

Prohledávání stavového prostoru

Řešení problému prohledáváním stavového prostoru:

- **stavový prostor**, předpoklady – statické a deterministické prostředí, diskrétní stavy
- *počáteční stav* **init(State)**
- *cílová podmínka* **goal(State)**
- *přechodové akce* **move(State, NewState)**

Prohledávací strategie – **prohledávací strom**:

- *kořenový uzel*
- *uzel* prohledávacího stromu:
 - *stav*
 - *rodičovský uzel*
 - *přechodová akce*
 - *hloubka uzlu*
 - *cena* – $g(n)$ cesty, $c(x, a, y)$ přechodu
- *(optimální) řešení*

Reálné problémy řešitelné prohledáváním

- hledání cesty z města A do města B
- hledání itineráře, problém obchodního cestujícího
- návrh VLSI čipu
- navigace auta, robota, ...
- postup práce automatické výrobní linky
- návrh proteinů – 3D-sekvence aminokyselin
- Internetové vyhledávání informací

Řešení problému prohledáváním

Kostra algoritmu:

```
solution(Solution) :- init(State),solve(State,Solution).
```

```
solve(State,[State]) :- goal(State).
```

```
solve(State,[State|Sol]) :- move(State,NewState),solve(NewState,Sol).
```

move(State,NewState) – definuje prohledávací **strategii**

Porovnání strategií:

složitost závisí na:

- úplnost
- optimálnost
- časová složitost
- prostorová složitost
- b – faktor větvení (branching factor)
- d – hloubka cíle (goal depth)
- m – maximální hloubka větve/délka cesty (maximum depth/path, může být ∞ ?)

Řešení problému prohledáváním

Kostra algoritmu:

```
solution(Solution) :- init(State),solve(State,Solution).
```

```
solve(State,[State]) :- goal(State).
```

```
solve(State,[State|Sol]) :- move(State,NewState),solve(NewState,Sol).
```

move(State,NewState) – definuje prohledávací **strategii**

Porovnání strategií:

složitosť závisí na:

- úplnost
 - optimálnost
 - časová složitost
 - prostorová složitost
- b – faktor větvení (branching factor)
 - d – hloubka cíle (goal depth)
 - m – maximální hloubka větve/délka cesty (maximum depth/path, může být ∞ ?)

Řešení problému prohledáváním

Kostra algoritmu:

```
solution(Solution) :- init(State),solve(State,Solution).
```

```
solve(State,[State]) :- goal(State).
```

```
solve(State,[State|Sol]) :- move(State,NewState),solve(NewState,Sol).
```

move(State,NewState) – definuje prohledávací **strategii**

Porovnání strategií:

složitosť závisí na:

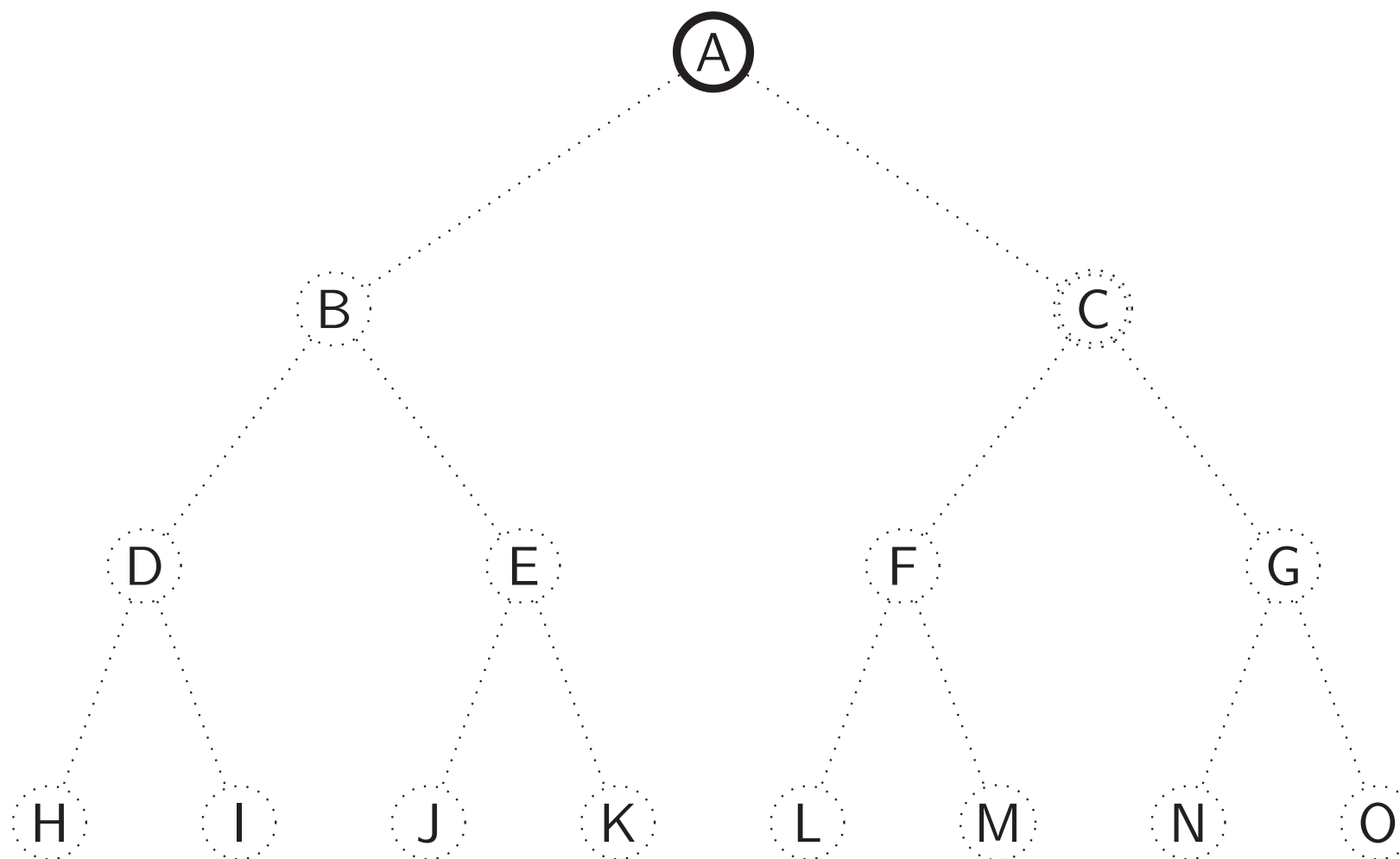
- úplnost
- optimálnost
- časová složitost
- prostorová složitost
- b – faktor **větvení** (branching factor)
- d – hloubka cíle (goal depth)
- m – maximální hloubka větve/délka cesty (maximum depth/path, může být ∞ ?)

Neinformované prohledávání

- prohledávání do hloubky
- prohledávání do hloubky s limitem
- prohledávání do šířky
- prohledávání podle ceny
- prohledávání s postupným prohlubováním

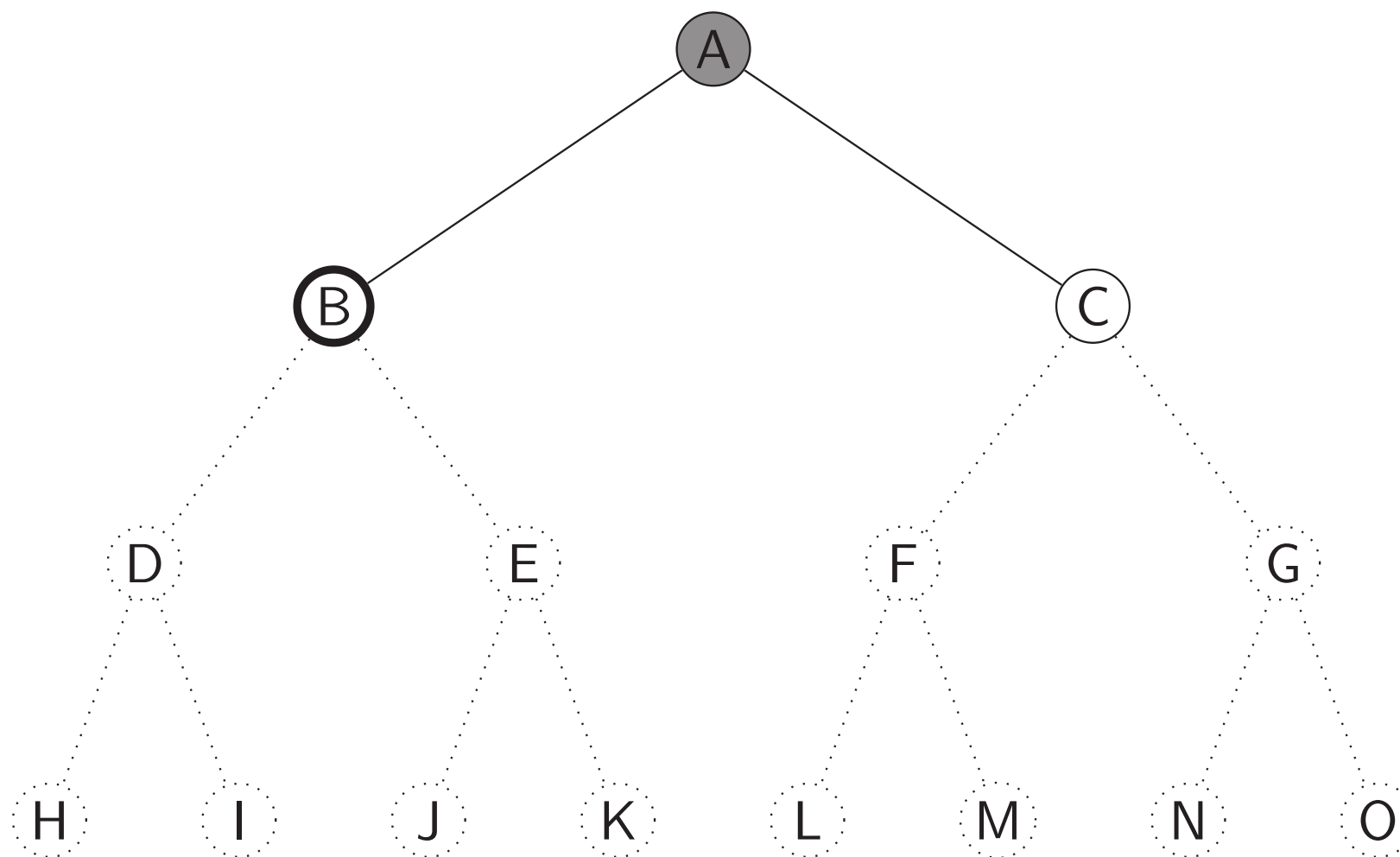
Prohledávání do hloubky

Prohledává se vždy nejlevější a nejhlubší neexpandovaný uzel (*Depth-first Search, DFS*)



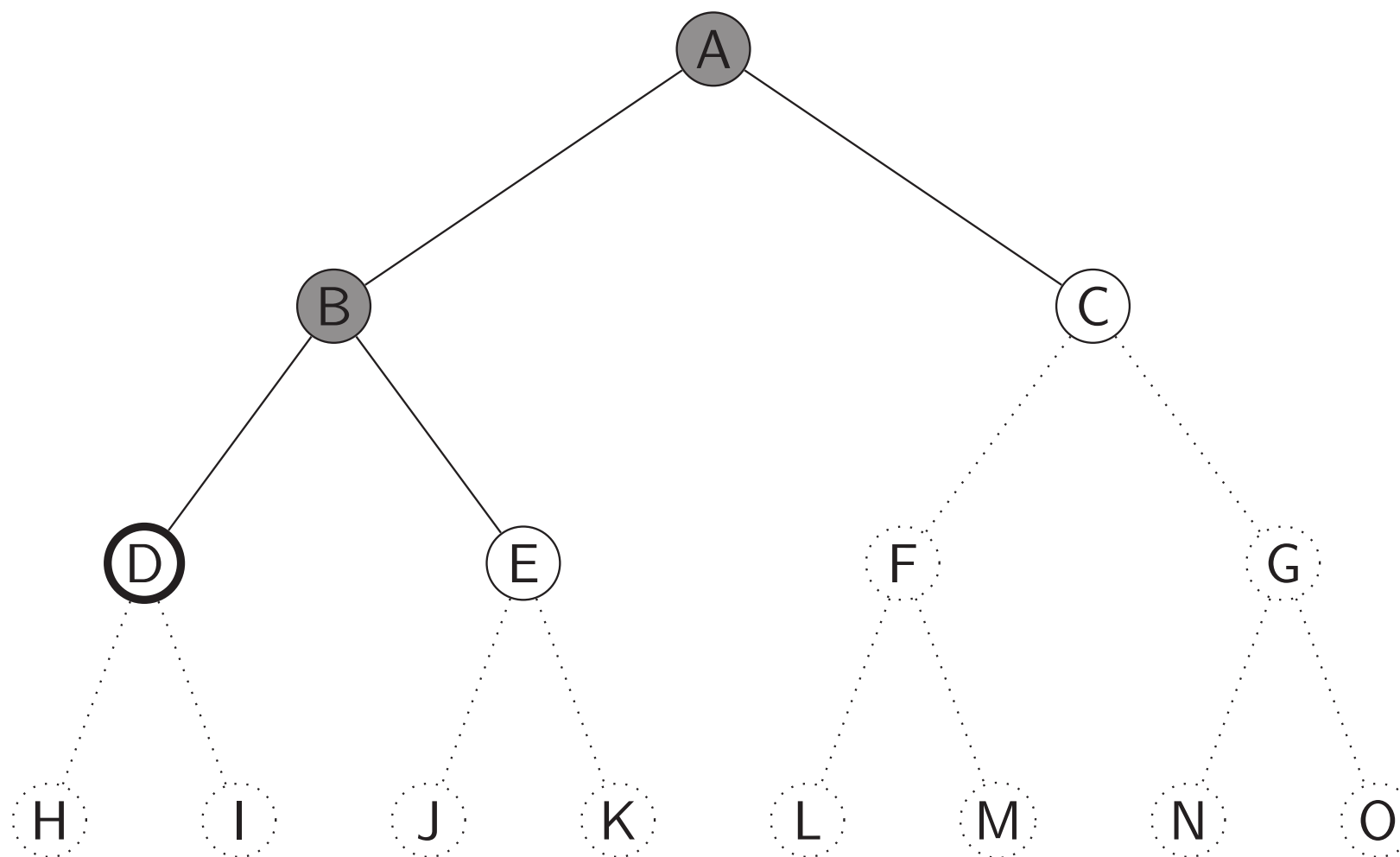
Prohledávání do hloubky

Prohledává se vždy nejlevější a nejhlubší neexpandovaný uzel (*Depth-first Search, DFS*)



