

Artificial Intelligence Programming

Heuristic Search

Cindi Thompson

Department of Computer Science
University of San Francisco

Review

Uninformed search in simple environments

- static
- observable
- discrete (and finite)
- single-agent
- deterministic

But sequential.

Review - uninformed search

Offline search to find the sequence of actions leading to a goal

- Breadth-first search
- Uniform cost search
- Depth-first search
- Depth-limited search
- Iterative deepening DFS

Review - Problem characteristics

- Initial state
- Goal state test
- Actions
- Successor function
- Path cost
- Solution

Plan for the Day

- Heuristic Search - exploiting knowledge about the problem
- Heuristic Search Algorithms
 - “Best-first” search
 - Greedy Search
 - A* Search
 - Extensions to A*
- Constructing Heuristics

Informing Search

- Previous algorithms were able to find solutions, but were very inefficient.
 - Exponential number of nodes expanded.
- By taking advantage of knowledge about the problem structure, we can improve performance.
- Two caveats:
 - We have to get knowledge about the problem from somewhere.
 - This knowledge has to be correct.

Best-first Search

- Recall Uniform-cost search
 - Nodes were expanded based on their total path cost
 - A priority queue was used to implement this.
- Path cost is an example of an *evaluation function*.
 - We'll use the notation $f(n)$ to refer to an evaluation function.
- An evaluation function tells us how promising a node is.
- Indicates the quality of the solution that node leads to.

Best-first Search

- By ordering and expanding nodes according to their f value, we search the “best” nodes “first”
- If f was perfect, we would expand a straight path from the initial state to the goal state.
 - Arad, Sibiu, Rimnicu Vilcea, Pitesti, Bucharest
- Of course, if we had a perfect f , we wouldn't need to solve the problem in the first place.
- Instead, we'll try to develop heuristic functions $h(n)$ that help us estimate $f(n)$.

Heuristic Functions

- $h(n)$ is an estimate of how much it might cost to get to the solution from node n
- Example for route planning?
- How could we use a heuristic function as part of Best-first search to find a goal quickly?

Best-first Search

● Best-first Pseudocode

```
enqueue(initialState)
do
  node = dequeue()
  if goalTest(node)
    return node
  else
    children = successor-fn(node)
    for child in children
      insert-with(child, f(child))
```

● where insert-with orders our priority queue according to f .

Best-first search

- (Almost) all searches are instances of best-first, with different evaluation functions f
- What functions f would yield the following searches:
 - DFS
 - Breadth-first search
 - Uniform cost search

Best-first search

- (Almost) all searches are instances of best-first, with different evaluation functions f
- What functions f would yield the following searches:
 - DFS: $f(n) = -depth(n)$
 - Breadth-first search: $f(n) = depth(n)$
 - Uniform cost search: $f(n) = \text{cost of path to } n$

Greedy Search

- Let's start with the opposite of uniform-cost search
- UCS used the solution cost to date as an estimate of f
- *Greedy search* uses an estimate of distance to the goal for f .
- Rationale: Always pick the node that looks like it will get you closest to the solution.
- Let's start with a simple estimate of f for the Romania domain.
 - $h(city)$ = Straight-line distance between that city and Bucharest.

SLD Heuristic

- Notice that there wasn't anything in the problem description about straight-line distance or the fact that that would be a useful estimate.
- We used our knowledge about the way roads generally work.
- This is sometimes called *common sense* knowledge.

Greedy Search Example

Arad	366	Mehadia	241
Bucharest	0	Neamt	234
Craiova	160	Oradea	380
Drobeta	242	Pitesti	100
Eforie	161	Rimnicu Vilcea	193
Fagaras	176	Sibiu	253
Giurgiu	77	Timisoara	329
Hirsova	151	Urziceni	80
Iasi	226	Vaslui	199
Lugoj	244	Zerind	374

- A = 336
- (dequeue A) S = 253, T = 329, Z = 374
- (dequeue S) F = 176, RV = 193, T = 329, A = 336, Z = 374, O = 380
- (dequeue F) B = 0, RV = 193, S = 253, T = 329, A = 336, Z = 374, O = 380
- dequeue B - solution: A → S → F → B

- We found a solution quickly, but it was not optimal.
- What was the problem with our approach?

Issues with Greedy Search

- Sometimes the optimal solution to a problem involves moving 'away' from the goal.
- For example, to solve 8-puzzle, you often need to 'undo' a partial solution.
- Greedy search has many of the same appeals and weaknesses as DFS.
 - Expands a linear number of nodes
 - Is not complete or optimal
- Its ability to cut toward a goal is appealing - can we salvage this?

A* search

- A* search is a combination of uniform cost search and greedy search.
- A node's $f(n) = g(n) + h(n)$
 - $g(n)$ = current path cost
 - $h(n)$ = heuristic estimate of distance to goal.
- Favors nodes that have a cheap solution to date and also look like they'll get close to the goal.
- If $h(n)$ satisfies certain conditions, A* is both complete (always finds a solution) and optimal (always finds the best solution).

A* example - Romania

● h = straight-line distance

Arad	366	Mehadia	241
Bucharest	0	Neamt	234
Craiova	160	Oradea	380
Drobeta	242	Pitesti	100
Eforie	161	Rimnicu Vilcea	193
Fagaras	176	Sibiu	253
Giurgiu	77	Timisoara	329
Hirsova	151	Urziceni	80
Iasi	226	Vaslui	199
Lugoj	244	Zerind	374

● Arad = 0 + 366 = 366

● (dequeue A: $g = 0$) $S = 140 + 253 = 393$, $T = 118 + 329 = 447$, $Z = 75 + 374 = 449$

● (dequeue S: $g = 140$) $RV = 220 + 193 = 413$, $F = 239 + 176 = 415$, $T = 118 + 329 = 447$, $Z = 374 + 75 = 449$, $A = 280 + 336 = 616$, $O = 291 + 380 = 671$,

● (dequeue RV: $g = 220$) $F = 239 + 176 = 415$, $P = 317 + 100 = 417$, $T = 118 + 329 = 447$, $Z = 374 + 75 = 449$, $C = 366 + 160 = 526$, $S = 300 + 253 = 553$, $A = 280 + 336 = 616$, $O = 291 + 380 = 671$

● (dequeue F: $g = 239$) $P = 317 + 100 = 417$, $T = 118 + 329 = 447$, $Z = 374 + 75 = 449$, $C = 366 + 160 = 526$, $B = 550 + 0 = 550$, $S = 300 + 253 = 553$, $S = 338 + 253 = 591$, $A = 280 + 336 = 616$, $O = 291 + 380 = 671$

A* example - Romania

Arad	366	Mehadia	241
Bucharest	0	Neamt	234
Craiova	160	Oradea	380
Drobeta	242	Pitesti	100
Eforie	161	Rimnicu Vilcea	193
Fagaras	176	Sibiu	253
Giurgiu	77	Timisoara	329
Hirsova	151	Urziceni	80
Iasi	226	Vaslui	199
Lugoj	244	Zerind	374


• (dequeue P: $g = 317$) $T = 118 + 329 = 447$, $Z = 374 + 75 = 449$, $B = 518 + 0 = 518$, $C = 366 + 160 = 526$, $B = 550 + 0 = 550$, $S = 300 + 253 = 553$, $S = 338 + 253 = 591$, $RV = 414 + 193 = 607$, $C = 455 + 160 = 615$, $A = 280 + 336 = 616$, $O = 291 + 380 = 671$


