Ch.4. Traditional Methods - Part 2

Branch and Bound

■ Branch & Bound

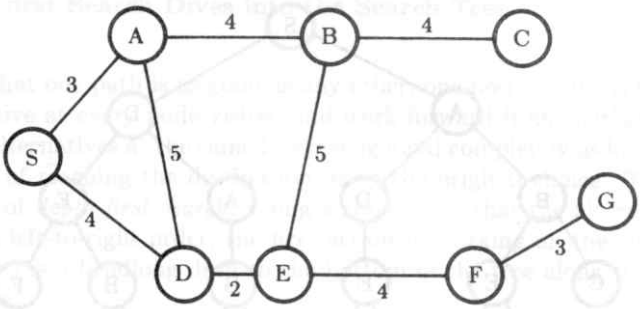
- Tree 구조 로 구성되는 문제에 대하여 partial search 를 통하여 global optimum 을 구하는 최적화 기법

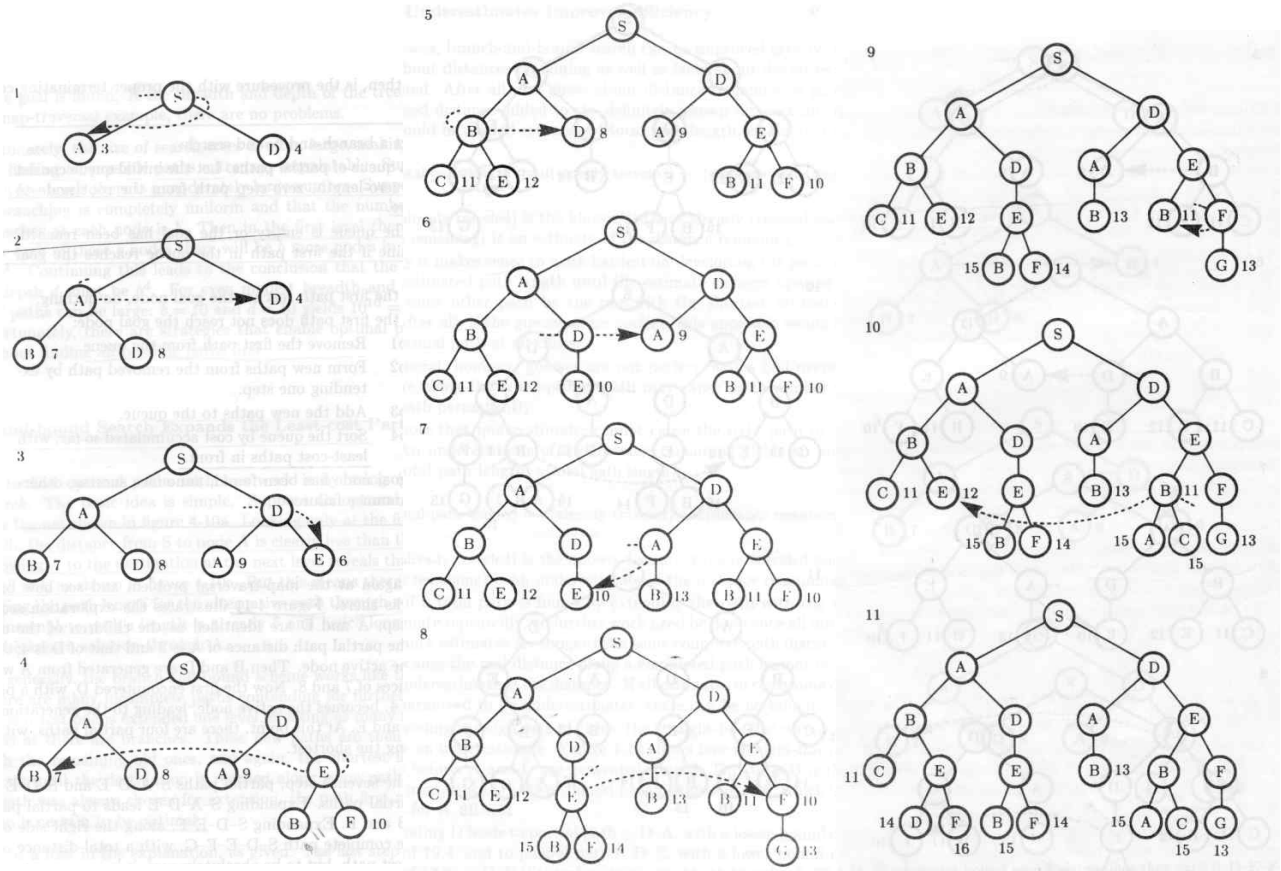
- Branch : 트리 (tree) 구조로 문제를 구성하여 탐색
  Bound : 트리 각 노드에 대하여 'lower bound' (또는 upper bound)를 설정하여, 탐색 영역 제한

