Motion planning: Beyond Navmeshes

CMSC425.01 Fall 2019

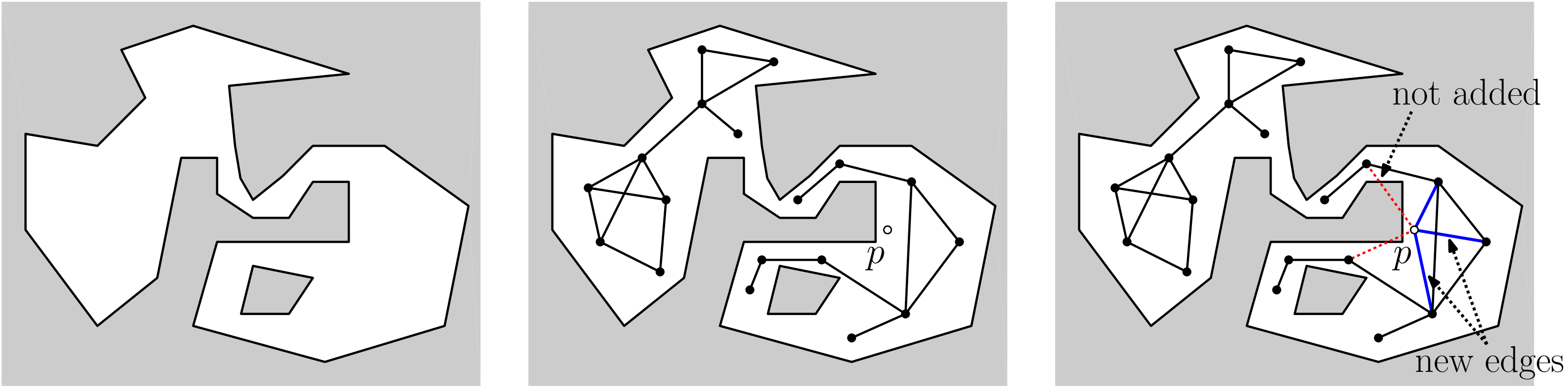
Today’s questions

Big question: Making intelligent agents
First question: Navigation

Finding paths in polygonal configuration space

- Version 1: Navmesh
- Others?

- Version 7: Randomized placement (sampling)

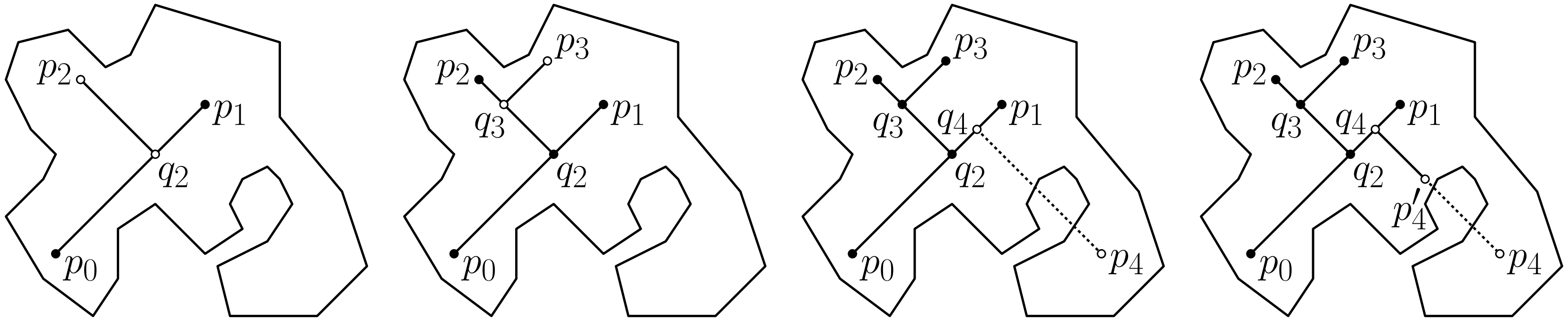


(a) (b) (c)

Finding paths in polygonal configuration space

- Version 1: Navmesh
- Others?

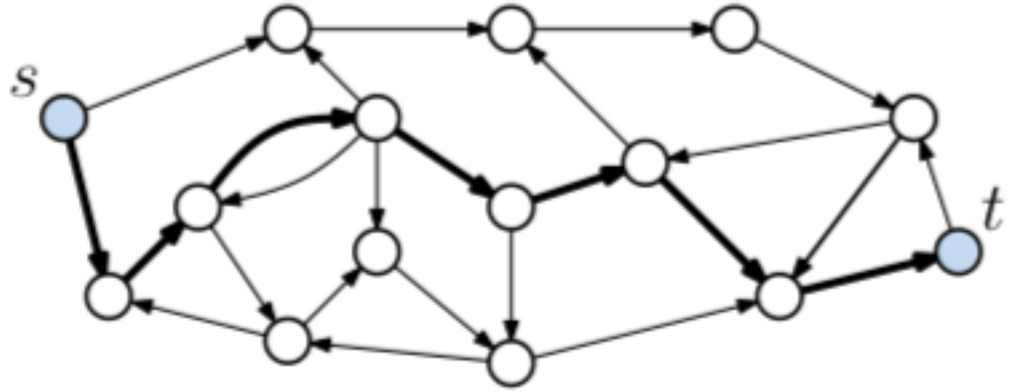
- Version 8: Rapidly-expanded Random Trees (RRTs)



