CS370

Symbolic Programming

Declarative Programming

LECTURE 13: Best-First Heuristic Search

Jong C. Park

park@cs.kaist.ac.kr

Computer Science Department

Korea Advanced Institute of Science and Technology

http://nlp.kaist.ac.kr/~cs370

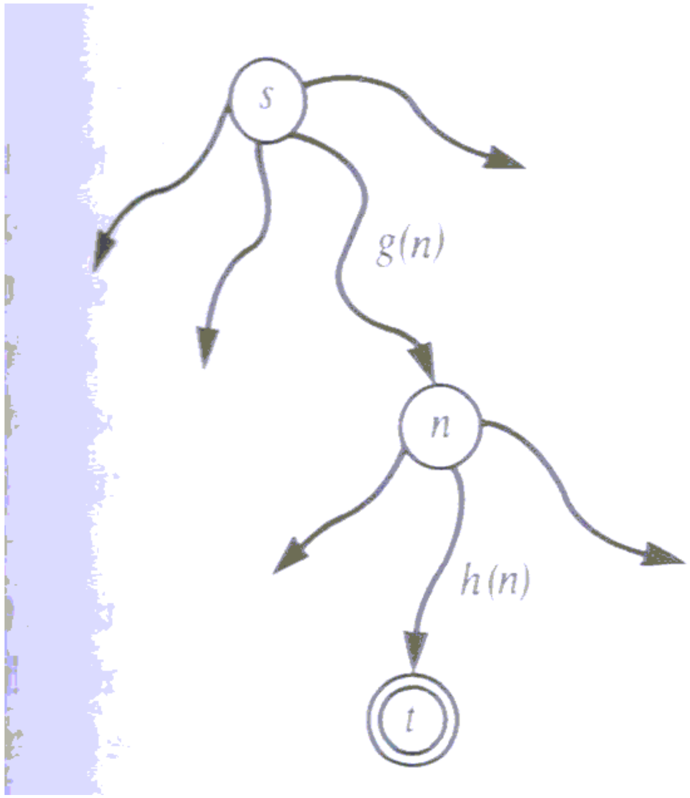
Best-First Heuristic Search

- Best-first search
- Best-first search applied to the eight puzzle
- Best-first search applied to scheduling
- Space-saving techniques for best-first search

Best-first search

- Refinement of a breadth-first search program
- Use heuristic estimate for candidate paths
- Always expand the best candidate path
- c: a cost function for the arcs
  - c(n,n')
- f: a heuristic estimator function for the nodes
  - f(n): the "difficulty" of node n