- Traditional Methods -

기법	Formulation	Search space	Optimality	Remarks
Exhaustive search	all	full	global	
Local search	all	neighborhood	local	global optimum for convex problem
Greedy	all	partial	local	
Dynamic programming	stage-stae	partial	global	
Branch & bound	tree	partial	global (near)	optimality depends on 'lower bound'
A*	tree	partial	global (?) (near)	optimality depends on 'lower bound' & 'heuristics'

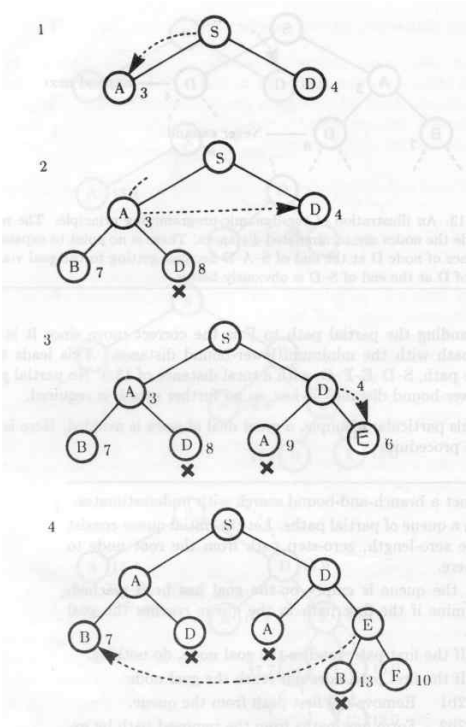
(1) Shortest Path Problem

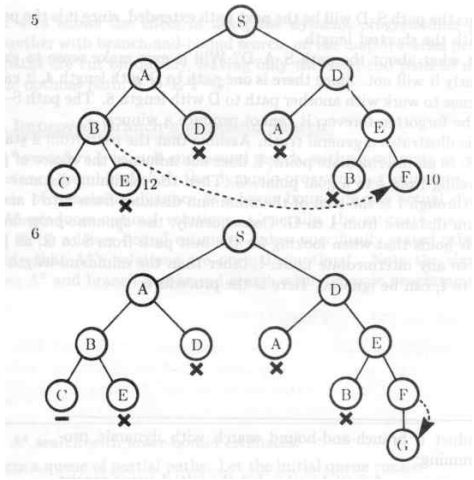




<방법 2> Branch & Bound

- bound: 부모 노드의 Cost ≤ 자식 노드의 Cost
- killed node 적용





Dijkstra's Algorithm

- 특징
  - Single-source shortest problem
  - Nonnegative-weighted edges
  - Running time: lower than Bellman-Ford algorithm (if good implementation)
- Data
  - Graph: V, E, w(u,v)
    - adjacency-list
    - adjacency-matrix
  - d[u]: vertex u 의 shortest-path estimate
  - π[u]: vertex u 의 predecessor
  - S: source s 로 부터의 최단거리가 결정된 vertex 의 집합
  - Q: vertex 들의 minimum-priority queue, key = d