• (dequeue T: $g = 118$) $Z = 374 + 75 = 449$, $L = 229 + 244 = 473$, $B = 518 + 0 = 518$, $C = 366 + 160 = 526$, $B = 550 + 0 = 550$, $S = 300 + 253 = 553$, $A = 236 + 336 = 572$, $S = 338 + 253 = 591$, $RV = 414 + 193 = 607$, $C = 455 + 160 = 615$, $A = 280 + 336 = 616$, $O = 291 + 380 = 671$


• (dequeue Z: $g = 75$) $L = 229 + 244 = 473$, $A = 150 + 336 = 486$, $B = 518 + 0 = 518$, $O = 146 + 380 = 526$, $C = 366 + 160 = 526$, $B = 550 + 0 = 550$, $S = 300 + 253 = 553$, $A = 236 + 336 = 572$, $S = 338 + 253 = 591$, $RV = 414 + 193 = 607$, $C = 455 + 160 = 615$, $A = 280 + 336 = 616$, $O = 291 + 380 = 671$

A* example - Romania

Arad	366	Mehadia	241
Bucharest	0	Neamt	234
Craiova	160	Oradea	380
Drobeta	242	Pitesti	100
Eforie	161	Rimnicu Vilcea	193
Fagaras	176	Sibiu	253
Giurgiu	77	Timisoara	329
Hirsova	151	Urziceni	80
Iasi	226	Vaslui	199
Lugoj	244	Zerind	374

 (dequeue L: $g = 229$) $A = 150 + 336 = 486$, $B = 518 + 0 = 518$, $O = 146 + 380 = 526$, $C = 366 + 160 = 526$, **$M = 299 + 241 = 540$** , $B = 550 + 0 = 550$, $S = 300 + 253 = 553$, $A = 236 + 336 = 572$, $S = 338 + 253 = 591$, $RV = 414 + 193 = 607$, $C = 455 + 160 = 615$, $A = 280 + 336 = 616$, **$T = 340 + 329 = 669$** , $O = 291 + 380 = 671$

 (dequeue A: $g = 150$) $B = 518 + 0 = 518$, $O = 146 + 380 = 526$, $C = 366 + 160 = 526$, $M = 299 + 241 = 540$, **$S = 290 + 253 = 543$** , $B = 550 + 0 = 550$, $S = 300 + 253 = 553$, $A = 236 + 336 = 572$, $S = 338 + 253 = 591$, **$T = 268 + 329 = 597$** , **$Z = 225 + 374 = 599$** , $RV = 414 + 193 = 607$, $C = 455 + 160 = 615$, $A = 280 + 336 = 616$, $T = 340 + 329 = 669$, $O = 291 + 380 = 671$

 (dequeue B: $g = 518$) solution. $A \rightarrow S \rightarrow RV \rightarrow P \rightarrow B$

Optimality of A*

- A* is optimal (finds the shortest solution) as long as our h function is *admissible*.
 - Admissible: always underestimates the cost to the goal.
- Proof: When we dequeue a goal state, we see $g(n)$, the actual cost to reach the goal. If h underestimates, then a more optimal solution would have had a smaller $g + h$ than the current goal, and thus have already been dequeued.
- Or: If h overestimates the cost to the goal, it's possible for a good solution to “look bad” and get buried further back in the queue.
- In our Romania example, SLD always underestimates.

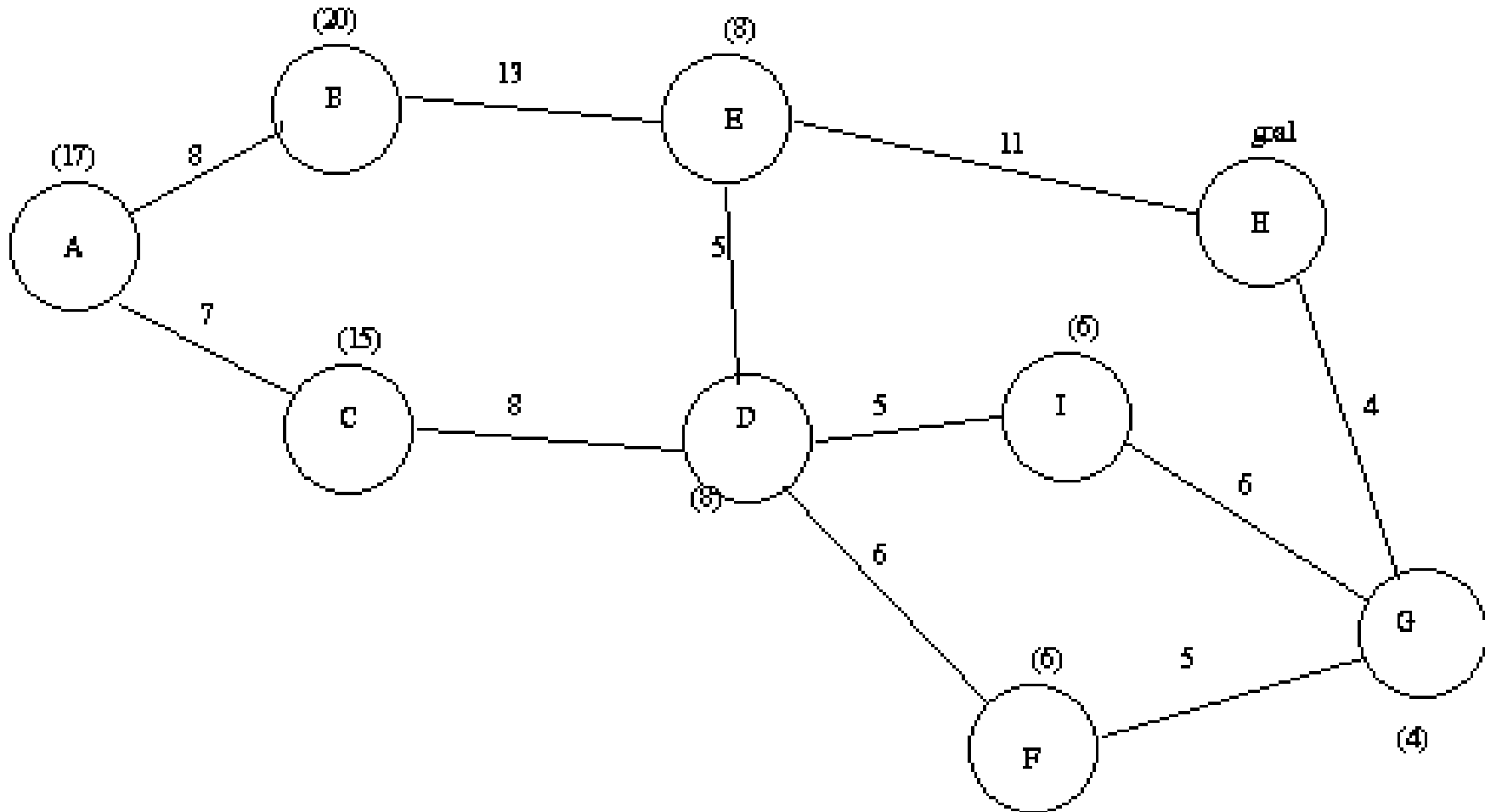
Optimality of A*

- Notice that we can't discard repeated states.
 - We could always keep the version of the state with the lowest g
- More simply, we can also ensure that we always traverse the best path to a node first.
- a *monotonic* heuristic guarantees this.
- A heuristic is monotonic if, for every node n and each of its successors (n') , $h(n)$ is less than or equal to $stepCost(n, n') + h(n')$.
 - In geometry, this is called the triangle inequality.