Best-first search



$$f(n) = g(n) + h(n)$$

g(n): an estimate of the cost of an optimal path from s to n

h(n): an estimate of the cost of an optimal path from n to t

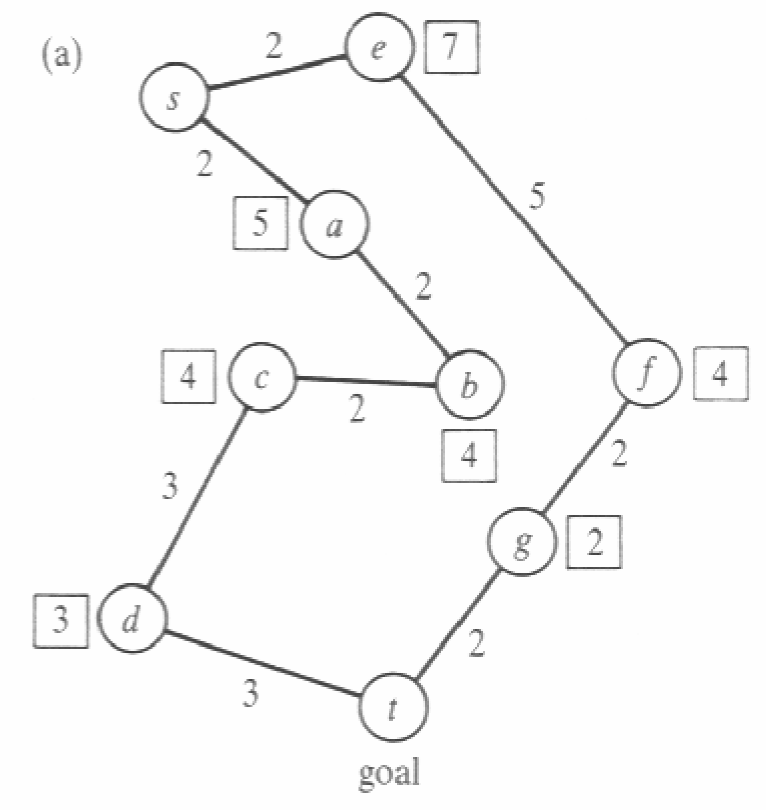
Best-first search

- **Use an activate-deactivate mechanism for multiple competing processes**
  - Each process for a candidate path explores its own subtree.
  - New subprocesses are created on alternatives.
  - Only one process is active at a time.
  - The active process is assigned some budget.

