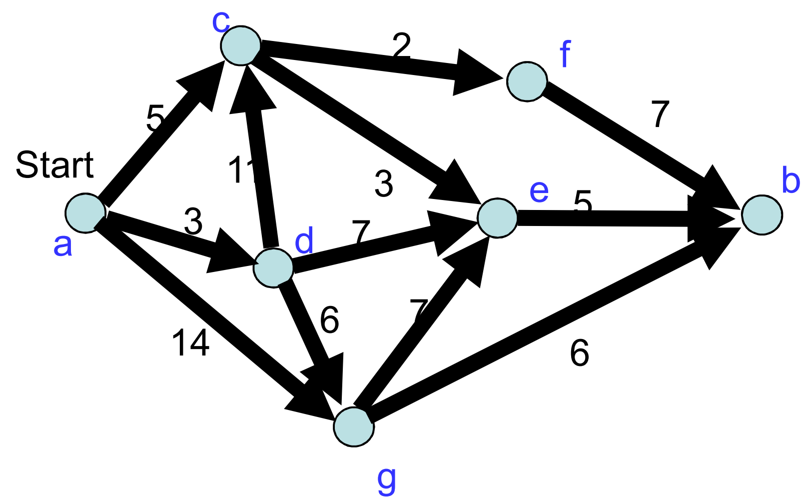
f = 2+c

Dynamic Planning

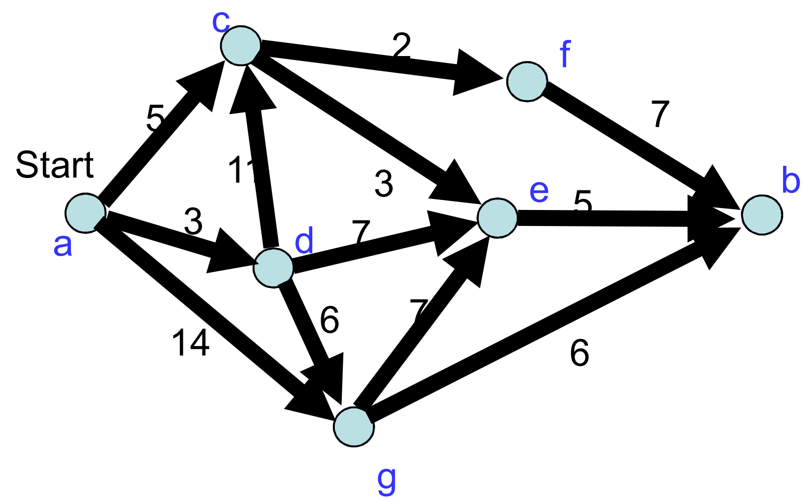





```
c = min(5+a, 11+d)

d = 3+a
```




```
b = min(6+g, 5+e, 7+f)

g = min(6+d, 14)

e = min(3+c, 7+d, 7+g)

f = 2+c

c = min(5, 11+d)

d = 3 via "a to d"
```



b = min(6+g, 5+e, 7+f)

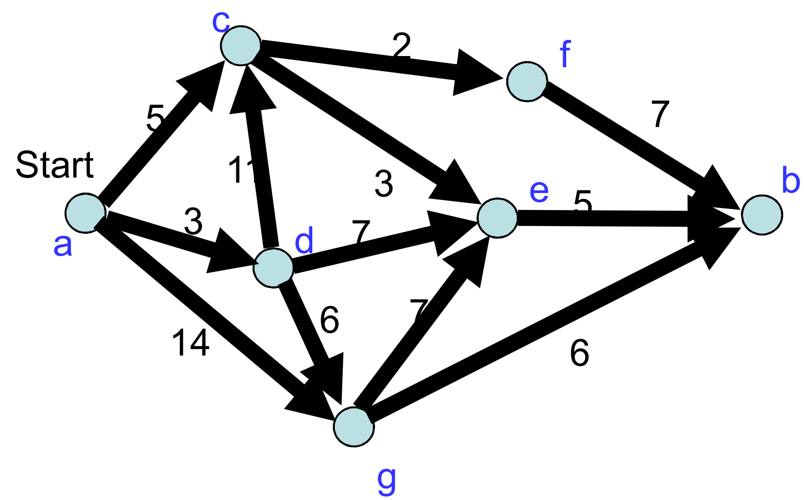
g = min(6+d, 14)

e = min(3+c, 7+d, 7+g)

f = 2+c

c = min(5, 11+d)

d = 3 via “a to d”



b = min(6+g, 5+e, 7+f)

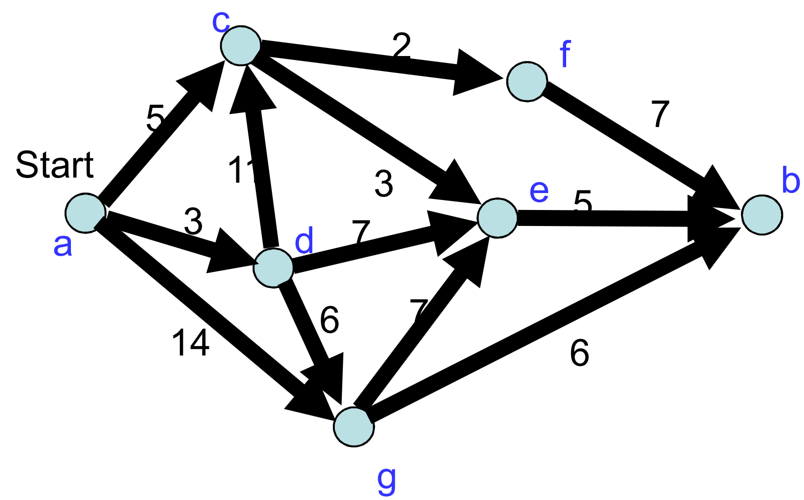
g = min(6+3, 14)

e = min(3+c, 7+3, 7+g)

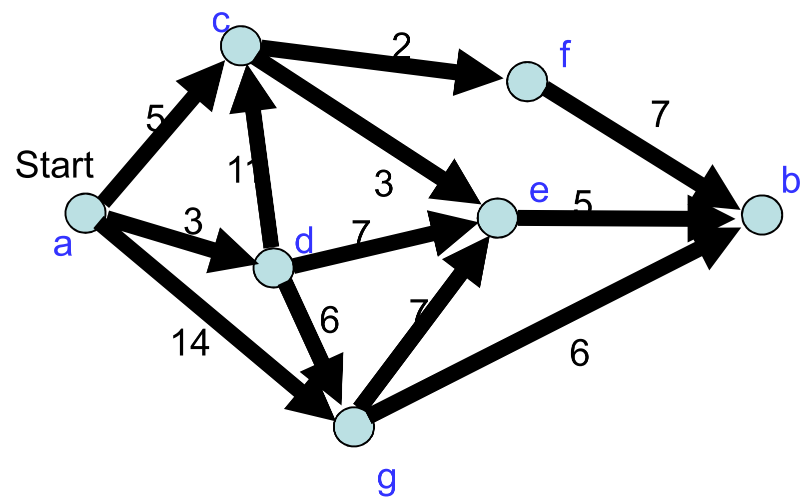
f = 2+c

c = min(5, 11+3)

d = 3 via “a to d”




```
b = min(6+g, 5+e, 7+f)

g = min(9, 14)

e = min(3+c, 10, 7+g)

f = 2+c

c = min(5, 14)

d = 3 via "a to d"
```



b = min(6+g, 5+e, 7+f)

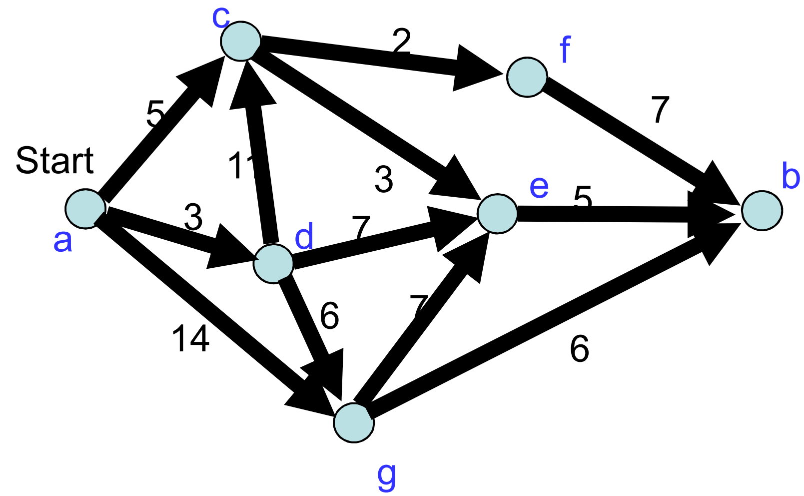
g = min(9, 14)

e = min(3+c, 10, 7+g)

f = 2+c

c = min(5, 14)

d = 3 via “a to d”



b = min(6+g, 5+e, 7+f)

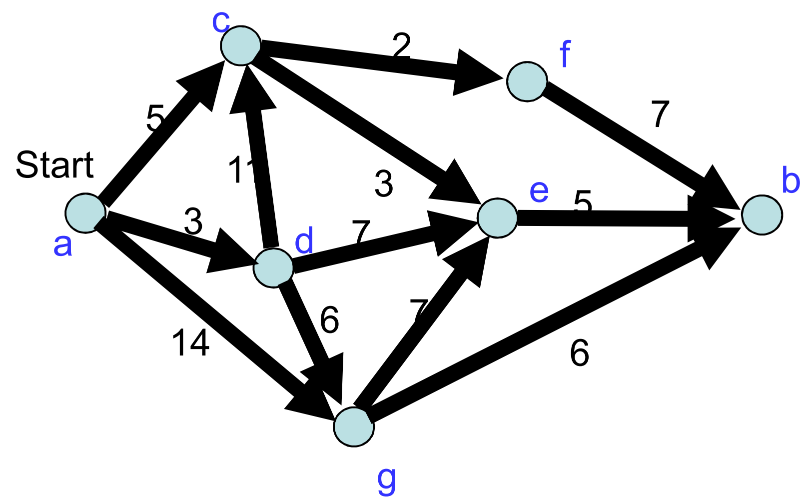
g = 9 via "a to d to g"

e = min(3+c, 10, 7+g)

f = 2+c

c = 5 via "a to c"

d = 3 via "a to d"



b = min(6+g, 5+e, 7+f)

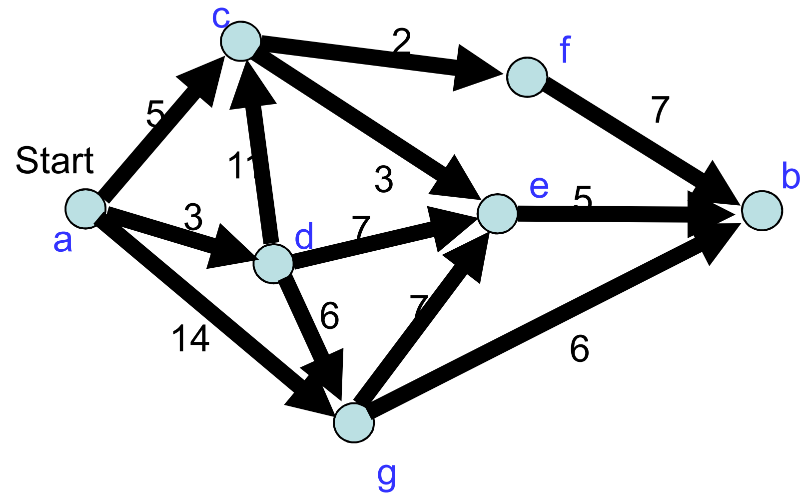
g = 9 via "a to d to g"

e = min(3+c, 10, 7+g)

f = 2+c

c = 5 via "a to c"

d = 3 via "a to d"



b = min(6+9, 5+e, 7+f)

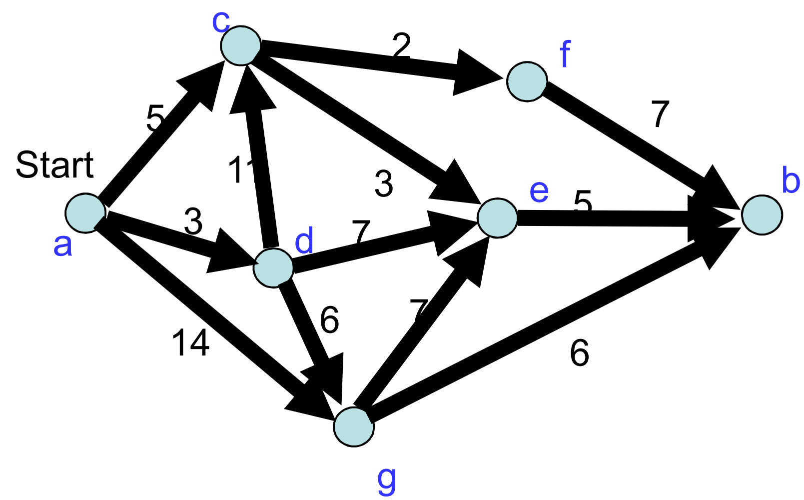
g = 9 via "a to d to g"

e = min(3+5, 10, 7+9)

f = 2+5

c = 5 via "a to c"

d = 3 via "a to d"



b = min(15, 5+e, 7+f)

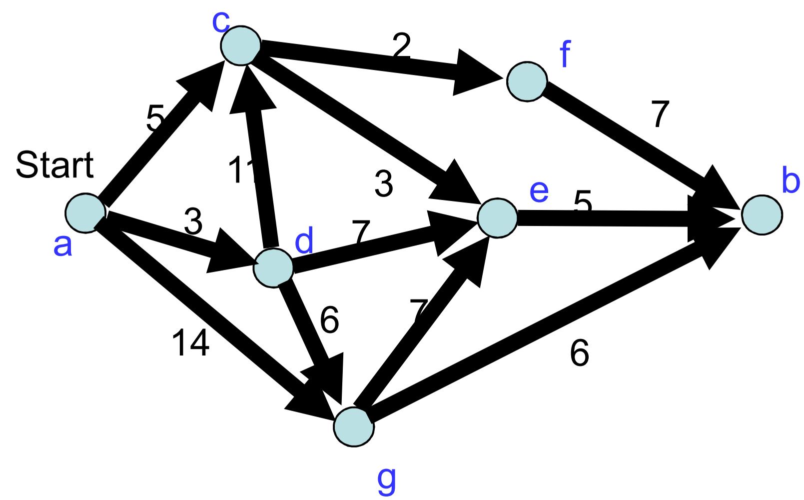
g = 9 via “a to d to g”

e = min(8, 10, 16)

f = 7 via “a to c to f”

c = 5 via “a to c”

d = 3 via “a to d”



b = min(15, 5+e, 7+f)

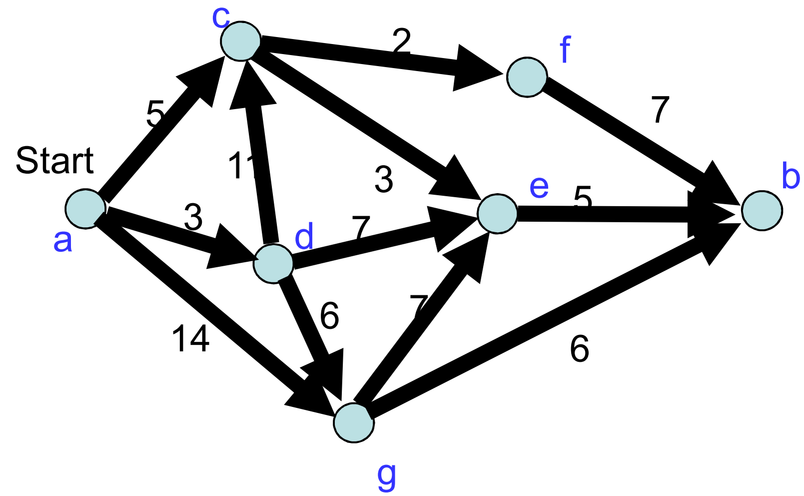
g = 9 via “a to d to g”

e = min(8, 10, 16)

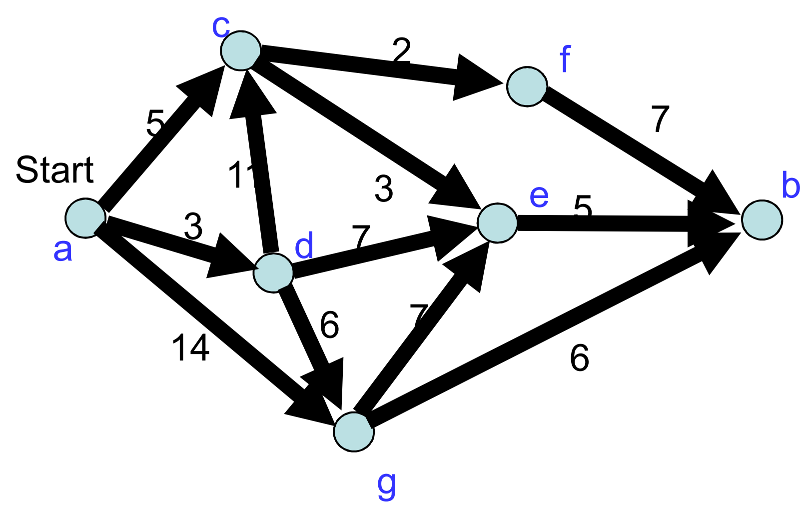
f = 7 via “a to c to f”

c = 5 via “a to c”

d = 3 via “a to d”




```
b = min(15, 5+e, 7+f)

g = 9 via "a to d to g"

e = 8 via "a to c to e"

f = 7 via "a to c to f"

c = 5 via "a to c"

d = 3 via "a to d"
```



b = min(15, 5+e, 7+f)

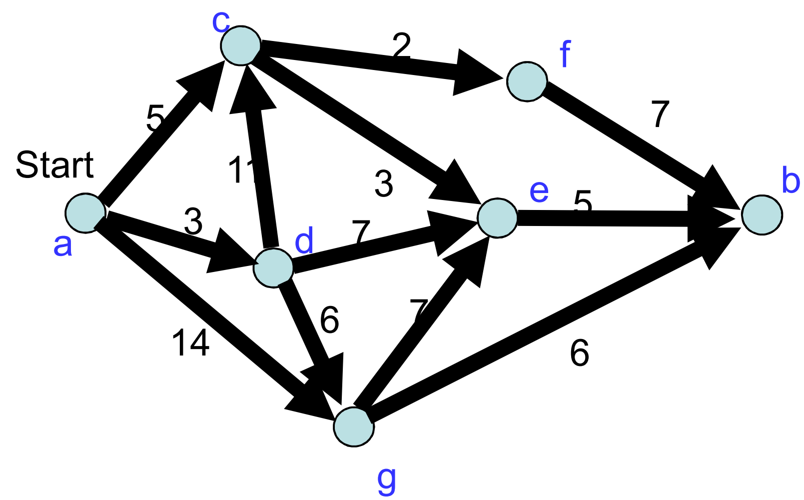
g = 9 via “a to d to g”

e = 8 via “a to c to e”

f = 7 via “a to c to f”

c = 5 via “a to c”

d = 3 via “a to d”



b = min(15, 5+8, 7+7)

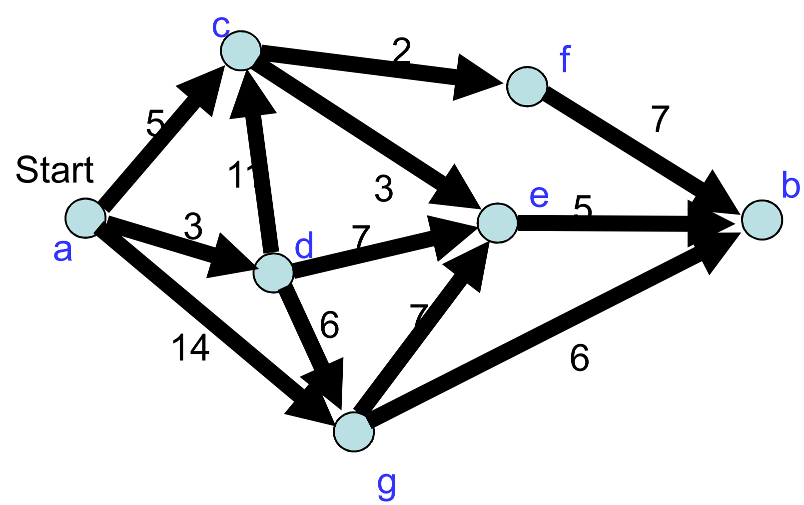
g = 9 via “a to d to g”

e = 8 via “a to c to e”

f = 7 via “a to c to f”

c = 5 via “a to c”

d = 3 via “a to d”



b = min(15, 13, 14)

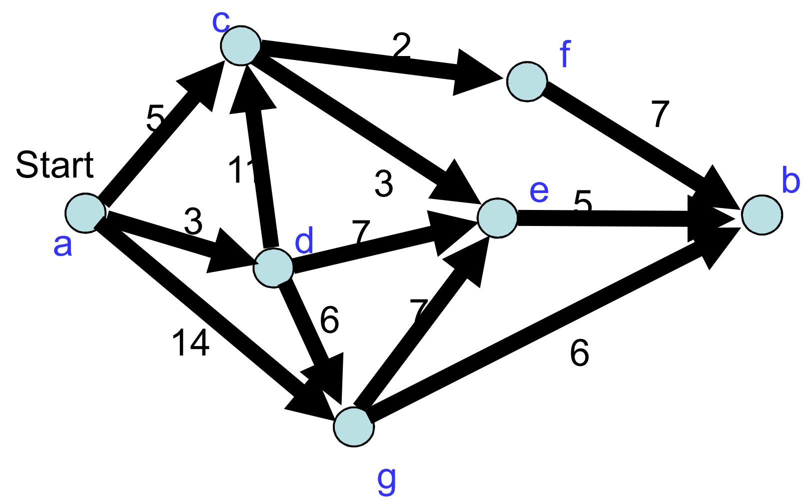
g = 9 via "a to d to g"

e = 8 via "a to c to e"

f = 7 via "a to c to f"

c = 5 via "a to c"

d = 3 via "a to d"



b = min(15, 13, 14)