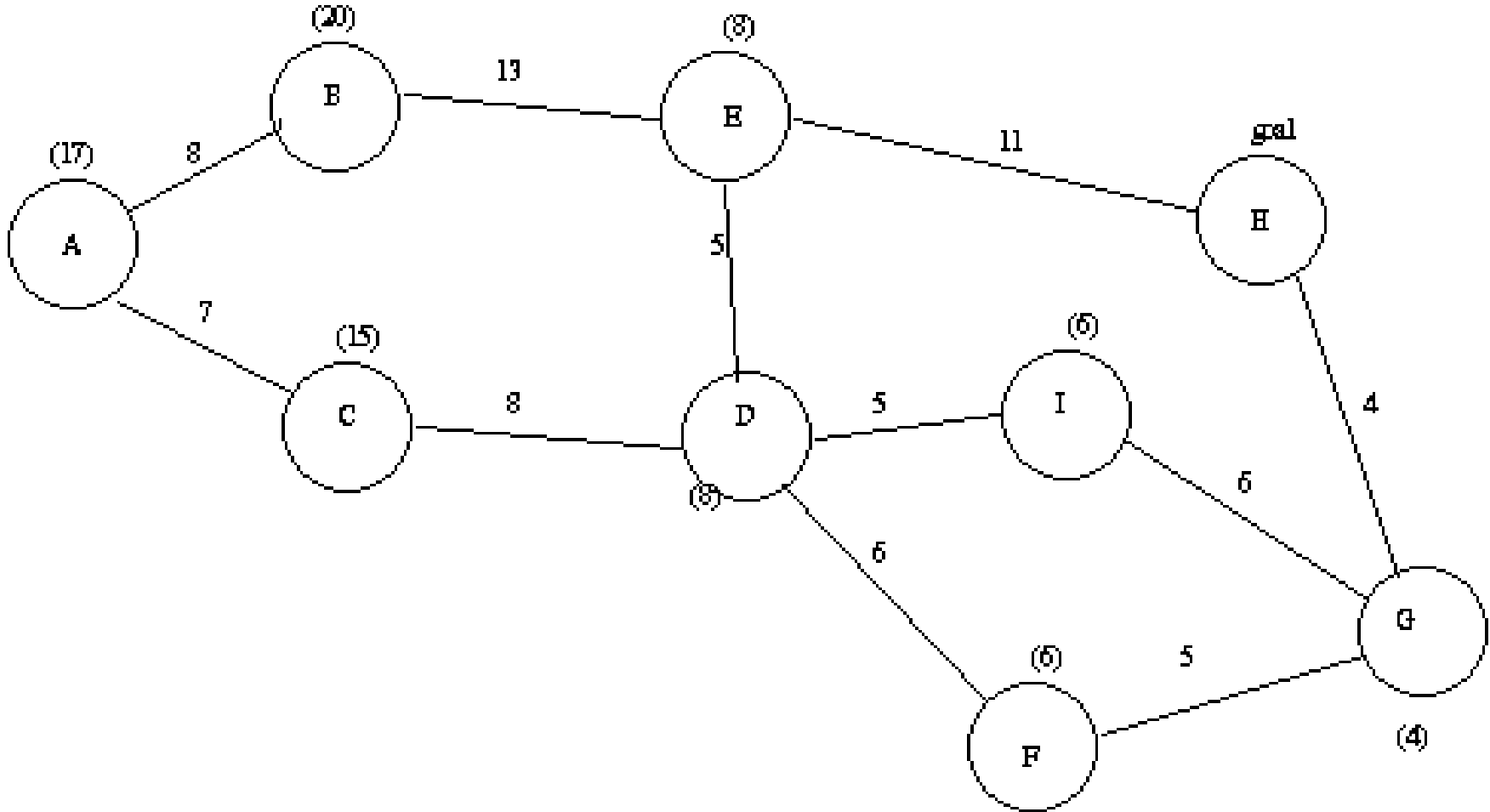
Optimality of A*

- SLD is monotonic. (In general, it's hard to find realistic heuristics that are admissible but not monotonic).
- Corollary: If h is monotonic, then f is nondecreasing as we expand the search tree.
- Alternative proof of optimality.
- Notice also that UCS is A* with $h(n) = 0$
- A* is also *optimally efficient*
 - No other complete and optimal algorithm is guaranteed to expand fewer nodes.

Another A* example

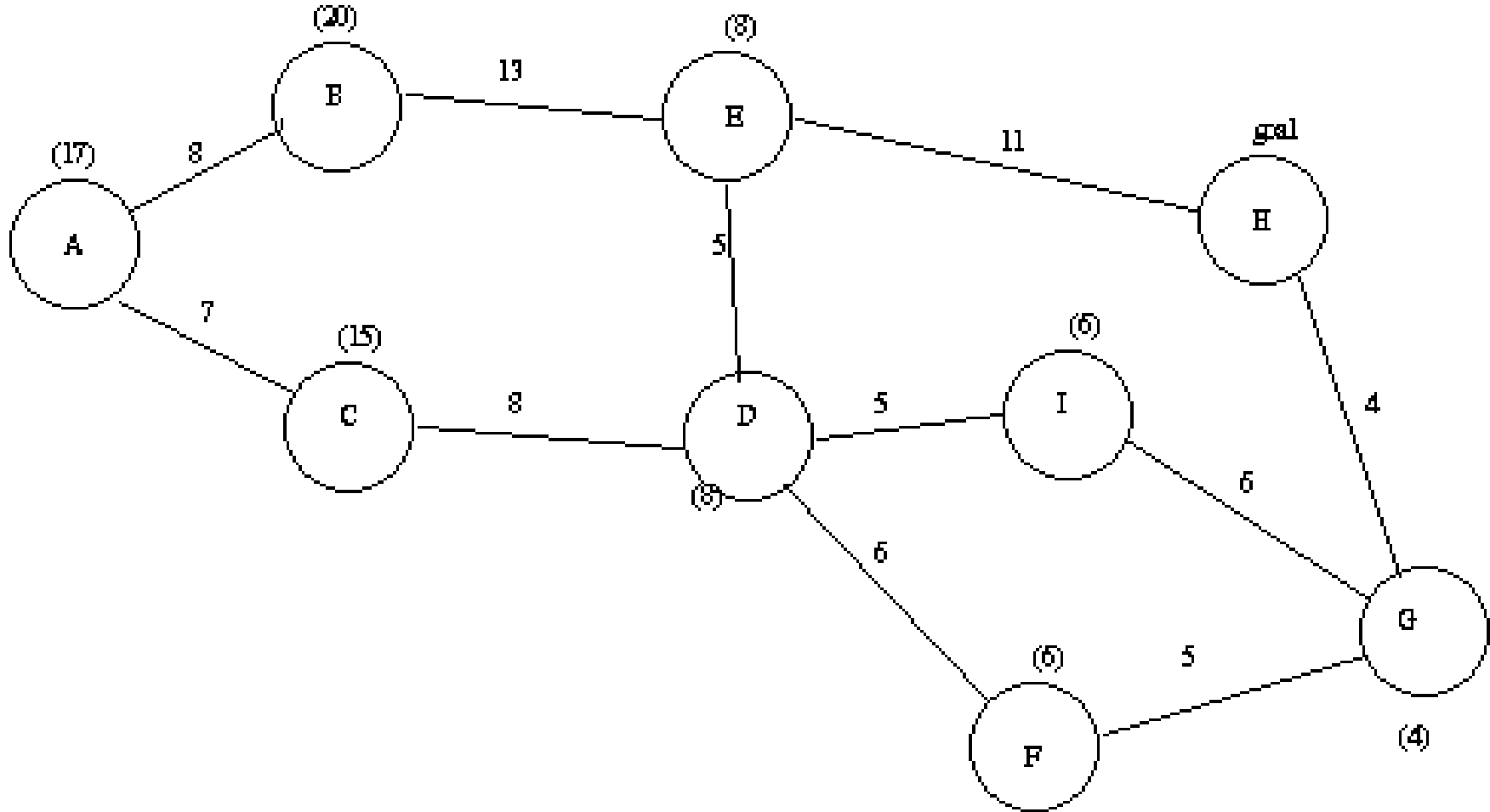


Another A* example



Node: Queue :
-- [(A f = 17, g = 0, h = 17)]

