Disclaimer: I use these notes as a guide rather than a comprehensive coverage of the topic. They are neither a substitute for attending the lectures nor for reading the assigned material.

“I may not have gone where I intended to go, but I think I have ended up where I needed to be.” – Douglas Adams

“All you need is the plan, the road map, and the courage to press on to your destination.” – Earl Nightingale

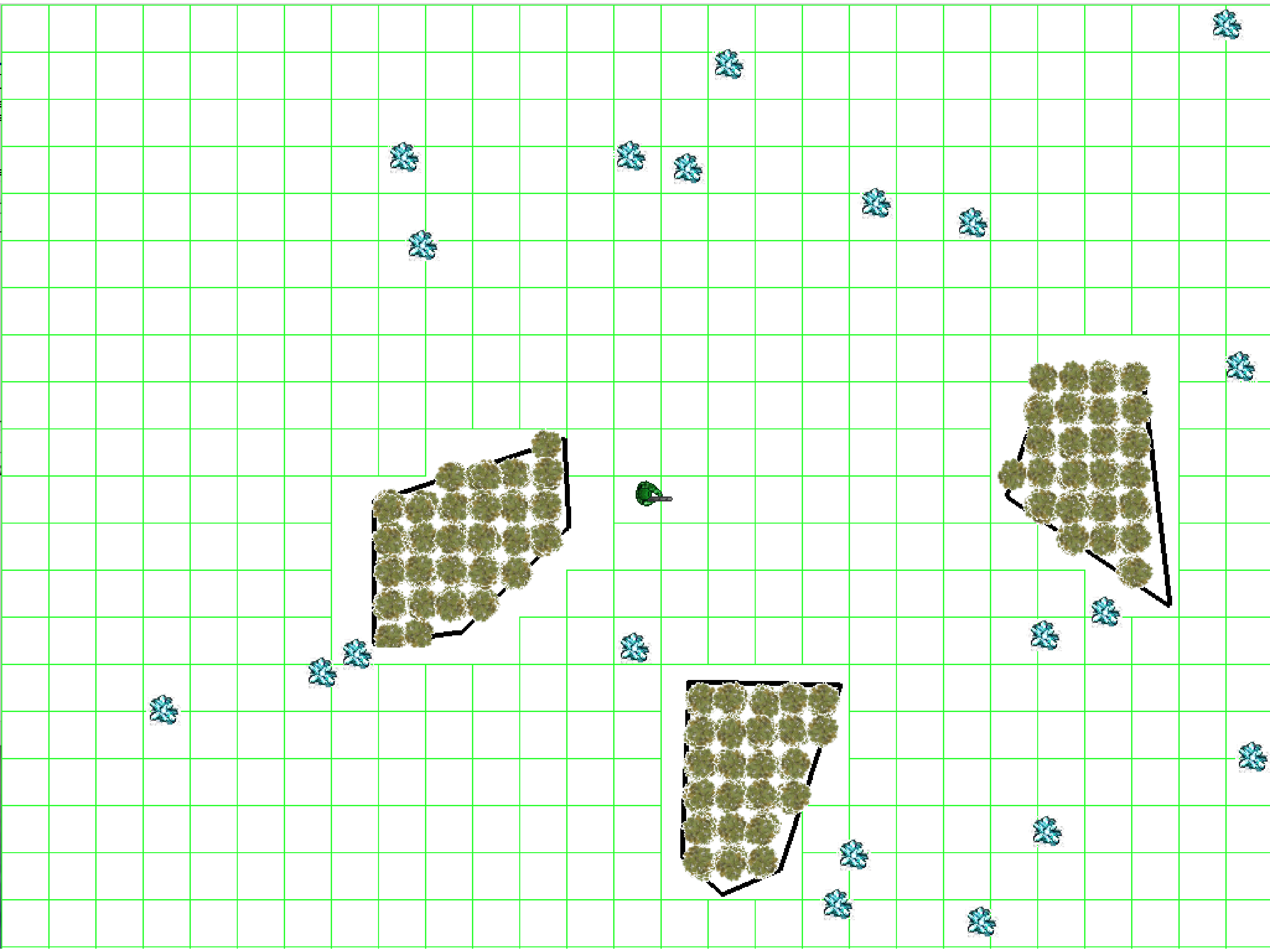
PREVIOUSLY ON…

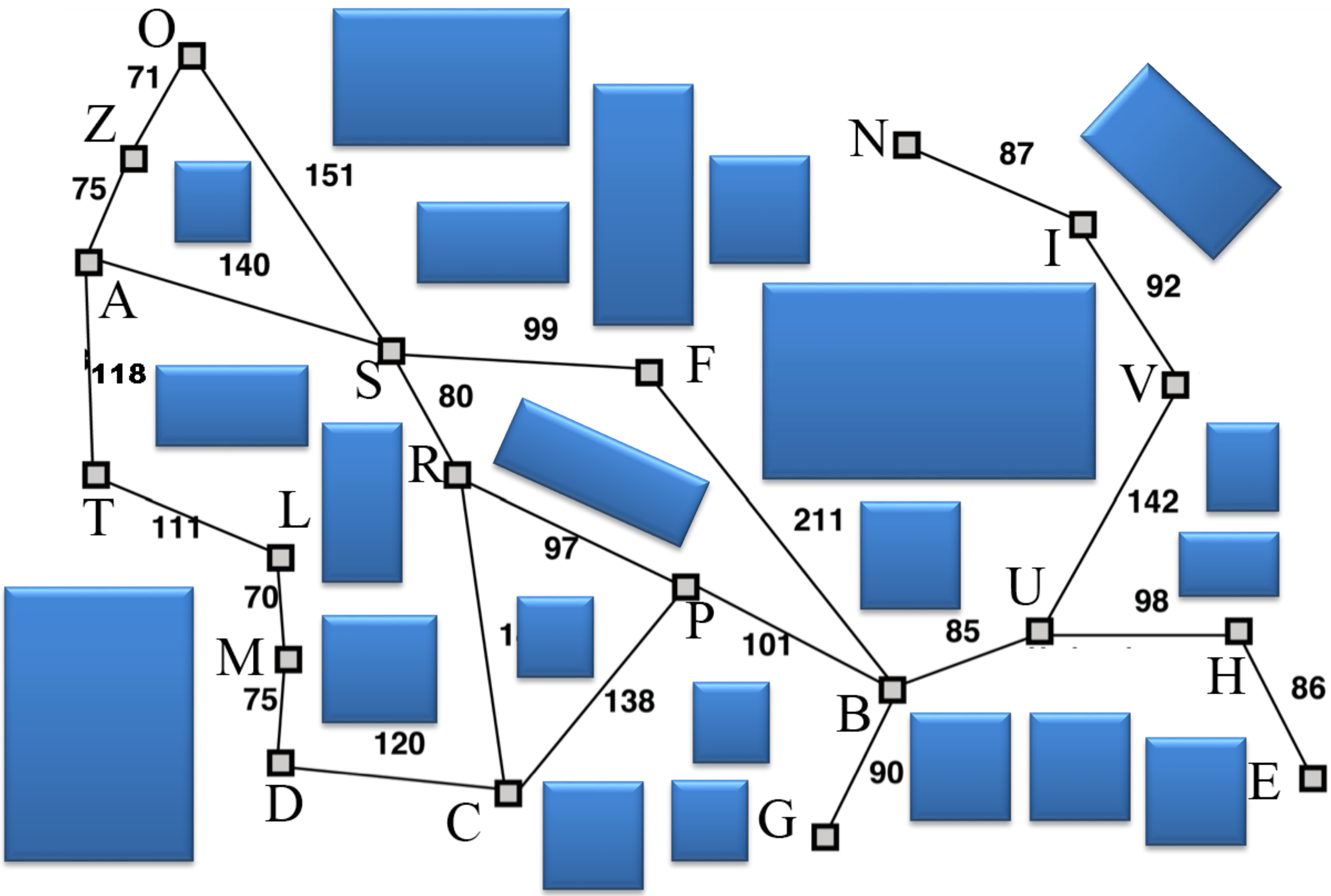
Class N-3

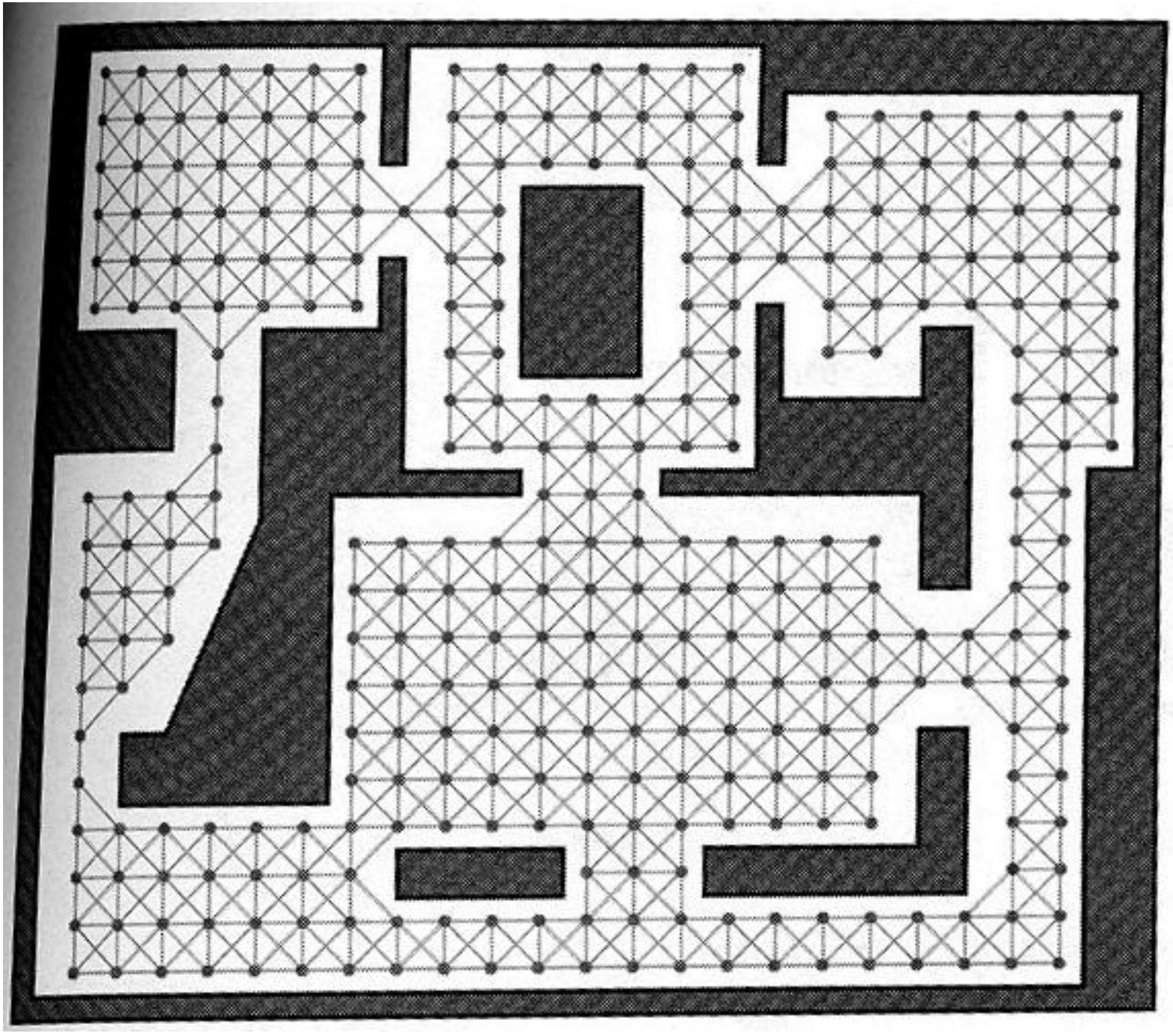
1. How would you describe AI (generally), to not us?
2. Game AI is really about
   - The I____ of I_____. Which is what?
   - Supporting the P_____ E______ which is all about… making the game more enjoyable
   - Doing all the things that a(nother) player or designer…
3. What are ways Game AI differs from Academic AI?
4. (academic) AI *in* games vs. AI *for* games. What's that?
5. What is the complexity fallacy?
6. The essence of a game is a g_ a set of r_, and a__?
7. What are three big components of game AI in-game?
8. What is a way game AI is used out-of-game?

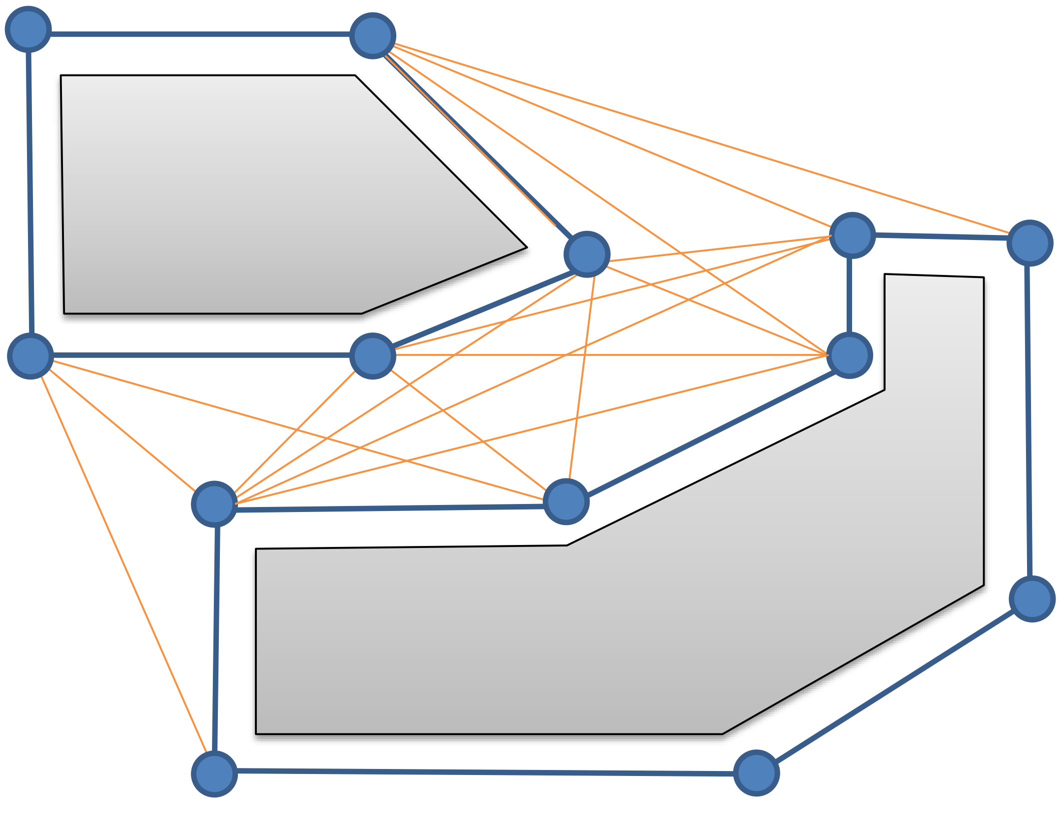
Class N-2

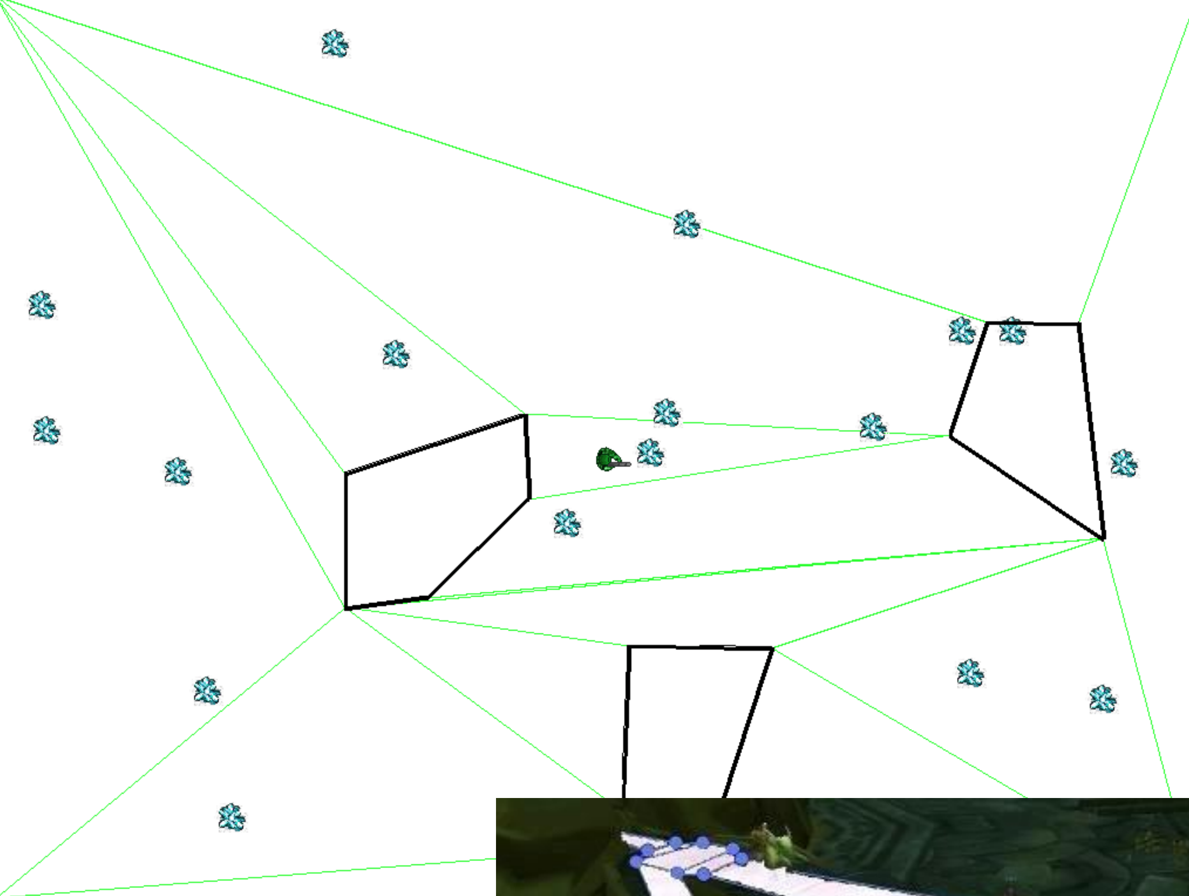
1. What is attack “kung fu” style?
2. How do intentional mistakes help games?
3. What defines a graph?
4. What defines graph search?
5. Name 3 uniformed graph search algorithms.
6. What is a heuristic?
7. Admissible heuristics never ___estimate
8. Examples of using graphs for games