(a) (b) (c) (d)

Computing shortest path

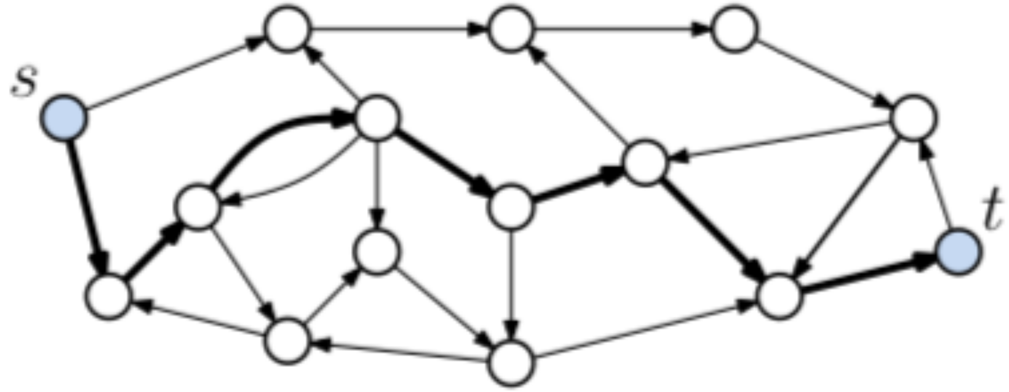
- Reduce navigation to path finding in graphs
  - Directed?
  - Weighted?



- $G = (V, E)$
  - Vertices $V = \{ u, v, \ldots \}$
  - Edges $E = \{ (u, v), \ldots \}$
  - Weight function $\mathrm{w}(u, v) \rightarrow reals$

Computing shortest path

- Reduce navigation to path finding in graphs
  - Directed?
  - Weighted?



- $G = (V, E)$
  - Vertices $V = \{ u, v, \dots \}$
  - Edges $E = \{ (u, v), \dots \}$
  - Weight function $\mathrm{w}(u, v) \rightarrow reals$

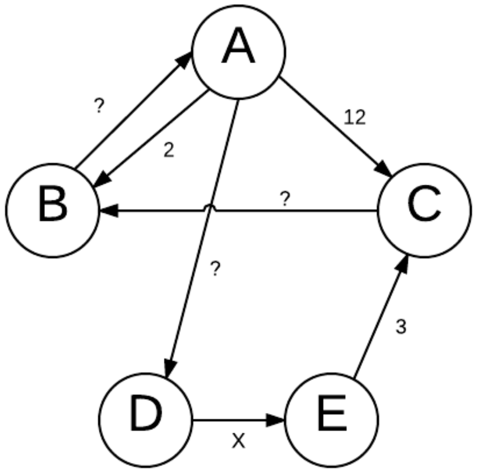
- Path sequence of nodes
  - $P = \langle u_{0,} u_{1,} \dots , u_k \rangle$
- Path cost
  - $cost(P) = \sum_{i=0}^{k} w(u_{i,} u_{i+1})$
- Lowest cost path $\partial(s, t)$

First: what's the problem?

- Compute one shortest path?
- Compute all shortest paths to store?

First: what's the problem?

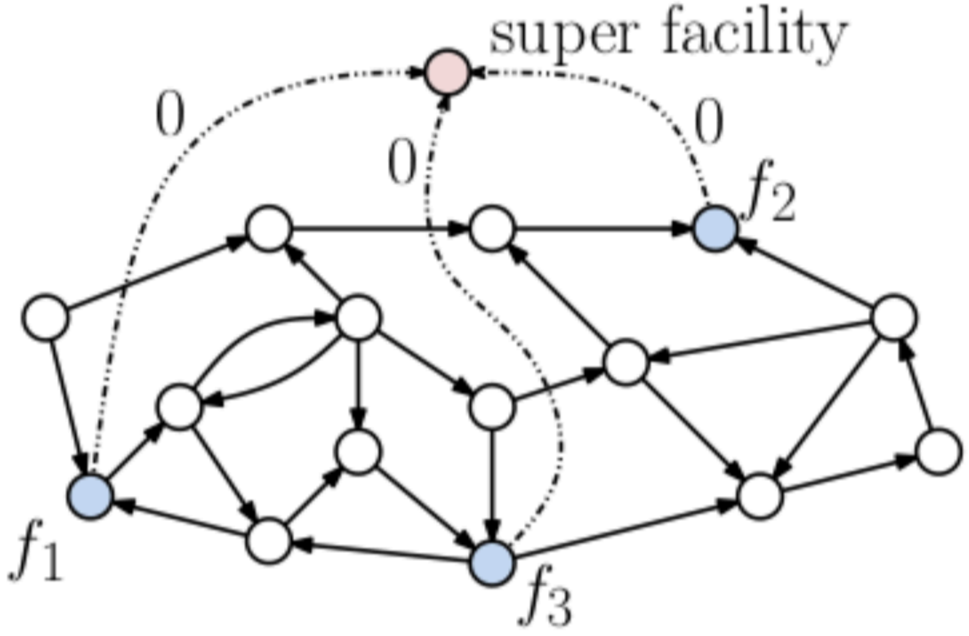
- Compute path here to there?
- Find fastest way to home base?
  - Reverse edges
  - Find shortest path to all from home
- Find closest facility (health, etc)?
  - Add Supernode connected to all facilities.
- Compute all shortest paths to store?
  - Floyd-Warshall