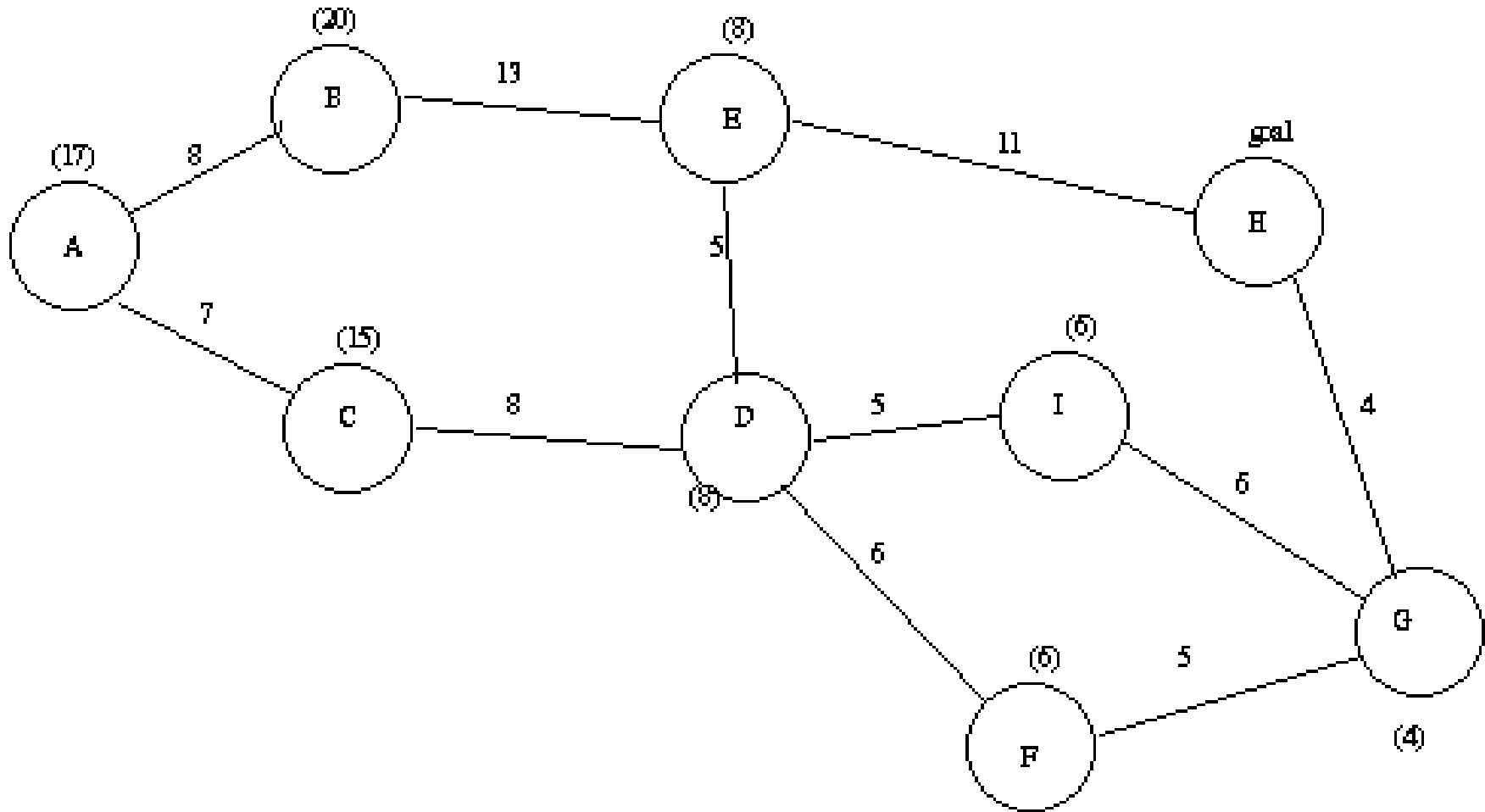
Another A* example



Node: Queue :

A [(C f = 22, g = 7, h = 15), (B f = 28, g = 8, h = 20)]

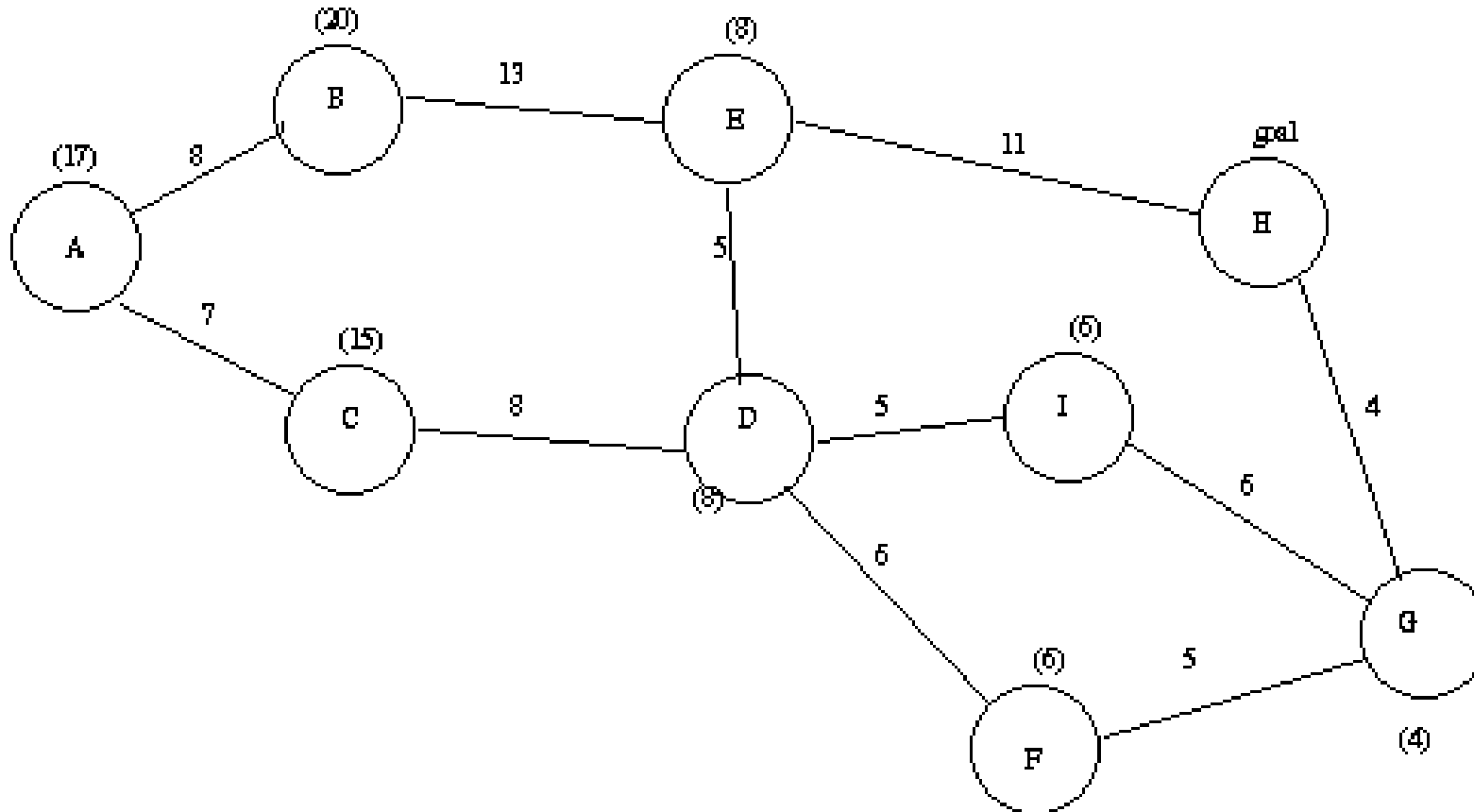
Another A* example



Node: Queue :

C [(D f = 23, g = 15, h = 8), (B f = 28, g = 8, h = 20)]

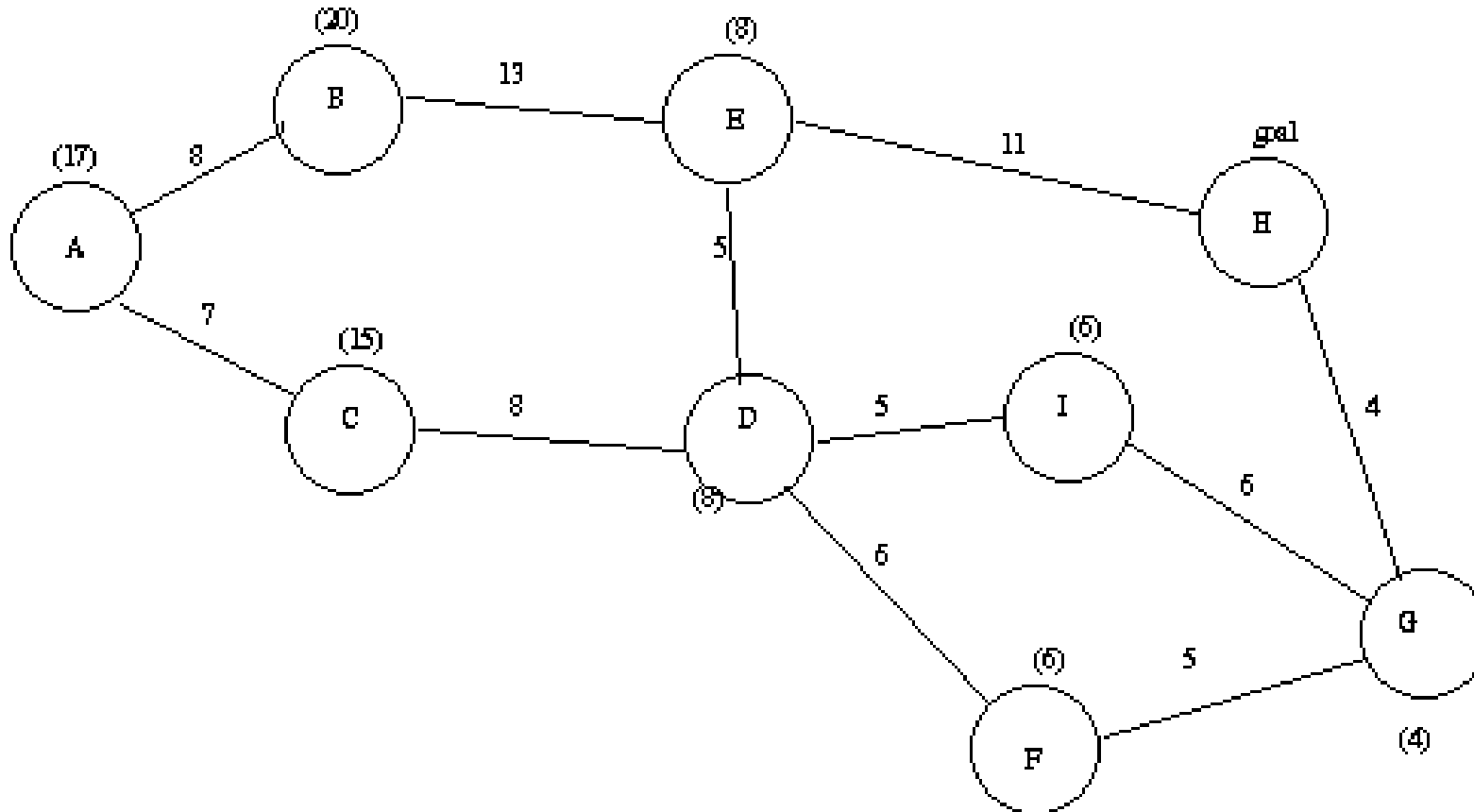
Another A* example



Node: Queue :

D [(I f = 26, g = 20, h = 6), (F f = 27, g = 21, h = 6),
(B f = 28, g = 8, h = 20), (E f = 28, g = 20, h = 8)]