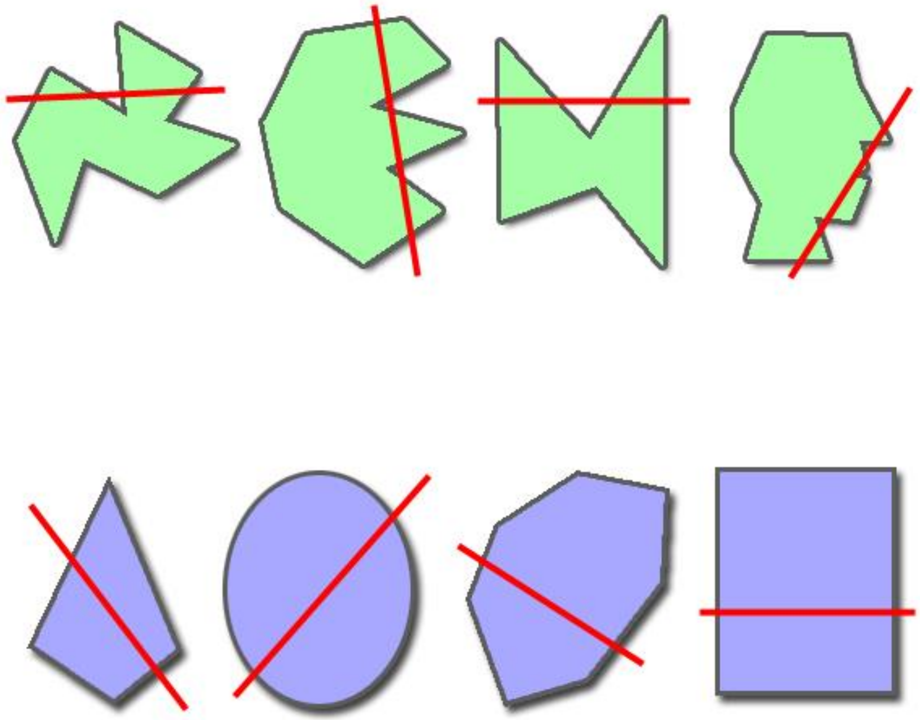


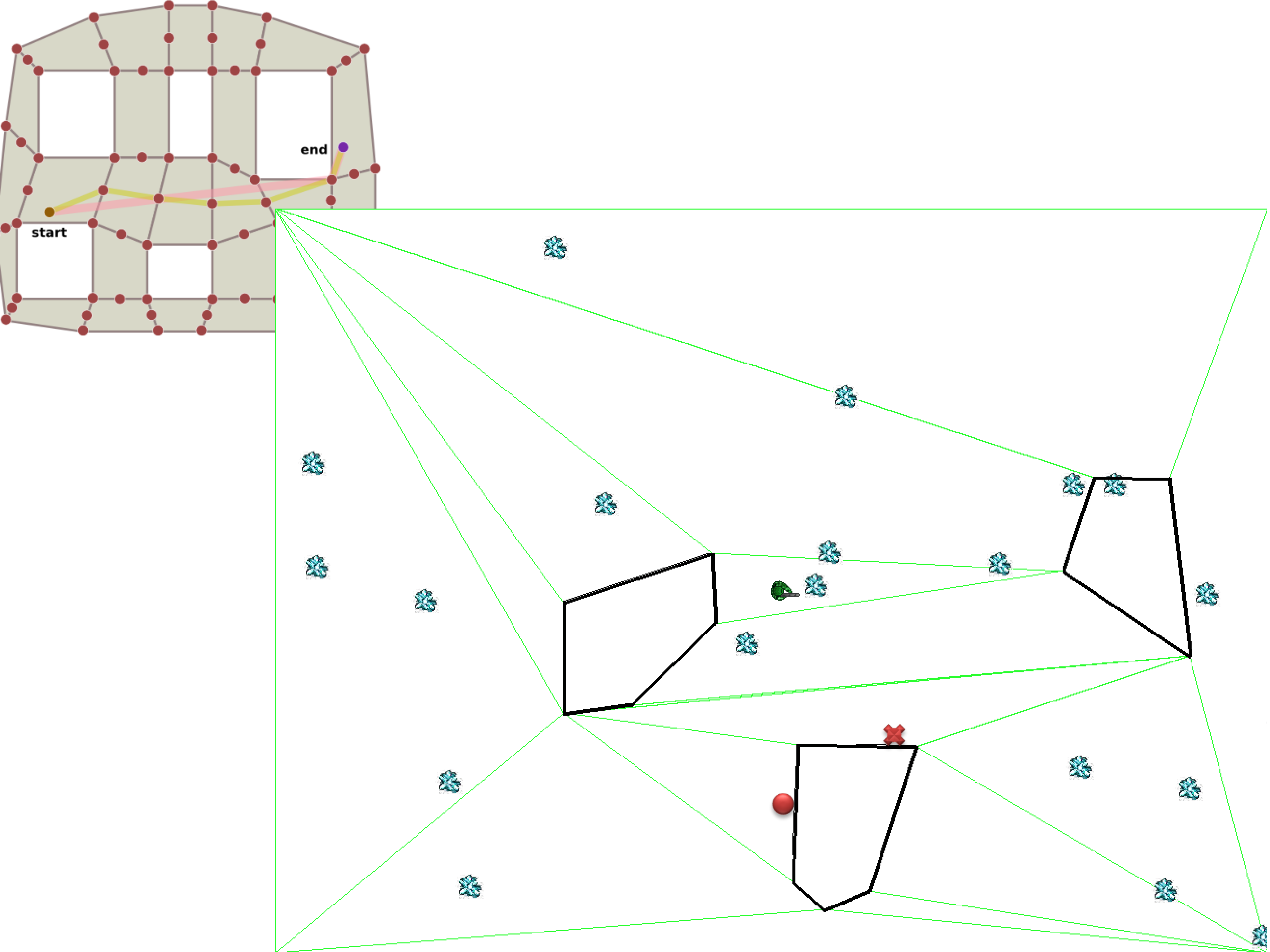

O
71
Z
75
151
140
A
118
S
99
F
80
R
T
111
L
97
P
70
M
75
120
D
C
138
101
B
90
G
211
85
U
98
H
86
E
N
87
I
92
V
142

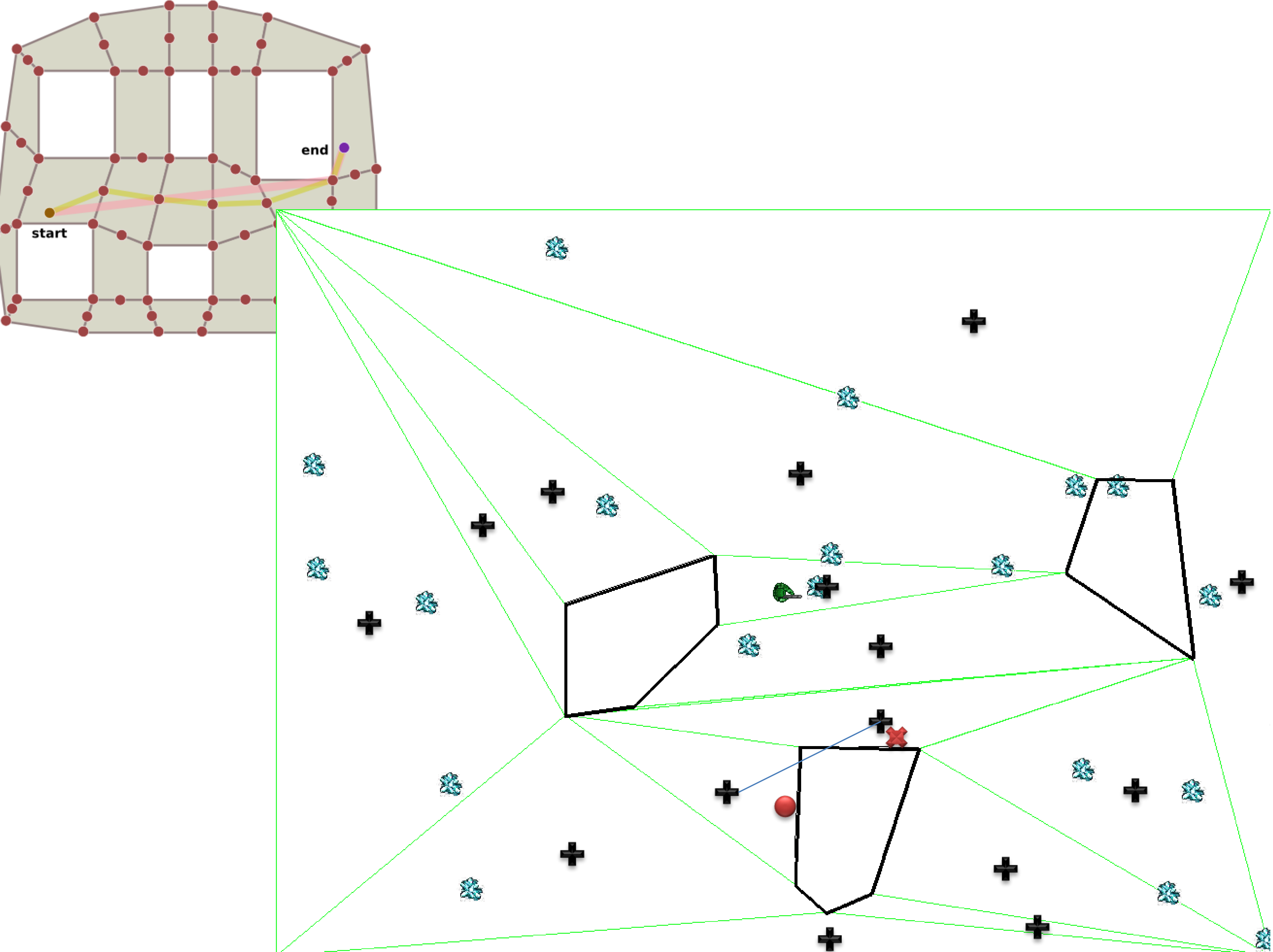


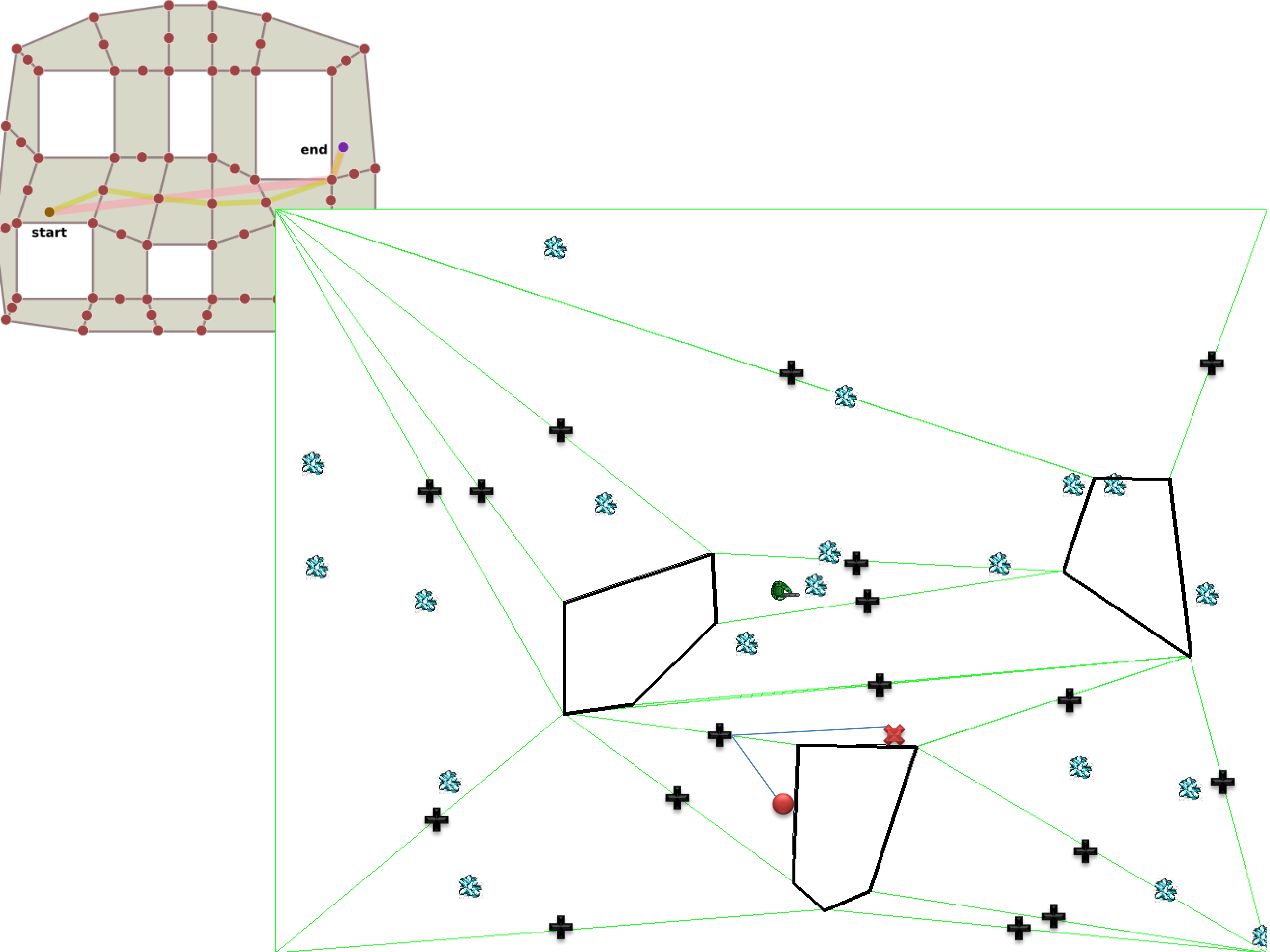


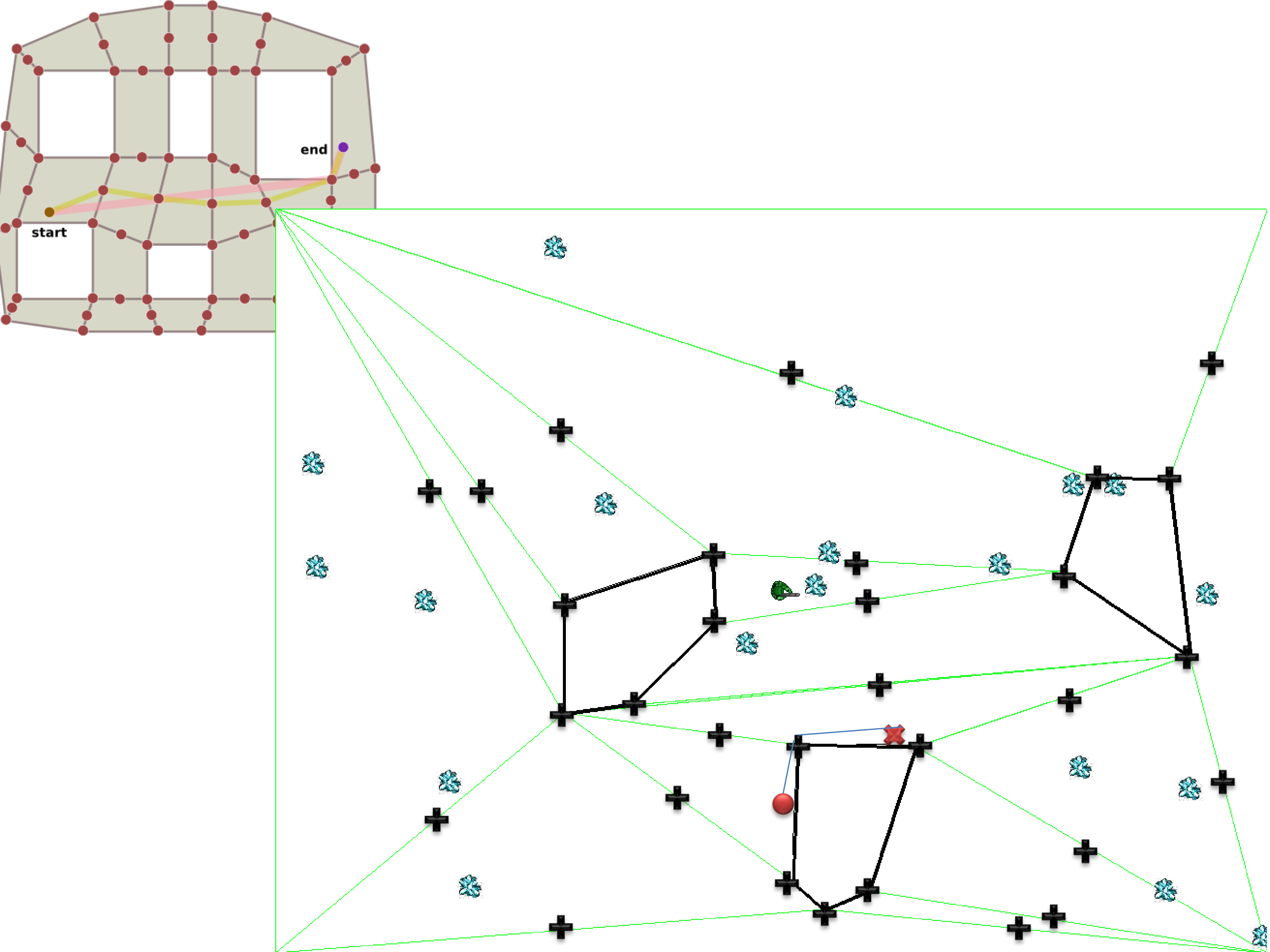


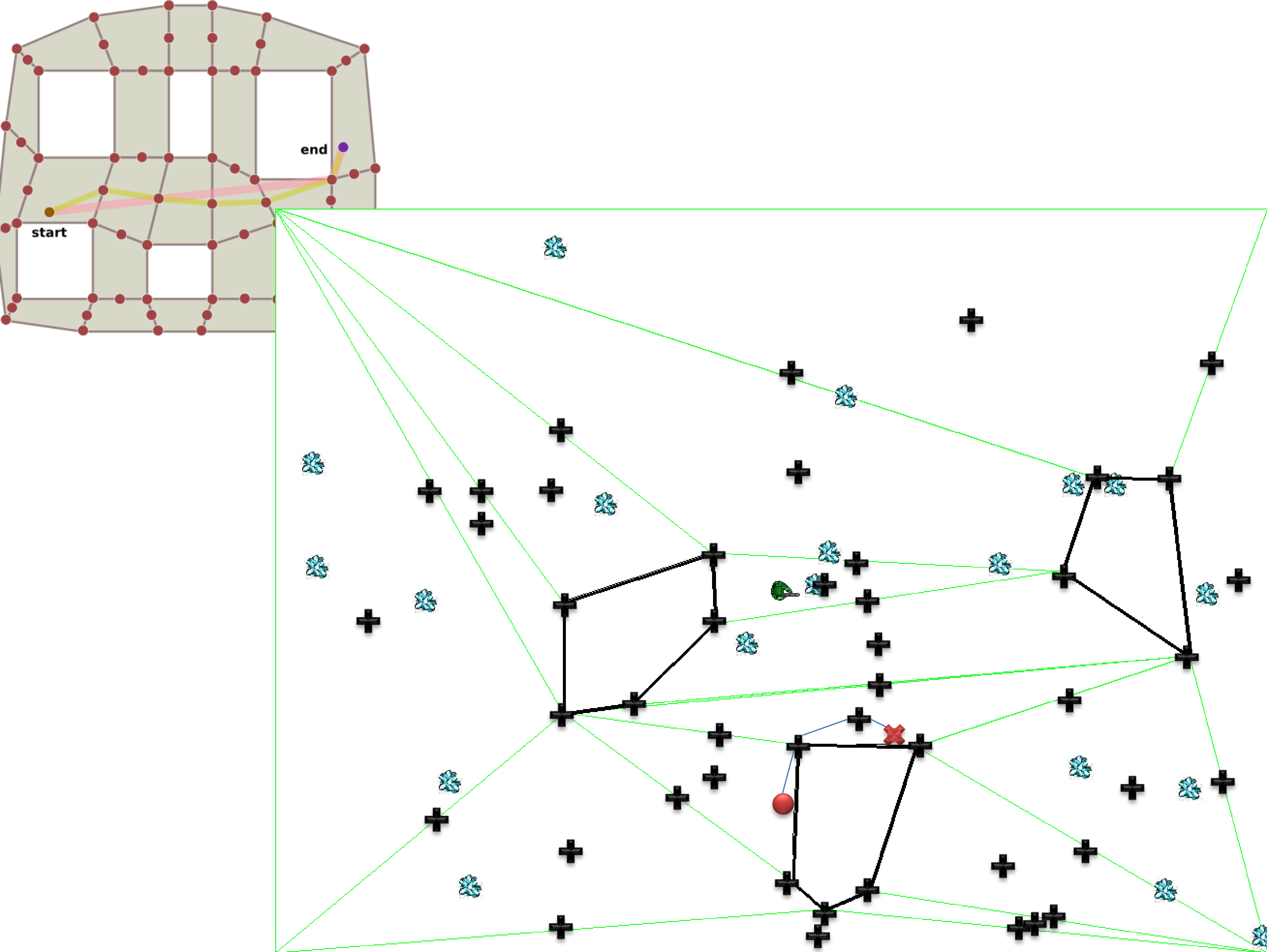


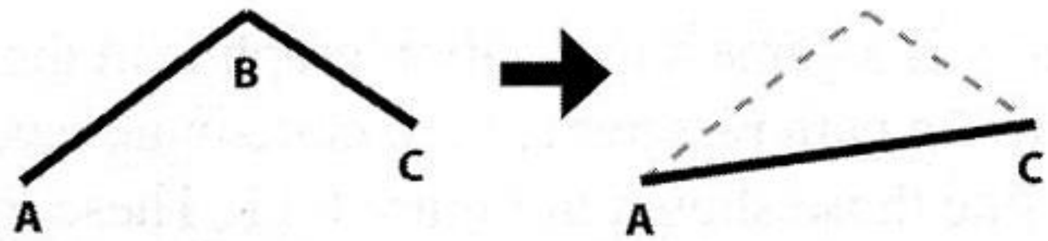

end
start


end
start

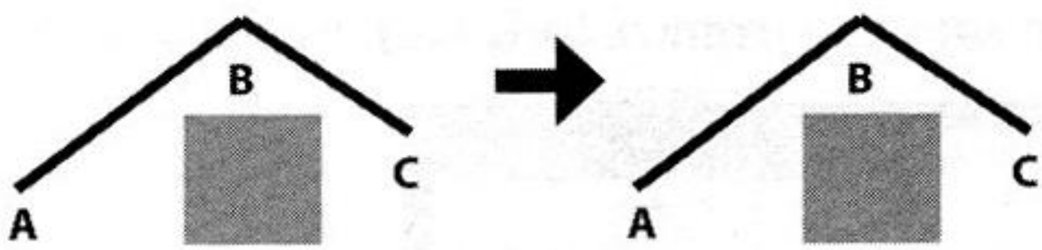

end
start


end
start


end
start



There is no obstacle obstructing the path from A to C so the two edges can be replaced with one.



With an obstacle in the way both edges are necessary

Is NavMesh good for all games?

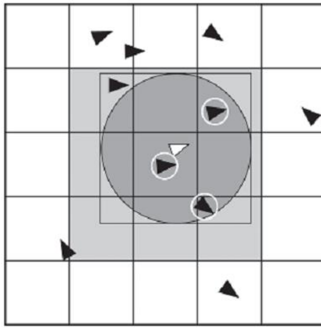
- Not necessarily
  - 2d Strategy game – grid gives fast random access
- Among the best for robust pathfinding and terrain reasoning in 3d worlds
  - Find the right solution for your problem

Class N-1

1. What are some benefits of path networks?
2. Cons of path networks?
3. What is the flood fill algorithm?
4. What is a simple approach to using path navigation nodes?
5. What is a navigation table?
6. How does the expanded geometry model work? Does it work with map gen features?
7. What are the major wins of a Nav Mesh?
8. Would you calculate an optimal nav-mesh?

findNearestWaypoint()

- Most engines provide a rapid "nearest" function for objects
- Spatial partitioning w/ special data structures:
  - Quad-trees (2d), oct-trees (3d), *k*-d trees
  - Binary space partitioning (BSP tree)
  - Multi-resolution maps (hierarchical grids)
- The gain over all-pairs techniques depends on number of agents/objects



Graphs, **Search, & Path Planning** Continued

2016-05-26

(and maybe kinematic motion; steering and flocking)

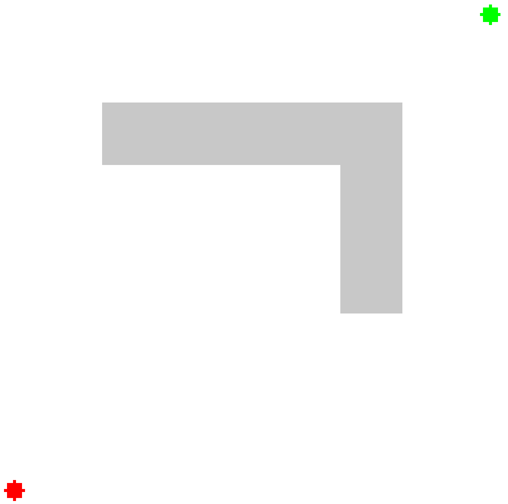
PATH NETWORK SEARCH

Precomputing Paths

- Why, When…
  – Faster than computation on the fly
  – Especially with large maps or lots of agents
- How…
  – Use Dijkstra's algorithm to create lookup tables
  – Lookup cost tables
  – Registering search requests
- What is the main problem with precomputed paths?

Dijkstra's algorithm

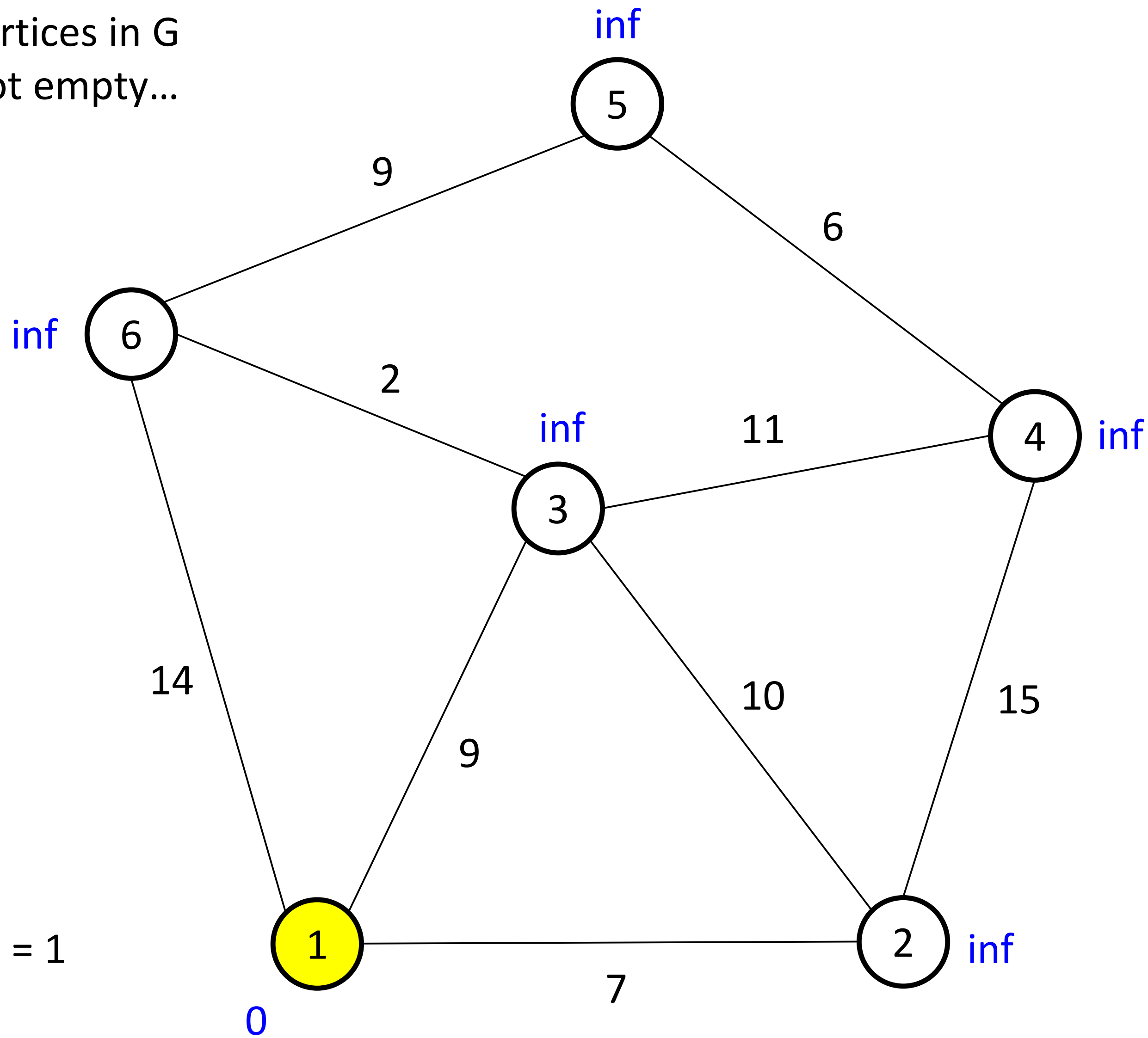
- A single-source, multi-target shortest path algorithm for arbitrary directed graphs with non-negative weights
- Tells you path from any one node to all other nodes



Given: G=(V,E), source

```
For each vertex v in G, set dist[v] to infinity
Set dist[source] = 0
Let Q = all vertices in G
While Q is not empty:
        Let u = get vertex in Q with smallest distance value
        Remove u from Q
        For each neighbor v of u:
                d = dist[u] + distance(u, v)
                if d < dist[v] then:
                        dist[v] = d
                        parent[v] = u
Return dist[]
```