	A	B	C	D	E
A	0	2	10	5	7
B	3	0	13	8	10
C	7	4	0	12	14
D	12	9	5	0	X
E	10	7	3	15	0

First: what's the problem?

- Find closest facility (health, etc)?
  - Add Supernode connected to all facilities.

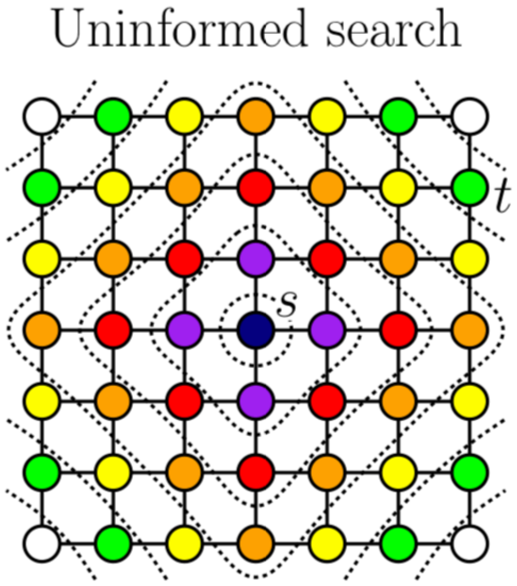


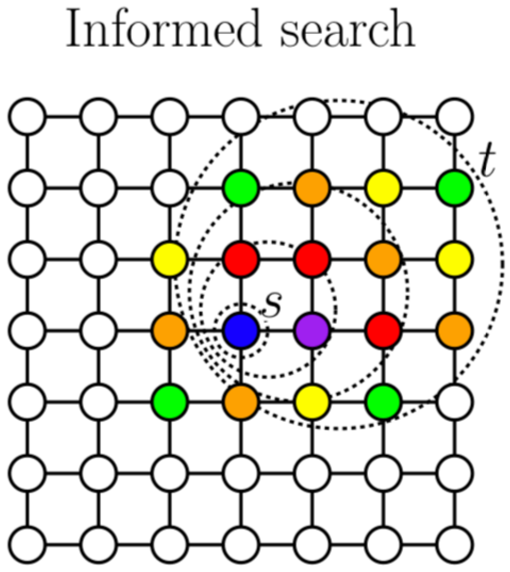
Uninformed vs. informed search

- Uninformed – follow weights
  - Pick next node on distance to d[u]

- Informed – add bias towards destination

- Heuristic
  - Pick next node on distance to goal h(u)