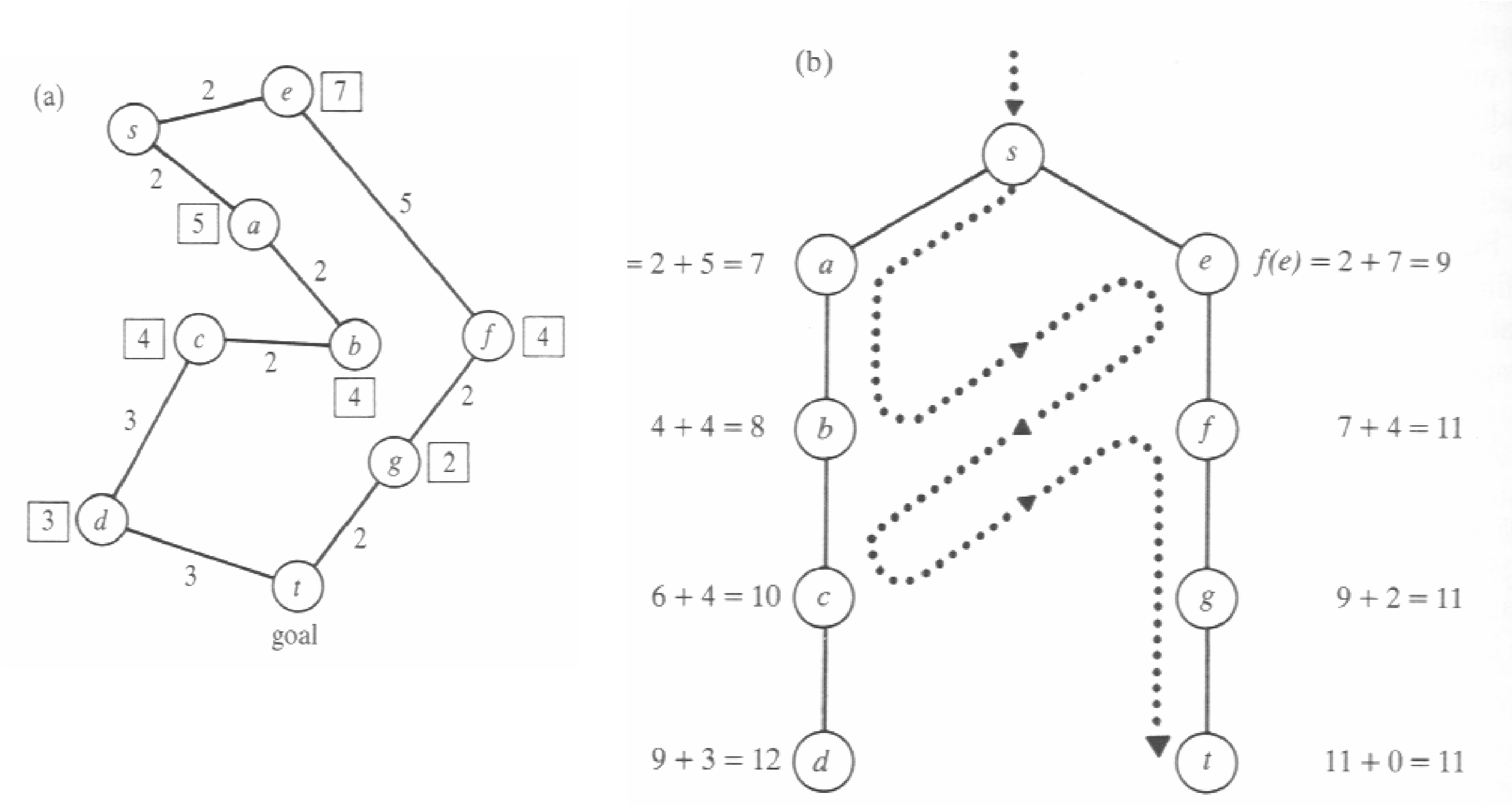
Example

- The shortest route between two cities
  - start city s
  - goal city t
  - straight-line distance for h(n)