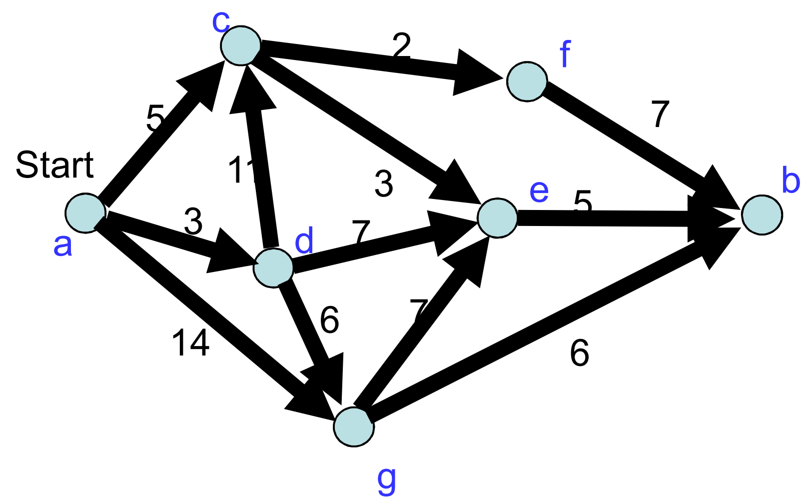
g = 9 via "a to d to g"

e = 8 via "a to c to e"

f = 7 via "a to c to f"

c = 5 via "a to c"

d = 3 via "a to d"



b = 13 via "a to c to e to b"

g = 9 via "a to d to g"

e = 8 via "a to c to e"

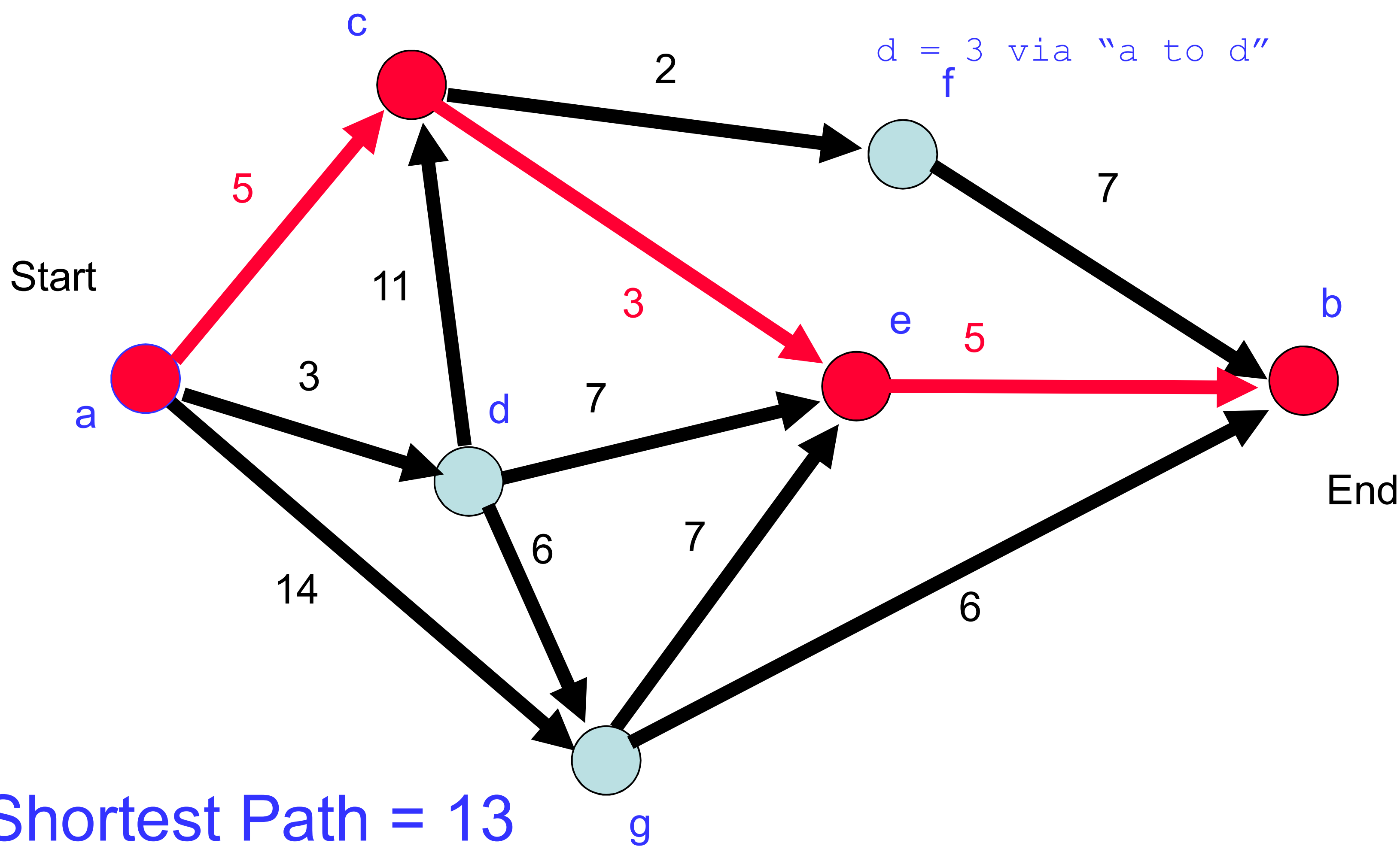
f = 7 via "a to c to f"

c = 5 via "a to c"

d = 3 via "a to d"

Dynamic Planning

```
b = 13 via "a to c to e to b"
g = 9 via "a to d to g"
e = 8 via "a to c to e"
f = 7 via "a to c to f"
c = 5 via "a to c"
d = 3 via "a to d"
```

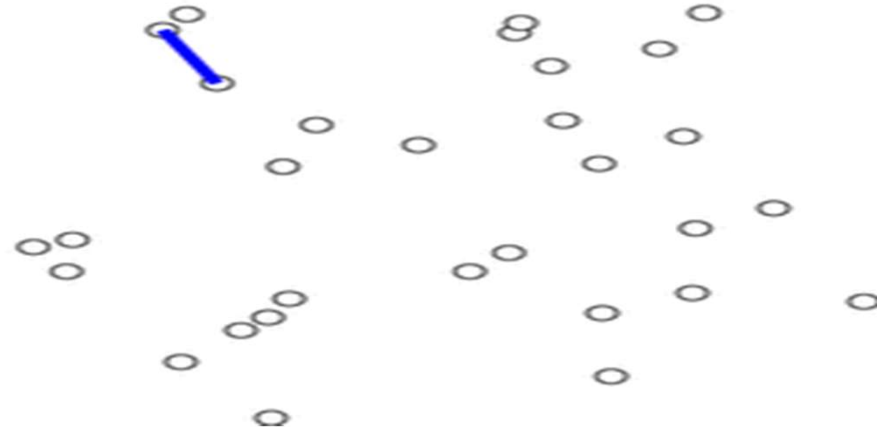


Shortest Path = 13

Summary

- Greedy algorithms
  - Make short-sighted, "best guess" decisions
  - Required less time/work
  - Provide approximate solutions
- Dynamic planning
  - Examines all possible solutions
  - Requires more time/work
  - Guarantees optimal solution

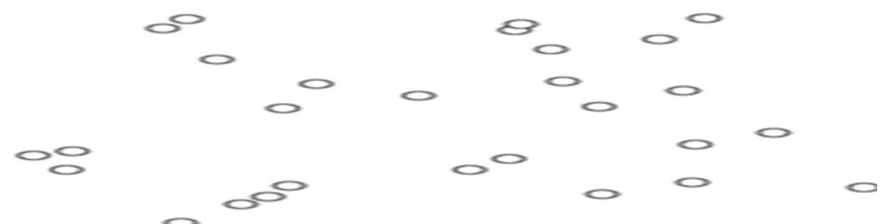
Prim's Algorithm



Finds a minimum spanning tree for a weighted undirected graph. This means it finds a subset of the edges that forms a tree that includes every vertex, where the total weight of all the edges in the tree is minimized.

1. **Initialize a tree with a single vertex, chosen arbitrarily**
2. **Grow the tree by one edge: of the edges that connect the tree to vertices not yet in the tree, find the minimum-weight edge, and transfer it to the tree.**
3. **Repeat step 2 (until all vertices are in the tree).**

Kruskal's Algorithm



Minimum-spanning-tree **algorithm** which finds an edge of the least possible weight that connects any two trees in the forest. In graph theory as it finds a minimum spanning tree for a connected weighted graph adding increasing cost arcs at each step.

1. Ascend & create a graph *F* (a set of trees), where each vertex in the graph is a separate tree
2. create a set *S* containing all the edges in the graph
3. while *S* is nonempty and *F* is not yet spanning
   1. remove an edge with minimum weight from *S*
   2. if the removed edge connects two different trees then add it to the forest *F*, combining two trees into a single tree

Prim's Algorithm

Pick any vertex and add it to "vertices" list

Loop

Exitif ("vertices" contains all the vertices in graph)

Select shortest, unmarked edge coming from "vertices" list

If (shortest edge does NOT create a cycle) then

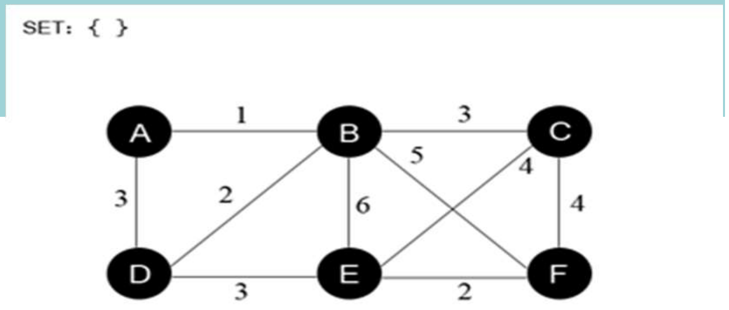
Add that edge to your "edges"

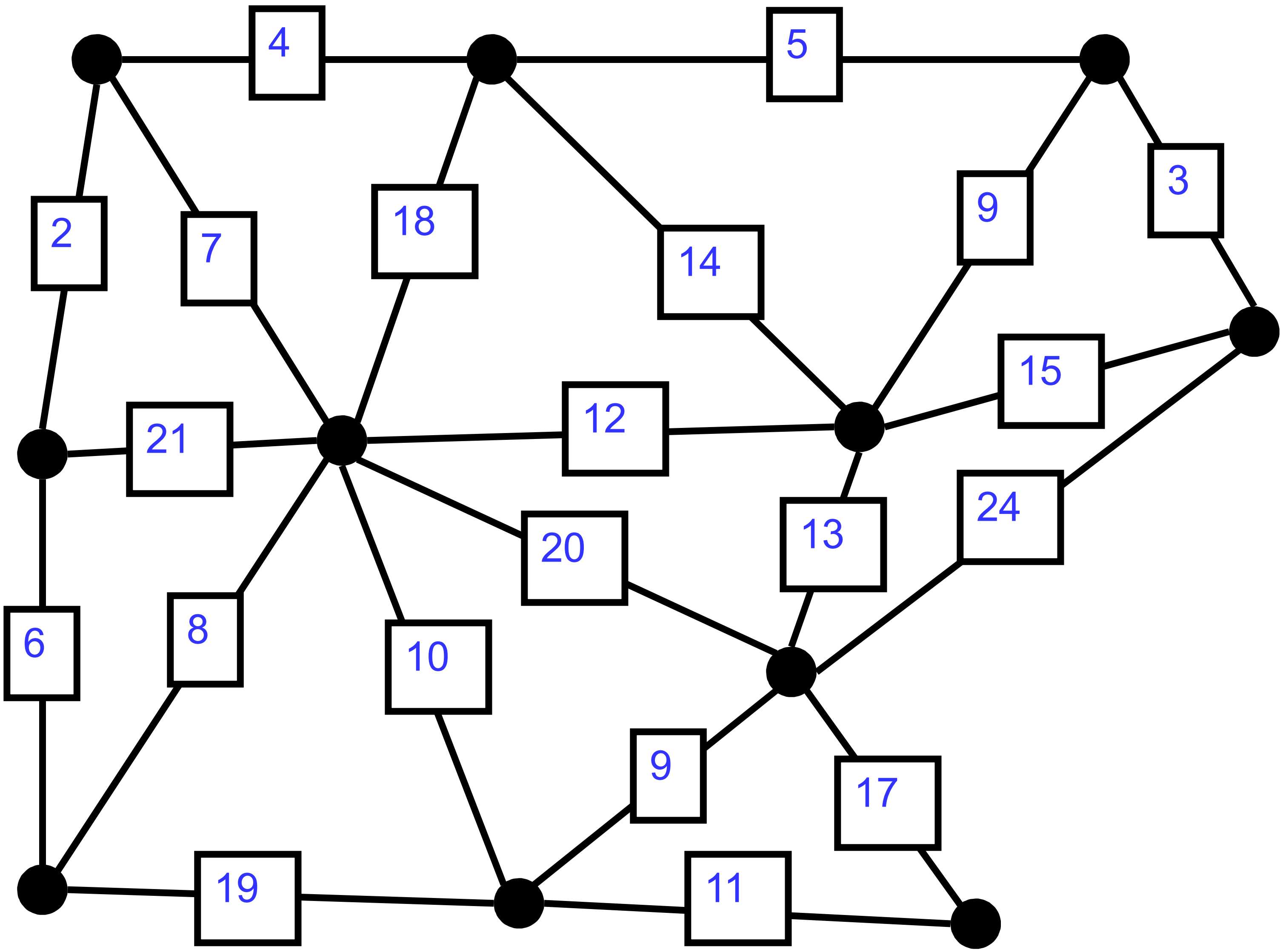
Add the adjoining vertex to the "vertices" list