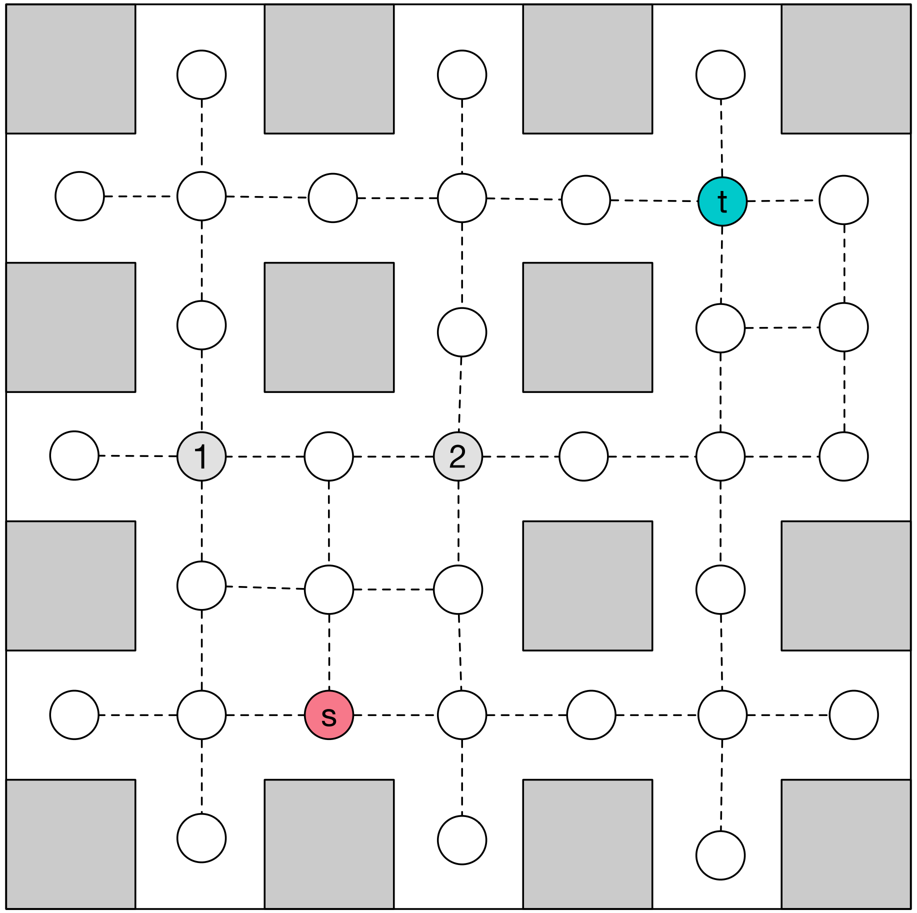




Informed search

- Distance functions
  - w(u,v) - distance node u to v
  - d[u] - distance traversed from start to node u
  - dist(u,t) - distance from u to t

- w(s,1) = ___ dist(1,t) = ___
- w(s,2) = ___ dist(1,t) = ___

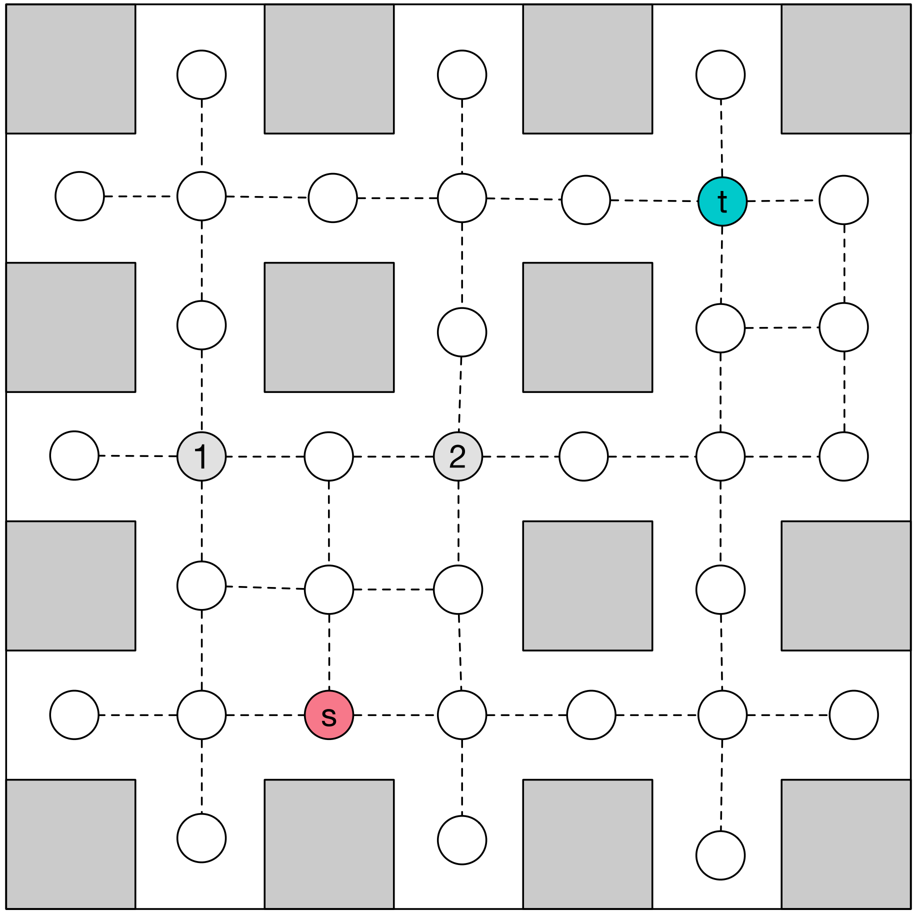


Informed search

- Distance functions
  - w(u,v)   - distance node u to v
  - d[u]   - distance traversed from start to node u
  - dist(u,t) - distance from u to t