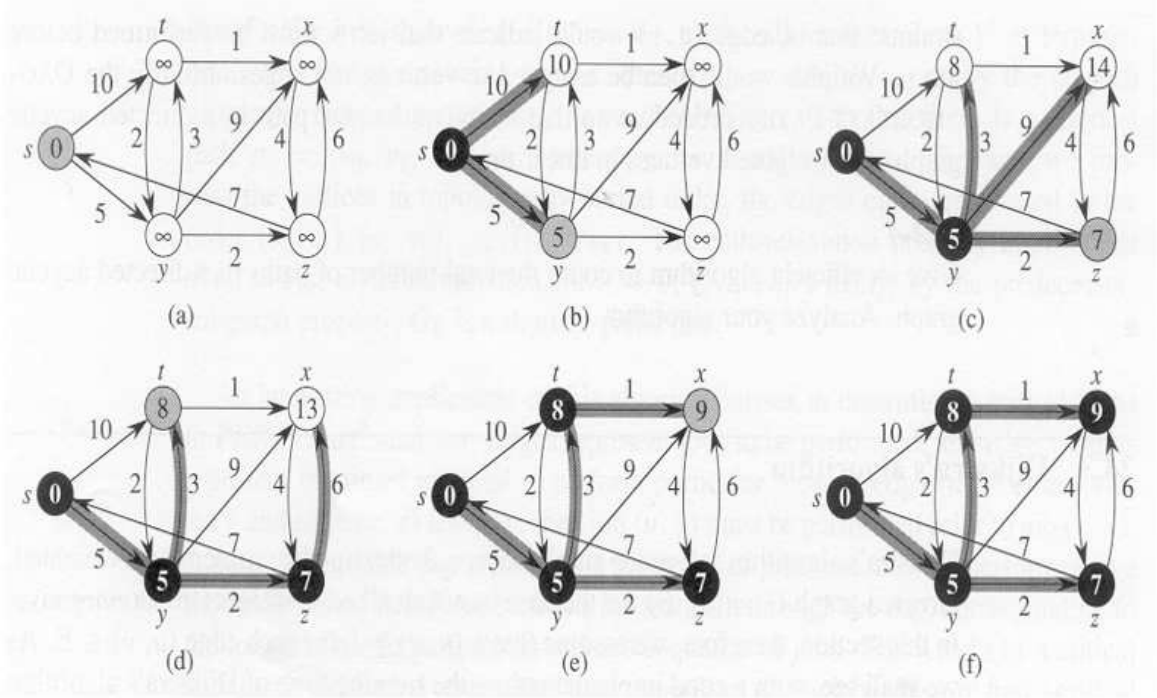
  (cf) breadth first search

- DIJKSTRA(G, w, s)
  - Greedy strategy
  - Optimal solution $d[u] = \delta(s,u)\ ,\ \forall\, u \in U$
  - Running time : O (V lg V + E)
    - line 4-8 : O(V)
    - EXTRACT-MIN(q): O (lg V)
    - Line 7-8: O(E)

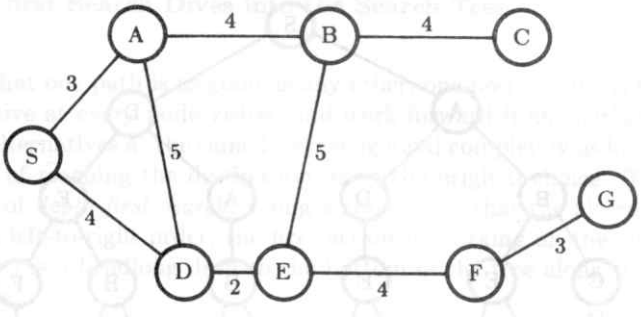
```
DIJKSTRA(G, w, s)
1  INITIALIZE-SINGLE-SOURCE(G, s)
2  S ← ∅
3  Q ← V[G]
4  while Q ≠ ∅
5        do u ← EXTRACT-MIN(Q)
6           S ← S ∪ {u}
7           for each vertex v ∈ Adj[u]
8                 do RELAX(u, v, w)
```

```
RELAX(u, v, w)
1  if d[v] > d[u] + w(u, v)
2     then d[v] ← d[u] + w(u, v)
3          π[v] ← u
```

- Operations



(ex)