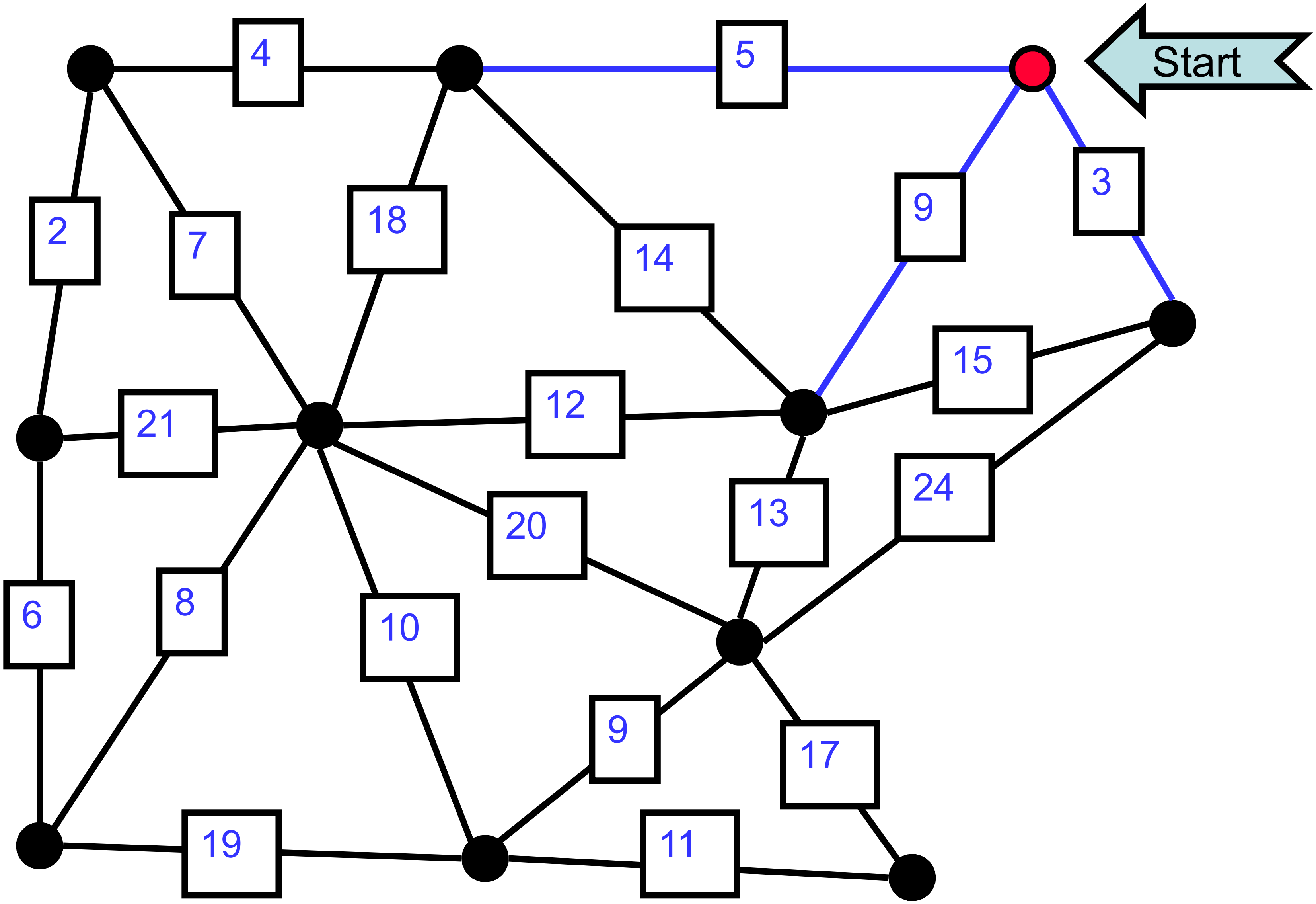
Endif

Mark that edge as having been considered

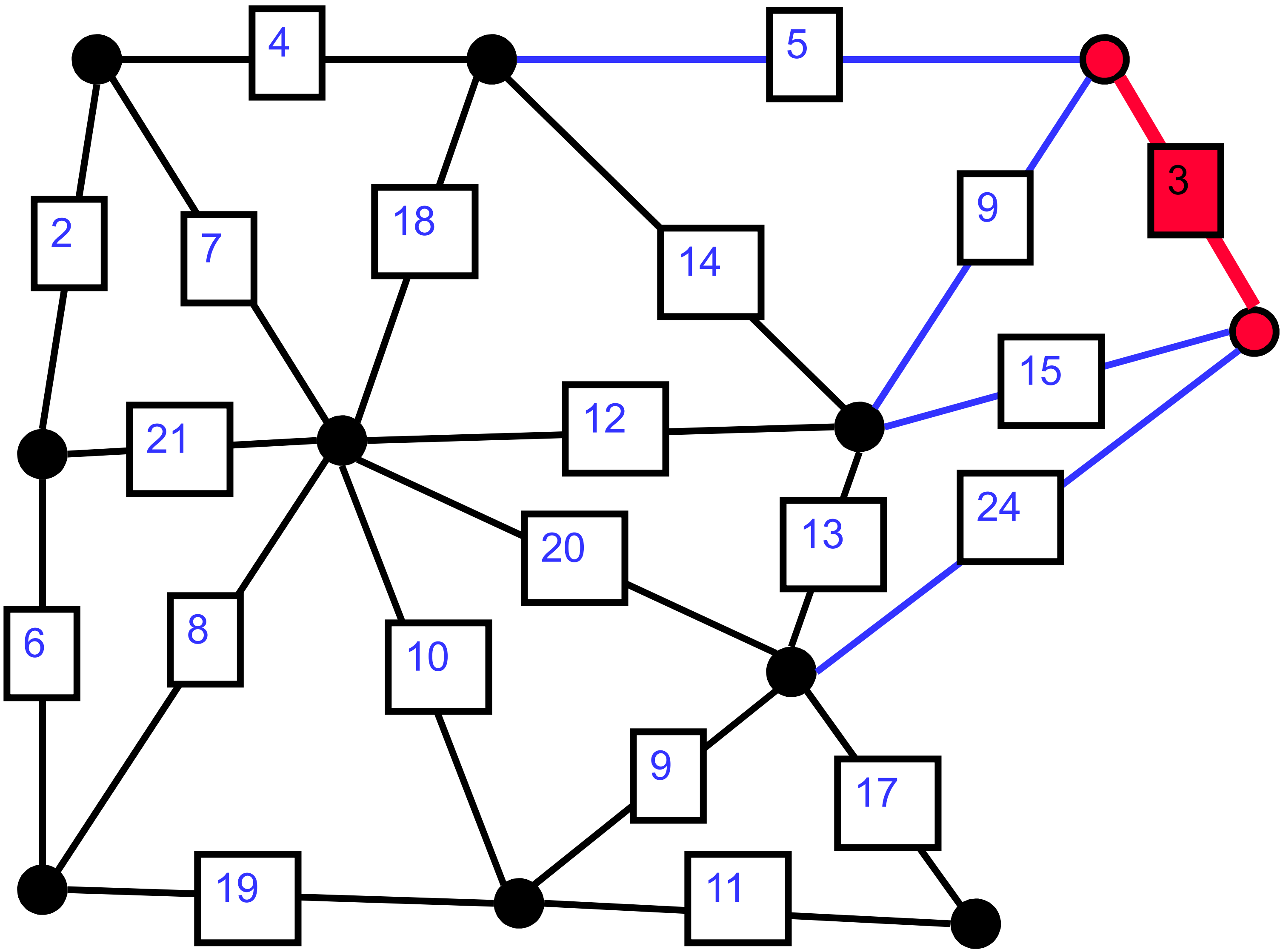
Endloop

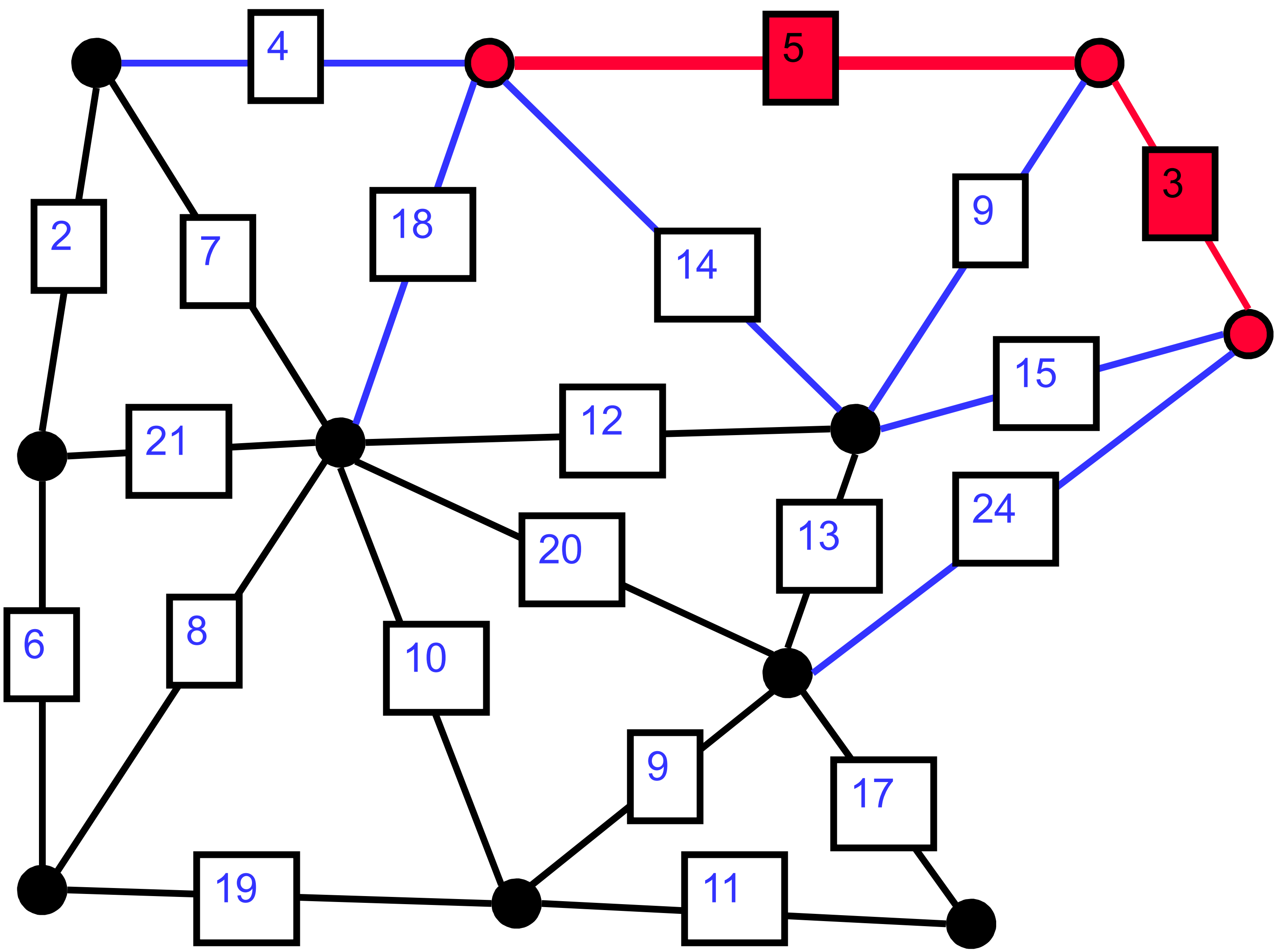


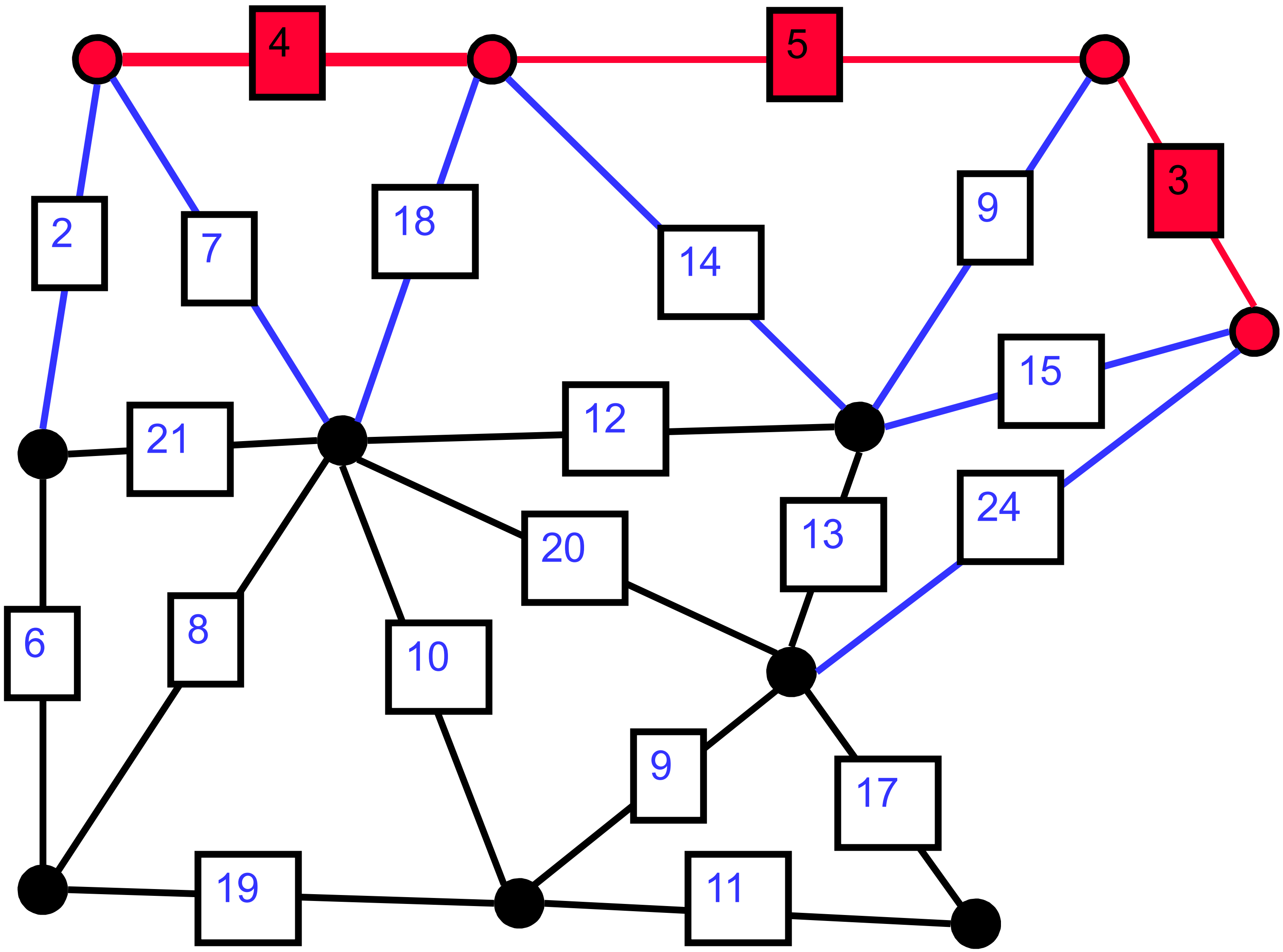

4
5
2
7
18
14
9
3
15
21
12
24
20
13
6
8
10
9
17
19
11

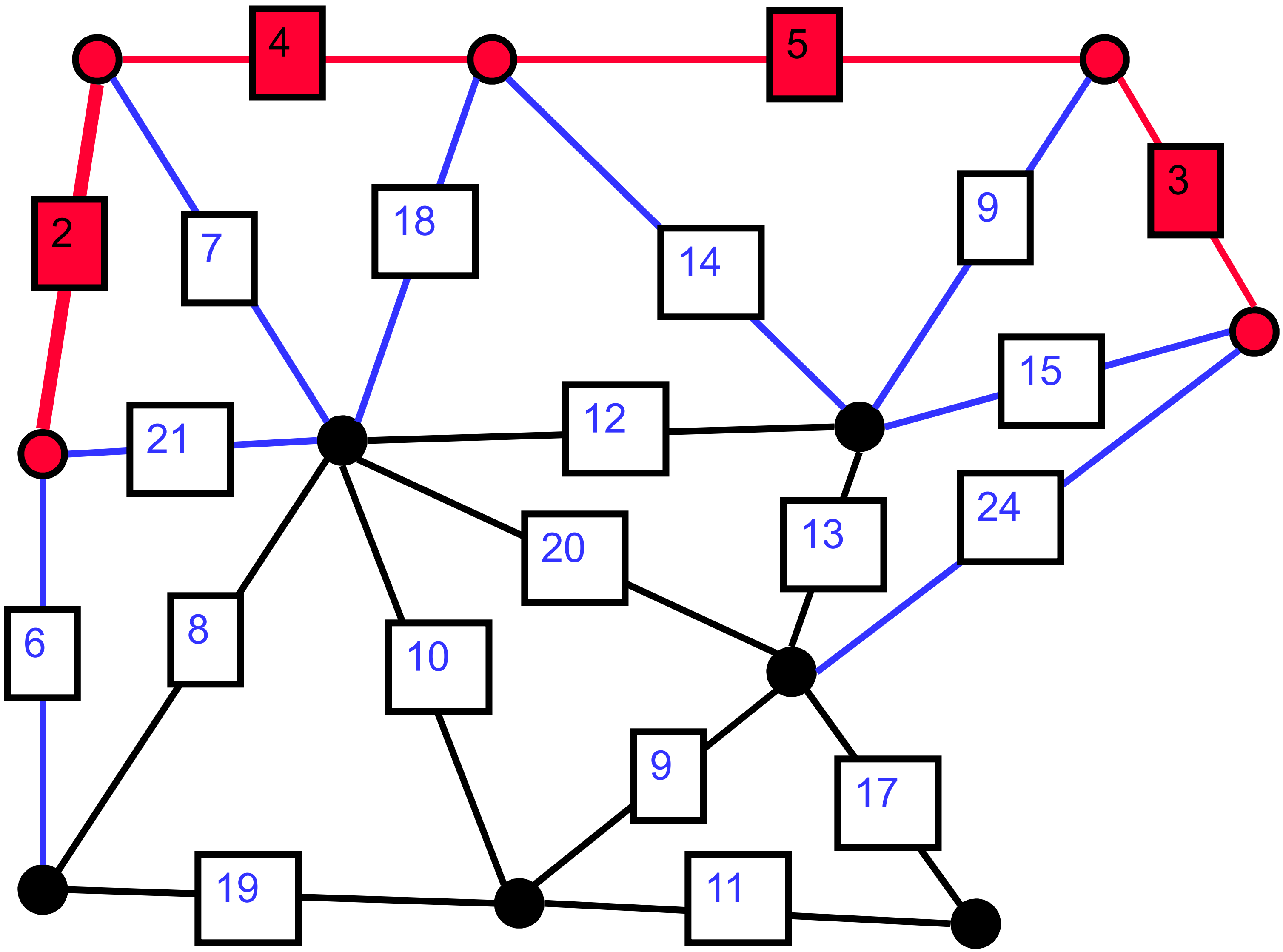


Start
4
5
2
7
18
14
9
3
15
21
12
24
13
20
6
8
10
9
17
19
11

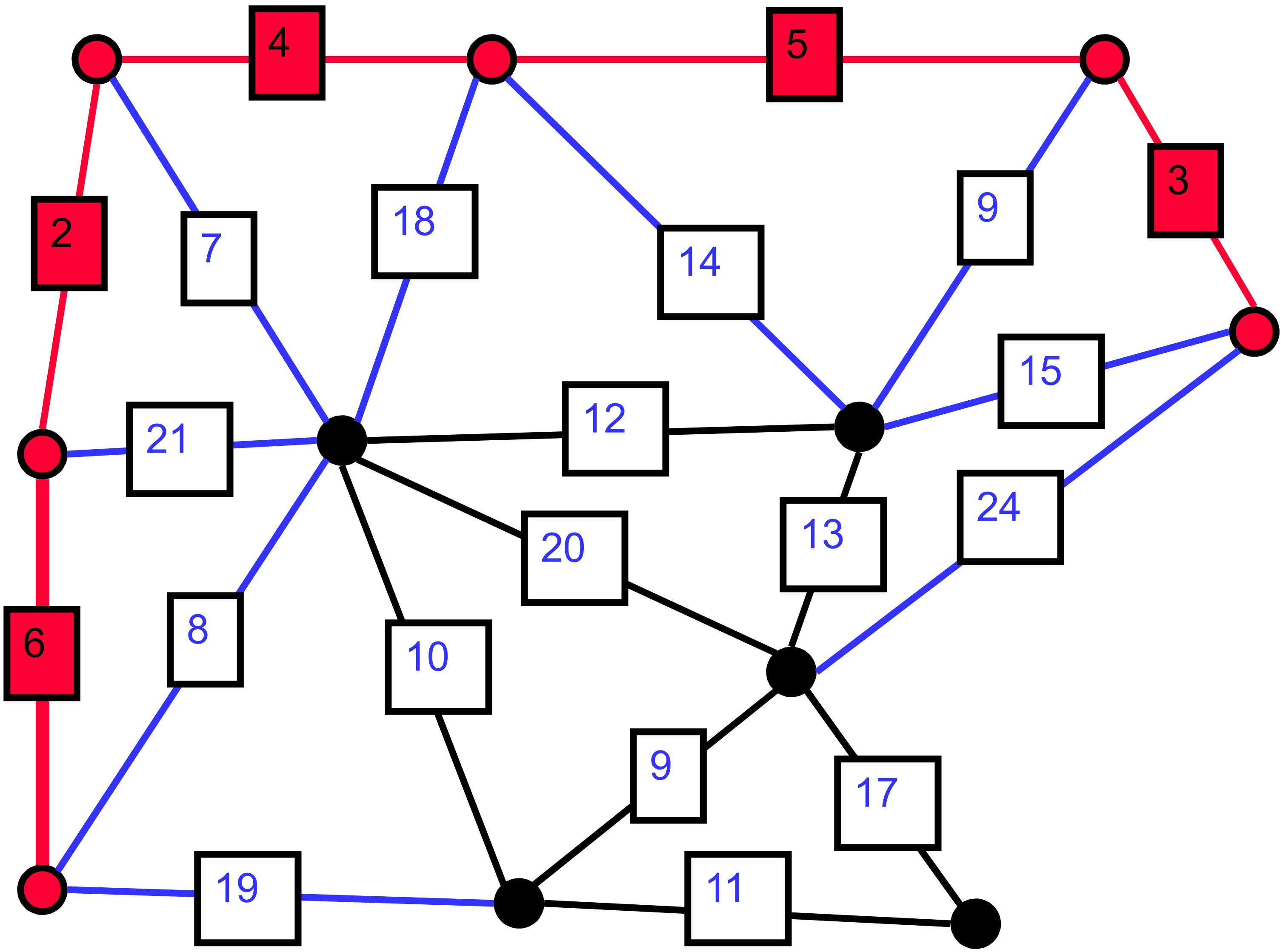

