UCLA Samueli School of Engineering

CS161 WEEK3 DISCUSSION 1C

Danfeng Guo

Contributed by Yewen W' and Shirley C's previous course materials

Agenda

UNIFORM-COST SEARCH

INFORMED SEARCH

- A* SEARCH

(CSP NEXT WEEK)

UNIFORM-COST SERACH

- Search the node(state) that has the smallest cost
- Update the distance record for neighboring nodes
- Keep searching even after a goal is found

Algorithm 4: Uniform-Cost Algorithm

Data: A weighted directed graph $G = (V, E)$ and a source vertex s
Result: A distance map where $dist[v]$ is the shortest path weight between s and v

Function `UniformCost`(G, s)**:**
    $dist[s] = 0$;
    Create an empty vertex set Q;
    Add s into Q with key $dist[s]$;
    while $Q \neq \emptyset$ **do**
        Find vertex u in Q whose $dist[u]$ is the smallest;
        Remove u from Q;
        foreach $v \in Adj(u)$ **do**
            $dist = dist[u] + w(u, v)$;
            if $v \in Q$ **then**
                if $dist < dist[v]$ **then**
                    $dist[v] = dist$;
                    Update Q on vertex v with new $dist[v]$ value;
                end
            else
                $dist[v] = dist$;
                Add v into Q with key $dist[v]$;
            end
        end
    end
    return $dist$;

UNIFORM-COST SERACH

EVALUATION

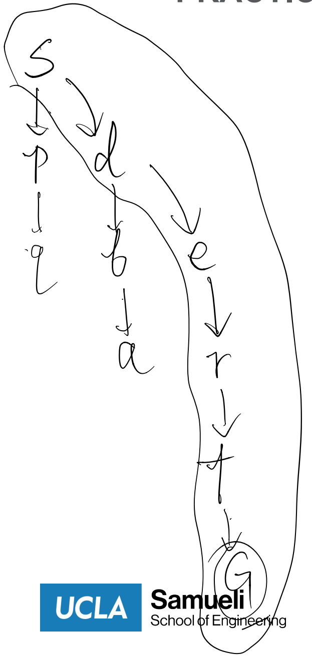
- Optimal? T
- Complete? T
- Time $O(b^{C/\epsilon})$
- Space $O(b^{C/\epsilon})$