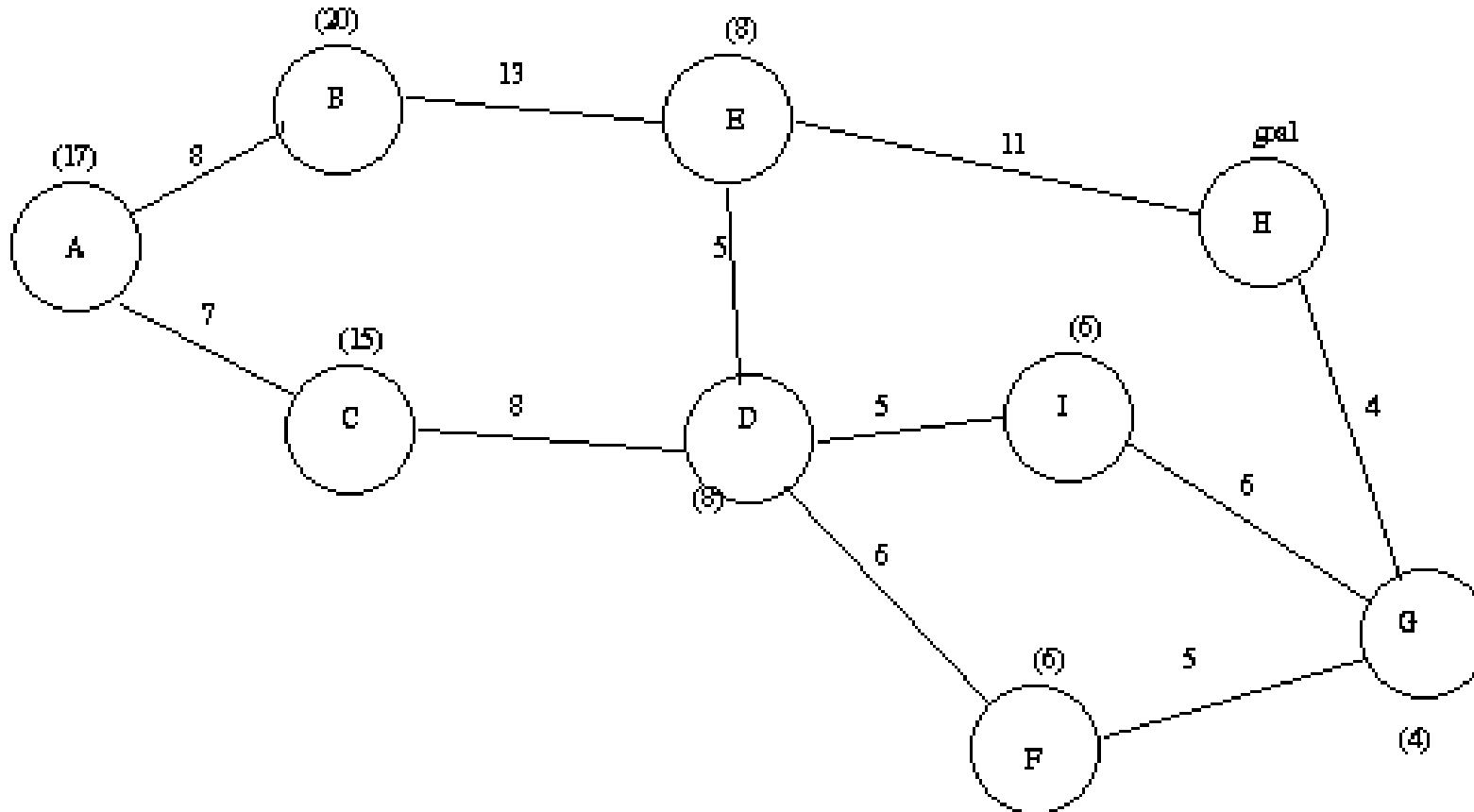
Another A* example



Node: Queue :

I [(F f = 27, g = 21, h = 6), (B f = 28, g = 8, h = 20),
(E f = 28, g = 20, h = 8), (G f = 30 g = 26, h = 4)]

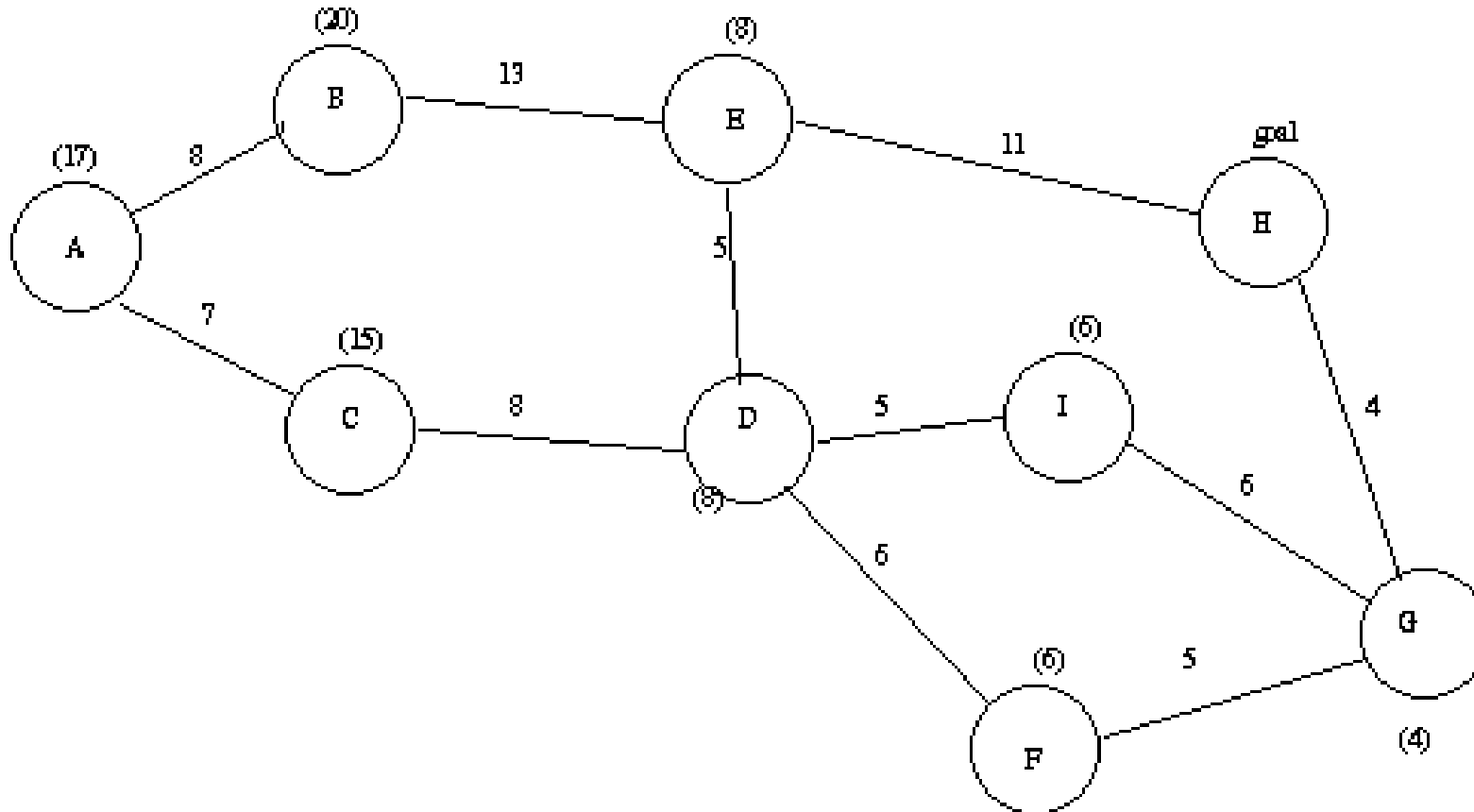
Another A* example



Node: Queue :

F [(B f = 28, g = 8, h = 20), (E f = 28, g = 20, h = 8),
 (G f = 30 g = 26, h = 4), (G f = 30 g = 26 h = 4)]