$$f(X) = g(X) + dist(X,t)$$



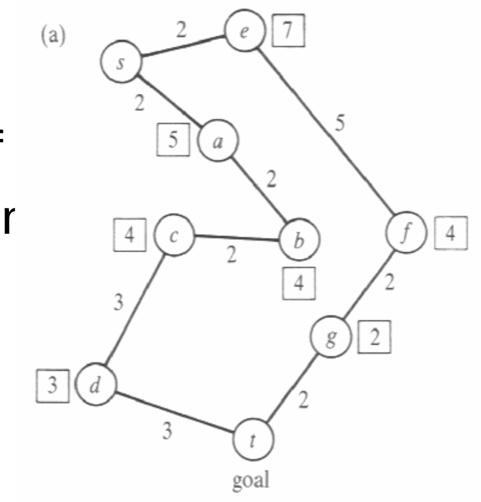
Best-first search



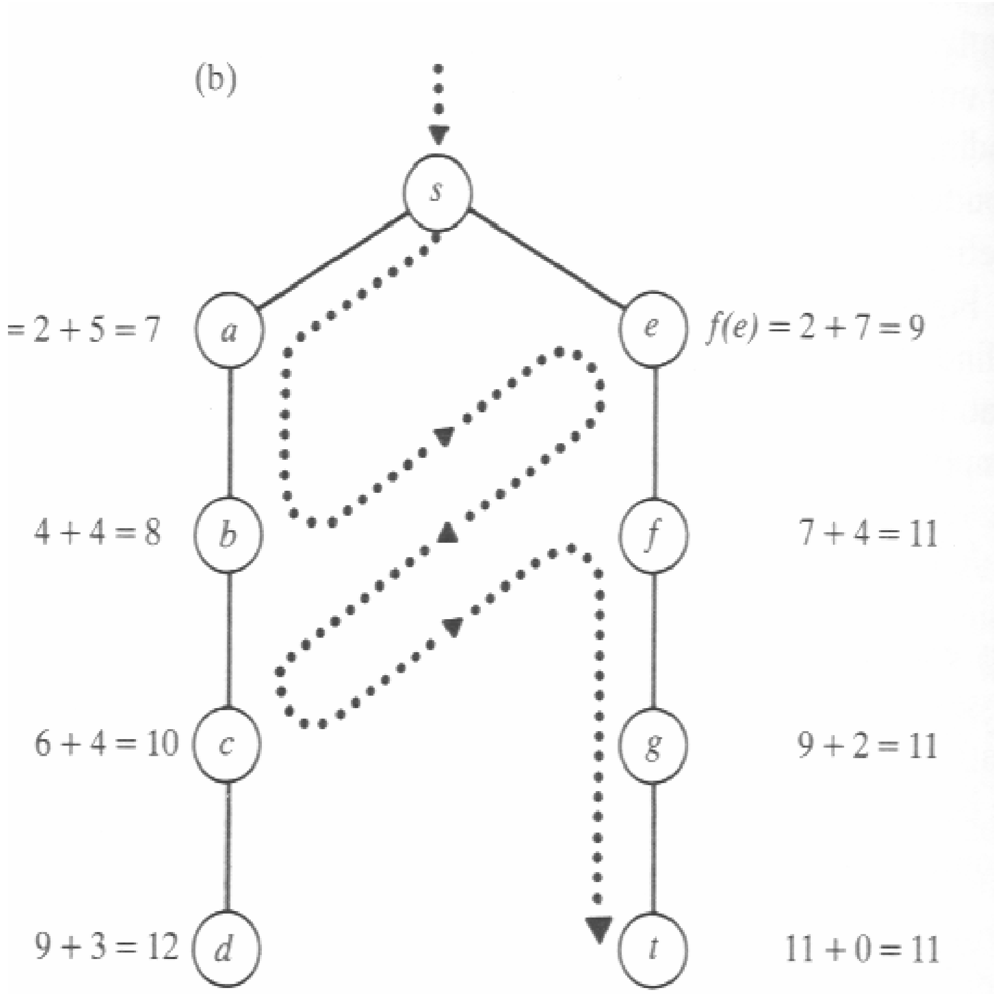
Best-first search

⦿ Implementation

- l(N,F/G): a single node tree (a leaf)
  - N is a node in the state space
  - G is g(N) and F is f(N) = G + h(N)
- t(N,F/G,Subs): a tree with non-empty subtrees
  - N is the root of the tree
  - Subs is a list of its subtrees
  - G is g(N); F is the updated f-value of
  - Subs is ordered according to increasir
    f-values of the subtrees



Best-first search



t(s,7/0,[l(a,7/2),l(e,9/2)])

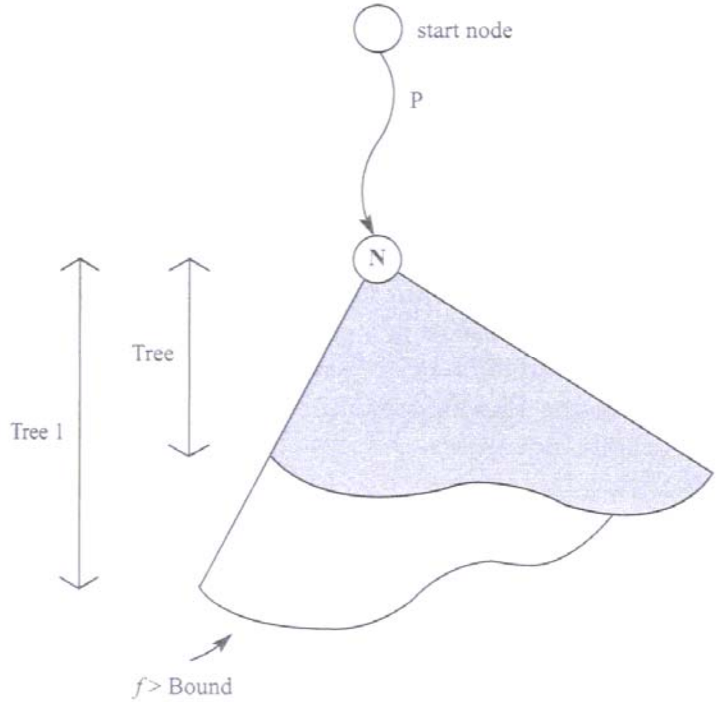
(after s is expanded)

t(s,9/0,[l(e,9/2),t(a,10/2,
[t(b,10/4,[l(c,10/6)])])])

(after b is expanded)

Implementation

- expand(P,Tree,Bound,Tree1,Solved,Solution)
  - P: Path between the start nod and Tree
  - Tree: Current search (sub)tre
  - Bound: f-limit for expansion o Tree
  - Tree1: Tree expanded within Bound
  - Solved: Indicator whose value is 'yes', 'no', 'never'
  - Solution: A solution path from the start node 'through Tree1' to a goal node within Bound (i it exists)



Best-first search

⦿ Implementation

```
bestfirst(Start, Solution) :-
  expand([ ],l(Start,0/0),9999,_,yes,Solution).
expand(P,l(N,_),_,_,yes,[N|P]) :- goal(N).
expand(P,l(N,F/G),Bound,Tree1,Solved,Sol) :-
  F =< Bound,
  (bagof(M/C,(s(N,M,C),not member(M,P)),Succ),
   !, succlist(G,Succ,Ts), bestf(Ts,F1),
  expand(P,t(N,F1/G,Ts),Bound,Tree1,Solved,Sol)
   ;  Solved = never ).
expand(P,t(N,F/G,[T|Ts]),Bound,Tree1,Solved,Sol) :-
  F =< Bound, bestf(Ts,BF), min(Bound,BF,Bound1),
  expand([N|P],T,Bound1,T1,Solved1,Sol),
  continue(P,t(N,F/G,[T1|Ts]),Bound,Tree1,Solved1,Solv
  ed,Sol).
expand(_,t(_,_,[ ]),_,_,never,_) :- !.
expand(_,Tree,Bound,Tree,no,_) :- f(Tree,F), F > Bound.
```