For each vertex v in G, set dist[v] to infinity
Set dist[source] = 0
Let Q = all vertices in G
While Q is not empty…



* Source = 1

x	dist[x]	par[x]
1	0	
2	inf	
3	inf	
4	inf	
5	inf	
6	inf	

Q=[1,2,3,4,5,6]
U=1

Let u = get vertex in Q with smallest distance value (node 1)

dist[v]

Remove u (node 1) from Q

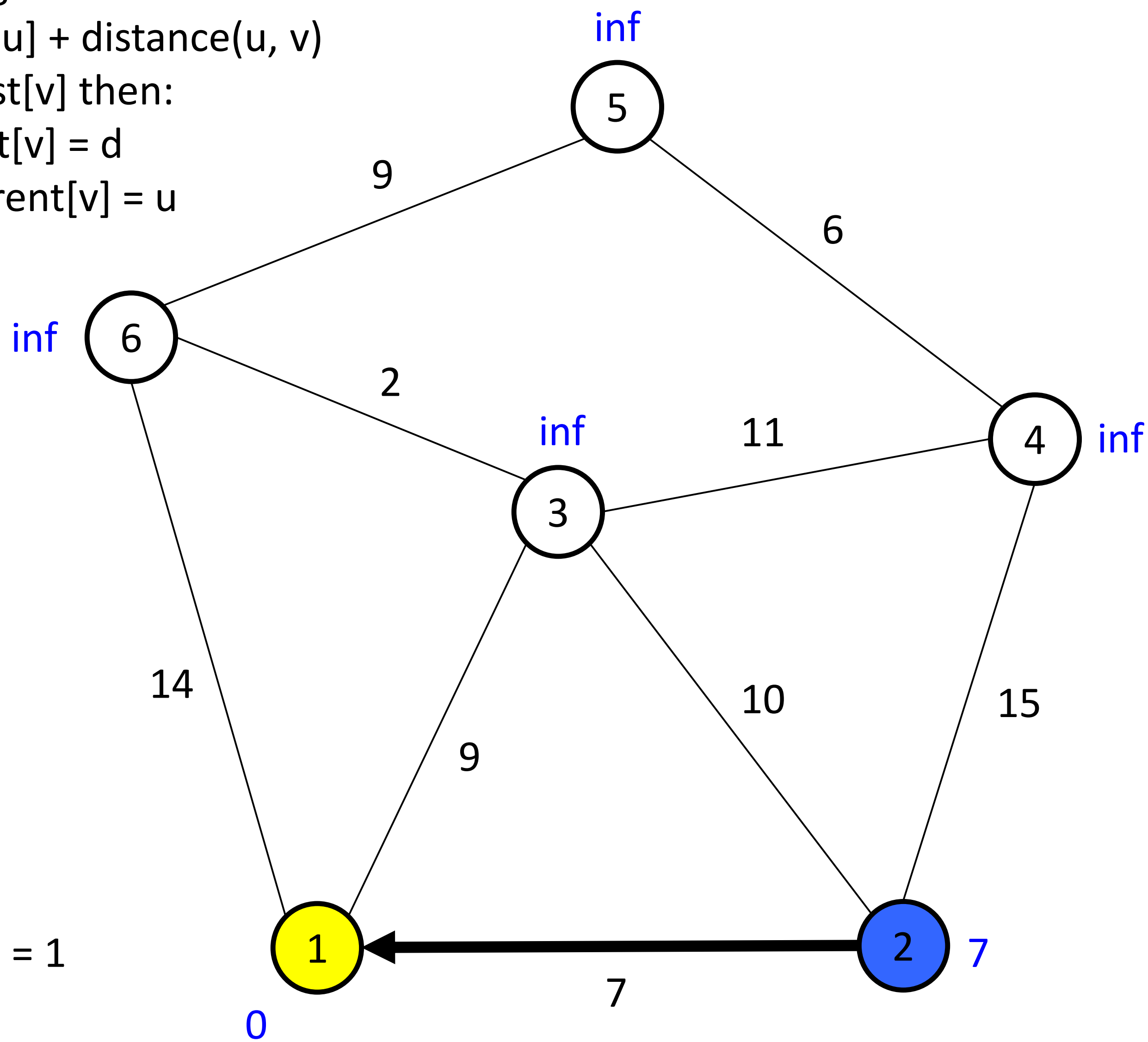
For each neighbor v of u:

    d = dist[u] + distance(u, v)

    if d < dist[v] then:

        dist[v] = d

        parent[v] = u



x	dist[x]	par[x]
1	0	
2	7	1
3	inf	
4	inf	
5	inf	
6	inf	

Q=[~~1,~~2,3,4,5,6]

U=1

V=2

* Source = 1

parent[v] dist[v]

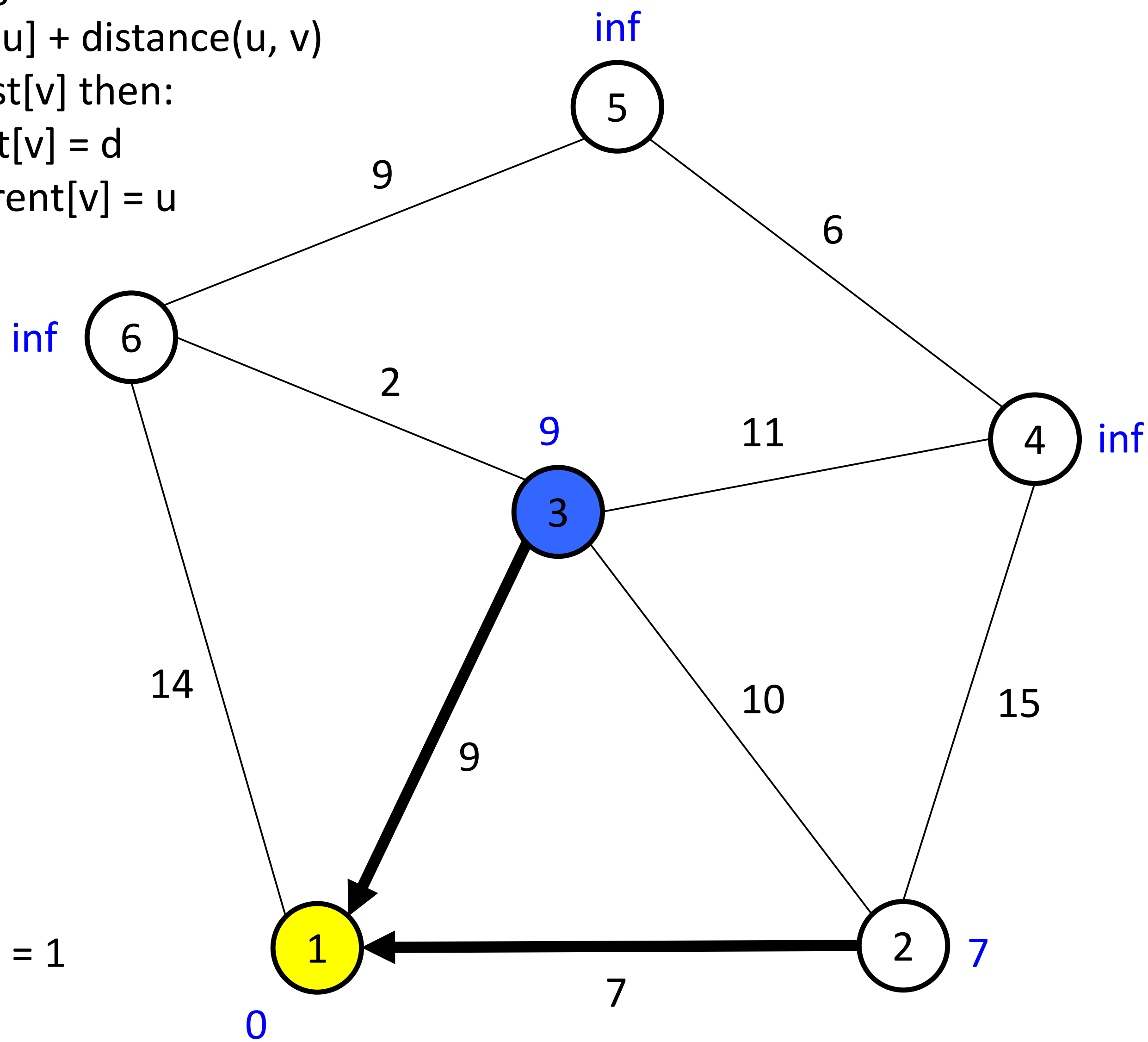
```
Remove u (node 1) from Q
For each neighbor v of u:
        d = dist[u] + distance(u, v)
        if d < dist[v] then:
                dist[v] = d
                parent[v] = u
```

x	dist[x]	par[x]
1	0	
2	7	1
3	9	1
4	inf	
5	inf	
6	inf	

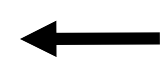
Q=[~~1,~~2,3,4,5,6]

U=1

V=3



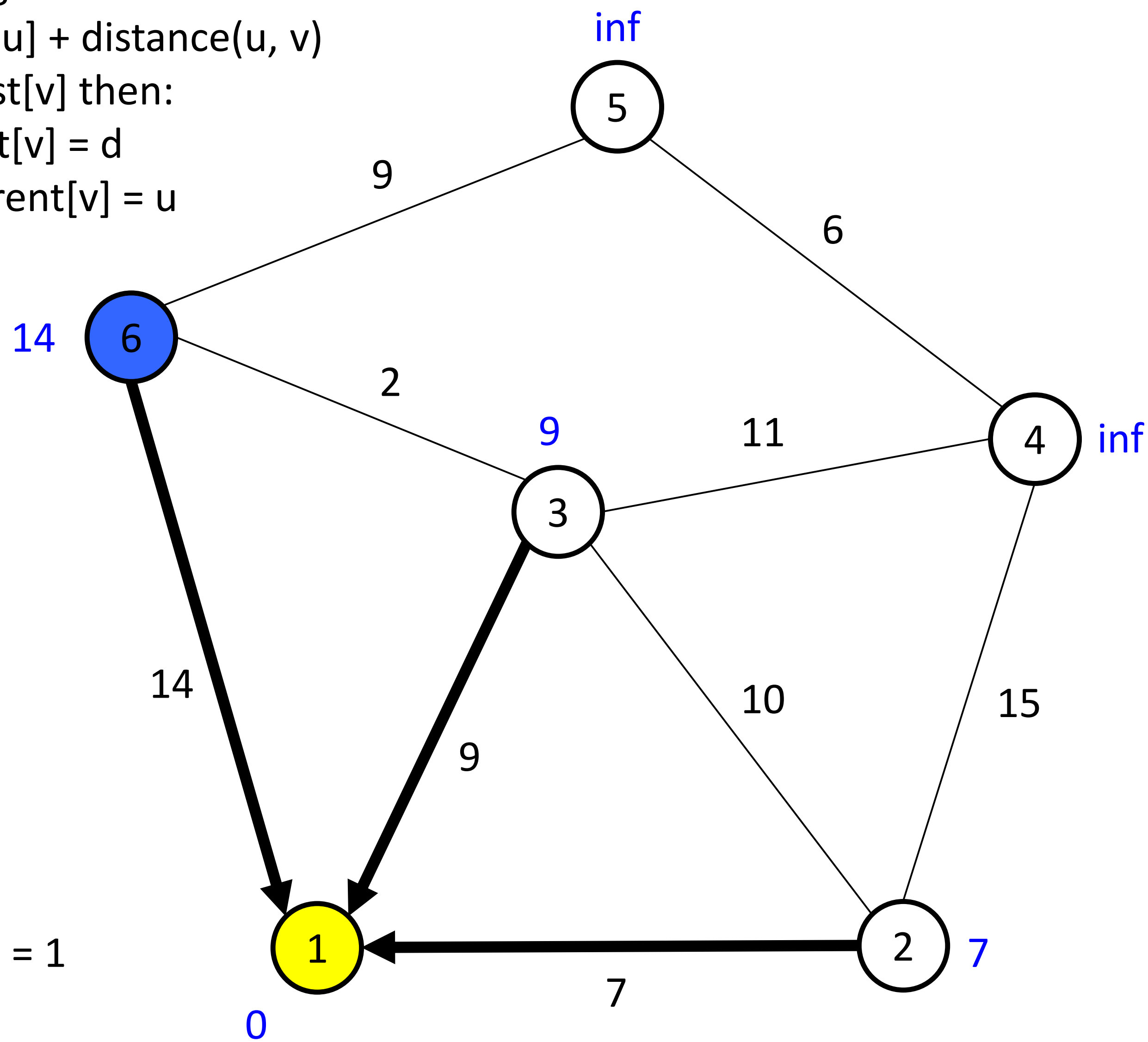
* Source = 1



parent[v] dist[v]

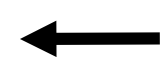
Remove u (node 1) from Q
For each neighbor v of u:
    d = dist[u] + distance(u, v)
    if d < dist[v] then:
        dist[v] = d
        parent[v] = u

x	dist[x]	par[x]
1	0	
2	7	1
3	9	1
4	inf	
5	inf	
6	14	1

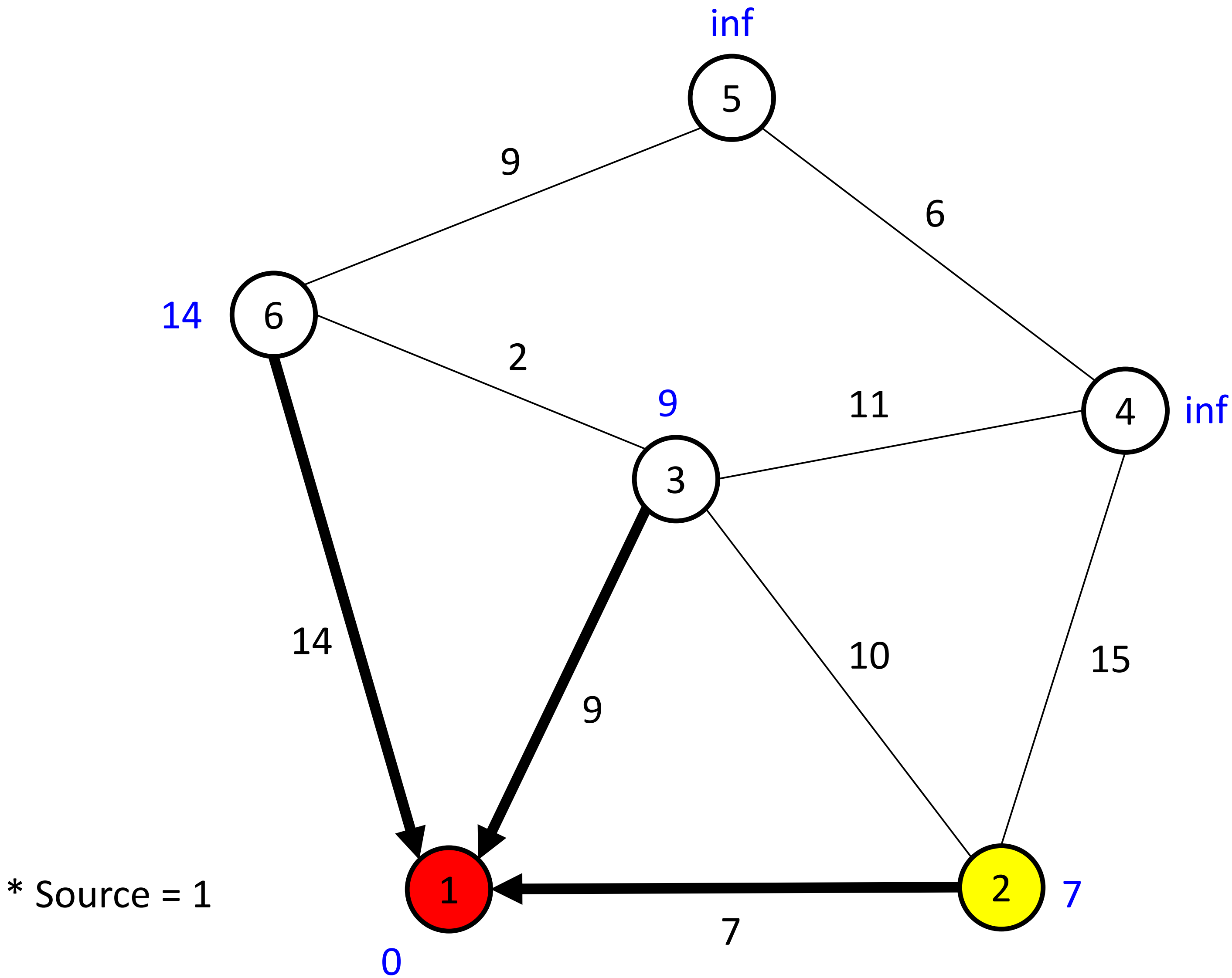


Q=[~~1,~~2,3,4,5,6]
U=1
V=6

* Source = 1



parent[v] dist[v]



x	dist[x]	par[x]
1	0	
2	7	1
3	9	1
4	inf	
5	inf	
6	14	1

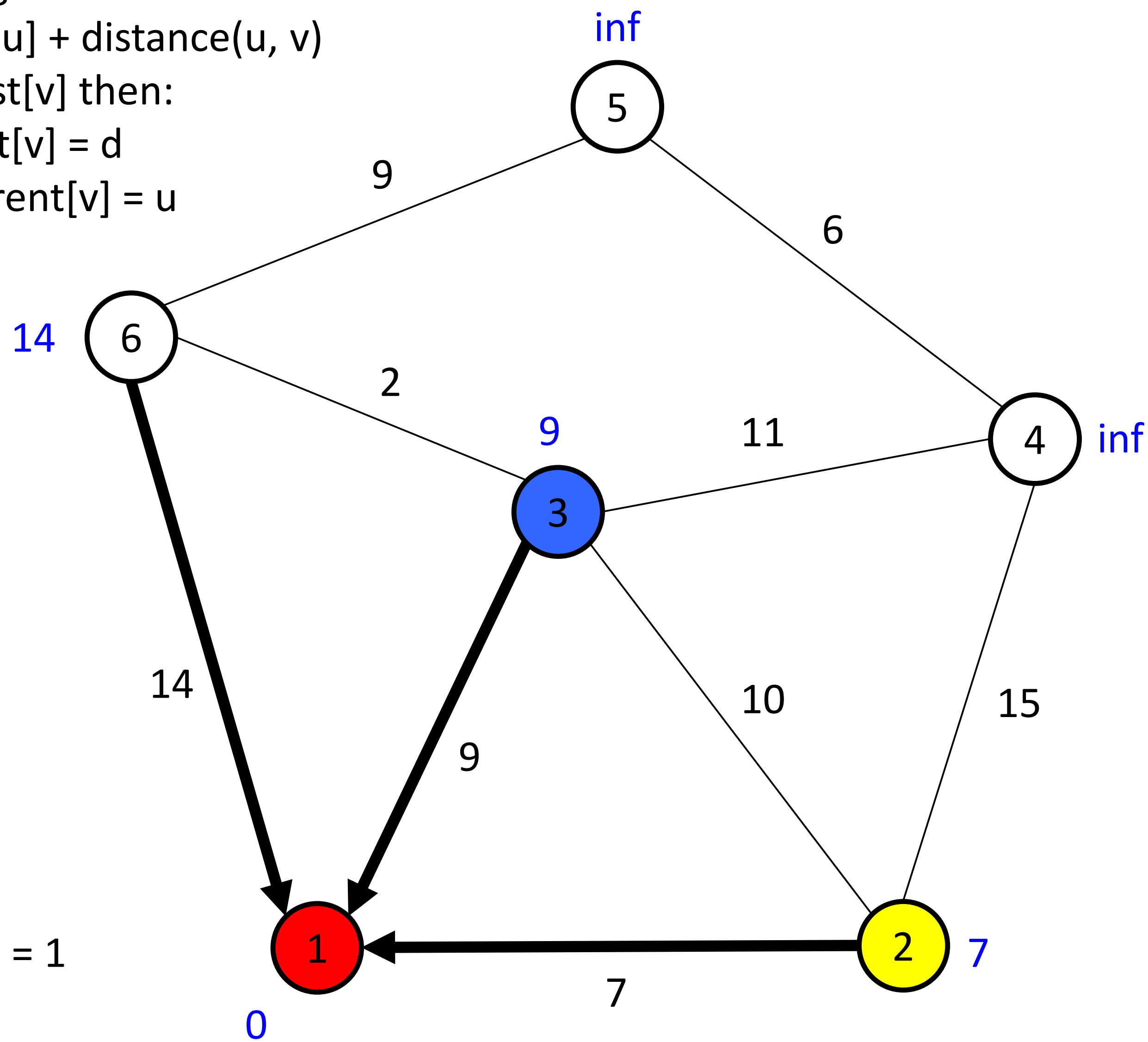
Q=[~~2,~~3,4,5,6]
U=2
V=

Let u = get vertex in Q with smallest distance value (node 2)

Remove u (node 2) from Q
For each neighbor v of u:
    d = dist[u] + distance(u, v)
    if d < dist[v] then:
        dist[v] = d
        parent[v] = u

x	dist[x]	par[x]
1	0	
2	7	1
3	9	1
4	inf	
5	inf	
6	14	1

Q=[~~2,~~3,4,5,6]
U=2
V=3



* Source = 1

parent[v] dist[v]