C: Total cost

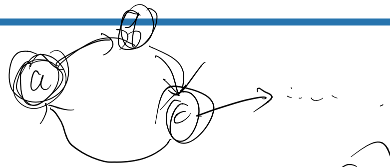
ϵ: minimum cost at each step

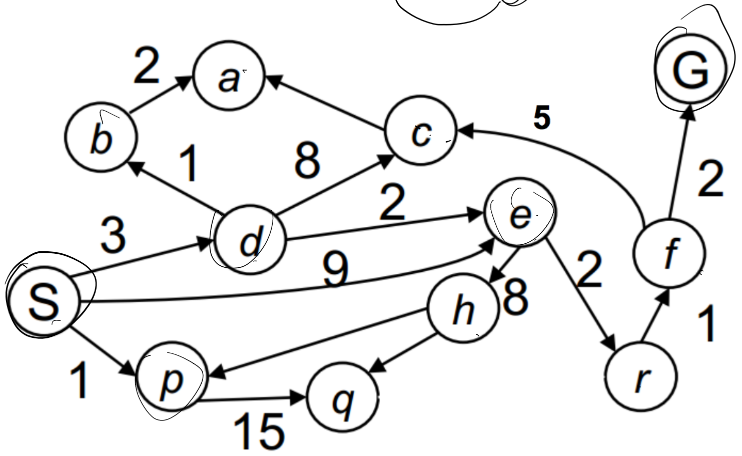
UNIFORM-COST SEARCH

PRACTICE



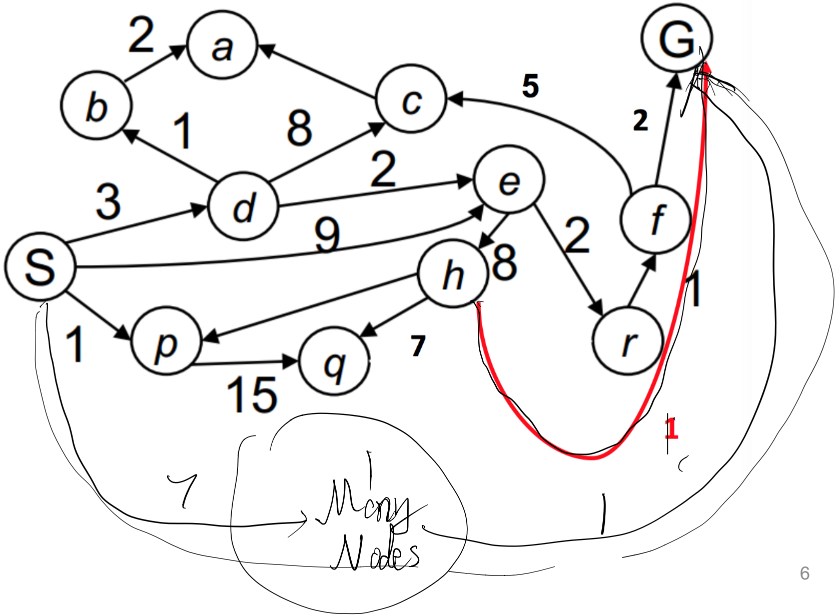
✓a 6
✓b 4
✓c 11
✓d 3
✓e 9→5
✓f 8
~~✓h 13~~
✓p 1
~~✓q 16~~
✓r 7
G





UNIFORM-COST SEARCH

PRACTICE



UNIFORM-COST SERACH

Uniform-cost search is smiliar to Dijkstra Algorithm

We must know the graph

The difference is that Dijkstra keeps the record of all nodes in the graph.

CA → DC

GREEDY BEST-FIRST SEARCH

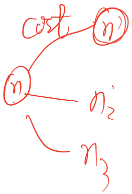
- Expand the node that is the closest to the goal
- ‘closest’ is defined by the heuristic function
- Complete? N
- Optimal? N
- Time $O(b^m)$
- Space $O(b^m)$

A* SEARCH

- Define the total cost function f(n) = g(n) + h(n)
  - g(n): path cost so far
  - h(n): estimated cost to goal (heuristic)
- Properties of heuristic
  - Admissibility $h(n) \le h^*(n)$
    - (h*(n) is the true cost from n to goal)
  - Consistency

$$h(n) \le cost(n, n') + h(n')$$

    - n’ is the successor of n; c is the cost of path from n to n’ by chossing an action a



A* SEARCH

OPTIMALITY

If h(n) is admissible, A* using tree-search is optimal

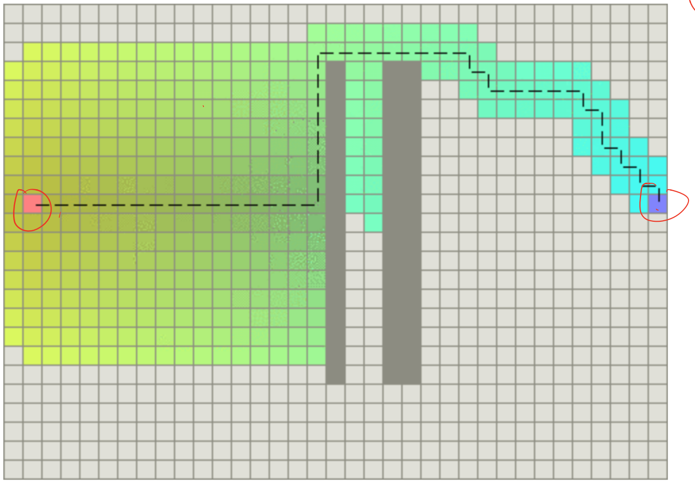
If h(n) is consistent, A* using graph-search is optimal

whether we keep a record of state/node visited before

A* SEARCH

PRACTICE

What are the possible heuristics for this question?