Best-first search

Admissibility of a search algorithm

- Always produces an optimal solution (i.e. a minimal cost path) when a solution exists
  - The previous implementation, which produces all solutions through backtracking, can be considered admissible if the first solution found is optimal.
  - Let h*(n) denote the cost of an optimal path from n to a goal node.
  - An A* algorithm that uses a heuristic function h such that for all nodes n in the state space h(n) <= h*(n) is admissible.

Best-first search applied to the eight puzzle

⦿ Problem

- goal

1	2	3
8		4
7	6	5

⦿ Problem-specific predicates

- s(Node,Node1,Cost)
- goal(Node)
- h(Node,H)

Best-first search applied to the eight puzzle

⊙Goal situation

```
goal([2/2, 1/3, 2/3, 3/3, 3/2, 3/1, 2/1, 1/1,
  1/2]).
```

3	**1**	**2**	**3**
2	**8**		**4**
1	**7**	**6**	**5**
	1	2	3

Best-first search applied to the eight puzzle

⊙ Heuristic estimate H

mandist(S1,S2,D): Manhattan distance

totdist: the total distance of the eight tiles in Pos from their home squares

1	2	3
8		4
7	6	5

1	3	4
8		2
7	6	5

(a)

2	8	3
1	6	4
7		5

(b)

2	1	6
4		8
7	5	3

(c)

Best-first search applied to the eight puzzle

⦿ Heuristic estimate H

seq: the sequence score that measures the degree to which the tiles are already ordered in the current position with respect to the order required in the goal.

1	2	3
8		4
7	6	5

1	3	4
8		2
7	6	5

(a)

2	8	3
1	6	4
7		5

(b)

2	1	6
4		8
7	5	3

(c)

Best-first search applied to the eight puzzle

- h(Pos,H)
  - H = totdist
  - H = totdist + 3*seq

1	2	3
8		4
7	6	5

1	3	4
8		2
7	6	5

(a)

2	8	3
1	6	4
7		5

(b)

2	1	6
4		8
7	5	3

(c)

Best-first search applied to the eight puzzle

Implementation

```
s([Empty|Tiles],[Tile|Tiles1],1) :-
    swap(Empty,Tile,Tiles,Tiles1).
swap(Empty,Tile,[Tile|Ts],[Empty|Ts]) :- mandist(Empty,Tile,1).
swap(Empty,Tile,[T1|Ts],[T1|Ts1]) :- swap(Empty,Tile,Ts,Ts1).
mandist(X/Y,X1/Y1,D) :- dif(X,X1,Dx), dif(Y,Y1,Dy), D is Dx+Dy.
dif(A,B,D) :- D is A-B, D >= 0, ! ; D is B-A.
h([Empty|Tiles],H) :-
    goal([Empty1 | GoalSquares]),
    totdist(Tiles,GoalSquares,D), seq(Tiles,S), H is D+3*S.
totdist([ ],[ ],0).
totdist([Tile|Tiles],[Square|Squares],D) :-
    mandist(Tile,Square,D1), totdist(Tiles,Squares,D2),
    D is D1+D2.
```

Best-first search applied to scheduling

⦿ task-scheduling problem

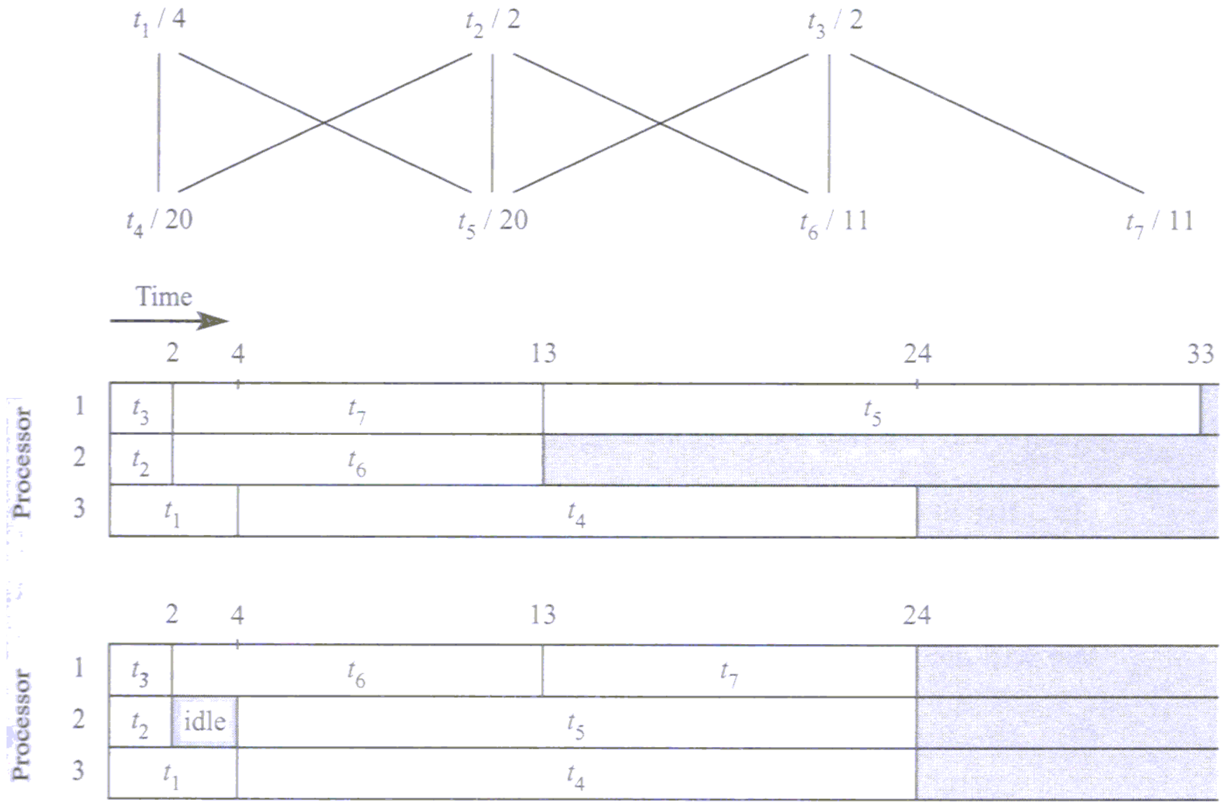
Given

- a collection of tasks t1, t2, ... with predefined execution times and a precedence relation
- a set of m identical processors, where any task can be executed on any processor and each processor can only execute one task at a time

Goal

- minimize the finishing time over all permissible schedules

Best-first search applied to scheduling



Space-saving techniques

⦿ Time and space complexity of A*

- Heuristic guidance results in the reduction of effective branching of search.
- The order of the complexity of A* is still exponential in the depth of search, w.r.t. both time and space.
  - Why?
    -
- Which is more costly: space or time?
  - In most practical situations space is more critical.
  - two space-saving techniques
    -
    -