Remove u (node 2) from Q

For each neighbor v of u:

    d = dist[u] + distance(u, v)

    if d < dist[v] then:

        dist[v] = d

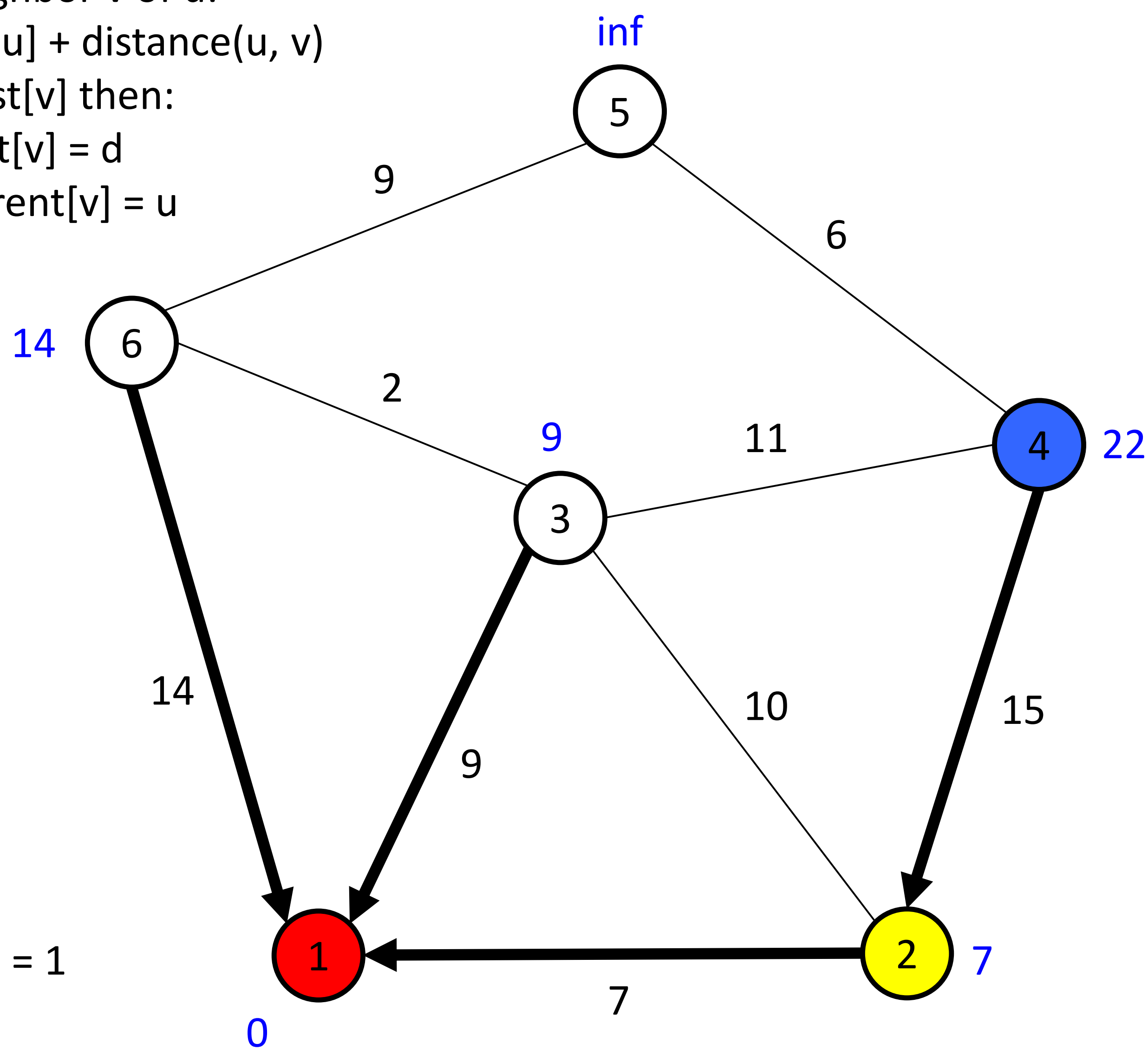
        parent[v] = u

x	dist[x]	par[x]
1	0	
2	7	1
3	9	1
4	22	2
5	inf	
6	14	1

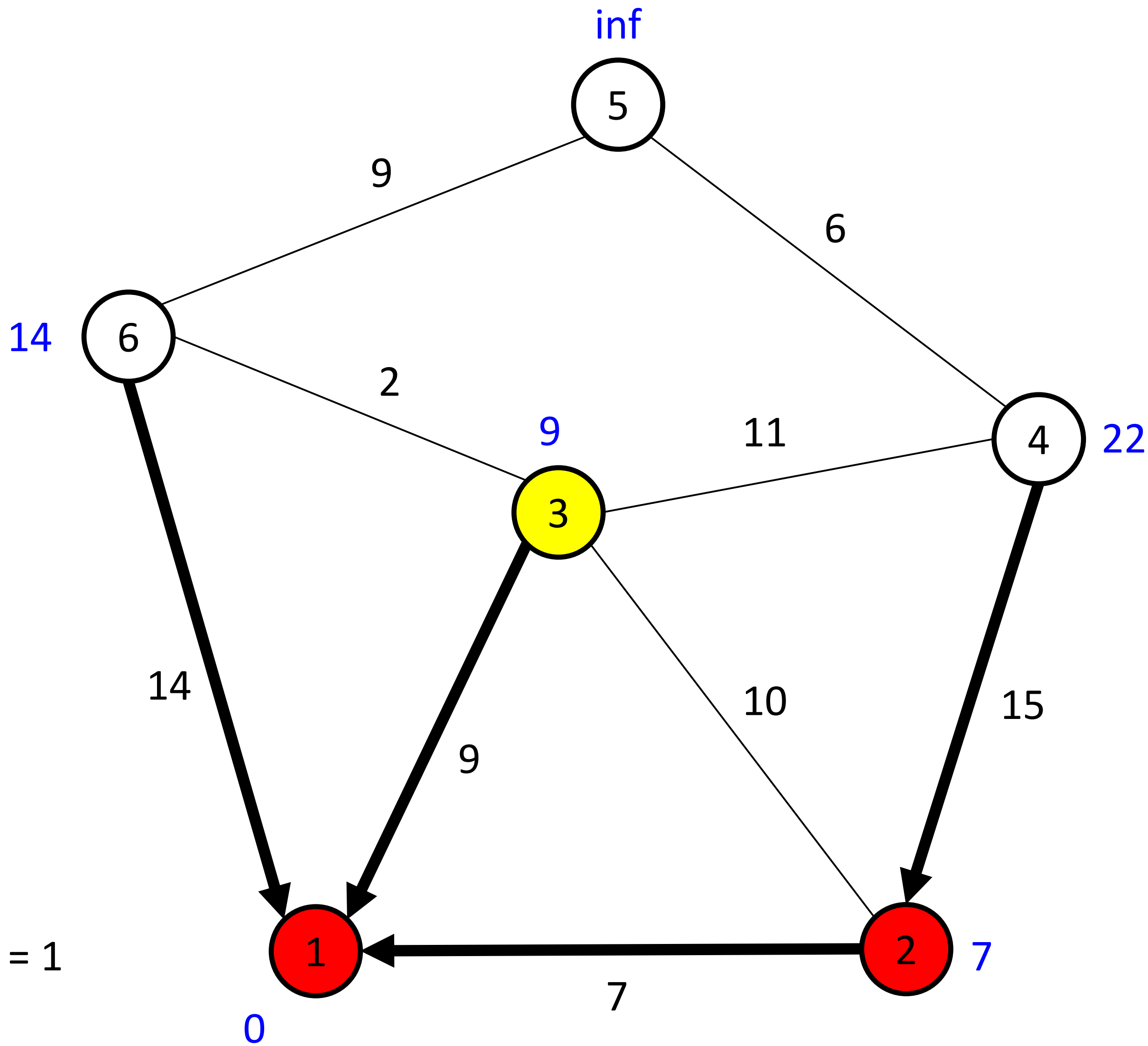
Q=[~~2,~~3,4,5,6]

U=2

V=4



* Source = 1



x	dist[x]	par[x]
1	0	
2	7	1
3	9	1
4	22	2
5	inf	
6	14	1

Q=[~~3,~~4,5,6]
U=3
V=

* Source = 1

Let u = get vertex in Q with smallest distance value (node 3)

Remove u (node 3) from Q

For each neighbor v of u:

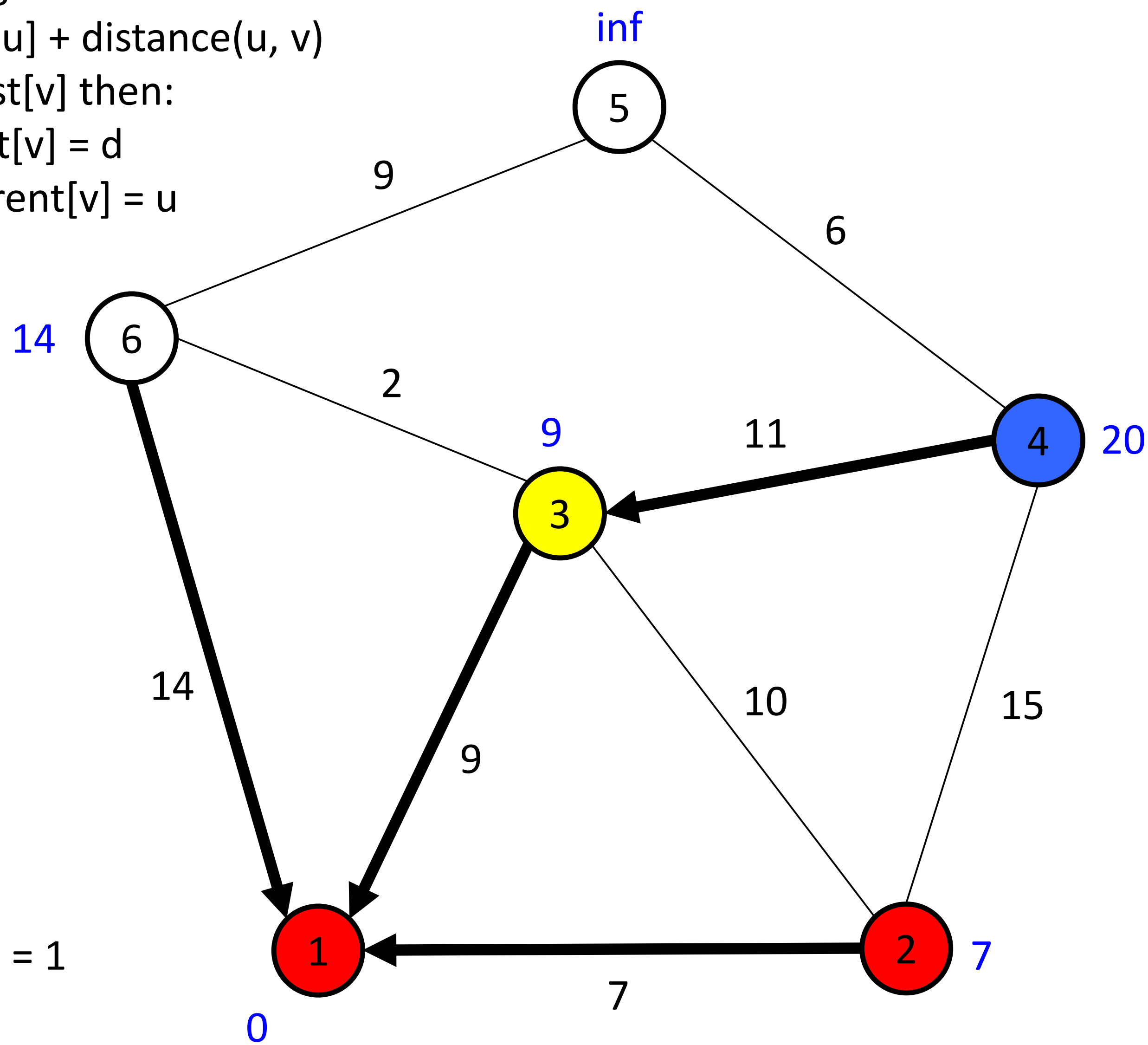
    d = dist[u] + distance(u, v)

    if d < dist[v] then:

        dist[v] = d

        parent[v] = u

x	dist[x]	par[x]
1	0	
2	7	1
3	9	1
4	20	3
5	inf	
6	14	1



Q=[~~3,~~4,5,6]

U=3

V=4

* Source = 1

parent[v]   dist[v]

Remove u (node 3) from Q

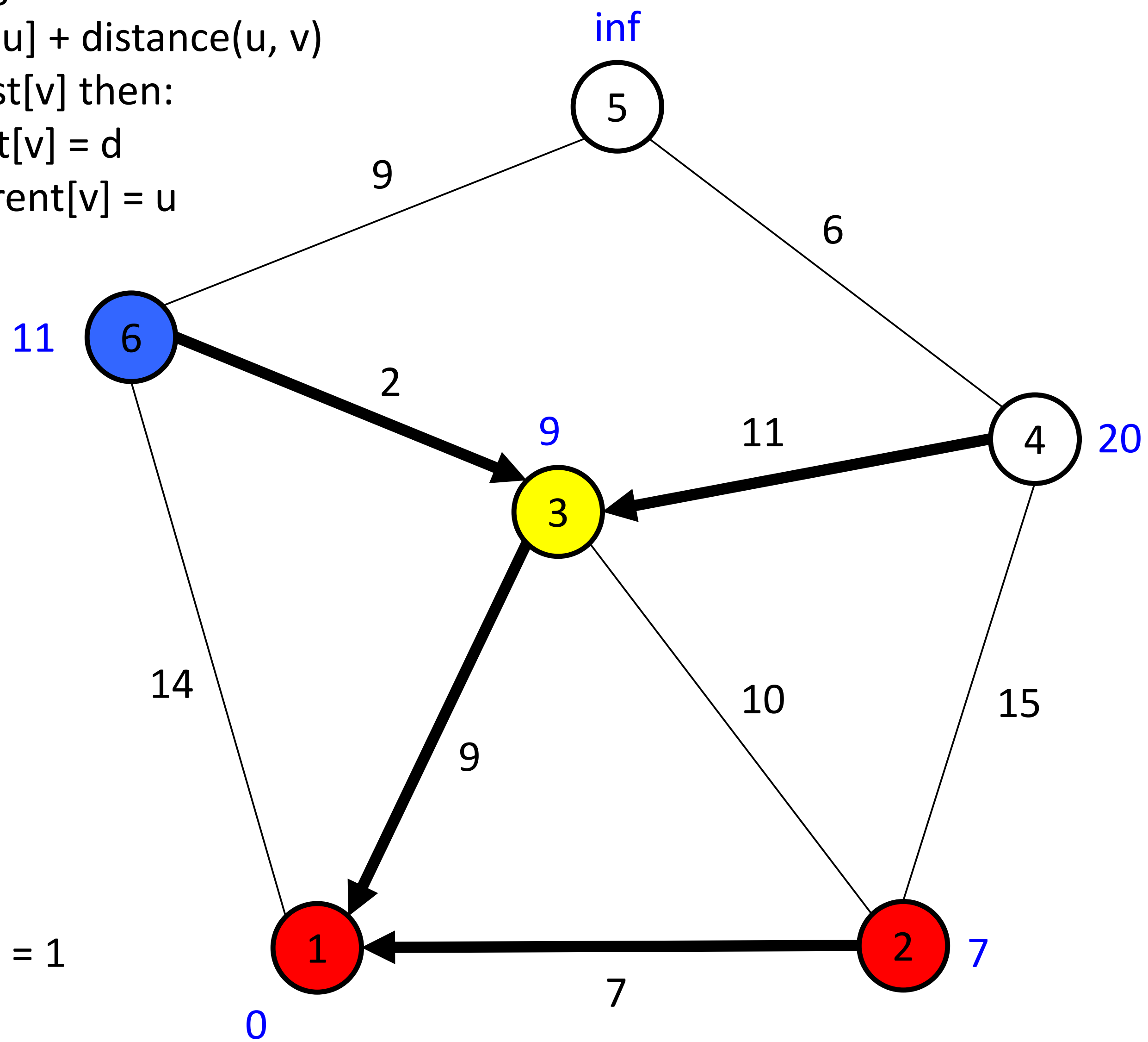
For each neighbor v of u:

    d = dist[u] + distance(u, v)

    if d < dist[v] then:

        dist[v] = d

        parent[v] = u



* Source = 1

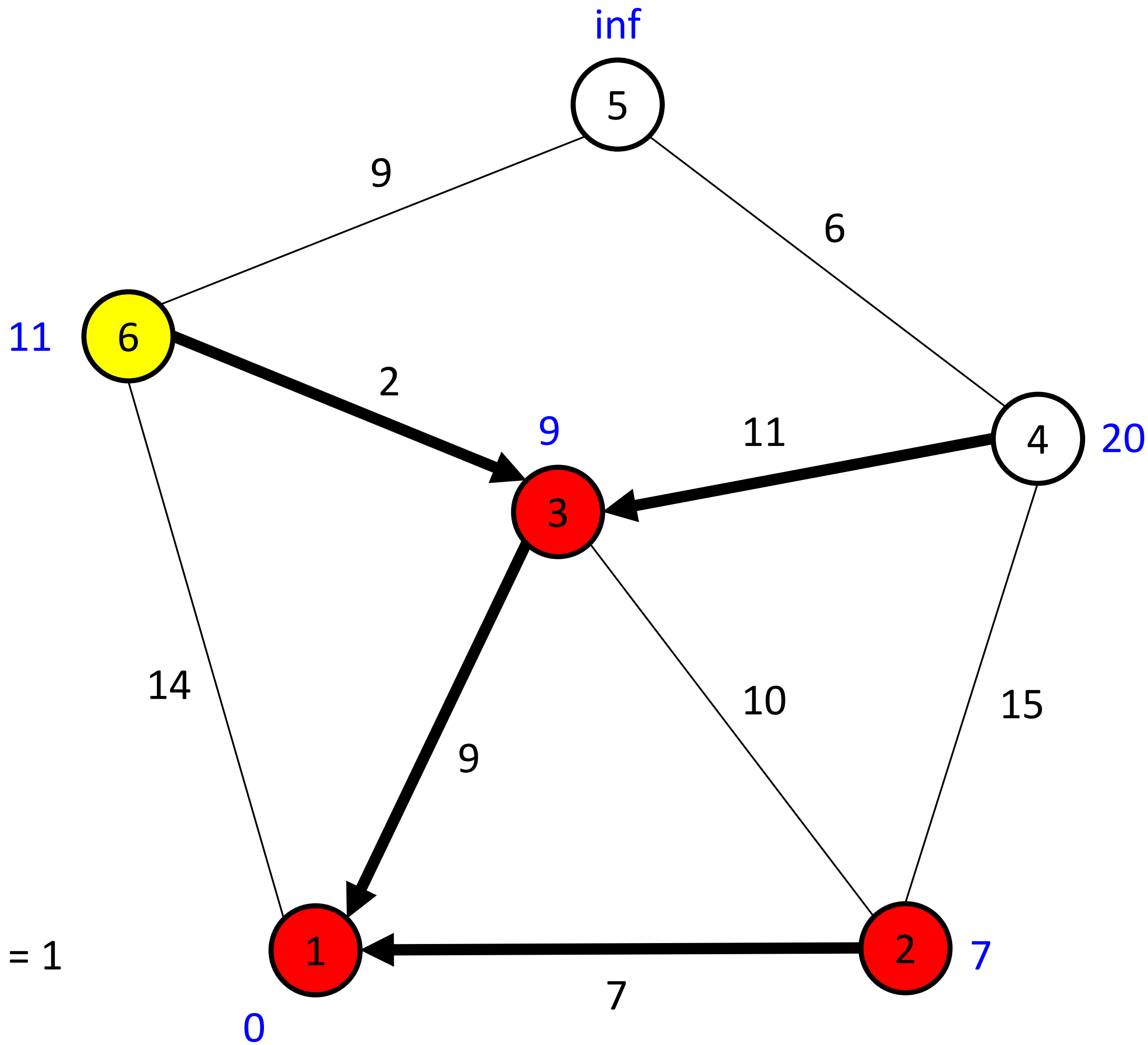
x	dist[x]	par[x]
1	0	
2	7	1
3	9	1
4	20	3
5	inf	
6	11	3

Q=[~~3,~~4,5,6]

U=3

V=6

parent[v]   dist[v]



x	dist[x]	par[x]
1	0	
2	7	1
3	9	1
4	20	3
5	inf	
6	11	3

Q=[4,5~~,6~~]
U=6
V=

* Source = 1

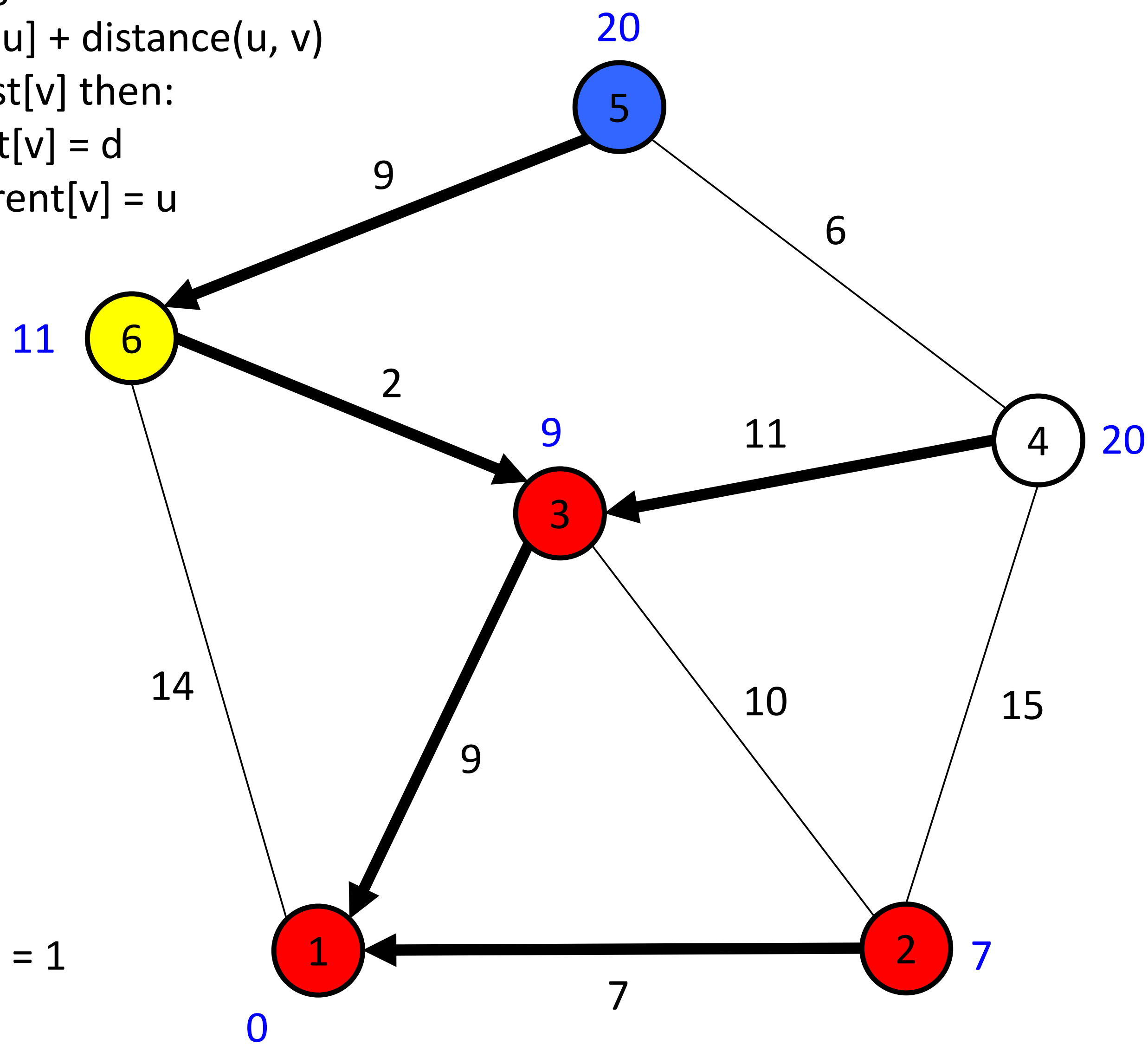
Let u = get vertex in Q with smallest distance value (node 6)

Remove u (node 6) from Q

For each neighbor v of u:

    d = dist[u] + distance(u, v)
    if d < dist[v] then:
        dist[v] = d
        parent[v] = u

x	dist[x]	par[x]
1	0	
2	7	1
3	9	1
4	20	3
5	20	6
6	11	3



Q=[4,5,~~6~~]