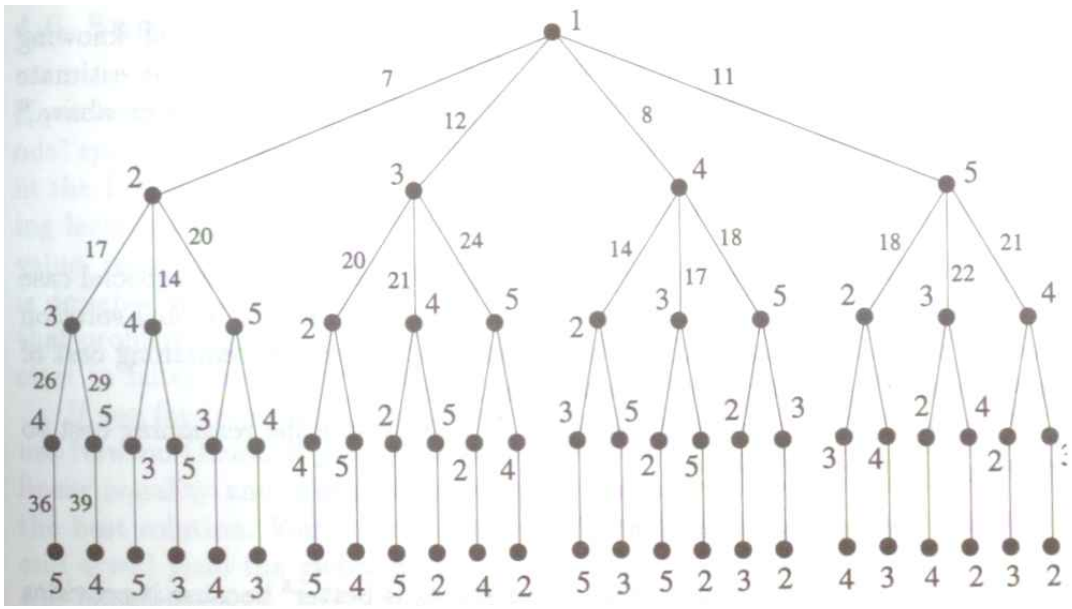
(2) TSP

- 5-city problem
- cost matrix

$$\begin{bmatrix} 0 & 7 & 12 & 8 & 11 \\ 7 & 0 & 10 & 7 & 13 \\ 12 & 10 & 0 & 9 & 12 \\ 8 & 7 & 9 & 0 & 10 \\ 11 & 13 & 12 & 10 & 0 \end{bmatrix}$$

<방법 1> Exhaustive search

- best-first search



<방법 2> Branch & bound

- Lower bound 설정

two shortest edges connected to each city

city 1: (7+8) / 2

city 2: (7+7) / 2

city 3: (9+10) / 2

city 4: (7+8) / 2

city 5: (10+11) / 2

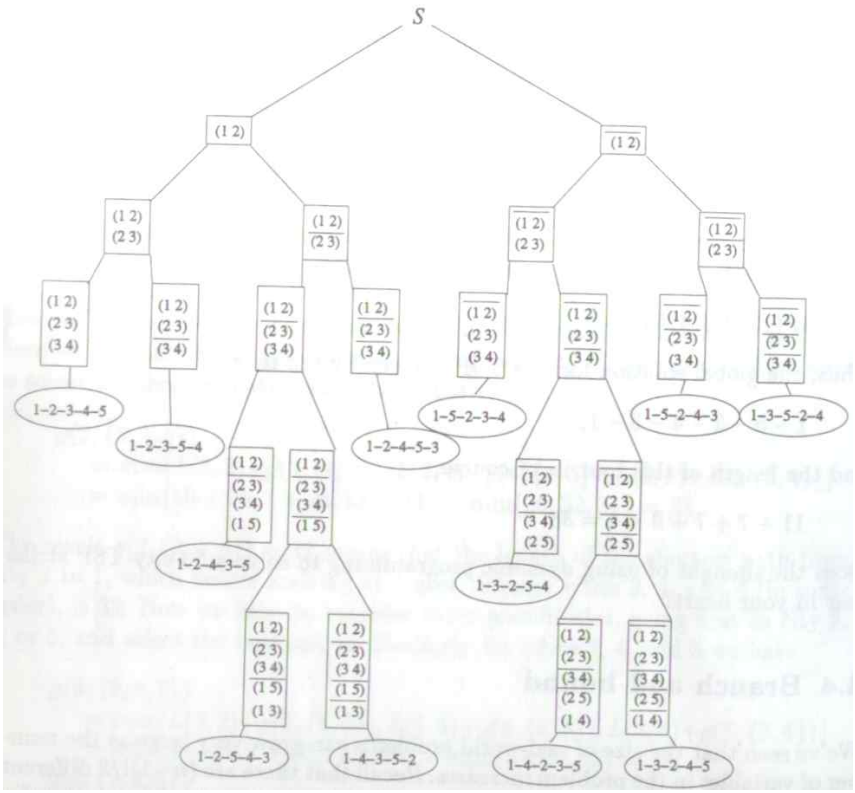
$$\begin{bmatrix} 0 & 7 & 12 & 8 & 11 \\ 7 & 0 & 10 & 7 & 13 \\ 12 & 10 & 0 & 9 & 12 \\ 8 & 7 & 9 & 0 & 10 \\ 11 & 13 & 12 & 10 & 0 \end{bmatrix}$$