- w(s,1) = 3   dist(1,t) = 6
- w(s,2) = 3   dist(1,t) = 4

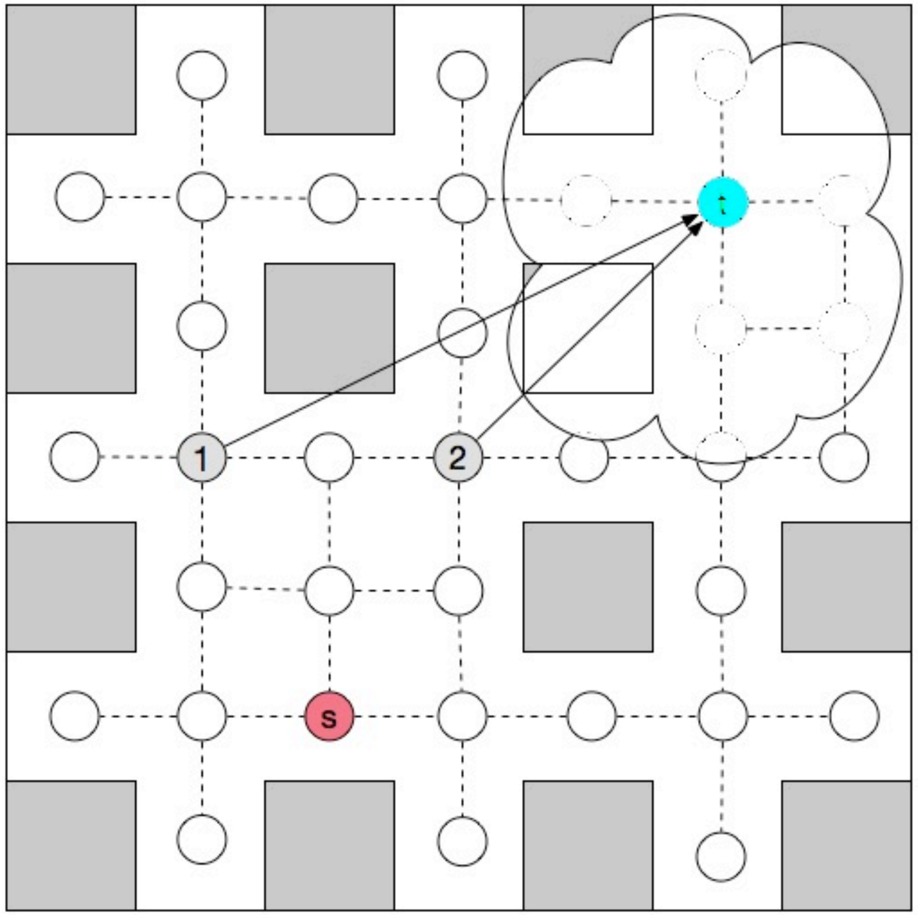
- dist(u,t) is a *heuristic*



Less perfect information?

- Can't see rest of graph until you expand it
- Need guess on what's to come

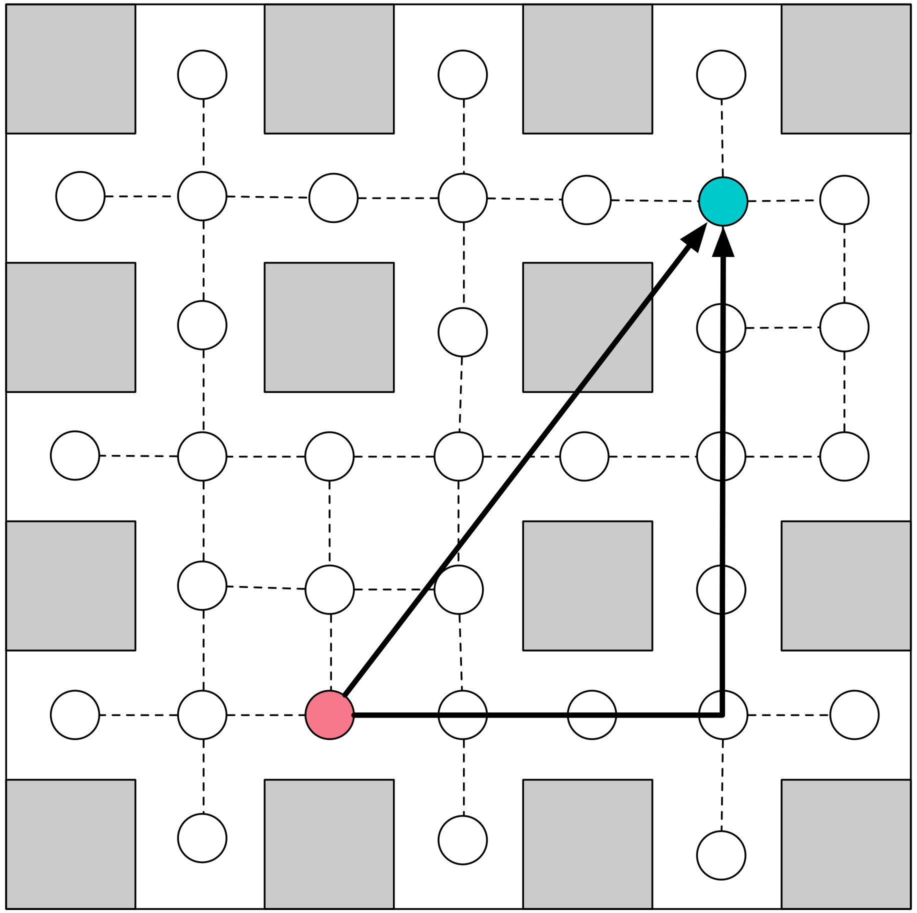
- dist(u,t) as Euclidean distance
- Approximates actual cost