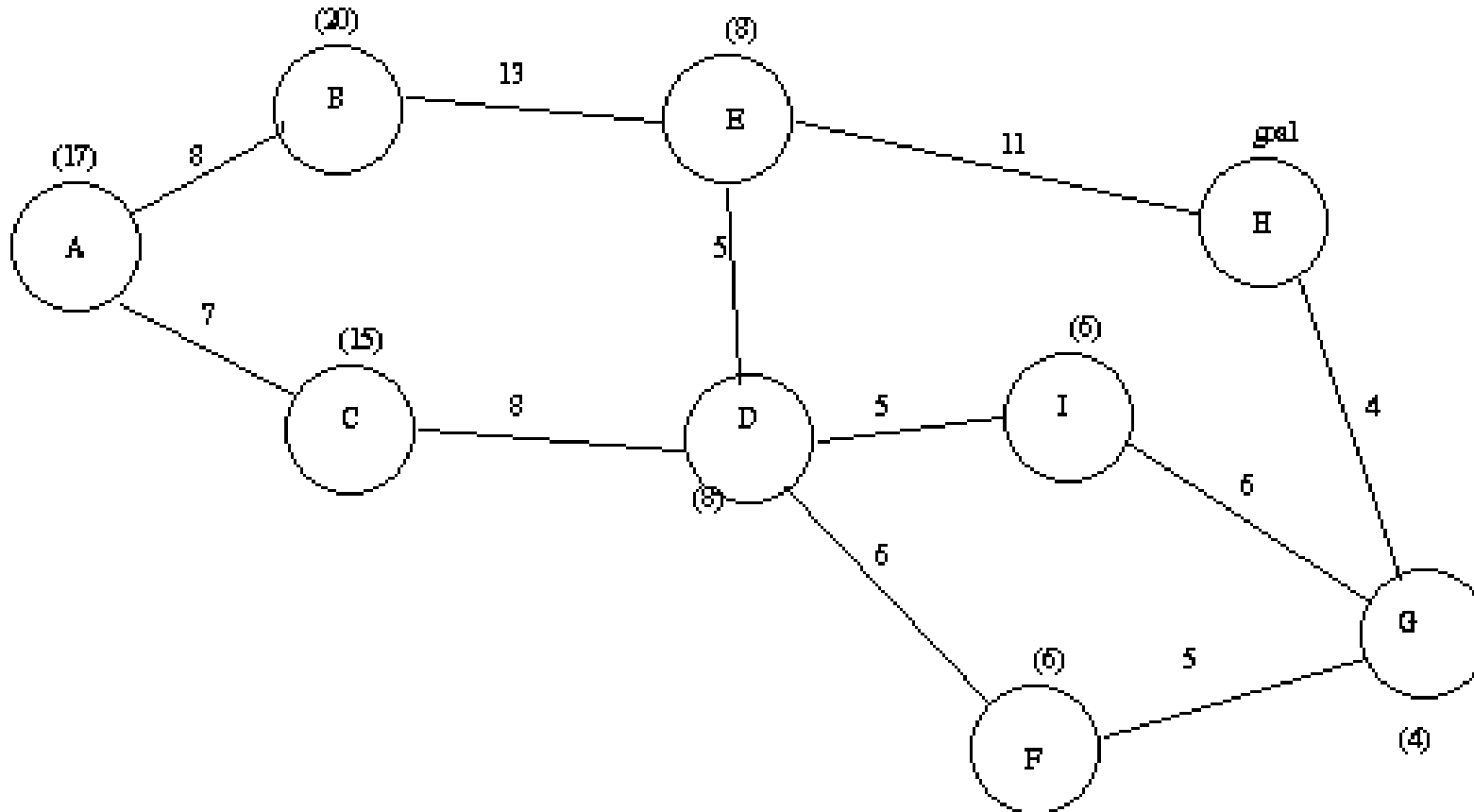
Another A* example



Node: Queue :

B [(E f = 28, g = 20, h = 8), (E f = 29, g = 21, h = 8),
(G f = 30, g = 26, h = 4), (G f = 30, g = 26, h = 4)]

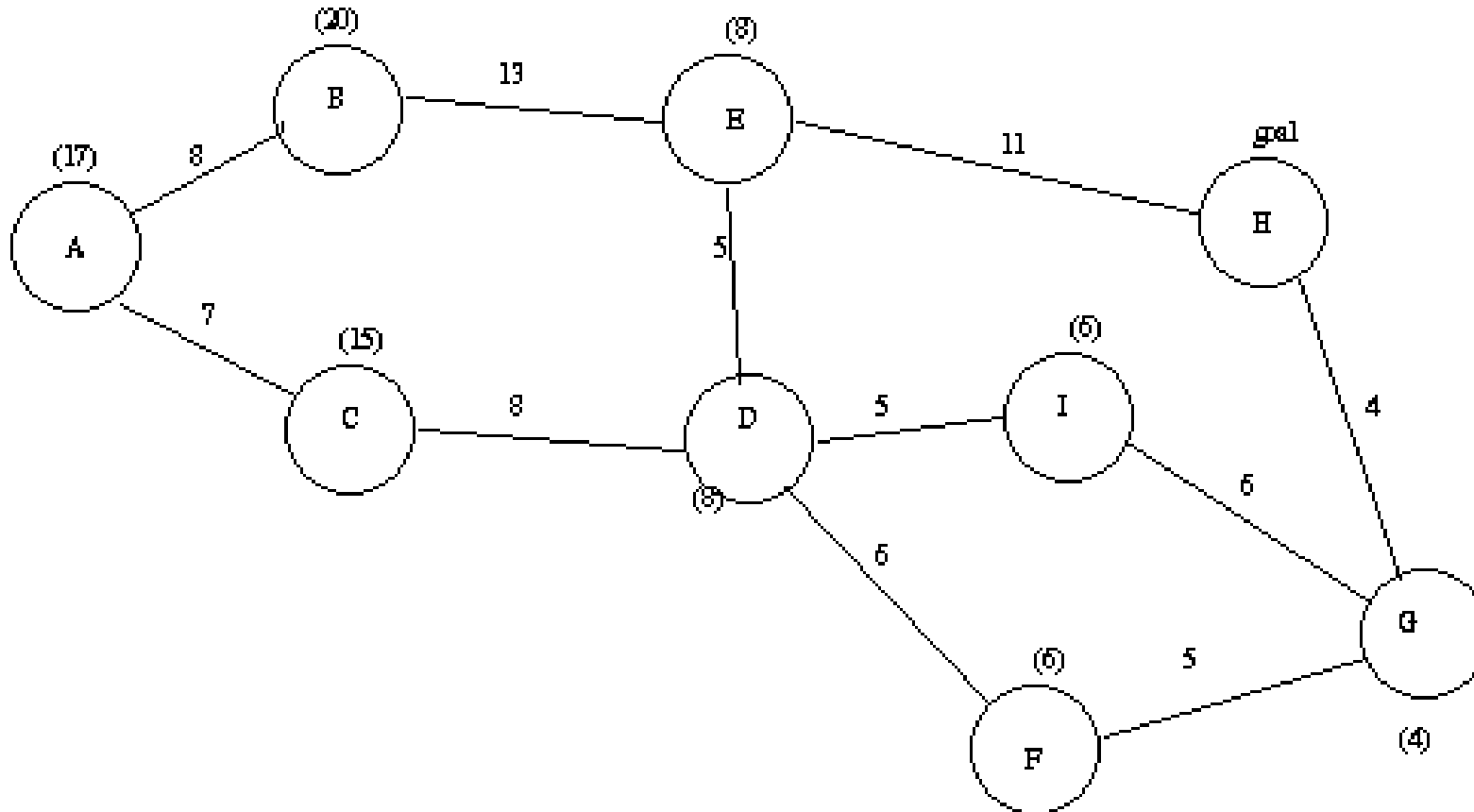
Another A* example



Node: Queue :

E [(E f = 29, g = 21, h = 8), (G f = 30 g = 26, h = 4),
 (G f = 30 g = 26 h = 4), (H f = 31, g = 31, h = 0)]
 (next E can be discarded)