(ex)

S : [(7+8) + (7+7) + (9+10) + (7+8) + (10+11)]/2 = 42

$\overline{(1,2)}$ (2,3): [(8+11) +(7+10) + (9+10) + (7+8) + (10+11)]/2 = 45.5

(1,2) (2,4) $\overline{(1,4)}$: [(7+11)+(7+7) + (9+10) + (7+9) +(10+11)]/2 = 44



- The better the lower bound, the faster the algorithm.

■ General Branch & Bound Algorithm

Problem:

$$\min\ f(x)$$

- $x \in D \subseteq R^n$
- $x^* = \{x \in D \mid f(x) = f^*\}$

$f^* = \inf_{x \in D} f(x)$

Notation

D_i: n-dimensional box in R^n

$$x \in D_i \Leftrightarrow l_k^i \le x_k \le u_k^i, \text{ for all } k = 1, \cdots, n$$

C: : candidate set

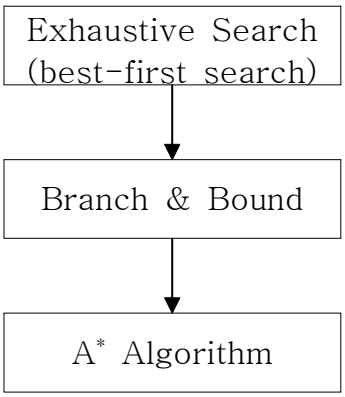
f_{bound}: current upper limit of f^* found by the algorithm

Algorithm

```
procedure branch and bound
begin
  initialize C
  initialize f_bound
  while (not termination-condition) do
    remove-best-box C → D_i
    reduce-or-subdivide D_i → D_i'
    update f_bound
    C ← C ∪ D_i'
    for all D_i ∈ C do
      if (lower bound of f(D_i)) > f_bound then remove D_i from C
end
```

A* Algorithm

■ Branch & Bound Algorithm 의 개선



- 빠른 탐색을 위하여 cost 계산 시 heuristics 추가

■ Algorithm

```
procedure best-first(v)
begin
   visit v
   for each available w do
      assign a heuristic value for w
   q ← the best available node
   best-first(q)
end
```