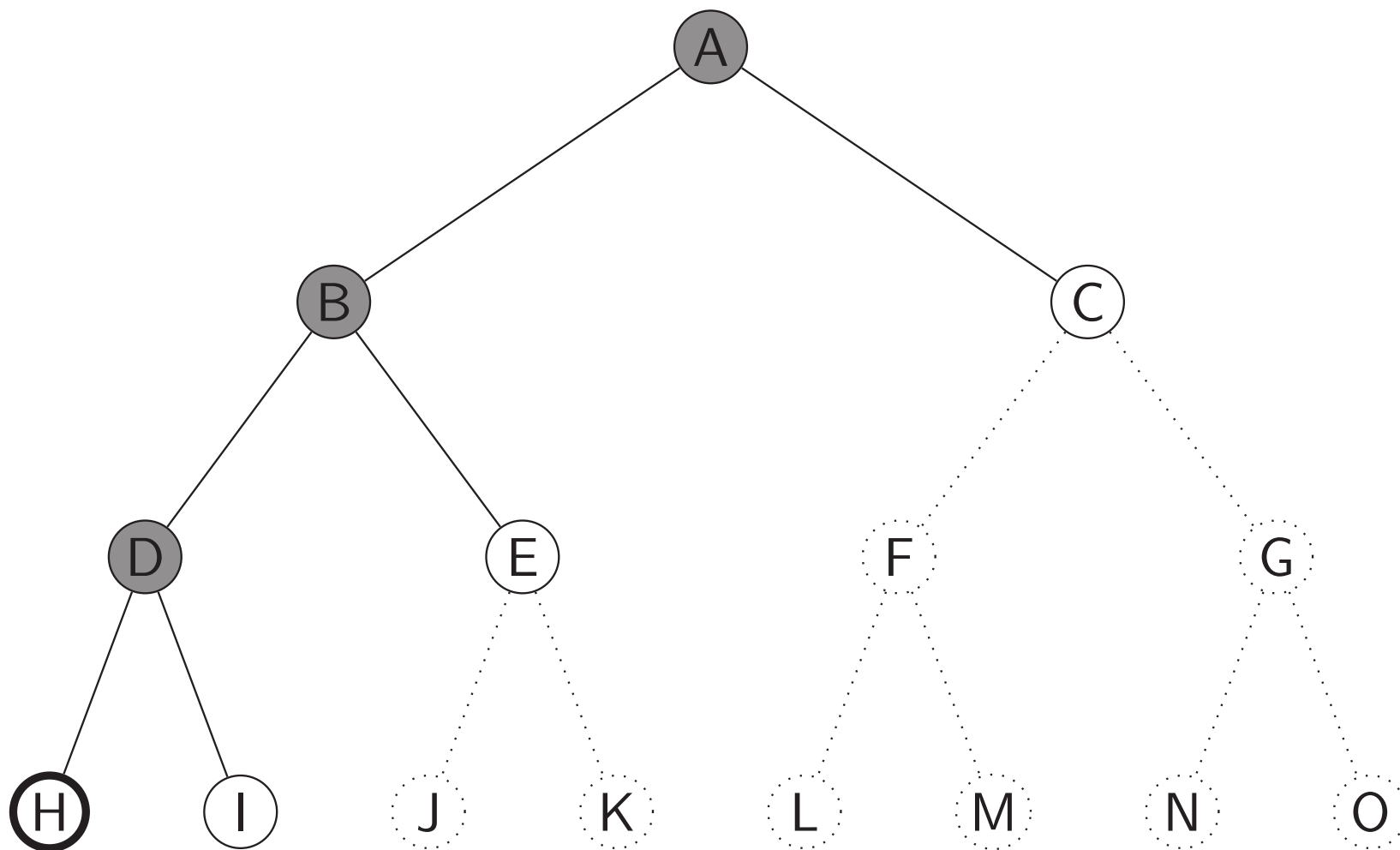
Prohledávání do hloubky

Prohledává se vždy nejlevější a nejhlubší neexpandovaný uzel (*Depth-first Search, DFS*)



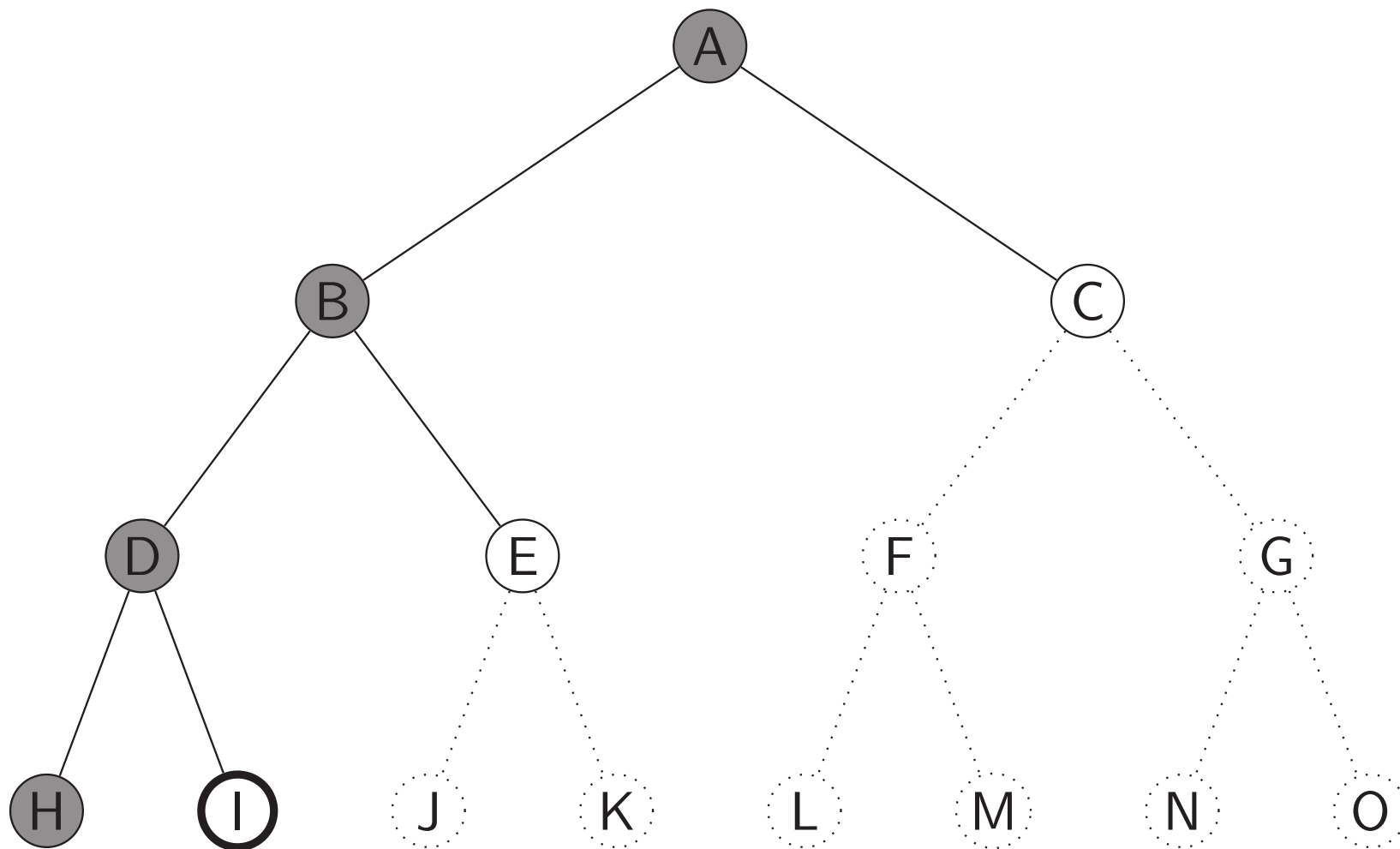
Prohledávání do hloubky

Prohledává se vždy nejlevější a nejhlubší neexpandovaný uzel (*Depth-first Search, DFS*)



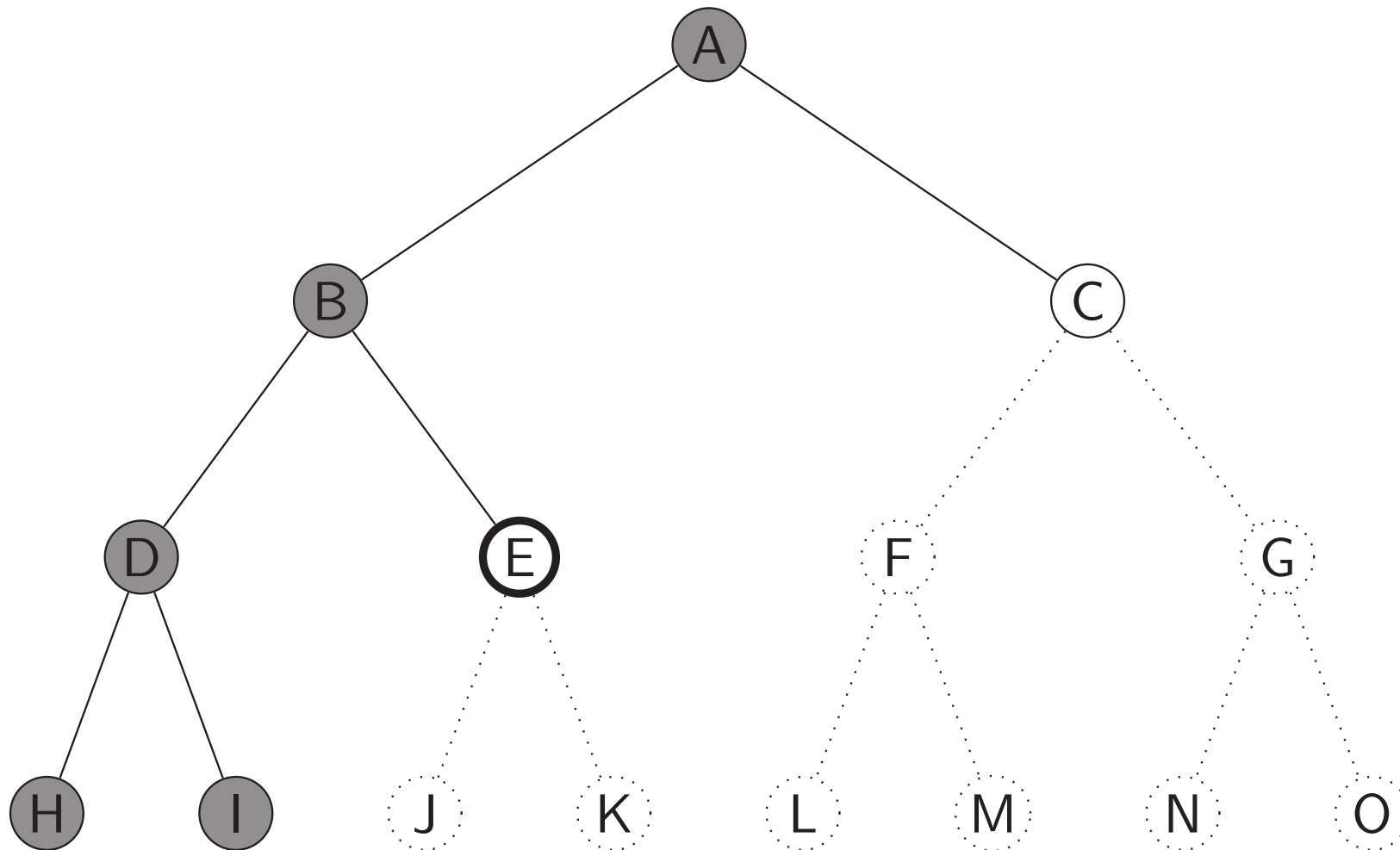
Prohledávání do hloubky

Prohledává se vždy nejlevější a nejhlubší neexpandovaný uzel (*Depth-first Search, DFS*)