Another A* example

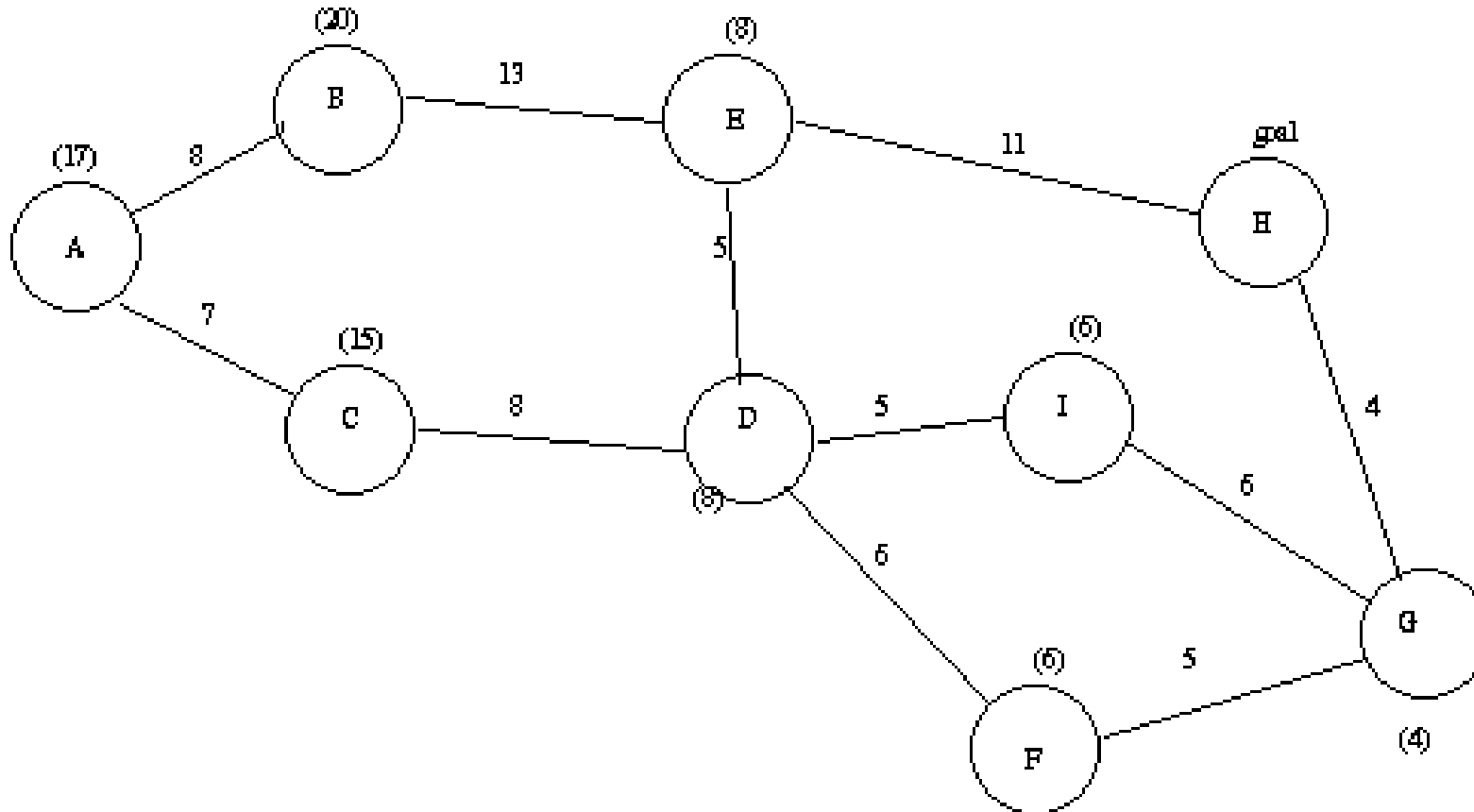


Node: Queue :

G [(G f = 30 g = 26 h = 4), (H f = 30, g = 30, h = 0),
(H f = 31, g = 31, h = 0)]

(next G can be discarded)

Another A* example

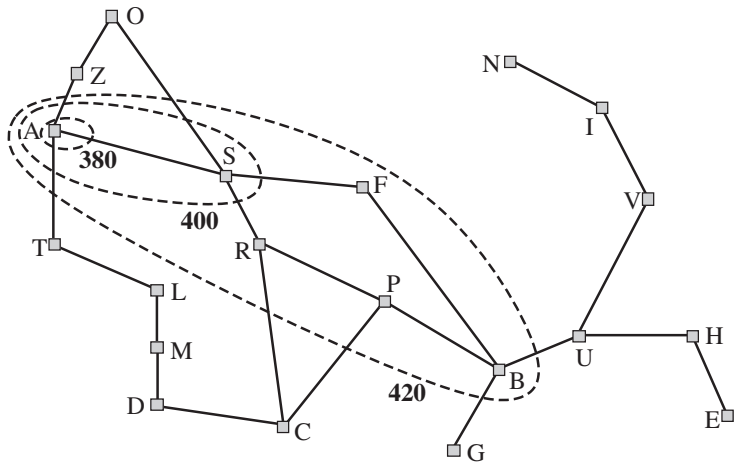


Node: Queue :

H. Solution. [(H f = 31, g = 31, h = 0)]

Solution: A,C,D,I,G,H (or A,C,D,F,G,H)

Pruning and Contours



- Topologically, we can imagine A* creating a set of contours corresponding to f values over the search space.
- A* will search all nodes within a contour before expanding.
- This is a form of *pruning* the search space.
 - The portion of the search tree corresponding to Zerind is pruned

Building Effective Heuristics

- While A^* is optimally efficient, actual performance depends on developing accurate heuristics.
- Ideally, h is as close to the actual cost to the goal (h^*) as possible while remaining admissible.
- Developing an effective heuristic requires some understanding of the problem domain.

Effective Heuristics - 8-puzzle

- h1 - number of misplaced tiles.
 - This is clearly admissible, since each tile will have to be moved at least once.
- h2 - *Manhattan distance* between each tile's current position and goal position.
 - Also admissible - best case, we'll move each tile directly to where it should go.
- Which heuristic is better?

Effective Heuristics - 8-puzzle

- h2 is better.
 - We want h to be as close to h^* as possible.
- If $h_2(n) > h_1(n)$ for all n , we say that h_2 *dominates* h_1 .
- We would prefer a heuristic that dominates other known heuristics.

Finding a heuristic

So how do we find a good heuristic?

Solve a *relaxed* version of the problem

- 8-puzzle: Tile can be moved from A to B if
 - A is adjacent to B
 - B is blank
- Remove restriction that A is adjacent to B
 - Misplaced tiles
- Remove restriction that B is blank
 - Manhattan distance

Finding a heuristic

Relaxing the problem of Romania path-finding

- Add an extra road from each city directly to the goal
- Decreases restrictions on where you can move

Straight-line distance heuristic falls out of this

Finding a heuristic

- Solve subproblems
 - Cost of getting a subset of the tiles in place, while ignoring the cost of moving other tiles
- Could save these subproblem solutions in a DB to consult
- Could be a large number of subproblems, depending on the problem

Iterative Deepening A*

IDA*

- A* has one big weakness - Like BFS, it potentially keeps an exponential number of nodes in memory at once.
- Iterative deepening A* is a workaround.
- Rather than searching to a fixed depth, we search to a fixed f -cost.
 - If the solution is not found, we increase f and start again.
- Works in worlds with uniform, discrete-valued step costs

Improving A*

- Recursive best-first search
 - Combination of DFS and A*.
 - Do DFS, but keep the f-cost of all fringe nodes.
 - If expansion leads to a node worse than something in the fringe, backtrack.
- Improvement over A*, but not spectacular.
- Both IDA* and RBFS throw away too much.

SMA*

Simplified memory-bounded A*

- Regular A*, plus a fixed limit on memory used.
- When memory is full, discard the node with the highest f.
- Value of discarded node is assigned to the parent.
 - Allows SMA* to 'remember' the value of that branch.
 - If all other branches get a higher f value, this child will be regenerated.
- SMA* is complete and optimal.
- On very hard problems, SMA* can wind up repeatedly deleting and regenerating branches.
 - Moral: Often, memory requirements make our problem intractable before time requirements.

Summary

- Problem-specific heuristics can improve search.
- Greedy search
- A^*
- Developing heuristics
 - Admissibility, monotonicity, dominance
- Memory issues