Best First Search (BestFS) – Non-Recursive

```
BestFS(s,problem)
    if problem.goaltest(s.state)  /* if s is a goal */
    then return problem.solution(s)
    else
        frontier = (s)
        loop
            if empty(frontier) then return FAIL
            else
                node = best(frontier)
                remove node from frontier
                for each a in problem.actions(node) do
                    s' = child(a,node)     /* apply(a,node) */
                    if problem.goaltest(s'.state)
                    then return problem.solution(s')
                    else frontier = add s' to frontier
```

A* Search – Change 1

```
A*(s,problem)
 if problem.goaltest(s.state)  /* if s is a goal */
 then return problem.solution(s)
 else
      frontier = (s)
      loop
           if empty(frontier) then return FAIL
           else
                node = best(frontier)  /* state with lowest f=g+h value */
                remove node from frontier
                if problem.goaltest(node.state) then return problem.solution(node)
                for each a in problem.actions(node) do
                     s' = child(a,node)     /* apply(a,node) */
                     if problem.goaltest(s'.state)
                     then return problem.solution(s')
                     else frontier = add s' to frontier
```

A* Search – Change 1

```
A*(s,problem)
 if problem.goaltest(s.state)  /* if s is a goal */
 then return problem.solution(s)
 else
      frontier = (s)
      loop
           if empty(frontier) then return FAIL
           else
                node = best(frontier)  /* state with lowest f=g+h value */
                remove node from frontier
                if problem.goaltest(node.state) then return problem.solution(node)
                for each a in problem.actions(node) do
                     s' = child(a,node)     /* apply(a,node) */
                     frontier = add s' to frontier
```

A* Search – Change 2

```
A*(s,problem)
 if problem.goaltest(s.state)  /* if s is a goal */
 then return problem.solution(s)
 else
      frontier = (s)
      loop
           if empty(frontier) then return FAIL
           else
                node = best(frontier)  /* state with lowest f=g+h value */
                remove node from frontier
                if problem.goaltest(node.state) then return problem.solution(node)
                for each a in problem.actions(node) do
                     s' = child(a,node)     /* apply(a,node) */
                     f = g(node) + c(node,s') + h(s')
                     if s' not in frontier then frontier = add s' to frontier
                     else if f < f(s') then f(s') = f
                                            update solution path to s'
```

A* Search – Change 3 (revisiting states)

```
A*(s,problem)
 if problem.goaltest(s.state)  /* if s is a goal */
 then return problem.solution(s)
 else
      frontier = (s)
      loop
           if empty(frontier) then return FAIL
           else
                node = best(frontier)  /* state with lowest f=g+h value */
                remove node from frontier
                if problem.goaltest(node.state) then return problem.solution(node)
                add node to explored
                for each a in problem.actions(node) do
                     s' = child(a,node)     /* apply(a,node) */
                     f = g(node) + c(node,s') + h(s')
                     if s' not in frontier or explored then frontier = add s' to frontier
                     else if f < f(s') then f(s') = f
                                            update solution path to s'
                                            if s' in explored move it to frontier
```

Admissible/Consistent Heuristics

c(s,s') = cost of going from s to s'

g(s) = the current cost from the initial state to s (sum of $c(s_i,s_{i+1})$ for all steps along path)

g*(s) = the true minimal cost from the initial state to s

h*(s) = the true minimal cost from s to closest goal state

f*(s) = g*(s) + h*(s)

f(s) = g(s) + h(s)

A heuristic h is admissible if $0 \leq h(s) \leq h^*(s)$ for all states s.
(h is "optimistic" – never over-estimates distances to goal)

A heuristic h is consistent if $h(s) \leq c(s,s') + h(s')$ for all states s, s'.
(h obeys the "triangle inequality")

Optimality of A*

A* without checking for revisited states will always generate an optimal solution if h is admissible

A* with checking for revisited states will always generate an optimal solution if h is consistent