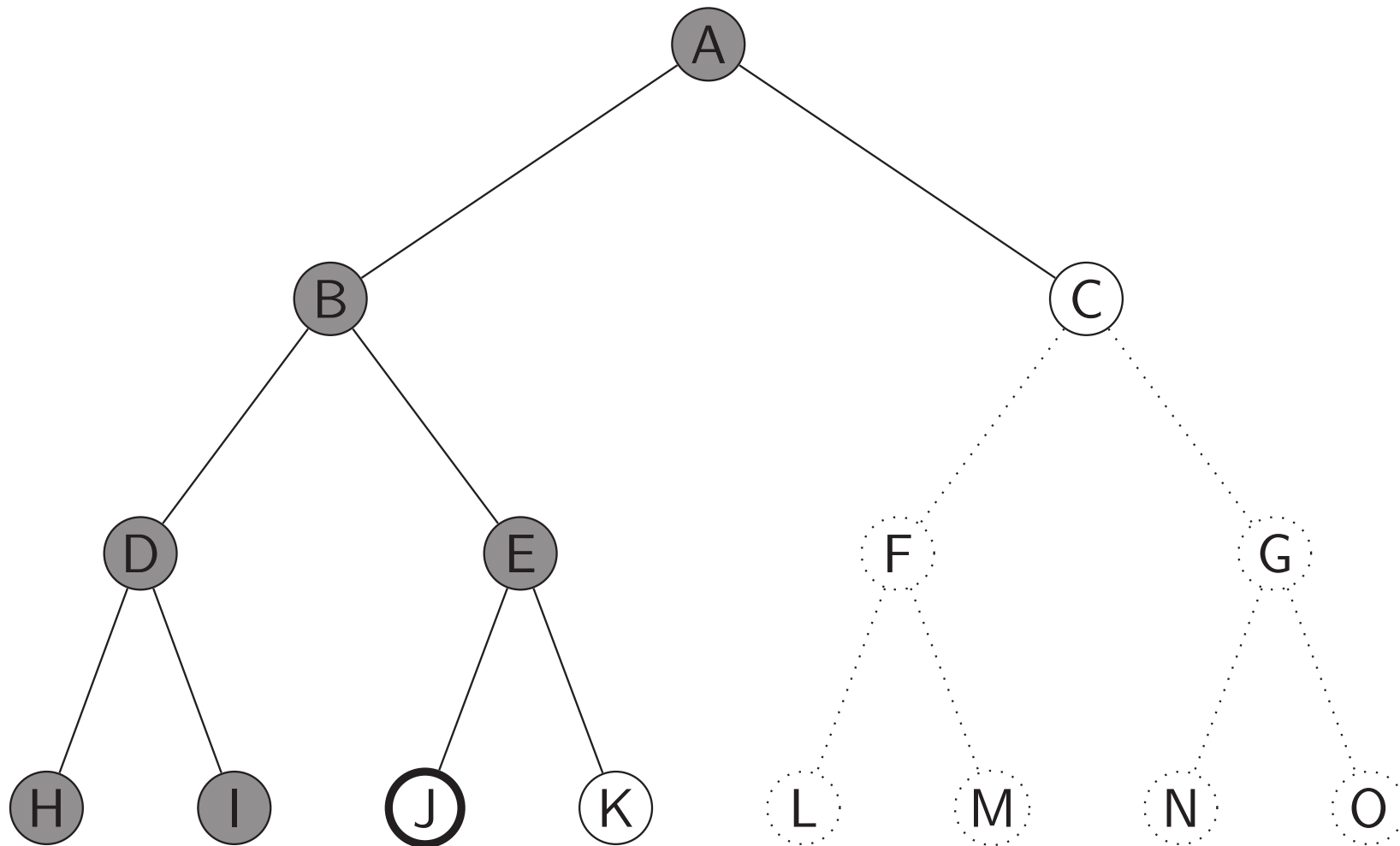
Prohledávání do hloubky

Prohledává se vždy nejlevější a nejhlubší neexpandovaný uzel (*Depth-first Search, DFS*)



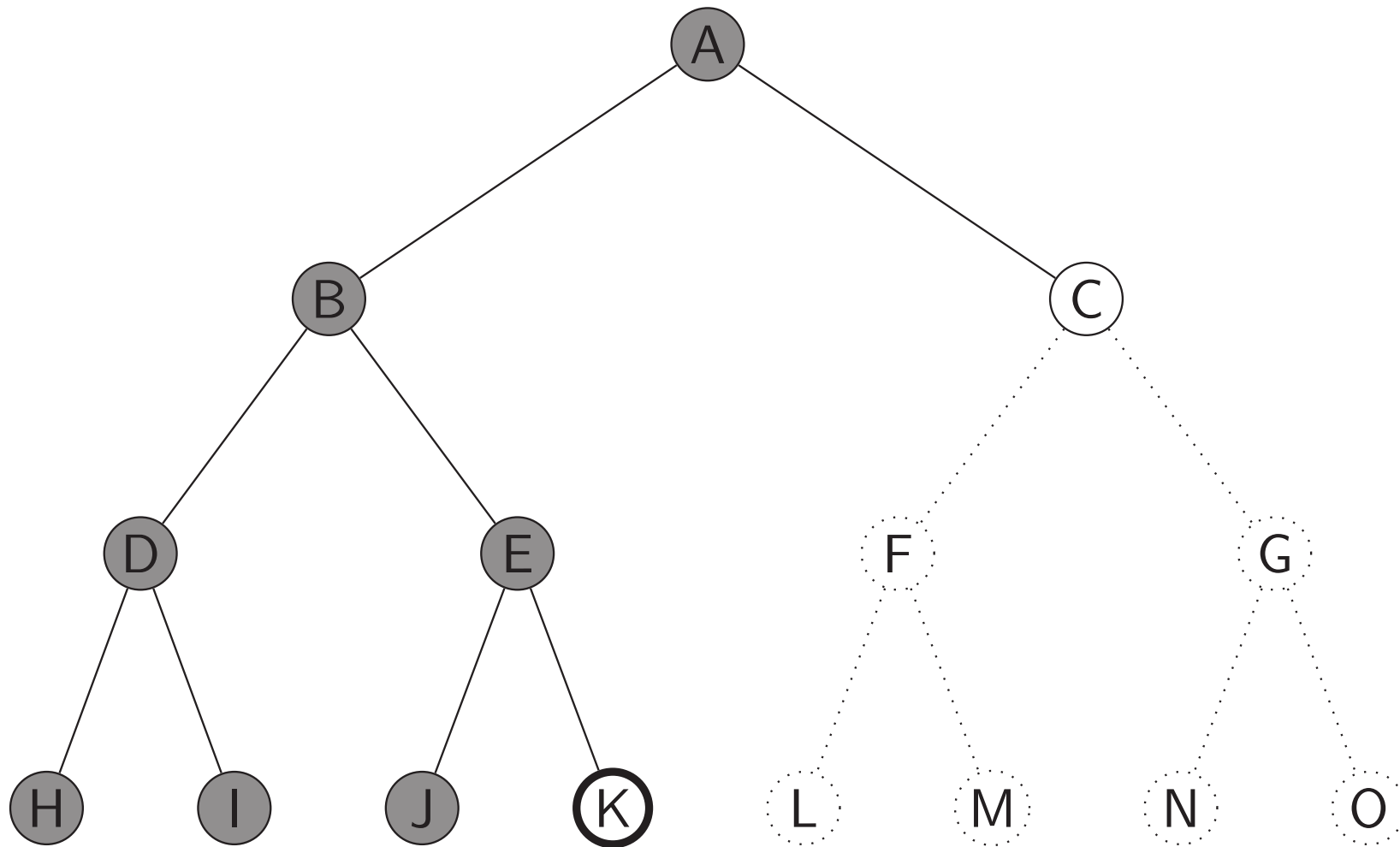
Prohledávání do hloubky

Prohledává se vždy nejlevější a nejhlubší neexpandovaný uzel (*Depth-first Search, DFS*)



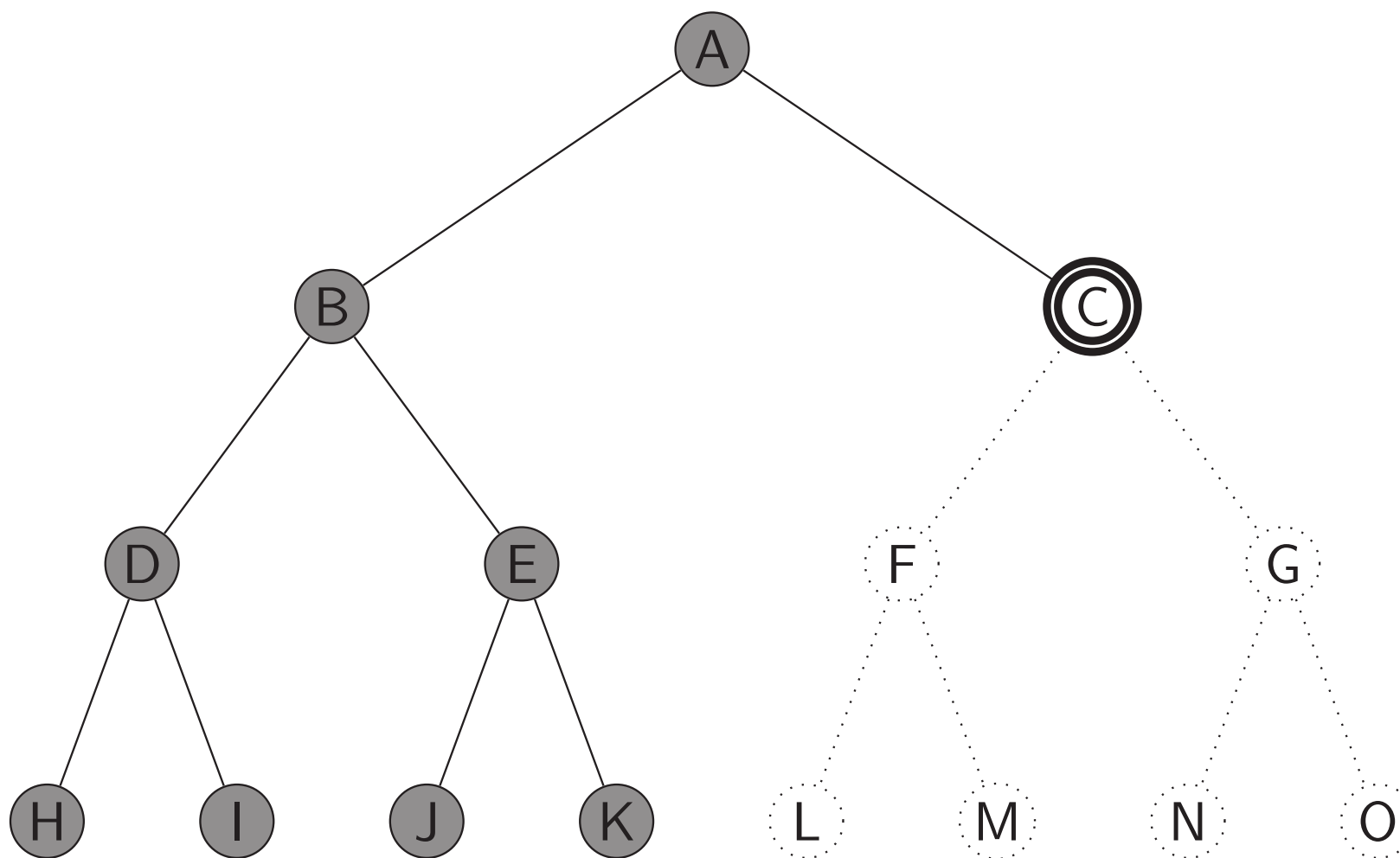
Prohledávání do hloubky

Prohledává se vždy nejlevější a nejhlubší neexpandovaný uzel (*Depth-first Search, DFS*)



Prohledávání do hloubky

Prohledává se vždy nejlevější a nejhlubší neexpandovaný uzel (*Depth-first Search, DFS*)



Prohledávání do hloubky

procedurální programovací jazyk – uzly se uloží do **zásobníku** (fronty LIFO) × Prolog – využití **rekurze**

```
solution(Node,Solution) :- depth_first_search([],Node,Solution).
```

```
depth_first_search(Path,Node,[Node|Path]) :- goal(Node).
```

```
depth_first_search(Path,Node,Sol) :- move(Node,Node1),  
    \+ member(Node1,Path),depth_first_search([Node|Path],Node1,Sol).
```


Prohledávání do hloubky

procedurální programovací jazyk – uzly se uloží do **zásobníku** (fronty LIFO) × Prolog – využití **rekurze**

```
solution(Node,Solution) :- depth_first_search([],Node,Solution).
```

```
depth_first_search(Path,Node,[Node|Path]) :- goal(Node).
```

```
depth_first_search(Path,Node,Sol) :- move(Node,Node1),  
    \+ member(Node1,Path),depth_first_search([Node|Path],Node1,Sol).
```

Prohledávání do hloubky – vlastnosti

<i>úplnost</i>	není úplný (nekonečná větev, cykly)
<i>optimálnost</i>	není optimální
<i>časová složitost</i>	$O(b^m)$
<i>prostorová složitost</i>	$O(bm)$, lineární

Největší problém – nekonečná větev = nenajde se cíl, program neskončí!