- Evaluation function

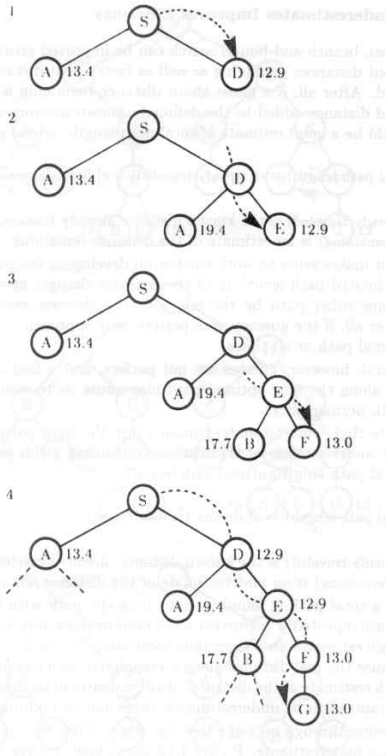
$$eval(q) = c(q) + h(q)$$

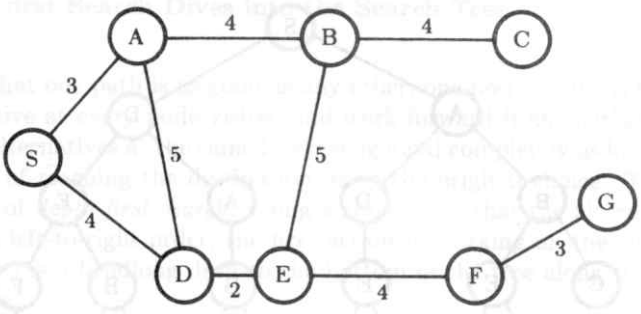
$c(q)$: 시작 노드에서 노드 q 까지의 누적 cost

$h(q)$: 노드 q 에서 마지막 노드까지의 추정 cost

- $h(q)$ 를 어떻게 정의하는 가에 따라 알고리즘 성능 달라짐

(ex) Shortest path problem





A* Algorithm for Robot Path Planning

- Input: graph, start and goal nodes
- Output: back-pointer path (goal → start)
- Data:

  O : open set, priority queue

  C : closed set, contains all processed nodes

  Star(n) : set of nodes which are adjacent to node n

  $c(n_1,n_2)$: length of edge connecting n_1 and n_2

  $g(n)$: total length of a backpointer path from n to q_{start}

  $h(n)$: heuristic cost function,

  estimated cost of shortest path from n to q_{goal}

  $f(n)=g(n)+h(n)$: estimated cost of shortest path from q_{start} to q_{goal} via n

- Algorithm

Algorithm 24 A^* Algorithm

Input: A graph

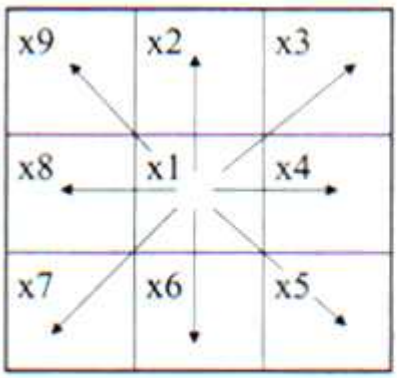
Output: A path between start and goal nodes

1: **repeat**
2: Pick n_{best} from O such that $f(n_{best}) \leq f(n), \forall n \in O$.
3: Remove n_{best} from O and add to C.
4: If $n_{best} = q_{goal}$, EXIT.
5: Expand n_{best}: for all $x \in$ Star(n_{best}) that are not in C.
6: **if** $x \notin O$ **then**
7: add x to O.
8: **else if** $g(n_{best}) + c(n_{best}, x) < g(x)$ **then**
9: update x's backpointer to point to n_{best}
10: **end if**
11: **until** O is empty

- Example of A* on a Grid

6	h =6 f = b=()	h =5 f = b=()	h =4 f = b=()	h =3 f = b=()	h =2 f = b=()	h =1 f = b=()	h =0 f = Goal
5	h =6.4 f = b=()	h =5.4 f = b=()	h =4.4 f = b=()	h =3.4 f = b=()	h =2.4 f = b=()	h =1.4 f = b=()	h =1 f = b=()
4	h =6.8 f = b=()	h =5.8 f = b=()	h =4.8 f = b=()	h =3.8 f = b=()	h =2.8 f = b=()	h =2.4 f = b=()	h =2 f = b=()
3	h =7.2 f = b=()	h =6.2 f = b=()	h =5.2 f = b=()	h =4.2 f = b=()	h =3.8 f = b=()	h =3.4 f = b=()	h =3 f = b=()
2	h =7.6 f = b=()	h =6.6 f = b=()	h =5.6 f = b=()	h =5.2 f = b=()	h =4.8 f = b=()	h =4.4 f = b=()	h =4 f = b=()
1	h =8.0 f = b=()	h =7.0 f = Start	h =6.6 f = b=()	h=6.2 f= b=()	h =5.8 f = b=()	h =5.4 f = b=()	h =5 f = b=()
r/c	1	2	3	4	5	6	7

Figure H.15 Heuristic values are set, but backpointers and priorities have not.



c(x1, x2) = 1
c(x1, x9) = 1.4

c(x1, x8) = 10000, if x8 is in obstacle, x1 is a free cell

c(x1,x9) = 10000.4, if x9 is in obstacle, x1 is a free cell

Figure H.16 Eight-point connectivity and possible cost values.