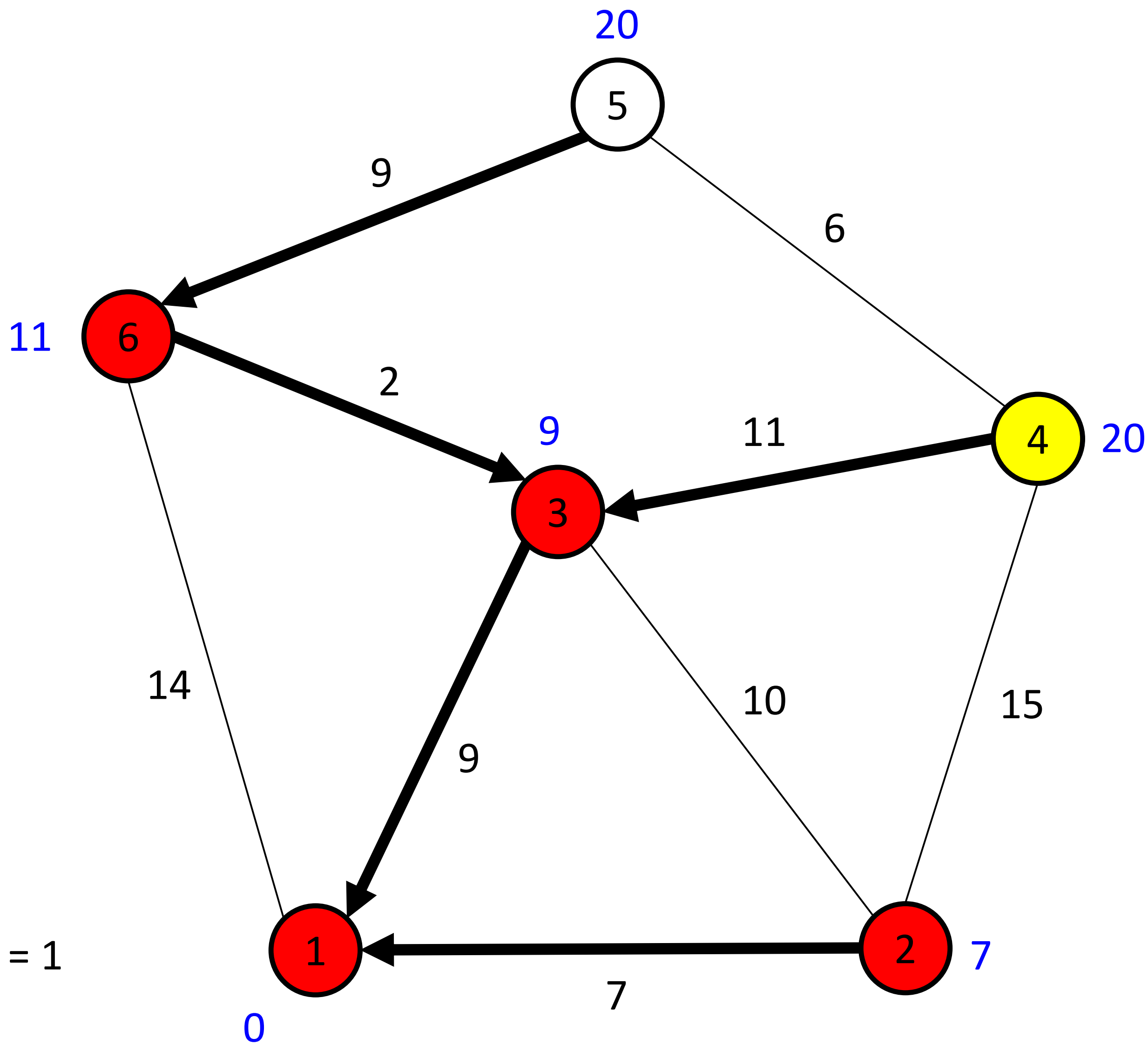
U=6

V=5

* Source = 1

parent[v] dist[v]

x	dist[x]	par[x]
1	0	
2	7	1
3	9	1
4	20	3
5	20	6
6	11	3

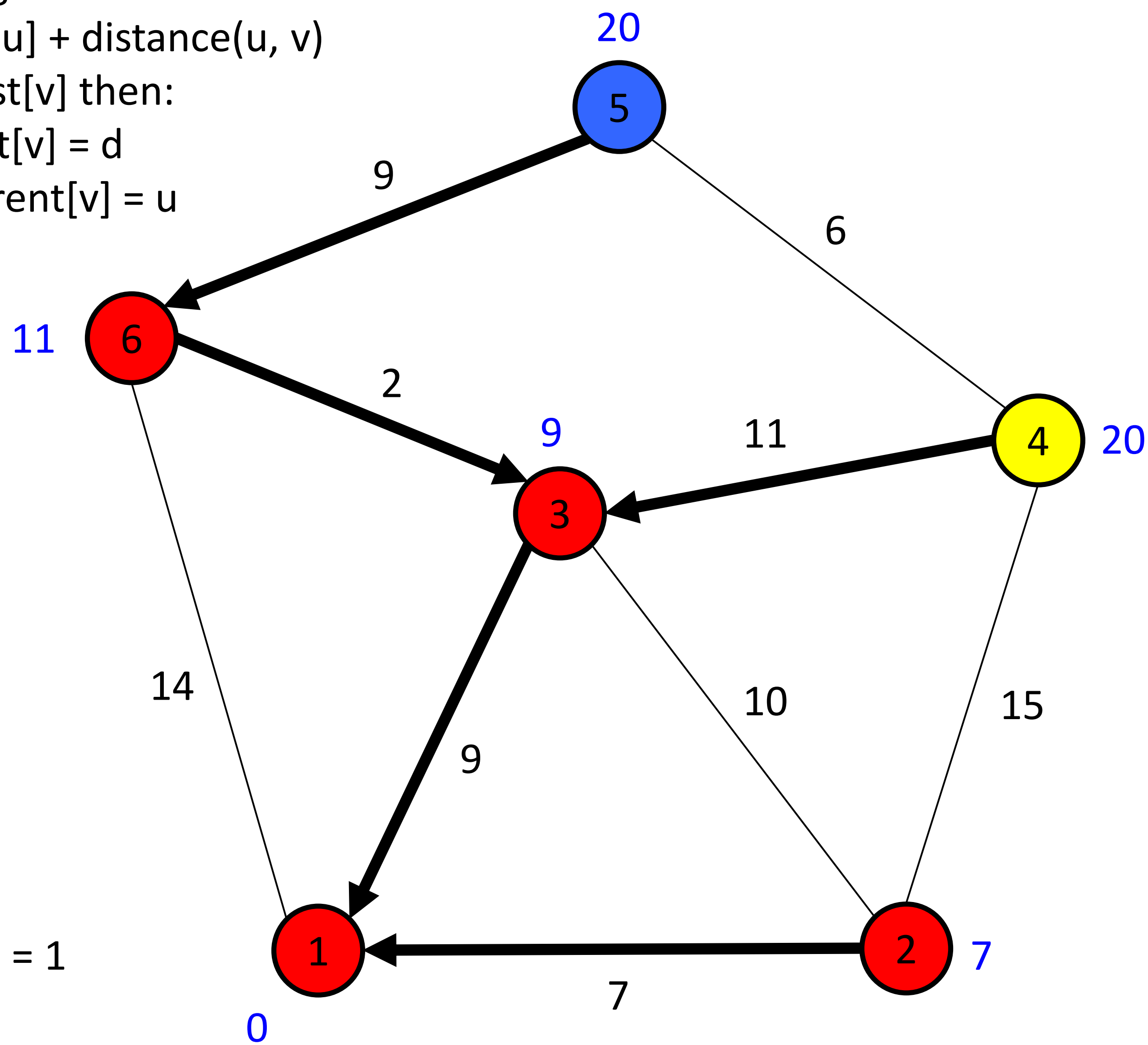


Q=[~~4,~~5]
U=4
V=

* Source = 1

Let u = get vertex in Q with smallest distance value (node 4)

Remove u (node 4) from Q
For each neighbor v of u:
    d = dist[u] + distance(u, v)
    if d < dist[v] then:
        dist[v] = d
        parent[v] = u

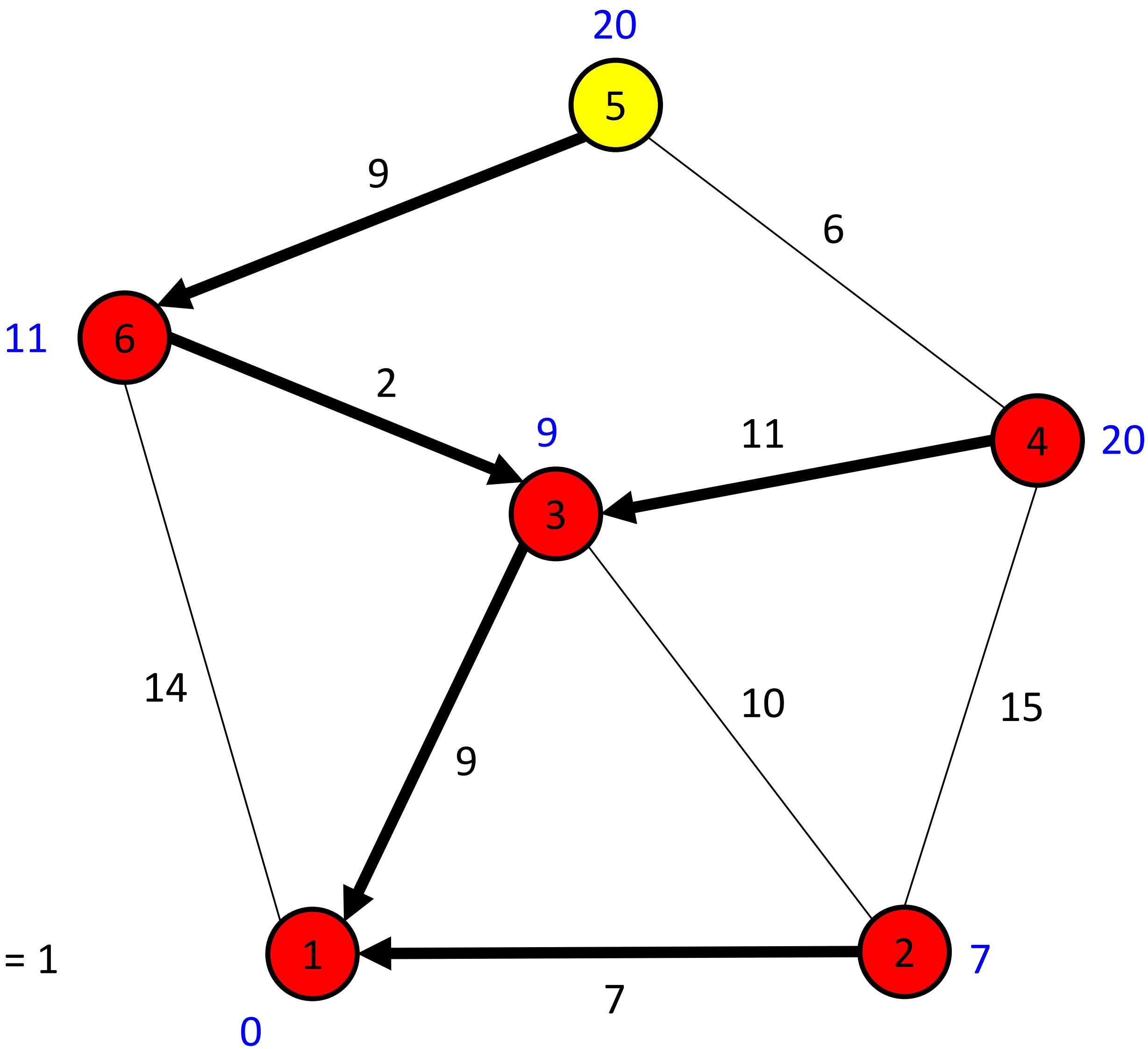


x	dist[x]	par[x]
1	0	
2	7	1
3	9	1
4	20	3
5	20	6
6	11	3

Q=[~~4,~~5]
U=4
V=5

* Source = 1

parent[v]   dist[v]

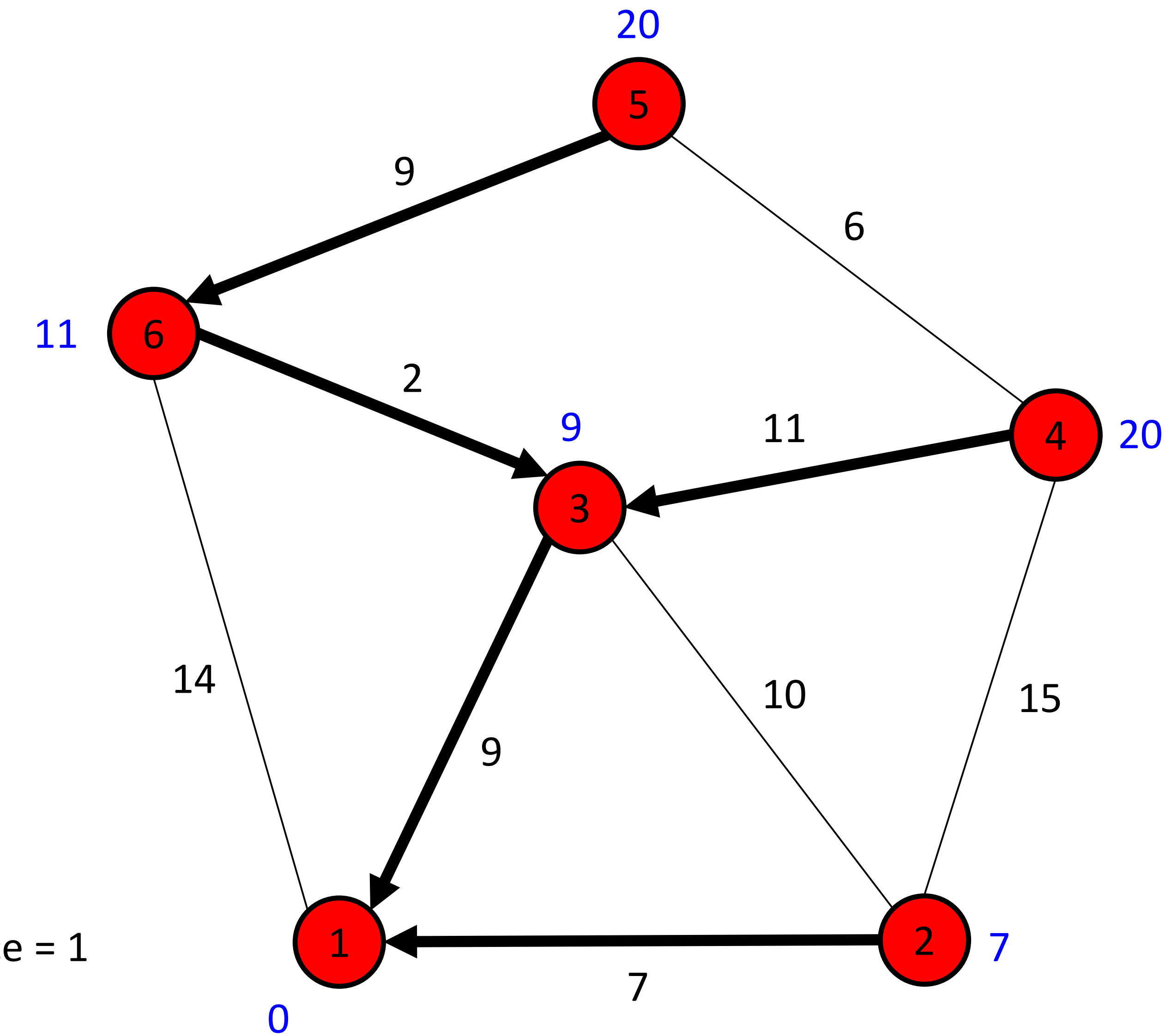


* Source = 1

x	dist[x]	par[x]
1	0	
2	7	1
3	9	1
4	20	3
5	20	6
6	11	3

Q=[~~5~~]
U=5
V=

Let u = get vertex in Q with smallest distance value (node 5)



x	dist[x]	par[x]
1	0	
2	7	1
3	9	1
4	20	3
5	20	6
6	11	3

Q=[ ]
U=5
V=

* Source = 1

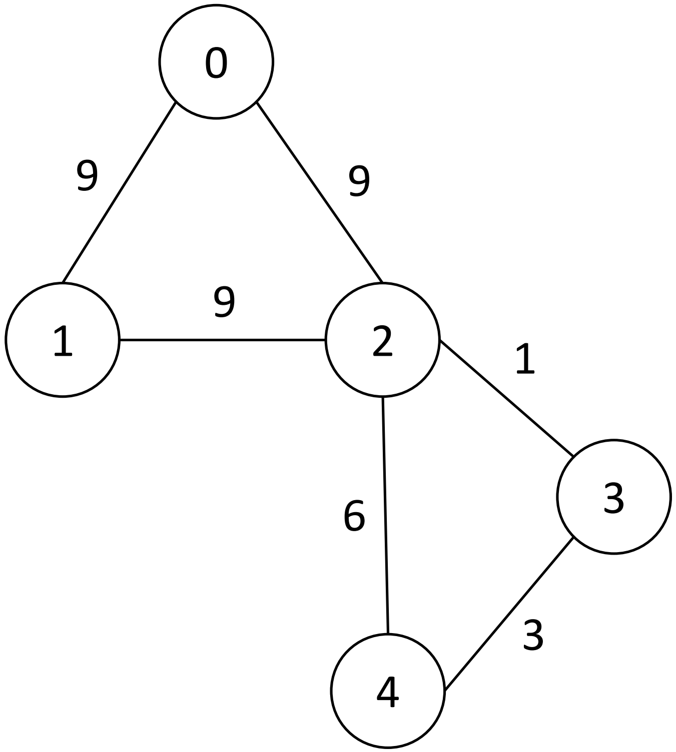
* We now know the shortest distance and shortest path to all nodes from node 1.

Floyd-Warshall algorithm

- All-pairs shortest path algorithm
- Tells you path from all nodes to all other nodes in weighted graph
- Positive or negative edge weights, but no negative cycles (edges sum to negative)
- Incrementally improves estimate
- $O(|V|^3)$
- [Use Dijkstra from each starting vertex when the graph is sparse and has non-negative edges]

Given: G=(V,E), source

```
For each edge (u, v) do:
    dist[u][v] = weight of edge (u, v) or infinity
    next[u][v] = v
For k = 1 to |V| do:                  ← Intermediate node
    for i = 1 to |V| do:              ← Start node
        for j = 1 to |V| do:          ← End node
            if dist[i][k] + dist[k][j] < dist[i][j] then:
                dist[i][j] = dist[i][k] + dist[k][j]
                next[i][j] = next[i][k]
```



Distance

	0	**1**	**2**	**3**	**4**
0	INF	9	9	INF	INF
1	9	INF	9	INF	INF
2	9	9	INF	1	6
3	INF	INF	1	INF	3
4	INF	INF	6	3	INF

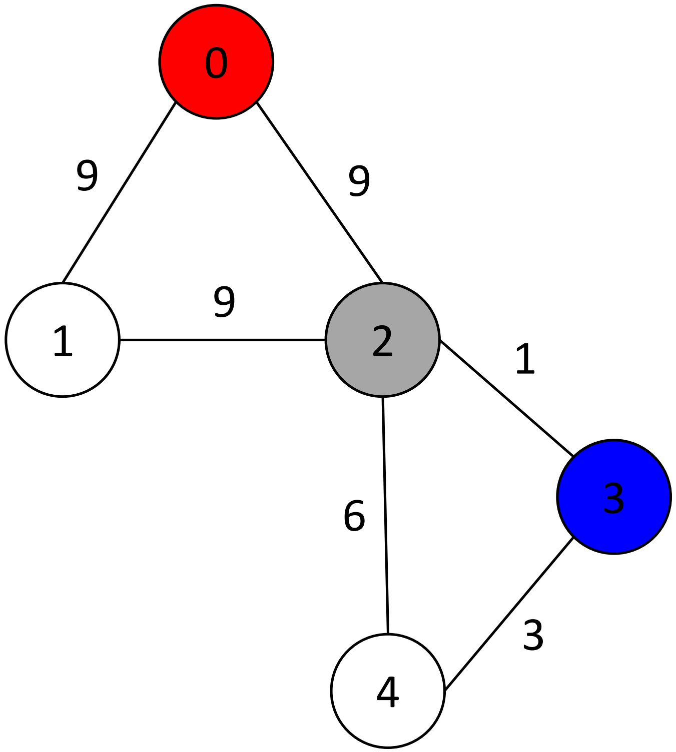
Next

	0	**1**	**2**	**3**	**4**
0		1	2		
1	0		2		
2	0	1		3	4
3			2		4
4			2	3	

k = 0

i = 0

j = 0



Distance

	0	1	2	3	4
0	INF	9	9	INF	INF
1	9	INF	9	INF	INF
2	9	9	INF	1	6
3	INF	INF	1	INF	3
4	INF	INF	6	3	INF

Next

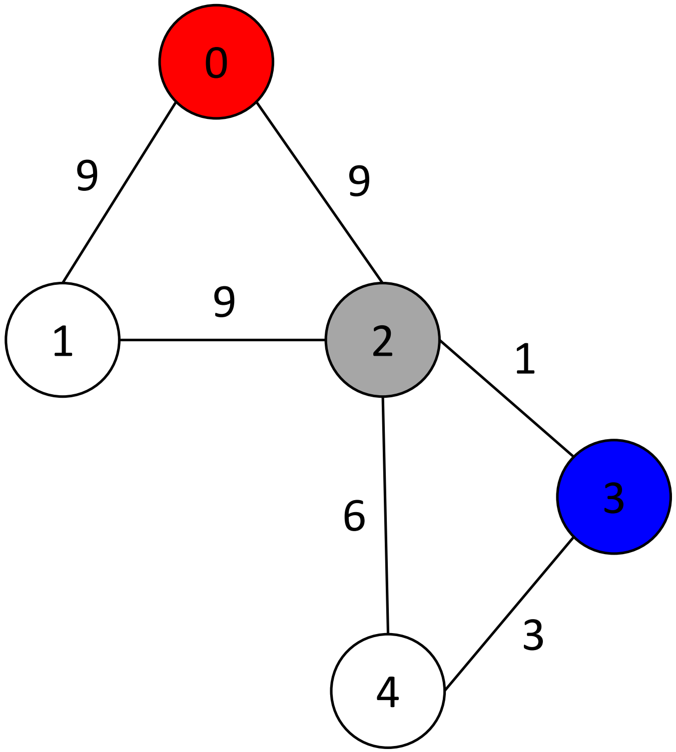
	0	1	2	3	4
0		1	2		
1	0		2		
2	0	1		3	4
3			2		4
4			2	3	

M k = 2

S i = 0

F j = 3

9 + 1 < INF



Distance

	0	1	2	3	4
0	INF	9	9	10	INF
1	9	INF	9	INF	INF
2	9	9	INF	1	6
3	INF	INF	1	INF	3
4	INF	INF	6	3	INF

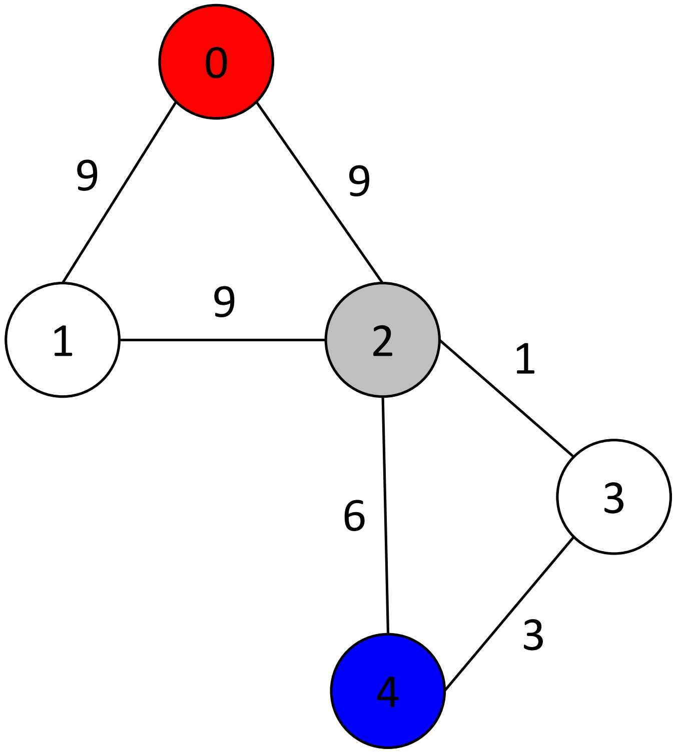
Next

	0	1	2	3	4
0		1	2	2	
1	0		2		
2	0	1		3	4
3			2		4
4			2	3	

M k = 2

S i = 0

F j = 3 0 -> 3: dist 10, goto 2



Distance

	0	1	2	3	4
0	INF	9	9	10	INF
1	9	INF	9	INF	INF
2	9	9	INF	1	6
3	INF	INF	1	INF	3
4	INF	INF	6	3	INF

Next

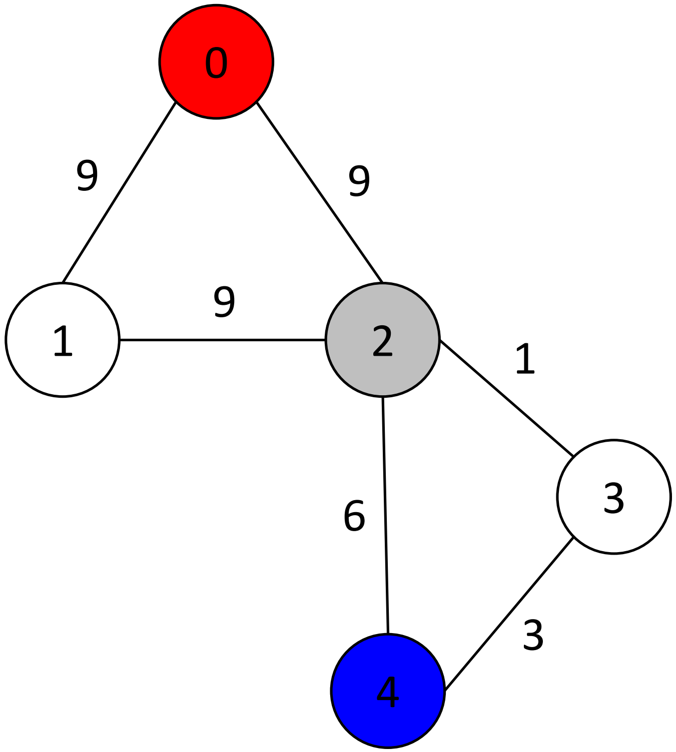
	0	1	2	3	4
0		1	2	2	
1	0		2		
2	0	1		3	4
3			2		4
4			2	3	