Prohledávání do hloubky – vlastnosti

<i>úplnost</i>	není úplný (nekonečná větev, cykly)
<i>optimálnost</i>	není optimální
<i>časová složitost</i>	$O(b^m)$
<i>prostorová složitost</i>	$O(bm)$, lineární

Největší problém – nekonečná větev = nenajde se cíl, program neskončí!

Prohledávání do hloubky – vlastnosti

<i>úplnost</i>	není úplný (nekonečná větev, cykly)
<i>optimálnost</i>	není optimální
<i>časová složitost</i>	$O(b^m)$
<i>prostorová složitost</i>	$O(bm)$, lineární

Největší problém – nekonečná větev = nenajde se cíl, program neskončí!

Prohledávání do hloubky – vlastnosti

<i>úplnost</i>	není úplný (nekonečná větev, cykly)
<i>optimálnost</i>	není optimální
<i>časová složitost</i>	$O(b^m)$
<i>prostorová složitost</i>	$O(bm)$, lineární

Největší problém – nekonečná větev = nenajde se cíl, program neskončí!

Prohledávání do hloubky – vlastnosti

<i>úplnost</i>	není úplný (nekonečná větev, cykly)
<i>optimálnost</i>	není optimální
<i>časová složitost</i>	$O(b^m)$
<i>prostorová složitost</i>	$O(bm)$, lineární

Největší problém – nekonečná větev = nenajde se cíl, program neskončí!

Prohledávání do hloubky – vlastnosti

<i>úplnost</i>	není úplný (nekonečná větev, cykly)
<i>optimálnost</i>	není optimální
<i>časová složitost</i>	$O(b^m)$
<i>prostorová složitost</i>	$O(bm)$, lineární

Největší problém – nekonečná větev = nenajde se cíl, program neskončí!

Prohledávání do hloubky s limitem

Řešení nekonečné větve – použití “zarážky” = limit hloubky ℓ

```
solution(Node,Solution) :- depth_first_search_limit(Node,Solution, $\ell$ ).
```

```
depth_first_search_limit(Node,[Node],_) :- goal(Node).
```

```
depth_first_search_limit(Node,[Node|Sol],MaxDepth) :- MaxDepth>0,  
    move(Node,Node1), Max1 is MaxDepth-1,  
    depth_first_search_limit(Node1,Sol,Max1).
```

neúspěch (**fail**) má dvě možné interpretace – vyčerpání limitu nebo neexistenci řešení

Vlastnosti:

<i>úplnost</i>	není úplný (pro $\ell < d$)
<i>optimálnost</i>	není optimální (pro $\ell > d$)
<i>časová složitost</i>	$O(b^\ell)$
<i>prostorová složitost</i>	$O(b\ell)$

dobrá volba limitu ℓ – podle znalosti problému

Prohledávání do hloubky s limitem

Řešení nekonečné větve – použití “zarážky” = limit hloubky ℓ

```
solution(Node,Solution) :- depth_first_search_limit(Node,Solution, $\ell$ ).
```

```
depth_first_search_limit(Node,[Node],_) :- goal(Node).
```

```
depth_first_search_limit(Node,[Node|Sol],MaxDepth) :- MaxDepth>0,  
    move(Node,Node1), Max1 is MaxDepth-1,  
    depth_first_search_limit(Node1,Sol,Max1).
```

neúspěch (**fail**) má dvě možné interpretace – vyčerpání limitu nebo neexistenci řešení

Vlastnosti:

<i>úplnost</i>	není úplný (pro $\ell < d$)
<i>optimálnost</i>	není optimální (pro $\ell > d$)
<i>časová složitost</i>	$O(b^\ell)$
<i>prostorová složitost</i>	$O(b\ell)$

dobrá volba limitu ℓ – podle znalosti problému

Prohledávání do hloubky s limitem

Řešení nekonečné větve – použití “zarážky” = limit hloubky ℓ

```
solution(Node,Solution) :- depth_first_search_limit(Node,Solution, $\ell$ ).
```

```
depth_first_search_limit(Node,[Node],_) :- goal(Node).
```

```
depth_first_search_limit(Node,[Node|Sol],MaxDepth) :- MaxDepth>0,  
    move(Node,Node1), Max1 is MaxDepth-1,  
    depth_first_search_limit(Node1,Sol,Max1).
```

neúspěch (**fail**) má dvě možné interpretace – vyčerpání limitu nebo neexistenci řešení

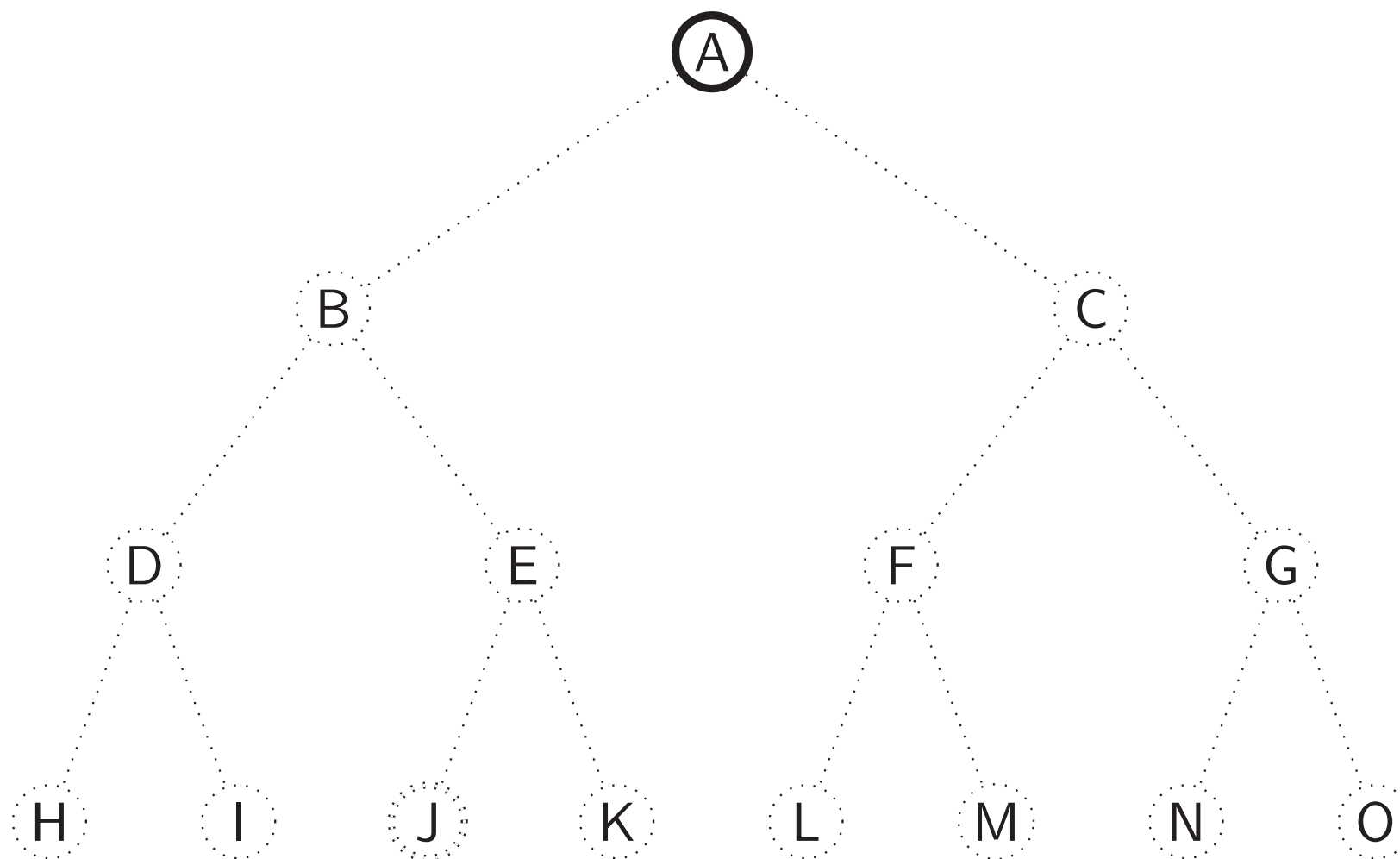
Vlastnosti:

<i>úplnost</i>	není úplný (pro $\ell < d$)
<i>optimálnost</i>	není optimální (pro $\ell > d$)
<i>časová složitost</i>	$O(b^\ell)$
<i>prostorová složitost</i>	$O(b\ell)$

dobrá volba limitu ℓ – podle znalosti problému

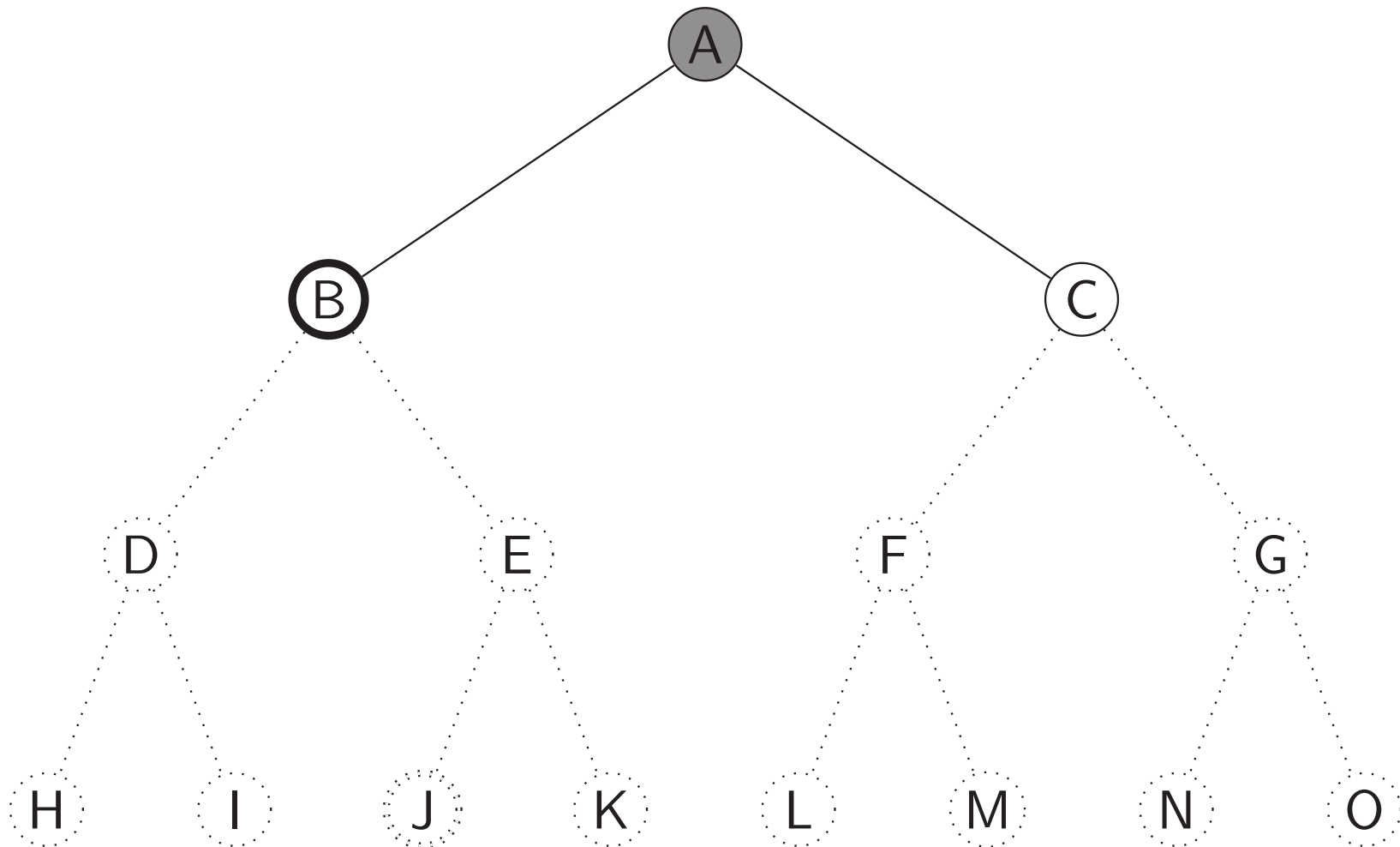
Prohledávání do šířky

Prohledává se vždy nejlevější neexpandovaný uzel s nejmenší hloubkou.
(*Breadth-first Search, BFS*)



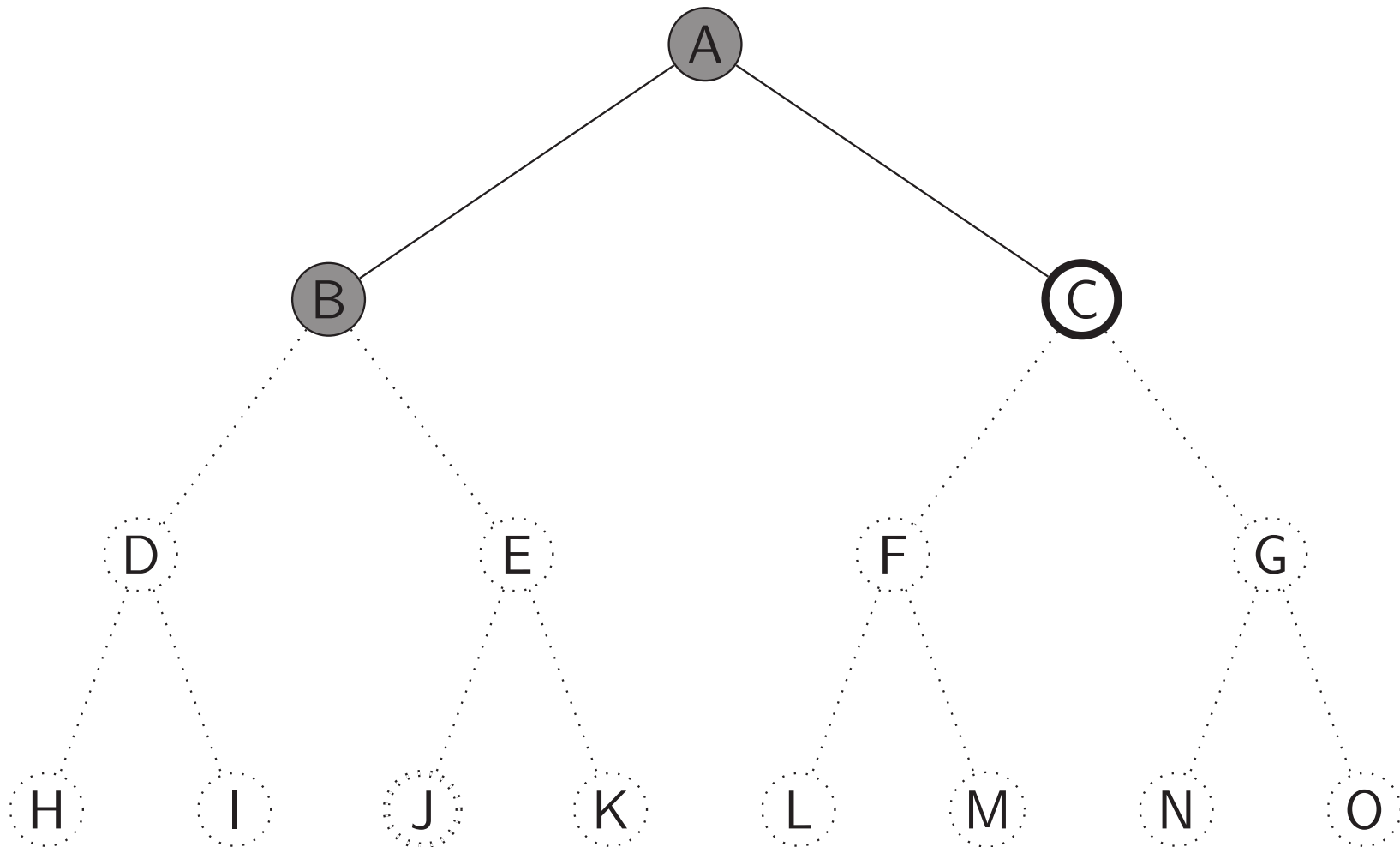
Prohledávání do šířky

Prohledává se vždy nejlevější neexpandovaný uzel s nejmenší hloubkou.
(*Breadth-first Search, BFS*)



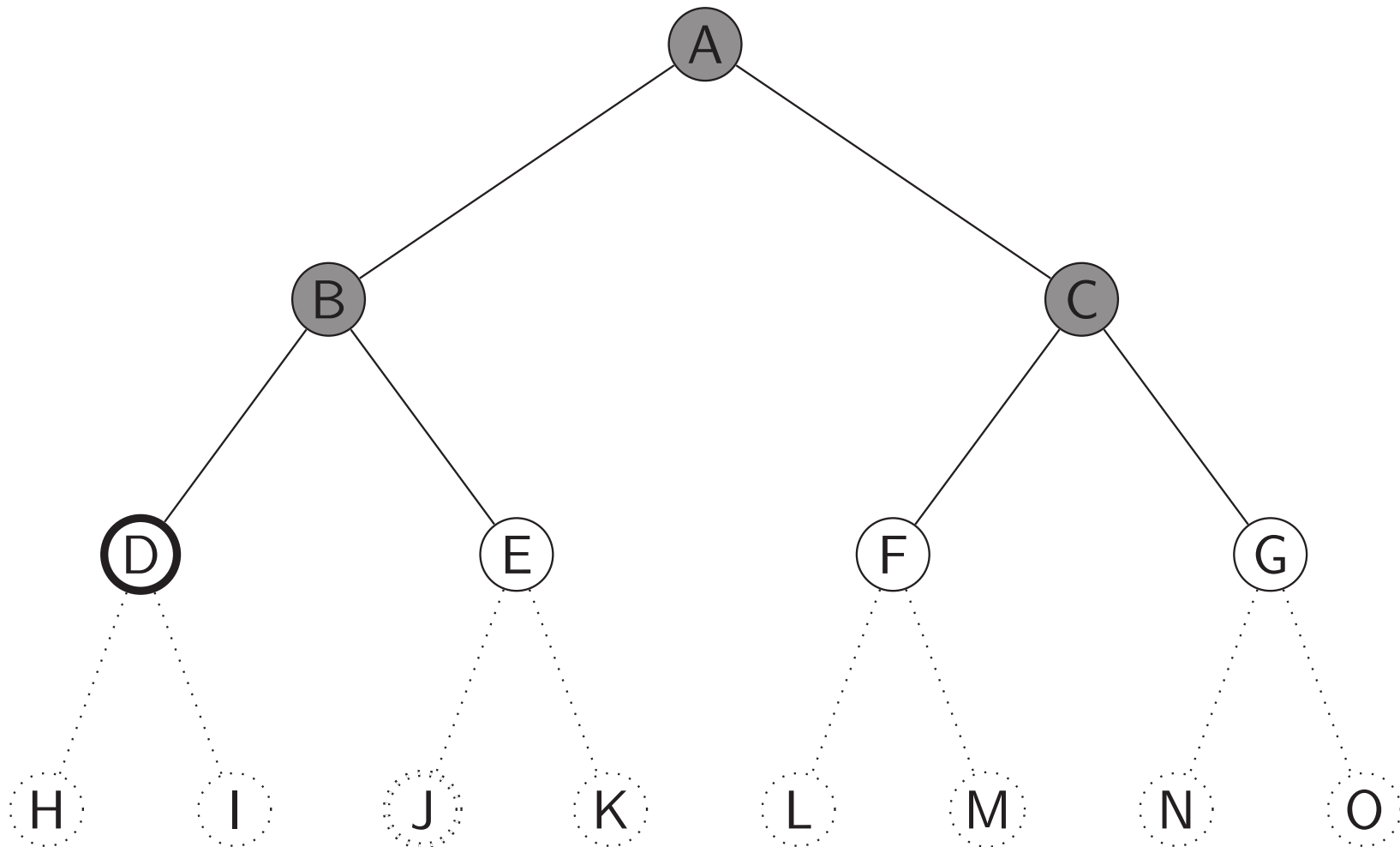
Prohledávání do šířky

Prohledává se vždy nejlevější neexpandovaný uzel s nejmenší hloubkou.
(*Breadth-first Search, BFS*)



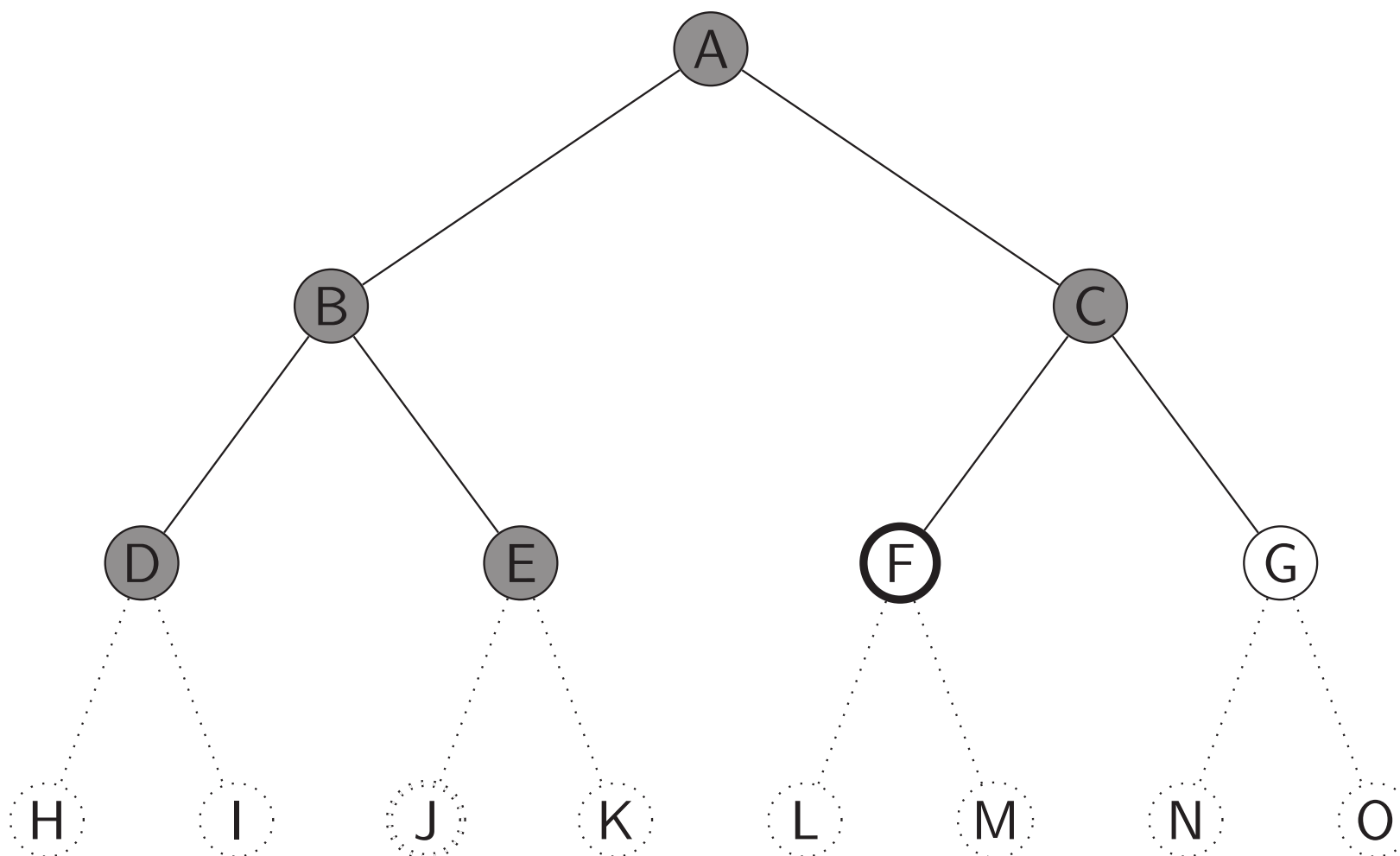
Prohledávání do šířky

Prohledává se vždy nejlevější neexpandovaný uzel s nejmenší hloubkou.
(*Breadth-first Search, BFS*)



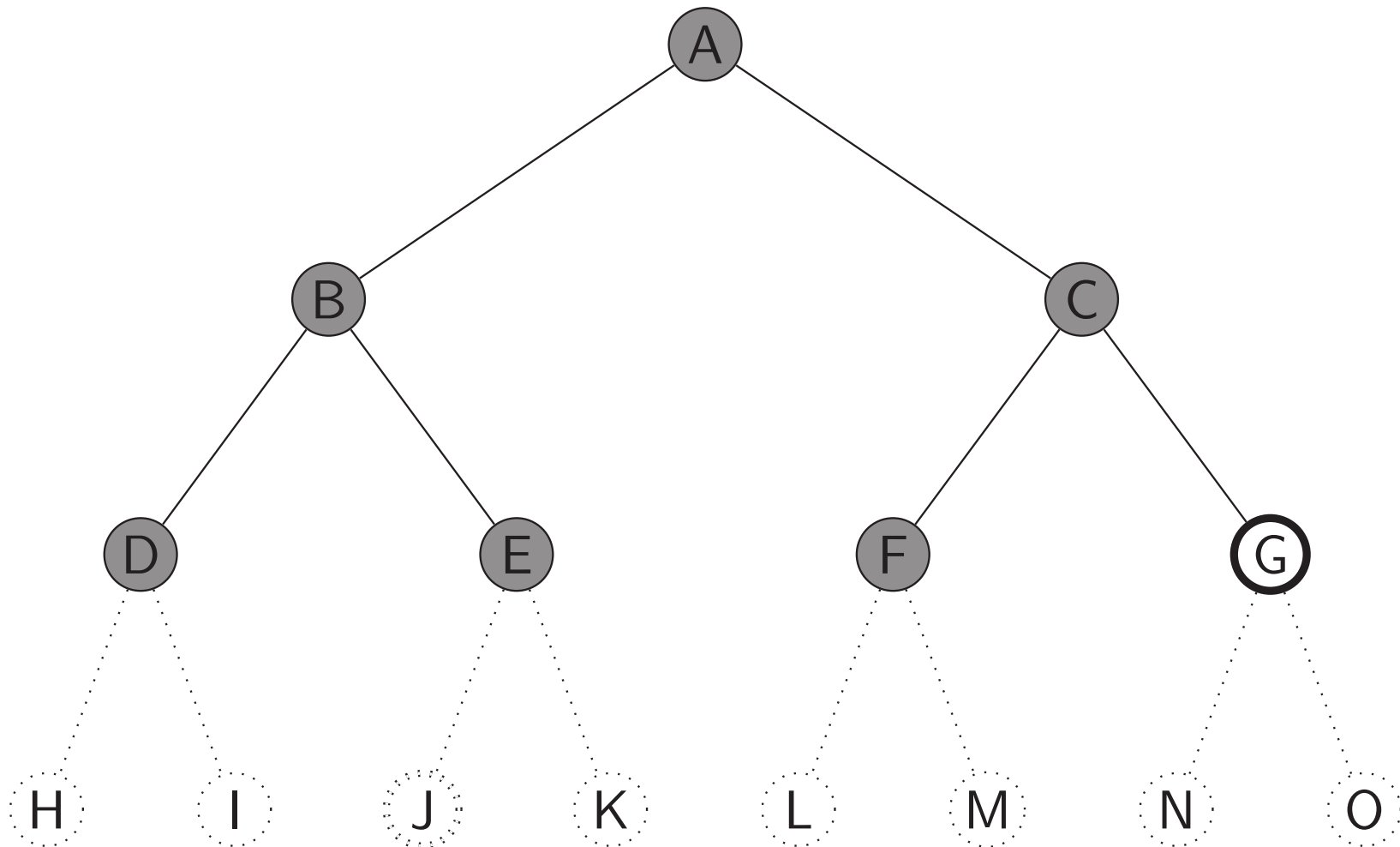
Prohledávání do šířky

Prohledává se vždy nejlevější neexpandovaný uzel s nejmenší hloubkou.
(*Breadth-first Search, BFS*)



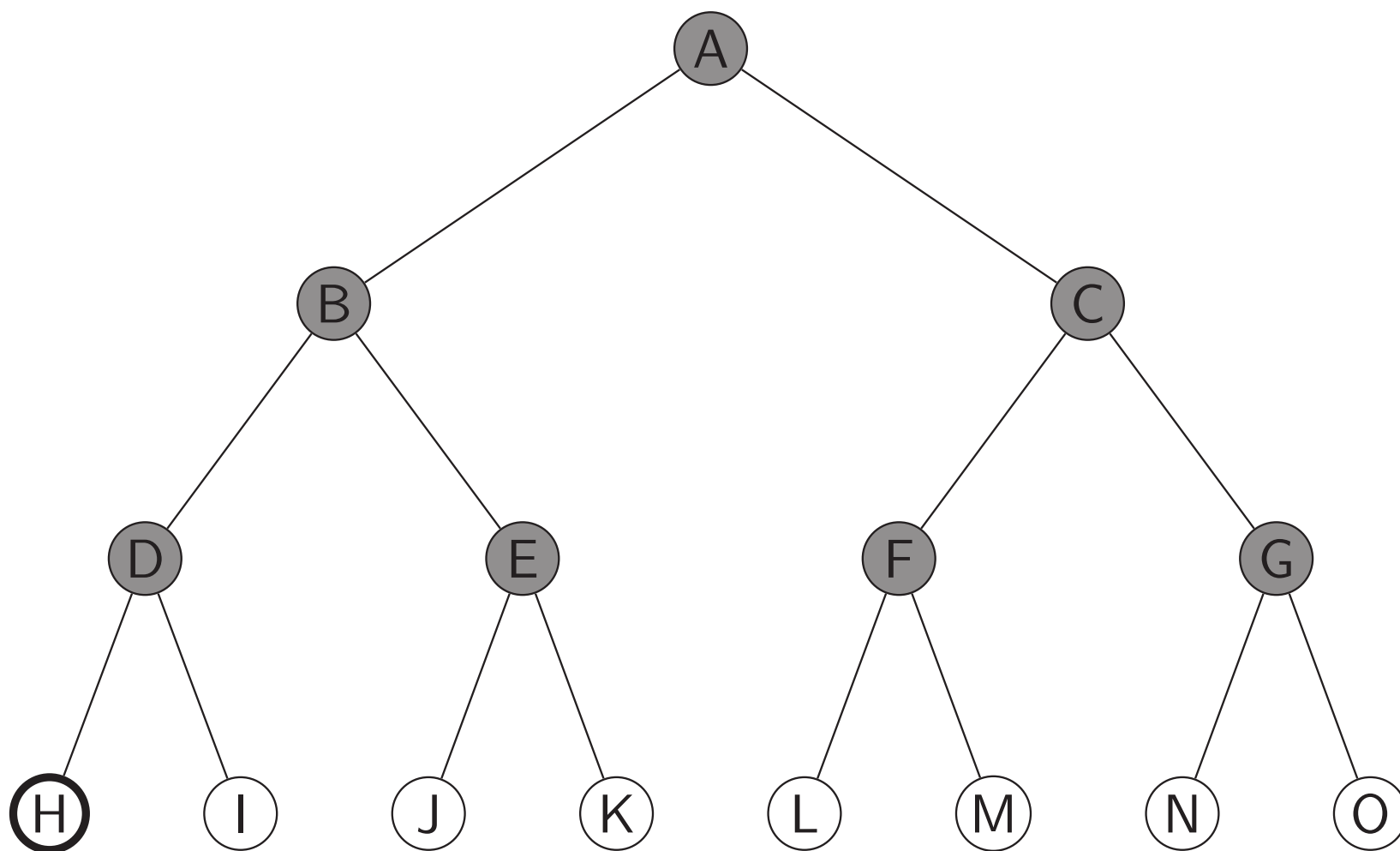
Prohledávání do šířky

Prohledává se vždy nejlevější neexpandovaný uzel s nejmenší hloubkou.
(*Breadth-first Search, BFS*)



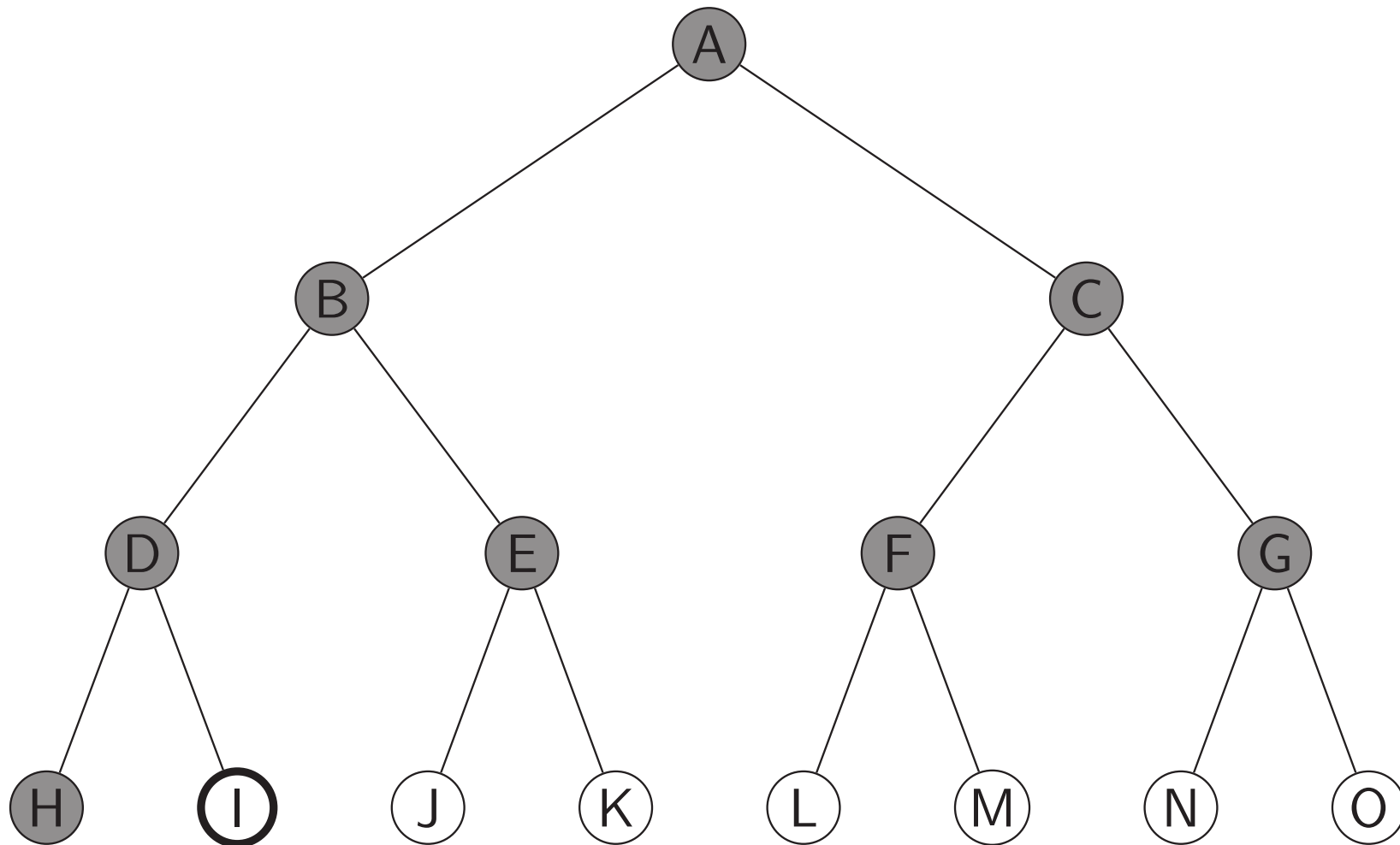
Prohledávání do šířky

Prohledává se vždy nejlevější neexpandovaný uzel s nejmenší hloubkou.
(*Breadth-first Search, BFS*)



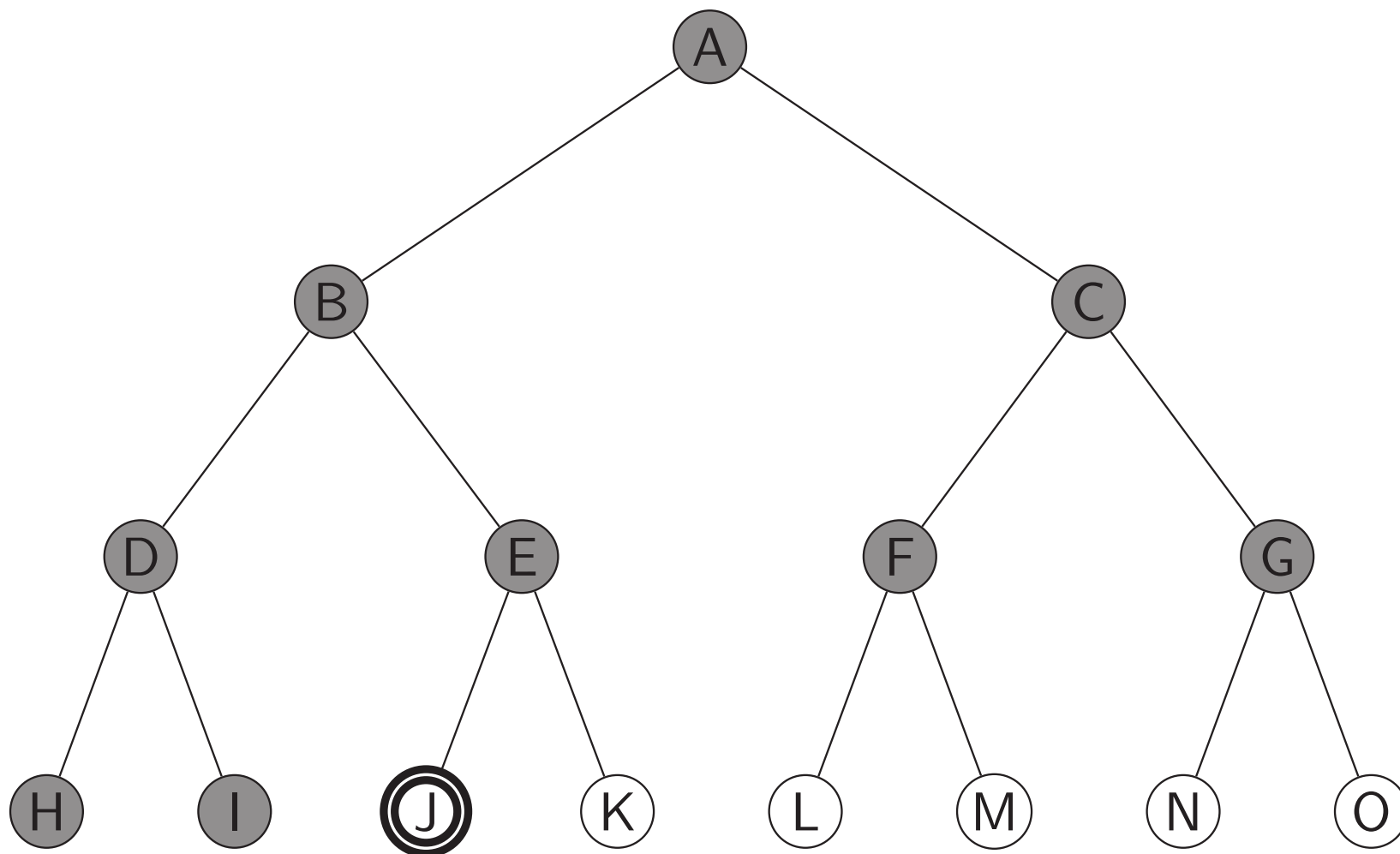
Prohledávání do šířky

Prohledává se vždy nejlevější neexpandovaný uzel s nejmenší hloubkou.
(*Breadth-first Search, BFS*)



Prohledávání do šířky

Prohledává se vždy nejlevější neexpandovaný uzel s nejmenší hloubkou.
(*Breadth-first Search, BFS*)



Prohledávání do šířky

procedurální programovací jazyk – uzly se uloží do **fronty** (FIFO) × Prolog
– udržuje **seznam cest**

```
solution(Start,Solution) :- breadth_first_search([[Start]],Solution).
```

```
breadth_first_search([[Node|Path]|_],[Node|Path]) :- goal(Node).
```

```
breadth_first_search([[N|Path]|Paths],Solution) :-
```

```
    bagof([M,N|Path], (move(N,M), \+ member(M,[N|Path])), NewPaths),
```

```
    NewPaths\=[], append(Paths,NewPaths,Path1), !,
```

```
    breadth_first_search(Path1,Solution); breadth_first_search(Paths,Solution).
```

Vylepšení:

- **append** → **append_dl**

- seznam cest:

[[a]]		l(a)
[[b,a],[c,a]]	→	t(a,[l(b),l(c)])
[[c,a],[d,b,a],[e,b,a]]		t(a,[t(b,[l(d),l(e)]),l(c)])
[[d,b,a],[e,b,a],[f,c,a],[g,c,a]]		t(a,[t(b,[l(d),l(e)]),t(c,[l(f),l(g)])])

Prohledávání do šířky

procedurální programovací jazyk – uzly se uloží do **fronty** (FIFO) × Prolog
– udržuje **seznam cest**

```
solution(Start,Solution) :- breadth_first_search([[Start]],Solution)
```

```
breadth_first_search([[Node|Path]|_],[Node|Path]) :- goal(Node).
```

```
breadth_first_search([[N|Path]|Paths],Solution) :-
```

```
    bagof([M,N|Path], (move(N,M), \+ member(M,[N|Path])), NewPaths),
```

```
    NewPaths\=[], append(Paths,NewPaths,Path1), !,
```

```
    breadth_first_search(Path1,Solution); breadth_first_search(Paths,Solution).
```

bagof(+Prom,+Cíl,-Sezn) postupně
vyhodnocuje **Cíl** a všechny vyhovující
instance **Prom** řadí do seznamu **Sezn**

Vylepšení:

- **append** → **append_dl**

- seznam cest:

```
[[a]]
[[b,a],[c,a]]
[[c,a],[d,b,a],[e,b,a]]
[[d,b,a],[e,b,a],[f,c,a],[g,c,a]]
```

→

```
l(a)
t(a,[l(b),l(c)])
t(a,[t(b,[l(d),l(e)]),l(c)])
t(a,[t(b,[l(d),l(e)]),t(c,[l(f),l(g)])])
```

Prohledávání do šířky

procedurální programovací jazyk – uzly se uloží do **fronty** (FIFO) × Prolog
– udržuje **seznam cest**

```
solution(Start,Solution) :- breadth_first_search([[Start]],Solution)
```

```
breadth_first_search([[Node|Path]|_],[Node|Path]) :- goal(Node).
```

```
breadth_first_search([[N|Path]|Paths],Solution) :-
```

```
    bagof([M,N|Path], (move(N,M), \+ member(M,[N|Path])), NewPaths),
```

```
    NewPaths\=[], append(Paths,NewPaths,Path1), !,
```

```
    breadth_first_search(Path1,Solution); breadth_first_search(Paths,Solution).
```

bagof(+Prom,+Cíl,-Sezn) postupně
vyhodnocuje **Cíl** a všechny vyhovující
instance **Prom** řadí do seznamu **Sezn**

p :- a,b;c. ⇔ p :- (a,b);c.

Vylepšení:

- **append** → **append_dl**

- seznam cest:

```
[[a]]
[[b,a],[c,a]]
[[c,a],[d,b,a],[e,b,a]]
[[d,b,a],[e,b,a],[f,c,a],[g,c,a]]
```

→

```
l(a)
t(a,[l(b),l(c)])
t(a,[t(b,[l(d),l(e)]),l(c)])
t(a,[t(b,[l(d),l(e)]),t(c,[l(f),l(g)])])
```

Prohledávání do šířky

procedurální programovací jazyk – uzly se uloží do **fronty** (FIFO) × Prolog
– udržuje **seznam cest**

```
solution(Start,Solution) :- breadth_first_search([[Start]],Solution)
```

```
breadth_first_search([[Node|Path]|_],[Node|Path]) :- goal(Node).
```

```
breadth_first_search([[N|Path]|Paths],Solution) :-
```

```
    bagof([M,N|Path], (move(N,M), \+ member(M,[N|Path])), NewPaths),
```

```
    NewPaths\=[], append(Paths,NewPaths,Path1), !,
```

```
    breadth_first_search(Path1,Solution); breadth_first_search(Paths,Solution).
```

bagof(+Prom,+Cíl,-Sezn) postupně
vyhodnocuje **Cíl** a všechny vyhovující
instance **Prom** řadí do seznamu **Sezn**

p :- a,b;c. ⇔ p :- (a,b);c.

Vylepšení:

- **append** → **append_dl**

- seznam cest:

```
[[a]]
[[b,a],[c,a]]
[[c,a],[d,b,a],[e,b,a]]
[[d,b,a],[e,b,a],[f,c,a],[g,c,a]]
```

→

```
l(a)
t(a,[l(b),l(c)])
t(a,[t(b,[l(d),l(e)]),l(c)])
t(a,[t(b,[l(d),l(e)]),t(c,[l(f),l(g)])])
```

Prohledávání do šířky – vlastnosti

<i>úplnost</i>	je úplný (pro konečné b)
<i>optimálnost</i>	je optimální podle délky cesty/ <i>není</i> optimální podle obecné ceny
<i>časová složitost</i>	$1 + b + b^2 + b^3 + \dots + b^d + b(b^d - 1) = O(b^{d+1})$, exponenciální v d
<i>prostorová složitost</i>	$O(b^{d+1})$ (každý uzel v paměti)

Největší problém – paměť:

Hloubka	Uzlů	Čas	Paměť
2	1100	0.11 sek	1 MB
4	111 100	11 sek	106 MB
6	10^7	19 min	10 GB
8	10^9	31 hod	1 TB
10	10^{11}	129 dnů	101 TB
12	10^{13}	35 let	10 PB
14	10^{15}	3 523 let	1 EB

Ani čas není dobrý → potřebujeme *informované* strategie prohledávání.

Prohledávání do šířky – vlastnosti

úplnost

je úplný (pro konečné b)

optimálnost

je optimální podle délky cesty/není optimální podle obecné ceny

časová složitost

$1 + b + b^2 + b^3 + \dots + b^d + b(b^d - 1) = O(b^{d+1})$,
exponenciální v d

prostorová složitost

$O(b^{d+1})$ (každý uzel v paměti)

Největší problém – paměť:

Hloubka	Uzlů	Čas	Paměť
2	1100	0.11 sek	1 MB
4	111 100	11 sek	106 MB
6	10^7	19 min	10 GB
8	10^9	31 hod	1 TB
10	10^{11}	129 dnů	101 TB
12	10^{13}	35 let	10 PB
14	10^{15}	3 523 let	1 EB

Ani čas není dobrý → potřebujeme informované strategie prohledávání.

Prohledávání do šířky – vlastnosti

<i>úplnost</i>	je úplný (pro konečné b)
<i>optimálnost</i>	je optimální podle délky cesty/ není optimální podle obecné ceny
<i>časová složitost</i>	$1 + b + b^2 + b^3 + \dots + b^d + b(b^d - 1) = O(b^{d+1})$, exponenciální v d
<i>prostorová složitost</i>	$O(b^{d+1})$ (každý uzel v paměti)

Největší problém – paměť:

Hloubka	Uzlů	Čas	Paměť
2	1100	0.11 sek	1 MB
4	111 100	11 sek	106 MB
6	10^7	19 min	10 GB
8	10^9	31 hod	1 TB
10	10^{11}	129 dnů	101 TB
12	10^{13}	35 let	10 PB
14	10^{15}	3 523 let	1 EB

Ani čas není dobrý → potřebujeme **informované** strategie prohledávání.

Prohledávání do šířky – vlastnosti

<i>úplnost</i>	je úplný (pro konečné b)
<i>optimálnost</i>	je optimální podle délky cesty/ <i>není</i> optimální podle obecné ceny
<i>časová složitost</i>	$1 + b + b^2 + b^3 + \dots + b^d + b(b^d - 1) = O(b^{d+1})$, exponenciální v d
<i>prostorová složitost</i>	$O(b^{d+1})$ (každý uzel v paměti)

Největší problém – paměť:

Hloubka	Uzlů	Čas	Paměť
2	1100	0.11 sek	1 MB
4	111 100	11 sek	106 MB
6	10^7	19 min	10 GB
8	10^9	31 hod	1 TB
10	10^{11}	129 dnů	101 TB
12	10^{13}	35 let	10 PB
14	10^{15}	3 523 let	1 EB

Ani čas není dobrý → potřebujeme *informované* strategie prohledávání.

Prohledávání do šířky – vlastnosti

<i>úplnost</i>	je úplný (pro konečné b)
<i>optimálnost</i>	je optimální podle délky cesty/ <i>není</i> optimální podle obecné ceny
<i>časová složitost</i>	$1 + b + b^2 + b^3 + \dots + b^d + b(b^d - 1) = O(b^{d+1})$, exponenciální v d
<i>prostorová složitost</i>	$O(b^{d+1})$ (každý uzel v paměti)

Největší problém – paměť:

Hloubka	Uzlů	Čas	Paměť
2	1100	0.11 sek	1 MB
4	111 100	11 sek	106 MB
6	10^7	19 min	10 GB
8	10^9	31 hod	1 TB
10	10^{11}	129 dnů	101 TB
12	10^{13}	35 let	10 PB
14	10^{15}	3 523 let	1 EB

Ani čas není dobrý → potřebujeme *informované* strategie prohledávání.

Prohledávání do šířky – vlastnosti

<i>úplnost</i>	je úplný (pro konečné b)
<i>optimálnost</i>	je optimální podle délky cesty/ <i>není</i> optimální podle obecné ceny
<i>časová složitost</i>	$1 + b + b^2 + b^3 + \dots + b^d + b(b^d - 1) = O(b^{d+1})$, exponenciální v d
<i>prostorová složitost</i>	$O(b^{d+1})$ (každý uzel v paměti)

Největší problém – paměť:

Hloubka	Uzlů	Čas	Paměť
2	1100	0.11 sek	1 MB
4	111 100	11 sek	106 MB
6	10^7	19 min	10 GB
8	10^9	31 hod	1 TB
10	10^{11}	129 dnů	101 TB
12	10^{13}	35 let	10 PB
14	10^{15}	3 523 let	1 EB

Ani čas není dobrý → potřebujeme **informované** strategie prohledávání.