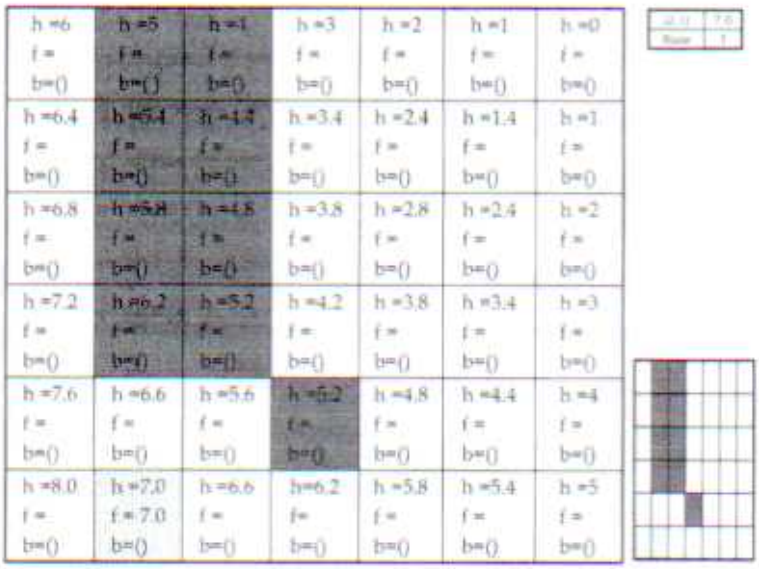


Figure H.17 Start node is put on priority queue, displayed in upper right.

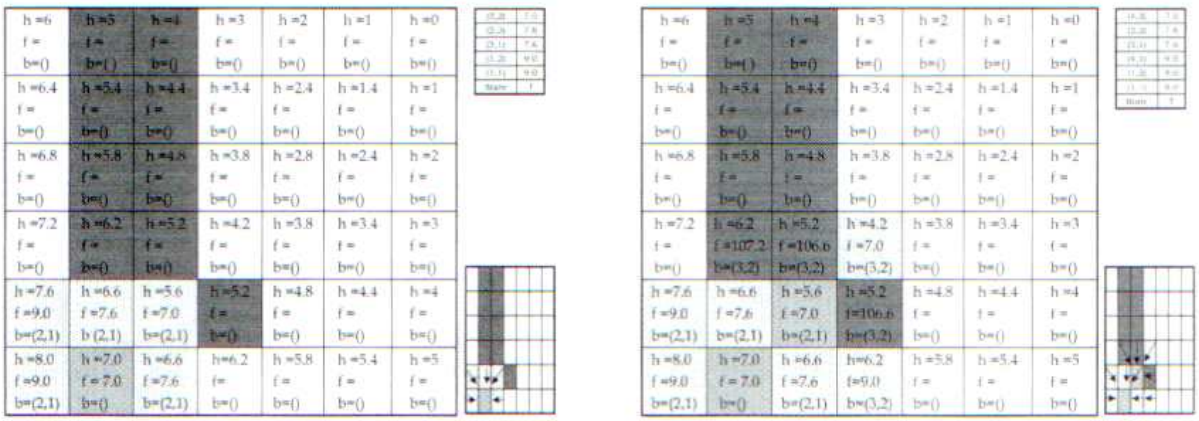


Figure H.18 (Left) The start node is expanded, the priority queue is updated, and the back-pointers are set, which are represented by the right bottom icon. $b = (i, j)$ points to cell (i, j). (Right) Cell (3, 2) was expanded. Note that pixels (3, 3), (2, 3), and (4, 2) are not displayed in the priority queue because they correspond to obstacles.

Project: