תרגול 2
אלגוריתמי חיפוש

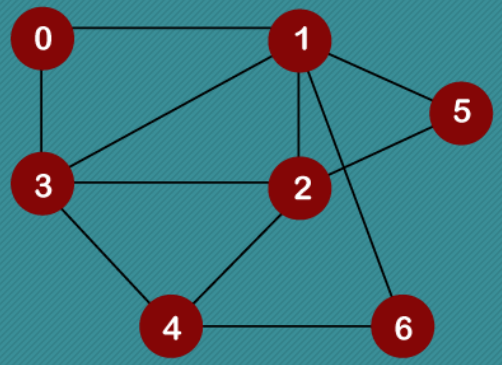
מבוא לבינה מלאכותית (236501)

מדעי המחשב, טכניון

חורף 2022-3

נושאי התרגול

- חזרה על אלגוריתמי חיפוש לא-מיודעים שנלמדו.
- אלגוריתם ID-DFS.
- שאלה ממבחן – אלגוריתמי חיפוש לא-מיודעים.
- אלגוריתם Uniform Cost Search.

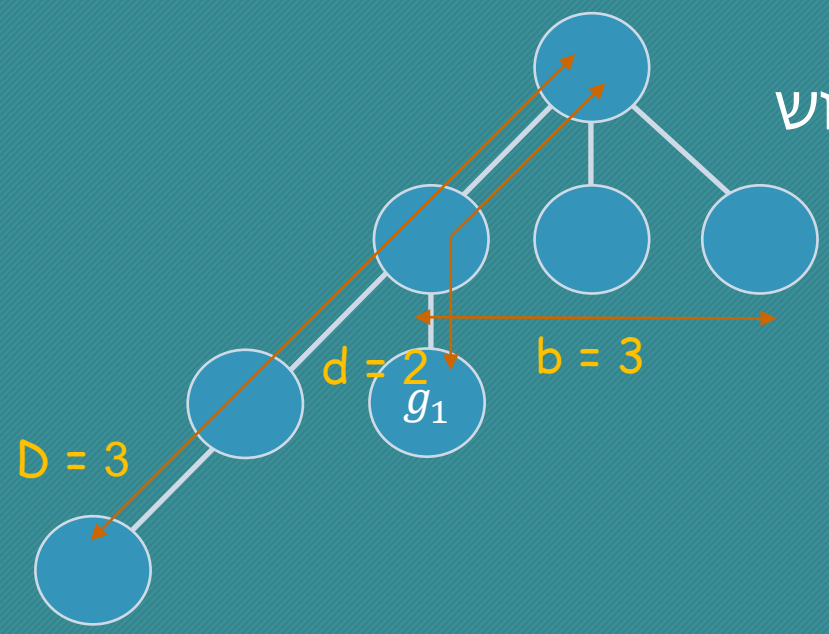


פרמטרים חשובים

b : מקדם הסיעוף (branching factor)

d : העומק של צומת המטרה הרדוד ביותר

D : העומק המקסימלי בעץ החיפוש



BFS – חיפוש לרוחב

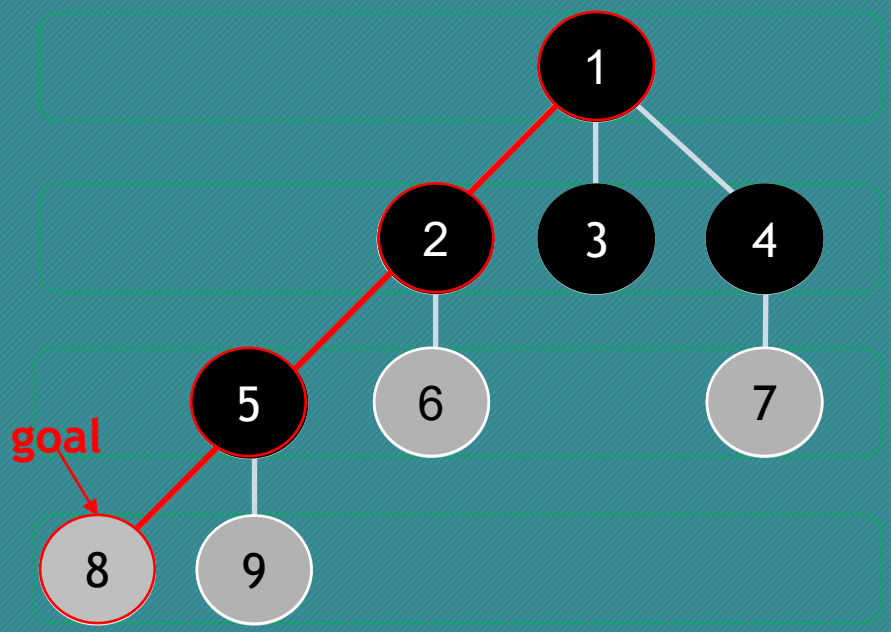
הצומת הבא לפיתוח: הצומת הרדוד ביותר.

Tree-Search

```
function Breadth-First-Search(problem):
    node ← make_node(problem.init_state, null)
    if problem.goal(node.state) then return solution(node)
    OPEN ← {node}  /* a FIFO queue with node as the only element */
    while OPEN is not empty do:
        node ← OPEN.pop()  /* chooses the shallowest node in OPEN */
        loop for s in expand(node.state):
            child ← make_node(s, node)
            if problem.goal(child.state) then return solution(child)
            OPEN.insert(child)
    return failure
```

Graph-Search

```
function Breadth-First-Search-Graph(problem):
    node ← make_node(problem.init_state, null)
    if problem.goal(node.state) then return solution(node)
    OPEN ← {node}  /* a FIFO queue with node as the only element */
    CLOSE ← {}  /* an empty set */
    while OPEN is not empty do:
        node ← OPEN.pop()  /* chooses the shallowest node in OPEN */
        CLOSE.add(node.state)
        loop for s in expand(node.state):
            child ← make_node(s, node)
            if child.state is not in CLOSE and child is not in OPEN:
                if problem.goal(child.state) then return solution(child)
                OPEN.insert(child)
    return failure
```



- צומת שנוצר
- צומת שפותח
- שכבה (לפי עומק)

תכונות BFS

האם האלגוריתם שלם?

כן. בהנחה שמספר הפעולות / מקדם הסיעוף סופי (נניח זאת כברירת מחדל).

האם האלגוריתם קביל?

כן (תחת מחיר אחיד על הקשתות). המסלול המוחזר הוא תמיד הקצר ביותר.
כל המסלולים בעומק d נבדקים לפני המסלולים בעומק d+1.

סיבוכיות זמן?

$O(b^d)$

סיבוכיות מקום?

$O(b^d)$

b = מקדם הסיעוף

d = עומק הפתרון

DFS – חיפוש לעומק

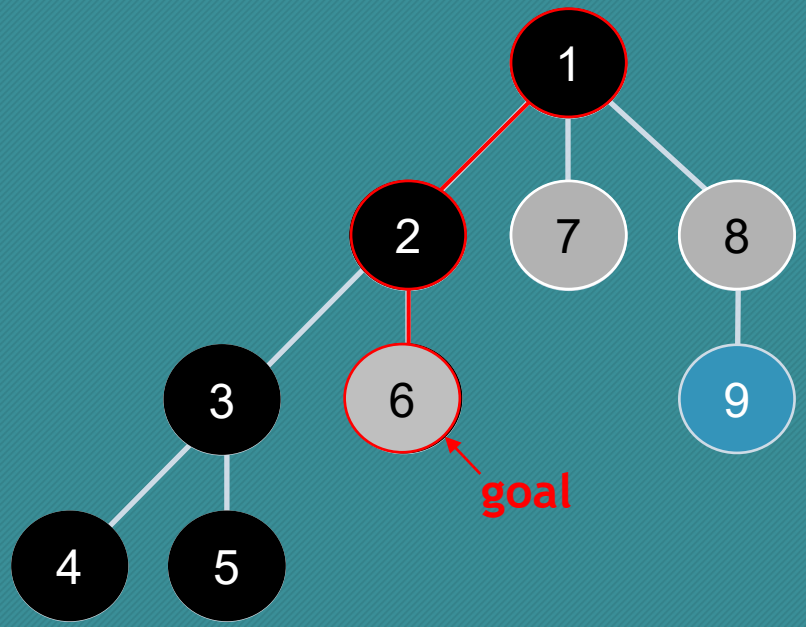
הצומת הבא לפיתוח: הצומת העמוק ביותר.

Tree-Search

```
function Depth-First-Search(problem):
        node ← make_node(problem.init_state, null)
        OPEN ← {node}  /* a LIFO queue with node as the only element */
        return Recursive-DFS(problem, OPEN)

function Recursive-DFS(problem, OPEN):
        node ← OPEN.pop()   /* chooses the deepest node in OPEN */
        if problem.goal(node.state) then return solution(node)
        loop for s in expand(node.state):
                child ← make_node(s, node)
                OPEN.insert(child)
                result ← Recursive-DFS(problem, OPEN)
                if result ≠ failure then return result
        return failure
```

Graph-Search

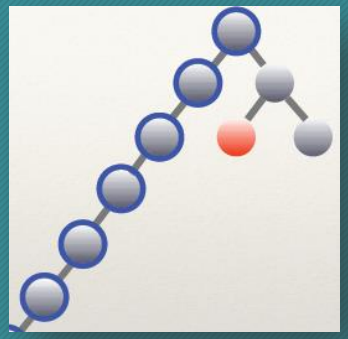
```
function Depth-First-Search-Graph(problem):
        node ← make_node(problem.init_state, null)
        OPEN ← {node}  /* a LIFO queue with node as the only element */
        CLOSE ← {}  /* an empty set */
        return Recursive-DFS(problem, OPEN, CLOSE)

function Recursive-DFS-G (problem, OPEN, CLOSE):
        node ← OPEN.pop()   /* chooses the deepest node in OPEN */
        CLOSE.add(node.state)
        if problem.goal(node.state) then return solution(node)
        loop for s in expand(node.state):
                child ← make_node(s, node)
                if child.state is not in CLOSE and child is not in OPEN:
                        OPEN.insert(child)
                        result ← Recursive-DFS-G(problem, OPEN, CLOSE)
                else: continue
                if result ≠ failure then return result
        return failure
```

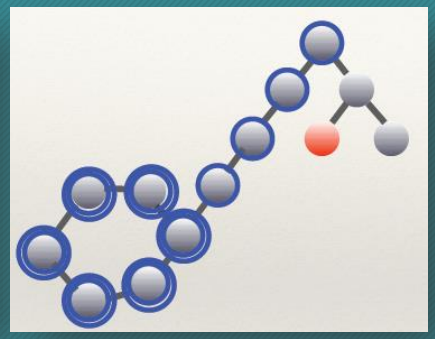


- צומת שנוצר

- צומת שפותח

תכונות DFS





האם האלגוריתם שלם?

לא. יתכנו מעגלים, יתכן מסלול אינסופי.

האם האלגוריתם קביל? (תחת מחיר אחיד על הקשתות)

לא. שלמות היא תנאי הכרחי לקבילות.

בשלילה: אם היה קביל אזי היה גם שלם - סתירה.

סיבוכיות זמן?

$O(b^D)$

סיבוכיות מקום?

$O(bD)$

b = מקדם הסיעוף

D = העומק המקסימלי בעץ החיפוש

חיפוש לעומק – Backtracking (עצל)

Tree-Search

```
function Depth-First-Search(problem):
        node ← make_node(problem.init_state, null)
        OPEN ← {node}  /* a LIFO queue with node as the only element */
        return Recursive-DFS(problem, OPEN)

function Recursive-DFS(problem, OPEN):
        node ← OPEN.pop()   /* chooses the deepest node in OPEN */
        if problem.goal(node.state) then return solution(node)
        loop for s in lazy_expand(node.state):
                child ← make_node(s, node)
                OPEN.insert(child)
                result ← Recursive-DFS(problem, OPEN)
                if result ≠ failure then return result
        return failure
```

זהה ל-DFS, אך יוצר צמתים עוקבים רק **מיד** לפני פיתוח שלהם (יצירה עצלה) ולכן חוסך בזיכרון מקום.

כאן ההבדל לעומת DFS:
פונקציית העוקב מייצרת כל פעם את המצב העוקב הבא לפי דרישה.

```
# a generator that yields items instead of returning a list
def firstn(n):
    num = 0
    while num < n:
        yield num
        num += 1
```

```
sum = 0
for x in firstn(1000000):
        sum += x
```

לקריאה עצמית:
זו הזדמנות טובה להכיר generators בפייתון.
המילה השמורה yield בפונקציה מאפשרת איטרציה עצלה.

תכונות Backtracking לעומת DFS

אילו מהתכונות משתנות ביחס ל-DFS?

האם האלגוריתם שלם?
לא. יתכנו מעגלים, יתכן מסלול אינסופי.

האם האלגוריתם קביל? (תחת מחיר אחיד על הקשתות)
לא. שלמות היא תנאי הכרחי לקבילות.
בשלילה: אם היה קביל אזי היה גם שלם - סתירה.

סיבוכיות זמן? $O(b^D)$
סיבוכיות מקום? $\cancel{O(bD)}$ $\boldsymbol{O(D)}$

b = מקדם הסיעוף
D = העומק המקסימלי בעץ החיפוש

DFS-L – חיפוש לעומק מוגבל

מה ההבדלים לעומת DFS?

Tree-Search

```
function DFS-L (problem, depth):
    node ← make_node(problem.init_state, null)
    OPEN ← {node}  /* a LIFO queue with node as the only element */
    return Recursive- DFS-L(problem, OPEN, depth)
```

```
function Recursive- DFS-L(problem, OPEN, depth):
    node ← OPEN.pop()   /* chooses the deepest node in OPEN */
    if problem.goal(node.state) then return solution(node)
    if depth == 0 then return failure
    loop for s in expand(node.state):
        child ← make_node(s, node)
        OPEN.insert(child)
        result ← Recursive- DFS-L(problem, OPEN, depth - 1)
        if result ≠ failure then return result
    return failure
```

תכונות DFS-L לעומת DFS

אילו מהתכונות משתנות ביחס ל-DFS?

האם האלגוריתם שלם?

~~לא. יתכנו מעגלים, יתכן מסלול אינסופי.~~

לא. יתכן שהפתרון בעומק גדול מהחסם.

האם האלגוריתם קביל? (תחת מחיר אחיד על הקשתות)

לא. שלמות היא תנאי הכרחי לקבילות.

סיבוכיות זמן?

$\cancel{O(b^D)}$ $\quad$ $\boldsymbol{O(b^l)}$

סיבוכיות מקום?

$\cancel{O(bD)}$ $\quad$ $\boldsymbol{O(bl)}$

b = מקדם הסיעוף

D = העומק המקסימלי בעץ החיפוש

l = חסם העומק

סיכום אלגוריתמי חיפוש לא-מיודעים

אלגוריתם	שלמות	קבילות	סיבוכיות זמן	סיבוכיות מקום
BFS	כן	כן	$O(b^d)$	$O(b^d)$
DFS	לא	לא	$O(b^D)$	$O(bD)$
Backtracking				$O(D)$
DFS-L	לא	לא	$O(b^l)$	$O(bl)$

b = מקדם הסיעוף המקסימלי d = עומק הפתרון

D = העומק המקסימלי בעץ חיפוש l = חסם העומק

- בכל הטבלה מניחים שמקדם הסיעוף חסום (b).
- קבילות <u>BFS</u> תלויה בכך שמחיר הקשתות אחיד.
- חסמי <u>DFS</u> , <u>Backtracking</u> מניחים גם שעומק החיפוש המקסימלי חסום (D).

מה ה**חסרונות** של כל אלגוריתם?

- **BFS**

זיכרון אקספוננציאלי

- **Backtracking + DFS**

לא שלם

- **DFS-L**

לא שלם אם הפתרון "רחוק" יותר מ- L

מטרה: רוצים "הכל מהכל"!
גם **זיכרון יעיל** וגם **שלמות**.

חיפוש העמקה הדרגתית – ID-DFS

```
function Iterative-Deepening-DFS (problem):
    L ← 0
    While Not Interrupted:
        result ← DFS-L (problem, L)
        if result ≠ failure then return result
        L ← L + 1
    return failure
```

תכונות ID-DFS

האם האלגוריתם שלם?...
כן. אם קיים פתרון בעומק סופי, הוא יימצא.

האם האלגוריתם קביל? (תחת מחיר אחיד על הקשתות)
כן. המסלול המוחזר הוא תמיד הקצר ביותר.
כמו בחיפוש לרוחב, כל המסלולים בעומק d נבדקים לפני המסלולים בעומק d+1.
פתרון אופטימלי הינו בעל העומק הקטן ביותר ולכן יימצא לפני כל פתרון אחר.

סיבוכיות זמן? $O(\Sigma_{i=1}^{d} b^i) = O(b^d)$

סיבוכיות מקום? $O(bd)$

b = מקדם הסיעוף
d = עומק הפתרון

השוואת אלגוריתמי חיפוש לא-מיודעים

אלגוריתם	שלמות	קבילות	סיבוכיות זמן	סיבוכיות מקום
BFS	כן	כן	$O(b^d)$	$O(b^d)$
DFS	לא	לא	$O(b^D)$	$O(bD)$
Backtracking				$O(D)$
DFS-L	לא	לא	$O(b^l)$	$O(bl)$
ID-DFS	כן	כן	$O(b^d)$	$O(bd)$

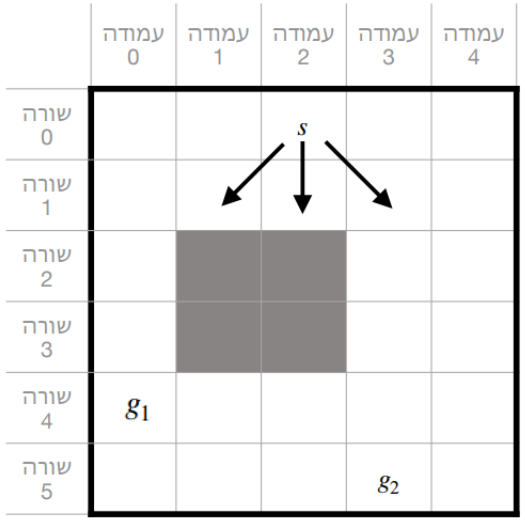
- בכל הטבלה מניחים שמקדם הסיעוף חסום (b).
- קבילות BFS , **ID-DFS** תלויה בכך שמחיר הקשתות אחיד.
- חסמי DFS , Backtracking מניחים גם שעומק החיפוש המקסימלי חסום (D).

מבוסס על שאלה ממבחן

חורף 2021-2 מועד ב'

שאלה 2 - חיפוש (18 נק')

נתון מרחב גריד בגודל 6 על 5 כאשר המצב ההתחלתי הוא בקואורדינטה (0, 2) וקיימים שני מצבים סופיים בקואורדינטות (5,0) ו-(5, 3). לנוחיותכם שמנו עותקים של הלוח בעמודים 11 ו- 12.



נתונה קבוצה של אופרטורים: $A = \{ \downarrow , \swarrow , \searrow \}$

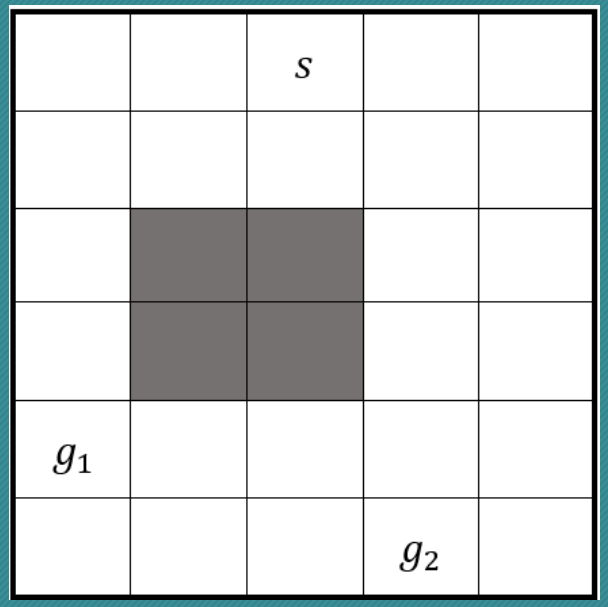
כאשר פיתוח הצמתים הינו בסדר הנתון:

1. דרום (↓)
2. דרום-מערב (↙)
3. דרום-מזרח (↘)

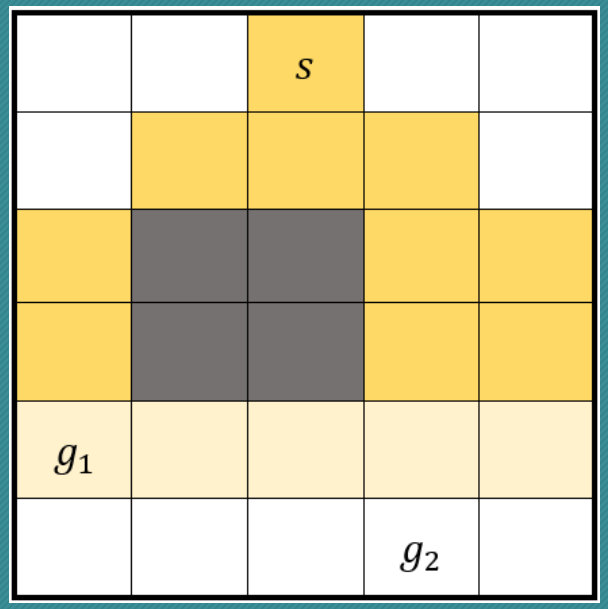
הפעולה אפשרית רק במידה והאריח שהסוכן מנסה ללכת אליו פנוי, באריחים (3, 2), (3, 1), (2, 2), (2, 1) יש בור ולכן הסוכן לא יכול לדרוך במצבים אלו.

עבור כל אחד מהאלגוריתמים הבאים, ציין את מספר המצבים השונים שפותחו, ולאיזה מצב סופי הסוכן יבחר ללכת.

אלגוריתם: BFS-G

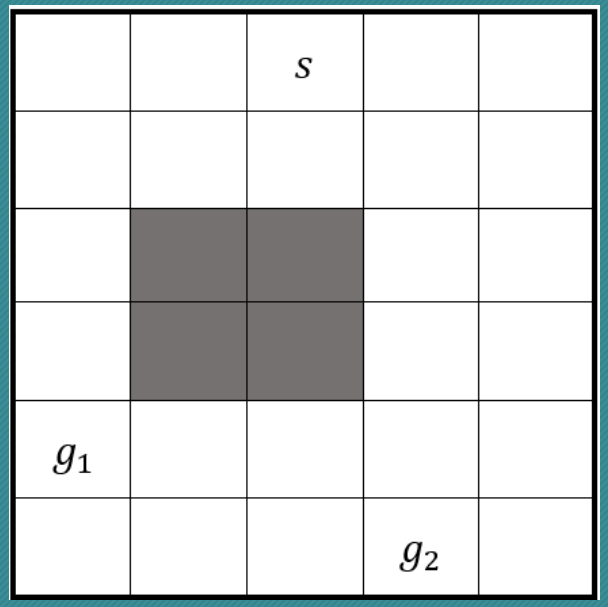


אלגוריתם: BFS-G

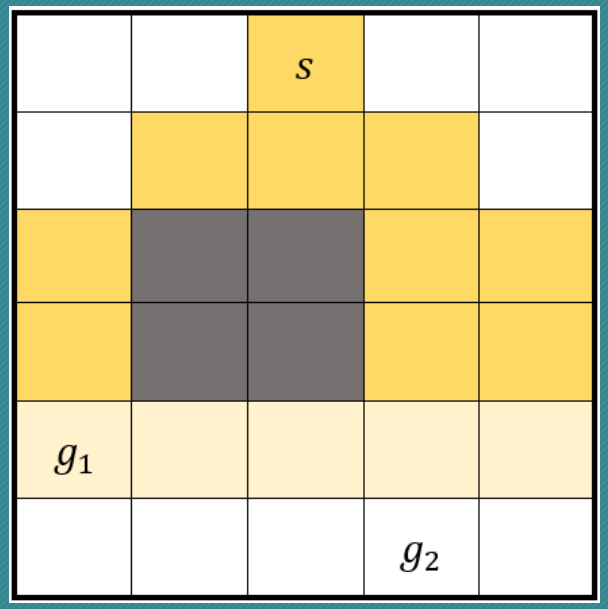


- אלגוריתם חיפוש בגרף מבטיח שאף מצב לא ייווצר פעמיים.
- המצב הסופי יימצא ביצירה של המצב הראשון מהמצב $(3,0)$.
- יפותחו בסה"כ 10 מצבים.
- המצב הסופי שיתקבל הוא g_1.
- מהו המסלול שהתקבל?

אלגוריתם: BFS

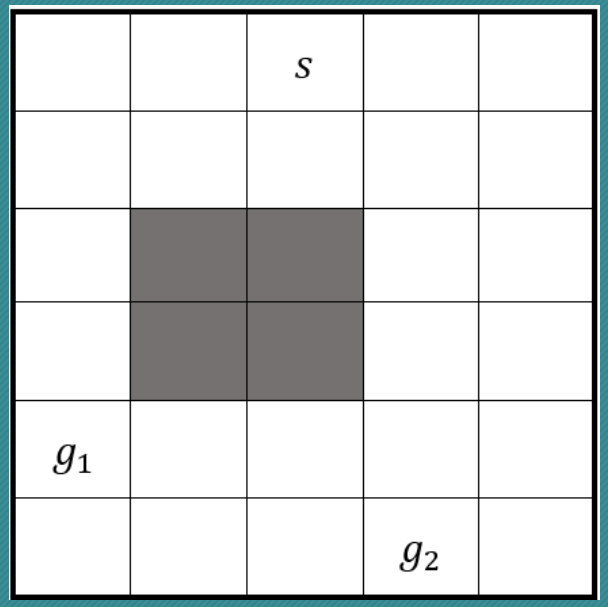


אלגוריתם: BFS



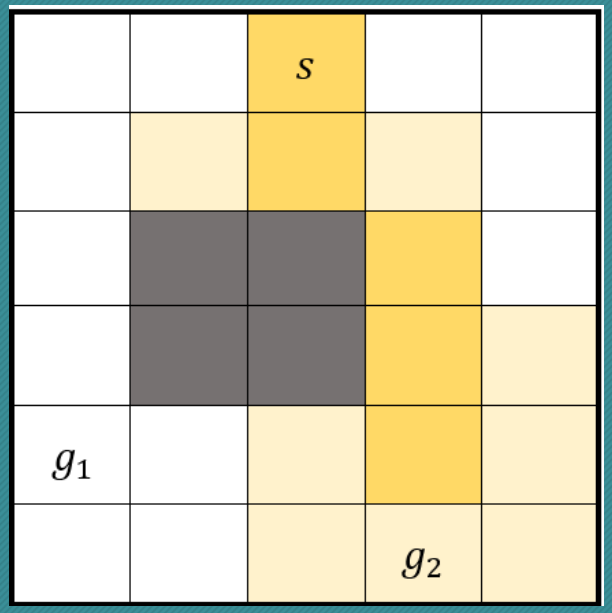
- מצב (2,3) יפותח פעמיים.
- המצב הסופי יימצא ביצירה של המצב הראשון מהמצב (3,0).
- יפותחו בסה"כ 11 מצבים.
- המצב הסופי שיתקבל הוא g_1.
- מהו המסלול שהתקבל?

אלגוריתם: DFS



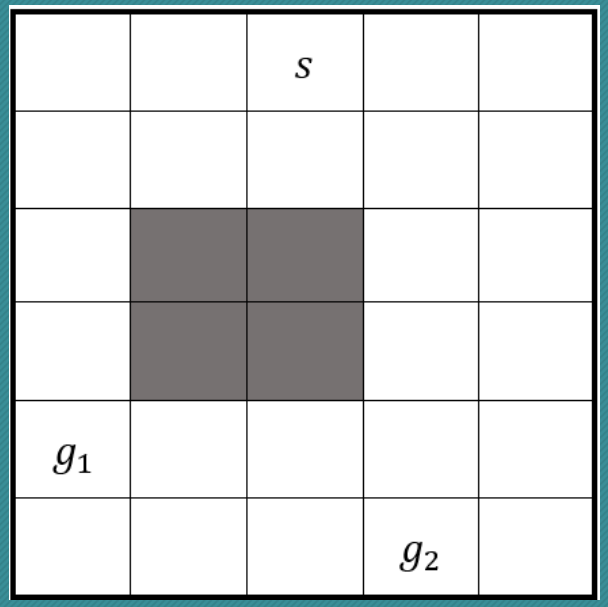
אלגוריתם: DFS

איך תיראה ריצה של
Backtracking-DFS?



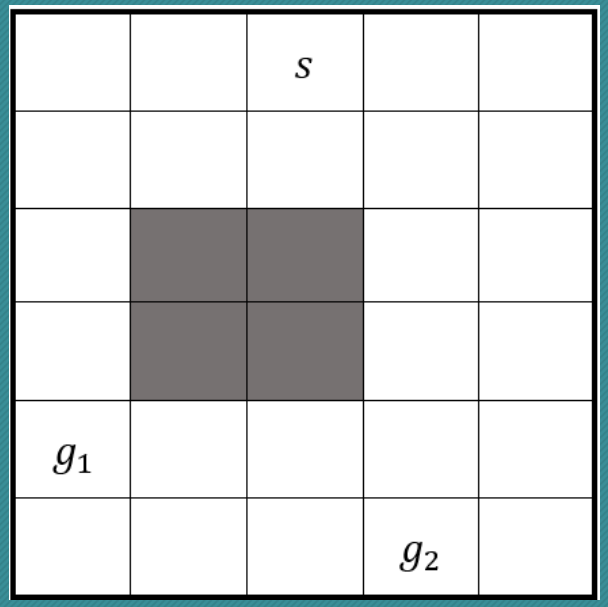
- אף מצב לא ייווצר פעמיים.
- המצב הסופי יימצא ביצירה של המצב הראשון מהמצב (4,3).
- יפותחו בסה"כ 5 מצבים.
- המצב הסופי שיתקבל הוא g_2.
- מהו המסלול שהתקבל?

אלגוריתם: DFS-G



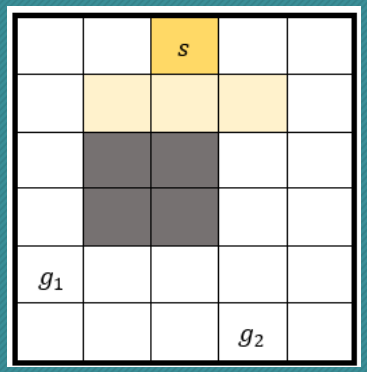
- זהה ל-DFS.

אלגוריתם: DFS-L

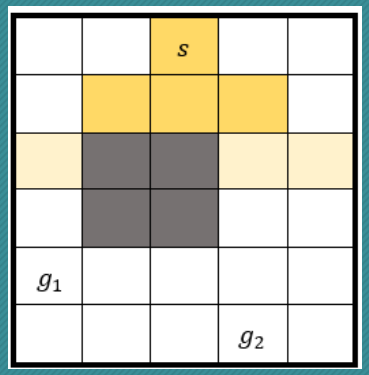


אלגוריתם: DFS-L

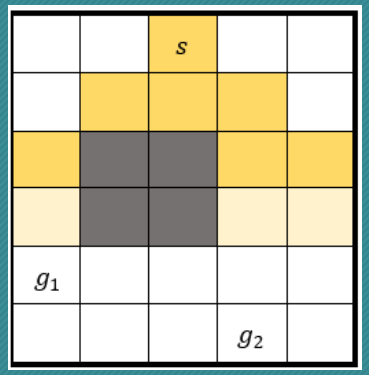
$L = 1$



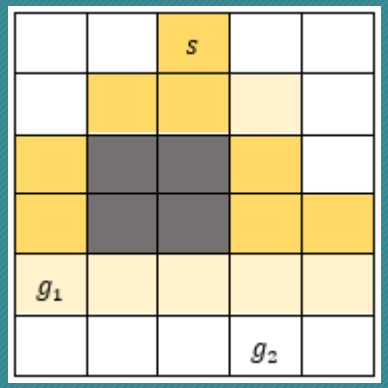
$L = 2$



$L = 3$

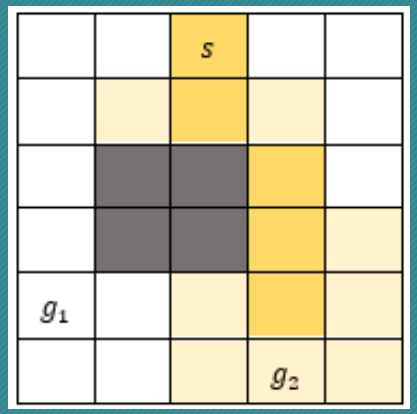


$L = 4$



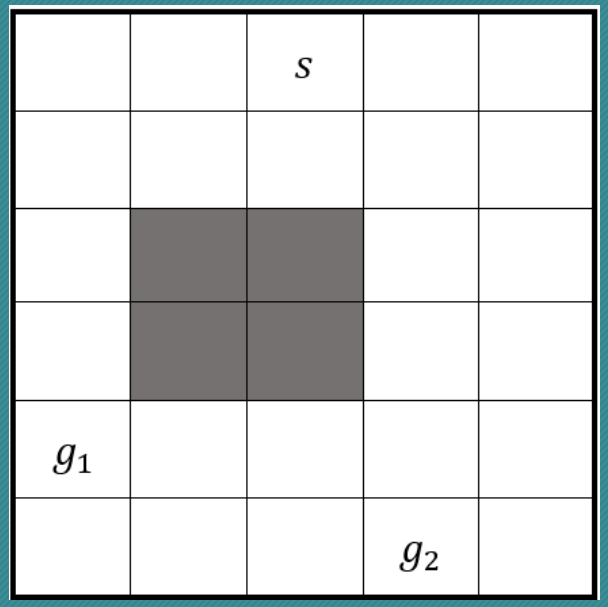
L	מספר צמתים שפותחו
1	1
2	4
3	7
4	8
≥ 5	5

$L \geq 5$



- אם $L = 4$ ימצא פתרון - g_1,
- אם $L \geq 5$ ימצא פתרון - g_2
אחרת- לא.
- מהו המסלול שהתקבל?

אלגוריתם: ID-DFS



- במידה ויש מספיק משאבים ימצא פתרון - g_1.
- מספר המצבים שיפותחו - $1 + 4 + 7 + 8 = 20\ states$

הגדרה

<u>מחיר צומת $g(v)$</u>: סך מחירי הקשתות במסלול המצטבר מצומת ההתחלה ועד לצומת v **תחת ריצה מסוימת**.
כלומר, בהינתן צומת בעץ החיפוש v, שהתקבל על ידי מסלול חיפוש

$$p_v : v_0 \to v_1 \to v_2 \to \cdots \to v_k = v \quad \text{(יחיד)}$$

$$v.g \equiv g(v) \equiv g(p_v) \equiv \sum_{edge \in p_v} cost(edge)$$

Uniform Cost Search

כאשר מחיר הקשתות **<u>אינו אחיד</u>**
בפועל: כמו **דייקסטרה** מ**אלגוריתמים 1**

Uniform Cost Search (UCS)

- שומר תור עדיפויות **OPEN** של צמתים פתוחים.
- התור **OPEN** ממוין לפי ערכי g.
  - **g** = המחיר של המסלול בעת החיפוש עד הצומת.

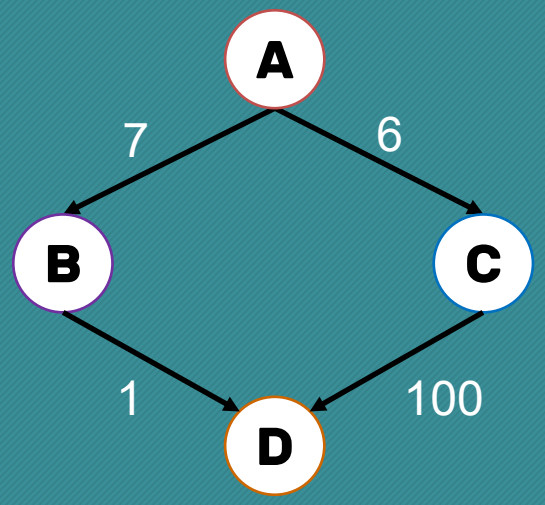
- ה**צומת** הבא לפיתוח?
  - ה**צומת** בעל **g** מינימלי ב- **OPEN**.

Uniform Cost Search (UCS)

<u>האם ייתכן</u>:
שנגלה מסלול זול יותר ל**צומת** שכבר **פותח**?
לא. <u>הסבר</u>: (מ**אלגוריתמים 1**)

בעת הוצאת צומת **v** מהתור, מחיר שאר הצמתים בתור >= **v.g**
לכן גם ל**צמתים** שנגלה מהם יהיה **מחיר** גדול יותר.

Uniform Cost Search (UCS)



<u>האם ייתכן:</u>
שנגלה מסלול זול יותר ל**צומת** שכבר ב- **open**?
(אבל עוד לא פותח)

node	A	C	B	D	D
g	0	6	7	106	8

<u>מה עושים?</u>
מעדכנים את ה**אב** של הצומת ואת ערך **g**,
ומכניס את הצומת ל- **open** מחדש.

Uniform Cost Search (UCS)

שאלה: יצרנו **צומת**. מתי נבדוק האם הוא **מטרה**?
תשובה: רק כשנוציא אותו מ- **open**.
תזכורת/חידוד: ב- BFS בדקנו זאת מיד כשיצרנו את הצומת.

Uniform Cost Search (UCS)

```
function Uniform-Cost-Search(problem):
    node ← make_node(problem.init_state, null, 0)
    OPEN ← {node}  /* a priority queue with node as the only element */
    CLOSE ← {}  /* an empty set */
    while OPEN is not empty do:
        node ← OPEN.pop() /*   The node with the minimum g is selected  */
        CLOSE.add(node.state)
        if problem.goal(node.state) then return solution(node)
        loop for s in expand(node.state):
            new_cost ← node.g + cost(node.state, s)
            child ← make_node(s, node, new_cost)
            if child.state is not in CLOSE and child is not in OPEN:
                OPEN.insert(child)
            else if child.state is in OPEN with higher g then
                replace that OPEN node with child
    return failure
```

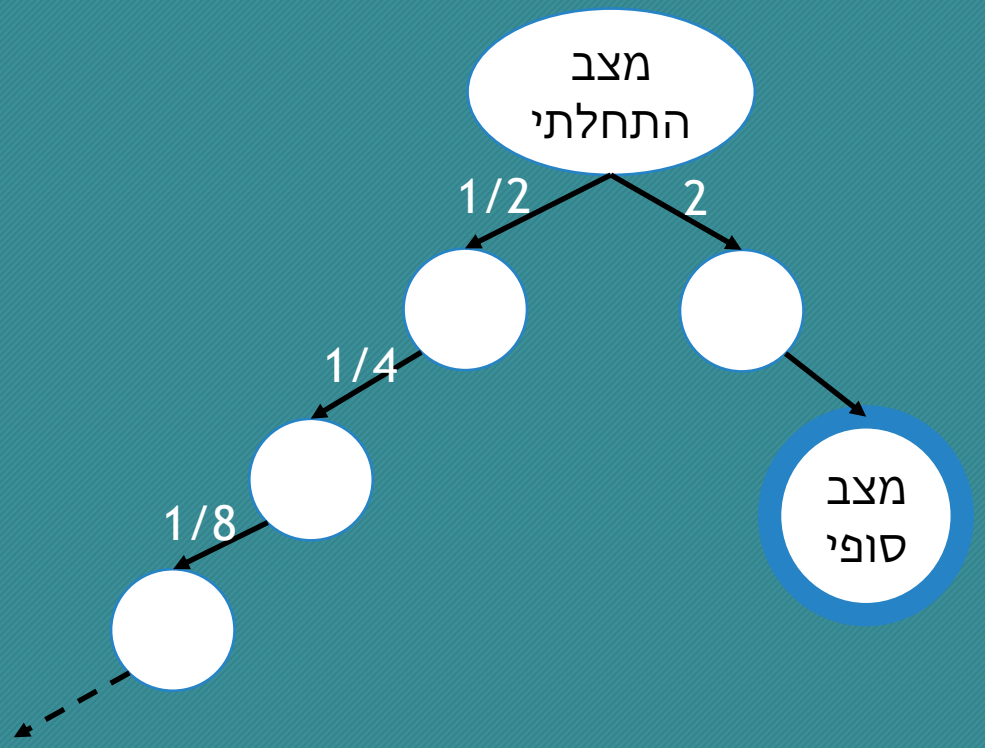
מצאו את השורה\שורות בקוד בהן האלגוריתם:

1. בוחר צומת בעל g מינימלי לפיתוח
2. נמנע מביקור בצמתים שפותחו
3. מעדכן את המחיר אם נמצא מסלול קל יותר
4. בודק האם צומת הוא מצב מטרה

Uniform Cost Search (UCS)

האם האלגוריתם **שלם**?

רק אם פונקציית ה**מחיר** חסומה **מלמטה** ע"י $\delta > 0$.

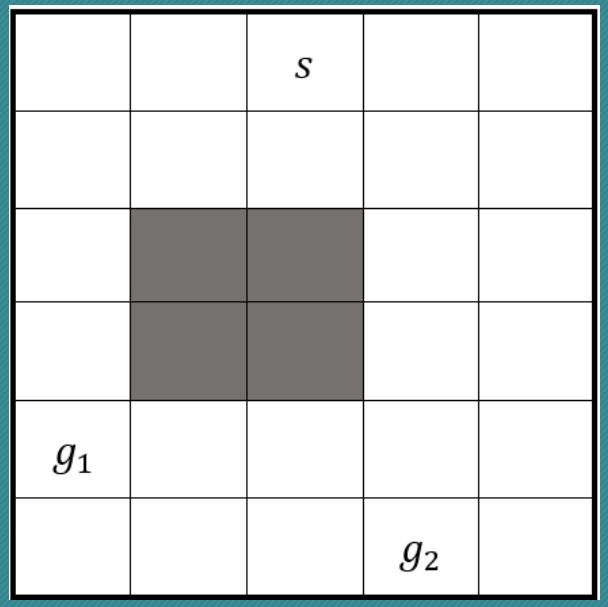


Uniform Cost Search (UCS)

האם האלגוריתם **קביל**?

- כן.
- האלגוריתם מחזיר פתרון כאשר הצומת ה**ראשון** ב- **open** הוא צומת **מטרה**.

אלגוריתם: UCS



- יפעל כמו BFS-G.

דוגמא