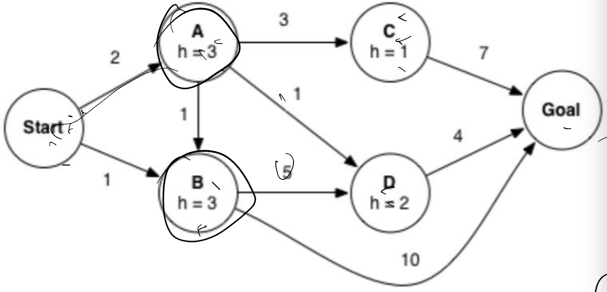
① Euclidean dist

② Manhattan dist

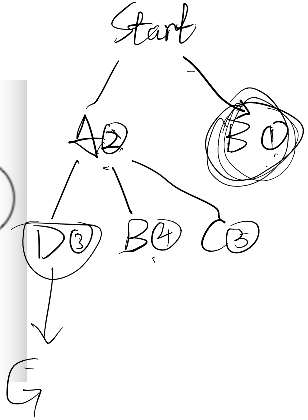
③ Diag

A* SEARCH

PRACTICE



A*



	f(n)
S→B	4
S→A	5
S-A-D	5
S-A-B	6
S-A-C	6
S-A-D-G	7

A* SEARCH

EVALUATION

Complete? -Y

Time: O(b^delta) where delta = h* - h

Space: O(b^d)

HOMEWORK2

DFID

Three parts

DFS1: Perform DFS with depth d

DFS2: Start from a depth. Add 1 to depth each step and stop until it reaches d.

DFS3: Call DFS2 with start step = 0

```
(defun dfs1 (tree depth)
  (cond ((null tree) NIL)
        ((atom tree) (list tree))
        ((= depth 0) NIL)
        (t (append (dfs1 (car tree) (- depth 1))
                   (dfs1 (cdr tree) depth)))))
```

```
(defun dfs2 (tree depth maxdepth)
(cond ((> depth maxdepth) NIL)
        (t (append (dfs tree depth) (dfs2 tree
(+ depth 1) maxdepth)))))
```

HOMEWORK2

MISSIONARY-CANNIBAL PROBLEM

MC-DFS: Given a state

- If it is a goal, add it
- If it has been visited, ignore it
- Otherwise, explore it

MULT-DFS: Given expanded states

- If nothing to expand, ignore it
- Otherwise, call MC-DFS for each state

$(S_n \; S_{n-1} \; \cdots \; S_0)$

$(S_0 \; \cdots \; S_n)$

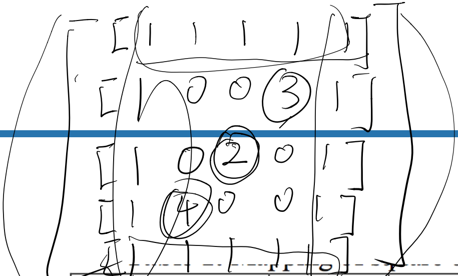
```
(defun mc-dfs (s path)
  (cond ((final-state s) (cons s path))
        ((on-path s path) nil)
        (t (mult-dfs (succ-fn s) (cons s path)))))

(defun mult-dfs (states path)
  (cond ((null states) nil)
        (t (or (mc-dfs (first states) path)
               (mult-dfs (rest states) path)))))
```

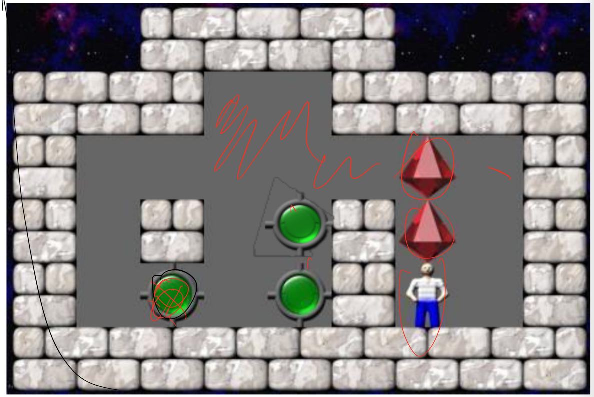
nil / nil / $(S_0 \to S_n)$

(search result of S_1) or (result of S_2) or (result of S_3) or nil

HOMEWORK3



SOKOBAN



Content	Integer	ASCII
Blank	0	' ' (white space)
Wall	1	'#'
Box	2	'$'
Keeper	3	'@'
Goal	4	'.'
Box + goal	5	'*'
Keeper + goal	6	'+'

HOMEWORK3

SOKOBAN

Break into parts

- Goal test
- Heuristic
- Action
    - Given current state
    - Move along a direction
    - Get the new state

Key function: try-move

You should consider all possible cases resulting from a move.

defun next-state ...

movement → blank

movement → goal

movement → push

Q&A