Footnote

- Euclidean distance
  - distE(p1,p2) =
    sqrt((x1-x2)^2 + (y1-y2)^2)

- Manhattan distance
  - distM(p1,p2) =
    abs(x1-x2) + abs(y1-y2)



```
Dijkstra(G, s, t) {
  foreach (node u) {                            // initialize
    d[u] = +infinity;  mark u undiscovered
  }
  d[s] = 0;  mark s discovered                  // distance to source is 0
  repeat forever {                              // go until finding t
    let u be the discovered node that minimizes d[u]
    if (u == t) return d[t]                     // arrived at the destination
    else {
      for (each unfinished node v adjacent to u) {
        d[v] = min(d[v], d[u] + w(u,v))         // update d[v]
        mark v discovered
      }
      mark u finished                           // we're done with u
  }
}
```

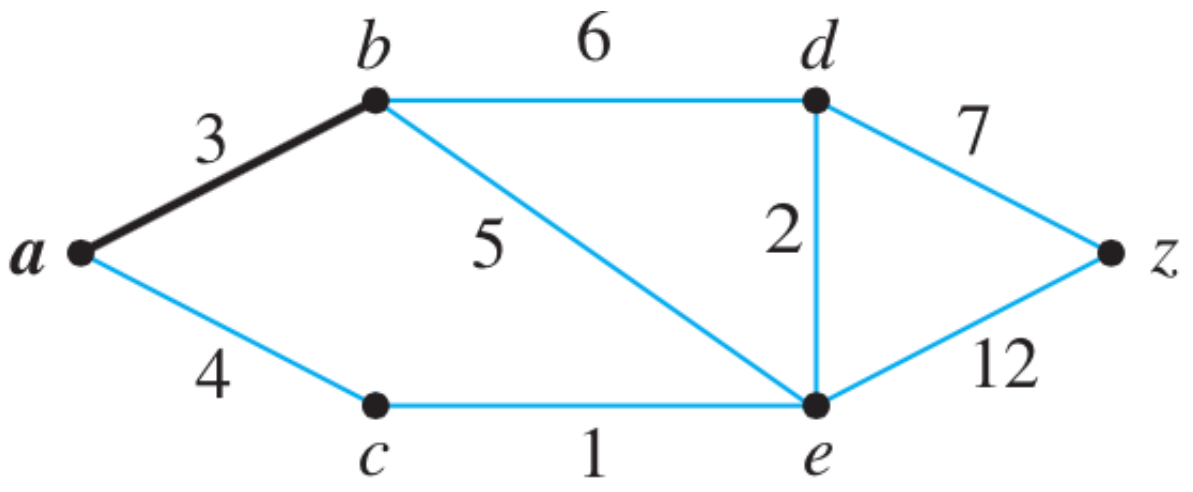
Example

- w(u,v) as given
- Start with d array as

a	b	c	d	e	z
0	INF	INF	INF	INF	INF

- End with?

a	b	c	d	e	z
0					



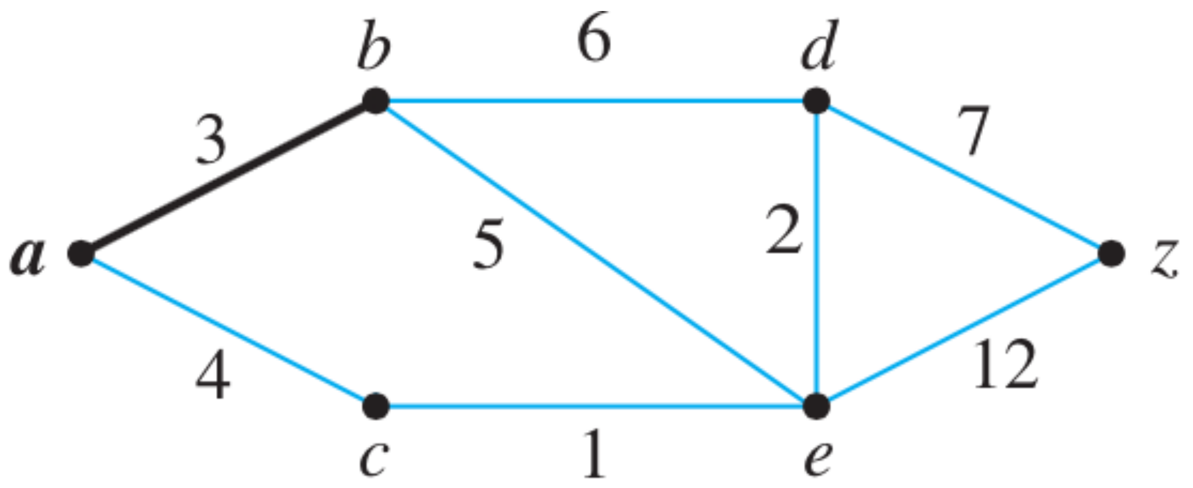
Example

- w(u,v) as given
- Start with d array as

a	b	c	d	e	z
0	INF	INF	INF	INF	INF

- End with?

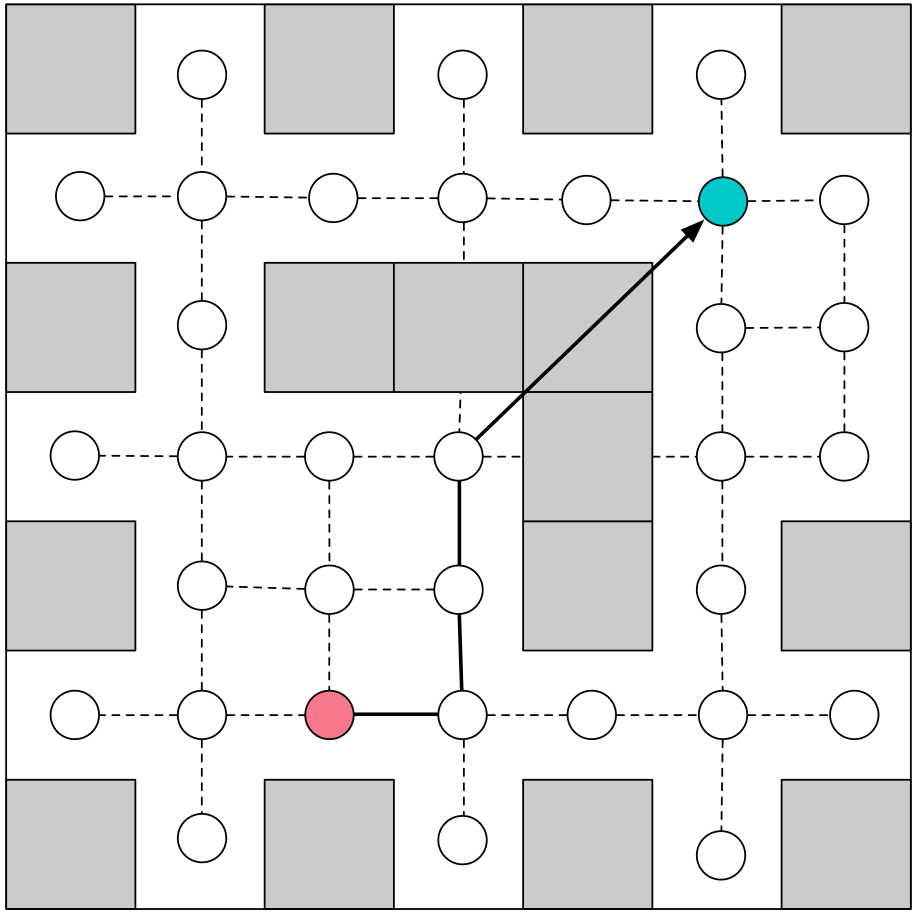
a	b	c	d	e	z
0	3	4	7	5	14



```
BestFirst(G, s, t) {
  foreach (node u) {                          // initialize
    d[u] = +infinity;  mark u undiscovered
  }
  d[s] = 0;  mark s discovered                // distance to source is 0
  repeat forever {                            // go until finding t
    let u be the discovered node that minimizes dist(u,t)
    if (u == t) return d[t]                   // arrived at the destination
    else {
      for (each unfinished node v adjacent to u) {
        d[v] = min(d[v], d[u] + w(u,v))  // update d[v]
        mark v discovered
      }
      mark u finished                         // we're done with u
  }
}
```

Best first bad case …

- Trapped in local minimum



A*

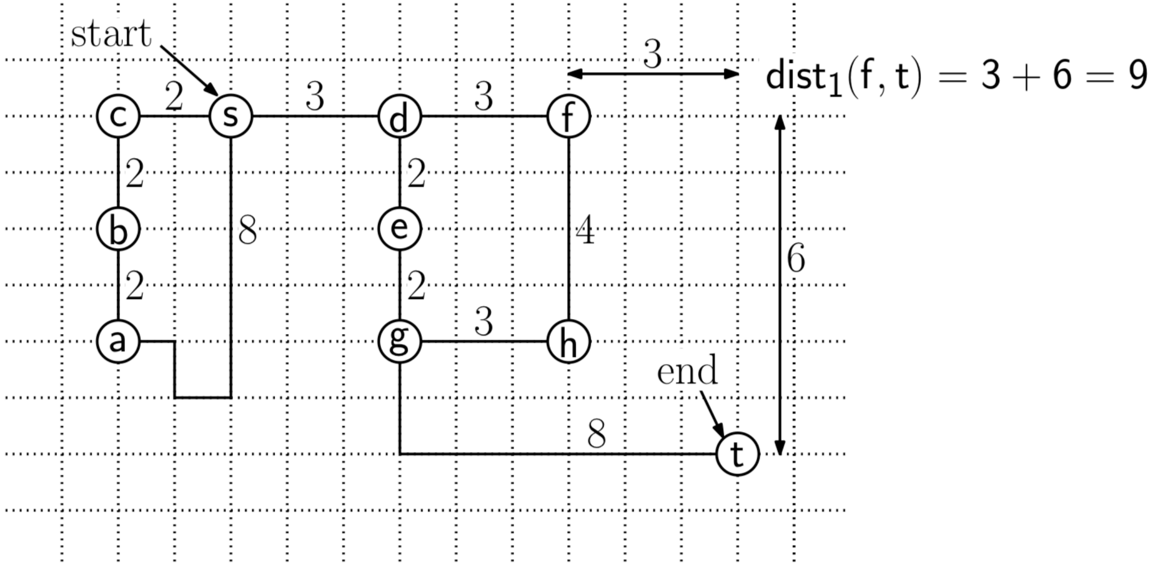
- Pick next node to expand based on sum of distance so far *and* heuristic