M k = 2

S i = 0

F j = 4

9 + 6 < INF



M k = 2

S i = 0

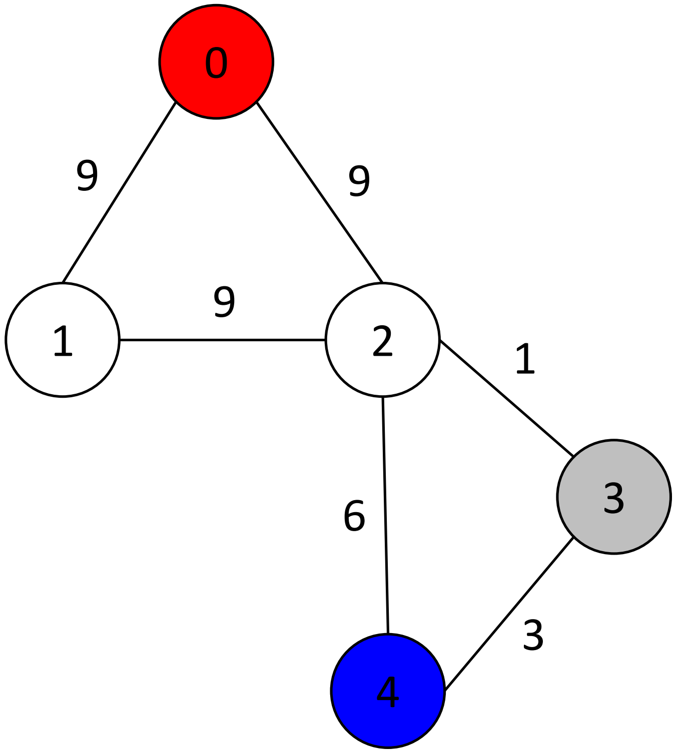
F j = 4 0 -> 3: dist 10, goto 2

Distance

	0	**1**	**2**	**3**	**4**
0	INF	9	9	10	15
1	9	INF	9	INF	INF
2	9	9	INF	1	6
3	INF	INF	1	INF	3
4	INF	INF	6	3	INF

Next

	0	**1**	**2**	**3**	**4**
0		1	2	2	2
1	0		2		
2	0	1		3	4
3			2		4
4			2	3	



Distance

	0	1	2	3	4
0	INF	9	9	10	15
1	9	INF	9	INF	INF
2	9	9	INF	1	6
3	INF	INF	1	INF	3
4	INF	INF	6	3	INF

Next

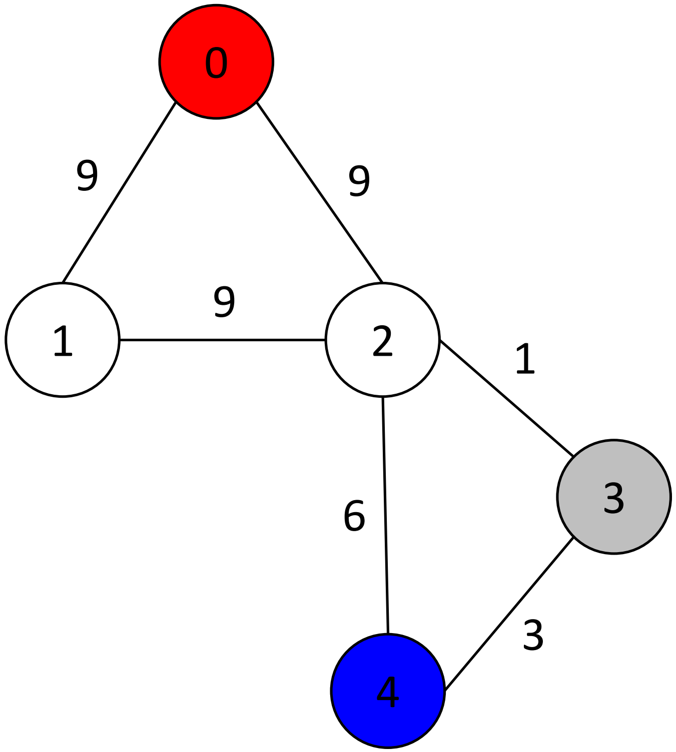
	0	1	2	3	4
0		1	2	2	2
1	0		2		
2	0	1		3	4
3			2		4
4			2	3	

M $k = 3$

S $i = 0$

F $j = 4$

$10 + 3 < 15$



Distance

	0	1	2	3	4
0	INF	9	9	10	13
1	9	INF	9	INF	INF
2	9	9	INF	1	6
3	INF	INF	1	INF	3
4	INF	INF	6	3	INF

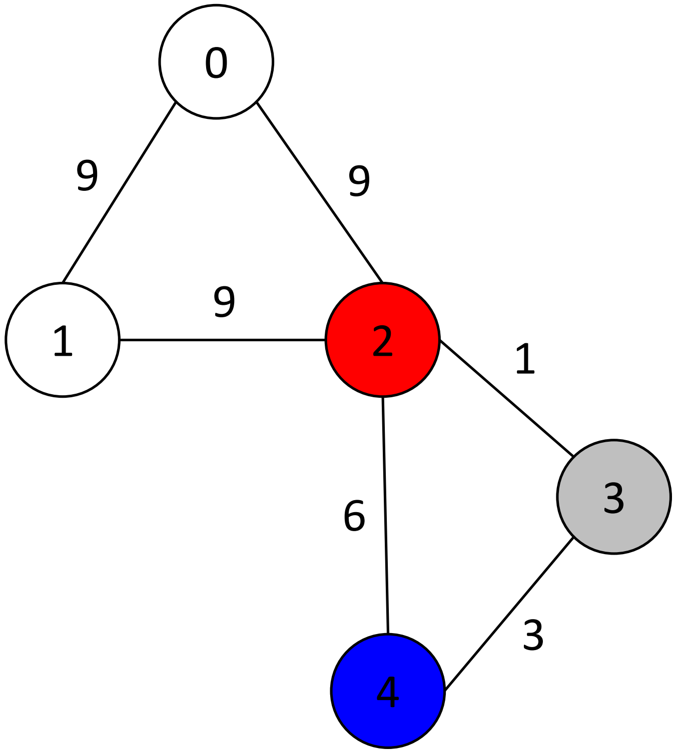
Next

	0	1	2	3	4
0		1	2	2	2
1	0		2		
2	0	1		3	4
3			2		4
4			2	3	

M k = 3

S i = 0

F j = 4 0 -> 4: dist 13, goto 2



Distance

	0	1	2	3	4
0	INF	9	9	10	13
1	9	INF	9	INF	INF
2	9	9	INF	1	6
3	INF	INF	1	INF	3
4	INF	INF	6	3	INF

Next

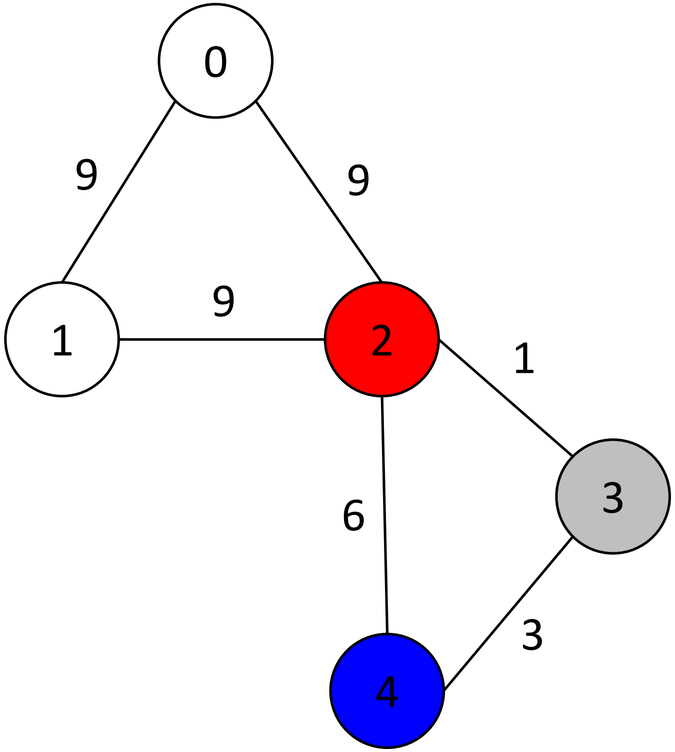
	0	1	2	3	4
0		1	2	2	2
1	0		2		
2	0	1		3	4
3			2		4
4			2	3	

M k = 3

S i = 2

F j = 4

1 + 3 < 6



Distance

	0	1	2	3	4
0	INF	9	9	10	13
1	9	INF	9	INF	INF
2	9	9	INF	1	4
3	INF	INF	1	INF	3
4	INF	INF	6	3	INF

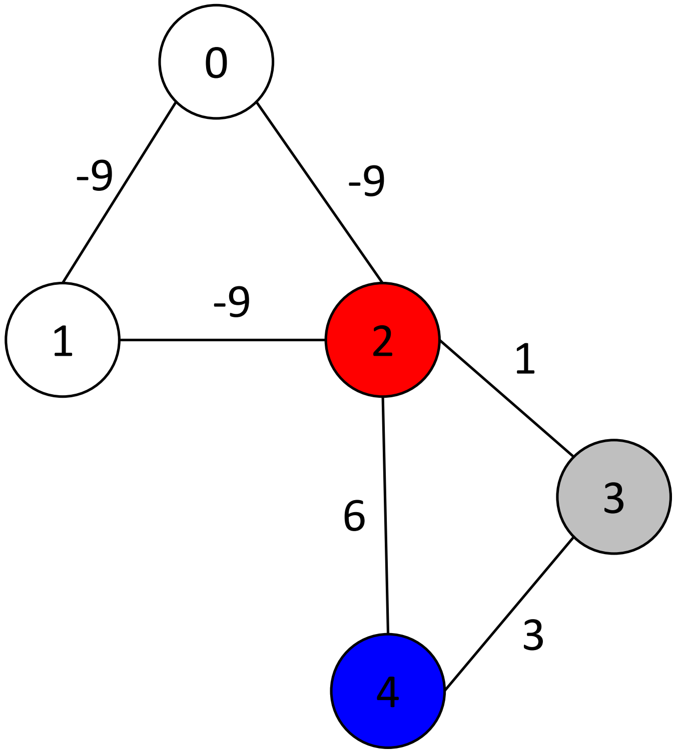
Next

	0	1	2	3	4
0		1	2	2	2
1	0		2		
2	0	1		3	3
3			2		4
4			2	3	

M k = 3

S i = 2

F j = 4 2 -> 4: dist 4, goto 3



Distance

	0	1	2	3	4
0	-3	-9	-9	8	4
1	-9	-3	-9	INF	INF
2	-9	-9	-3	1	4
3	INF	INF	1	INF	3
4	INF	INF	6	3	INF

Reconstructing the path

Want to go from u to v

```
if next[u][v] is empty then return null path
path = (u)
while u <> v do:
        u = next[u][v]
        path.append(u)
return path
```

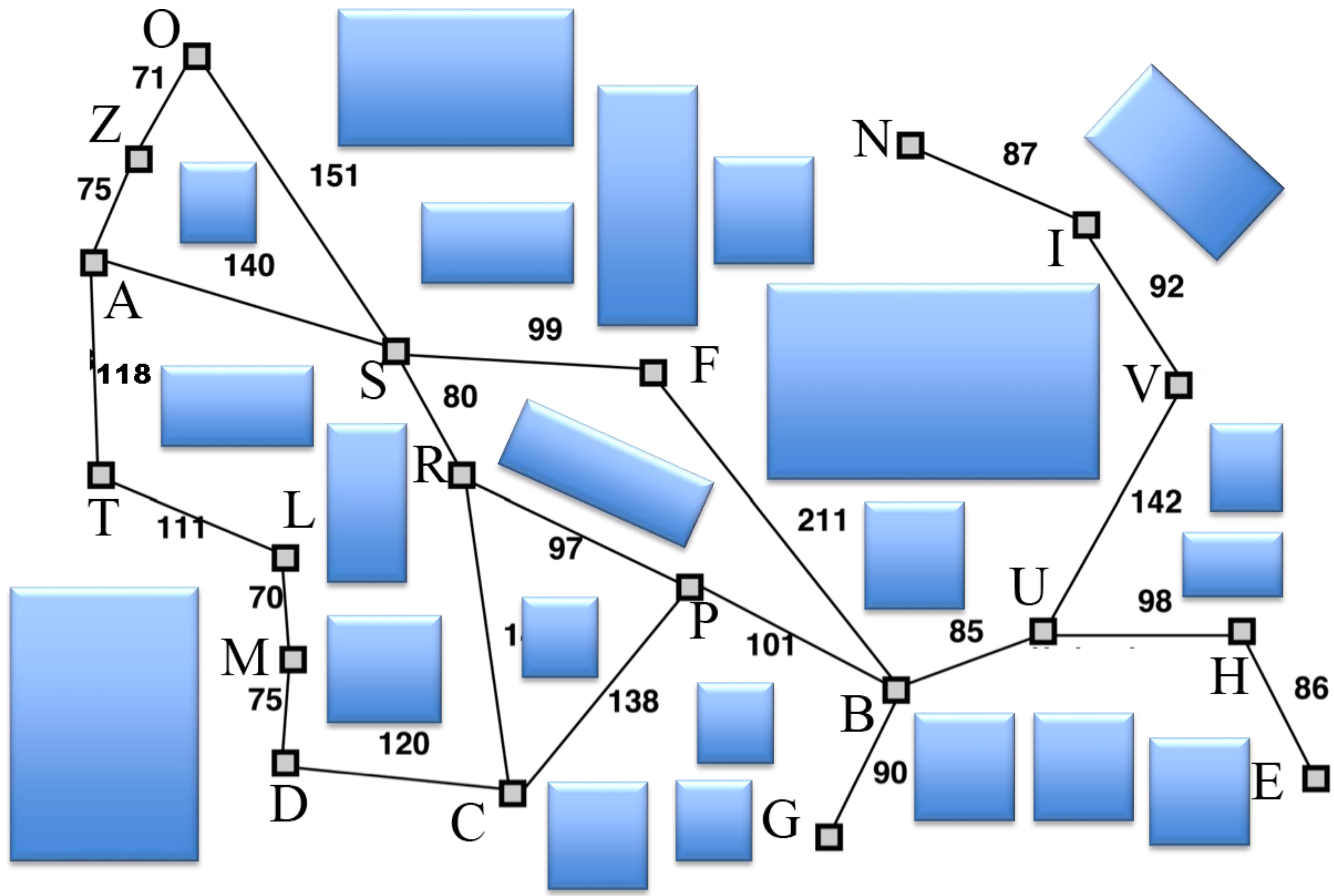
Dynamic environments

- Terrain can change
  - Jumpable?
  - Kickable?
  - Too big to jump/kick?
- Typically: destructible environments
- Path network edges can be eliminated
- Path network edges can be created

Heuristic Search

- Find shortest path from a single source to a single destination
- Heuristic function:
  - We have some knowledge about how far away any given state from the goal, in terms of operation cost
  - For navigation: Euclidean distance, Manhattan distance

A	366
B	0
C	160
D	242
E	161
F	176
G	77
H	151
I	226
L	244
M	241
N	234
O	380
P	100
S	253
T	329
U	80
V	199
Z	374



A* Search

- Single source, single target graph search
- Generalization of Dijkstra
- Guaranteed to return the optimal path if the heuristic is admissible; quick and accurate
- Evaluate each state: f(n) = g(n) + h(n)
- Open list: nodes that are known and waiting to be visited
- Closed list: nodes that have been visited

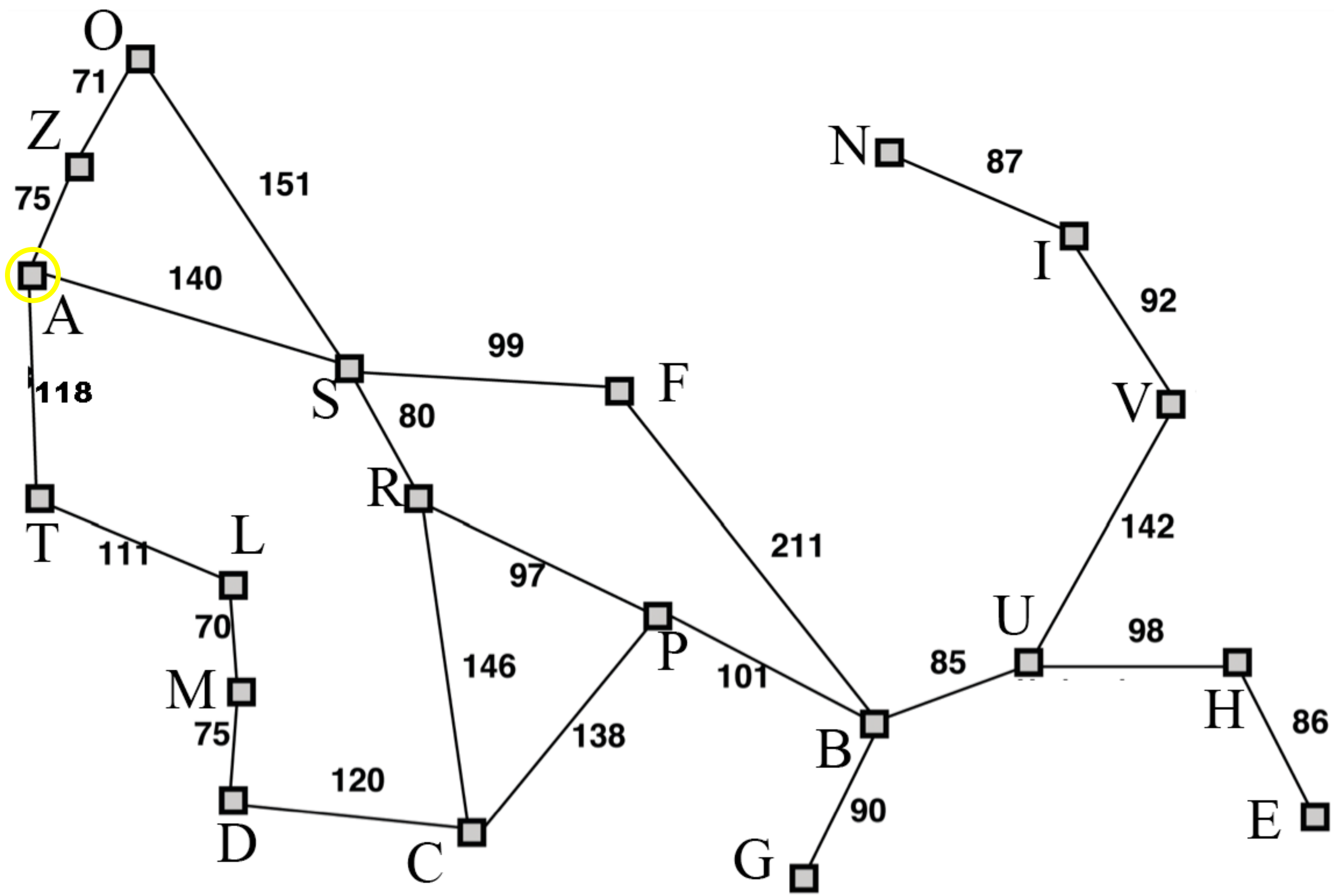
A*

Given: init, goal(s), ops

```
ops = {…}
closed = nil
open = {init}
current = init
while (NOT isgoal(current) AND open <> nil)
      closed = closed + {current}
      open = open – {current}
                  + (successors(current, ops) – closed)   * Insert according to
      current = first(open)                                  evaluation function
end while
if isgoal(current) then reconstruct solution
else fail
```

Evaluation function f(n) = g(n) + h(n)

A	366
B	0
C	160
D	242
E	161
F	176
G	77
H	151
I	226
L	244
M	241
N	234
O	380
P	100
S	253
T	329
U	80
V	199
Z	374

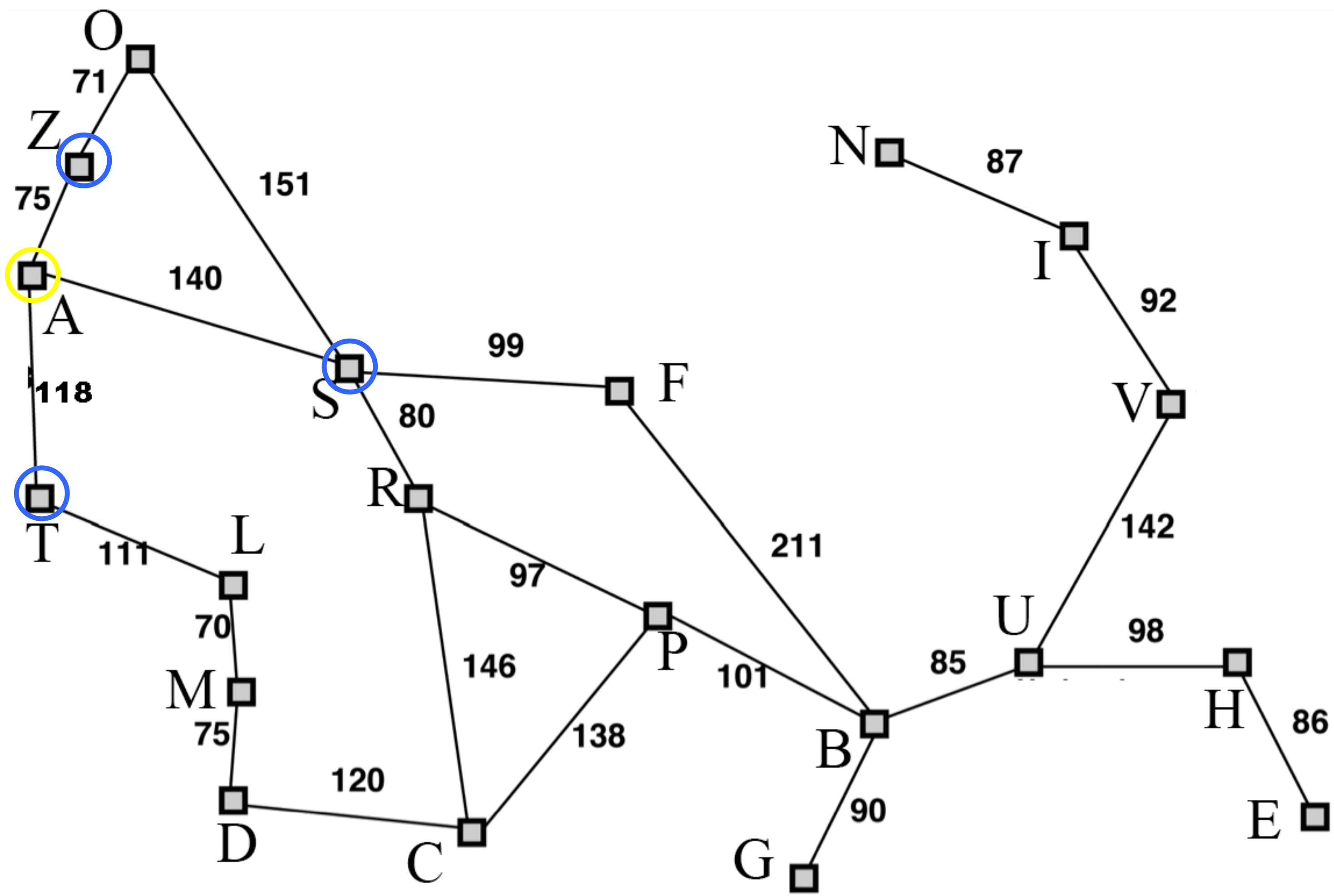


Open: A(366)

Closed:

Evaluation function f(n) = g(n) + h(n)

A	366
B	0
C	160
D	242
E	161
F	176
G	77
H	151
I	226
L	244
M	241
N	234
O	380
P	100
S	253
T	329
U	80
V	199
Z	374

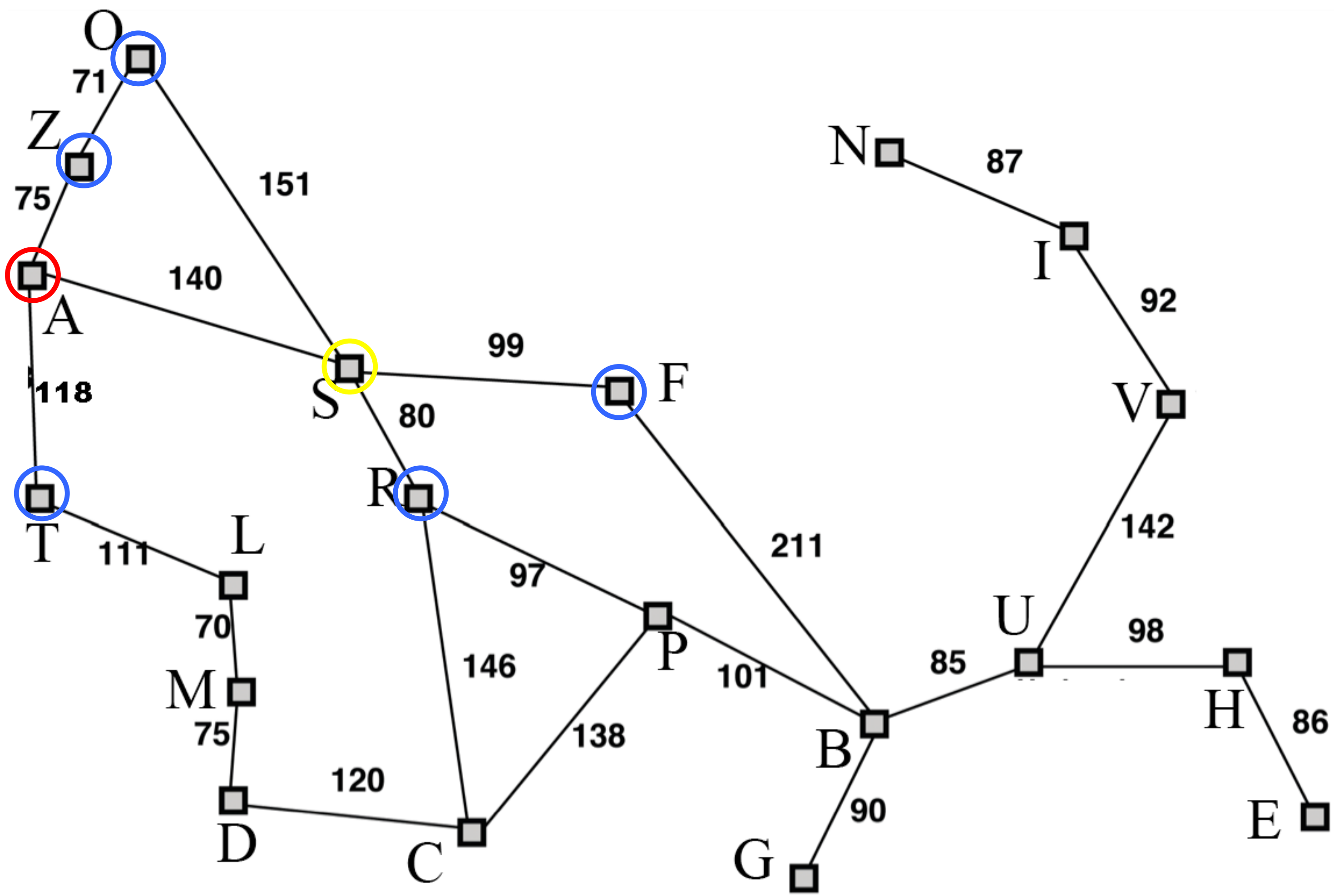


Open: S(253+140=393), T(329+118=447), Z(374+75=449)

Closed: A(366)

Evaluation function f(n) = g(n) + h(n)

A	366
B	0
C	160
D	242
E	161
F	176
G	77
H	151
I	226
L	244
M	241
N	234
O	380
P	100
S	253
T	329
U	80
V	199
Z	374

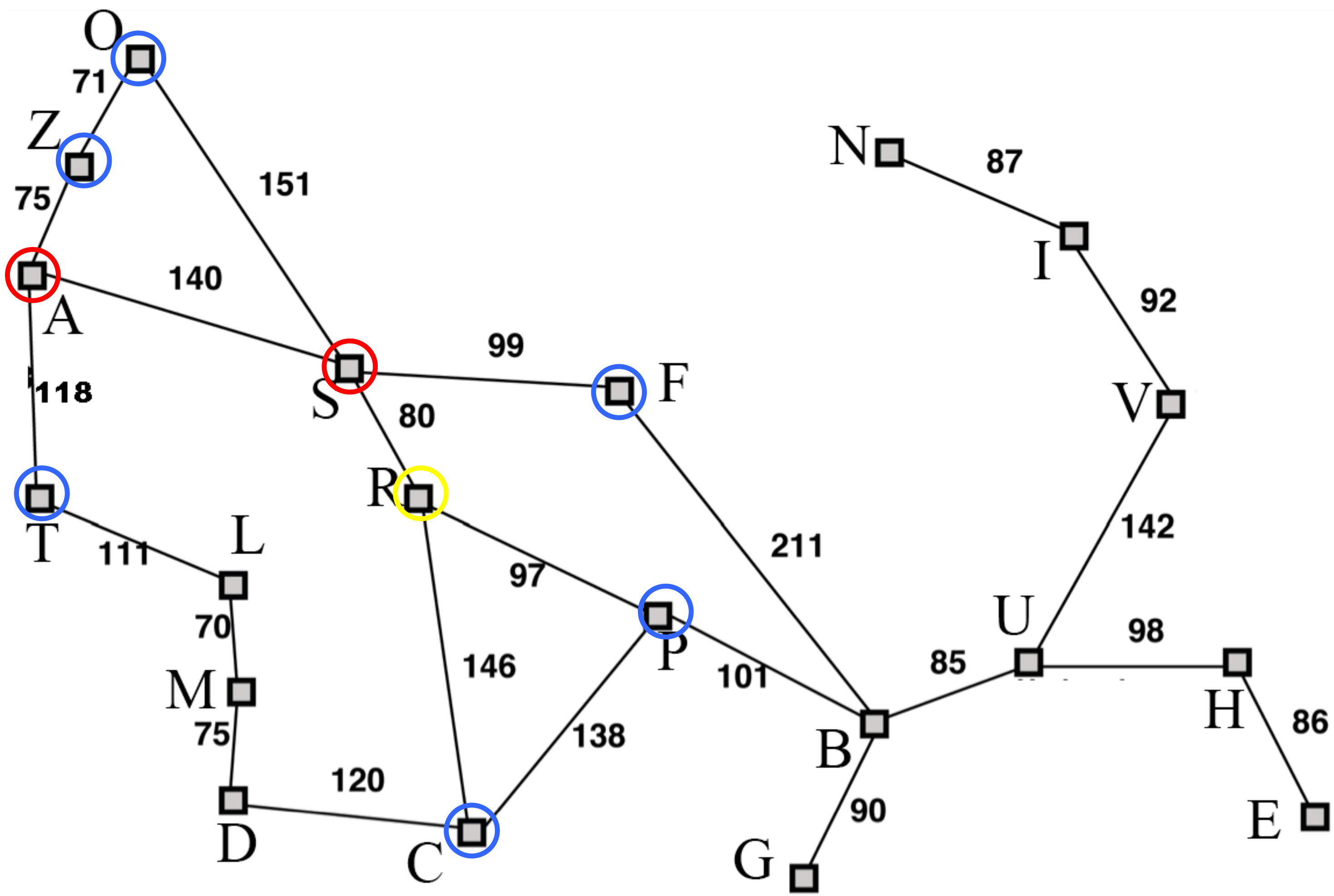


Open: R(220+193=413), F(239+176=415), T(329+118=447), Z(374+75=449), O(291+380=671)

Closed: S(393), A(366)

Evaluation function f(n) = g(n) + h(n)

A	366
B	0
C	160
D	242
E	161
F	176
G	77
H	151
I	226
L	244
M	241
N	234
O	380
P	100
S	253
T	329
U	80
V	199
Z	374

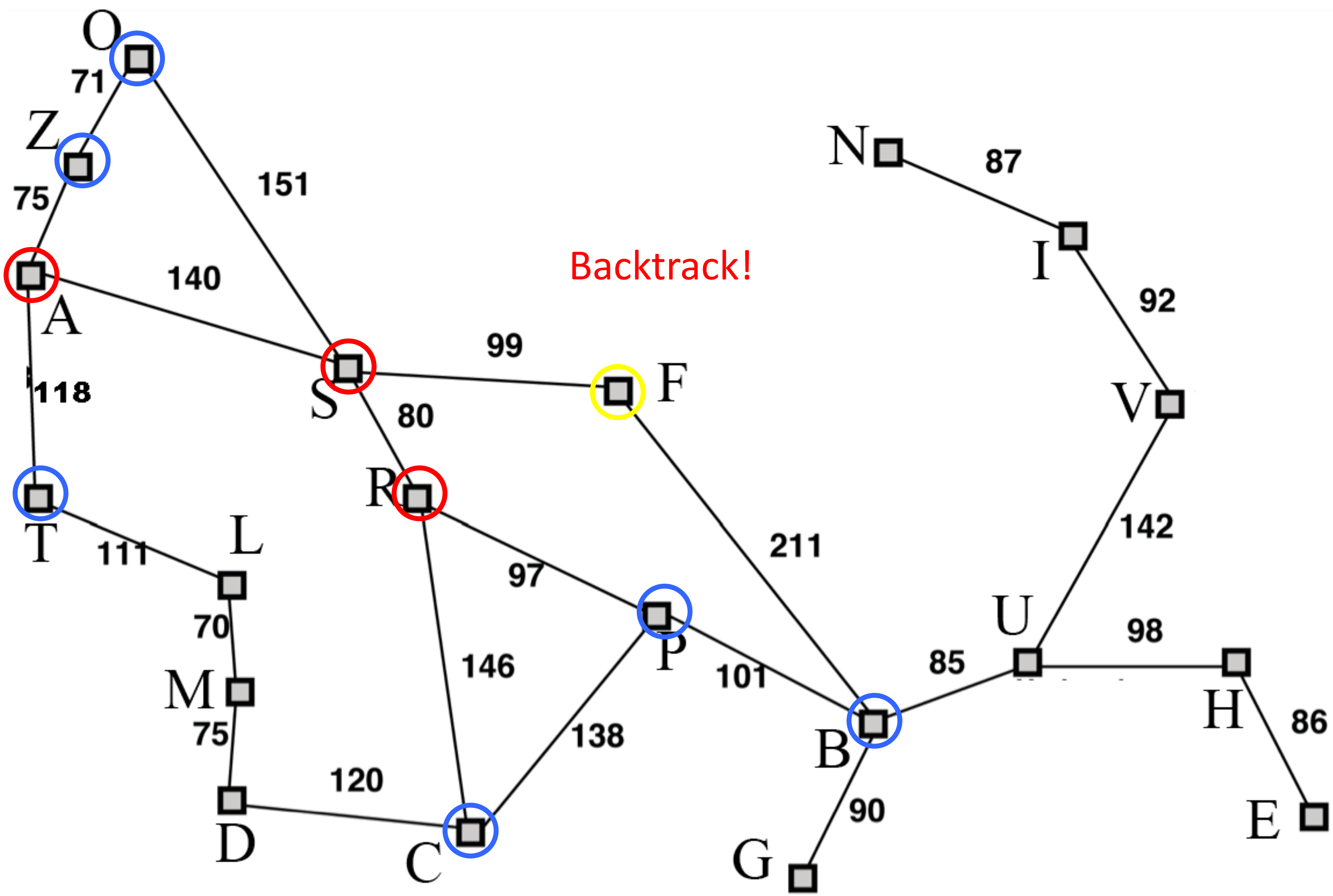


Open: F(239+176=415), P(317+100=417), T(329+118=447), Z(374+75=449), C(366+160=526),

Closed: R(413), S(393), A(366)

Evaluation function f(n) = g(n) + h(n)

A	366
B	0
C	160
D	242
E	161
F	176
G	77
H	151
I	226
L	244
M	241
N	234
O	380
P	100
S	253
T	329
U	80
V	199
Z	374

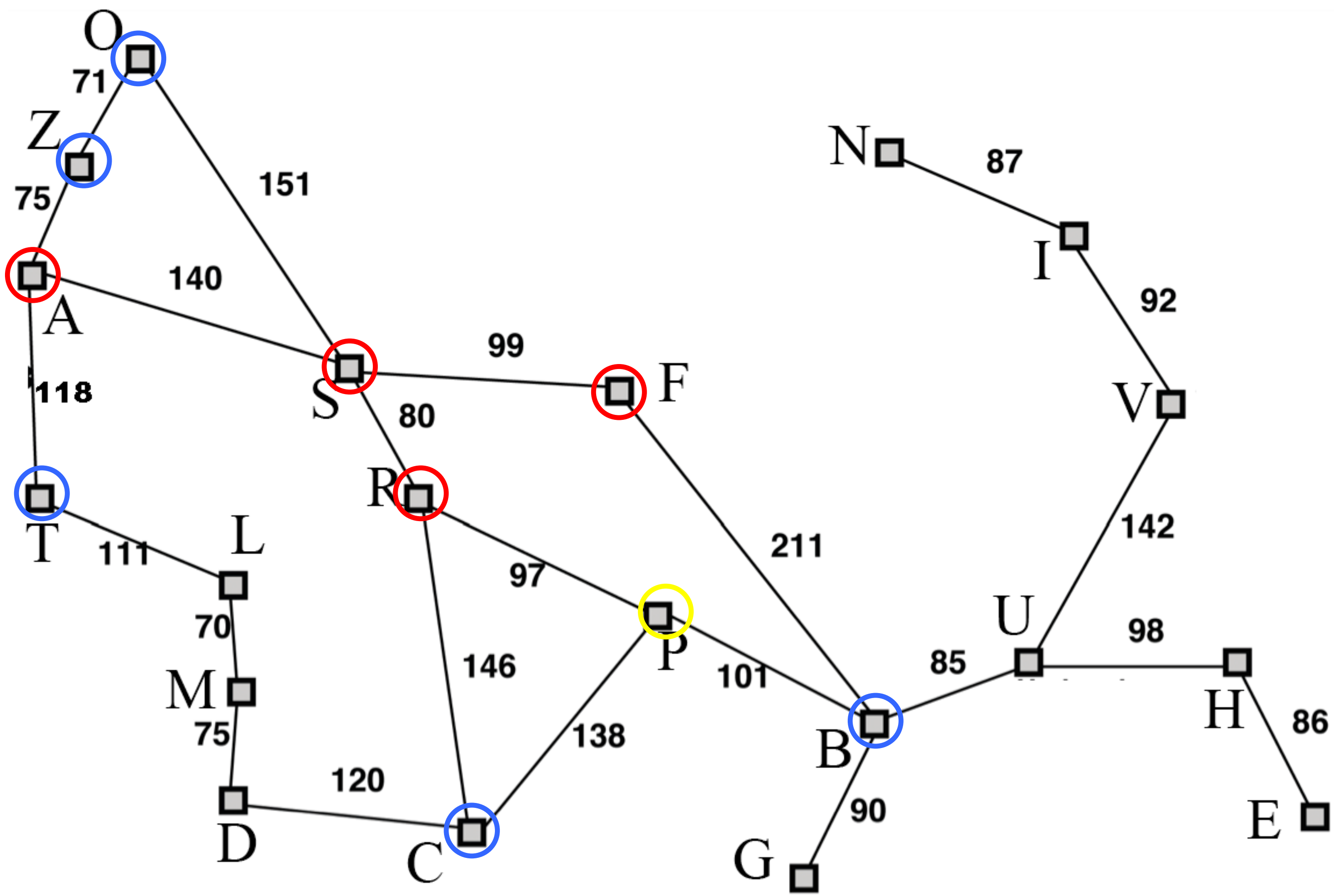


Open: P(317+100=417), T(329+118=447), Z(374+75=449), <u>B(450+0=450)</u>, C(366+160=526), O(

Closed: F(415), R(413), S(393), A(366)

Evaluation function f(n) = g(n) + h(n)

A	366
B	0
C	160
D	242
E	161
F	176
G	77
H	151
I	226
L	244
M	241
N	234
O	380
P	100
S	253
T	329
U	80
V	199
Z	374

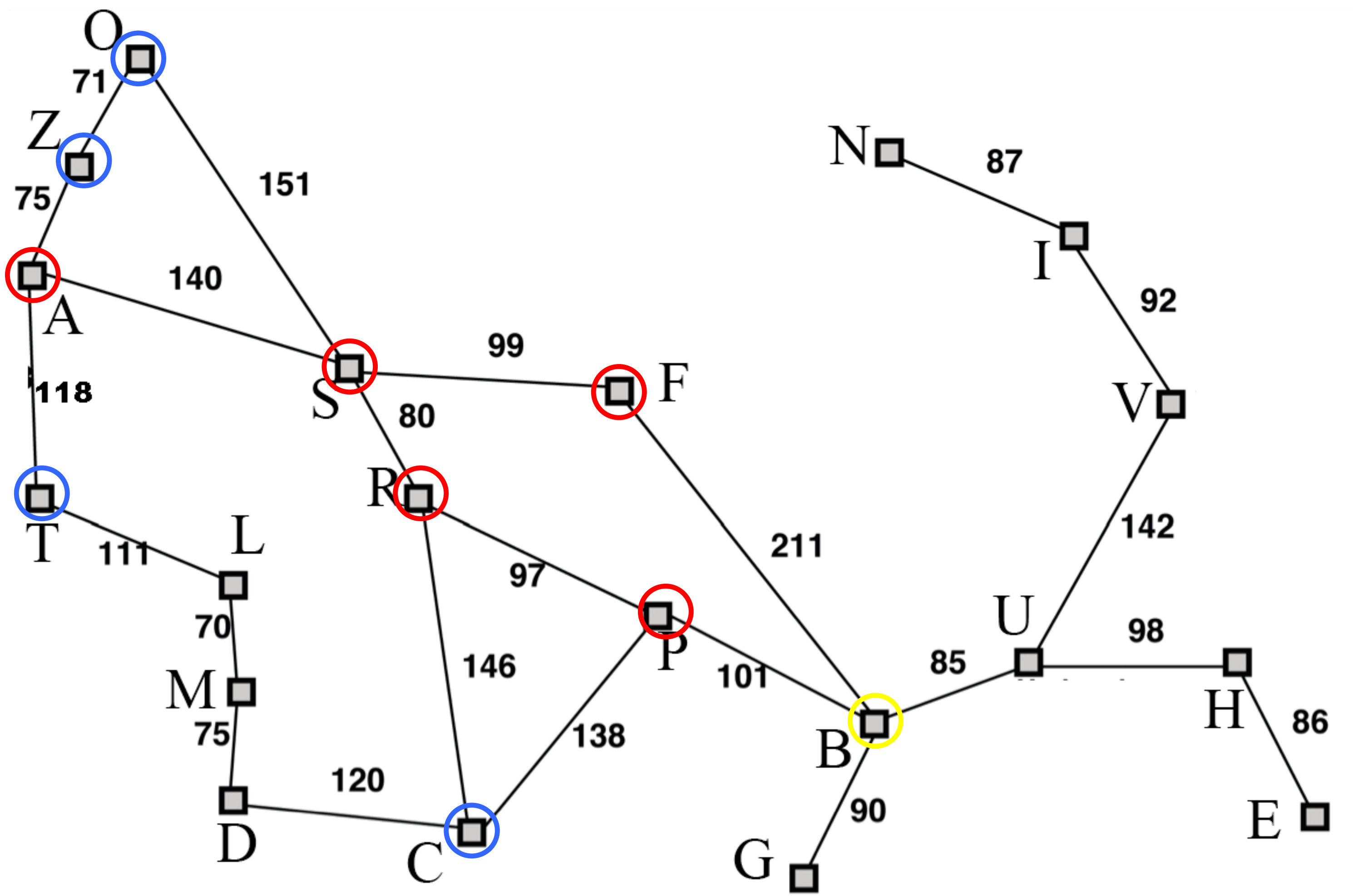


Open: B(418+0=418), T(329+118=447), Z(374+75=449), C(366+160=526), O(291+380=671)

Closed: P(417), F(415), R(413), S(393), A(366)

Evaluation function f(n) = g(n) + h(n)

A	366
B	0
C	160
D	242
E	161
F	176
G	77
H	151
I	226
L	244
M	241
N	234
O	380
P	100
S	253
T	329
U	80
V	199
Z	374



Solution: A-S-R-P-B

Open: T(329+118=447), Z(374+75=449), C(366+160=526), O(291+380=671)

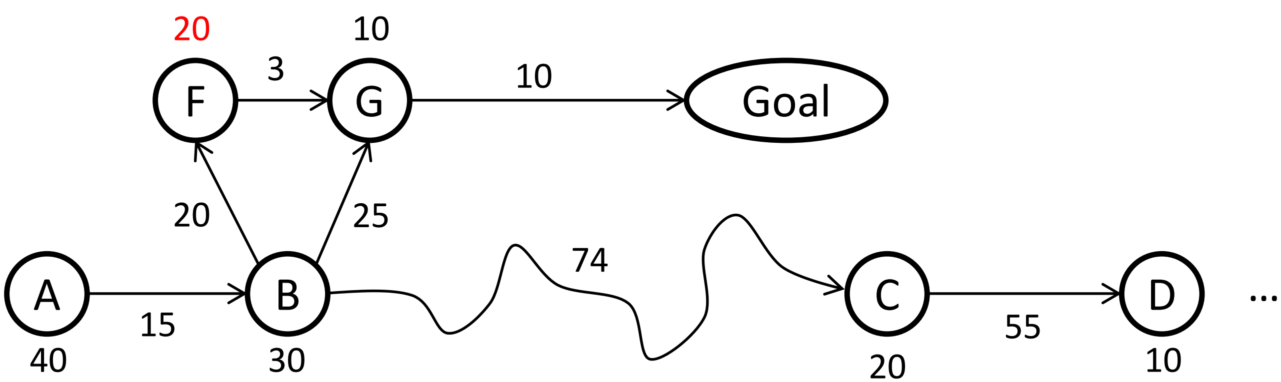
Closed: B(418), P(417), F(415), R(413), S(393), A(366)

A* Search

- A* is optimal…
- …but only if you use an **admissible** heuristic
- An admissible heuristic is mathematically guaranteed to underestimate the cost of reaching a goal
- What is an admissible heuristic for path finding on a path network?

Non-Admissible Heuristics

- What happens if you have a non-admissible heuristic?