Space-saving techniques

⊙ IDA* - iterative deepening A*

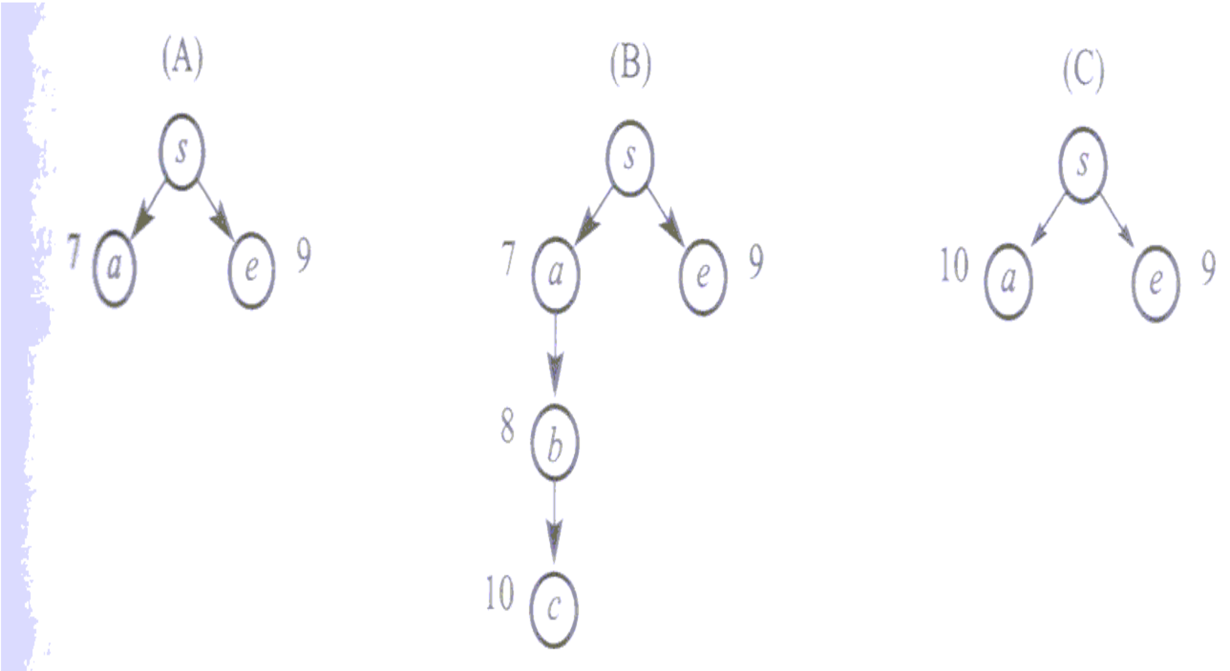
- In IDA*, the successive depth-first searches are bounded by the current limit in the values of the nodes (heuristic f-values of the nodes).
- the evaluation function f
  - How good f is depends on how many nodes have equal f-values.
    -
    -

IDA* - iterative deepening A*

- Properties of IDA*
  - acceptability of the overheads of repeated searches
  - addmissibility
    - If h is admissible (h(N) <= h*(N) for all N), then IDA* is guaranteed to find an optimal solution.
  - It does not guarantee that the nodes are explored in the best-first order (i.e. the order of increasing f-values).
    -