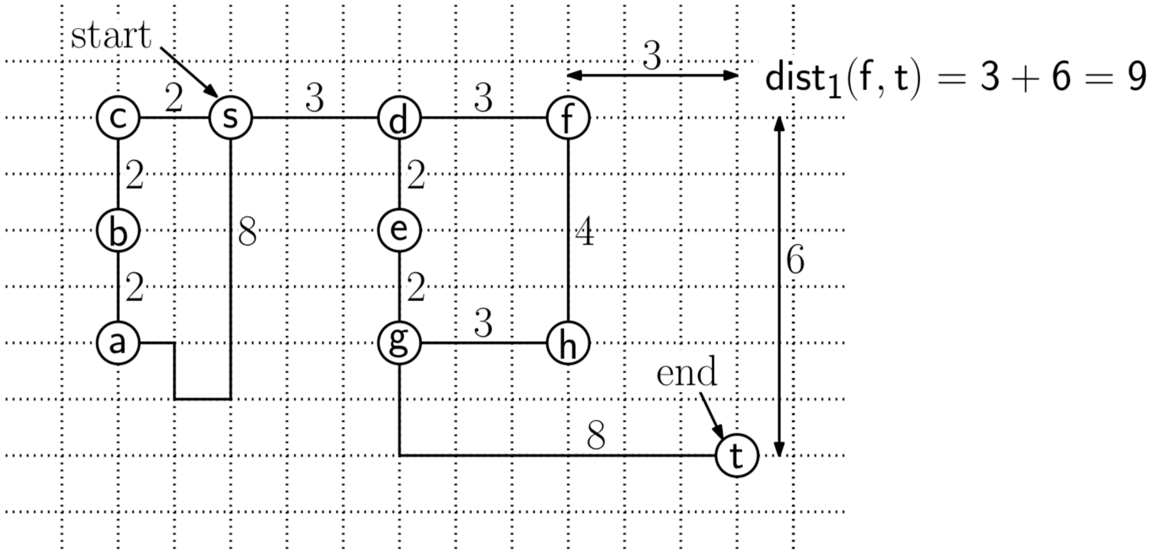
$$f(u) \;=\; d[u] + h(u) \;=\; d[u] + \mathrm{dist}(u,t)$$

```
A-Star(G, s, t) {
  foreach (node u) {                          // initialize
    d[u] = +infinity;  mark u undiscovered
  }
  d[s] = 0;  mark s discovered                // distance to source is 0
  repeat forever {                            // go until finding t
    let u be the discovered node that minimizes d[u] + dist(u,t)
    if (u == t) return d[t]                   // arrived at the destination
    else {
      for (each unfinished node v adjacent to u) {
        d[v] = min(d[v], d[u] + w(u,v))       // update d[v]
        mark v discovered
      }
      mark u finished                         // we're done with u
  }
}
```

A* Example

- Manhattan distance





A* Search – Each entry is $d[u] : f(u)$										
Stage	$d[s]$	$d[a]$	$d[b]$	$d[c]$	$d[d]$	$d[e]$	$d[f]$	$d[g]$	$d[h]$	$d[t]$
$h(u)$	15	13	15	17	12	10	9	8	5	0
Init	0:15	∞:13	∞:15	∞:17	∞:12	∞:10	∞:9	∞:8	∞:5	∞:0
1: s	**0**	8:13	–	2:17	3:12	–	–	–	–	–
2: d	↓	8:13	–	2:17	**3**	5:10	6:9	–	–	–
3: e		8:13	–	2:17	↓	**5**	6:9	7:8	–	–
4: f		8:13	–	2:17		↓	**6**	7:8	–	15:0
5: t		8:13	–	2:17			↓	7:8	–	**15**
Final	0	8	∞	2	3	5	6	7	∞	15

Good heuristics

- For A* to compute correctly the heuristic h(u) must be:

- Admissible: h(u) never overestimates the graph distance from node u to goal t

- Consistent: h(u') <= delta(u',u'') + h(u'')