Non-admissible heuristics

- Discourage agent from being in particular states
- Encourage agent from being in particular states

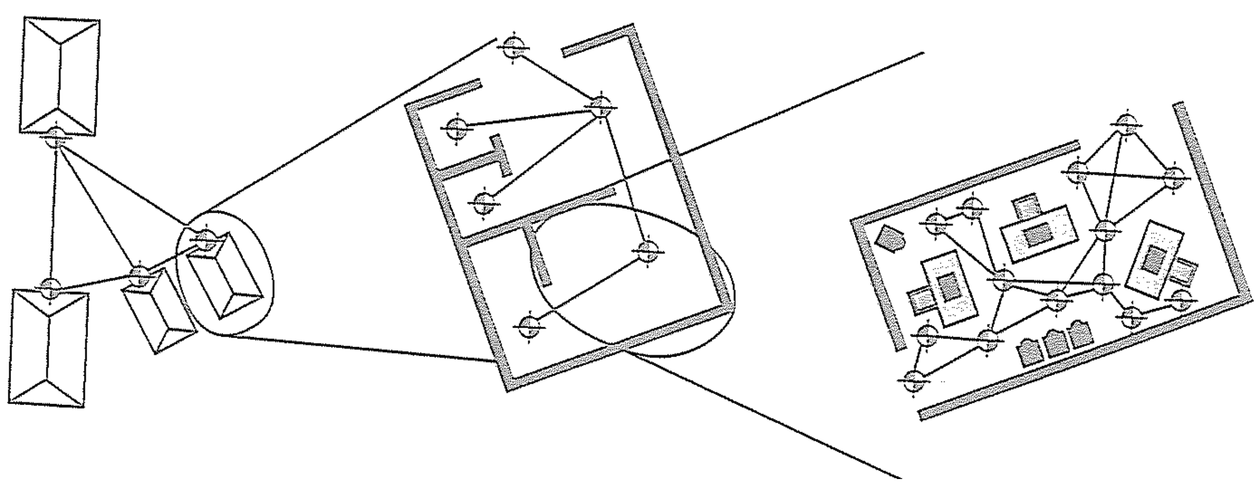
Hierarchical Path Planning

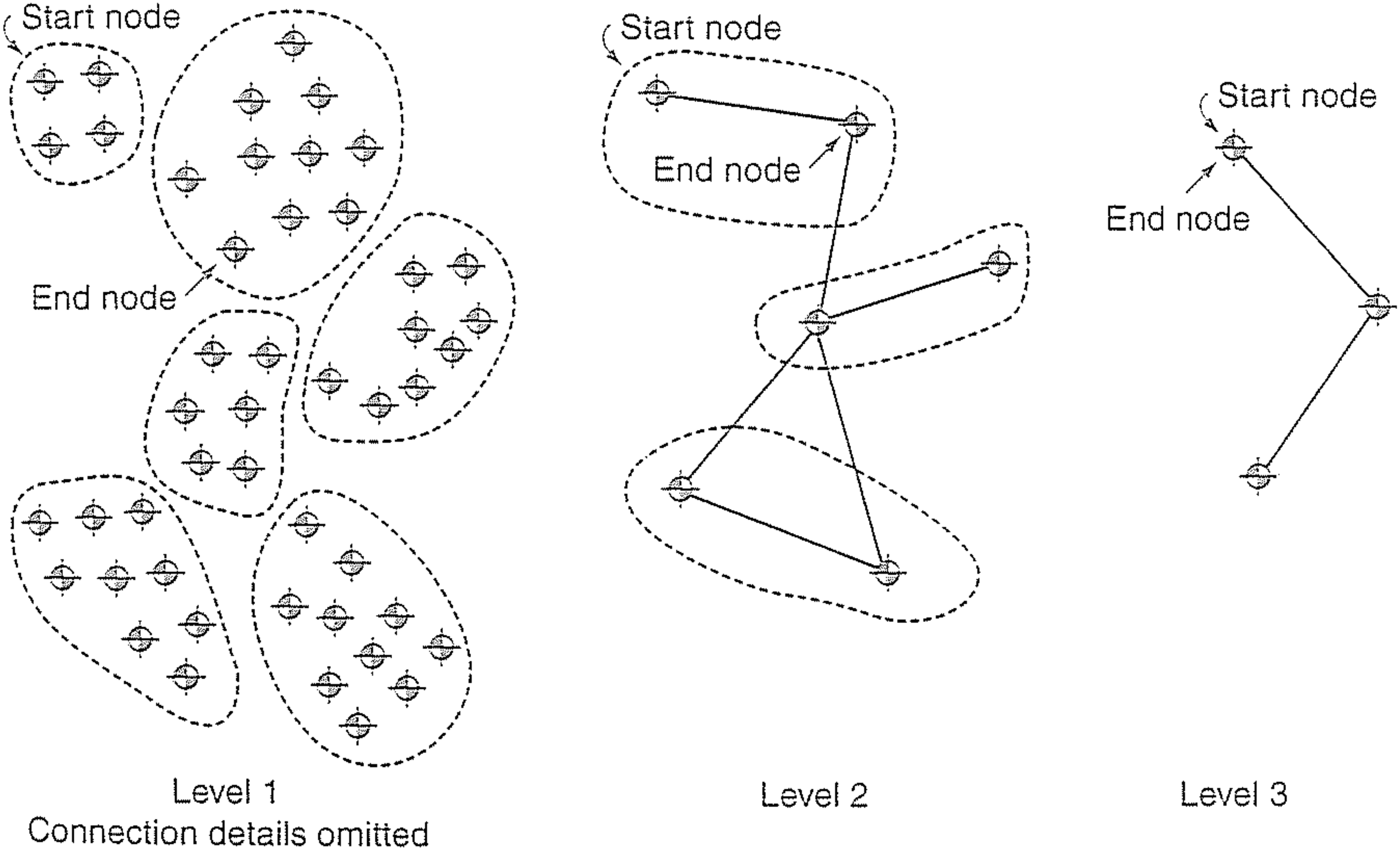
- Used to reduce CPU overhead of graph search
- Plan with coarse-grained and fine-grained maps
- People think hierarchically (more efficient)
- We can prune a large number of states

- Example: Planning a trip to NYC based on states, then individual roads

Hierarchical A*

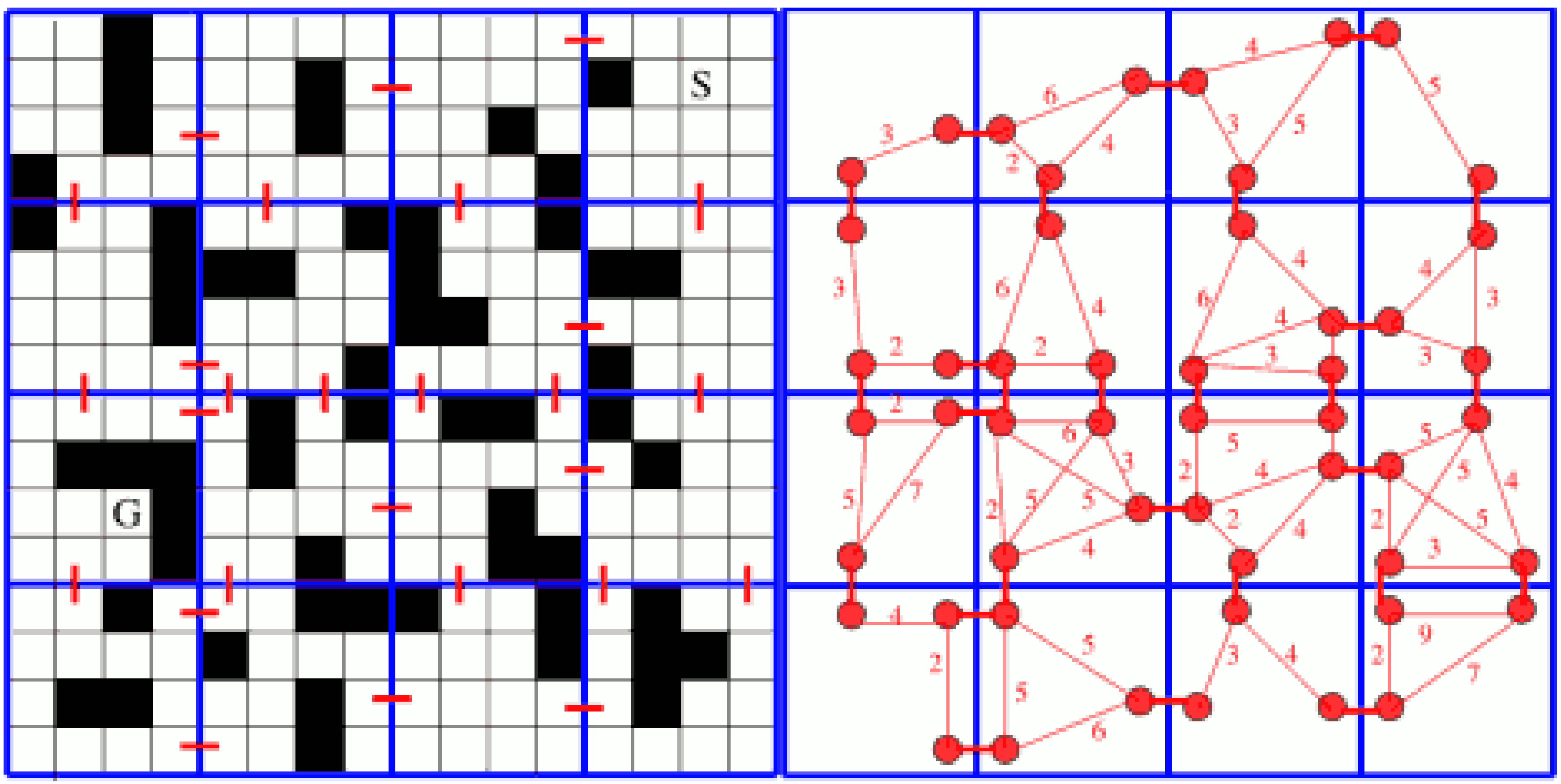
- http://www.cs.ualberta.ca/~mmueller/ps/hpastar.pdf
- Within 1% of optimal path length, but up to 10 times faster





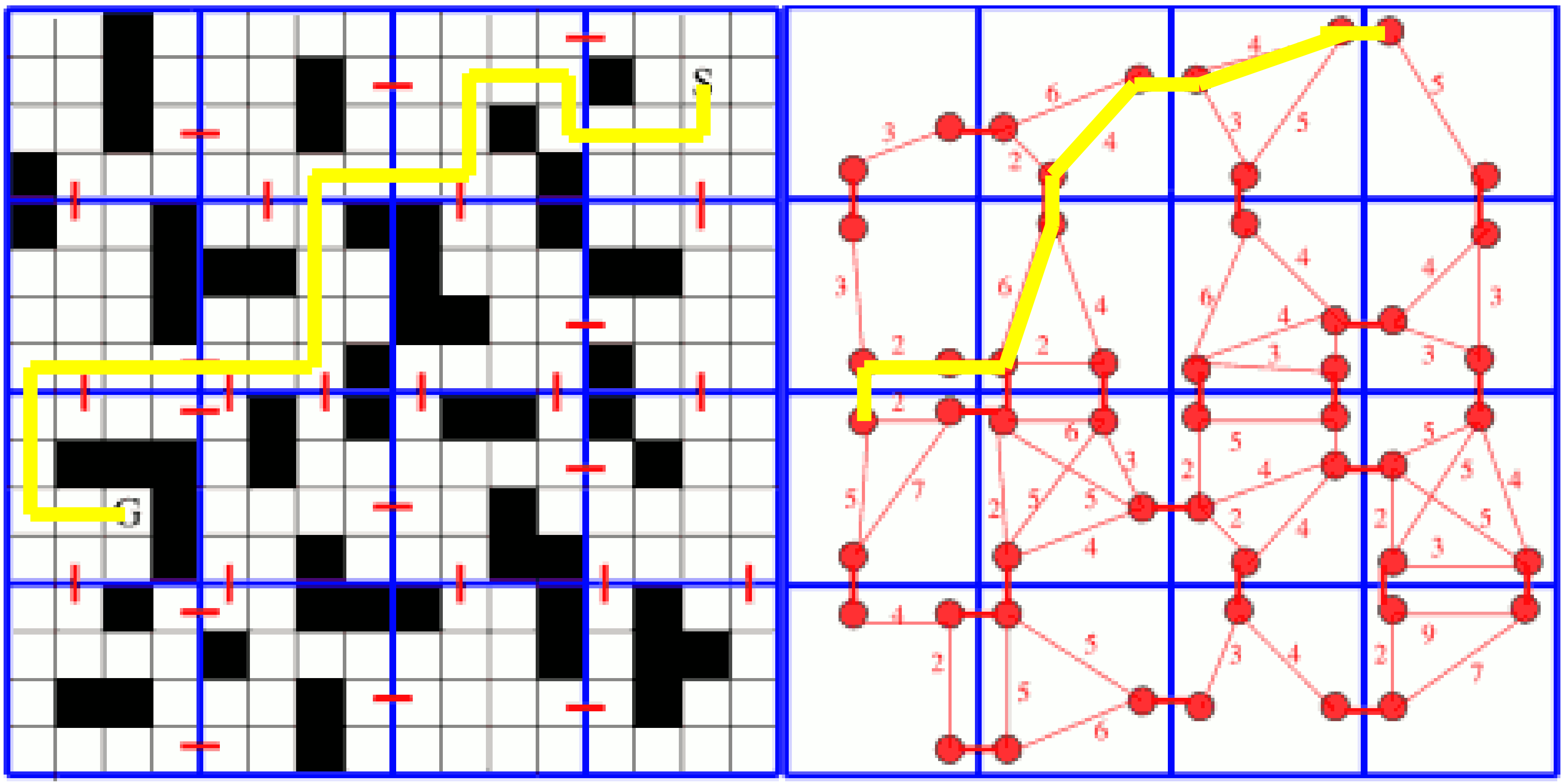
How high up do you go? As high as you can without start and end being in the same node.

1. Build clusters. Can be arbitrary
2. Find transitions, a (possibly empty) set of obstacle-free locations.
3. Inter-edges: Place a node on either side of transition, and link them (cost 1).
4. Intra-edges: Search between nodes inside cluster, record cost.
    * Can keep optimal intra-cluster paths, or discard for memory savings.

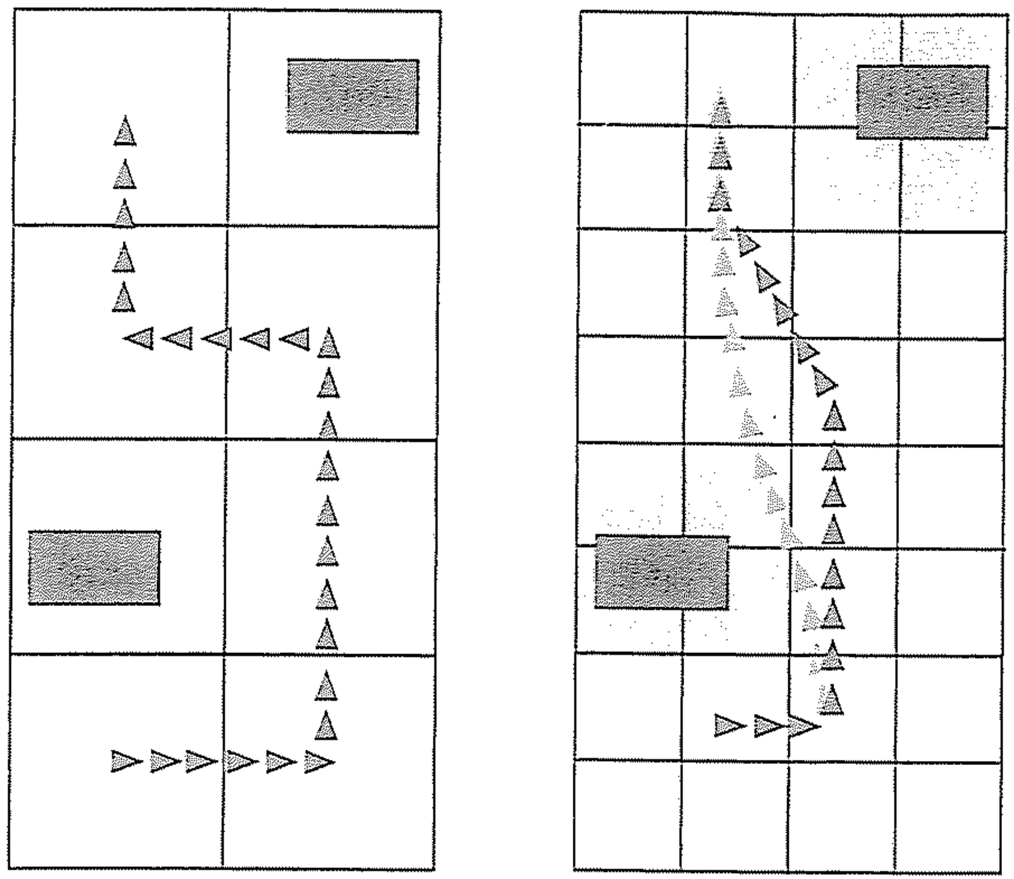


1. Start cluster: Search within cluster to the border
2. Search across clusters to the goal cluster
3. Goal cluster: Search from border to goal
4. Path smoothing

* Really just adds start and goal to the hierarchy graph



Path Smoothing in Hierarchical A*



High-level plan | Low-level plan

Sticky Situations

- Dynamic environments can ruin plans
- What do we do when an agent has been pushed back through a doorway that it has already "visited"?
- What do we do in "fog of war" situations?
- What if we have a moving target?

Real Time A*

- Online search: execute as you search
  - Because you can’t look at a state until you get there
  - You can’t backtrack
  - No open list
- Modified cost function f()
  - g(n) is actual distance from n to current state (instead of initial state)
- Use a hash-table to keep track of h() for nodes you have visited (because you might visit them again)
- Pick node with lowest f-value from immediate successors
- Execute move immediately
- After you move, update previous location
  - h(prev) = second best f-value
  - Second best f-value represents the estimated cost of returning to the previous state (and then add g)

RTA* with lookahead

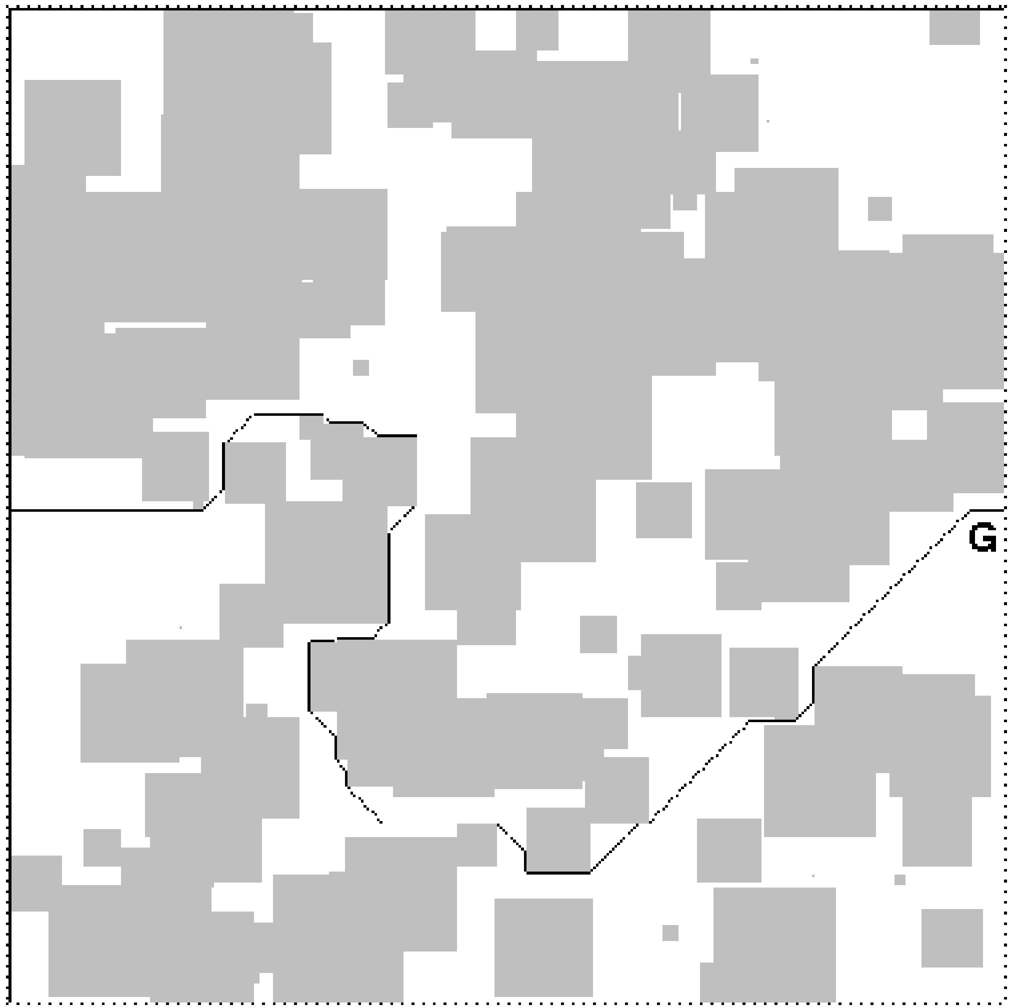
- At every node you can see some distance
- DFS, then back up the value (think of it as minimin with alpha-pruning)
- Search out to known limit
- Pick best, then move
- Repeat, because something might change in the environment that change our assessment
  - Things we discover as our horizon moves
  - Things that change behind us

D* Lite

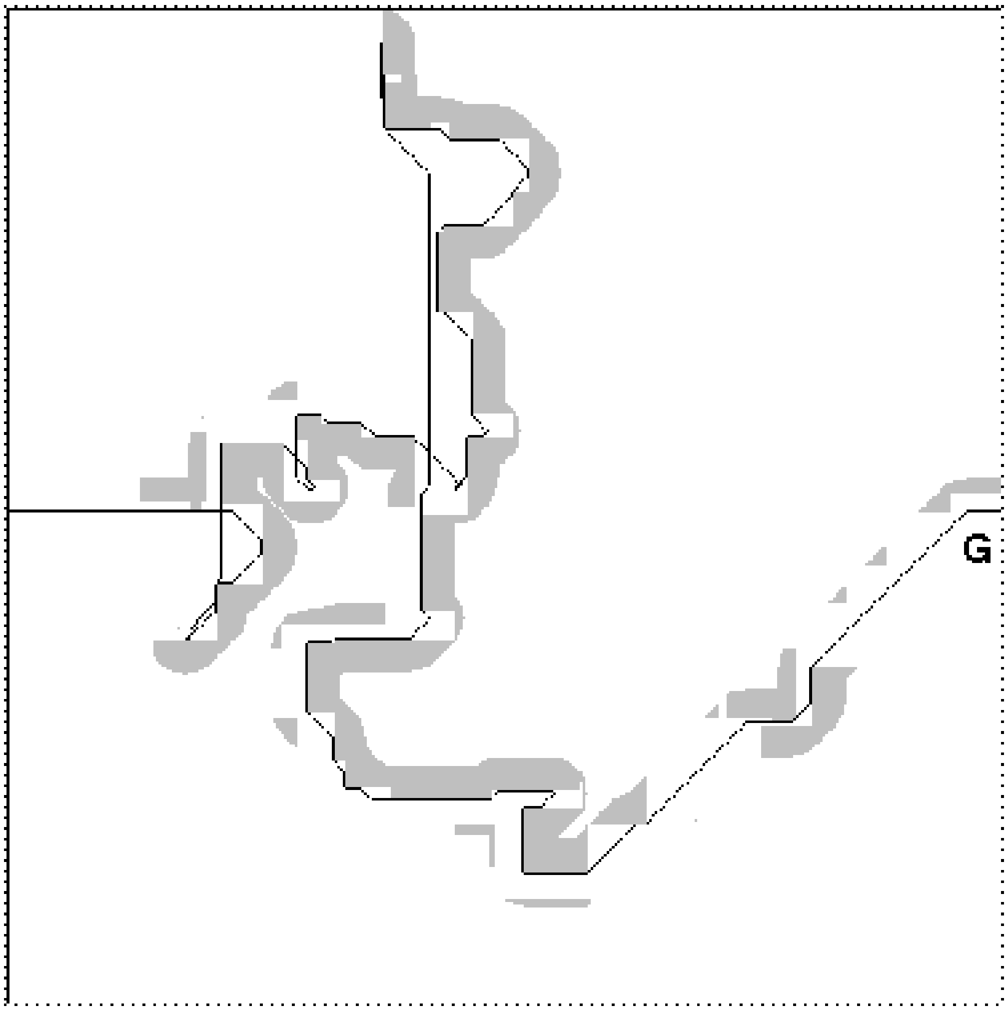
- Incremental search: replan often, but reuse search space if possible
- In unknown terrain, assume anything you don’t know is clear (optimistic)
- Perform A*, execute plan until discrepancy, then replan
- D* Lite achieves 2x speedup over A* (when replanning)

http://idm-lab.org/bib/abstracts/papers/aaai02b.pdf

“Omniscient optimal”: given complete information



“Optimistic optimal”: assume empty for parts you don’t know.



Heuristic Search Recap

- A*
  - Can't precompute
    - Dynamic environments
    - Memory issues
  - Optimal when heuristic is admissible (and assuming no changes)
  - Replanning can be slow on really big maps
- Hierarchical A* is the same, but faster

Heuristic Search Recap

- Real-time A*
  - Stumbling in the dark, 1 step lookahead
  - Replan every step, but fast!
  - Realistic? For a blind agent that knows nothing
  - Optimal when completely blind

Heuristic Search Recap

- Real-time A* with lookahead
  - Good for fog-of-war
  - Replan every step, with fast bounded lookahead to edge of known space
  - Optimality depends on lookahead

Heuristic Search Recap

- D* Lite
  - Assume everything is open/clear
  - Replan when necessary
  - **Worst case: Runs like Real-Time A***
  - Best case: Never replan
  - Optimal including changes

See also

- AI Game Programming wisdom 2, CH 2
- Buckland CH 8
- Millington CH 4

Next time…

(KINEMATIC) MOVEMENT, STEERING, FLOCKING, FORMATIONS