An Introduction to Heuristic Search-Based Planning

Rick Valenzano and Sheila McIlraith

UNIVERSITY OF TORONTO

Lecture Plan

- Planning as pathfinding in a graph
  - Heuristic-based planning
- From Dijkstra's to Uniform-Cost Search
- Heuristics from abstraction and relaxation
- The A* Algorithm

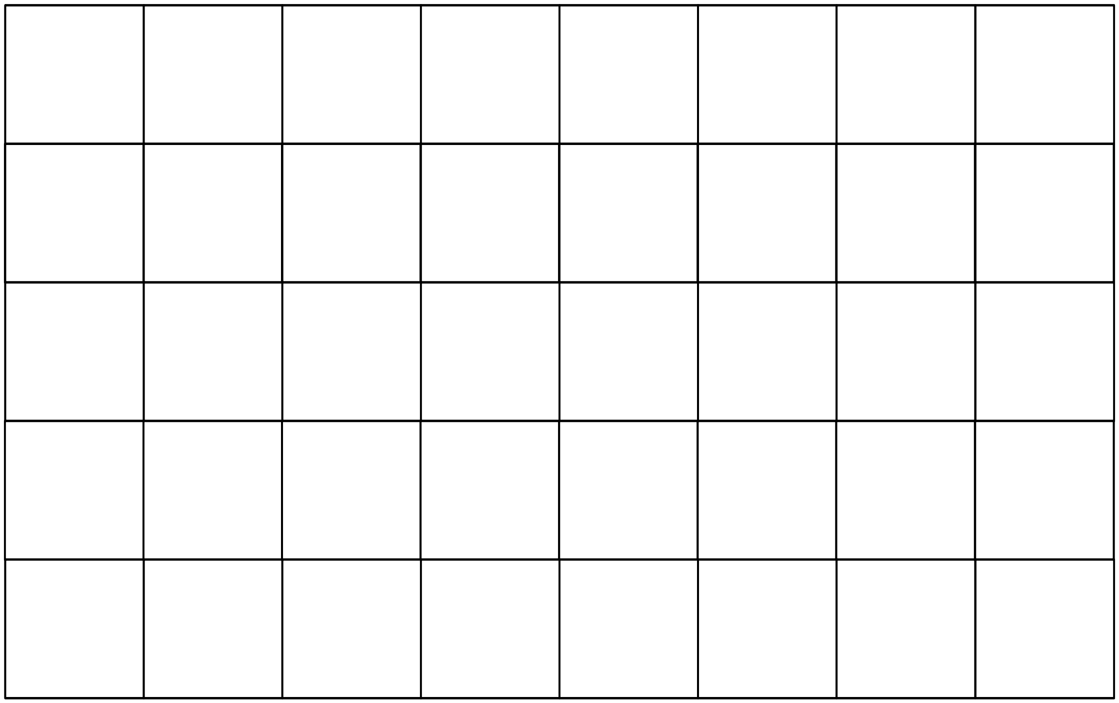
UNIVERSITY OF TORONTO

Quick Survey

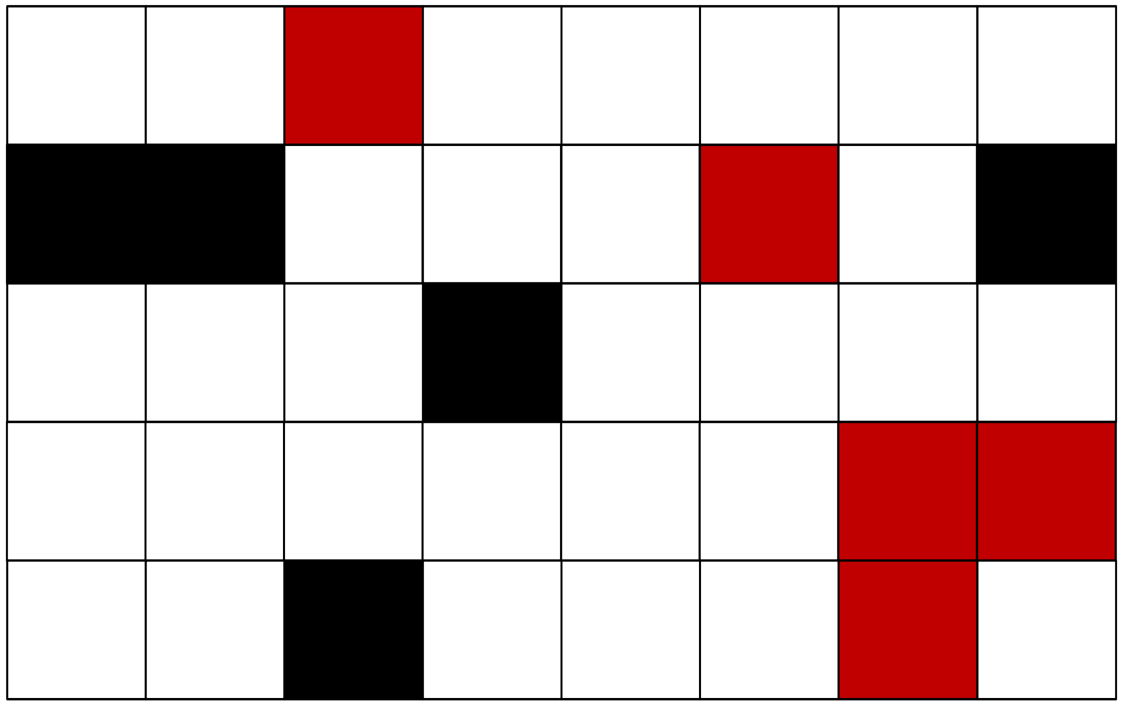
- A*?
- IDA*?
- Weighted A*?
- Greedy Best-First Search?
- Enforced Hill-Climbing?
- A_{ϵ}^{*}? EES?

UNIVERSITY OF TORONTO

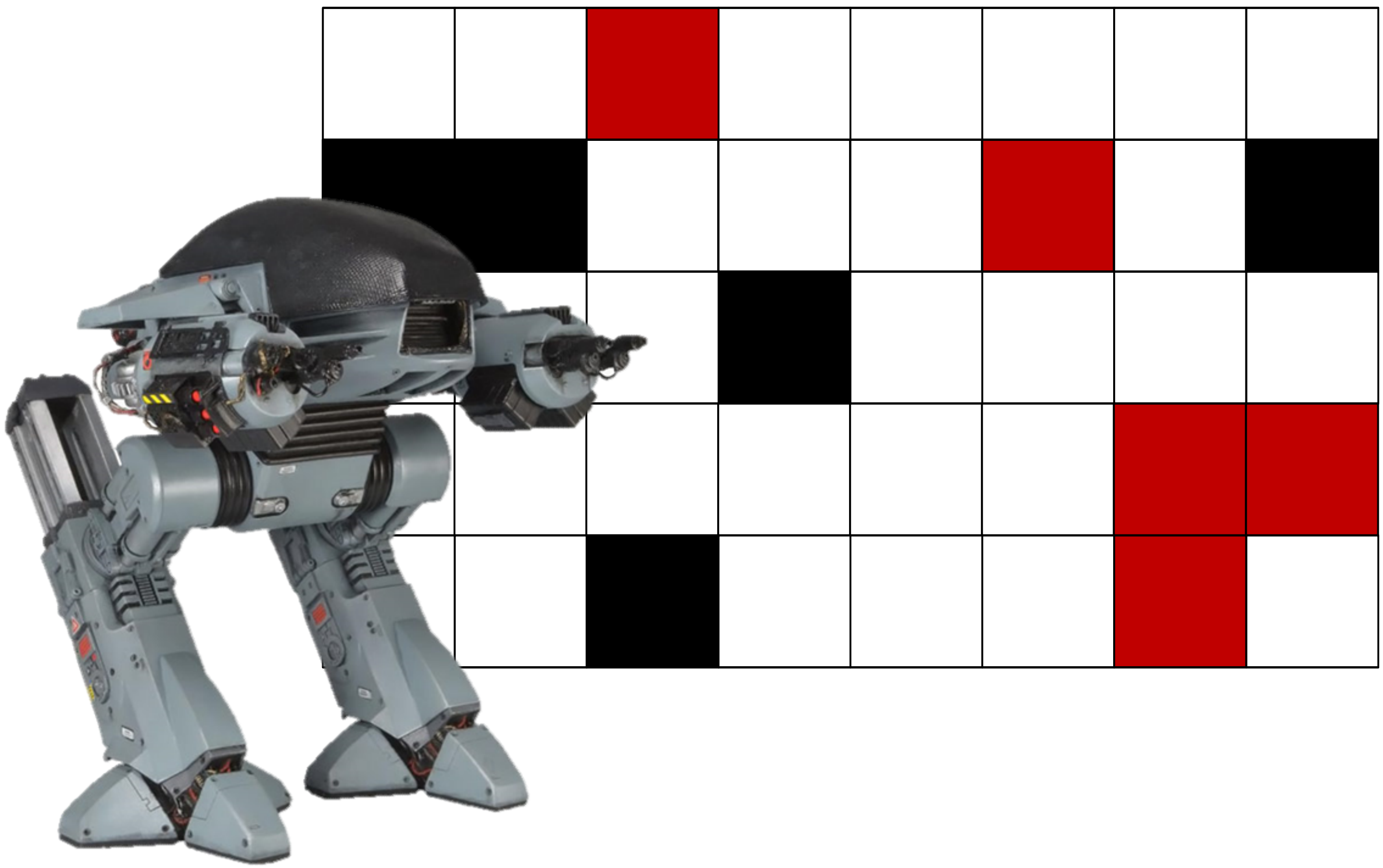
Floortile from IPC 2011



Floortile from IPC 2011

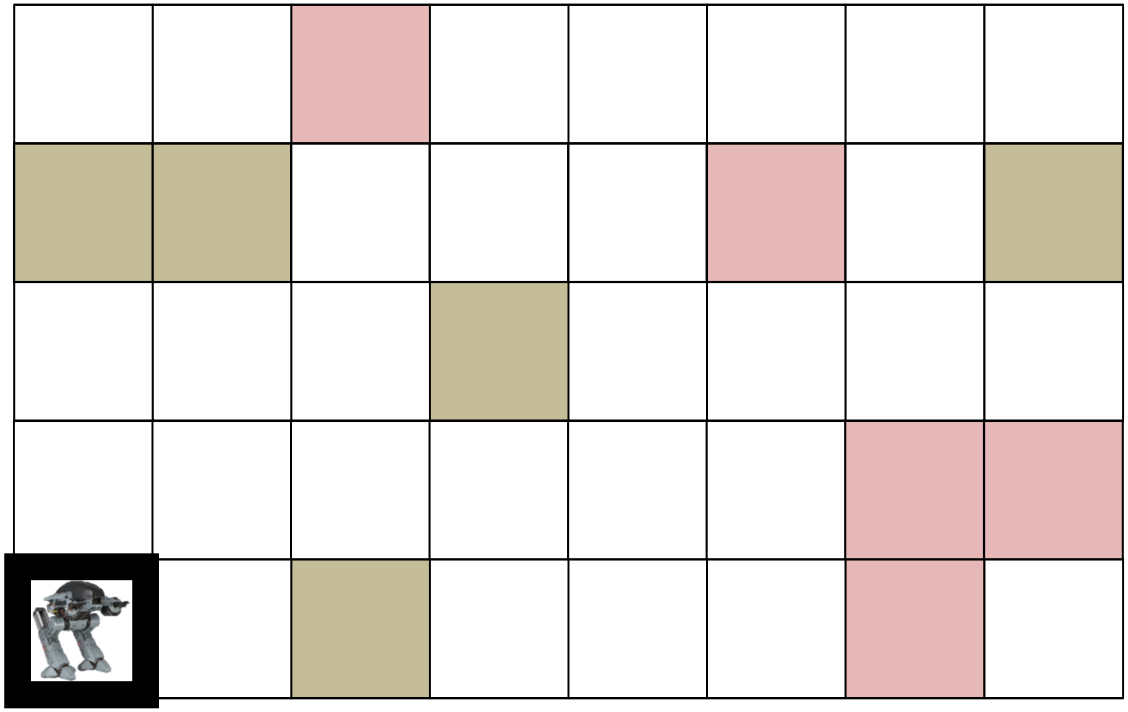


Floortile from IPC 2011

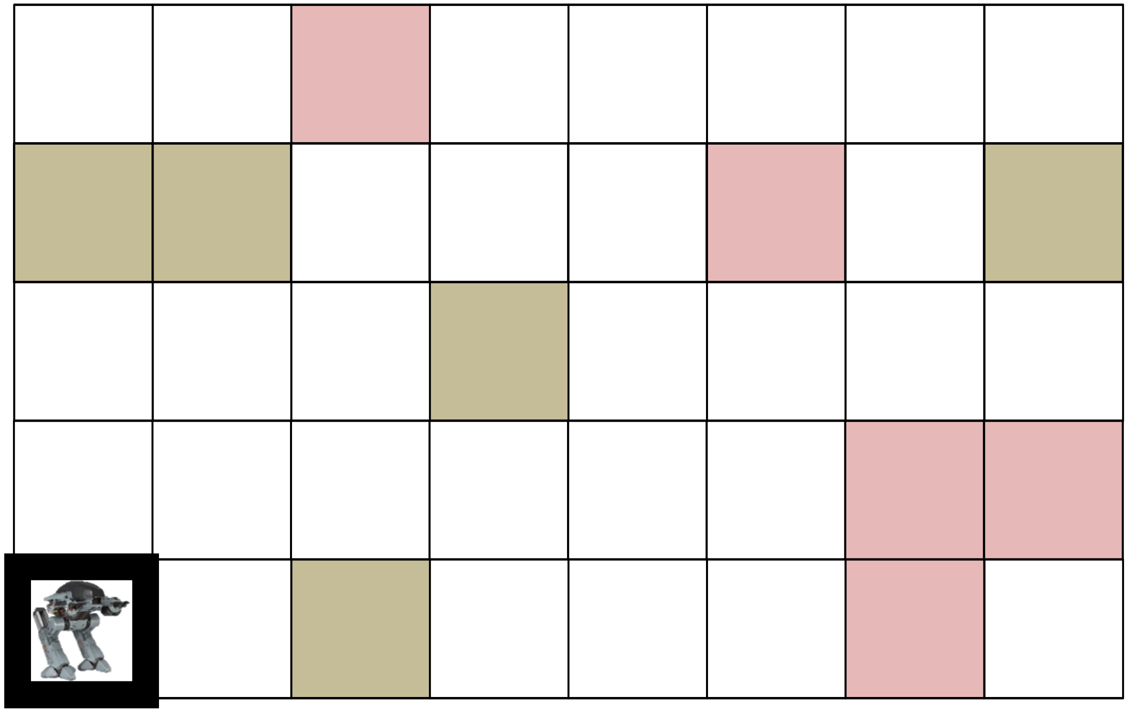


UNIVERSITY OF TORONTO

Floortile from IPC 2011



Floortile from IPC 2011

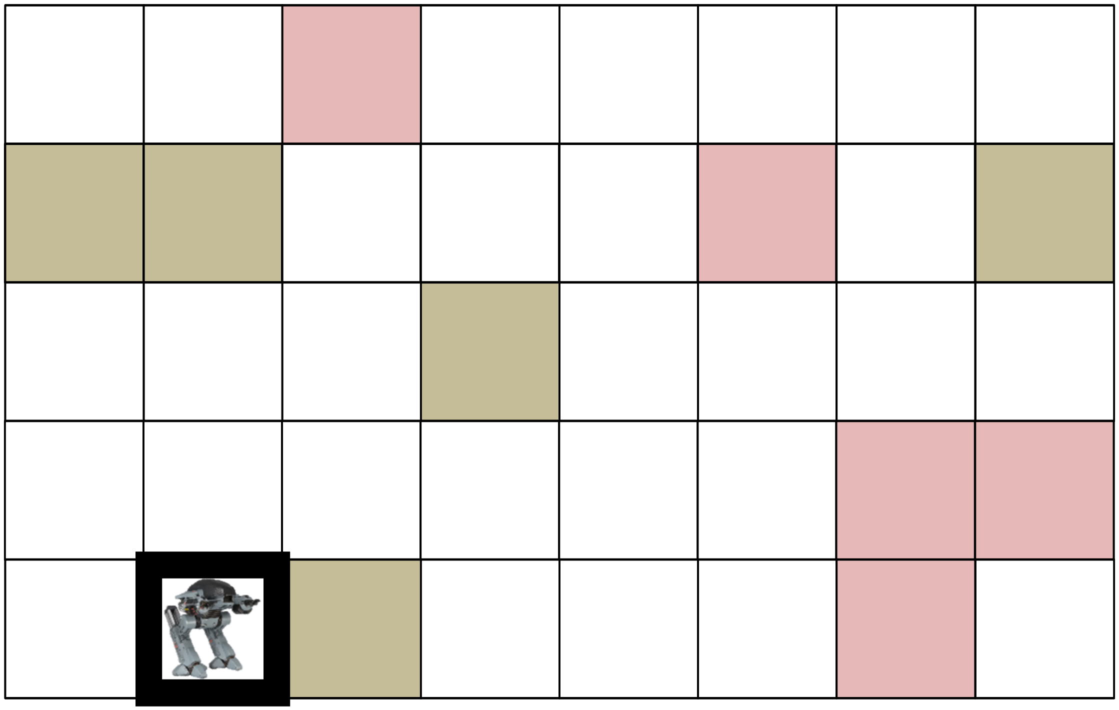


Move Action: MOVE-0-0-0-1

Pre: AT-0-0, WHITE-0-1

Post: AT-0-1, not(AT-0-0)

Floortile from IPC 2011

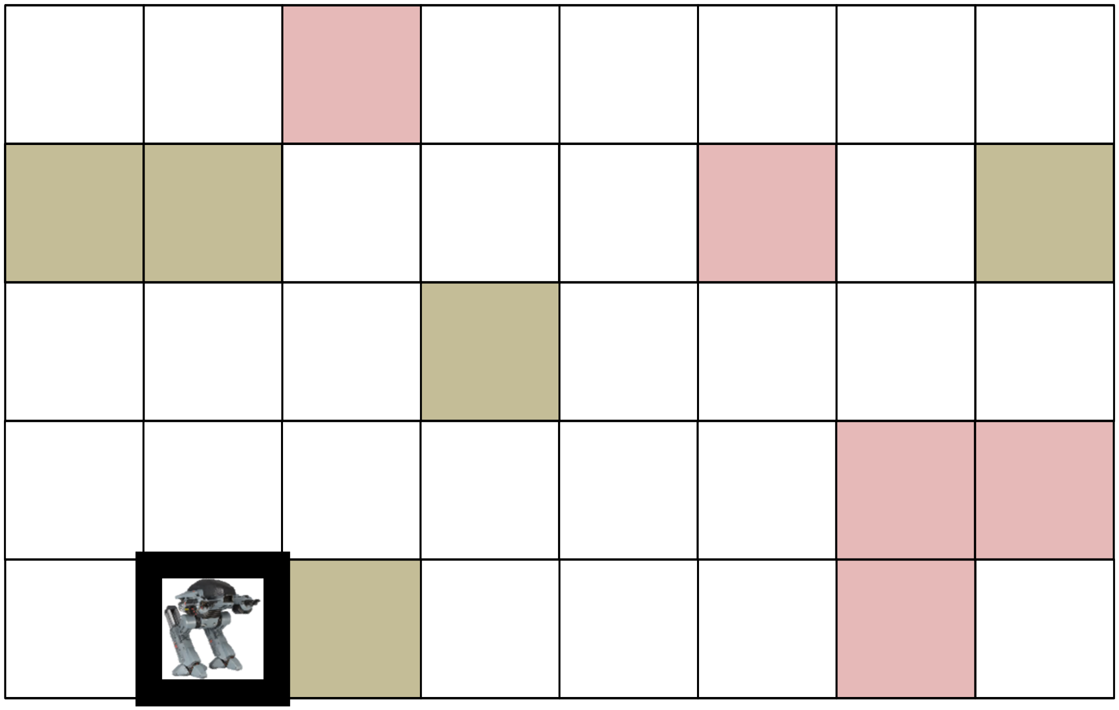


Move Action: MOVE-0-0-0-1

Pre: AT-0-0, WHITE-0-1

Post: AT-0-1, not(AT-0-0)

Floortile from IPC 2011



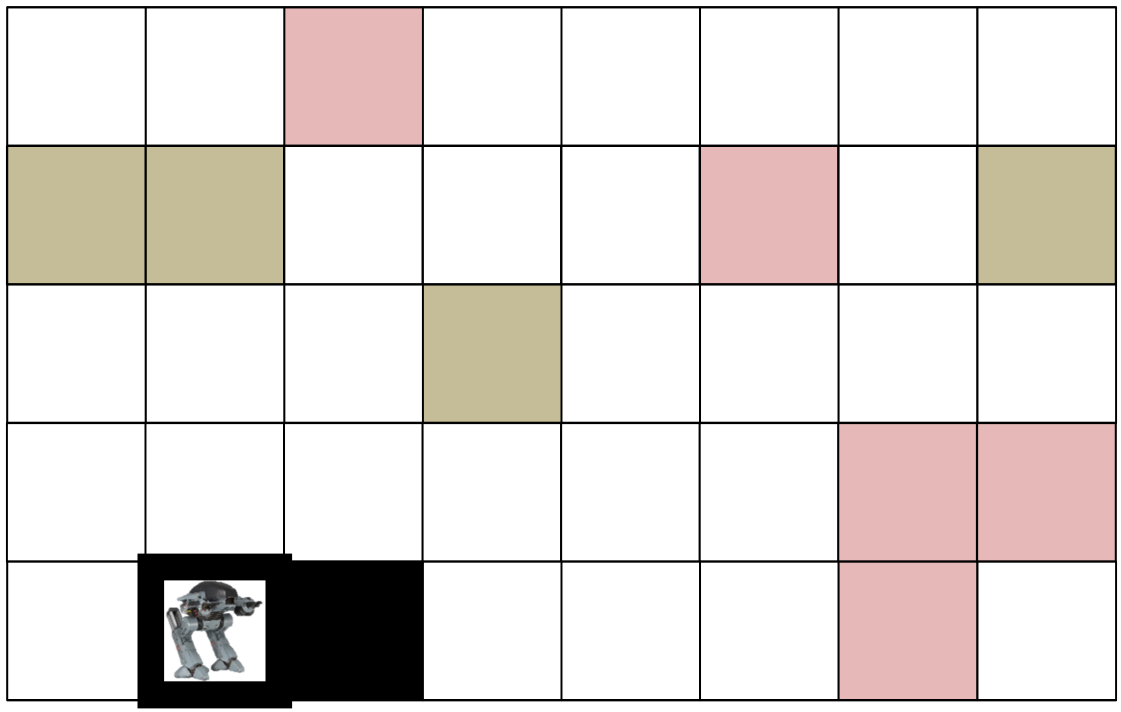
Paint Action: PAINT-B-0-1-0-2

Pre: AT-0-1, LOADED-B,
WHITE-0-2, NEED-B-0-2

Post: BLACK-0-2, not(WHITE-0-2)

UNIVERSITY OF TORONTO

Floortile from IPC 2011



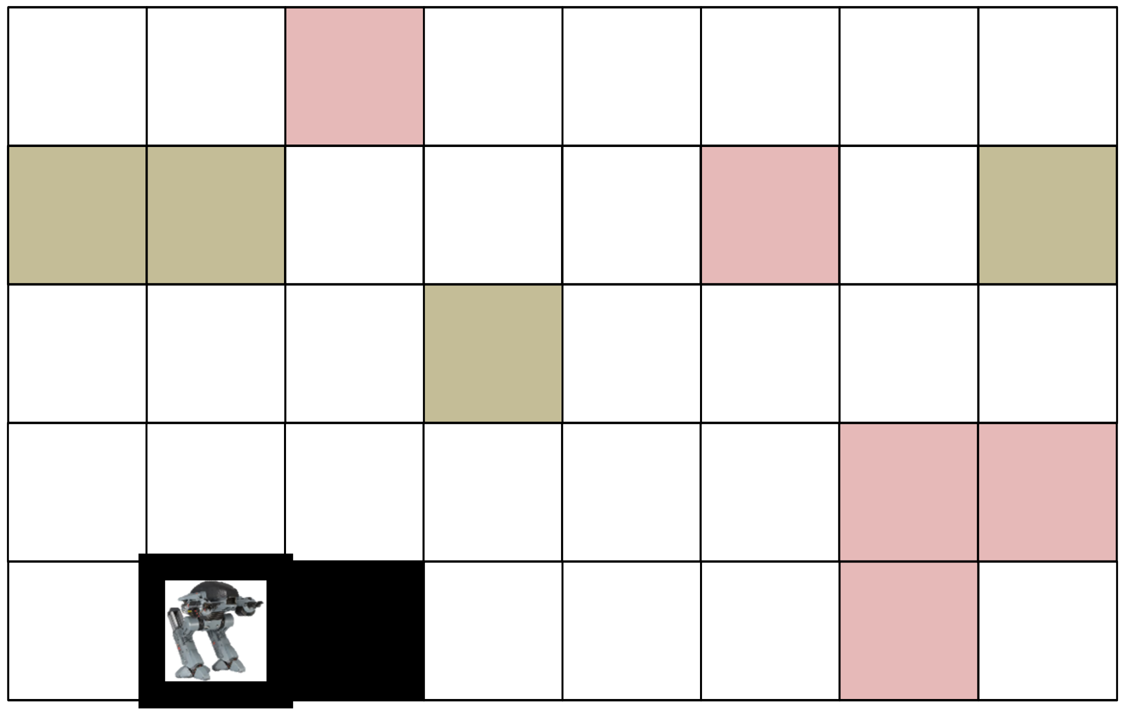
Paint Action: PAINT-B-0-1-0-2

Pre: AT-0-1, LOADED-B,
WHITE-0-2, NEED-B-0-2

Post: BLACK-0-2, not(WHITE-0-2)

UNIVERSITY OF TORONTO

Floortile from IPC 2011



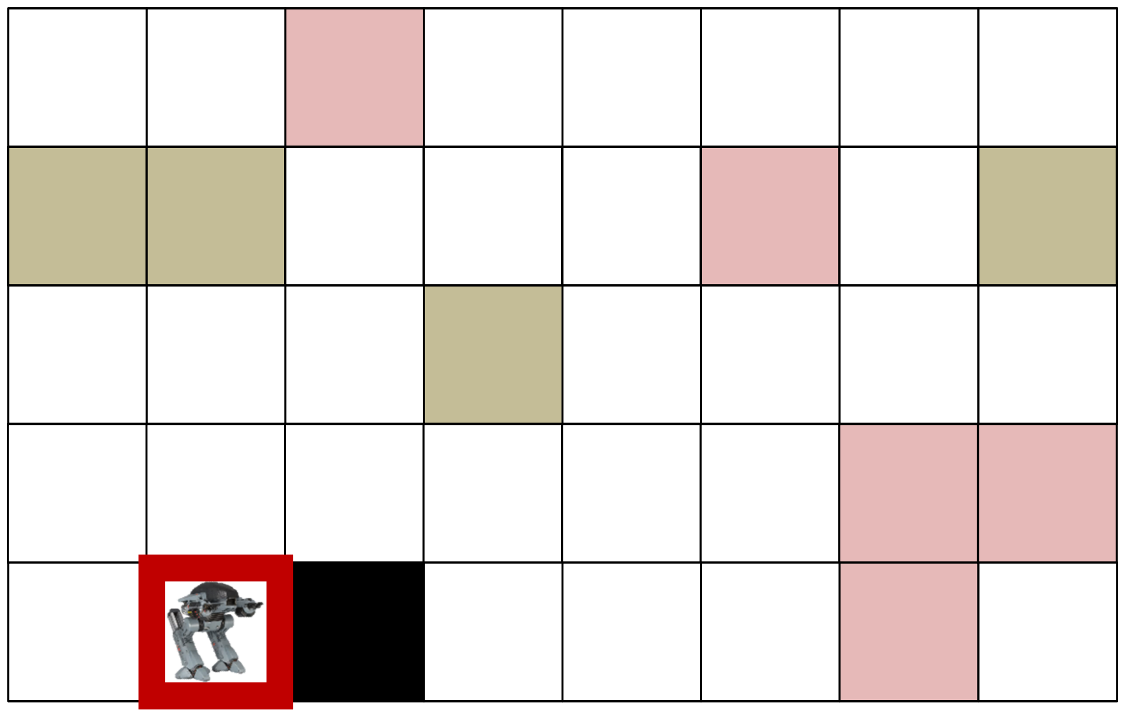
Load Action: LOAD-RED

Pre: LOADED-B

Post: LOADED-R, not(LOADED-B)

UNIVERSITY OF TORONTO

Floortile from IPC 2011



Load Action: LOAD-RED

Pre: LOADED-B

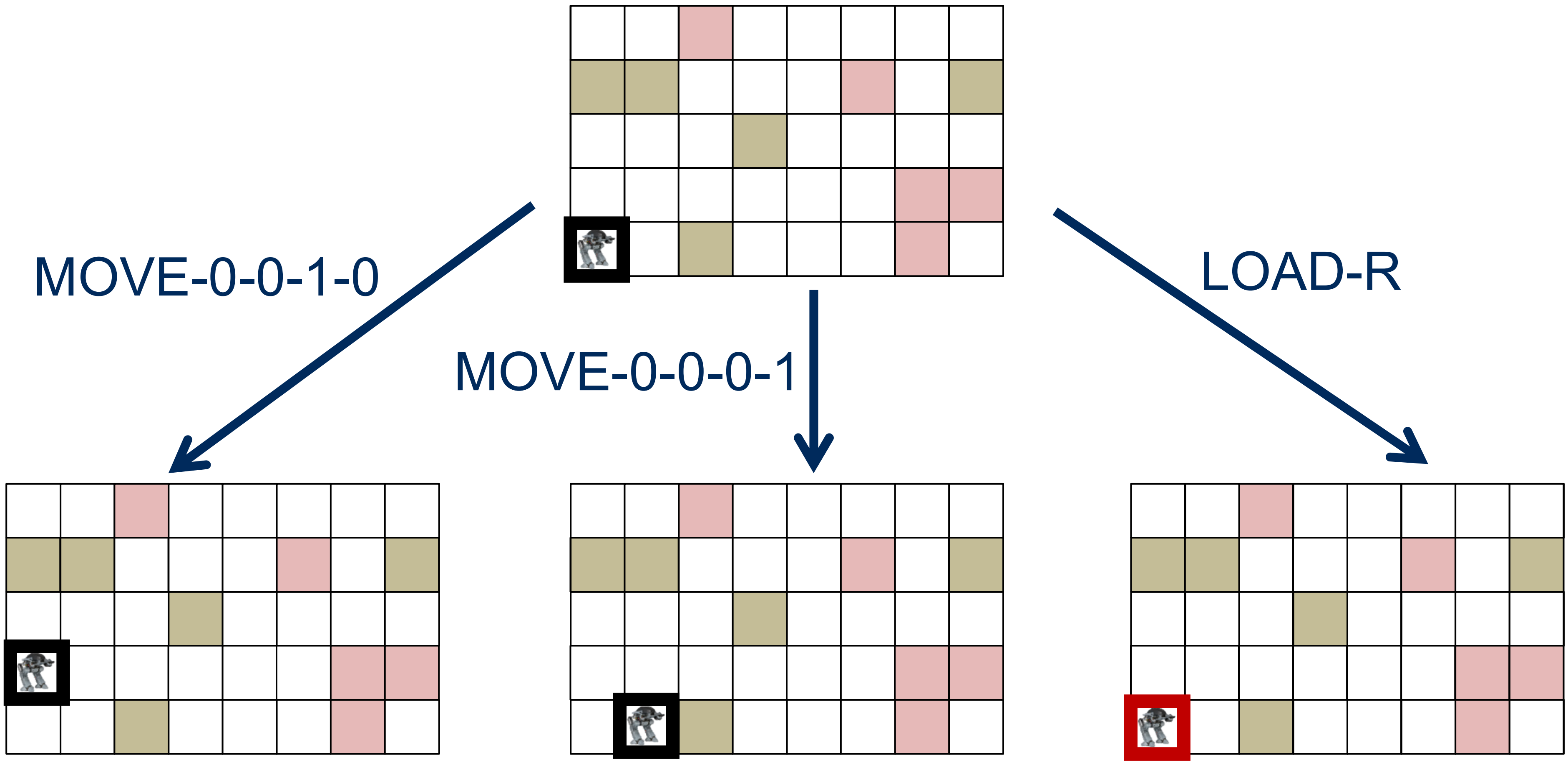
Post: LOADED-R, not(LOADED-B)

Floortile from IPC 2011

Initial State	Goal
AT-0-0	BLACK-0-2
LOADED-B	BLACK-3-0
WHITE-0-0	BLACK-3-1
WHITE-0-1	RED-4-2
WHITE-0-2	BLACK-2-3
NEED-B-0-2	RED-3-5
…	…

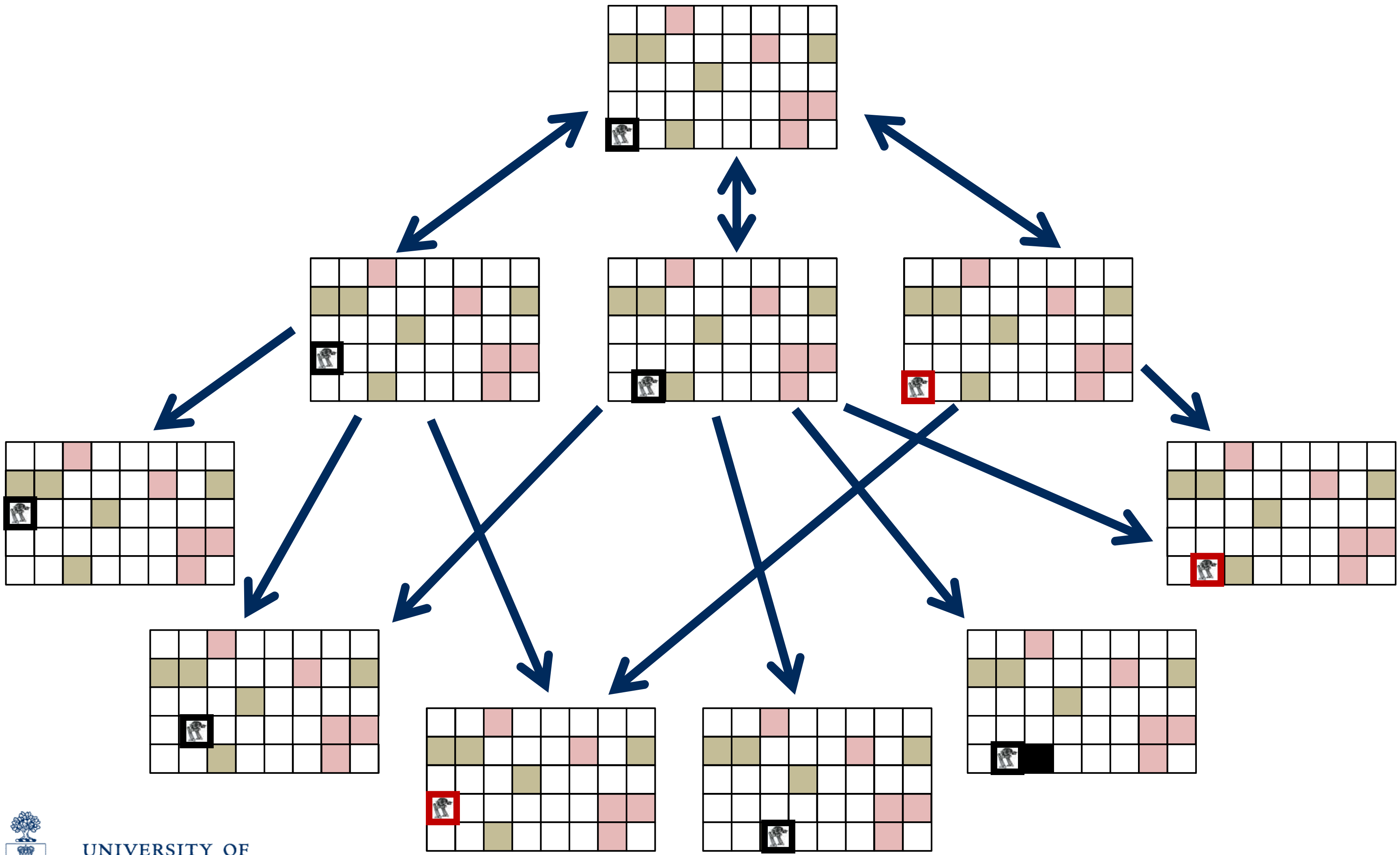
UNIVERSITY OF TORONTO

Floortile from IPC 2011



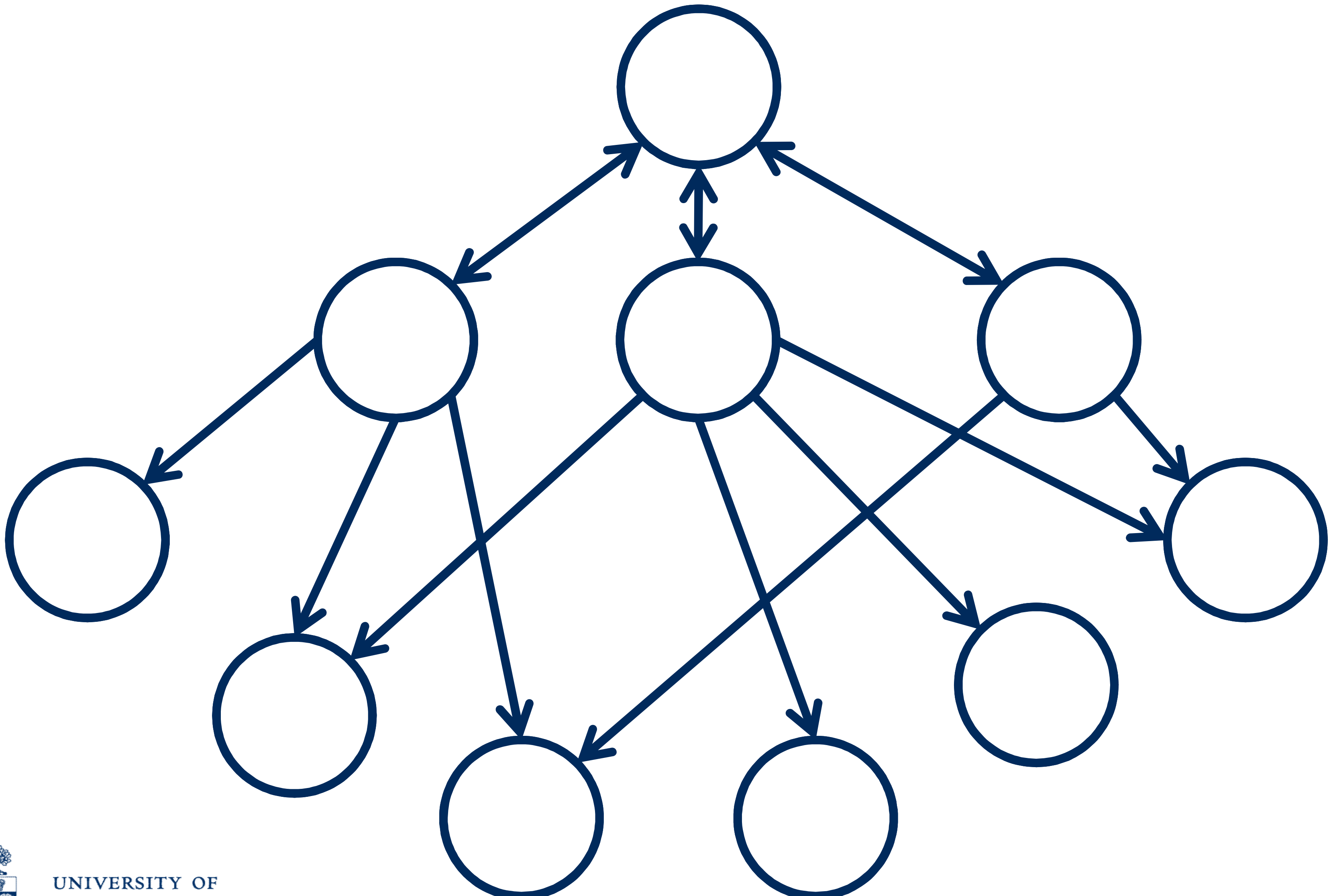
UNIVERSITY OF TORONTO

Floortile from IPC 2011



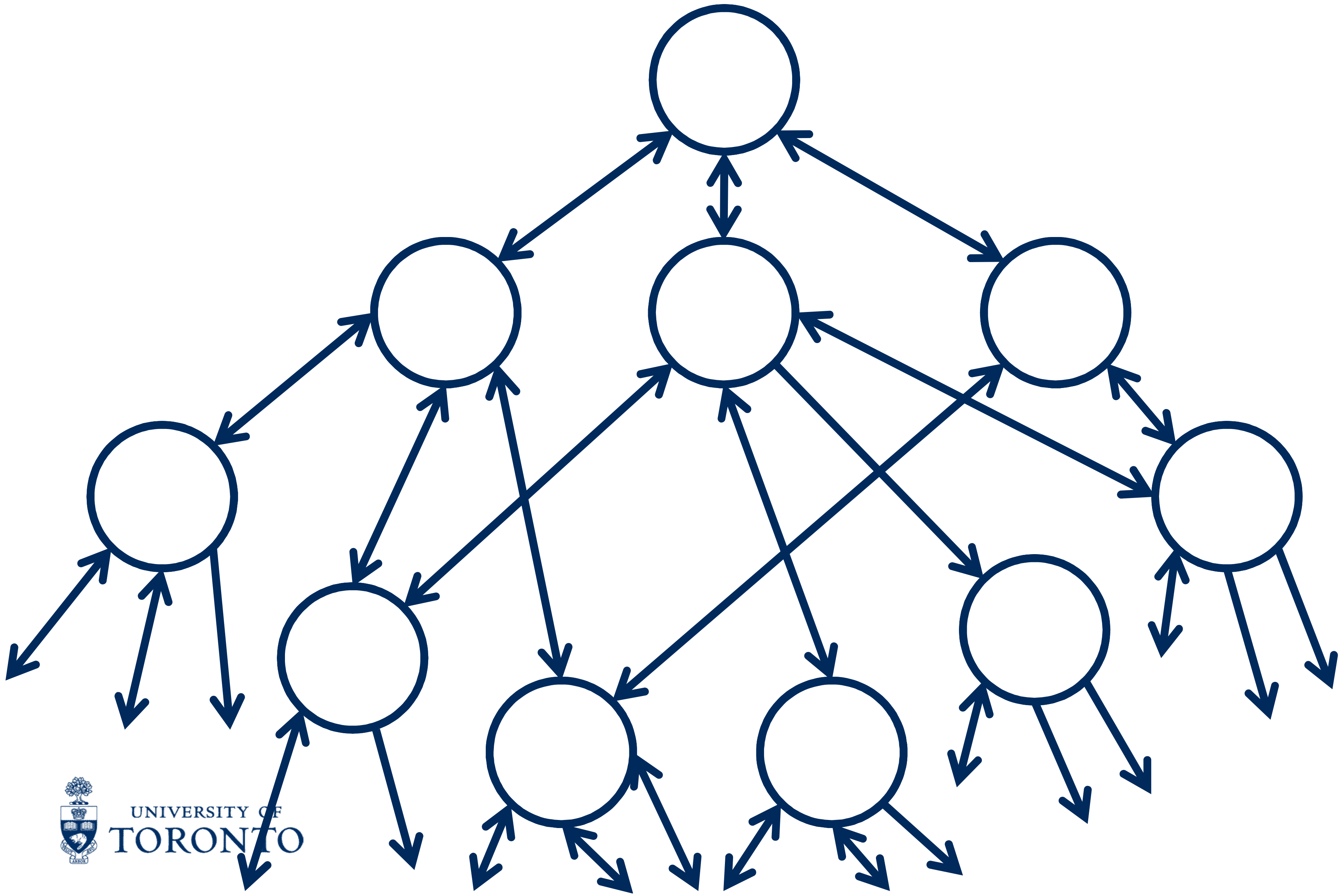
UNIVERSITY OF TORONTO

Graph Underlying Floortile



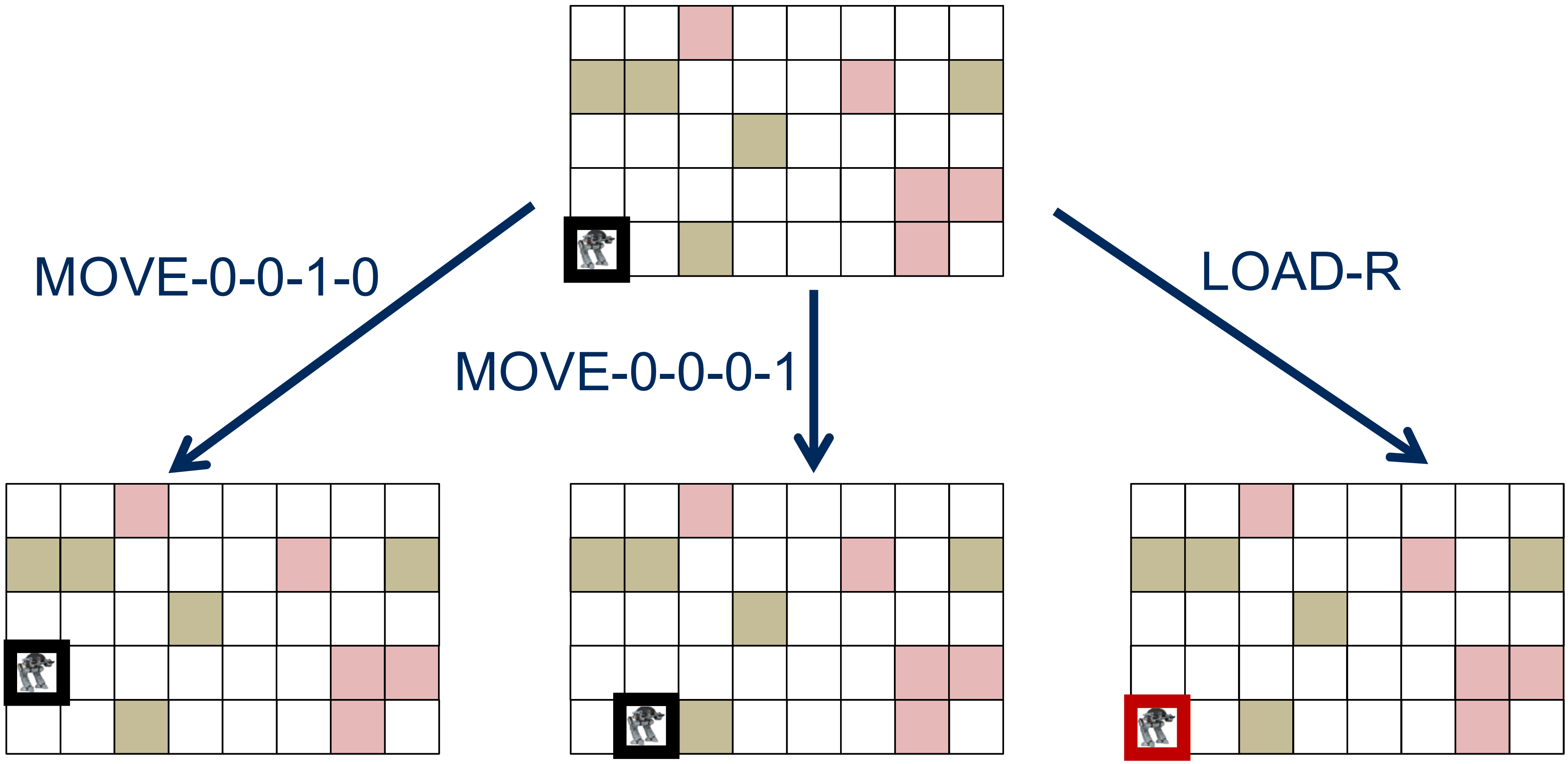
UNIVERSITY OF TORONTO

Graph Underlying Floortile

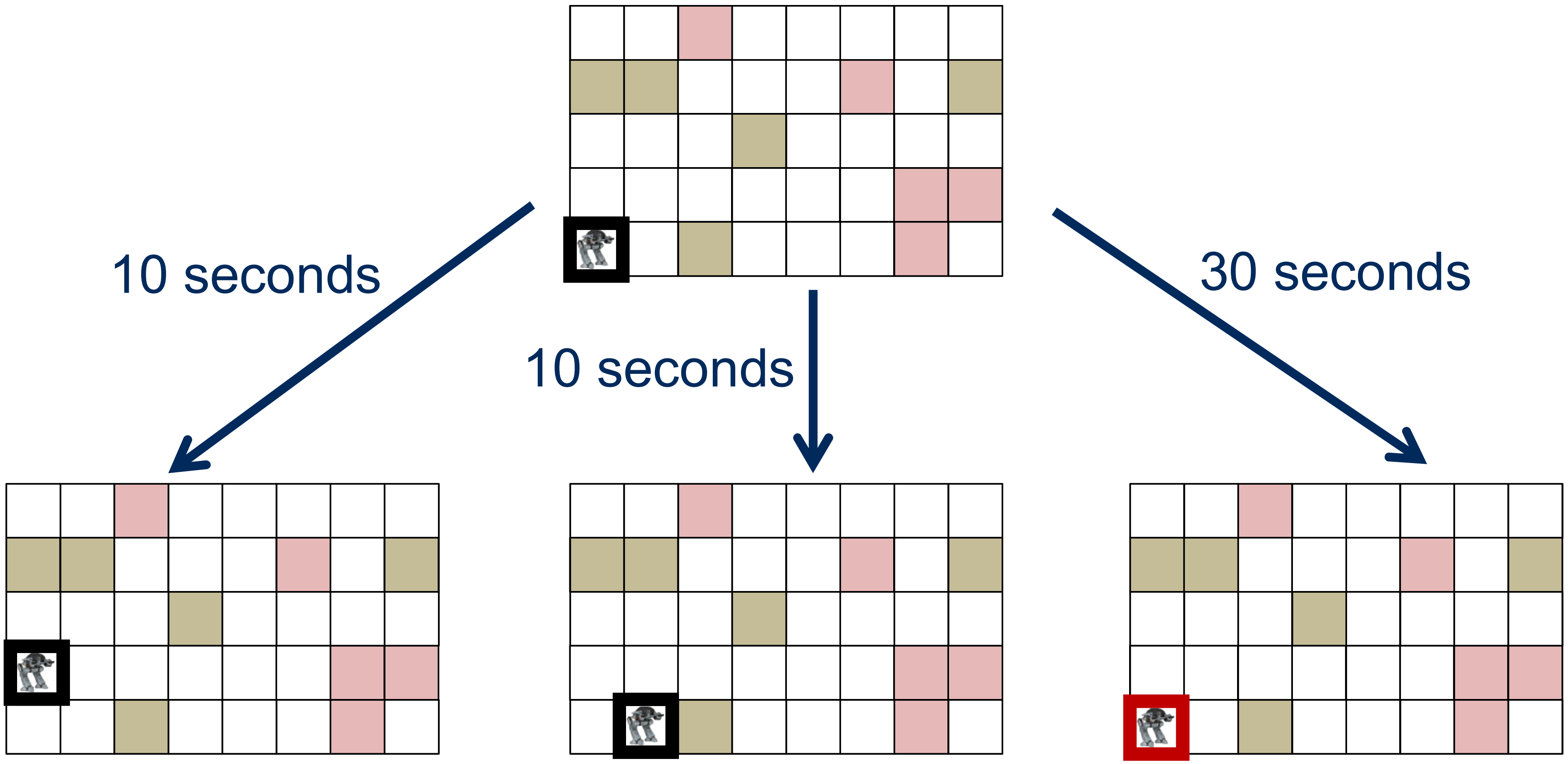


UNIVERSITY OF TORONTO

Action Costs

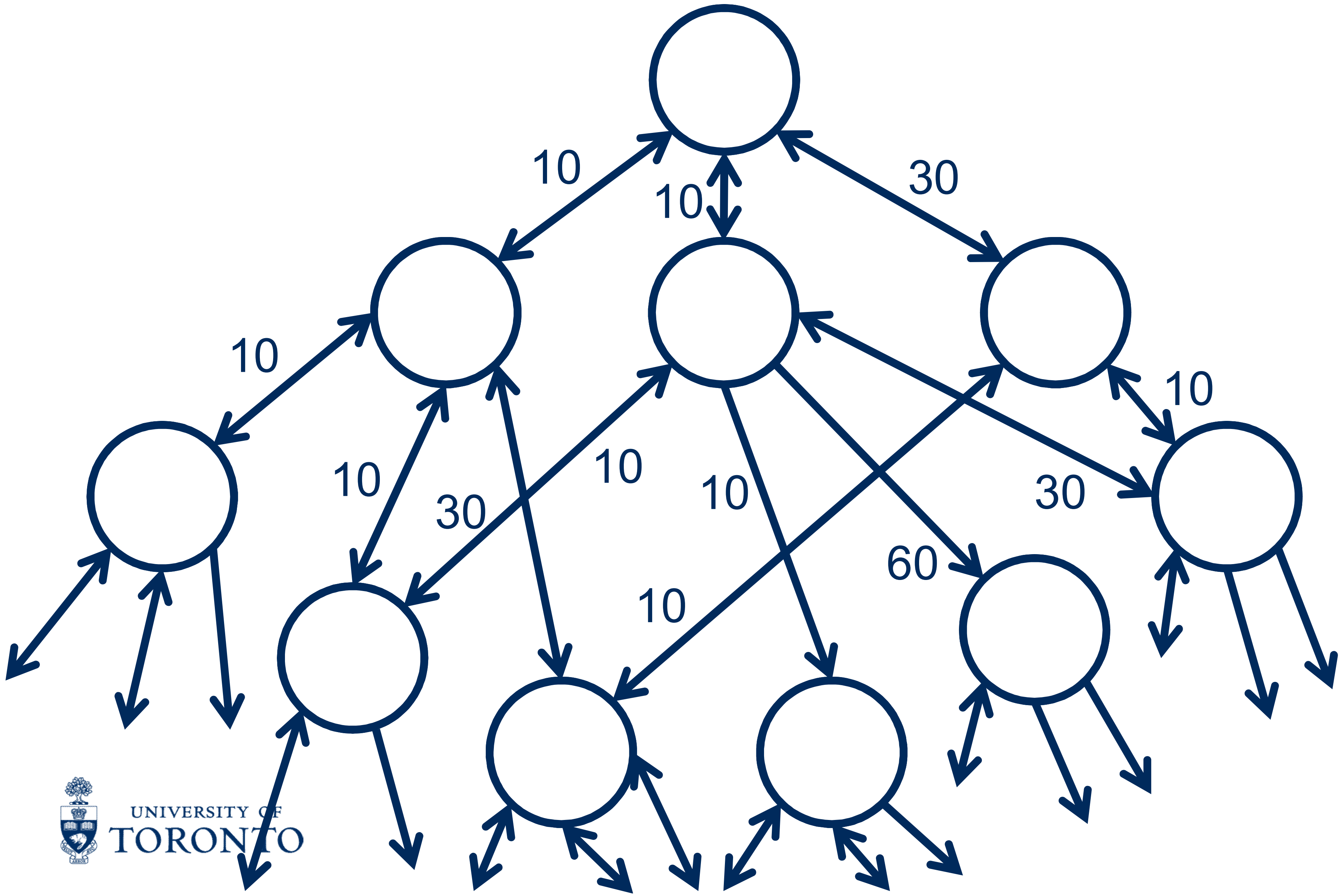


Action Costs



UNIVERSITY OF TORONTO

Edge Weights

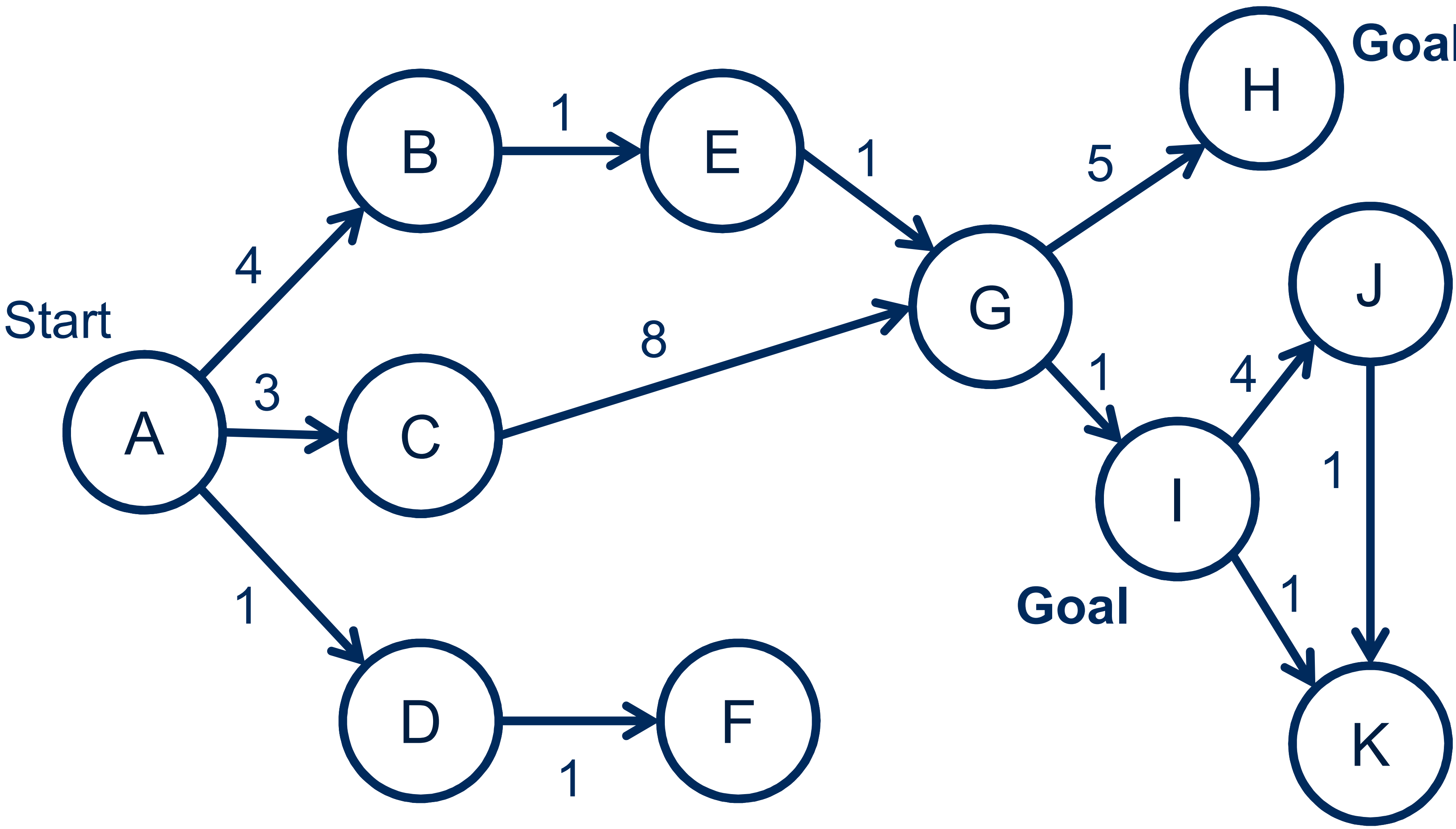


UNIVERSITY OF TORONTO

Planning as Graph-Search

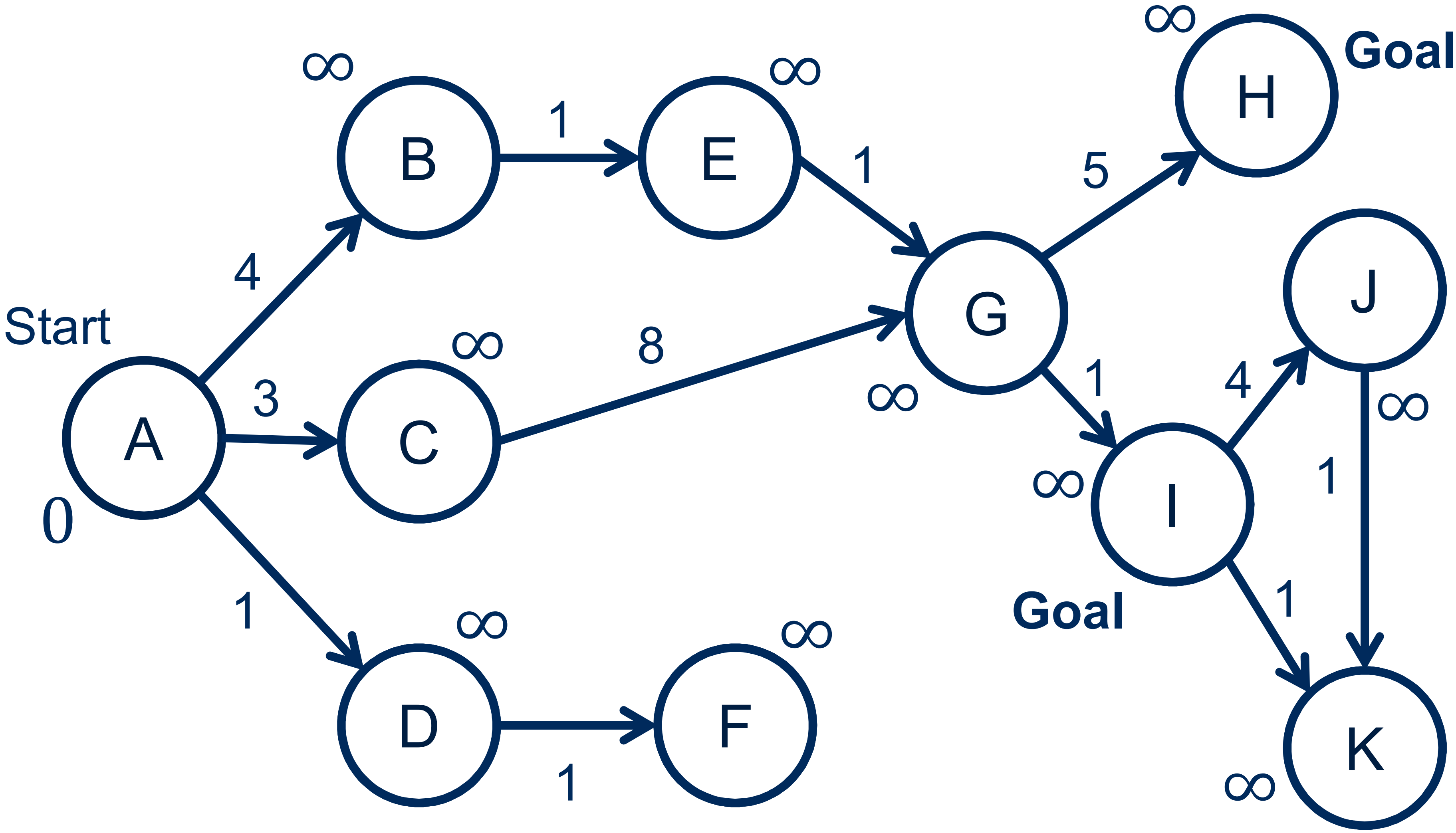
- Can generate the underlying graph
- Use the goal test function to label goal nodes
- Use a standard graph-search algorithm

Dijkstra's Search



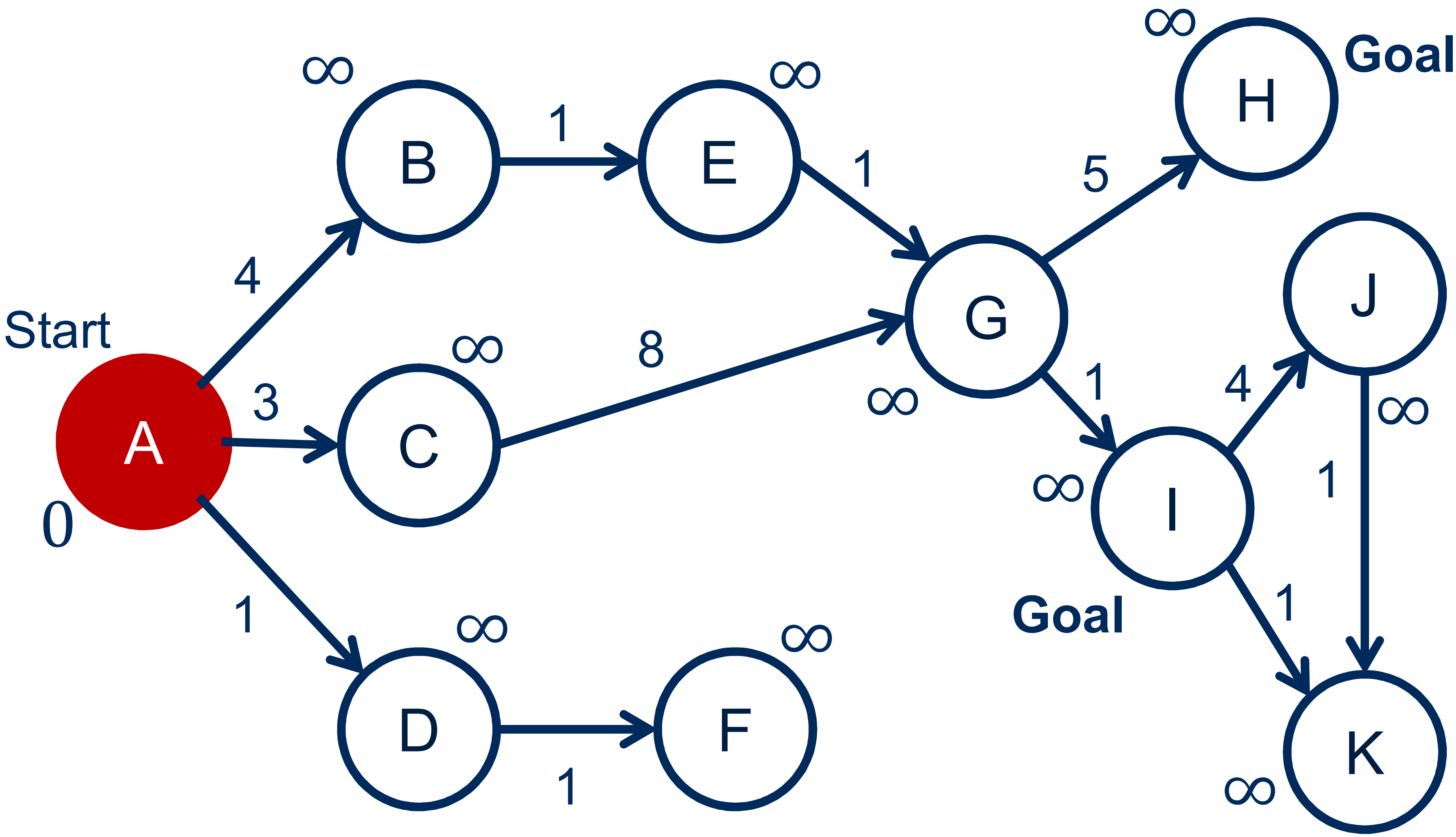
UNIVERSITY OF TORONTO

Dijkstra's Search



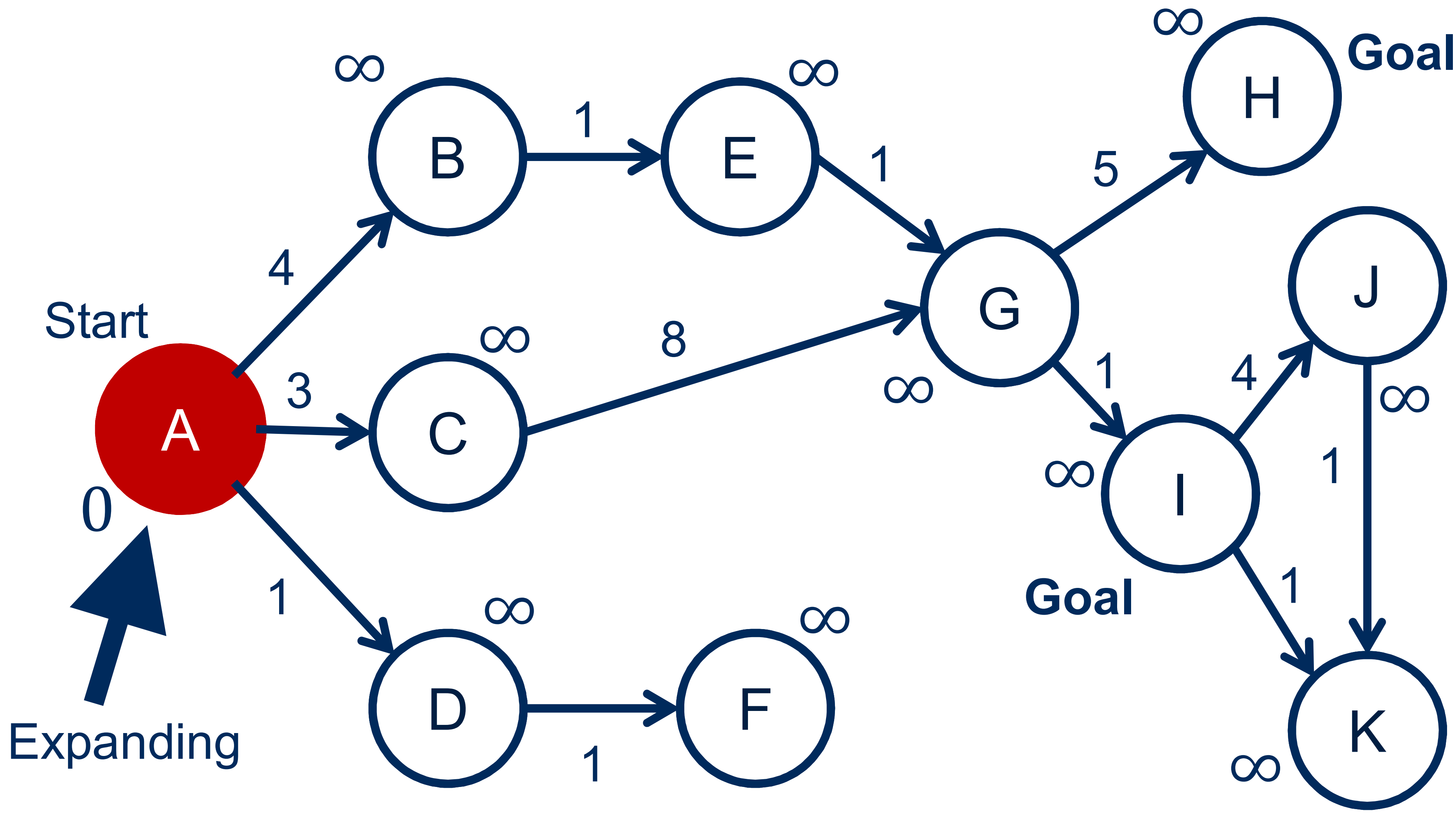
UNIVERSITY OF TORONTO

Dijkstra's Search



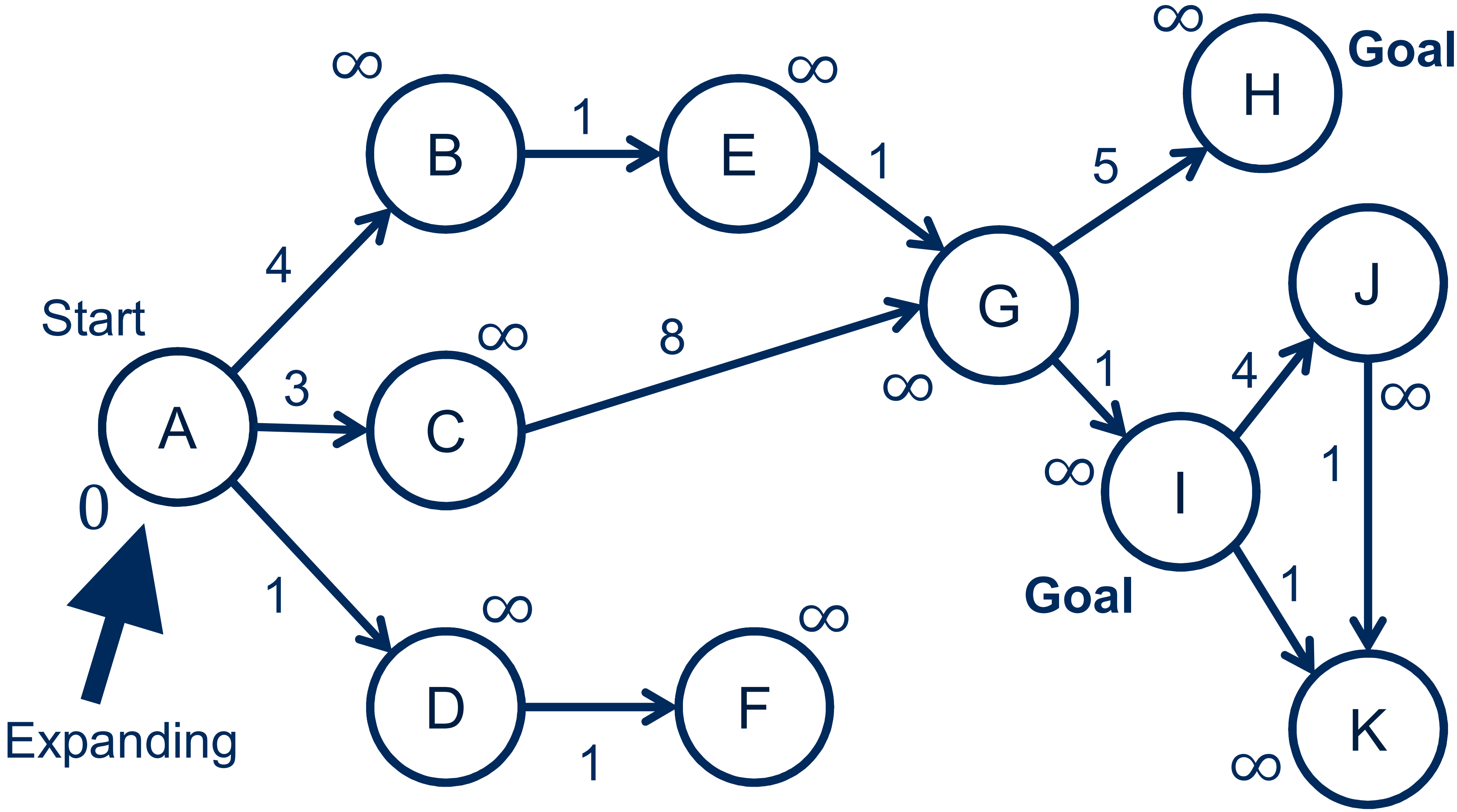
UNIVERSITY OF TORONTO

Dijkstra's Search



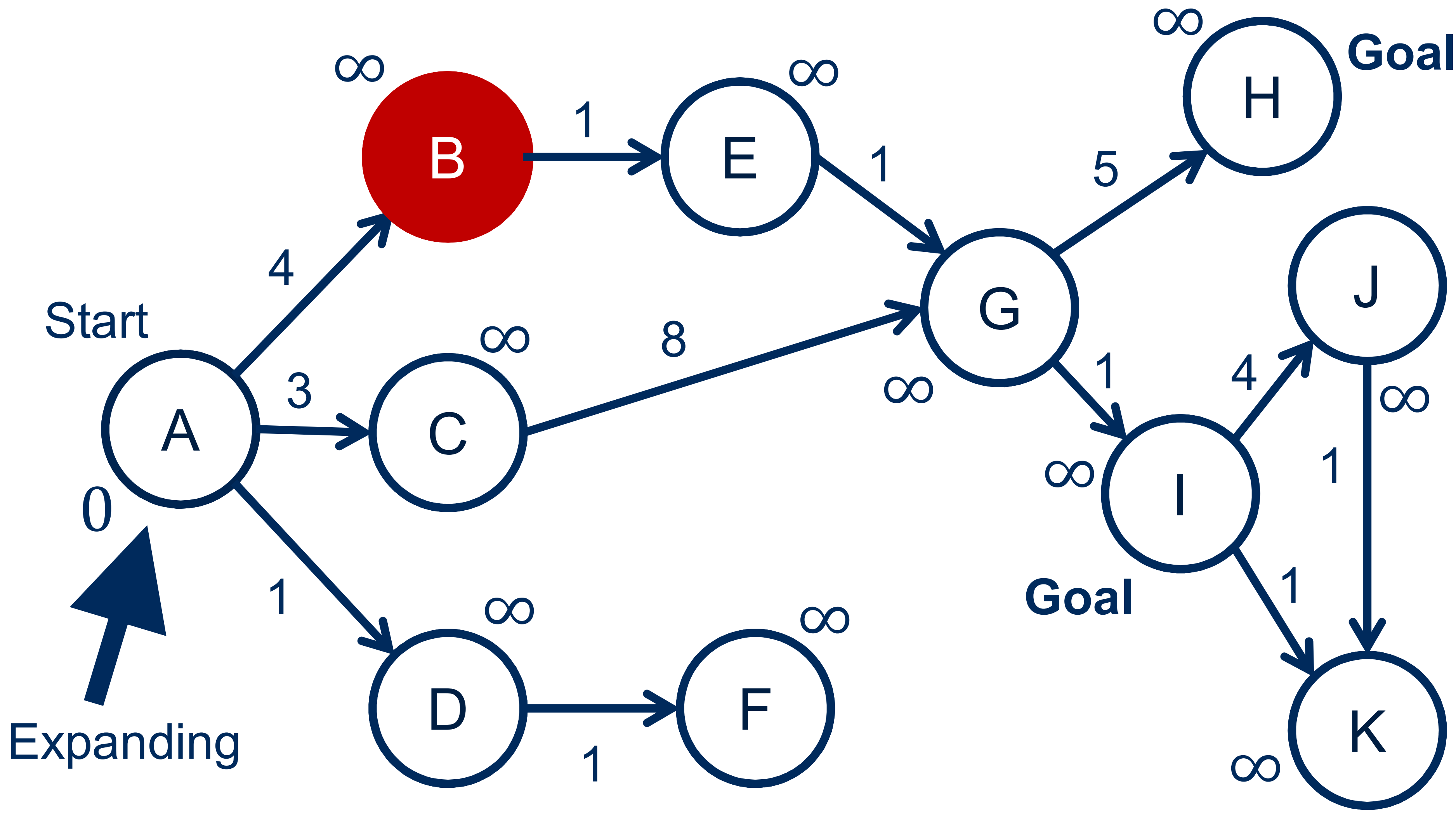
UNIVERSITY OF TORONTO

Dijkstra's Search



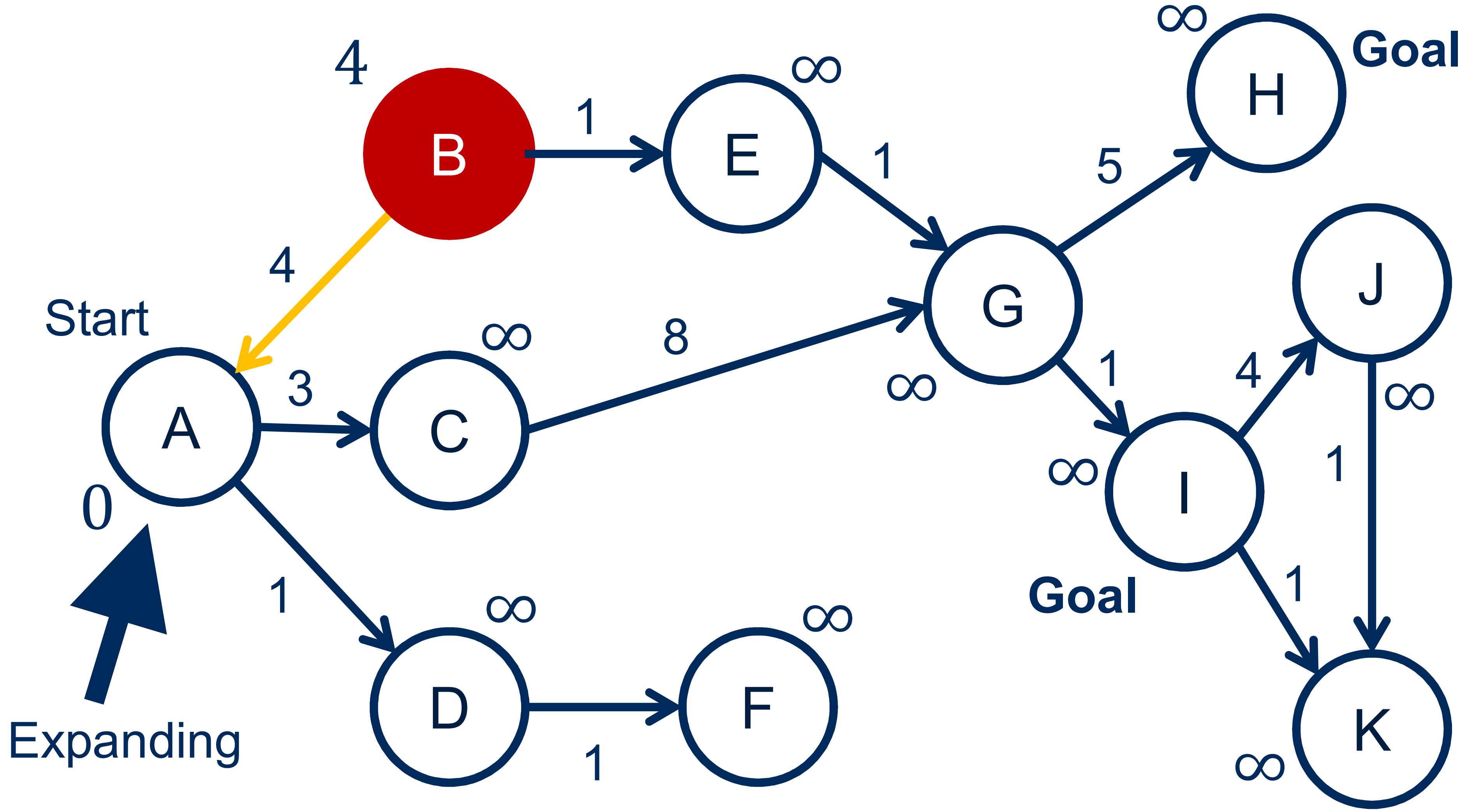
UNIVERSITY OF TORONTO

Dijkstra's Search



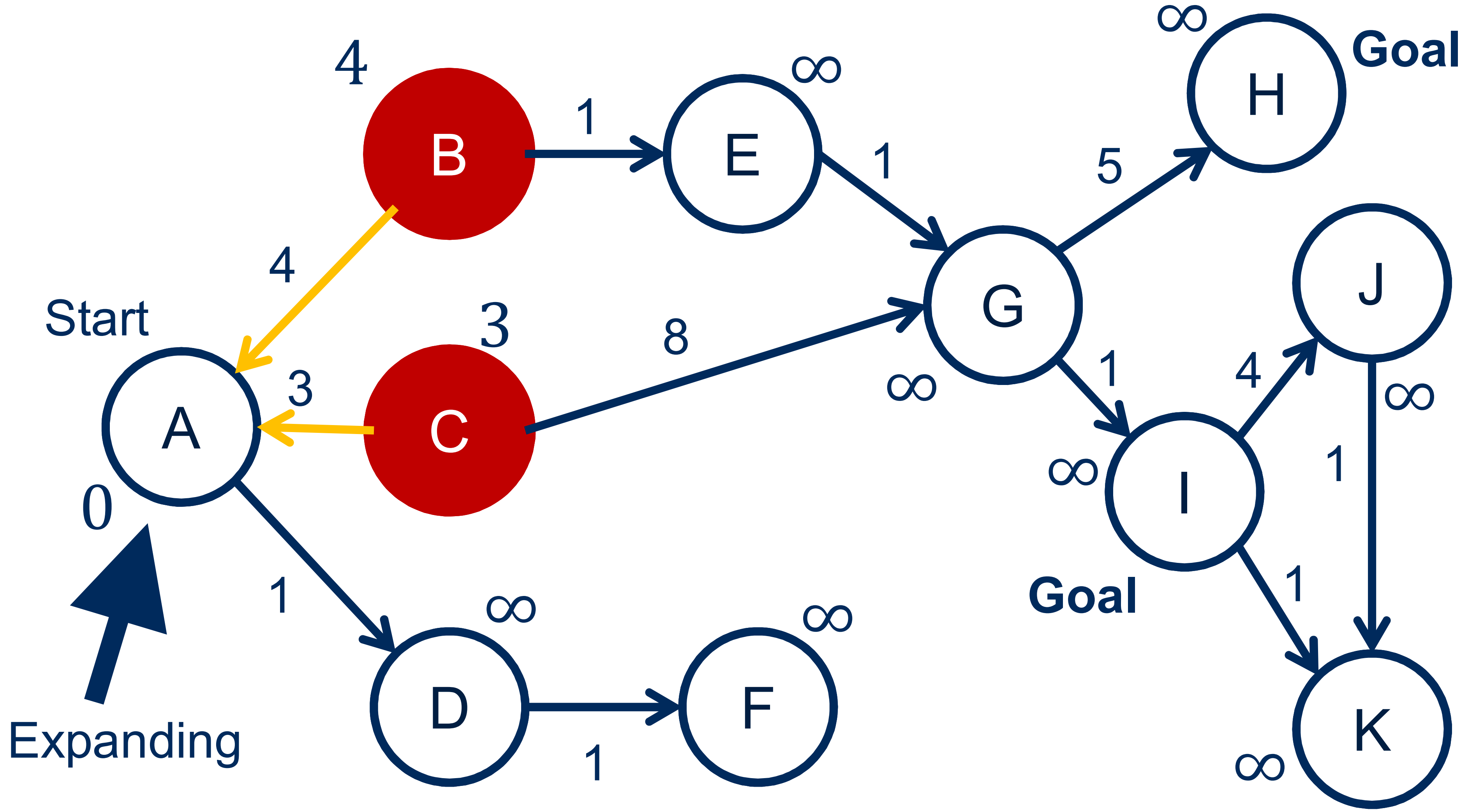
UNIVERSITY OF TORONTO

Dijkstra's Search

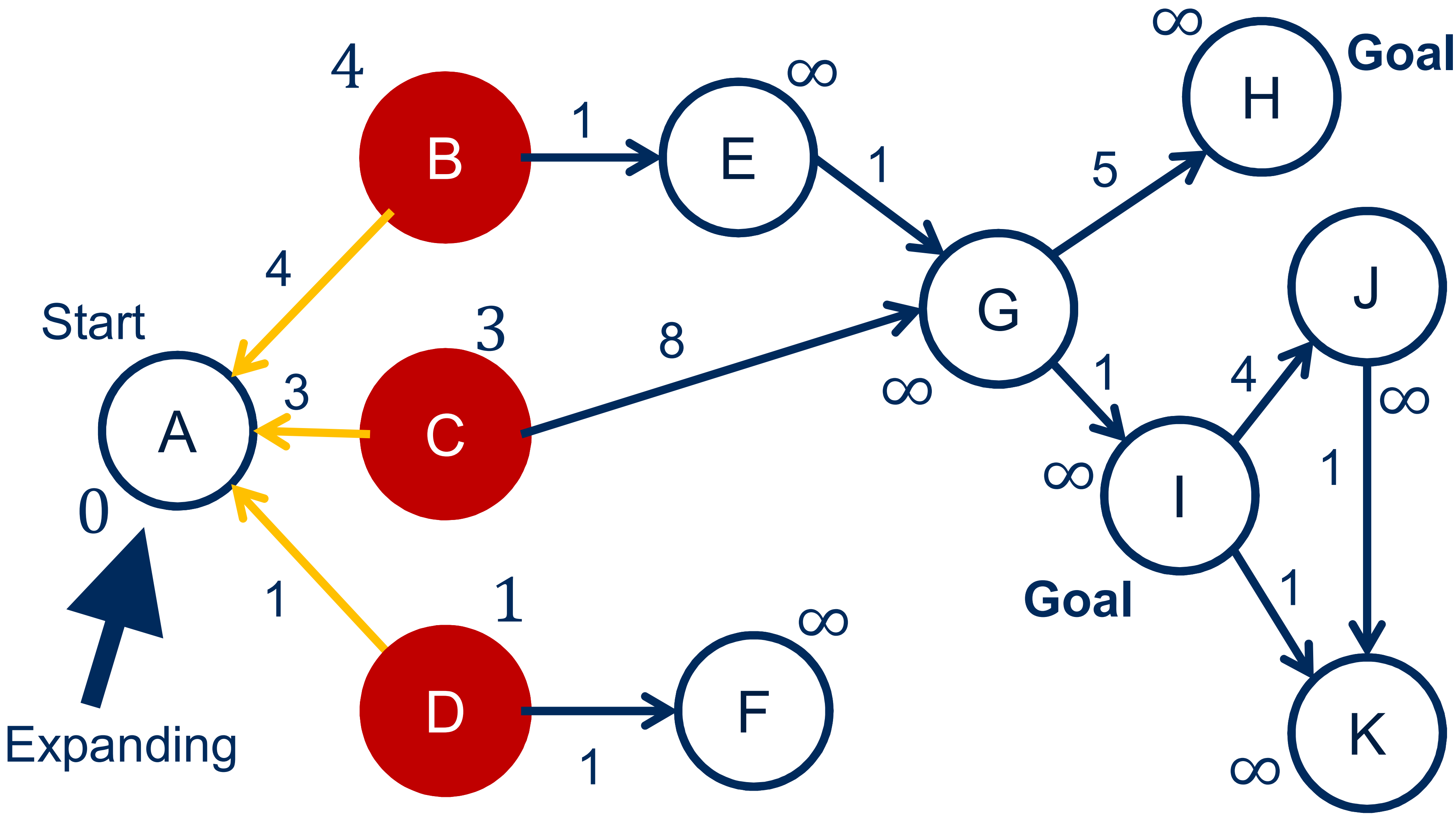


UNIVERSITY OF TORONTO

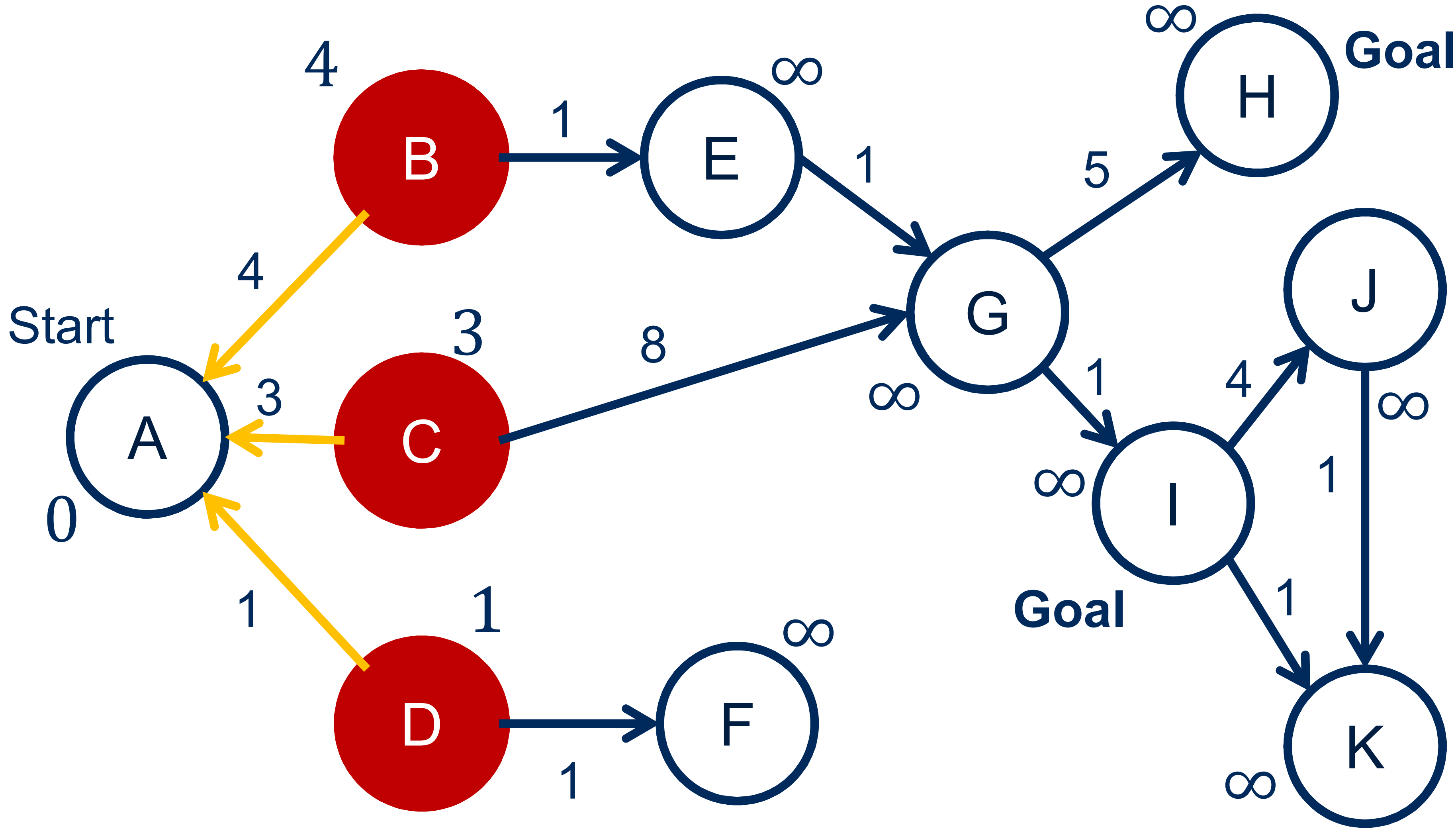
Dijkstra's Search



Dijkstra's Search

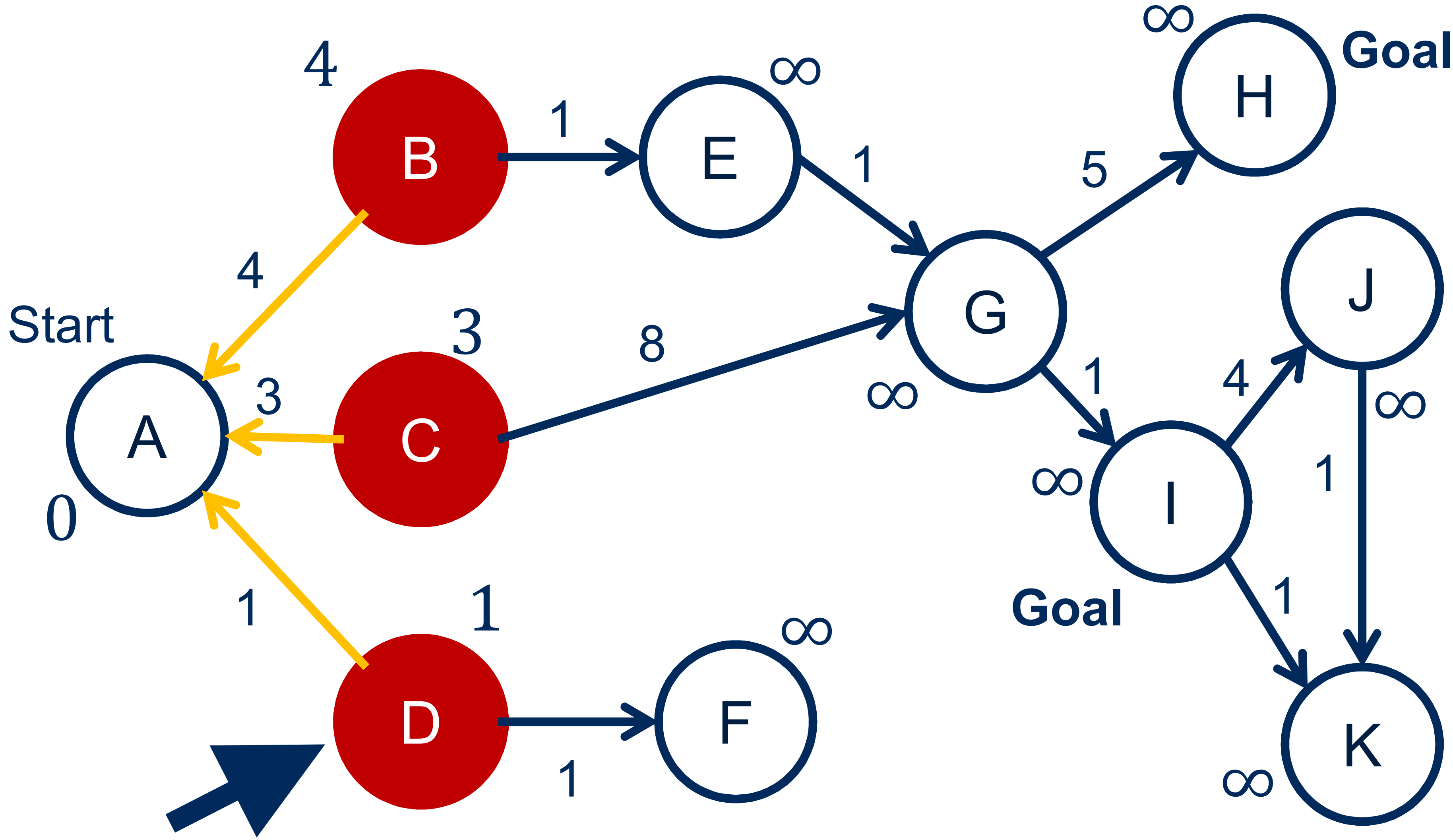


Dijkstra's Search



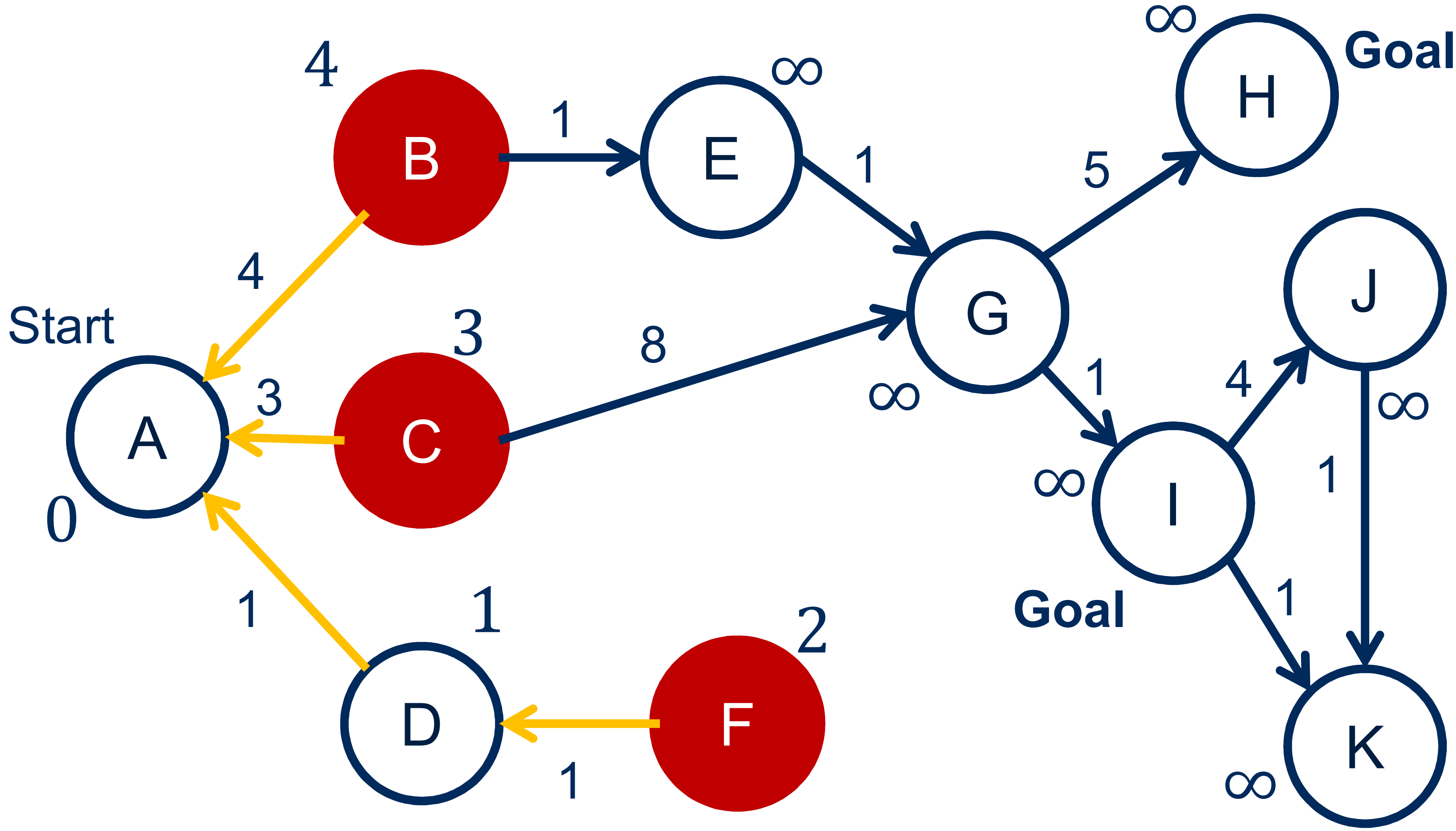
UNIVERSITY OF TORONTO

Dijkstra's Search



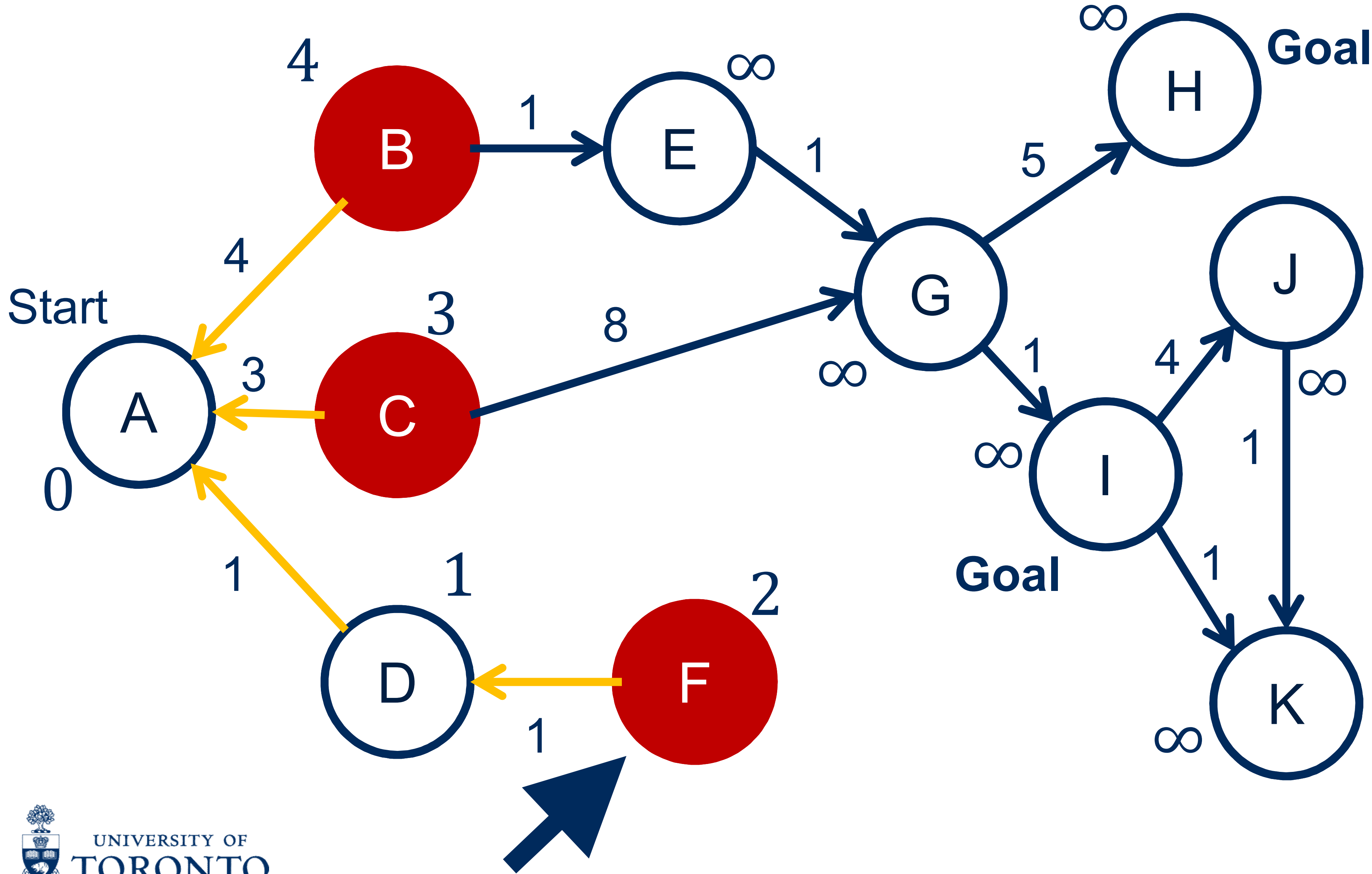
UNIVERSITY OF TORONTO

Dijkstra's Search



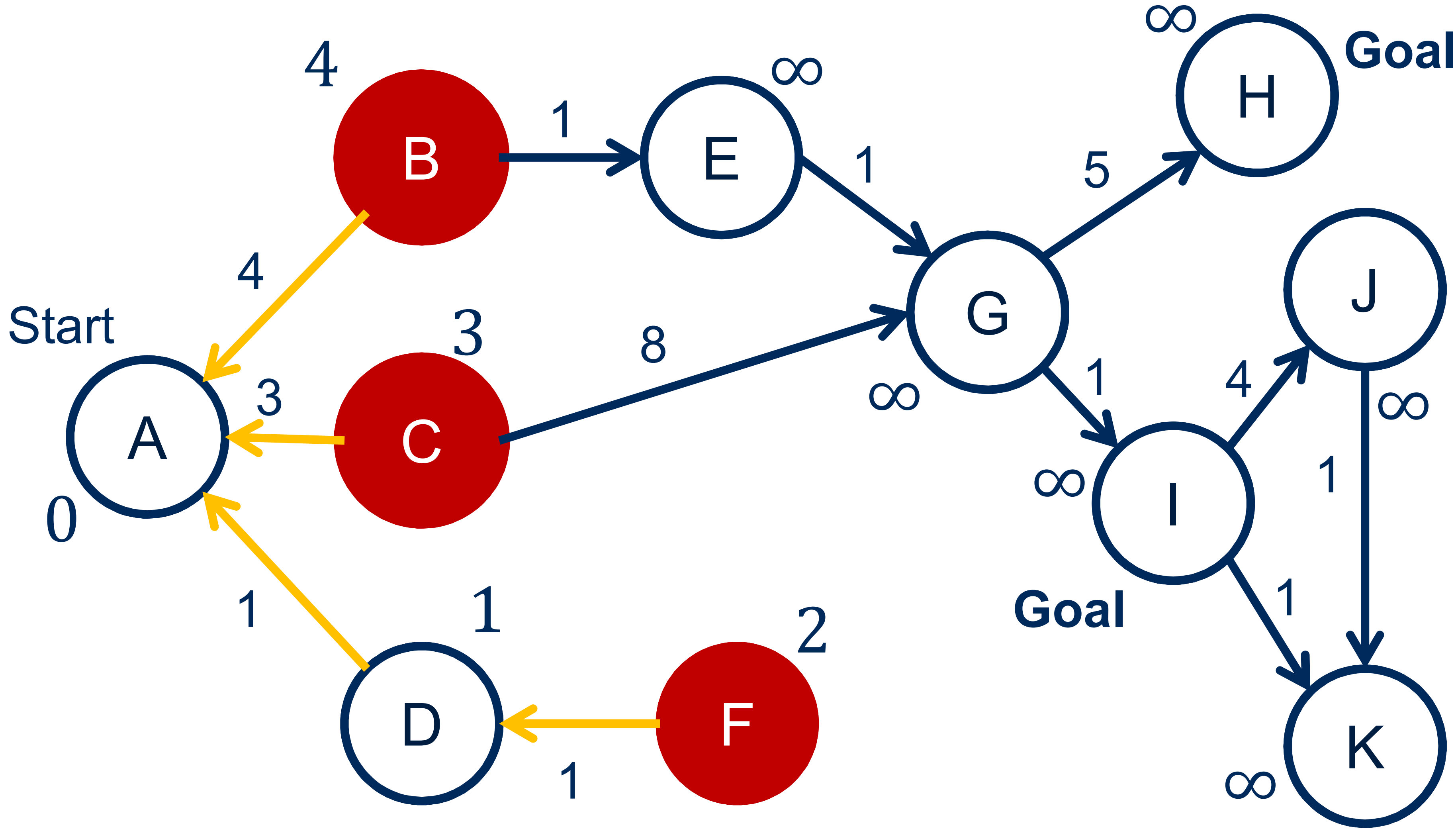
UNIVERSITY OF TORONTO

Dijkstra's Search



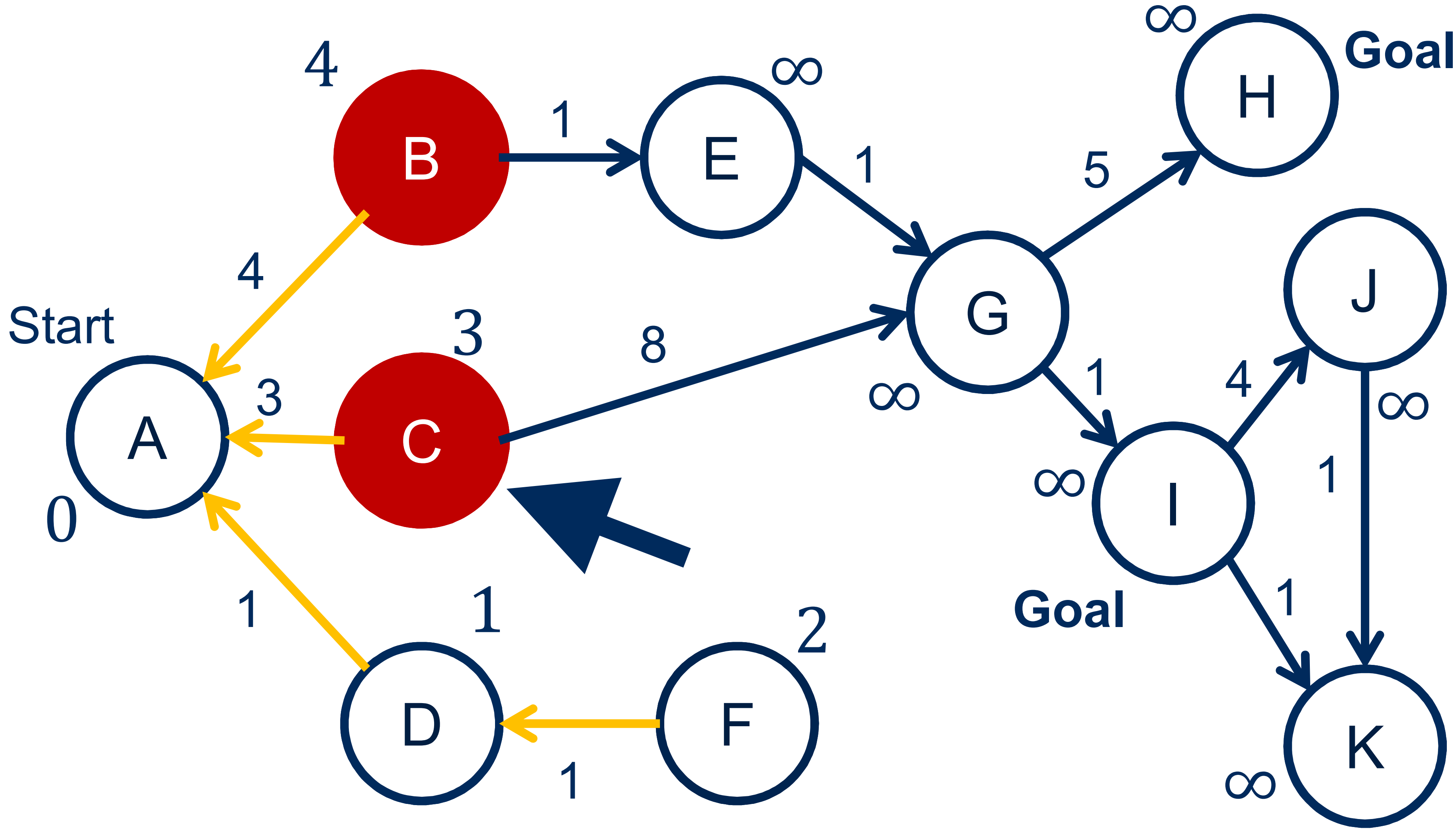
UNIVERSITY OF TORONTO

Dijkstra's Search



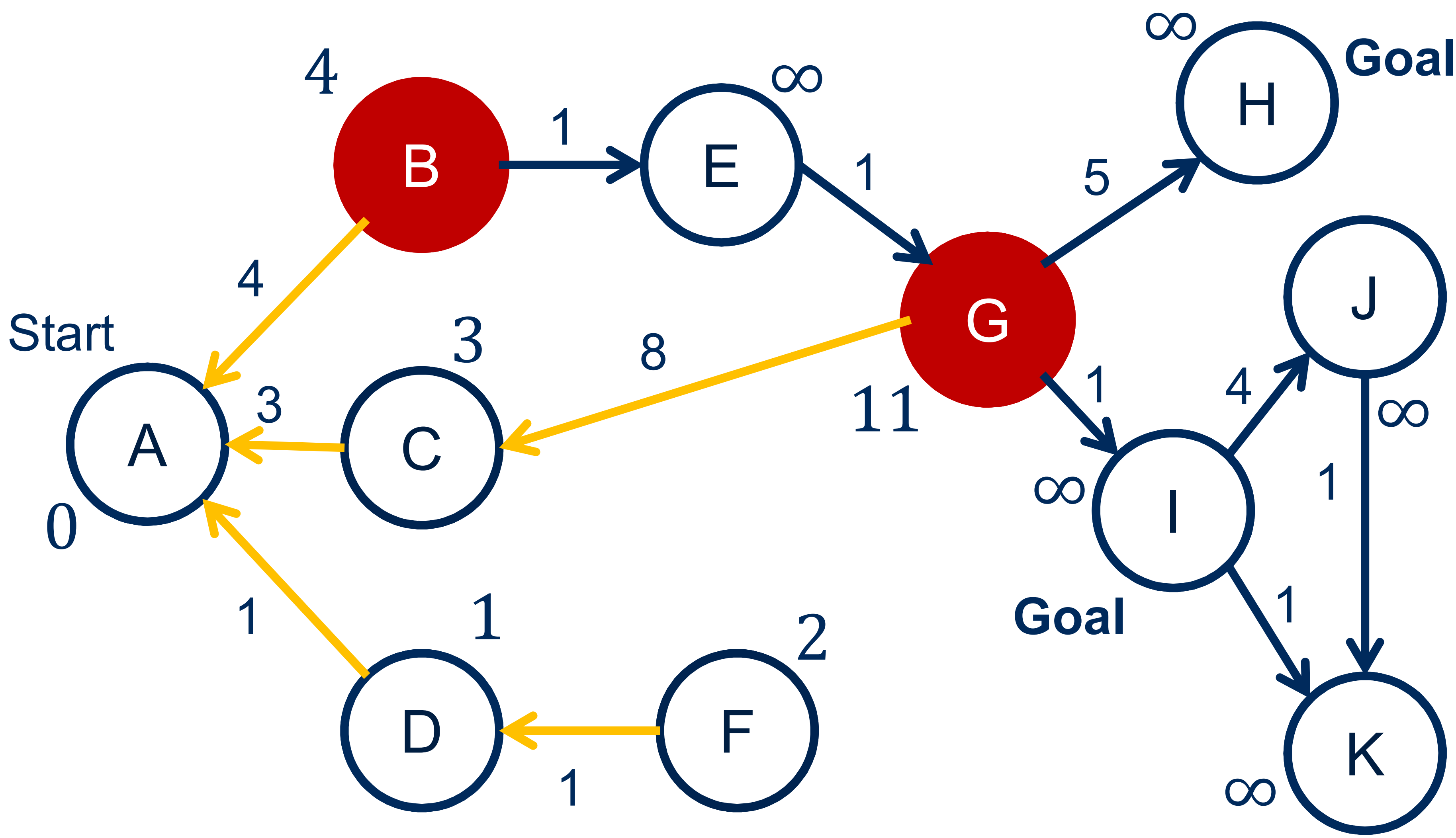
UNIVERSITY OF TORONTO

Dijkstra's Search



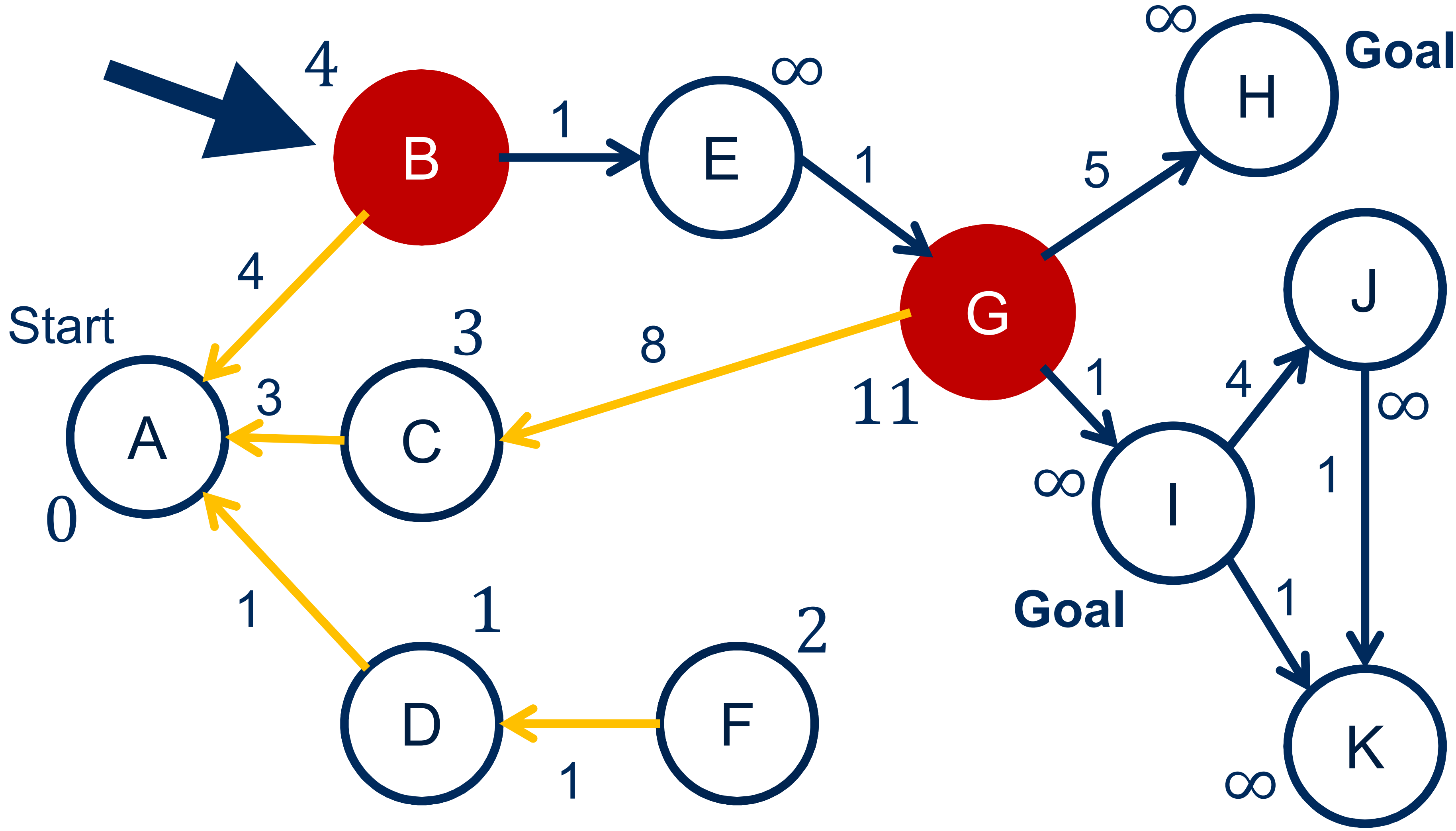
UNIVERSITY OF TORONTO

Dijkstra's Search



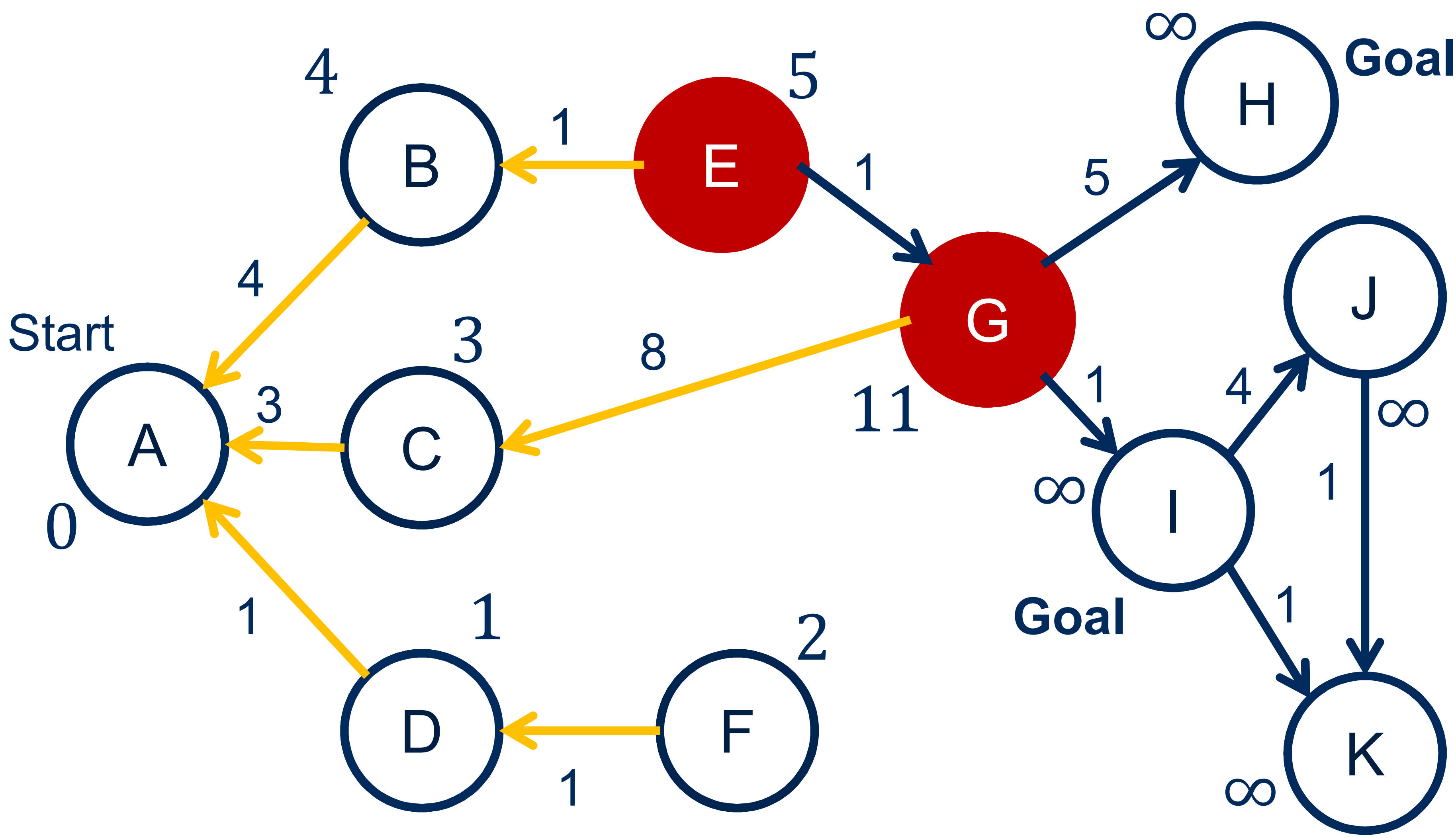
UNIVERSITY OF TORONTO

Dijkstra's Search



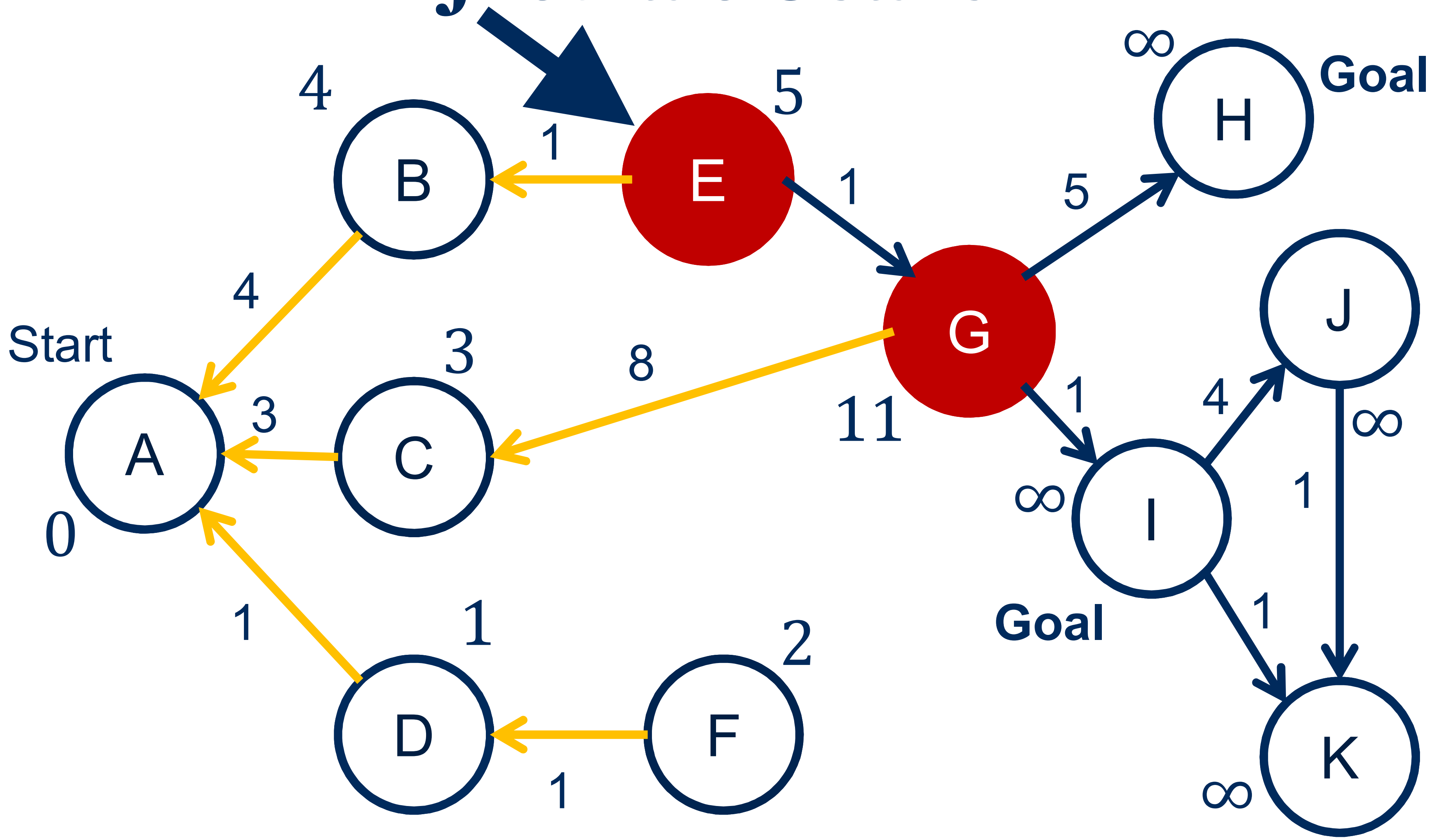
UNIVERSITY OF TORONTO

Dijkstra's Search



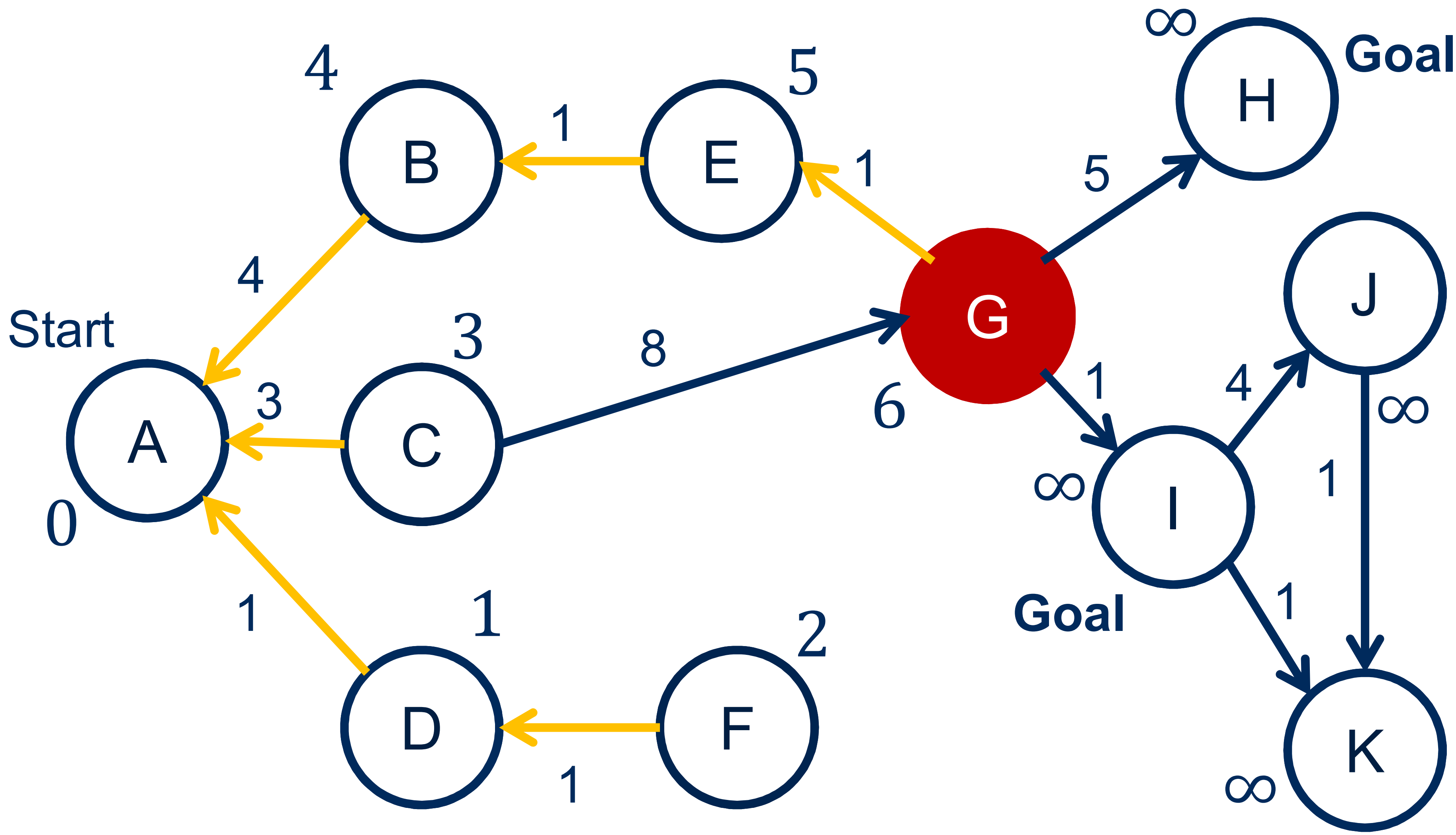
UNIVERSITY OF TORONTO

Dijkstra's Search



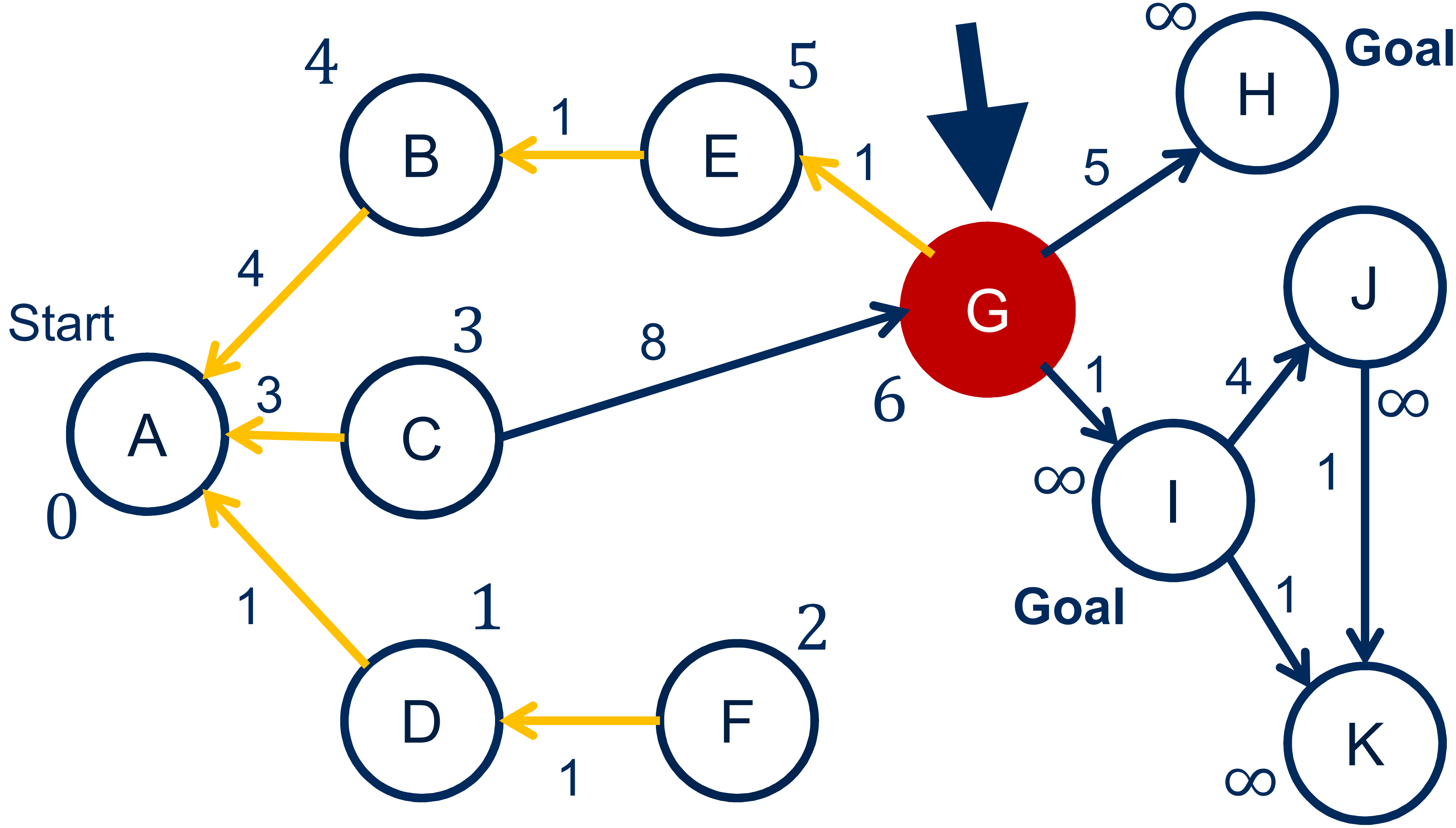
UNIVERSITY OF TORONTO

Dijkstra's Search



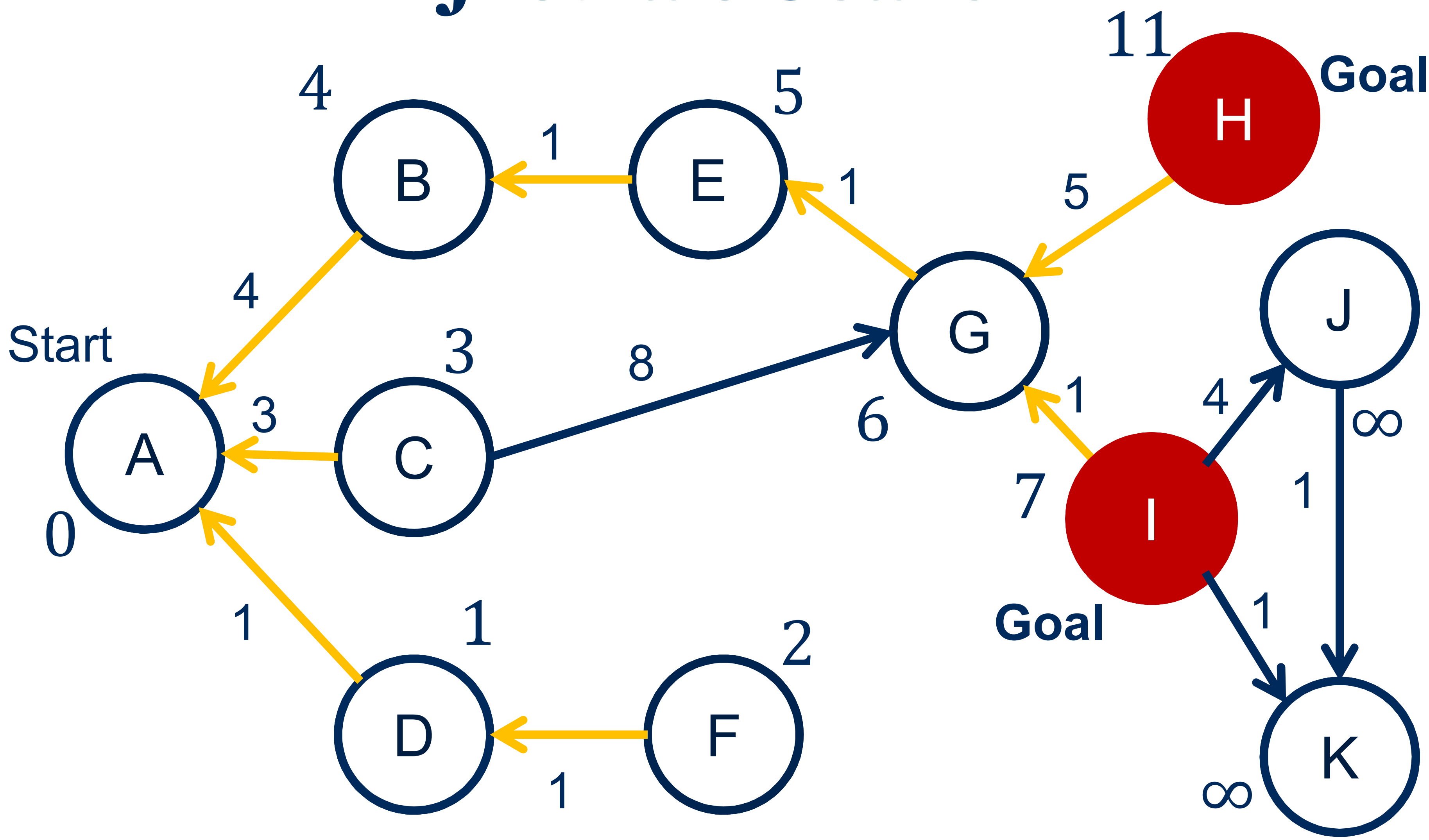
UNIVERSITY OF TORONTO

Dijkstra’s Search

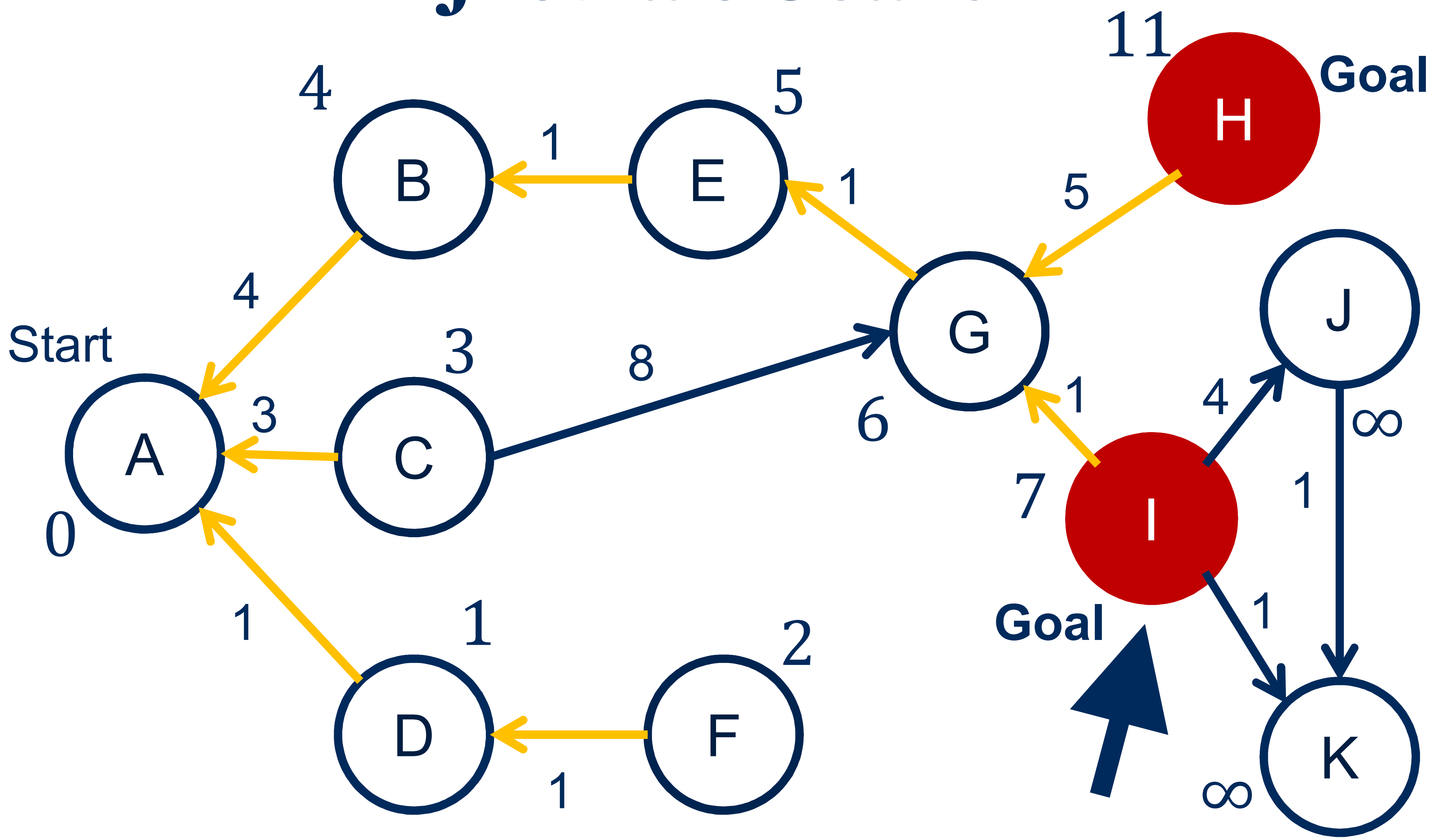


UNIVERSITY OF TORONTO

Dijkstra's Search

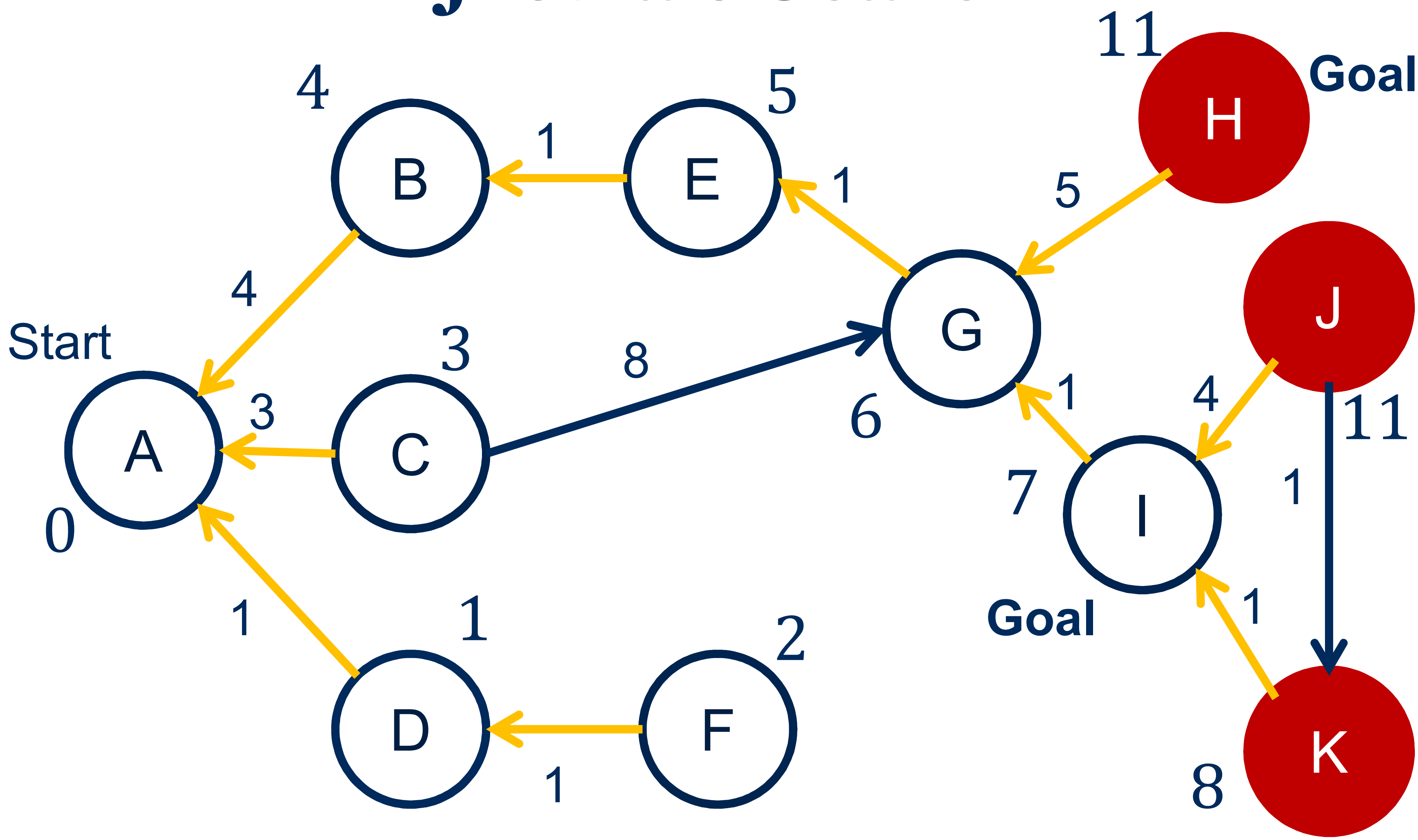


Dijkstra's Search

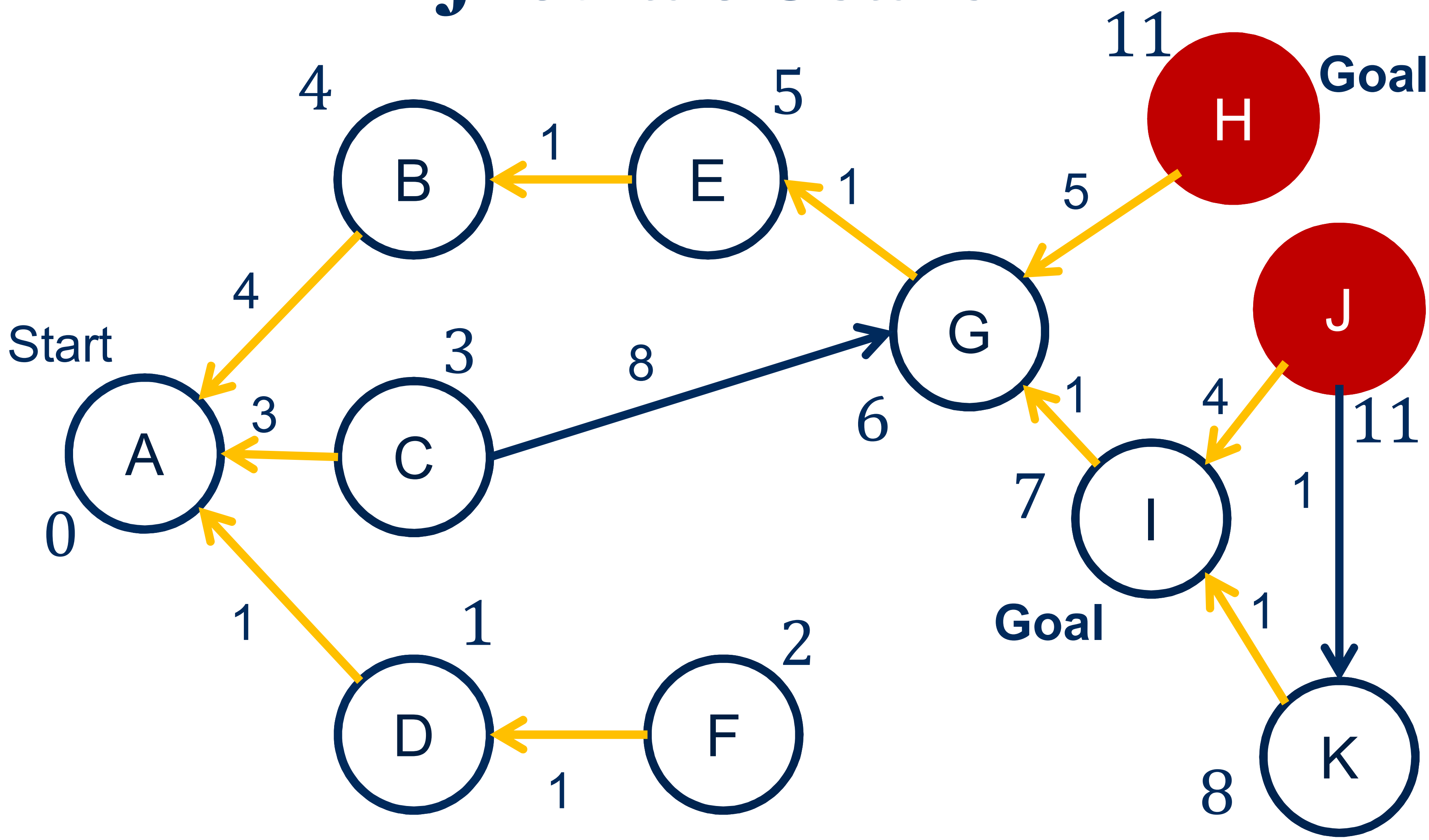


UNIVERSITY OF TORONTO

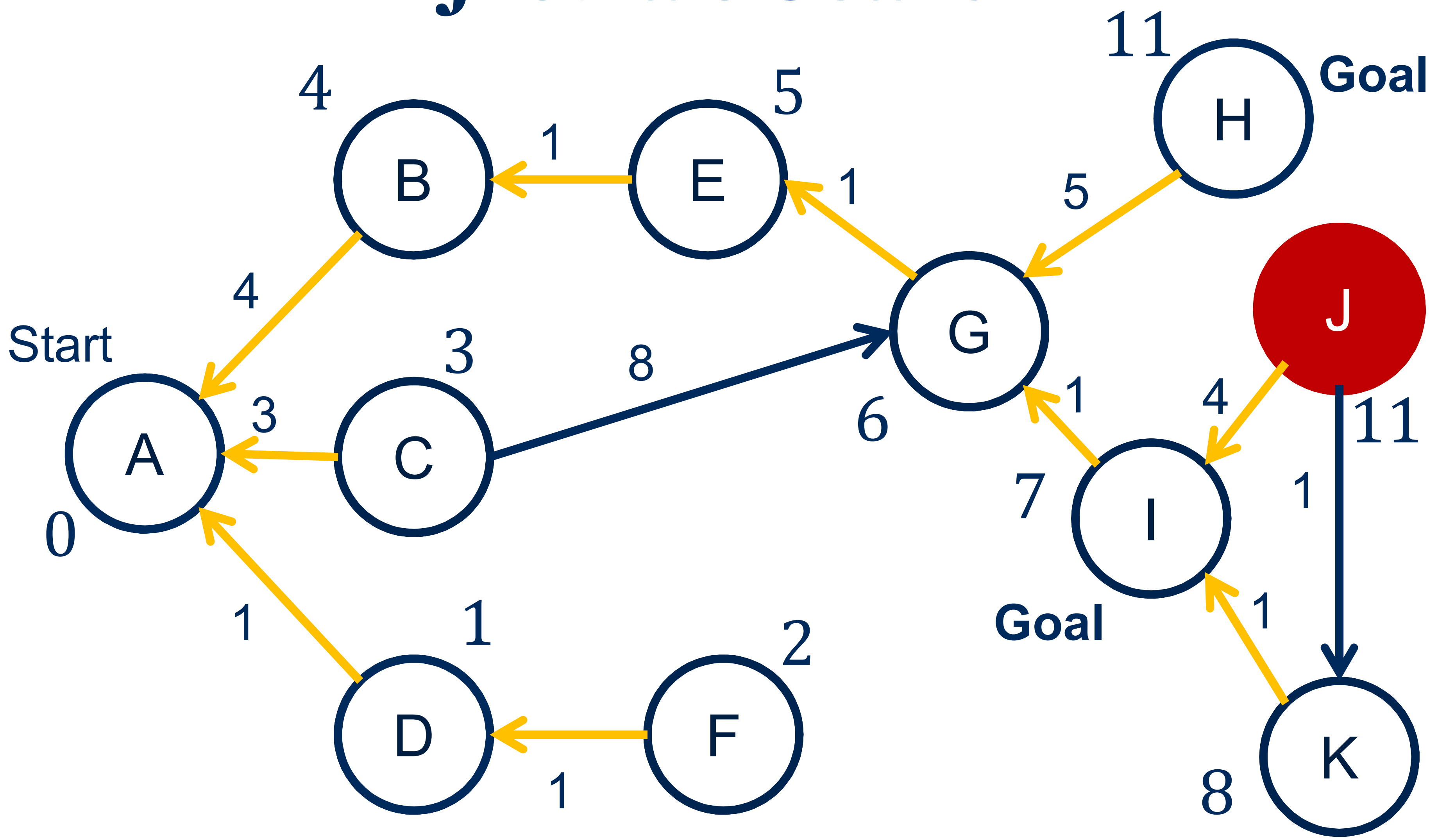
Dijkstra's Search



Dijkstra's Search

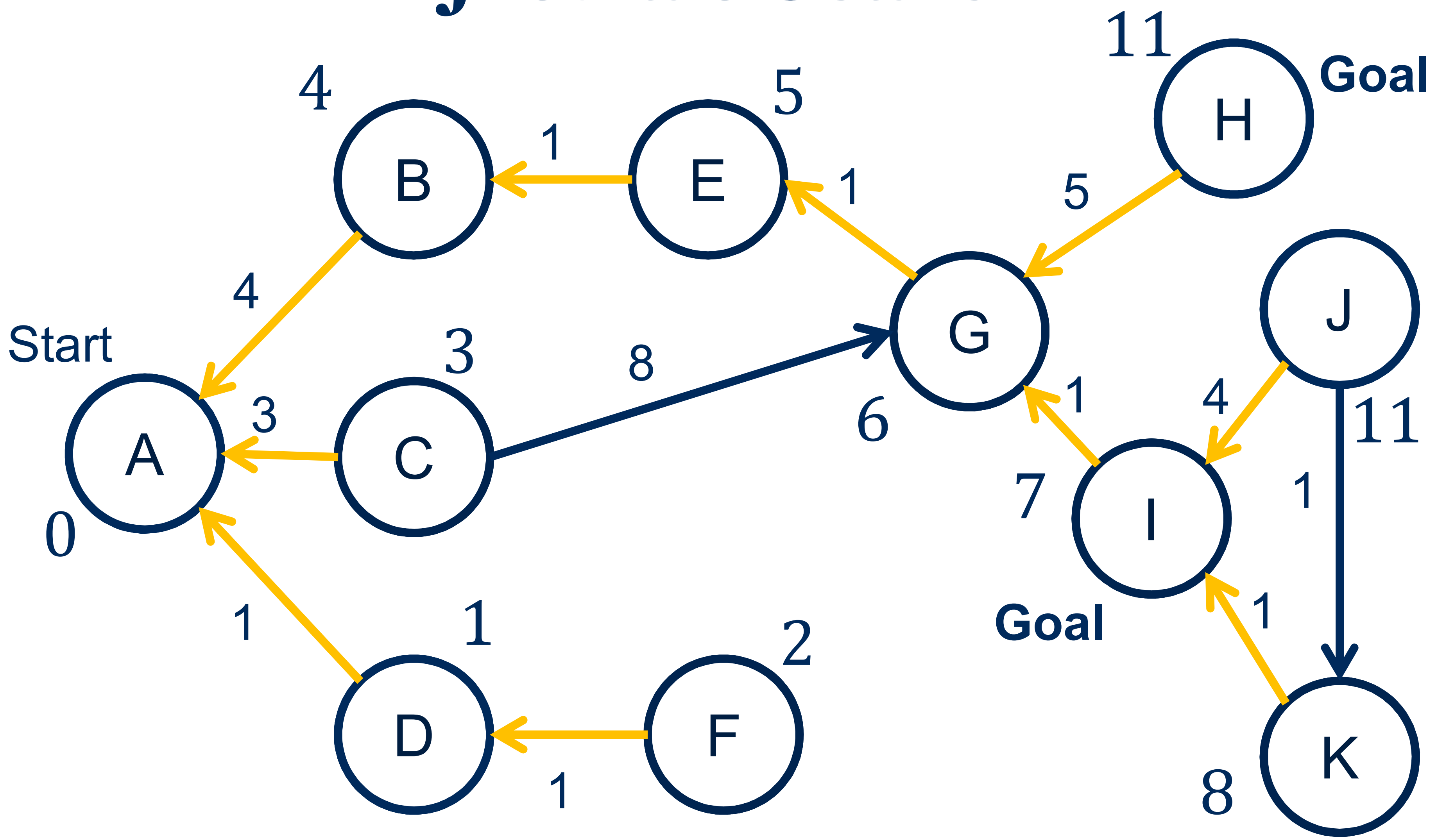


Dijkstra's Search



UNIVERSITY OF TORONTO

Dijkstra's Search



UNIVERSITY OF TORONTO

Dijkstra's Search

- Now have a shortest path to every vertex in graph
  - Can iterate through goals and return lowest-cost solution

- Dijkstra's search will look at O(|V|) vertices
  - So planning can be done in poly-time, right?

Dijkstra's Search

- Dijkstra's search is polynomial in |V|
  - But not polynomial in the given problem representation

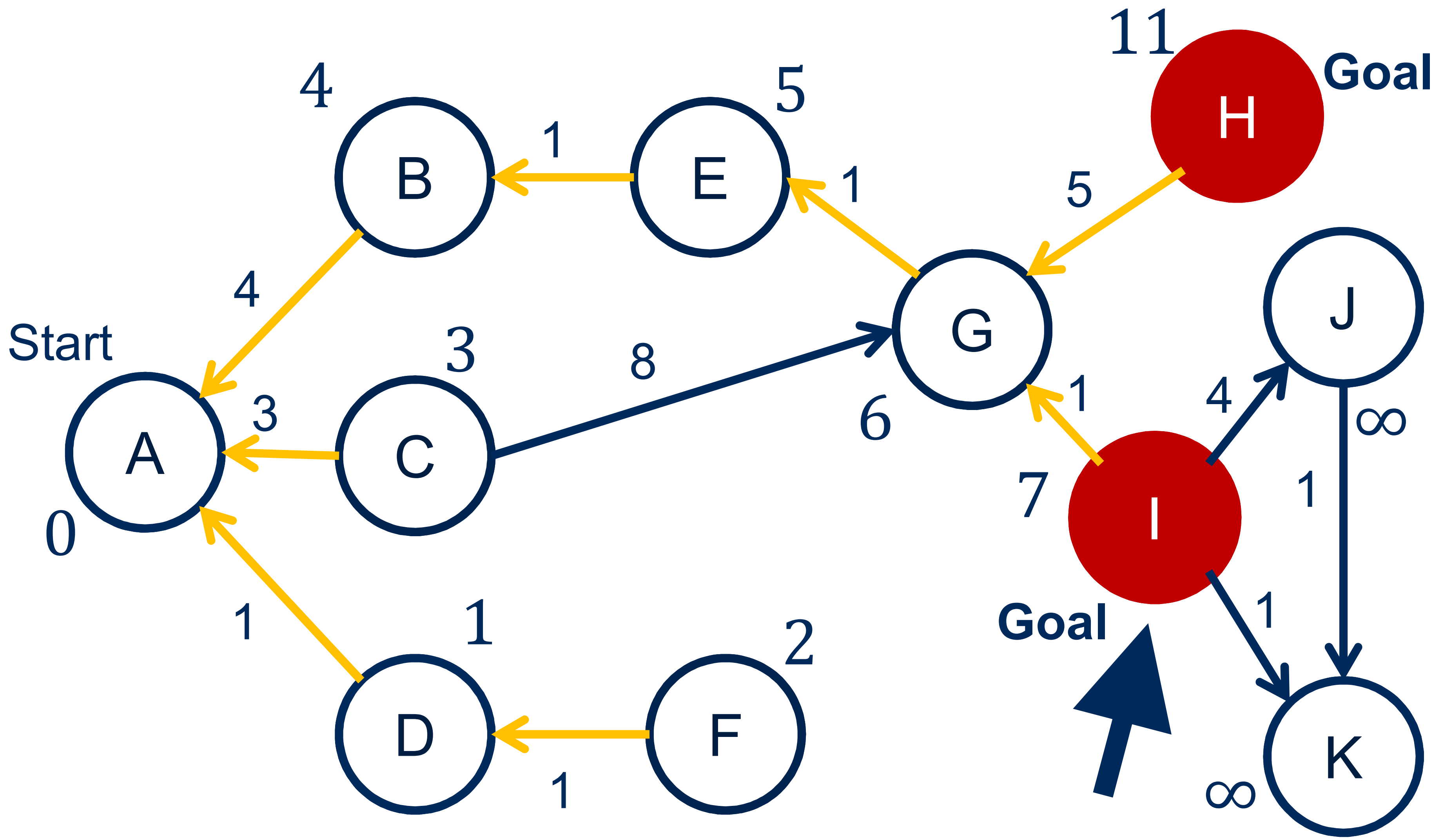
- Consider floortile on an N x N grid with K locations that need to be painted
  - Robot can be in either LOADED-B or LOADED-R
  - Robot can be in any of N x N locations
  - Any combination of the K locations can be painted
  - $O(2 \cdot N \cdot N \cdot 2^K)$ states

Uniform Cost Search

- Two changes to Dijkstra's Algorithm

1. Stop after a goal node is first expanded.

Uniform Cost Search



UNIVERSITY OF TORONTO

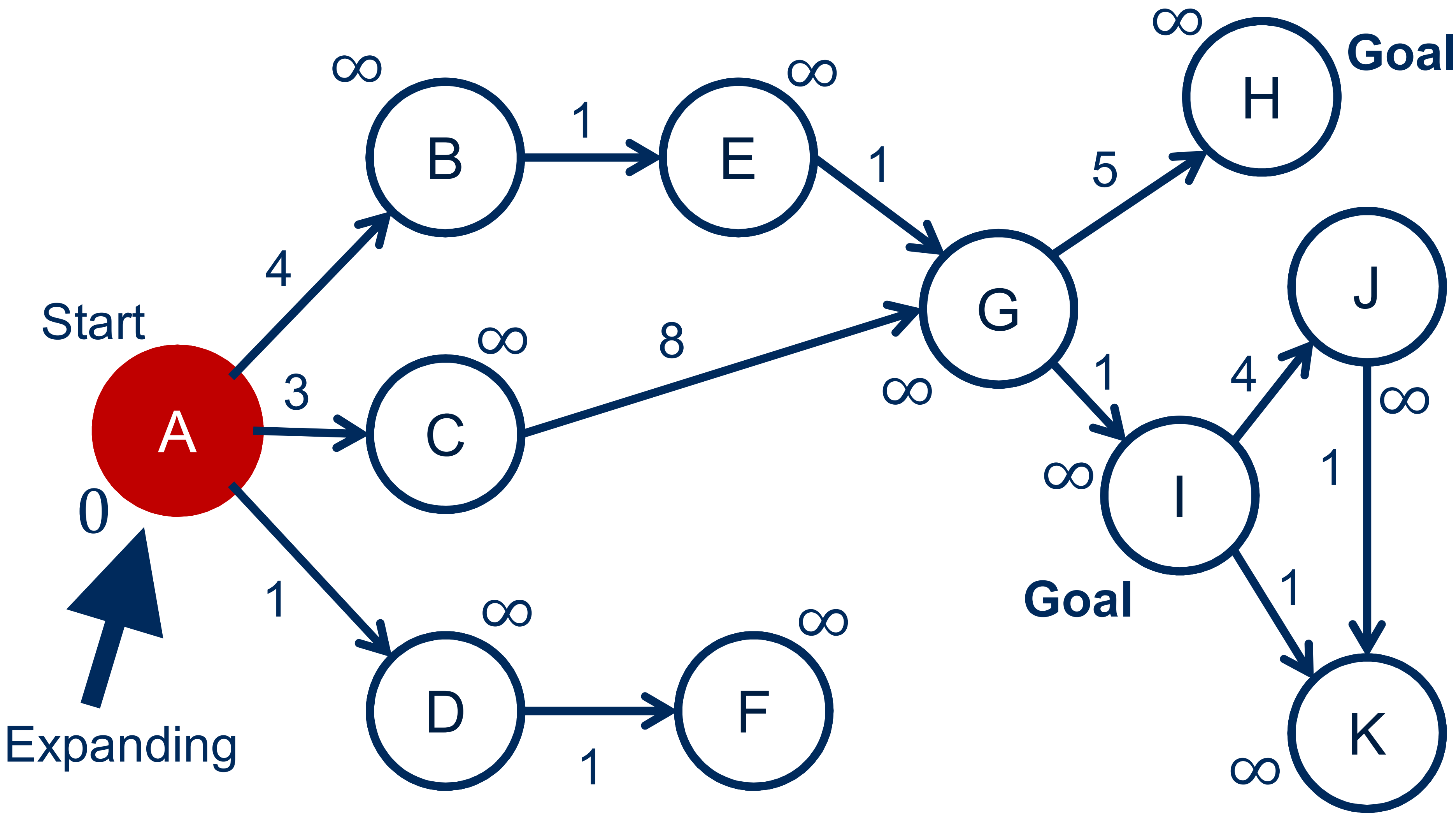
Uniform Cost Search

- Two changes to Dijkstra's Algorithm

1. Stop after a goal node is first expanded.

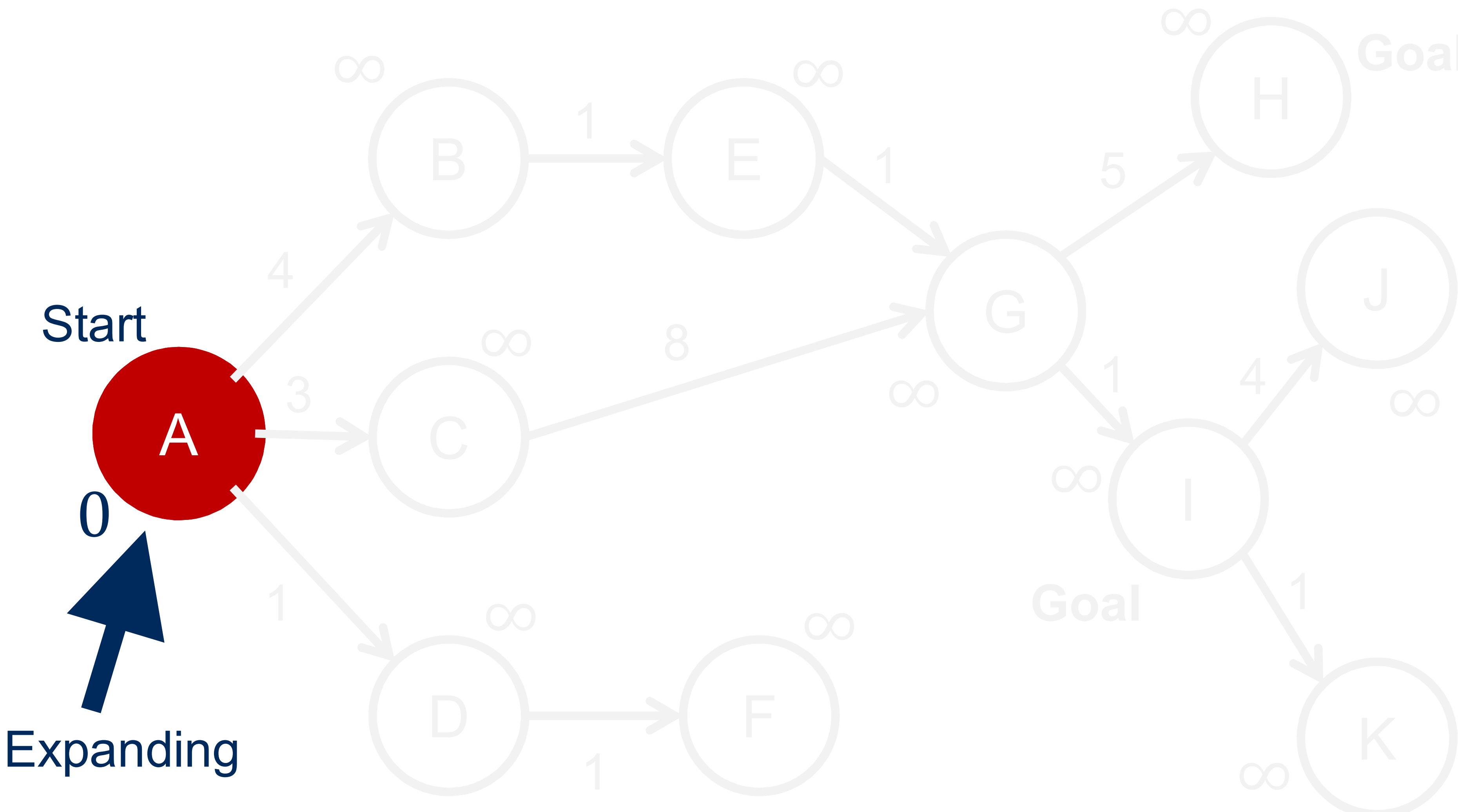
2. Use implicit action definition to generate the graph on-the-fly.

Dijkstra's Search



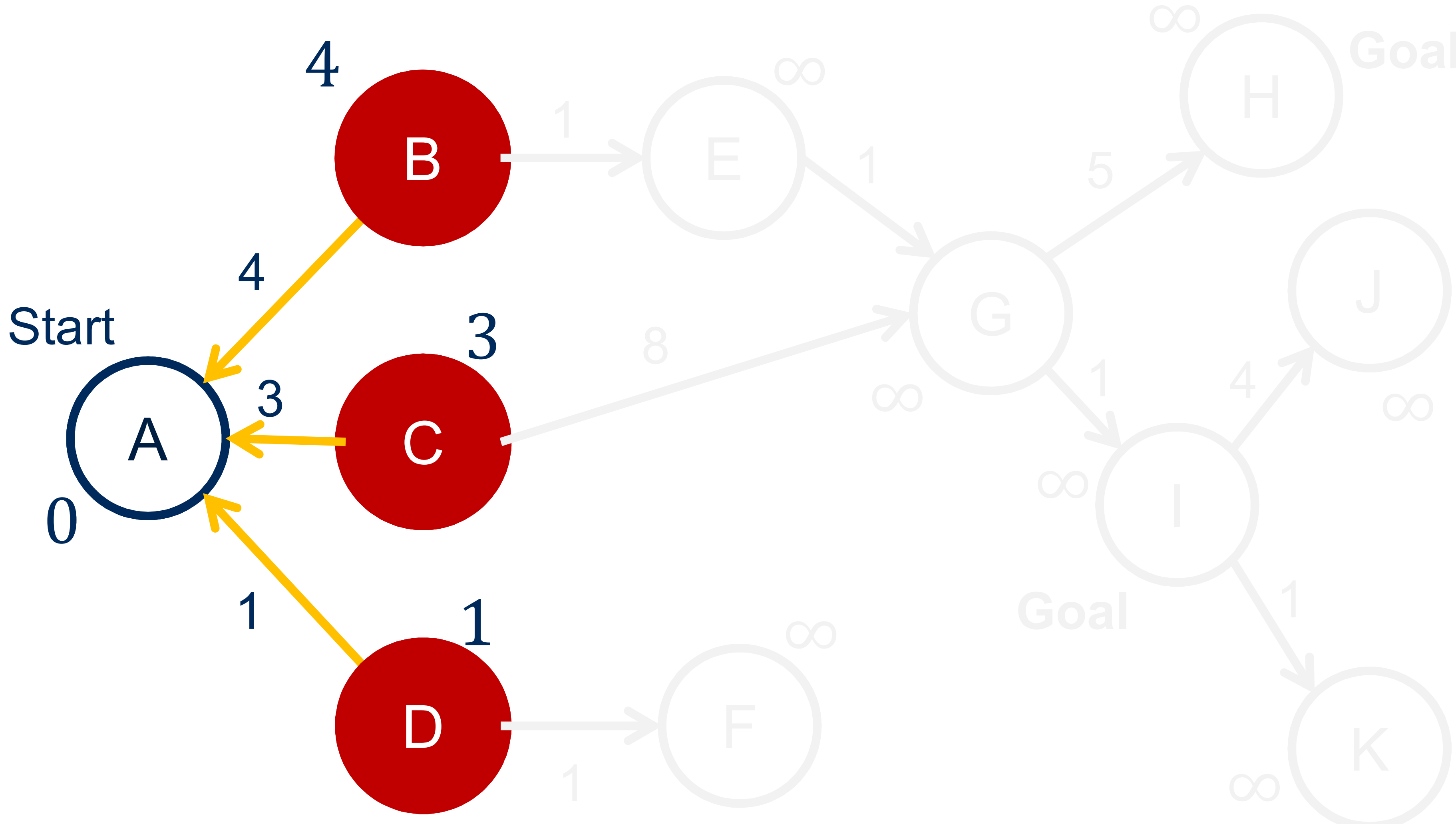
UNIVERSITY OF TORONTO

Uniform Cost Search



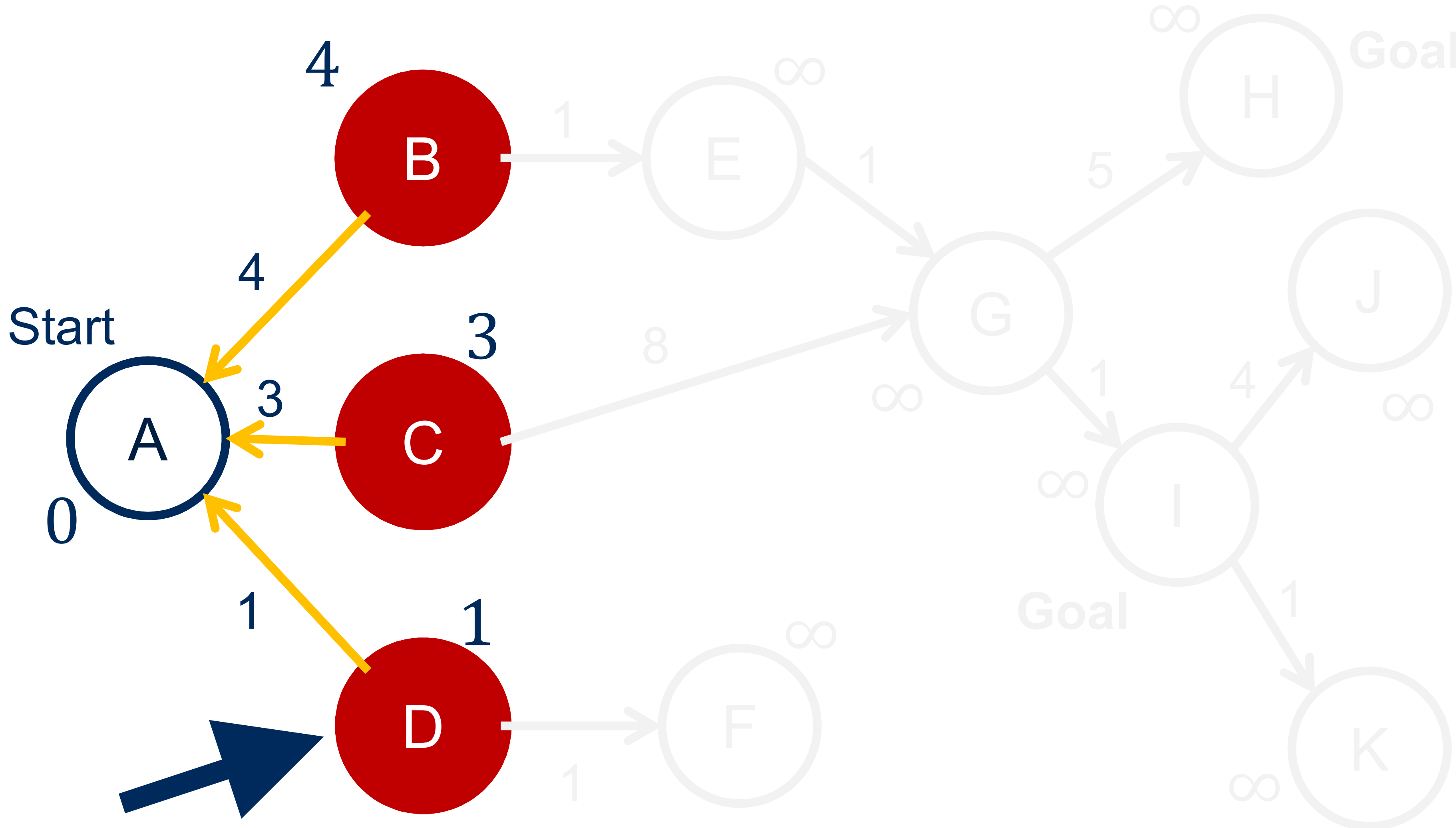
UNIVERSITY OF TORONTO

Uniform Cost Search



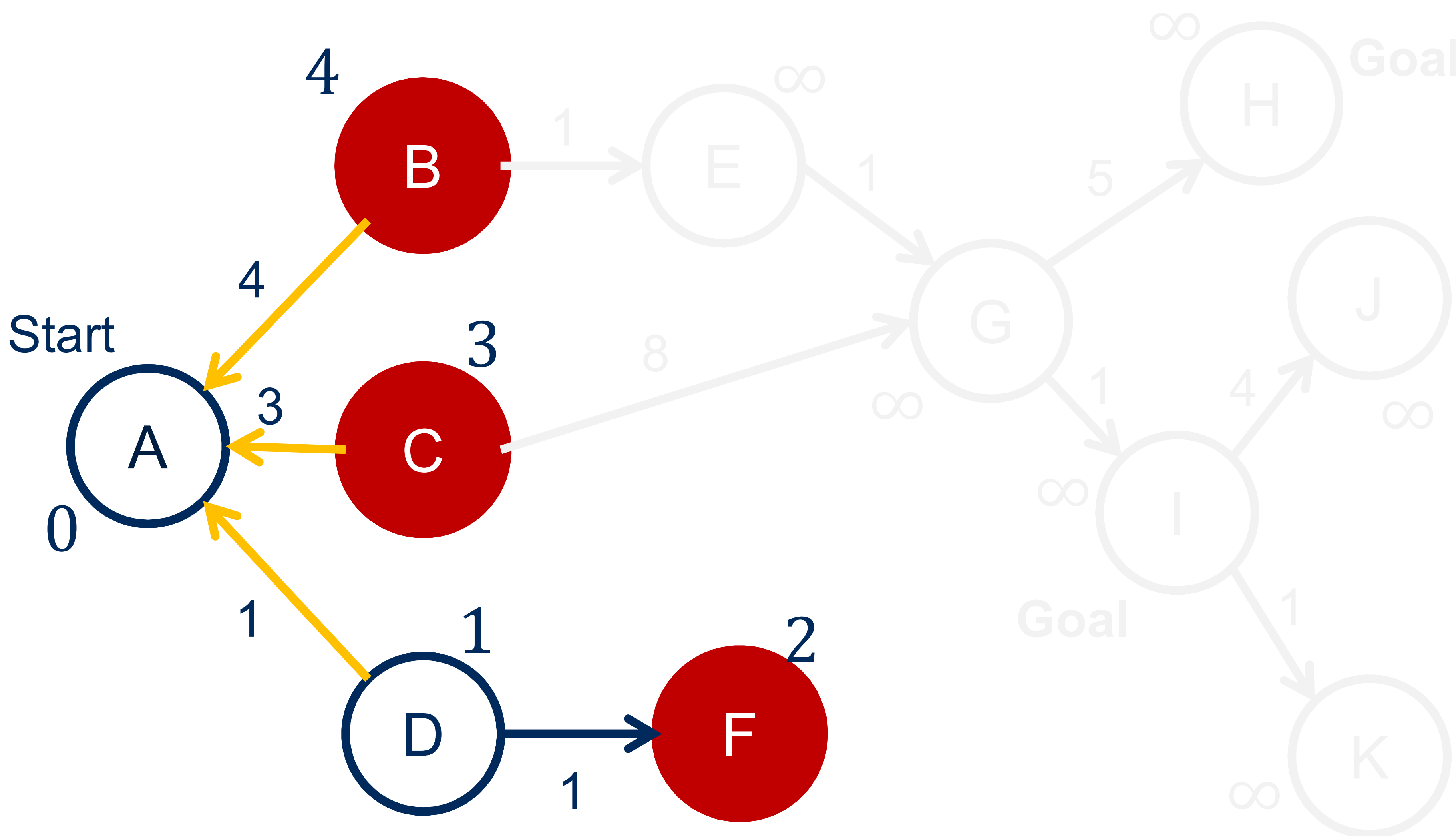
UNIVERSITY OF TORONTO

Uniform Cost Search



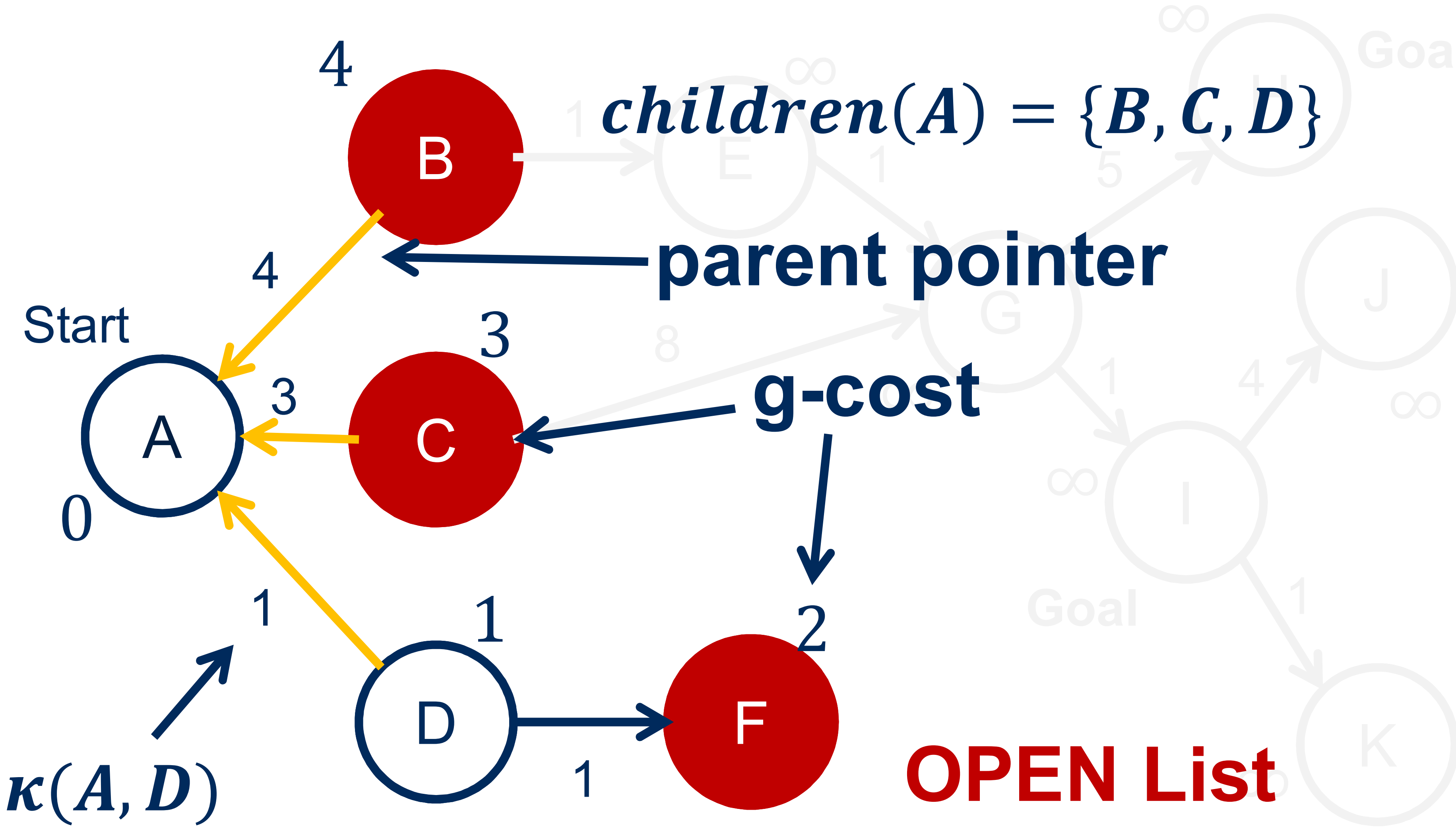
UNIVERSITY OF TORONTO

Uniform Cost Search



UNIVERSITY OF TORONTO

Uniform Cost Search



UNIVERSITY OF TORONTO

```
def UniformCostSearch(s_I):
    OPEN ← {s_I}, CLOSED ← {},
    g(s_I) = 0, parent(s_I) = ∅
    while OPEN ≠ {}:
        p ← argmin_{s′∈OPEN} g(s′)
        if p is a goal, return path to p
        for c ∈ children(p):
            if c ∉ OPEN ∪ CLOSED:
                g(c) = g(p) + κ(p, c)
                parent(c) = p
                OPEN ← OPEN ∪ {c}
            else if g(c) > g(p) + κ(p, c):
                g(c) = g(p) + κ(p, c)
                parent(c) = p
                if c ∈ CLOSED:
                    OPEN ← OPEN ∪ {c}
                    CLOSED ← CLOSED − {c}
        OPEN ← OPEN − {p}, CLOSED ← CLOSED ∪ {p}
    return No solution exists
```

def UniformCostSearch(s_I):

$OPEN \leftarrow \{s_I\}, CLOSED \leftarrow \{\},$
$g(s_I) = 0, parent(s_I) = \emptyset$

while $OPEN \neq \{\}$:

p $\leftarrow$ $\text{argmin}_{\{s' \in OPEN\}} g(s')$

if p is a goal, **return** path to p

for $c \in children(p)$:

if $c \notin OPEN \cup CLOSED$:

$g(c) = g(p) + \kappa(p, c)$
$parent(c) = p$
$OPEN \leftarrow OPEN \cup \{c\}$

else if $g(c) > g(p) + \kappa(p, c)$:

$g(c) = g(p) + \kappa(p, c)$
$parent(c) = p$

if $c \in CLOSED$:

$OPEN \leftarrow OPEN \cup \{c\}$
$CLOSED \leftarrow CLOSED - \{c\}$

$OPEN \leftarrow OPEN - \{p\}, CLOSED \leftarrow CLOSED \cup \{p\}$

return No solution exists

Initialize Search

Get node from OPEN

```
def UniformCostSearch(s_I):
    OPEN ← {s_I}, CLOSED ← {},
    g(s_I) = 0, parent(s_I) = ∅
    while OPEN ≠ {}:
        p ← argmin_{s' ∈ OPEN} g(s')
        if p is a goal, return path to p
        for c ∈ children(p):
            if c ∉ OPEN ∪ CLOSED:
                g(c) = g(p) + κ(p, c)
                parent(c) = p
                OPEN ← OPEN ∪ {c}
            else if g(c) > g(p) + κ(p, c):
                g(c) = g(p) + κ(p, c)
                parent(c) = p
                if c ∈ CLOSED:
                    OPEN ← OPEN ∪ {c}
                    CLOSED ← CLOSED − {c}
        OPEN ← OPEN − {p}, CLOSED ← CLOSED ∪ {p}
    return No solution exists
```

Generate and handle children

```
def UniformCostSearch(s_I):
    OPEN ← {s_I}, CLOSED ← {},
    g(s_I) = 0, parent(s_I) = ∅
    while OPEN ≠ {}:
        p ← argmin_{s'∈OPEN} g(s')
        if p is a goal, return path to p
        for c ∈ children(p):
            if c ∉ OPEN ∪ CLOSED:
                g(c) = g(p) + κ(p, c)
                parent(c) = p
                OPEN ← OPEN ∪ {c}
            else if g(c) > g(p) + κ(p, c):
                g(c) = g(p) + κ(p, c)
                parent(c) = p
                if c ∈ CLOSED:
                    OPEN ← OPEN ∪ {c}
                    CLOSED ← CLOSED − {c}
        OPEN ← OPEN − {p}, CLOSED ← CLOSED ∪ {p}
    return No solution exists
```

Generate and handle children

def UniformCostSearch(s_I):

$OPEN \leftarrow \{s_I\}, CLOSED \leftarrow \{\},$

$g(s_I) = 0, parent(s_I) = \emptyset$

while $OPEN \neq \{\}$:

p $\leftarrow \text{argmin}_{\{s' \in OPEN\}} g(s')$

if p is a goal, **return** path to p

for $c \in children(p)$:

if $c \notin OPEN \cup CLOSED$:

$g(c) = g(p) + \kappa(p, c)$

$parent(c) = p$

$OPEN \leftarrow OPEN \cup \{c\}$

else if $g(c) > g(p) + \kappa(p, c)$:

$g(c) = g(p) + \kappa(p, c)$

$parent(c) = p$

if $c \in CLOSED$:

$OPEN \leftarrow OPEN \cup \{c\}$

$CLOSED \leftarrow CLOSED - \{c\}$

$OPEN \leftarrow OPEN - \{p\}, CLOSED \leftarrow CLOSED \cup \{p\}$

return No solution exists

def UniformCostSearch(s_I):

$OPEN \leftarrow \{s_I\}, CLOSED \leftarrow \{\},$
$g(s_I) = 0, parent(s_I) = \emptyset$
while $OPEN \neq \{\}$:
p $\leftarrow \text{argmin}_{\{s' \in OPEN\}} g(s')$
if p is a goal, **return** path to p
for $c \in children(p)$:
if $c \notin OPEN \cup CLOSED$:
$g(c) = g(p) + \kappa(p, c)$
$parent(c) = p$
$OPEN \leftarrow OPEN \cup \{c\}$
else if $g(c) > g(p) + \kappa(p, c)$:
$g(c) = g(p) + \kappa(p, c)$
$parent(c) = p$
if $c \in CLOSED$:
$OPEN \leftarrow OPEN \cup \{c\}$
$CLOSED \leftarrow CLOSED - \{c\}$
$OPEN \leftarrow OPEN - \{p\}, CLOSED \leftarrow CLOSED \cup \{p\}$
return No solution exists

Close expanded node

def UniformCostSearch(s_I):

$OPEN \leftarrow \{s_I\}, CLOSED \leftarrow \{\},$
$g(s_I) = 0, parent(s_I) = \emptyset$

while $OPEN \neq \{\}$:

p ← argmin$_{\{s' \in OPEN\}} g(s')$

if p is a goal, **return** path to p

for $c \in children(p)$:

if $c \notin OPEN \cup CLOSED$:

$g(c) = g(p) + \kappa(p, c)$
$parent(c) = p$

$OPEN \leftarrow OPEN \cup \{c\}$

else if $g(c) > g(p) + \kappa(p, c)$:

$g(c) = g(p) + \kappa(p, c)$
$parent(c) = p$

if $c \in CLOSED$:

$OPEN \leftarrow OPEN \cup \{c\}$

$CLOSED \leftarrow CLOSED - \{c\}$

CLOSED ← $CLOSED - \{p\}$

return No solution exists

Repeat

Uniform Cost Search

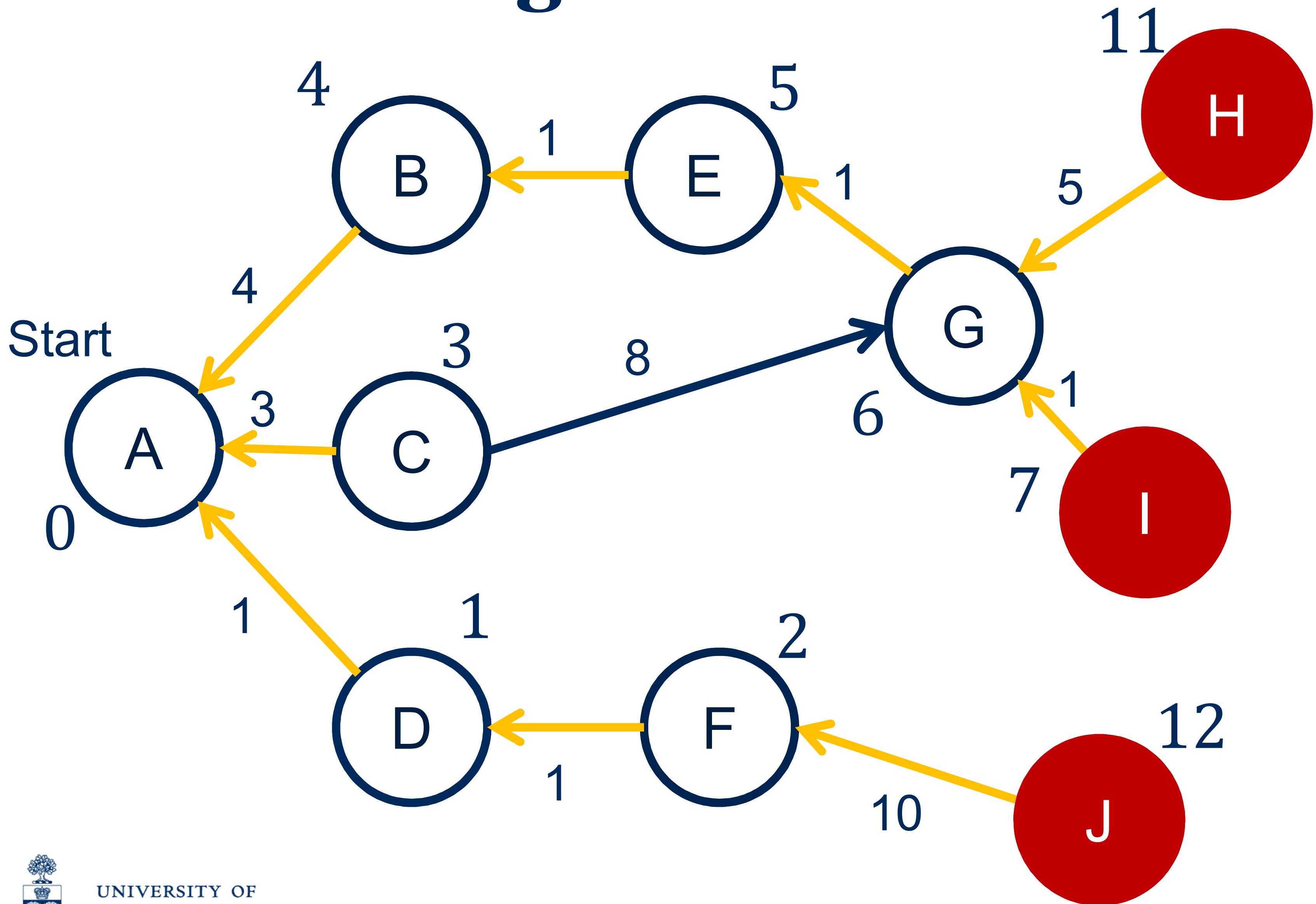
- UCS is completely **exhaustive** and **brute-force**
- Makes it prohibitively expensive

Depth	Nodes	Time	Memory
2	1100	.11 seconds	1 megabyte
4	111,100	11 seconds	106 megabytes
6	10^7	19 minutes	10 gigabytes
8	10^9	31 hours	1 terabytes
10	10^{11}	129 days	101 terabytes
12	10^{13}	35 years	10 petabytes
14	10^{15}	3,523 years	1 exabyte

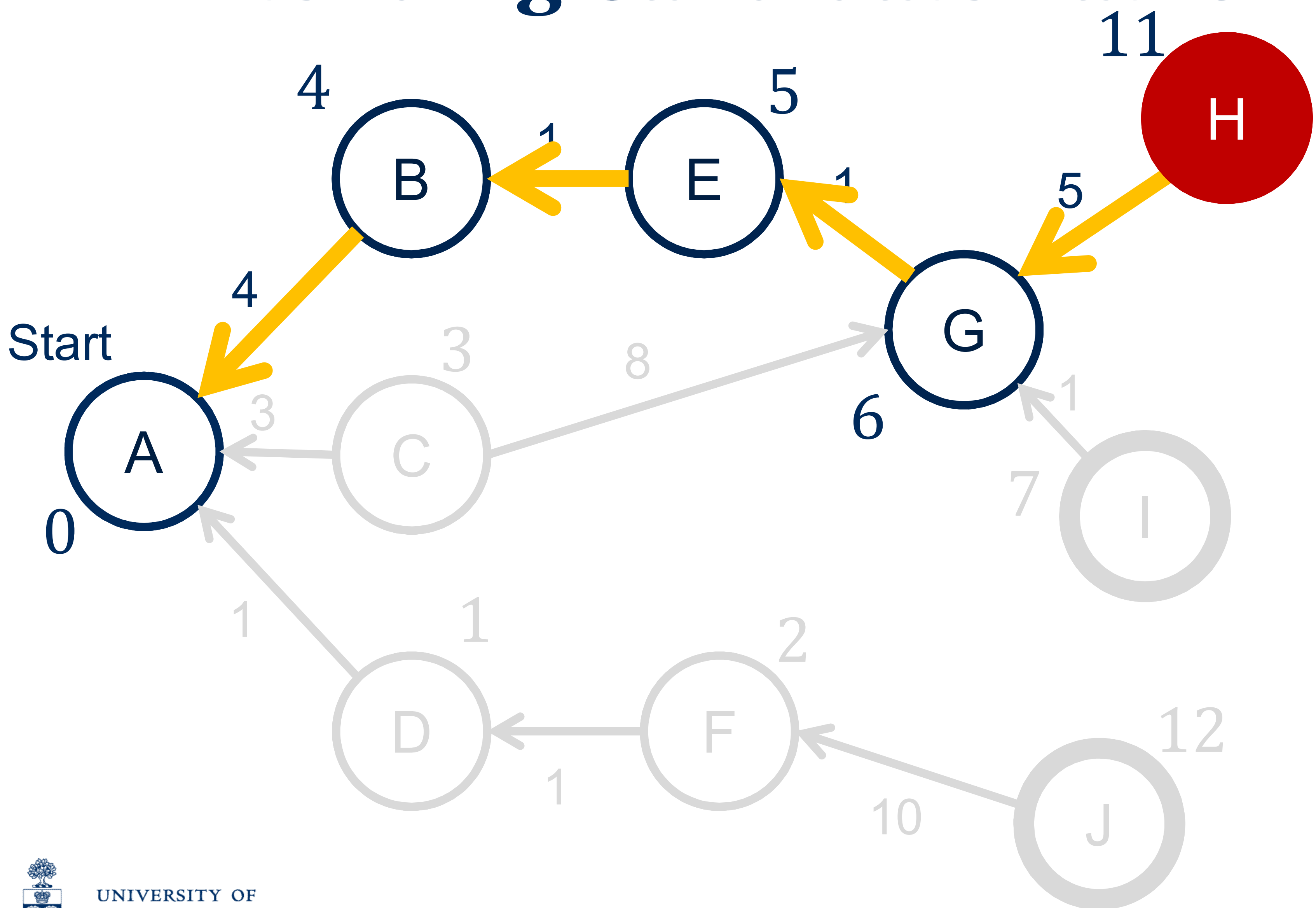
Figure 3.11 Time and memory requirements for breadth-first search. The numbers shown assume branching factor $b = 10$; 10,000 nodes/second; 1000 bytes/node.

UNIVERSITY OF TORONTO

Extending Candidate Paths

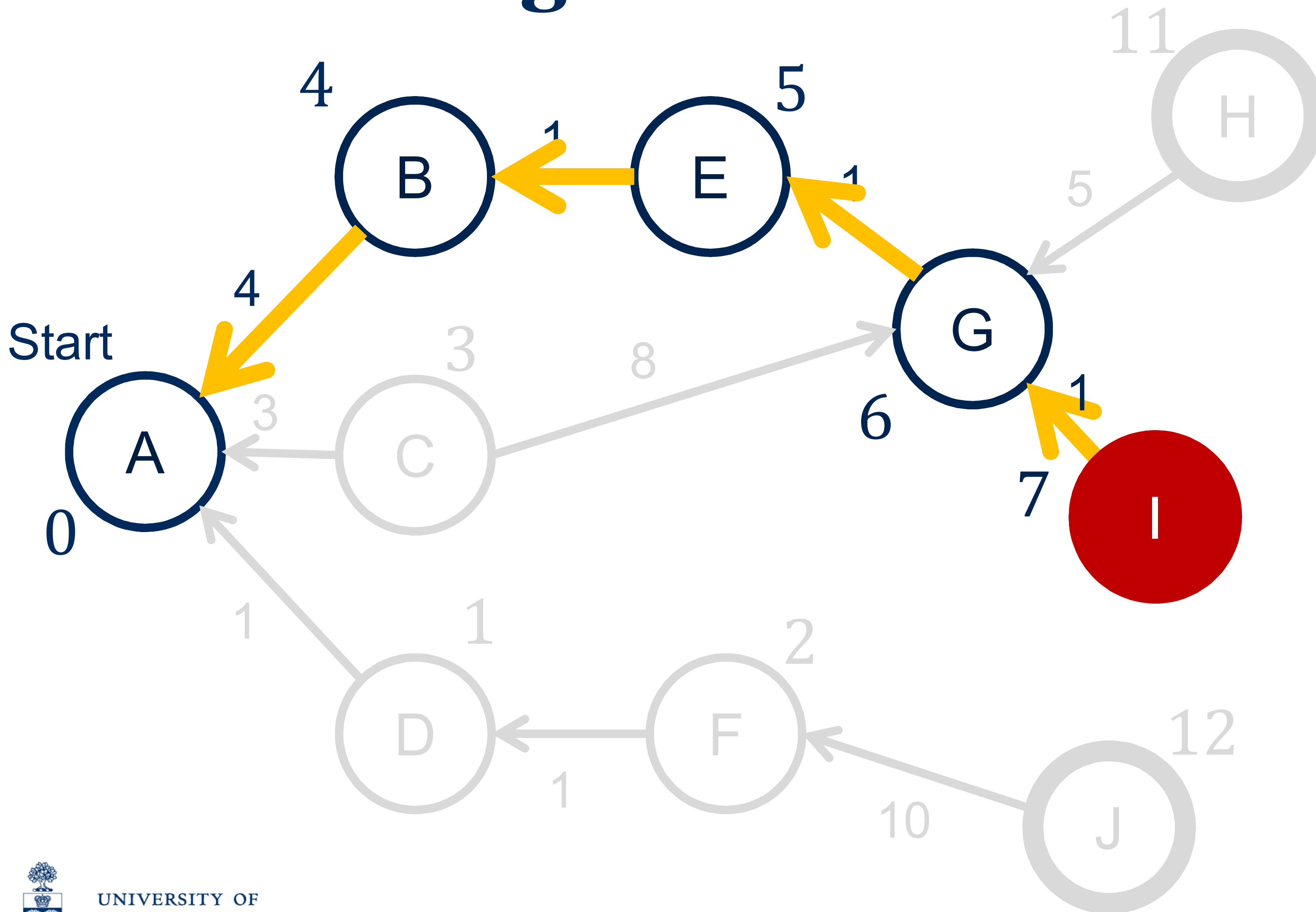


Extending Candidate Paths



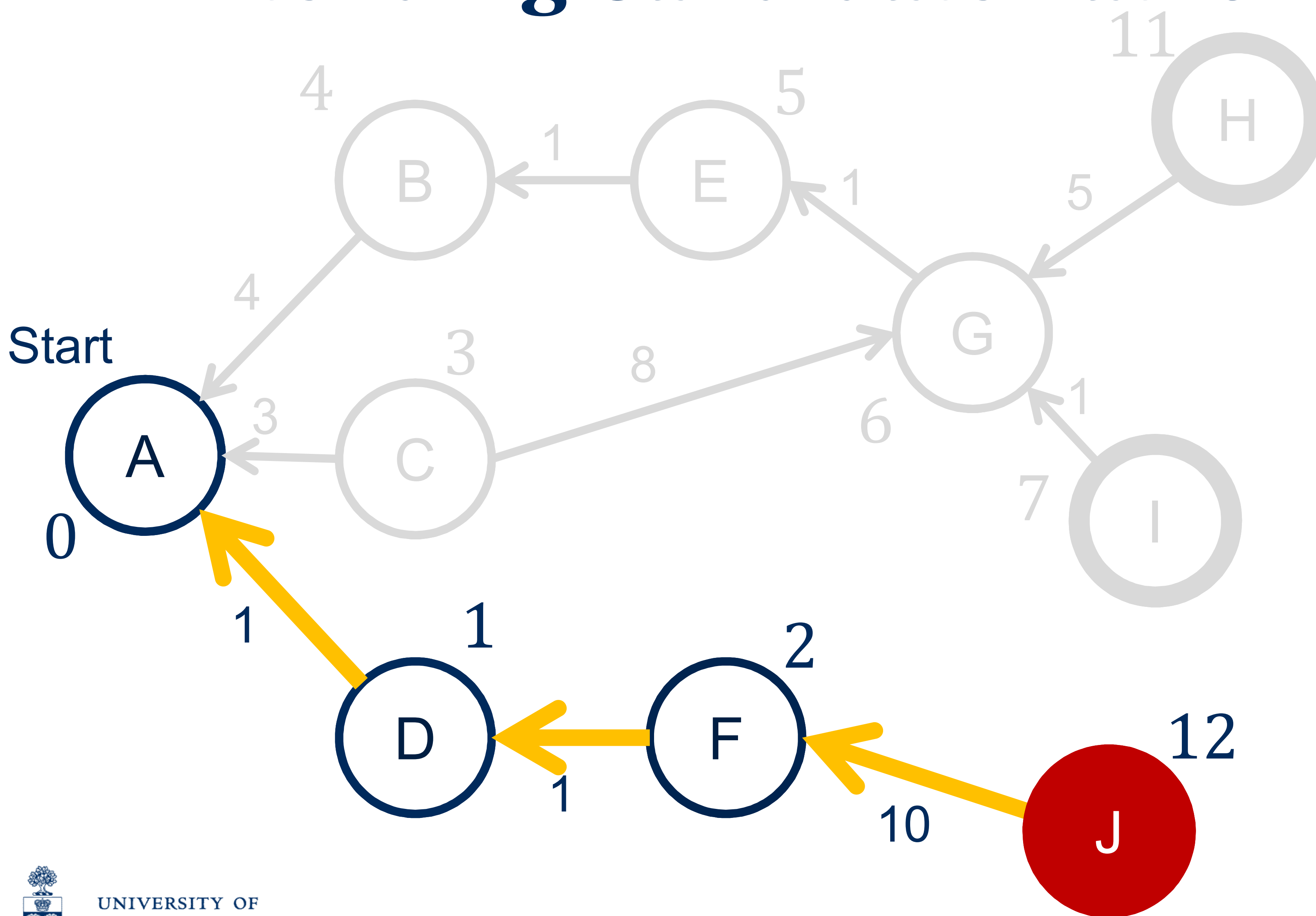
UNIVERSITY OF TORONTO

Extending Candidate Paths



UNIVERSITY OF
TORONTO

Extending Candidate Paths



Uniform Cost Search

- Iteratively extending some **candidate path**
- Uses the g-cost as the basis of this selection
  - Only info that uniform cost search has about a state
  - Only "uses" the transition function
- But each vertex represents a state
  - There is more information that can be used

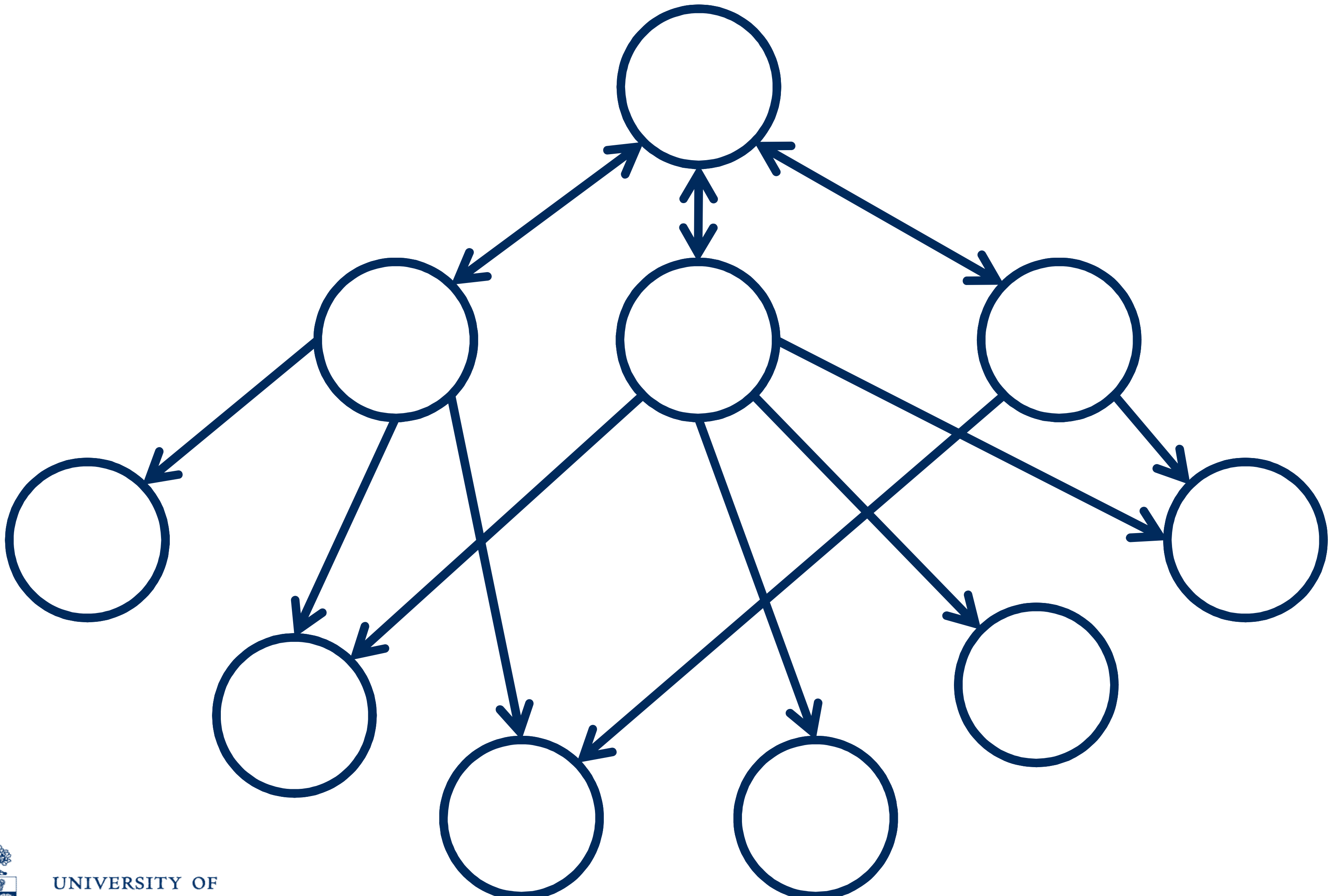
Uniform Cost Search

- Iteratively extending some **candidate path**

- Uses the g-cost as the basis of this selection
  – Only info that uniform cost search has about a state
  – Only "uses" the transition function

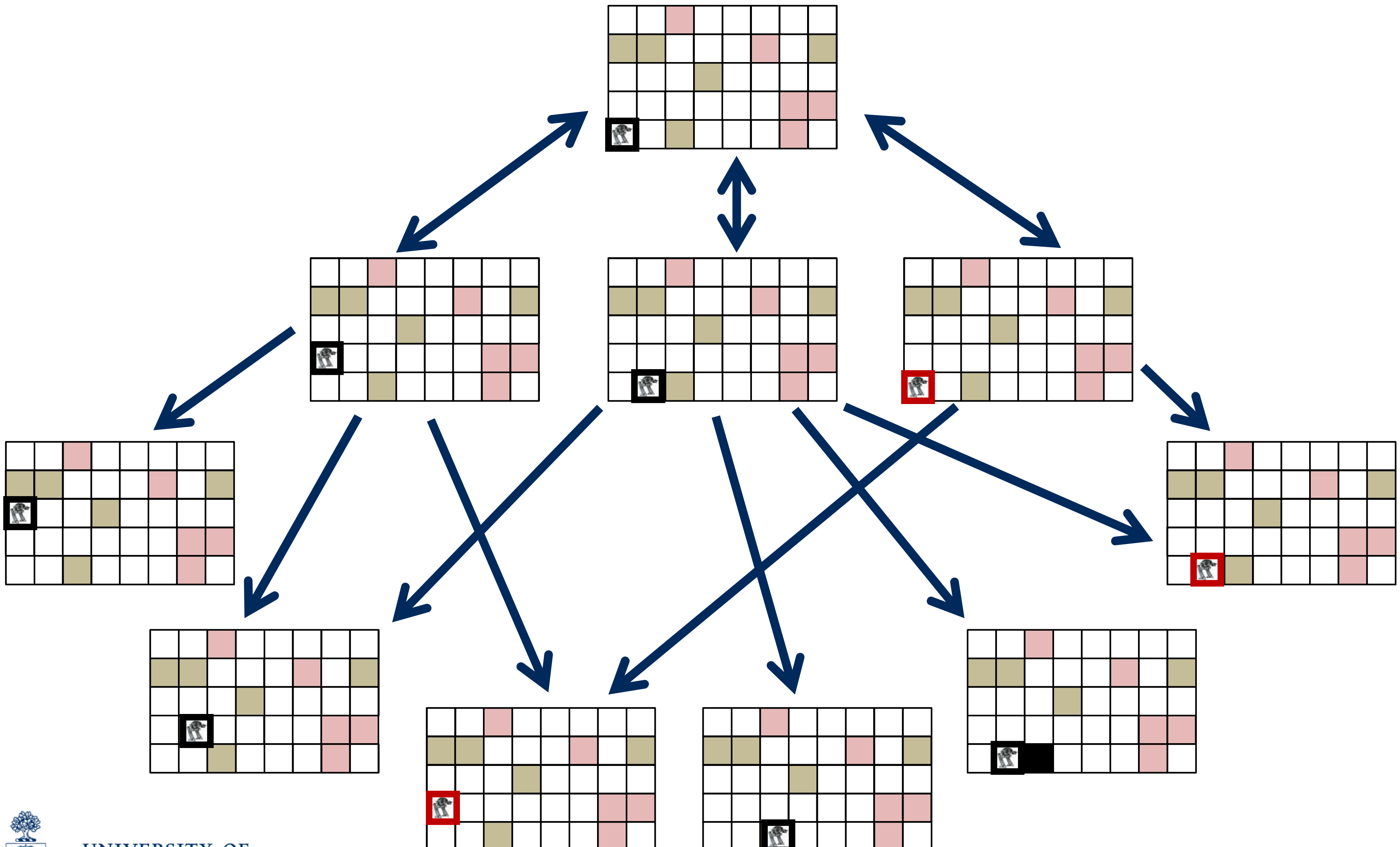
- But each vertex represents a state
  – There is more information that can be used

Graph Underlying Floortile



UNIVERSITY OF TORONTO

States Corresponding to Vertices



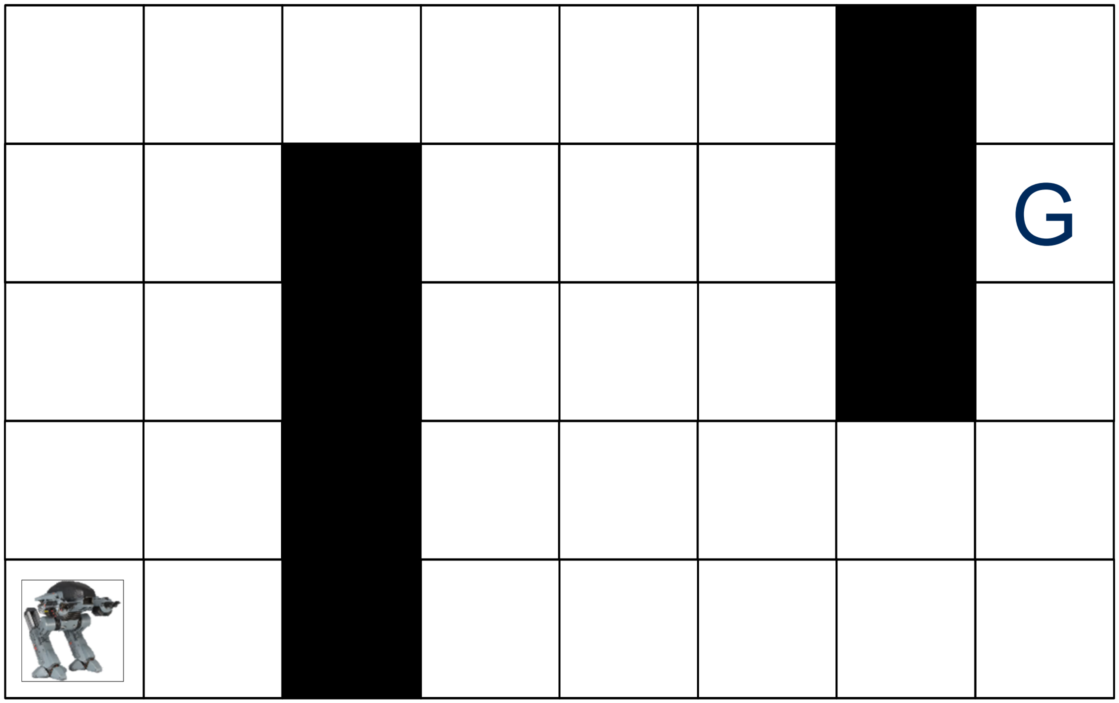
Heuristic Guidance

- A **heuristic function** *h* is a function from states* to the non-negative real values
  - Estimate the cost to reach the goal from the state
  - Other algorithms use such functions to change how they determine the order for extending candidate paths
- Often based on domain knowledge or domain simplification

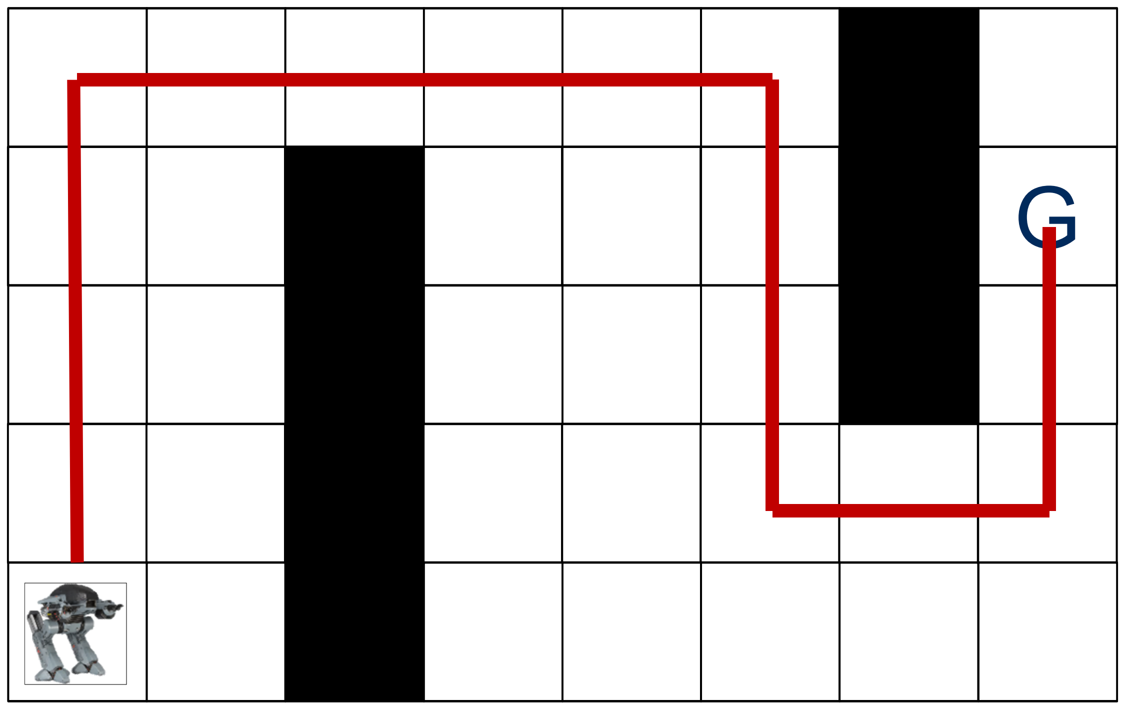
* Or sometimes candidate paths to real values

UNIVERSITY OF TORONTO

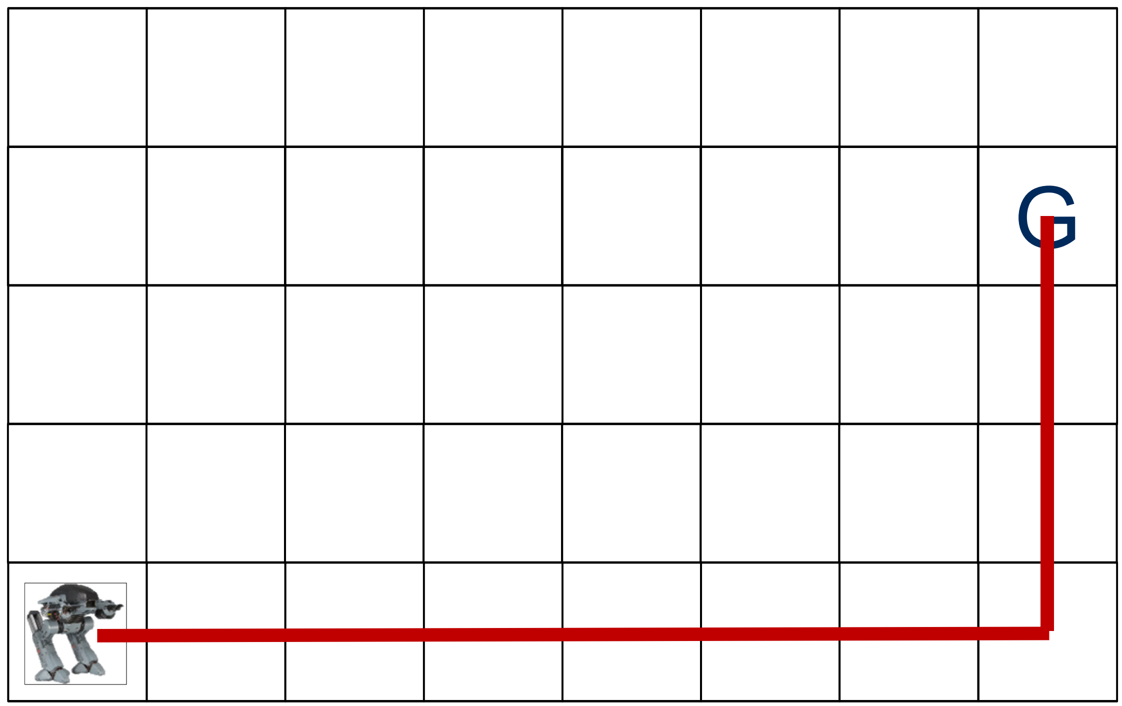
Pathfinding



Pathfinding

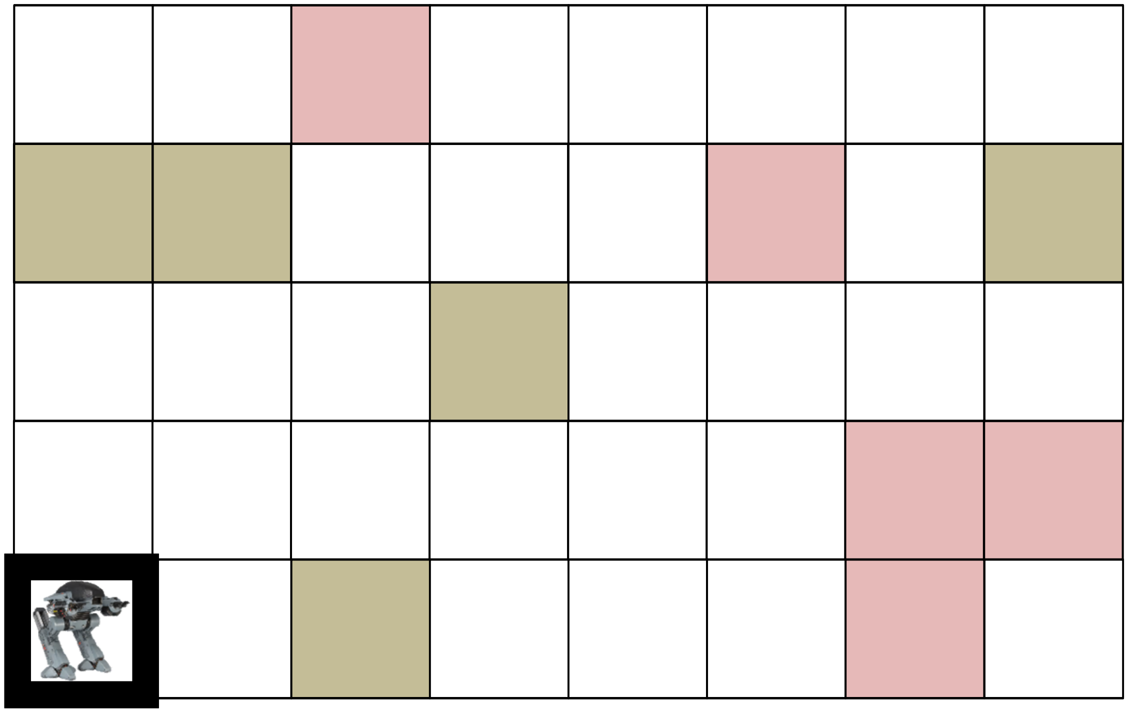


Pathfinding



UNIVERSITY OF TORONTO

Floortile from IPC 2011



- What are possible heuristics or simplications here?

Automatic Heuristic Generation

- Can use domain knowledge

- Many automatic heuristic generation techniques
  - Delete relaxation
  - Pattern databases
  - Landmark-based heuristics
  - Merge-and-Shrink
  - Counterexample guided abstraction refinement heuristics
  - …

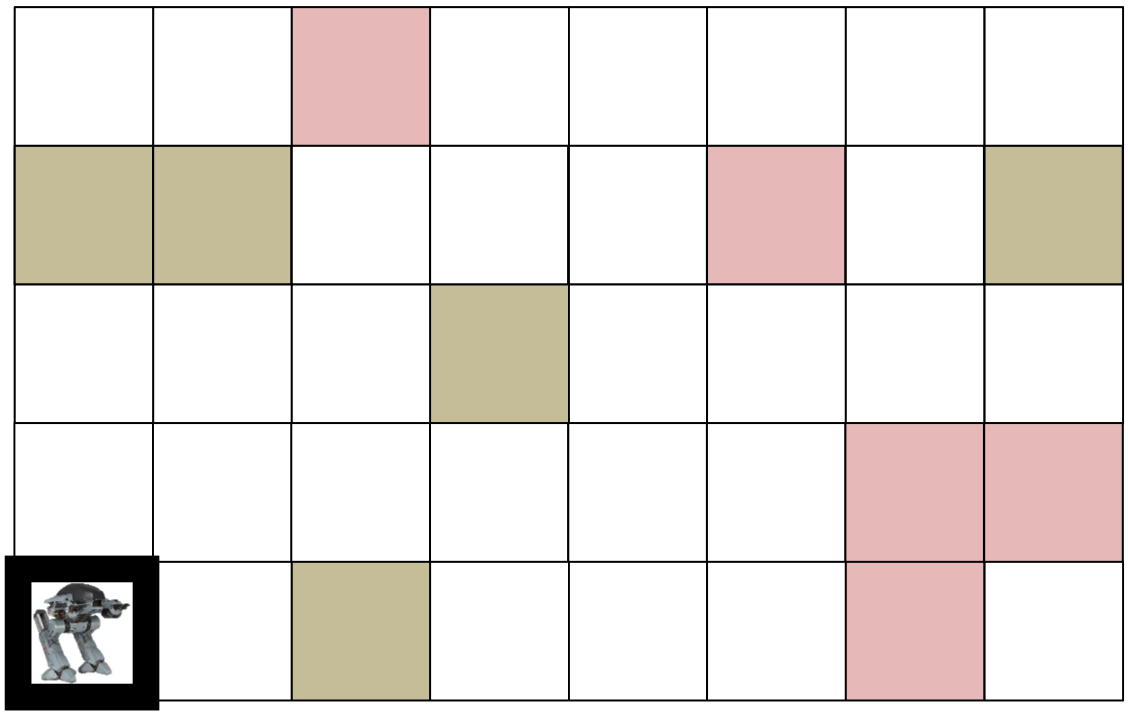
Automatic Heuristic Generation

- Can use domain knowledge

- Many automatic heuristic generation techniques
  - **Delete relaxation**
  - Pattern databases
  - Landmark-based heuristics
  - Merge-and-Shrink
  - Counterexample guided abstraction refinement heuristics
  - …

Delete Relaxation

- Can only achieve new facts, never delete them



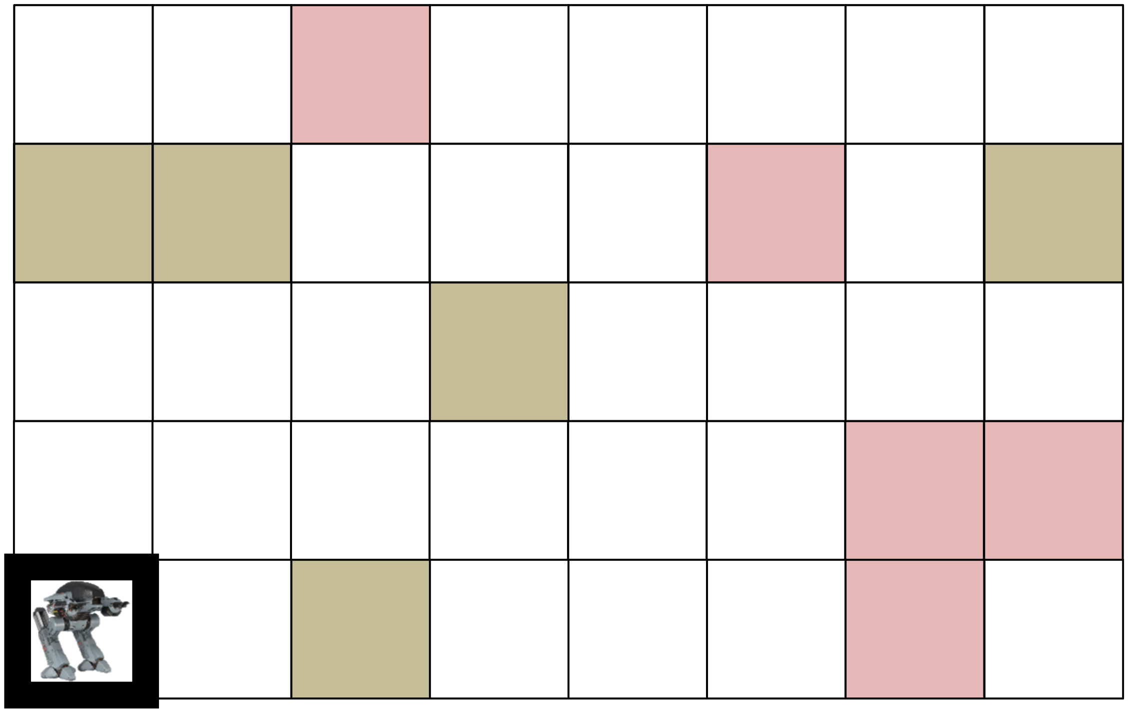
Move Action: MOVE-0-0-0-1

Pre: AT-0-0, WHITE-0-1

Post: AT-0-1, not(AT-0-0)

Delete Relaxation

- Can only achieve new facts, never delete them



Move Action: MOVE-0-0-0-1

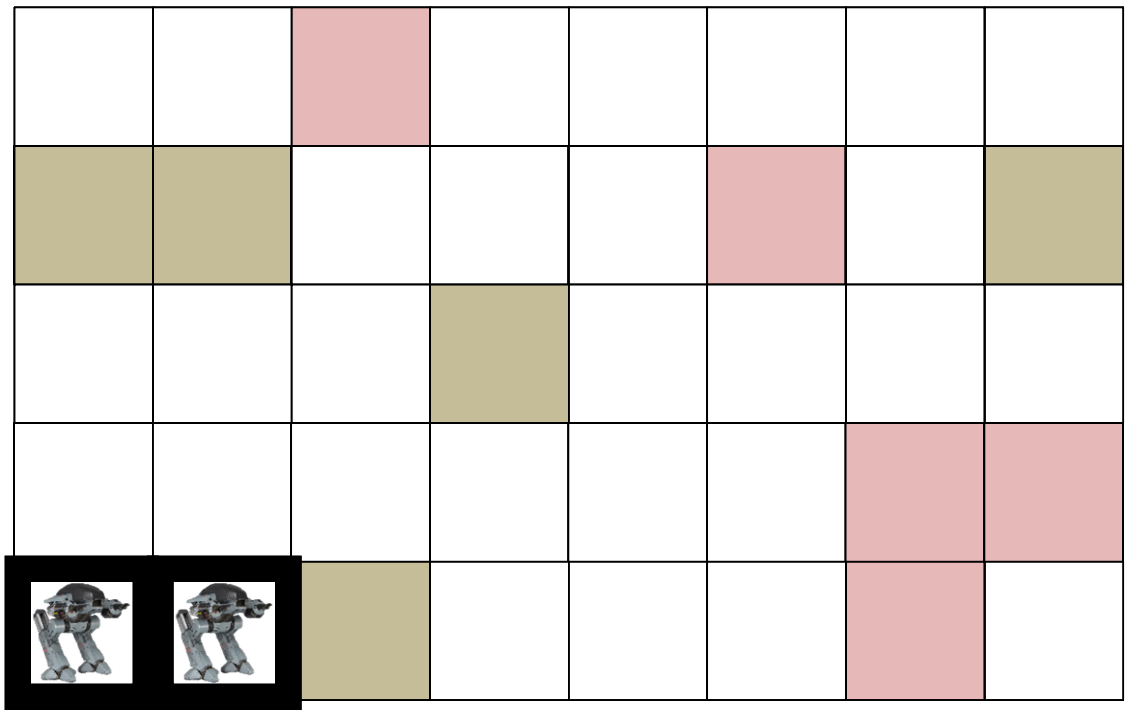
Pre: AT-0-0, WHITE-0-1

Post: AT-0-1, ~~not(AT-0-0)~~

UNIVERSITY OF TORONTO

Delete Relaxation

- Can only achieve new facts, never delete them



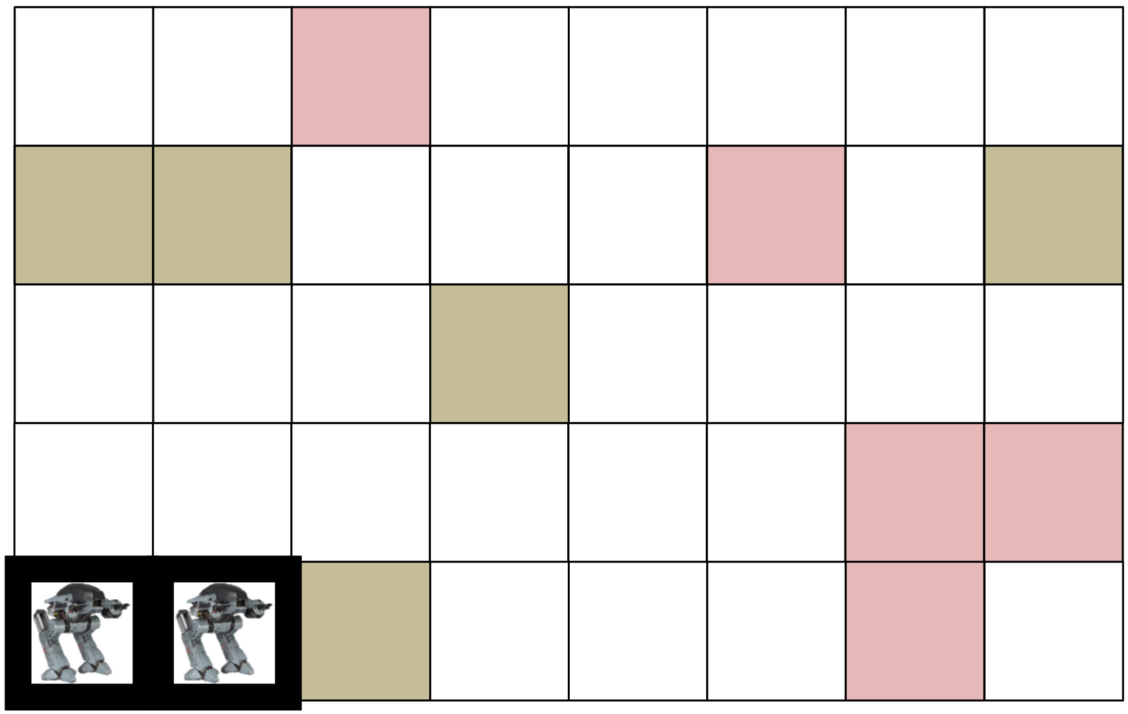
Move Action: MOVE-0-0-0-1

Pre: AT-0-0, WHITE-0-1

Post: AT-0-1, ~~not(AT-0-0)~~

Delete Relaxation

- Can only achieve new facts, never delete them



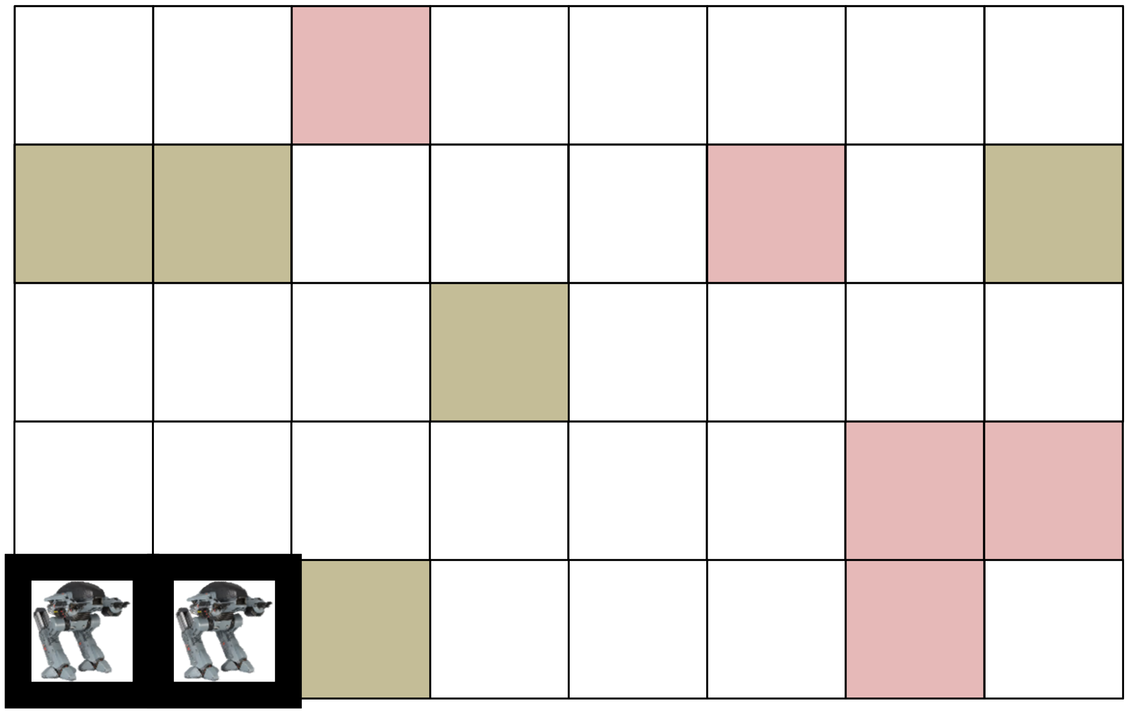
Paint Action: PAINT-B-0-1-0-2

Pre: AT-0-1, LOADED-B,
WHITE-0-2, NEED-B-0-2

Post: BLACK-0-2, not(WHITE-0-2)

Delete Relaxation

- Can only achieve new facts, never delete them



Paint Action: PAINT-B-0-1-0-2

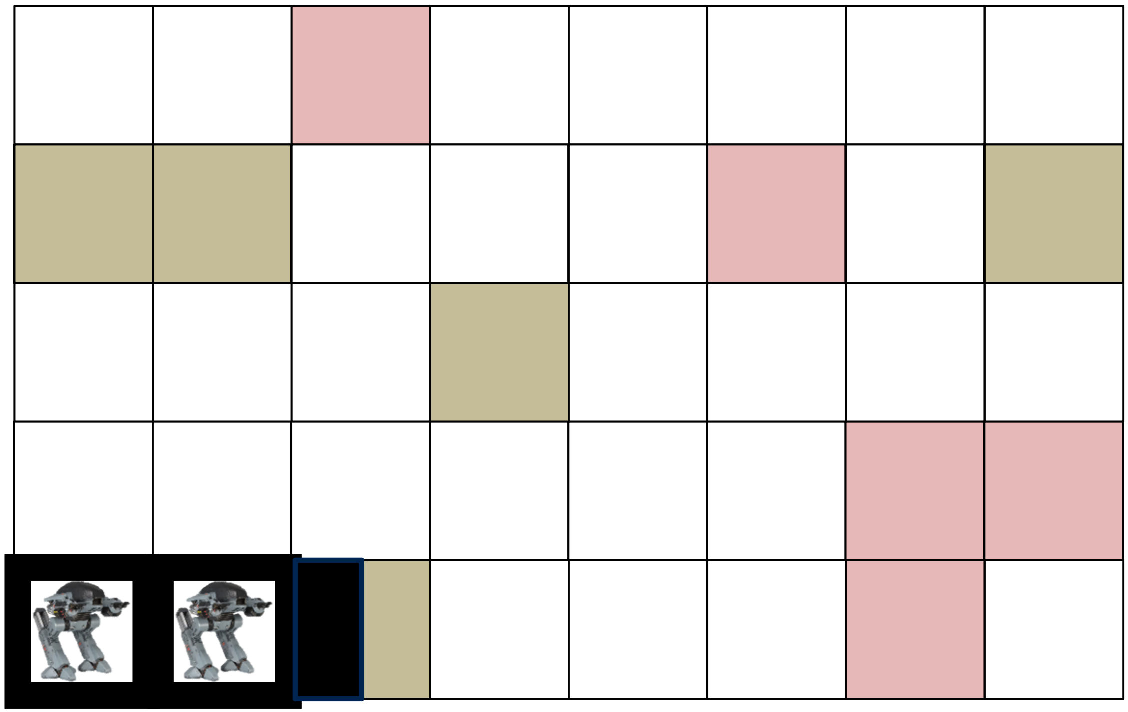
Pre: AT-0-1, LOADED-B,
WHITE-0-2, NEED-B-0-2

Post: BLACK-0-2, ~~not(WHITE-0-2)~~

UNIVERSITY OF TORONTO

Delete Relaxation

- Can only achieve new facts, never delete them



Paint Action: PAINT-B-0-1-0-2

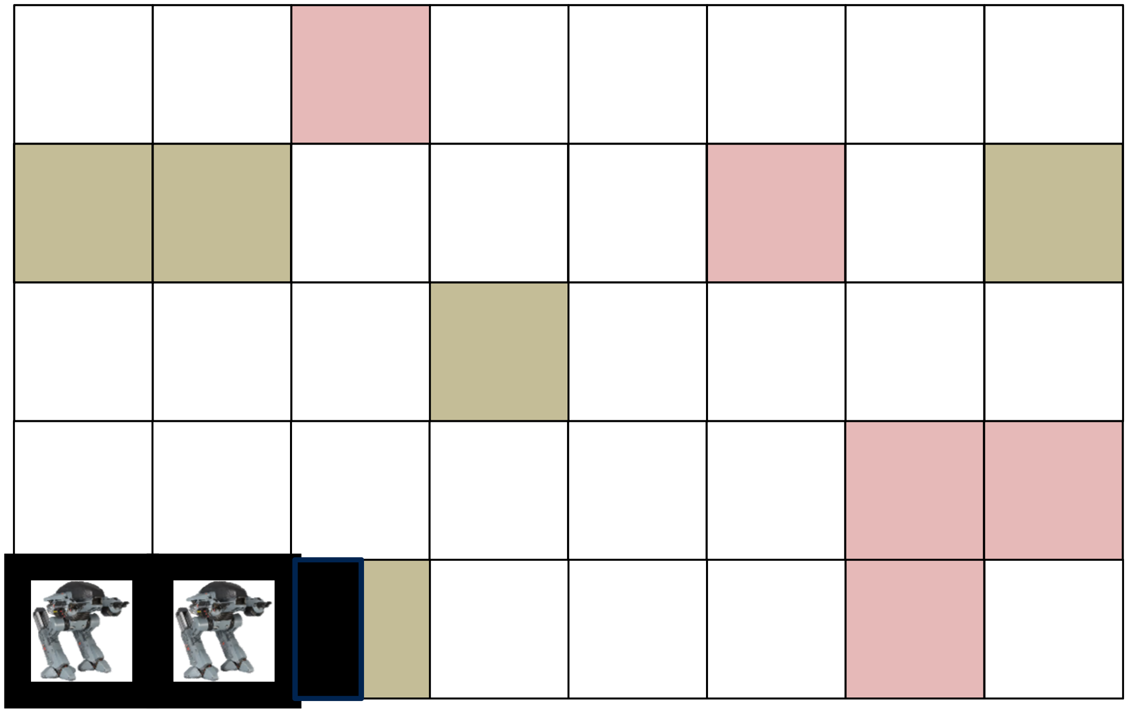
Pre: AT-0-1, LOADED-B,
WHITE-0-2, NEED-B-0-2

Post: BLACK-0-2, ~~not(WHITE-0-2)~~

UNIVERSITY OF TORONTO

Delete Relaxation

- Can only achieve new facts, never delete them



Load Action: LOAD-RED

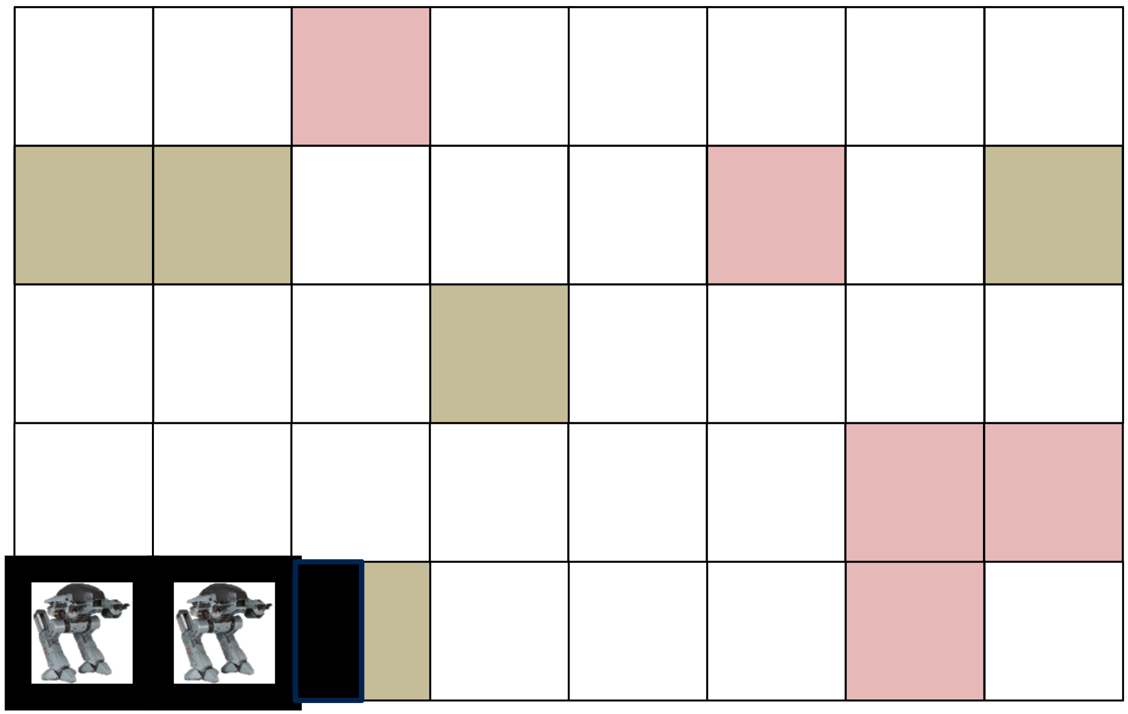
Pre: LOADED-B

Post: LOADED-R, not(LOADED-B)

UNIVERSITY OF TORONTO

Delete Relaxation

- Can only achieve new facts, never delete them



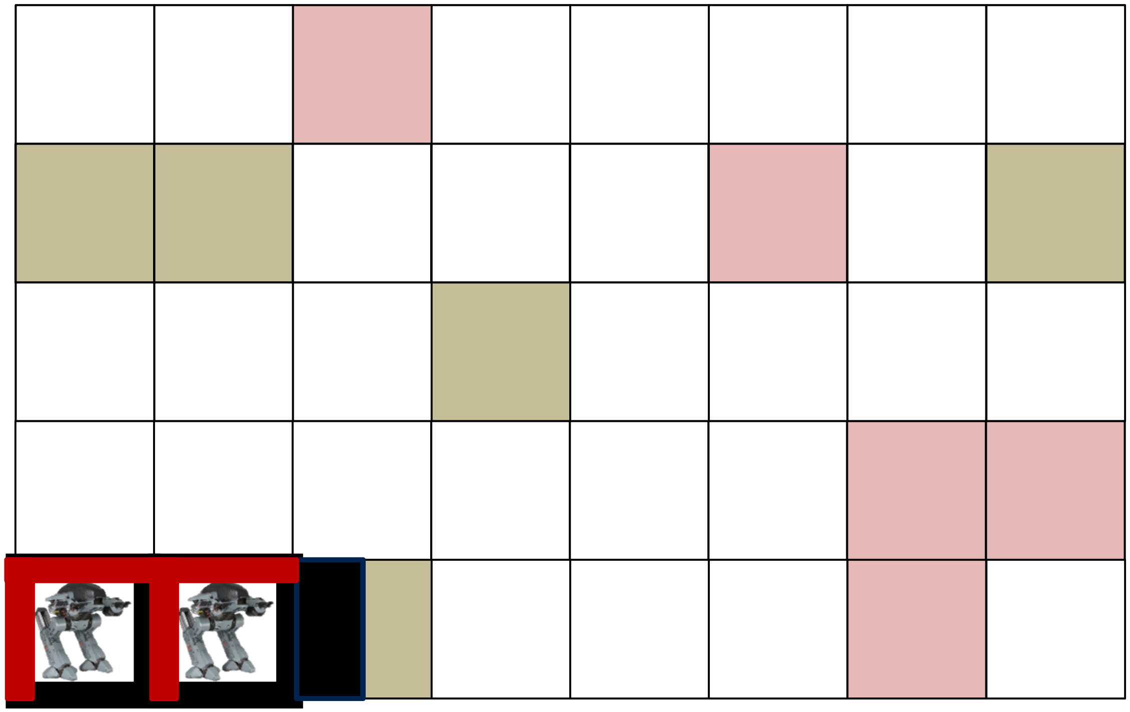
Load Action: LOAD-RED

Pre: LOADED-B

Post: LOADED-R, ~~not(LOADED-B)~~

Delete Relaxation

- Can only achieve new facts, never delete them



Load Action: LOAD-RED

Pre: LOADED-B

Post: LOADED-R, ~~not(LOADED-B)~~

UNIVERSITY OF TORONTO

Delete Relaxation

- Still NP-complete to optimally solve delete relaxed problems
  - Better than PSPACE-hard, but still …

- Do have polynomial ways to solve them suboptimally or come up with a lower bound

Summary

- Can solve planning using graph search
  - Generate graph and use Dijkstra's search
- Can incrementally generate the graph and stop early
  - Uniform cost search is this adjustment
- Uniform cost search is only using transition function
  - Ignoring state information
- Heuristic functions use state information to generate an estimate of the cost to a goal