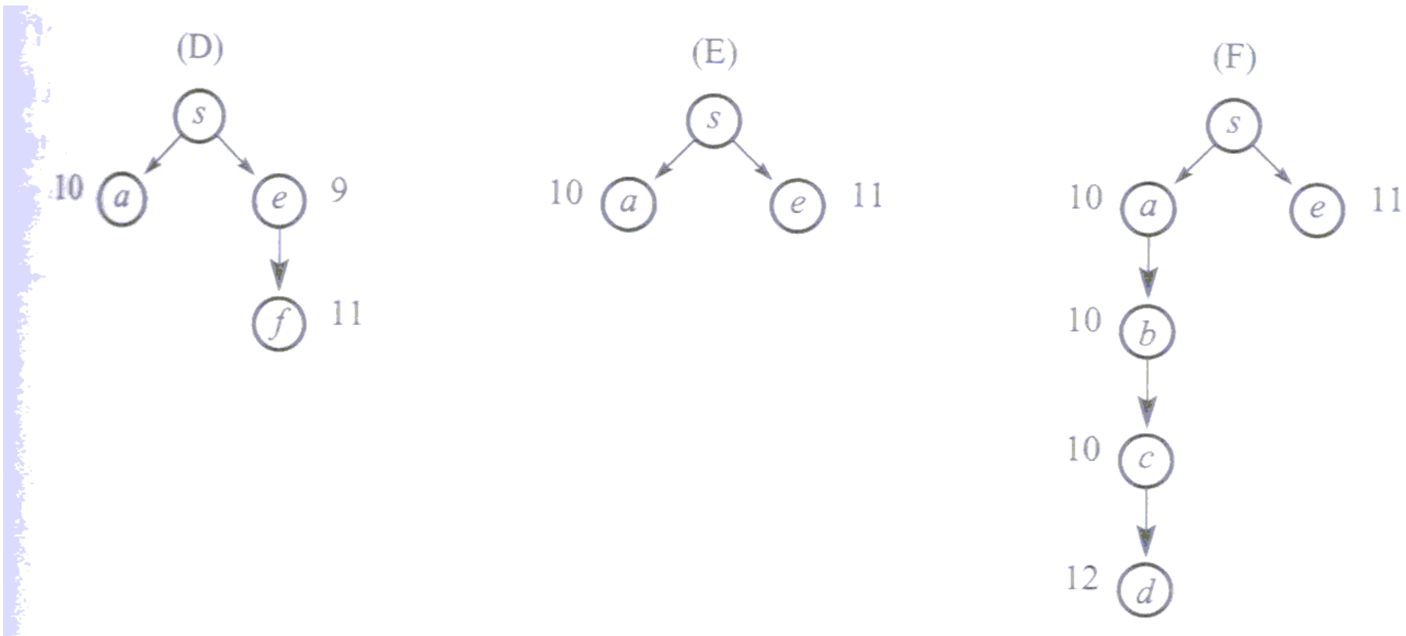
RBFS - recursive best-first search

- Unlike A*, RBFS only keeps the current search path and the sibling nodes along this path.

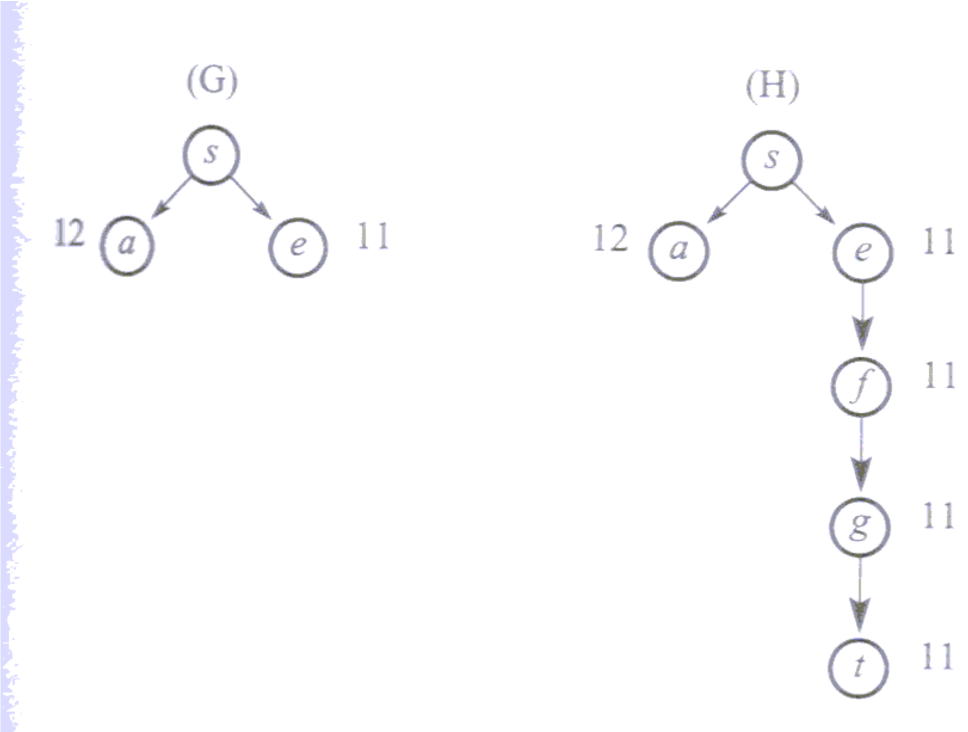


Space-saving techniques

⦿ RBFS - recursive best-first search



⦿RBFS - recursive best-first search



Space-saving techniques

RBFS - recursive best-first search

- Characteristics
  - The space complexity is linear in the depth of search, at the expense of the time for regenerating already generated nodes.
  - It expands the nodes in the best-first order.

Summary

- Best-first search
- Best-first search applied to the eight puzzle
- Best-first search applied to scheduling
- Space-saving techniques for best-first search