4
5
3
2
7
18
14
9
15
21
12
13
24
20
6
8
10
9
17
19
11

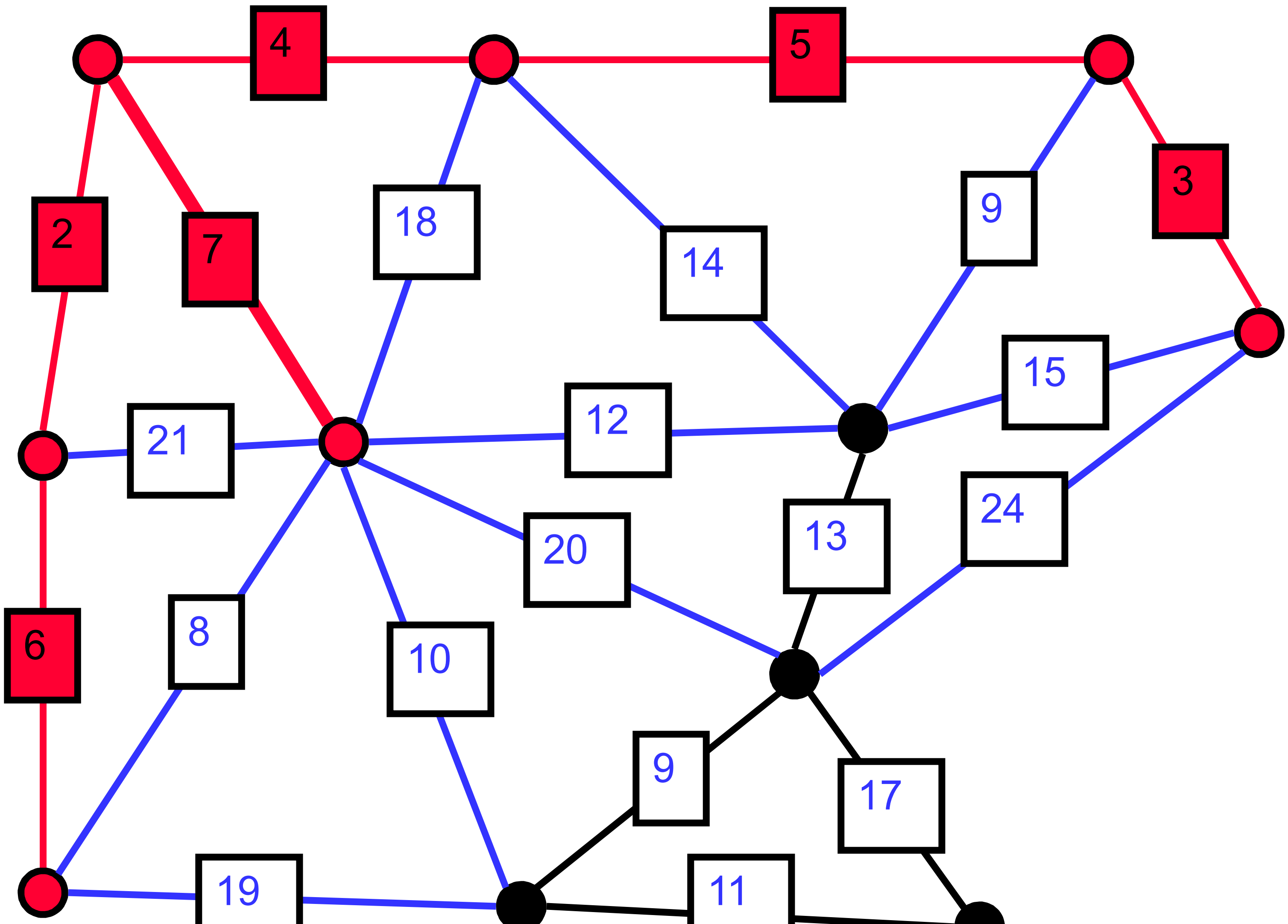

4
5
3
2
7
18
14
9
15
21
12
13
24
20
6
8
10
9
17
19
11

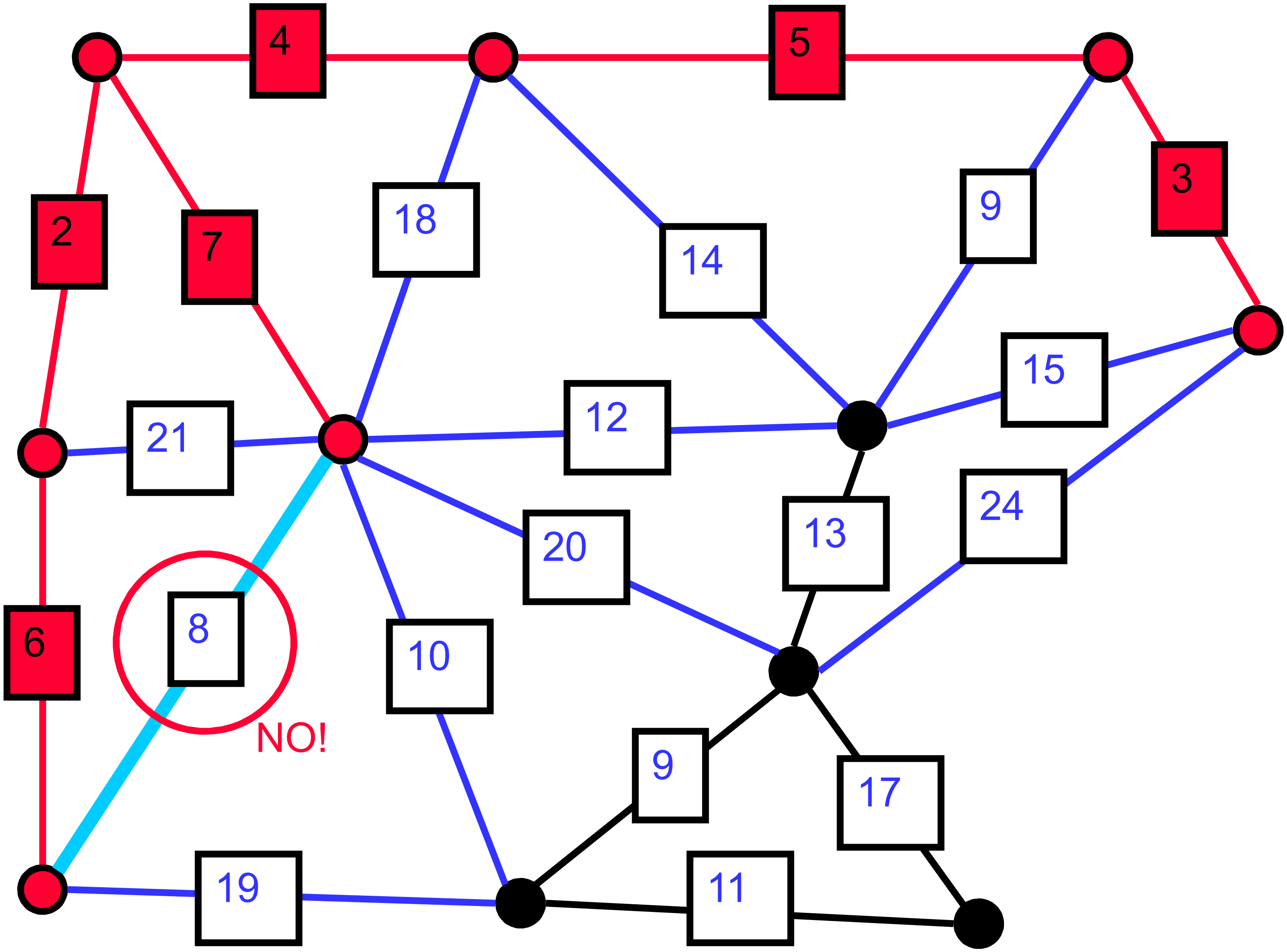

4
5
3
2
7
18
14
9
15
21
12
13
24
20
6
8
10
9
17
19
11

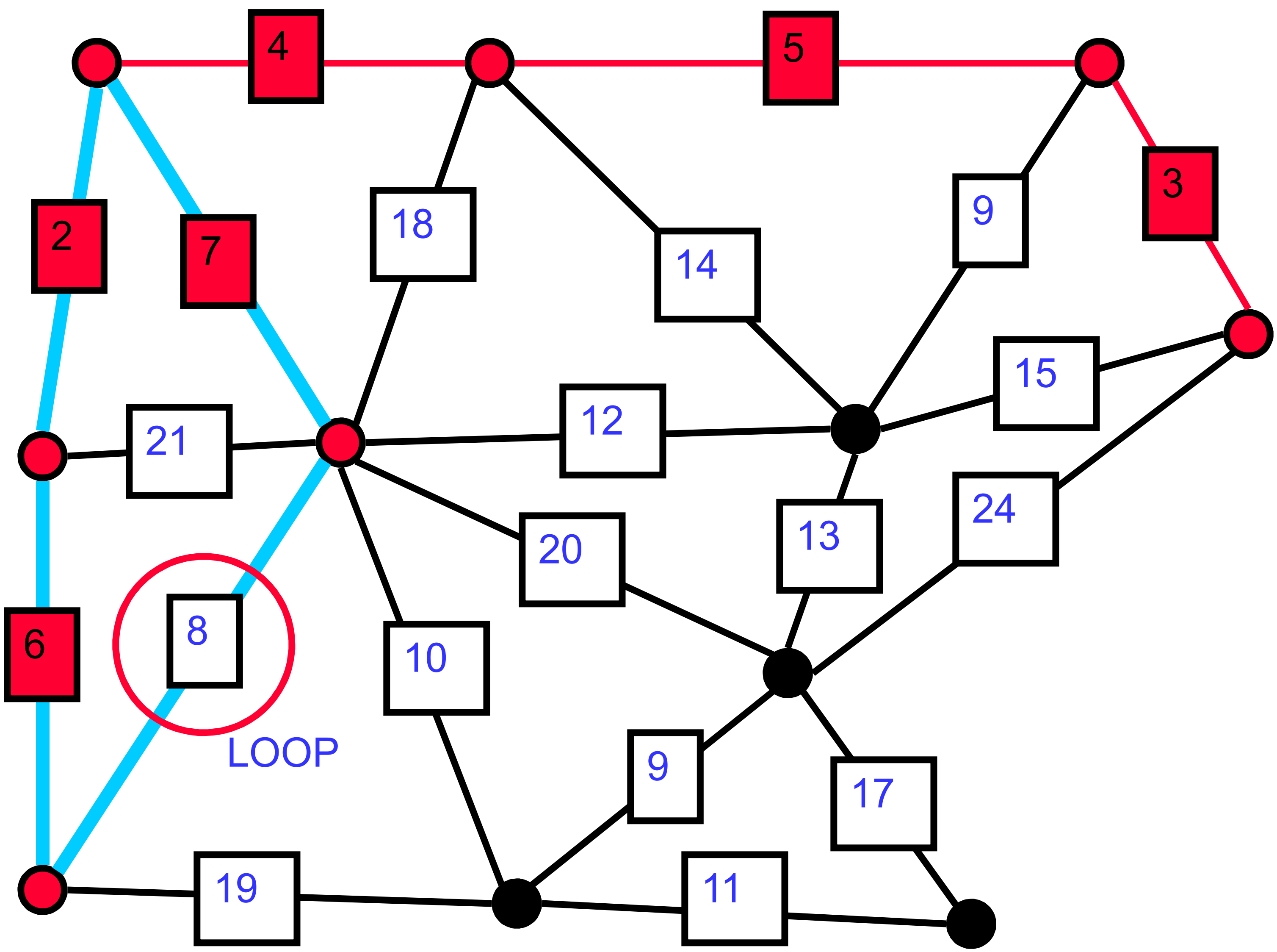


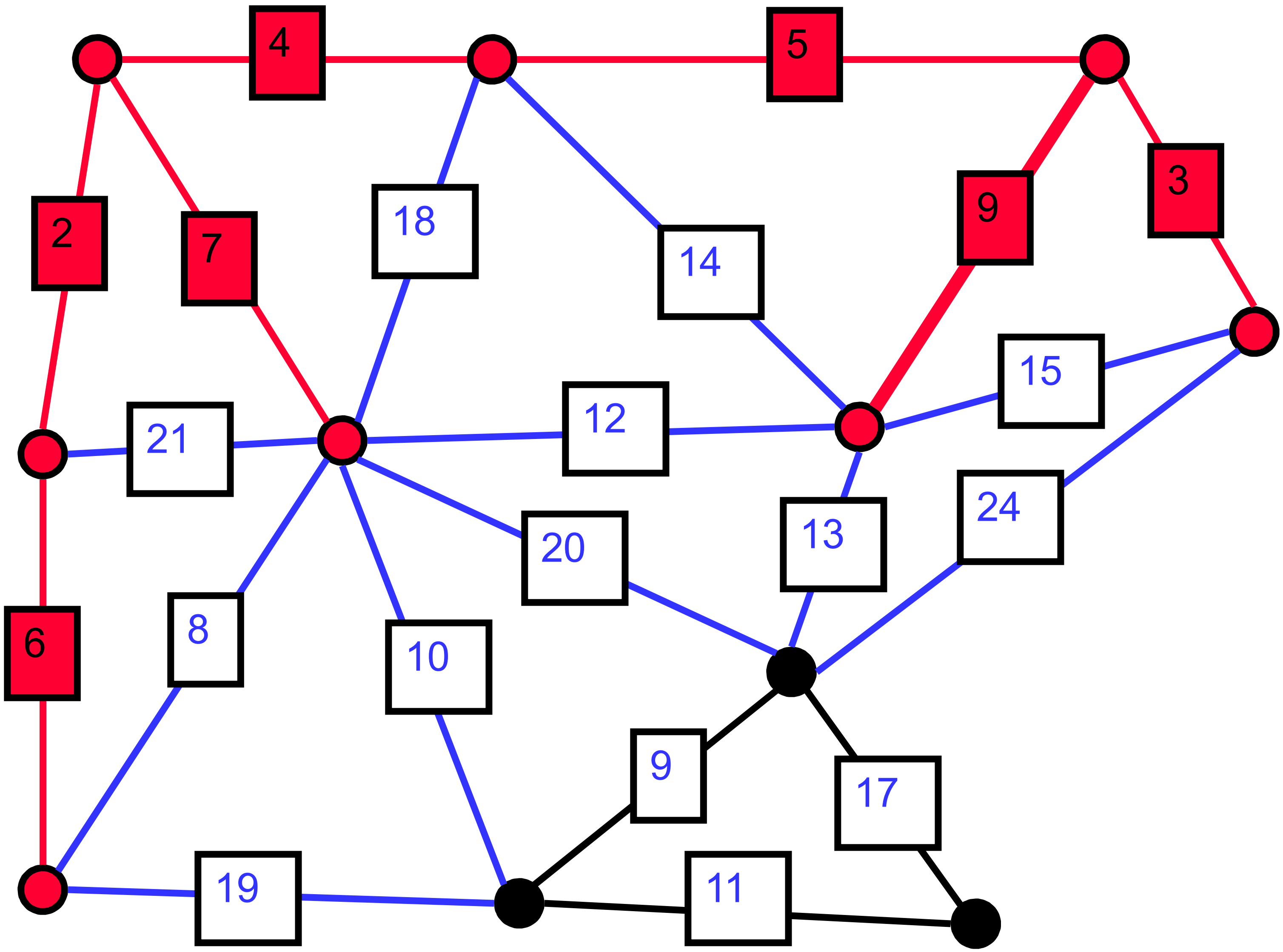
4
5
2
7
18
14
9
3
15
21
12
24
20
13
6
8
10
9
17
19
11

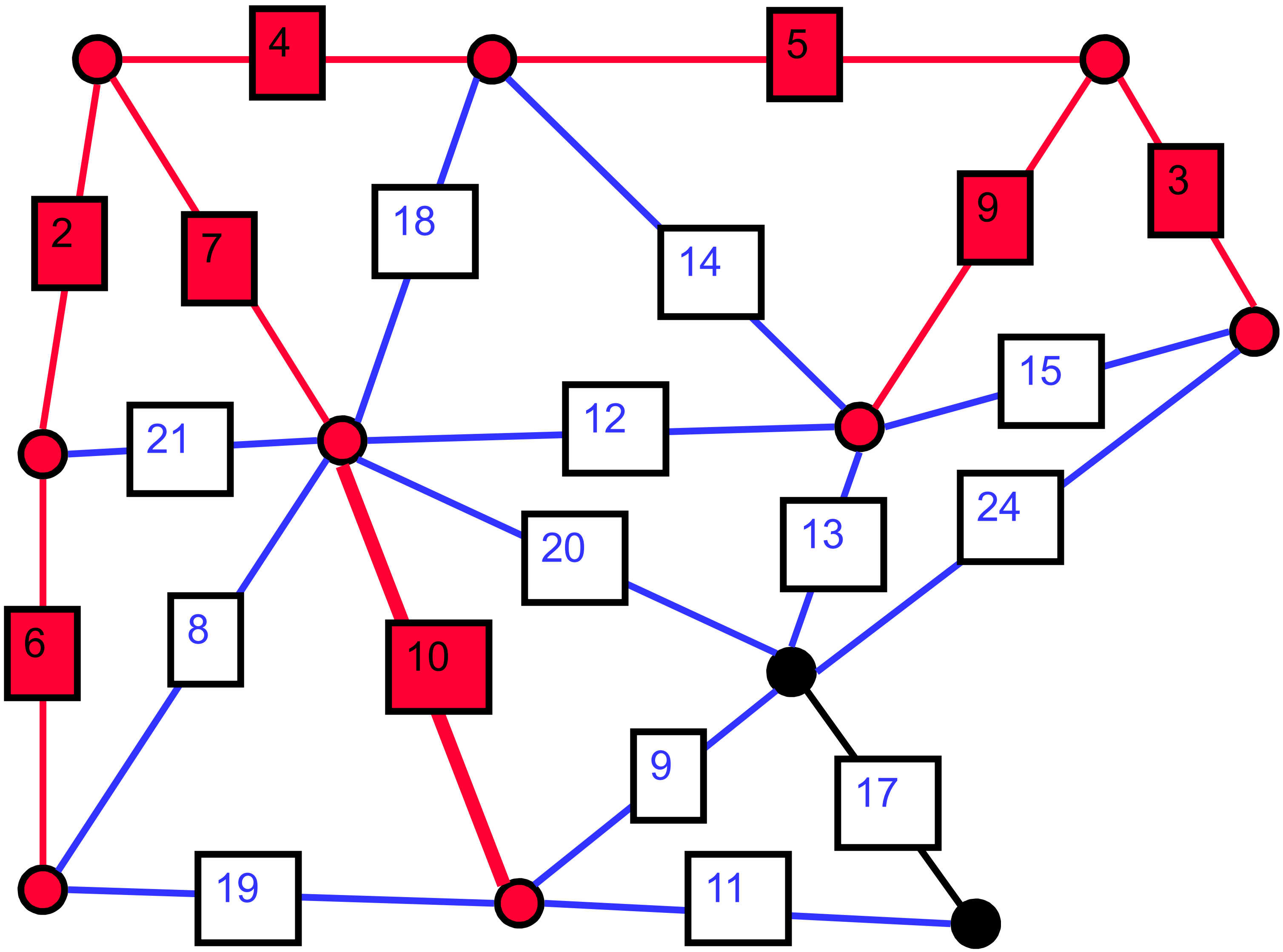

4
5
2
7
18
14
9
3
15
21
12
24
20
13
6
8
10
9
17
19
11

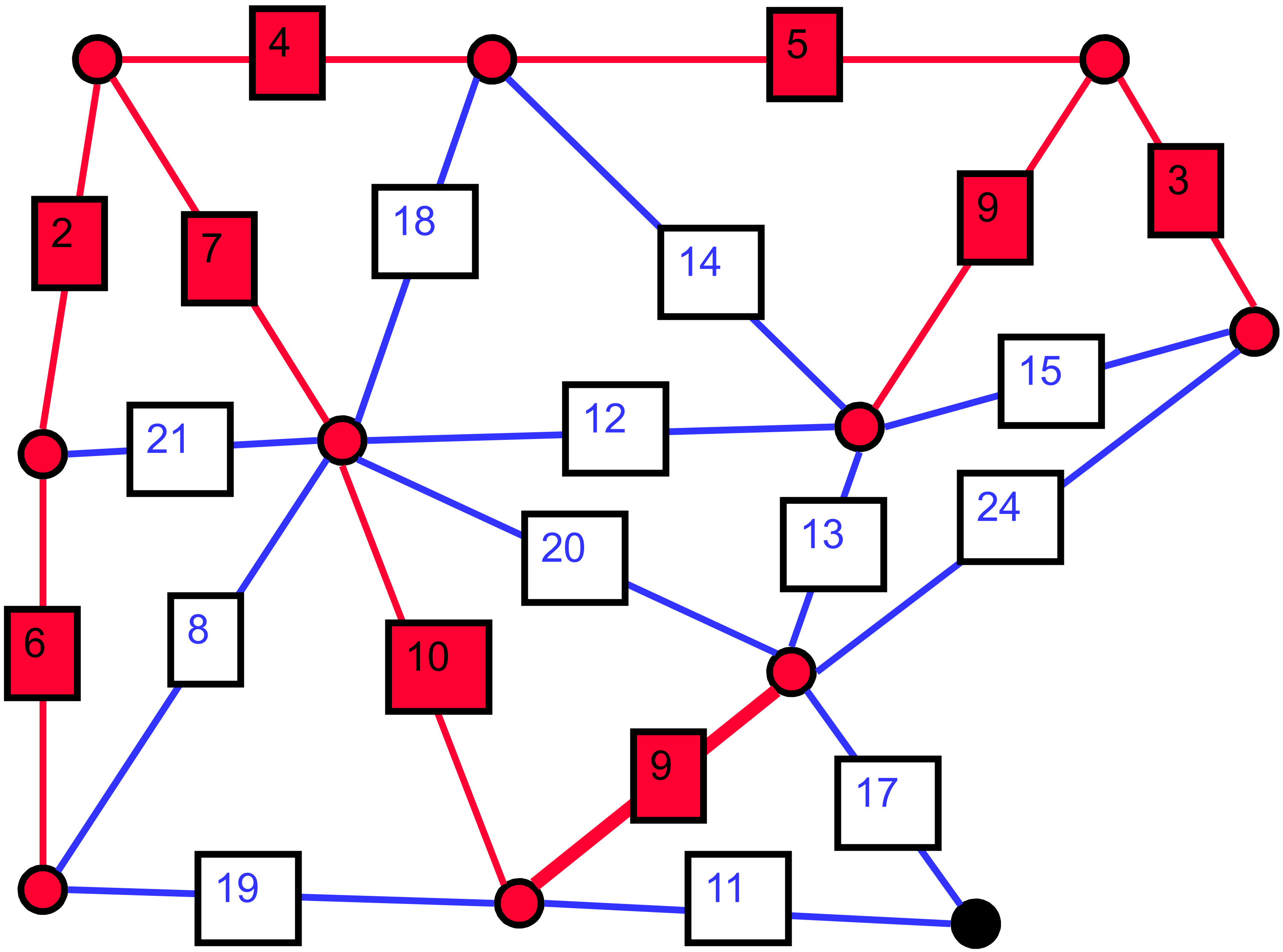

4
5
2
7
18
14
9
3
15
21
12
20
13
24
6
8
10
9
17
19
11

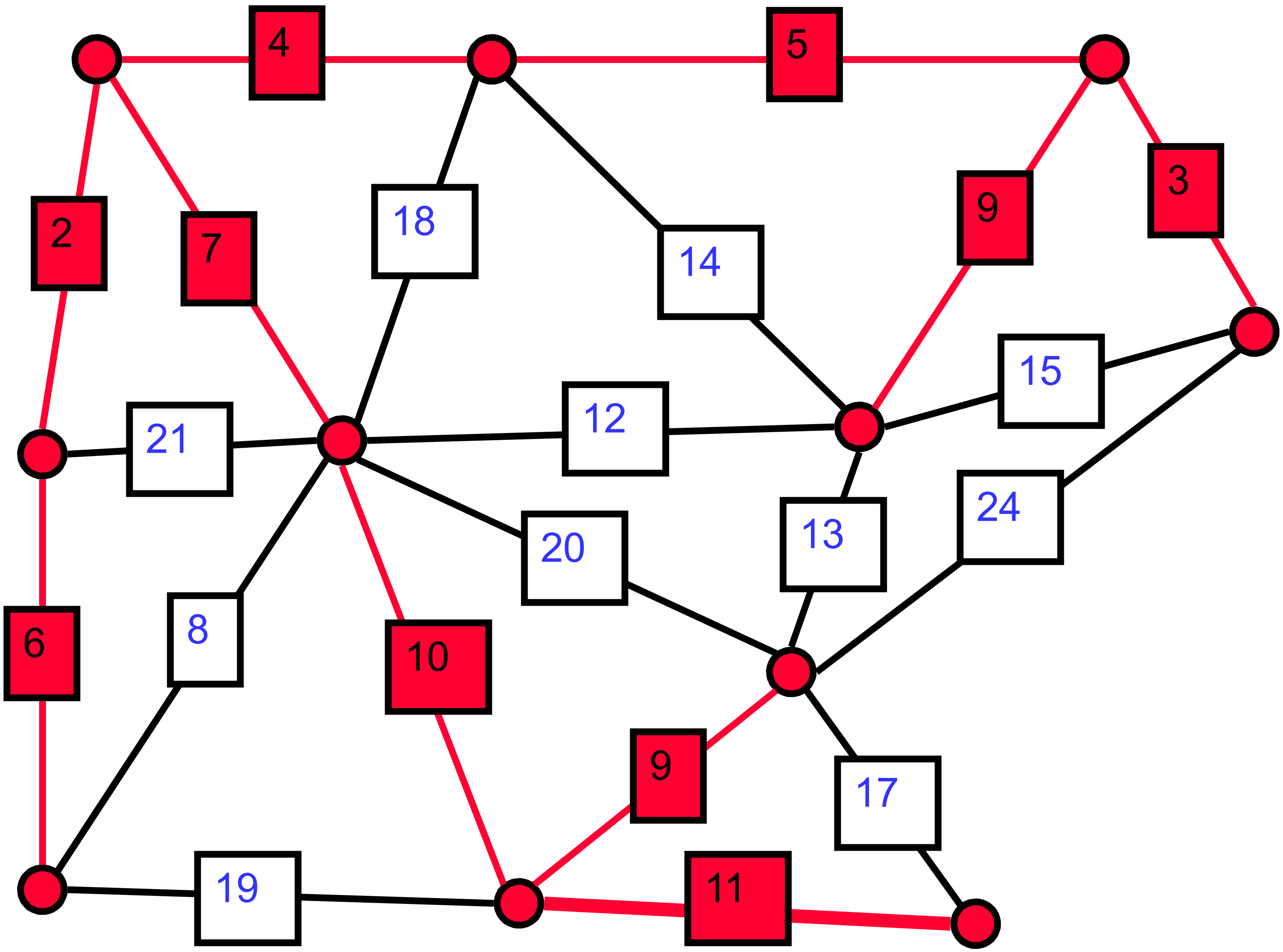

4
5
2
7
18
14
9
3
15
21
12
13
24
20
8
NO!
6
10
9
17
19
11

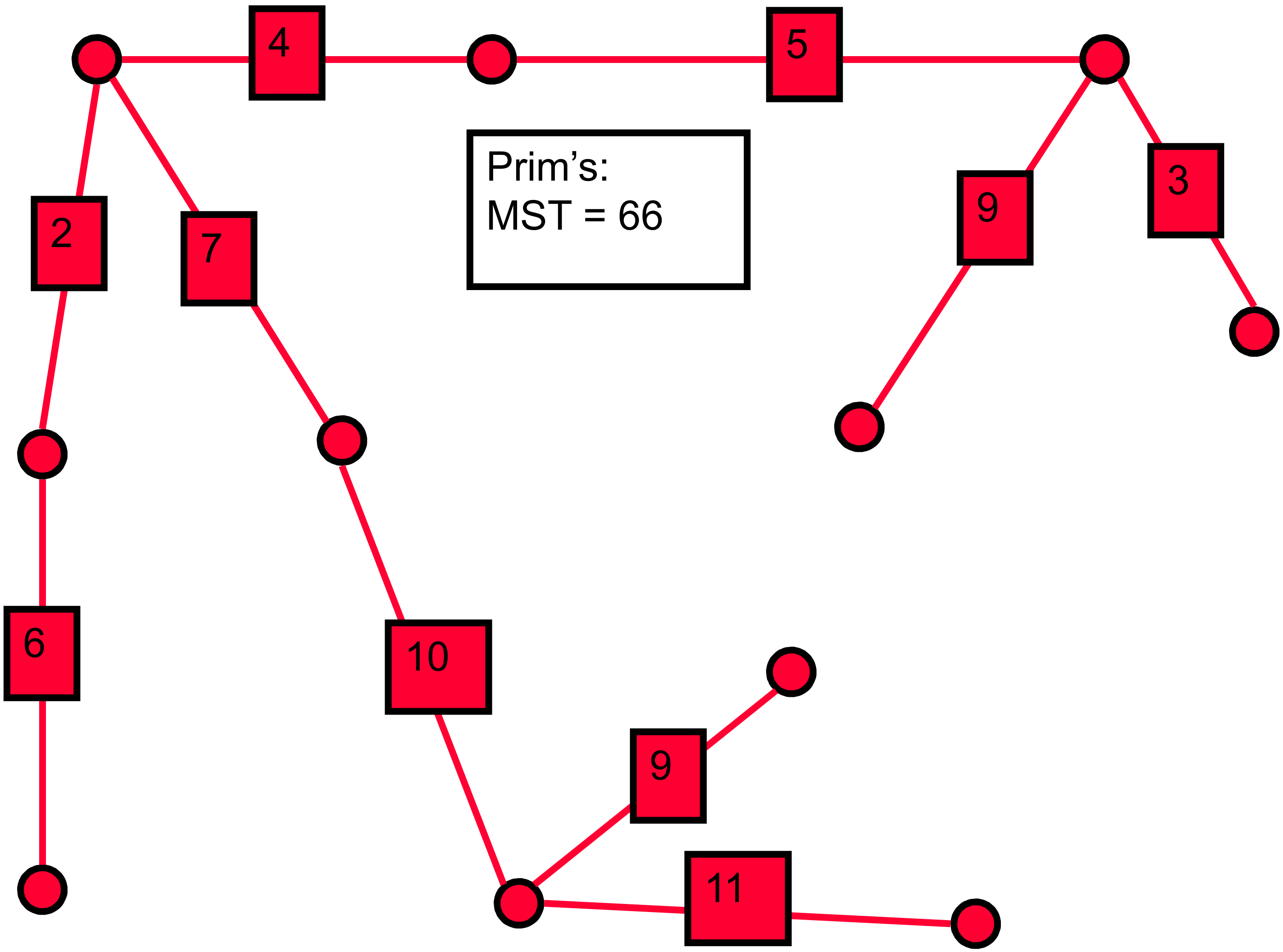

4
5
2
7
18
14
9
3
15
21
12
24
20
13
6
8
10
LOOP
9
17
19
11


4
5
2
7
18
14
9
3
15
21
12
24
20
13
6
8
10
9
17
19
11


4
5
2
7
18
14
9
3
15
21
12
24
20
13
6
8
10
9
17
19
11


4
5
2
7
18
14
9
3
15
21
12
24
13
20
6
8
10
9
17
19
11

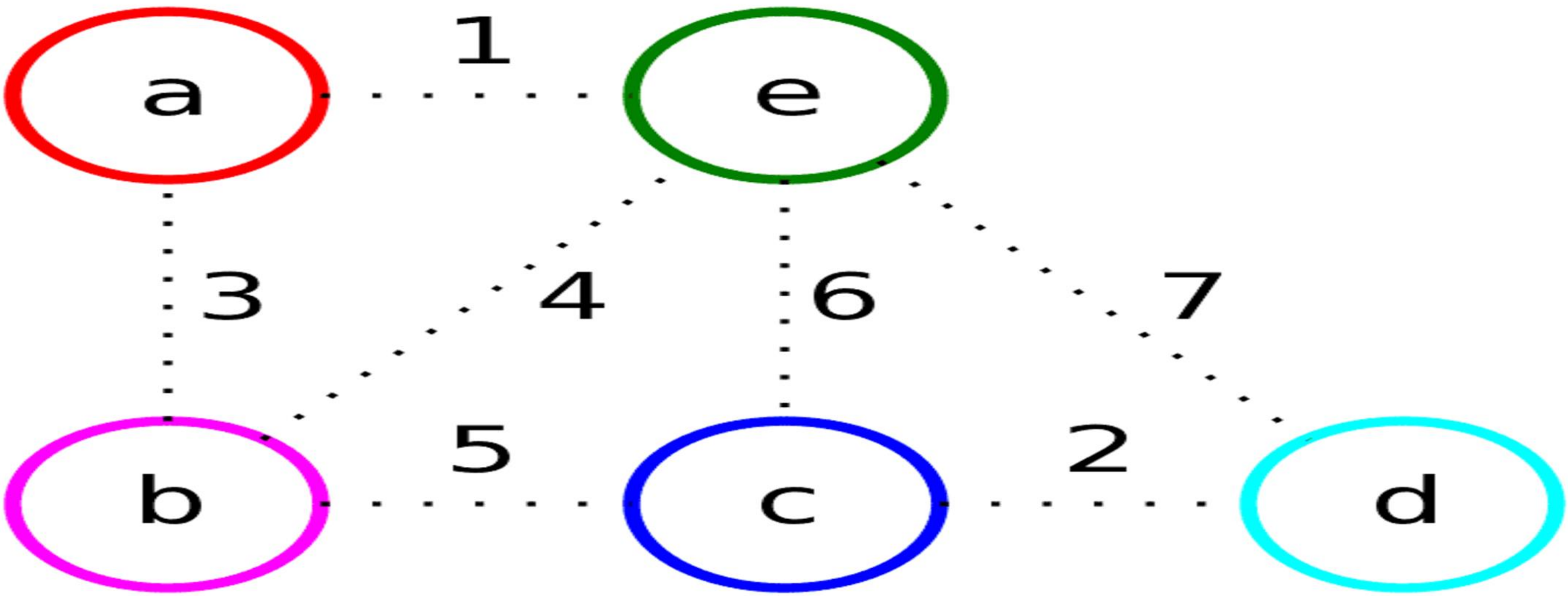

4
5
2
7
18
14
9
3
15
21
12
20
13
24
6
8
10
9
17
19
11



Prim's:
MST = 66

An Alternate Algorithm: Kruskal's Algorithm

Edge	ab	ae	bc	be	cd	ed	ec
Weight	3	1	5	4	2	7	6



Kruskal's Algorithm

Sort edges in graph in increasing order

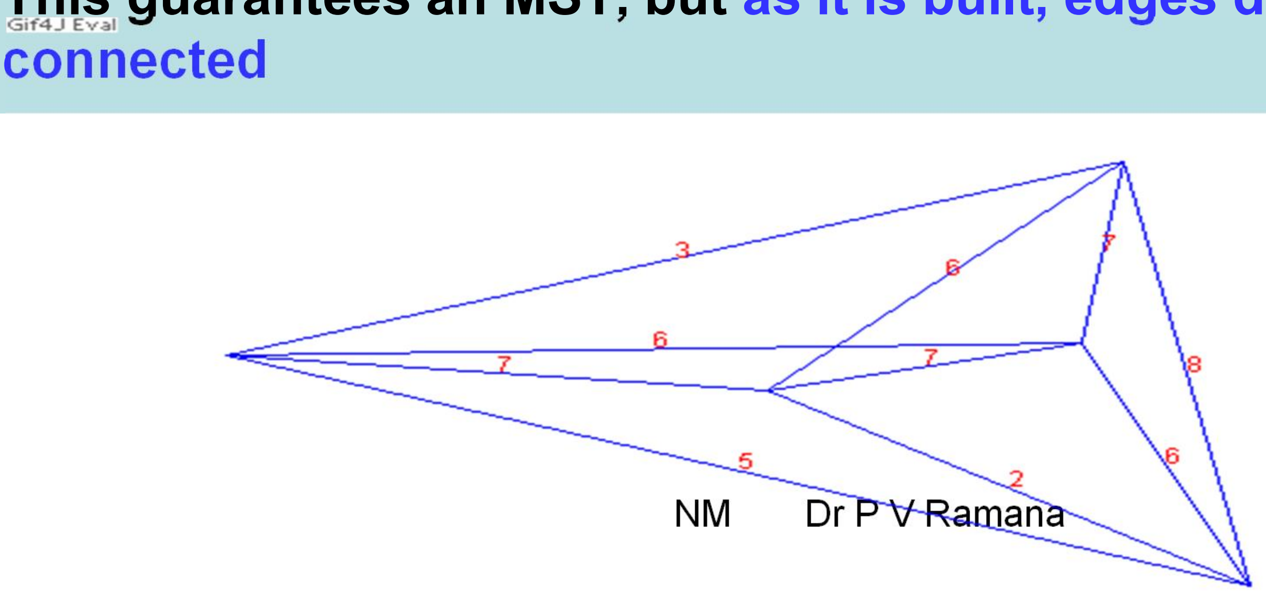
Select the shortest edge

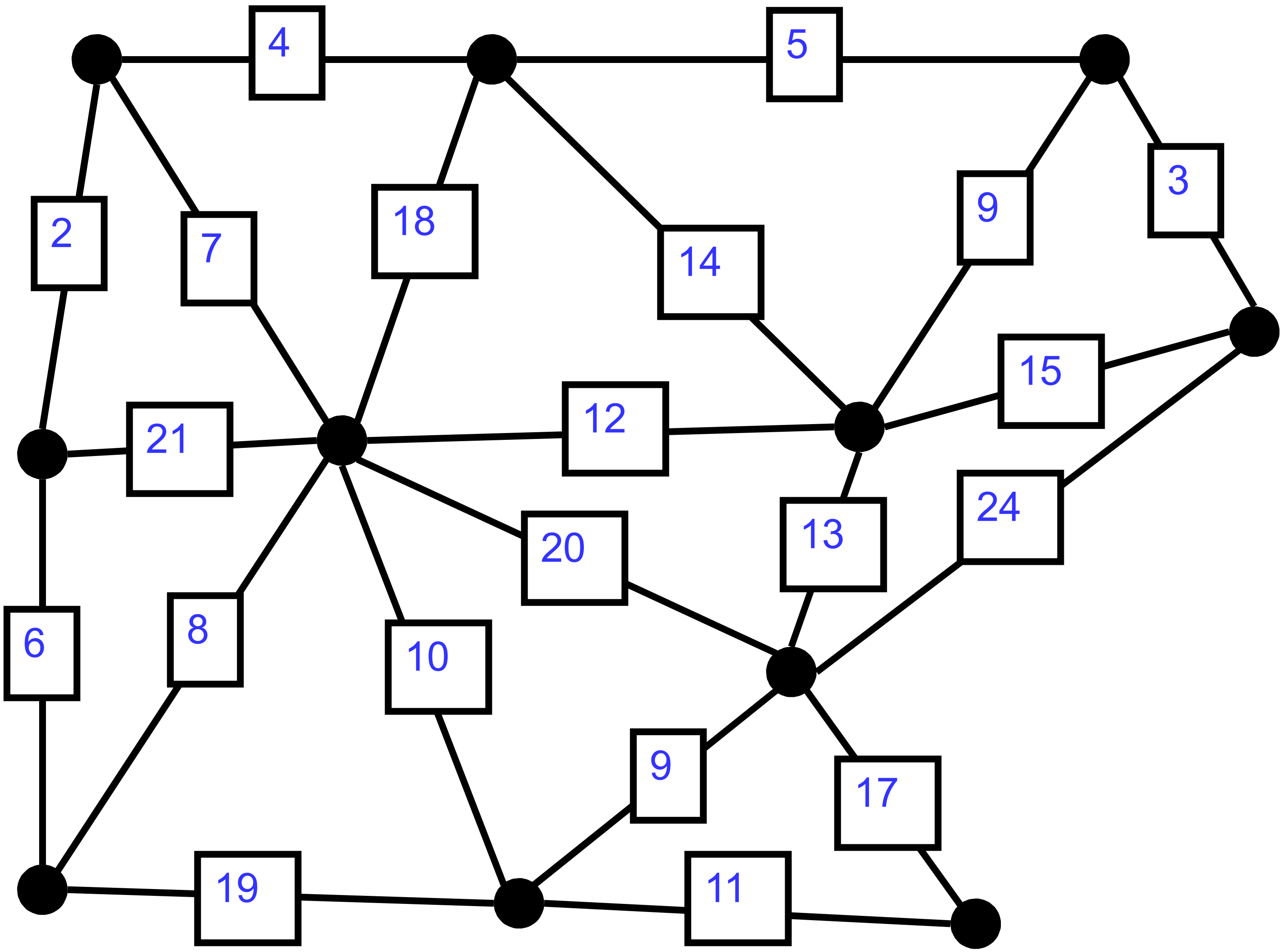
Loop

- Exitif all edges examined
- Select next shortest edge (if no cycle created)
- Mark this edge as examined

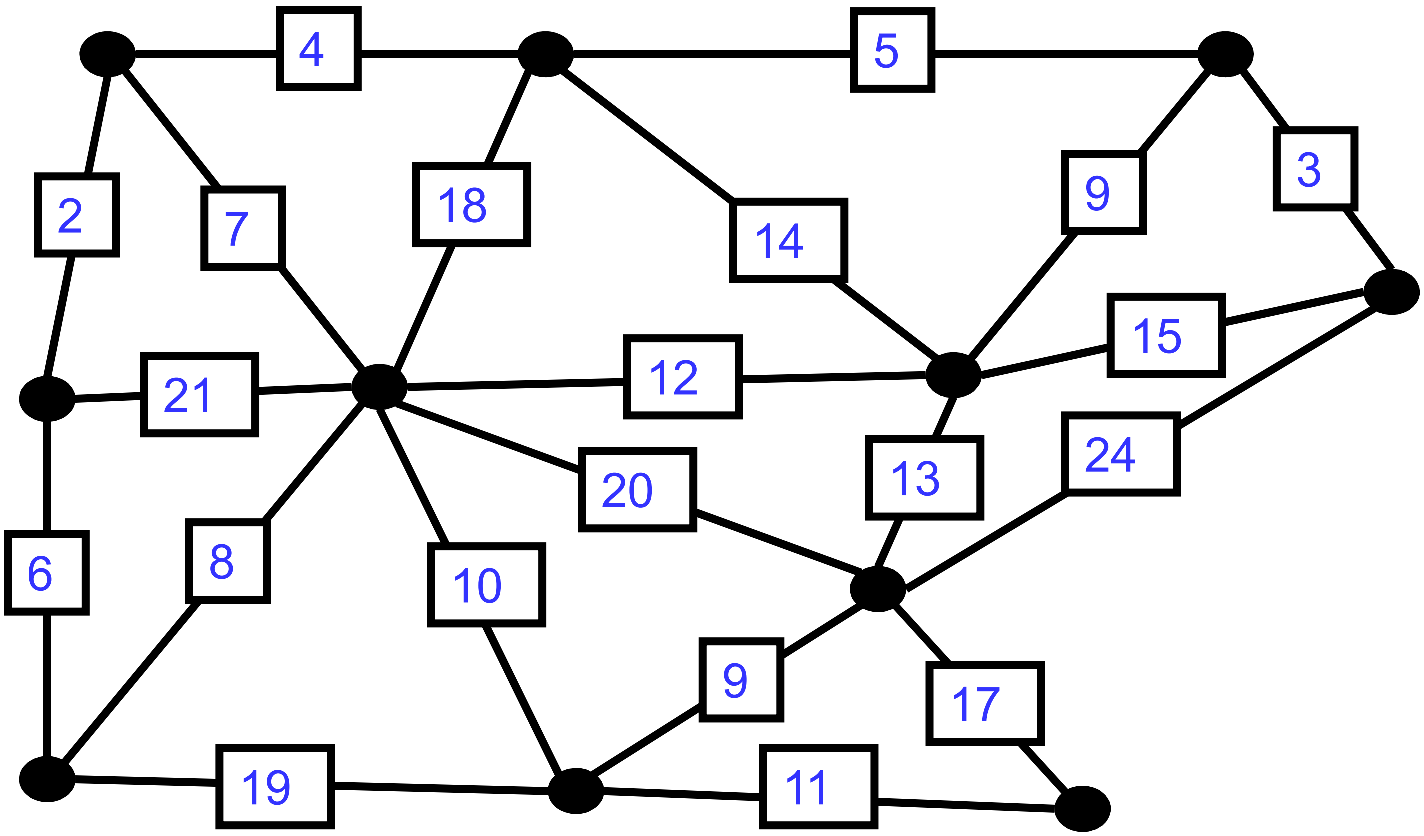
Endloop

This guarantees an MST, but as it is built, edges do not have to be connected



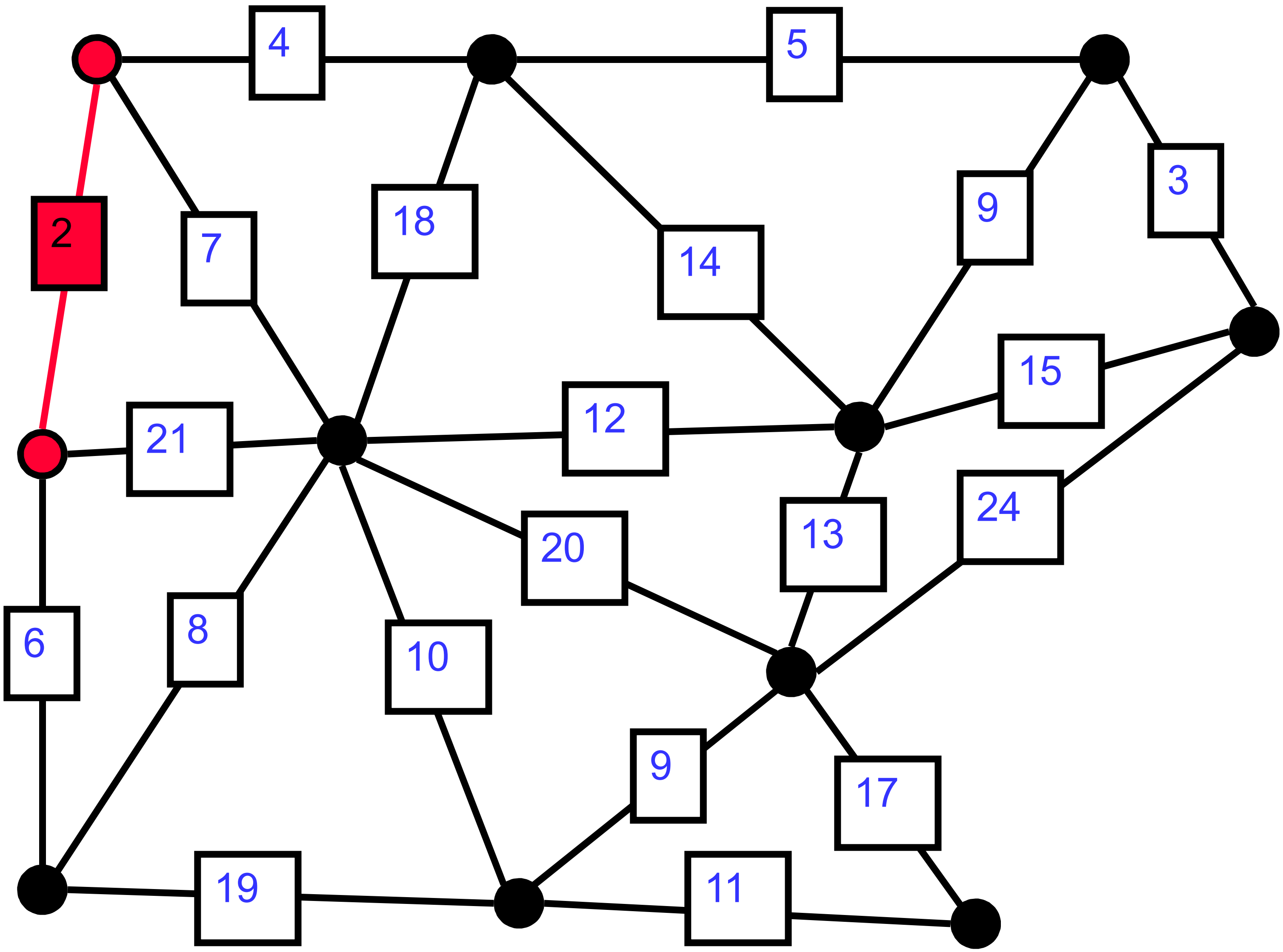

4
5
2
7
18
14
9
3
15
21
12
24
13
20
6
8
10
9
17
19
11

Sort the Edges

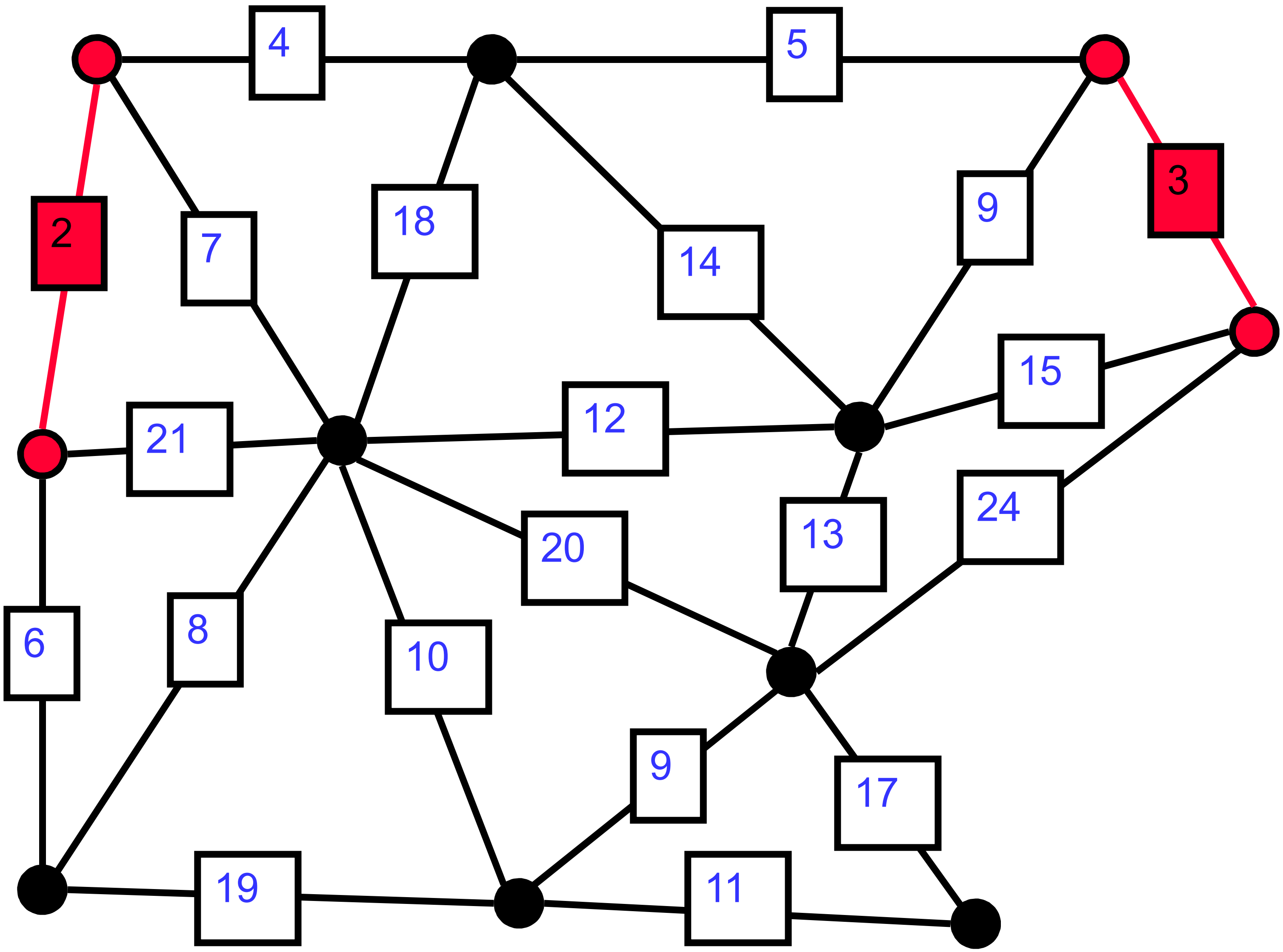


2, 3, 4, 5, 6, 7, 8, 9, 9, 10, 11, 12, 13. 14. 15. 17. 18. 19. 20. 21. 24

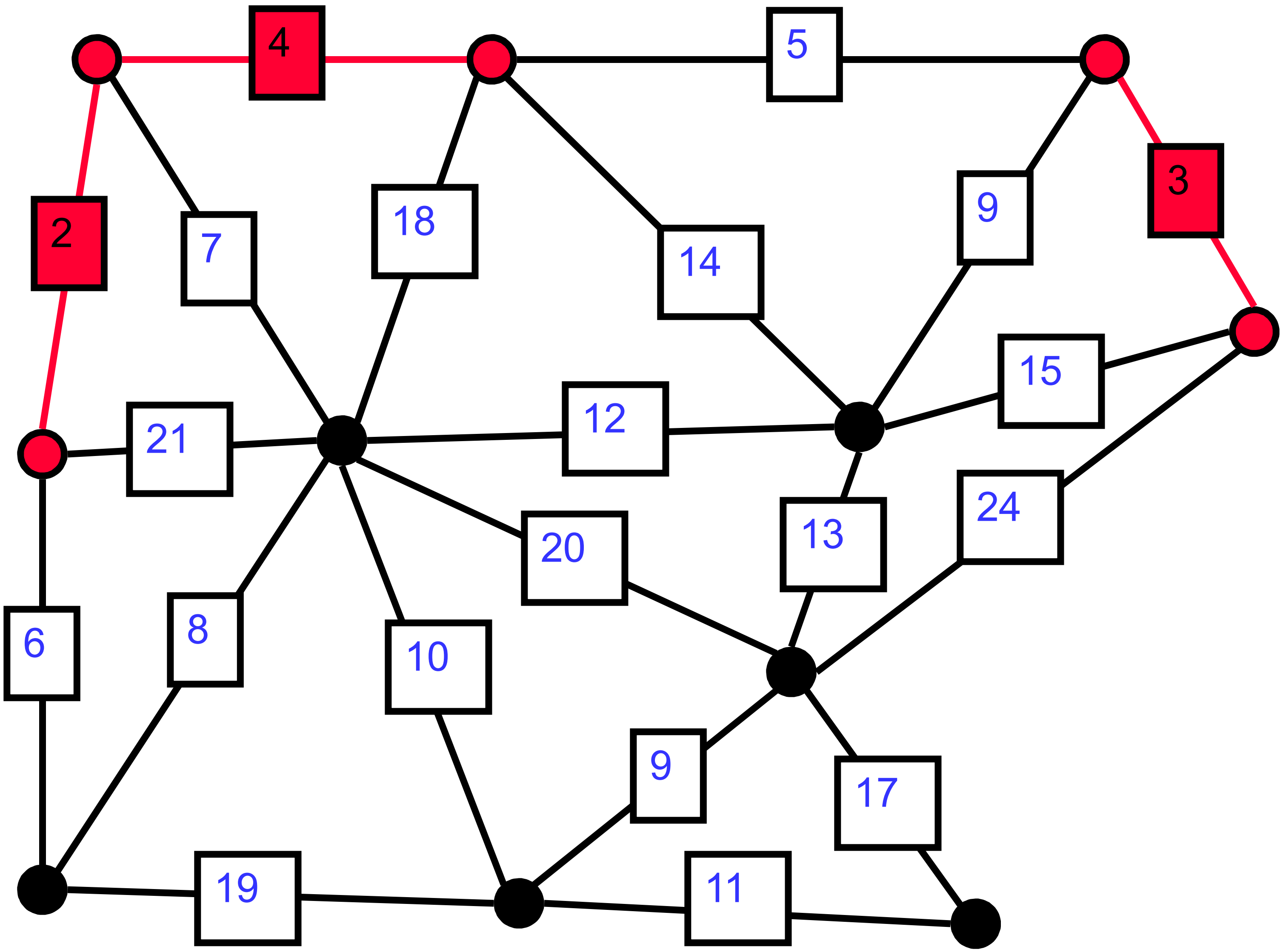
2, 3, 4, 5, 6, 7, 8, 9, 9, 10, 11, 12, 13, 14, 15, 17, 18, 19, 20, 21, 24



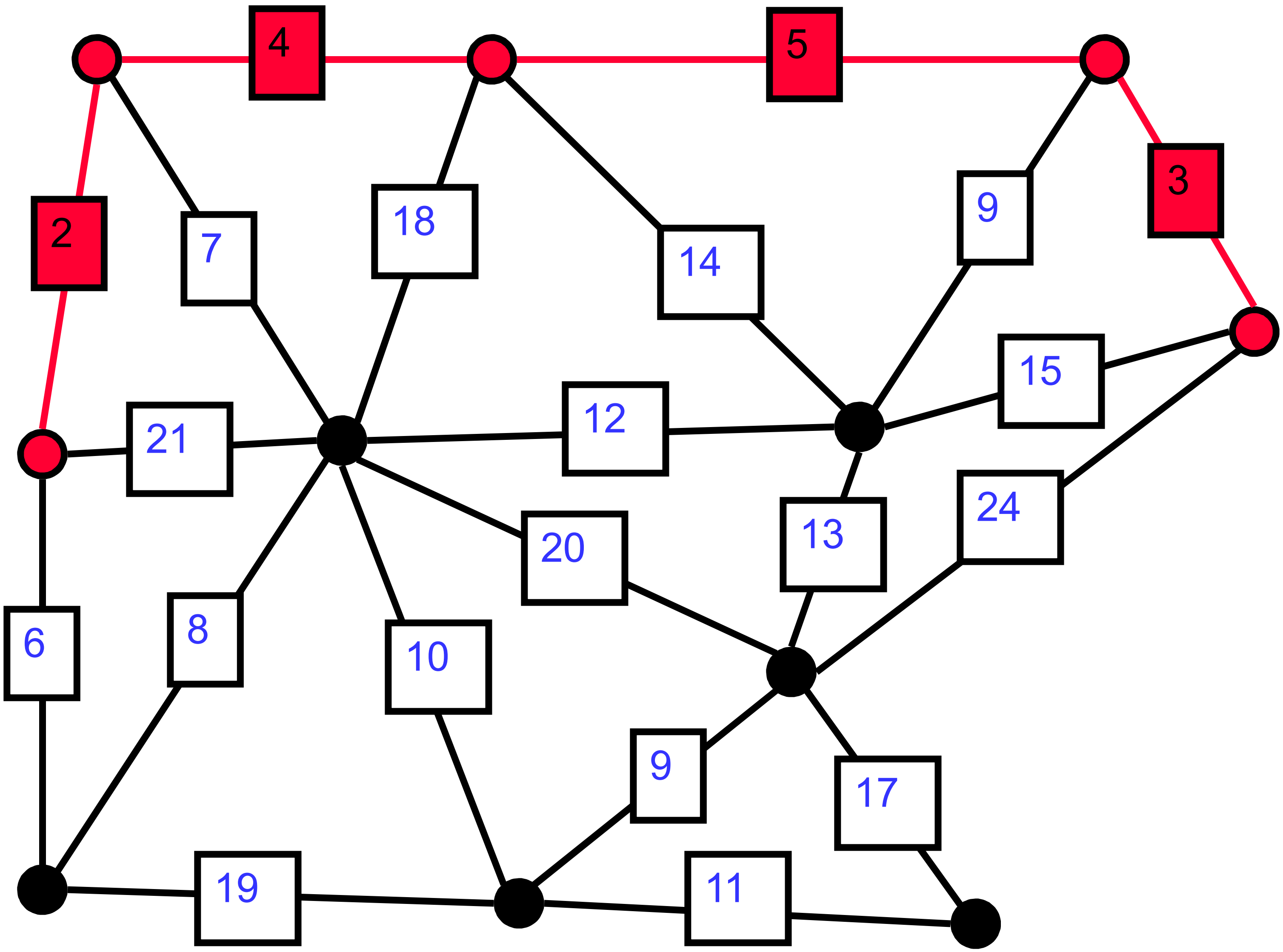
2, 3, 4, 5, 6, 7, 8, 9, 9, 10, 11, 12, 13, 14, 15, 17, 18, 19, 20, 21, 24



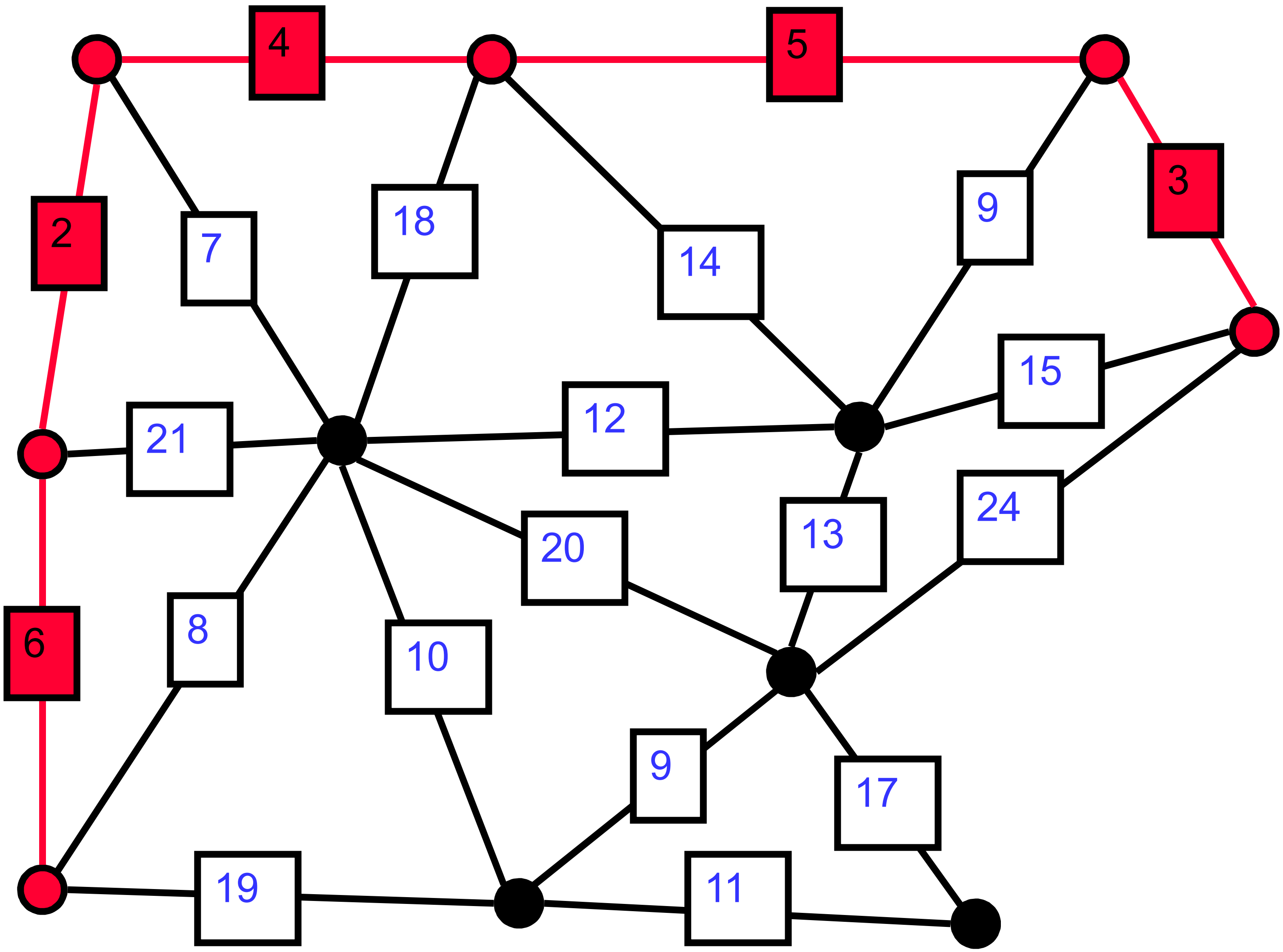
2, 3, 4, 5, 6, 7, 8, 9, 9, 10, 11, 12, 13, 14, 15, 17, 18, 19, 20, 21, 24



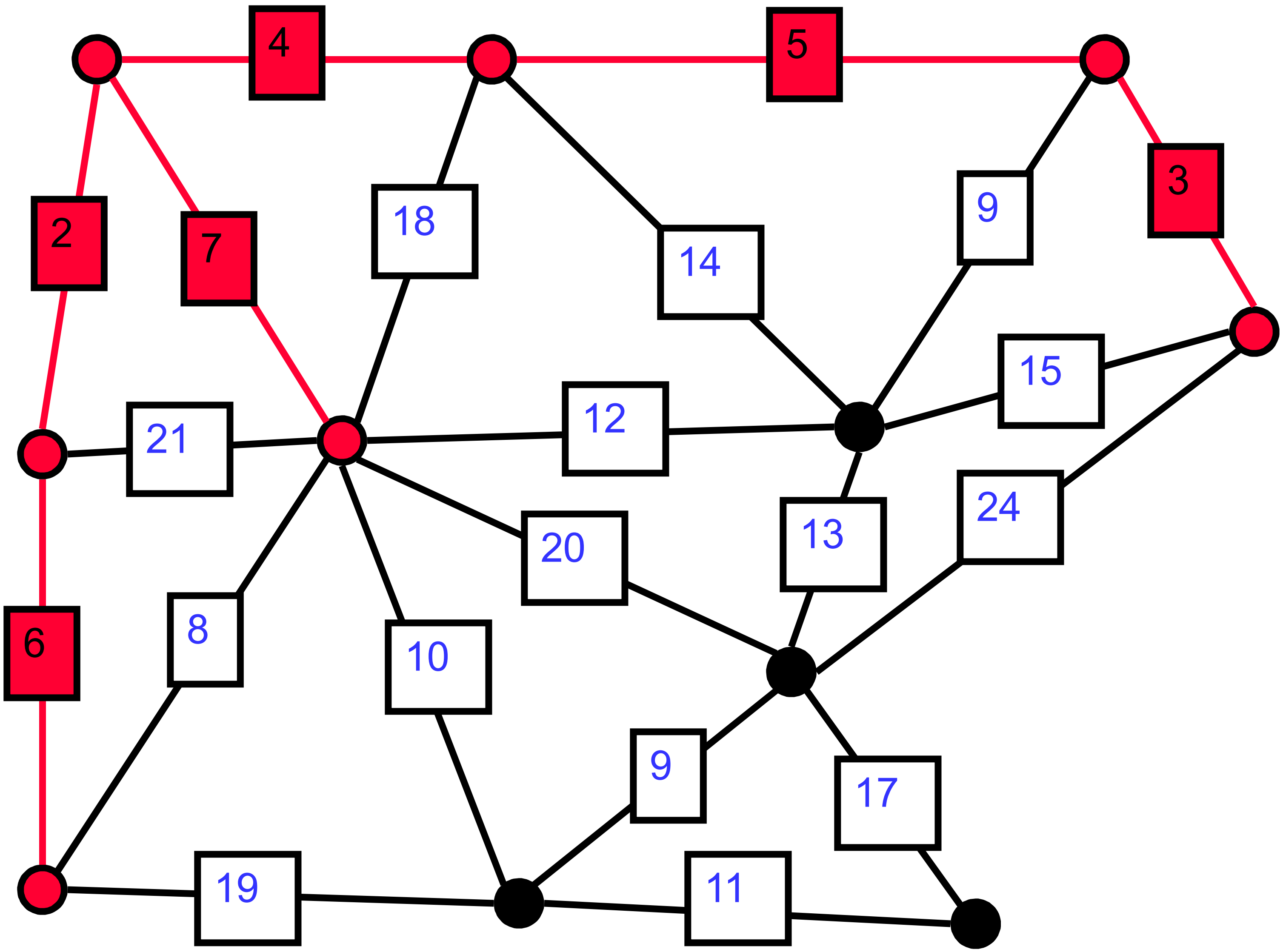
2, 3, 4, 5, 6, 7, 8, 9, 9, 10, 11, 12, 13, 14, 15, 17, 18, 19, 20, 21, 24



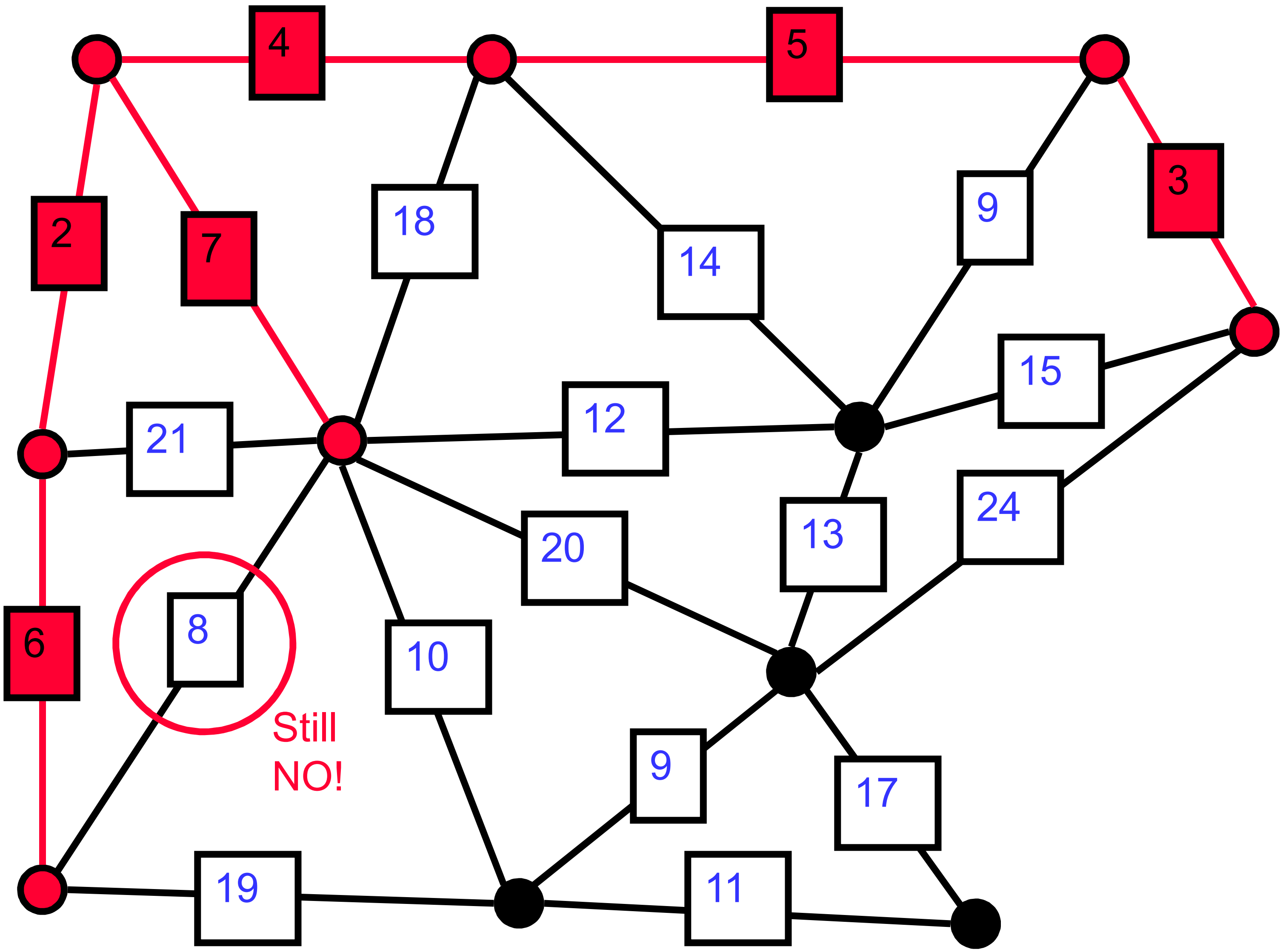
2, 3, 4, 5, 6, 7, 8, 9, 9, 10, 11, 12, 13, 14, 15, 17, 18, 19, 20, 21, 24



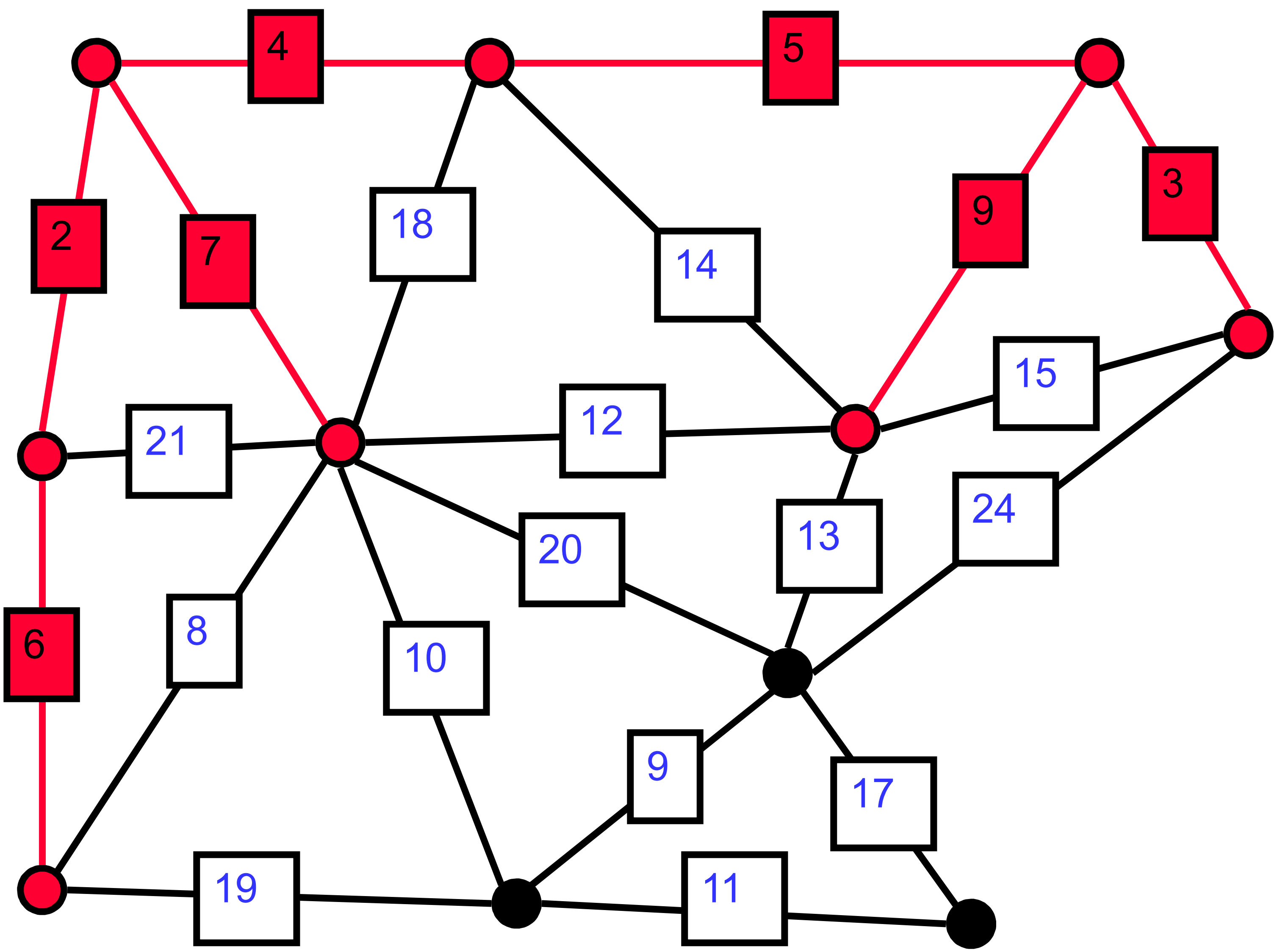
2, 3, 4, 5, 6, 7, 8, 9, 9, 10, 11, 12, 13, 14, 15, 17, 18, 19, 20, 21, 24



2, 3, 4, 5, 6, 7, 8, 9, 9, 10, 11, 12, 13, 14, 15, 17, 18, 19, 20, 21, 24

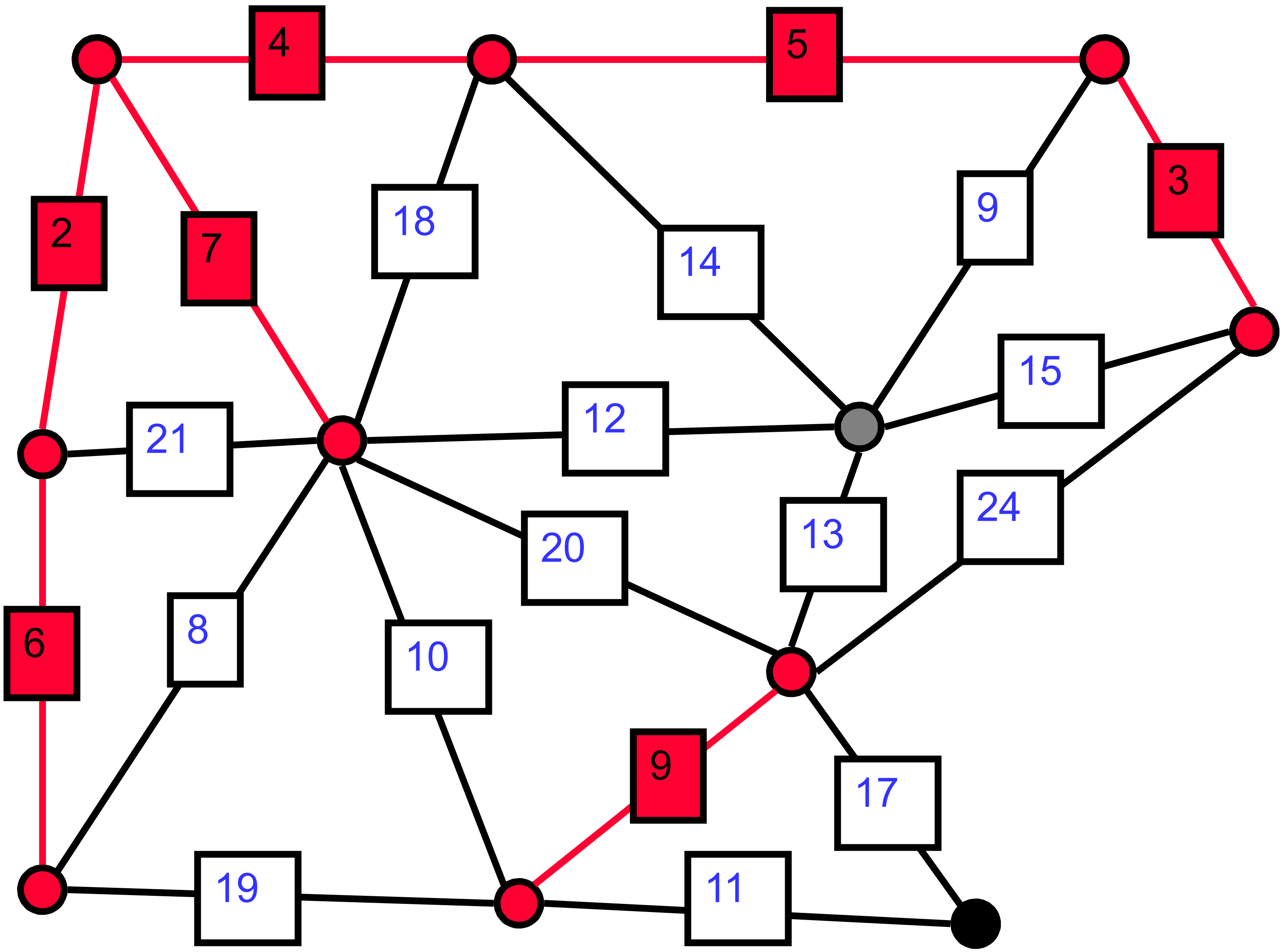


2, 3, 4, 5, 6, 7, 8, 9, 9, 10, 11, 12, 13, 14, 15, 17, 18, 19, 20, 21, 24

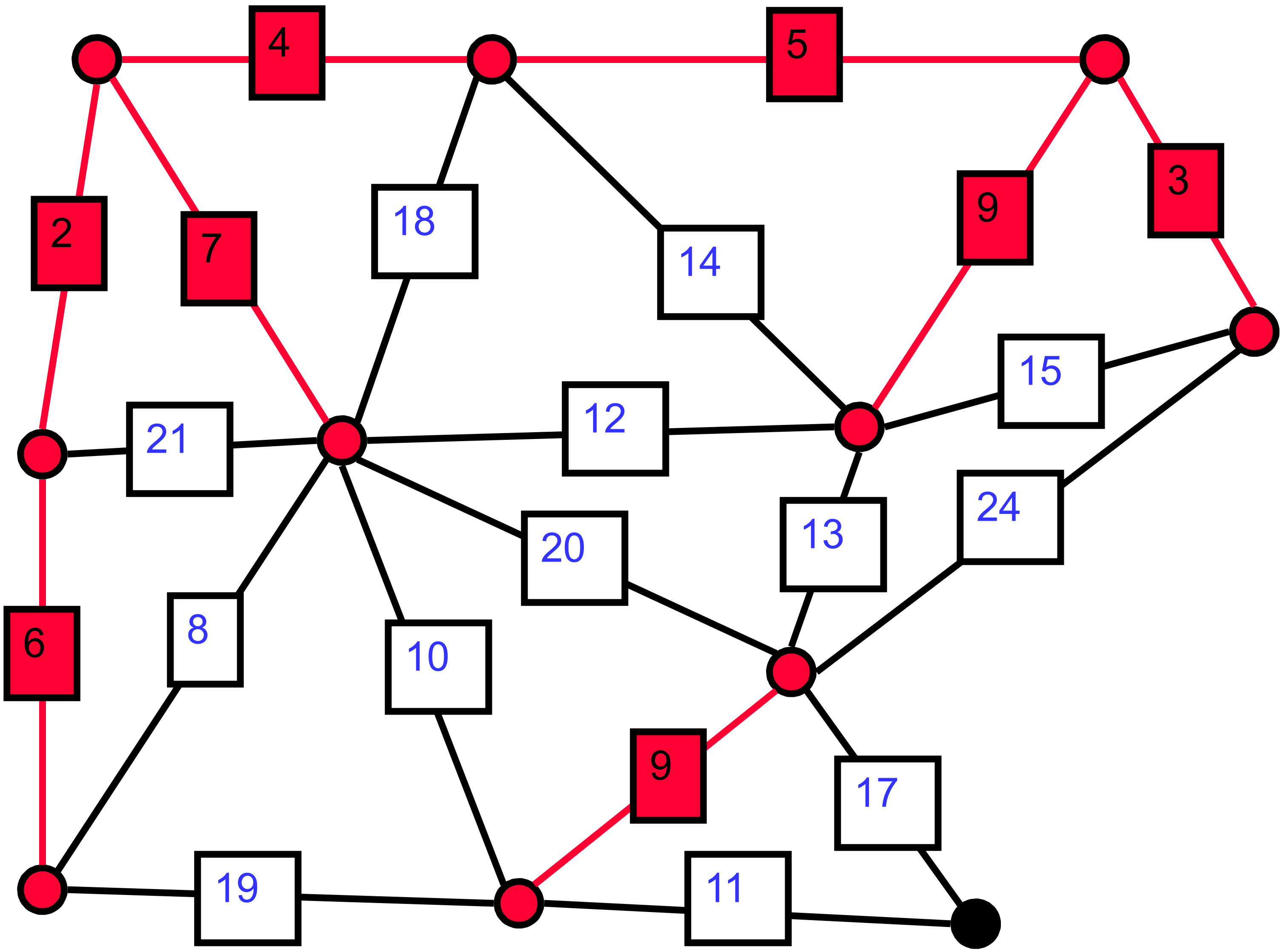


Or

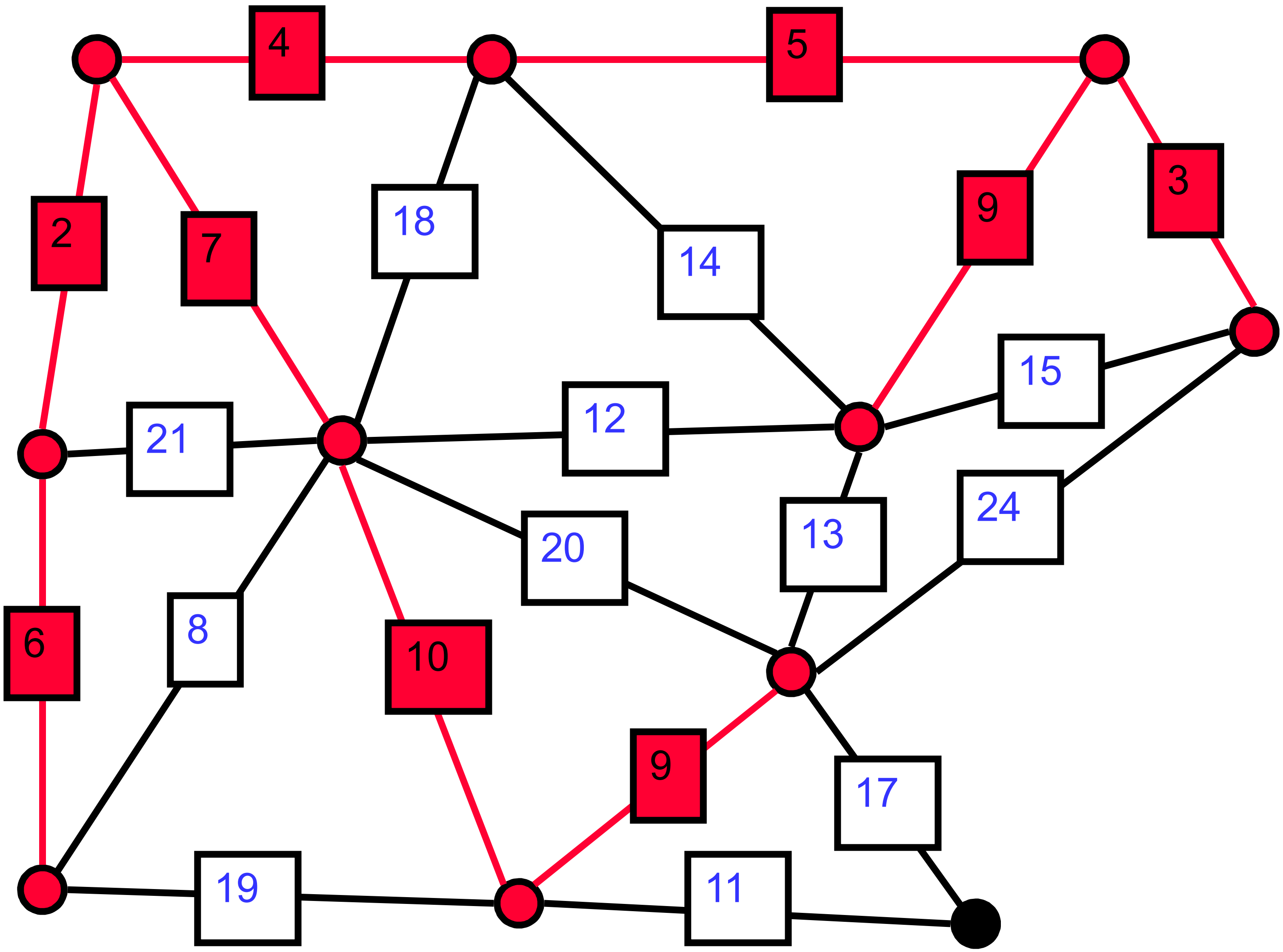
2, 3, 4, 5, 6, 7, 8, 9, 9, 10, 11, 12, 13, 14, 15, 17, 18, 19, 20, 21, 24



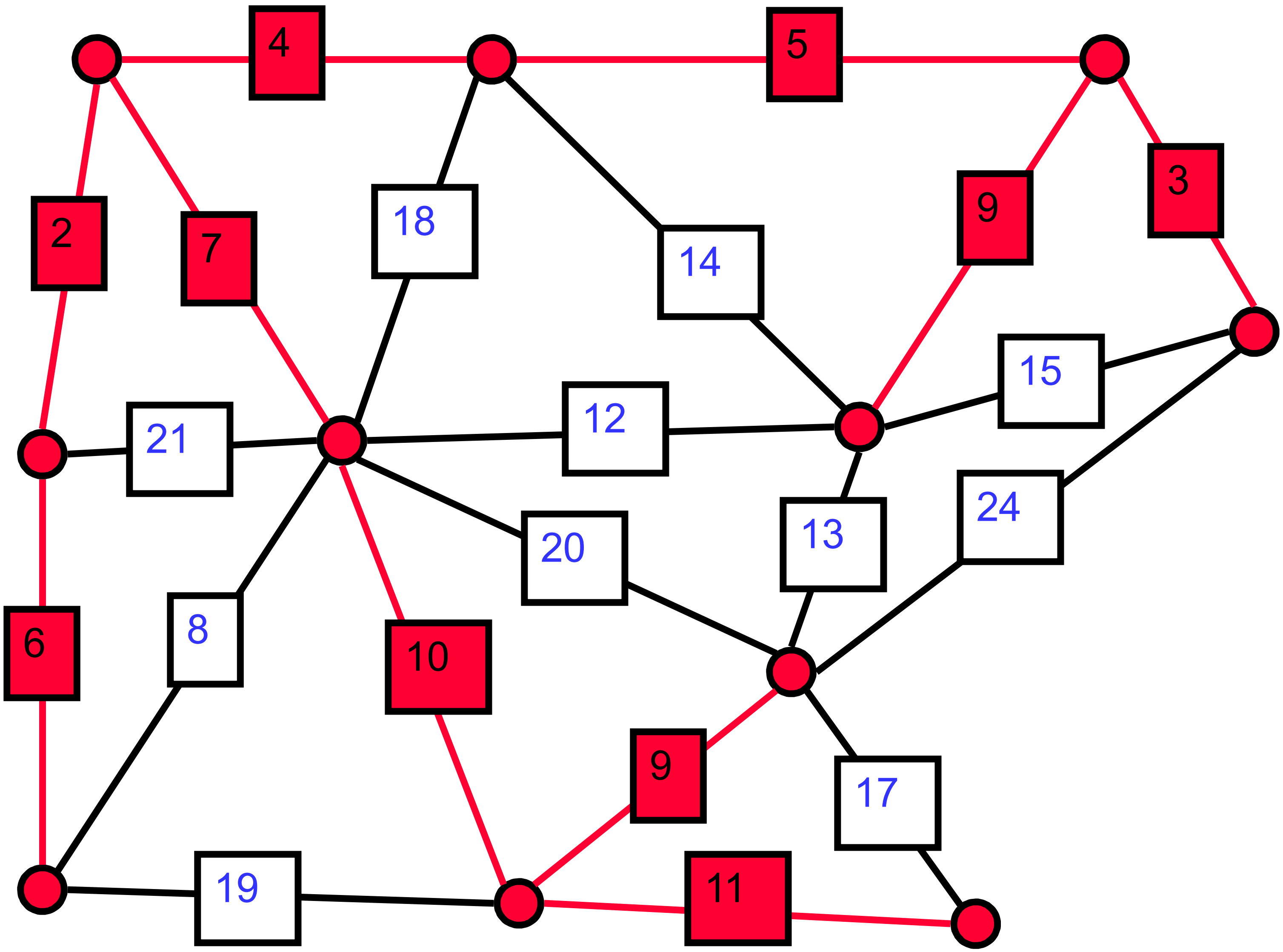
2, 3, 4, 5, 6, 7, 8, 9, 9, 10, 11, 12, 13, 14, 15, 17, 18, 19, 20, 21, 24

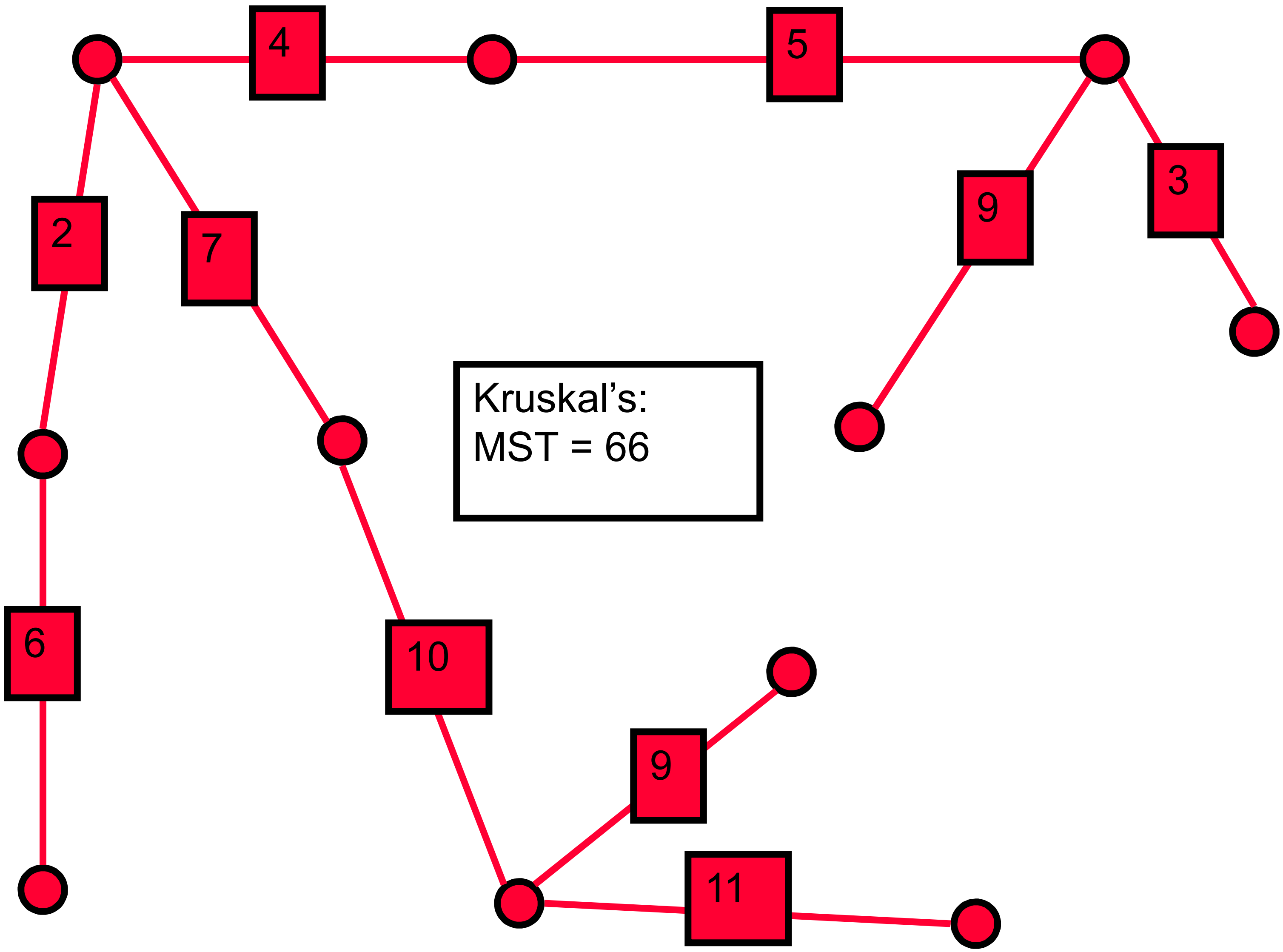


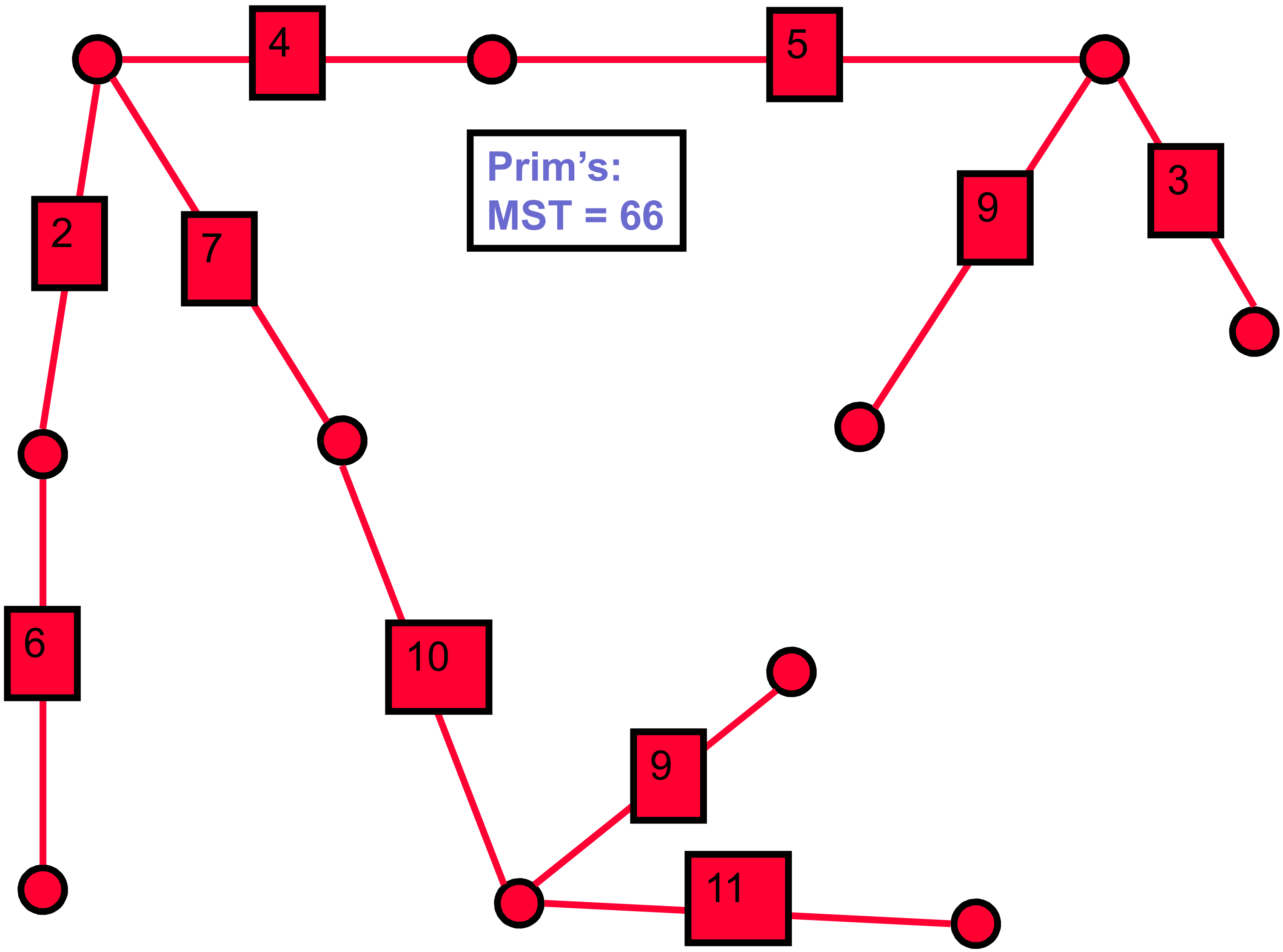
2, 3, 4, 5, 6, 7, 8, 9, 9, 10, 11, 12, 13, 14, 15, 17, 18, 19, 20, 21, 24



2, 3, 4, 5, 6, 7, 8, 9, 9, 10, 11, 12, 13, 14, 15, 17, 18, 19, 20, 21, 24




4
5
2
7
9
3
Kruskal's:
MST = 66
6
10
9
11


4
5
2
7
Prim's:
MST = 66
9
3
6
10
9
11

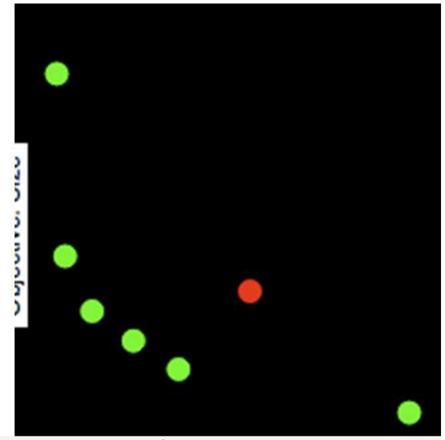
Summary

- **Minimum spanning trees**
  - **Connect all vertices**
  - **With no cycles**
  - **And minimize the total edge cost**

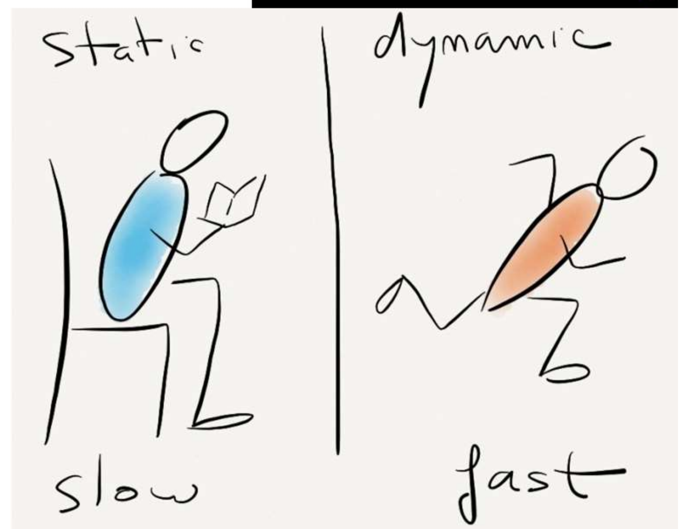
- **Prim's and Kruskal's algorithm**
  - **Generate minimum spanning trees**
  - **Give same total cost, but may give different trees (if graph has edges with same weight)**

Optimization models

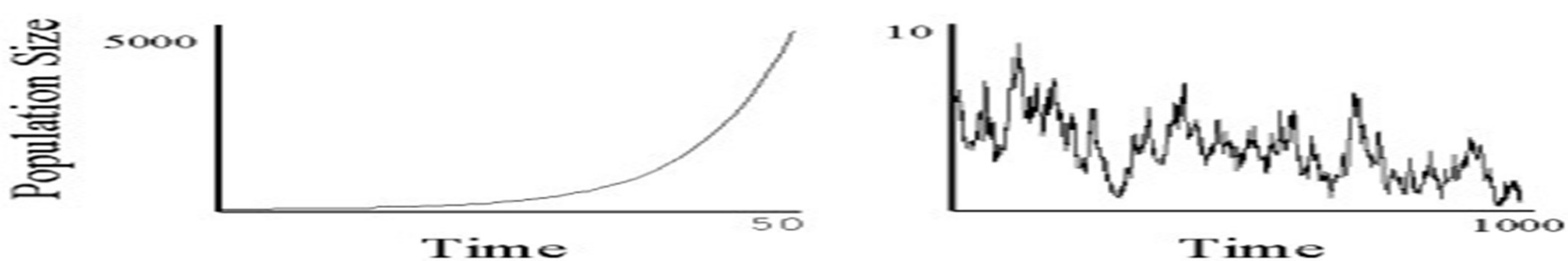
- Single x Multiobjective models



- Static x Dynamic models



- Deterministic x Stochastic models



Problem specification

Suppose have a cost function (or objective function)

$$f(\mathbf{x}) : \mathbb{R}^N \longrightarrow \mathbb{R}$$

Our aim is to find values of the parameters (decision variables) $\mathbf{x}$ that minimize this function

$$\mathbf{x}^* = \arg\min_{\mathbf{x}} f(\mathbf{x})$$

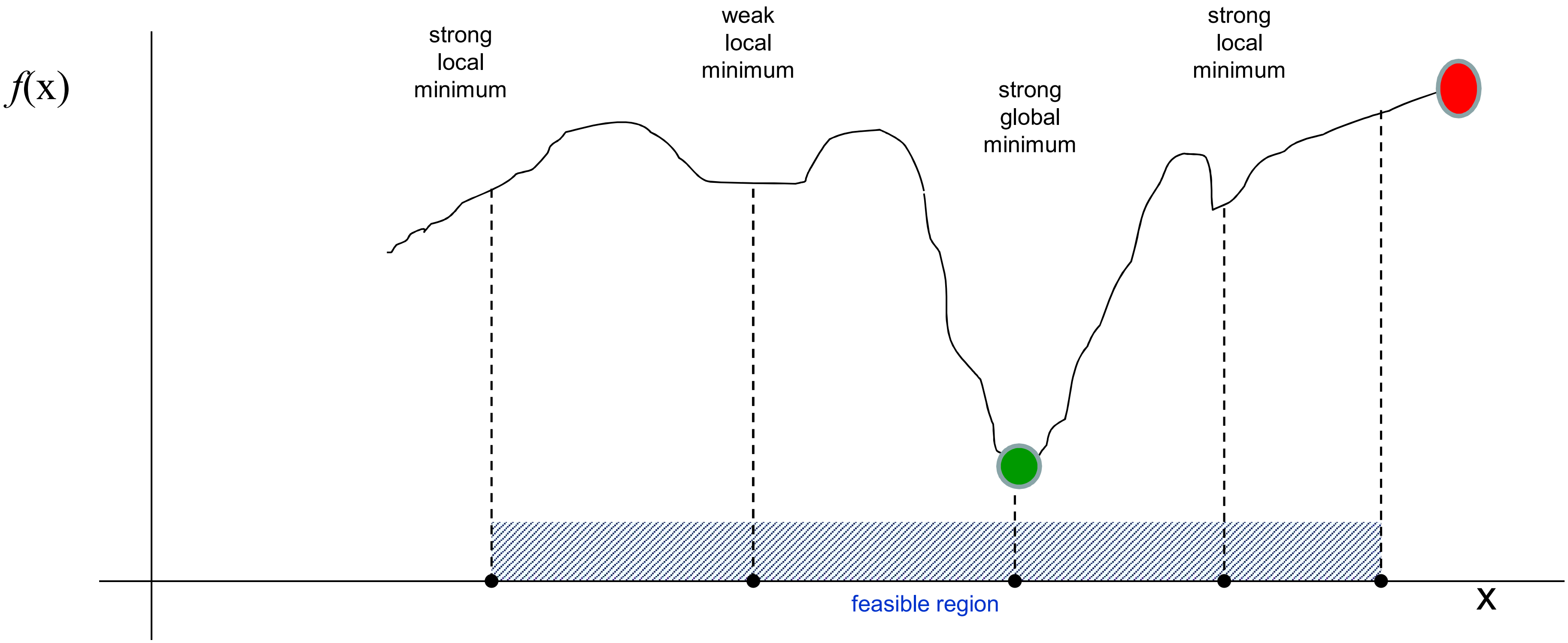
Subject to the following constraints:

Unconstraint

- equality: $$c_i(\mathbf{x}) = 0$$
- nonequality: $$c_j(\mathbf{x}) \geq 0$$

If seek a maximum of $f(\mathbf{x})$ (profit function) it is equivalent to seeking a minimum of $-f(\mathbf{x})$

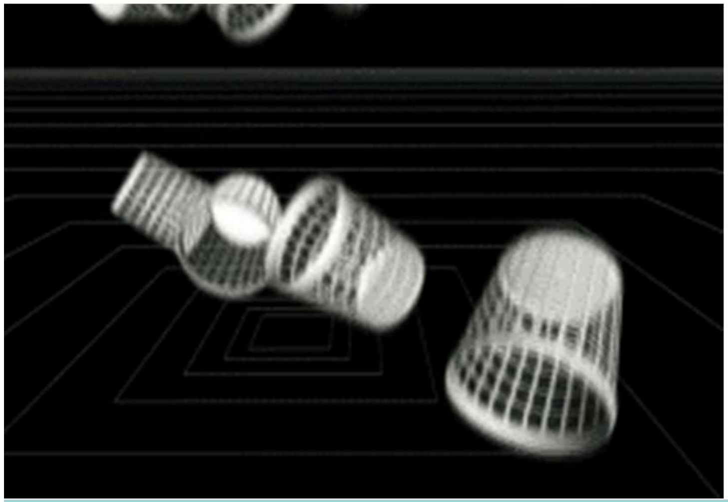
Types of minima

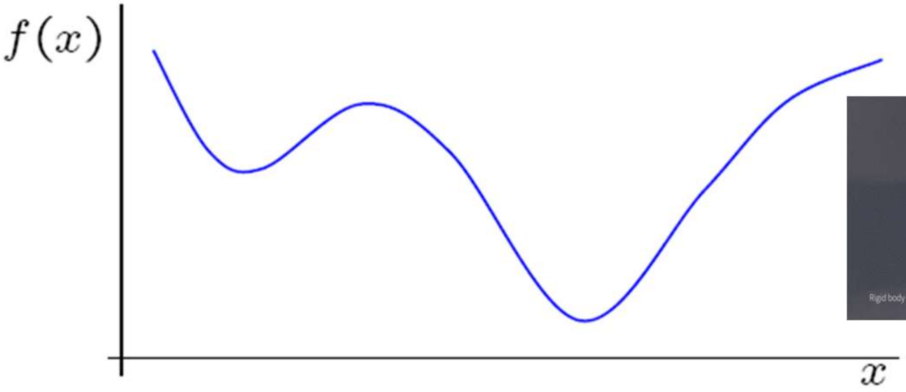


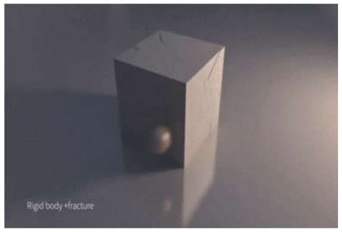
- which of the minima is found depends on the starting point
- such minima often occur in real applications

Unconstrained univariate optimization

Assume can start close to the global minimum $\min_x f(x)$







How to determine the minimum?

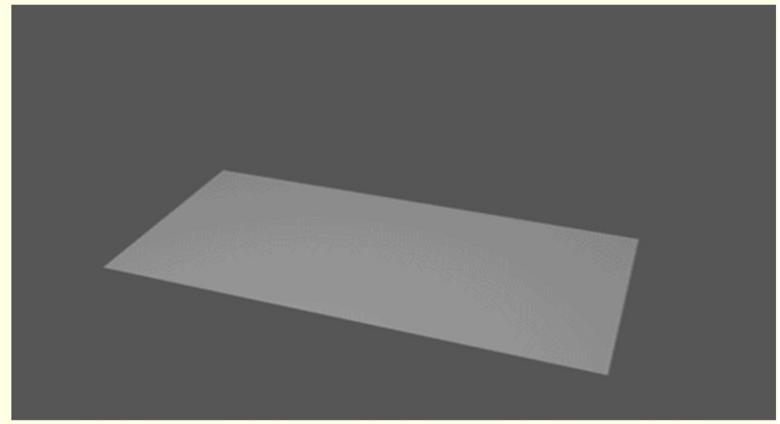
- Search methods (Dichotomous, Fibonacci, Golden-Section)
- Approximation methods
  1. Polynomial interpolation
  2. Newton method
- Combination of both (alg. of Davies, Swann, and Campey)
- Inexact Line Search (Fletcher)

Classification of Optimization Problems

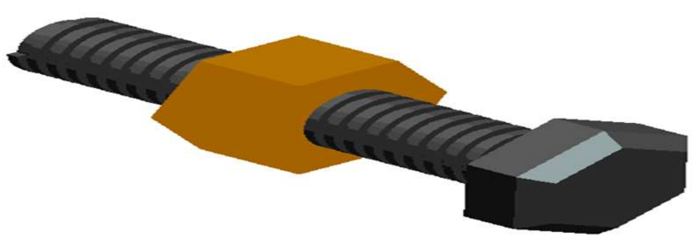
- If $f(x)$ and the constraints are linear, have *linear programming*.
  - e.g.: Maximize $x + y$ subject to

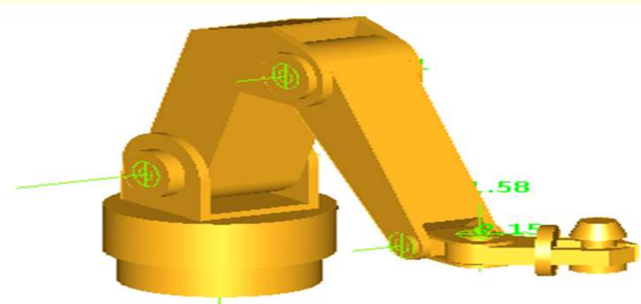
$$3x + 4y \leq 2$$

$$y \leq 5$$



- **If $f(x)$ is quadratic and the constraints are linear, have quadratic programming.**
- **If $f(x)$ is not linear or quadratic and/or the constraints are nonlinear, have nonlinear**

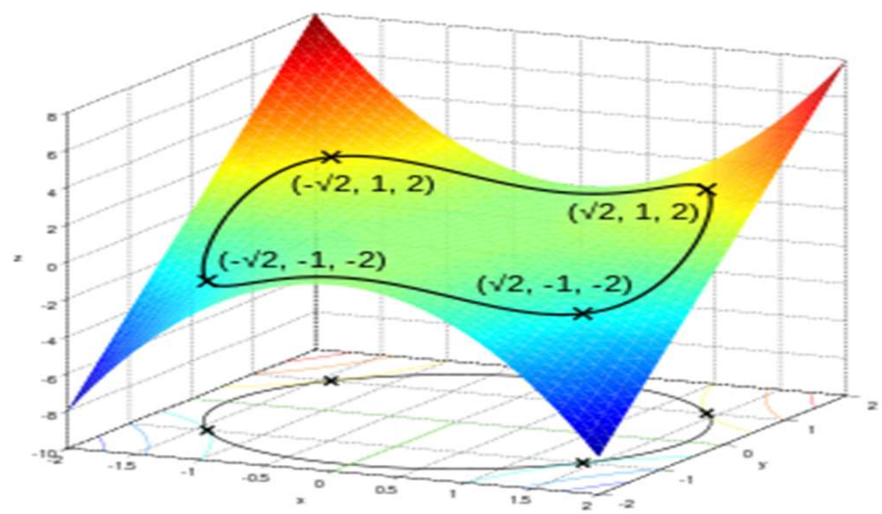


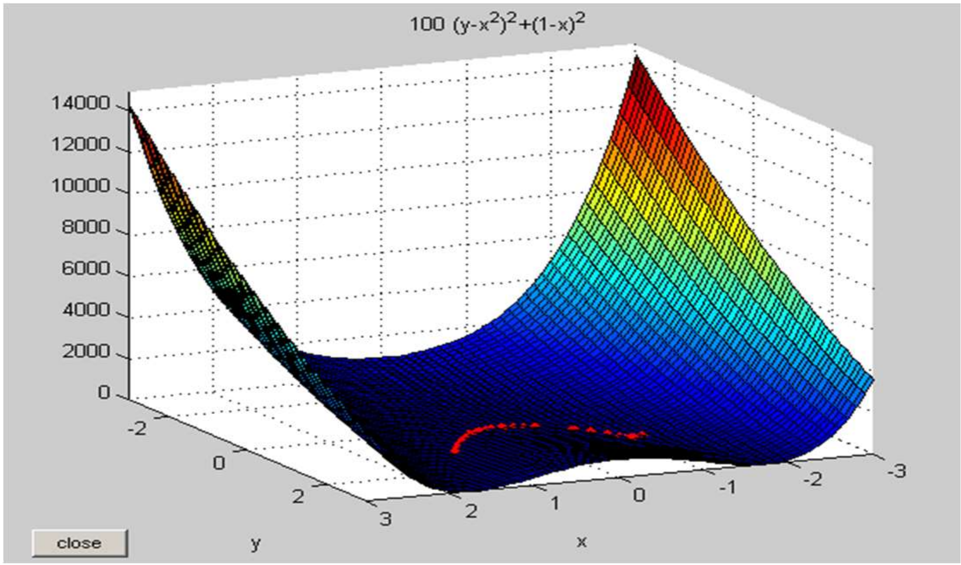


Classification of Optimization Problems

When constraints (equations marked with *) are included, have a *constrained optimization* problem.

Otherwise, have an *unconstrained optimization* problem.

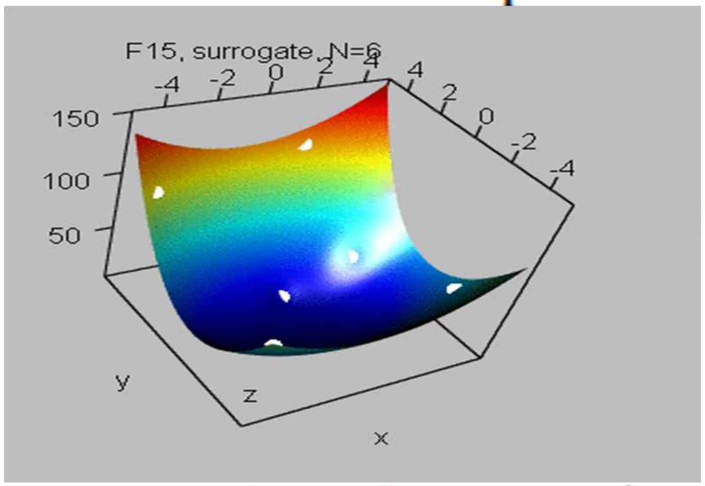




Classification of Optimization Problems

- Unconstrained optimization problem $\min_x F(x)$ or $\max_x F(x)$

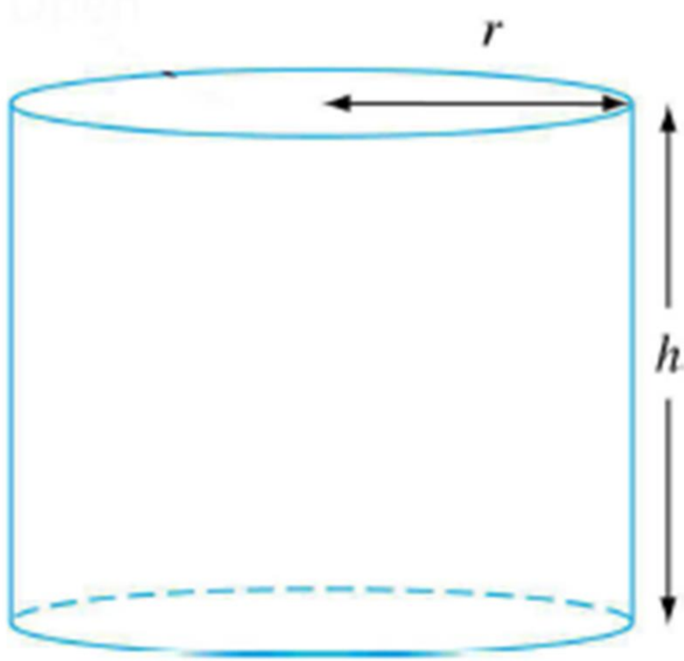
- Constrained optimization problem



$$\min_x F(x) \text{ or } \max_x F(x)$$

$$\text{subject to } g(x) = 0$$

$$\text{and/or } h(x) < 0 \text{ or } h(x) > 0$$



Example: minimize the outer area of a cylinder subject to a fixed volume.

Objective function

$$F(x) = 2\pi r^2 + 2\pi r h, \; x = \begin{bmatrix} r \\ h \end{bmatrix}$$

Constraint: $2\pi r^2 h = V$

Optimization Methods

One-Dimensional Unconstrained Optimization

- **Golden-Section Search**
- **Newton's Method**
- Quadratic Interpolation

Multi-Dimensional Unconstrained Optimization

- **Non-gradient or direct methods**
- Gradient methods

Linear Programming (Constrained)

- **Graphical Solution**
- **Simplex Method**