Prohledávání podle ceny

- BFS je optimální pro rovnoměrně ohodnocené stromy × **prohledávání podle ceny** (**Uniform-cost Search**) je optimální pro **obecné ohodnocení**
- fronta uzlů se udržuje **uspořádaná** podle ceny cesty

Vlastnosti:

<i>úplnost</i>	je úplný (pro $\epsilon \geq 0$)
<i>optimálnost</i>	je optimální (pro $\epsilon \geq 0$, $g(n)$ roste)
<i>časová složitost</i>	počet uzlů s $g \leq C^*$, $O(b^{1+\lceil C^*/\epsilon \rceil})$, kde C^* ... cena optimálního řešení
<i>prostorová složitost</i>	počet uzlů s $g \leq C^*$, $O(b^{1+\lceil C^*/\epsilon \rceil})$

Prohledávání podle ceny

- BFS je optimální pro rovnoměrně ohodnocené stromy × **prohledávání podle ceny** (**Uniform-cost Search**) je optimální pro **obecné ohodnocení**
- fronta uzlů se udržuje **uspořádaná** podle ceny cesty

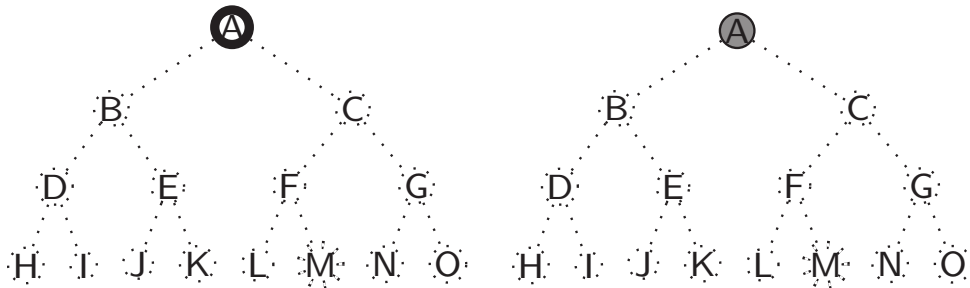
Vlastnosti:

<i>úplnost</i>	je úplný (pro $\epsilon \geq 0$)
<i>optimálnost</i>	je optimální (pro $\epsilon \geq 0$, $g(n)$ roste)
<i>časová složitost</i>	počet uzlů s $g \leq C^*$, $O(b^{1+\lfloor C^*/\epsilon \rfloor})$, kde C^* ... cena optimálního řešení
<i>prostorová složitost</i>	počet uzlů s $g \leq C^*$, $O(b^{1+\lfloor C^*/\epsilon \rfloor})$

Prohledávání s postupným prohlubováním

prohledávání do hloubky s postupně se zvyšujícím limitem (Iterative deepening DFS, IDS)

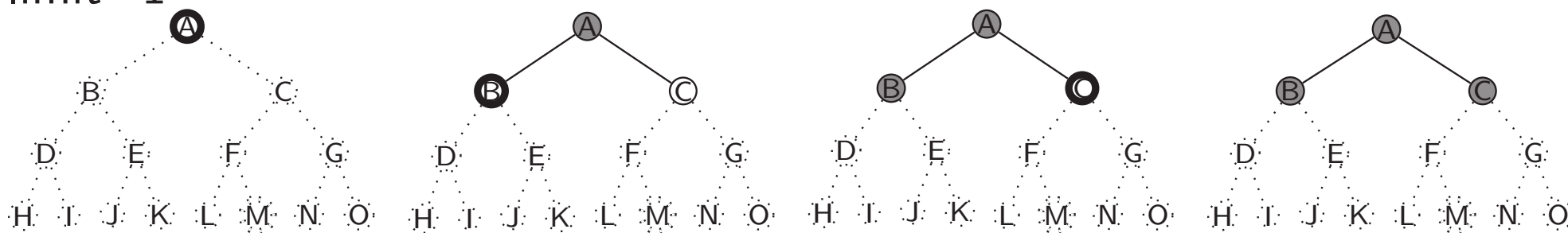
limit=0



Prohledávání s postupným prohlubováním

prohledávání do hloubky s postupně se zvyšujícím limitem (Iterative deepening DFS, IDS)

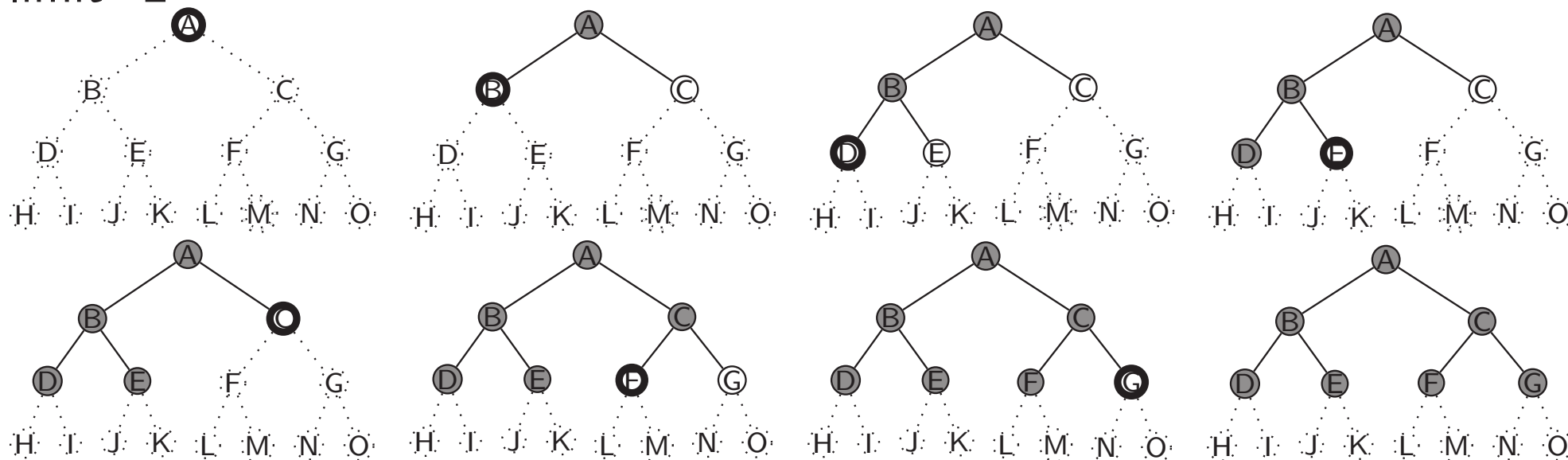
limit=1



Prohledávání s postupným prohlubováním

prohledávání do hloubky s postupně se **zvyšujícím limitem** (Iterative deepening DFS, IDS)

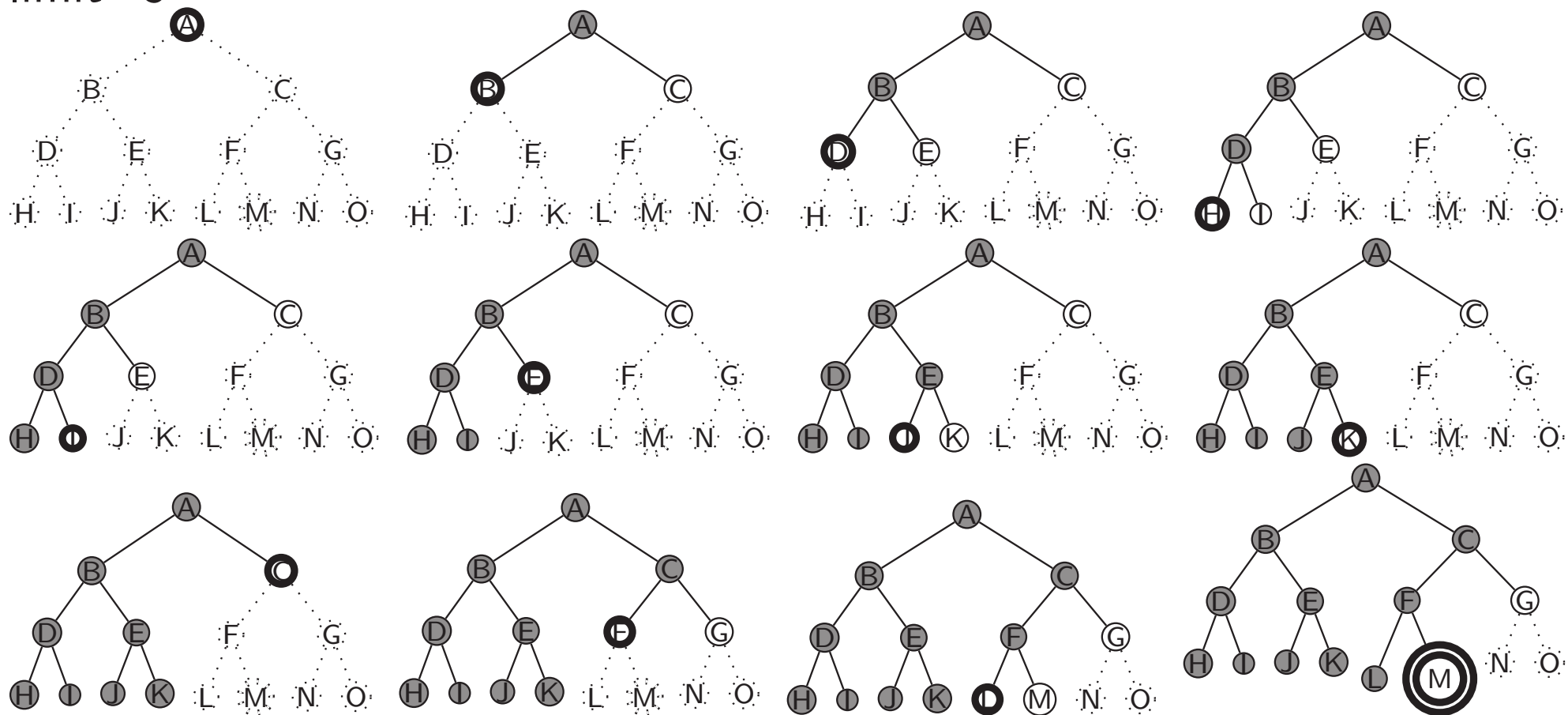
limit=2



Prohledávání s postupným prohlubováním

prohledávání do hloubky s postupně se **zvyšujícím limitem** (Iterative deepening DFS, IDS)

limit=3



Prohledávání s postupným prohlubováním – vlastnosti

<i>úplnost</i>	je úplný (pro konečné b)
<i>optimálnost</i>	je optimální (pro $g(n)$ rovnoměrně neklesající funkce hloubky)
<i>časová složitost</i>	$d(b) + (d - 1)b^2 + \dots + 1(b^d) = O(b^d)$
<i>prostorová složitost</i>	$O(bd)$

- kombinuje výhody BFS a DFS:

- nízké paměťové nároky – lineární
- optimálnost, úplnost

- zdánlivé plýtvání opakovaným generováním

ALE generuje o jednu úroveň míň, např. pro $b = 10, d = 5$:

$$N(\text{IDS}) = 50 + 400 + 3\,000 + 20\,000 + 100\,000 = 123\,450$$

$$N(\text{BFS}) = 10 + 100 + 1\,000 + 10\,000 + 100\,000 + 999\,990 = 1\,111\,100$$

IDS je nejvhodnější neinformovaná strategie pro velké prostory a neznámou hloubku řešení.

Prohledávání s postupným prohlubováním – vlastnosti

<i>úplnost</i>	je úplný (pro konečné b)
<i>optimálnost</i>	je optimální (pro $g(n)$ rovnoměrně neklesající funkce hloubky)
<i>časová složitost</i>	$d(b) + (d - 1)b^2 + \dots + 1(b^d) = O(b^d)$
<i>prostorová složitost</i>	$O(bd)$

- kombinuje výhody BFS a DFS:
 - nízké paměťové nároky – lineární
 - optimálnost, úplnost
- zdánlivé plýtvání opakovaným generováním

ALE generuje o jednu úroveň míň, např. pro $b = 10, d = 5$:

$$N(\text{IDS}) = 50 + 400 + 3\,000 + 20\,000 + 100\,000 = 123\,450$$

$$N(\text{BFS}) = 10 + 100 + 1\,000 + 10\,000 + 100\,000 + 999\,990 = 1\,111\,100$$

IDS je nejvhodnější neinformovaná strategie pro velké prostory a neznámou hloubku řešení.

Prohledávání s postupným prohlubováním – vlastnosti

<i>úplnost</i>	je úplný (pro konečné b)
<i>optimálnost</i>	je optimální (pro $g(n)$ rovnoměrně neklesající funkce hloubky)
<i>časová složitost</i>	$d(b) + (d - 1)b^2 + \dots + 1(b^d) = O(b^d)$
<i>prostorová složitost</i>	$O(bd)$

- kombinuje výhody BFS a DFS:
 - nízké paměťové nároky – lineární
 - optimálnost, úplnost
- zdánlivé plýtvání opakovaným generováním

ALE generuje o jednu úroveň míň, např. pro $b = 10, d = 5$:

$$N(\text{IDS}) = 50 + 400 + 3\,000 + 20\,000 + 100\,000 = 123\,450$$

$$N(\text{BFS}) = 10 + 100 + 1\,000 + 10\,000 + 100\,000 + 999\,990 = 1\,111\,100$$

IDS je nejvhodnější neinformovaná strategie pro velké prostory a neznámou hloubku řešení.

Prohledávání s postupným prohlubováním – vlastnosti

<i>úplnost</i>	je úplný (pro konečné b)
<i>optimálnost</i>	je optimální (pro $g(n)$ rovnoměrně neklesající funkce hloubky)
<i>časová složitost</i>	$d(b) + (d - 1)b^2 + \dots + 1(b^d) = O(b^d)$
<i>prostorová složitost</i>	$O(bd)$

- kombinuje výhody BFS a DFS:
 - nízké paměťové nároky – lineární
 - optimálnost, úplnost
- zdánlivé plýtvání opakovaným generováním

ALE generuje o jednu úroveň míň, např. pro $b = 10, d = 5$:

$$N(\text{IDS}) = 50 + 400 + 3\,000 + 20\,000 + 100\,000 = 123\,450$$

$$N(\text{BFS}) = 10 + 100 + 1\,000 + 10\,000 + 100\,000 + 999\,990 = 1\,111\,100$$

IDS je **nejvhodnější** neinformovaná strategie pro **velké prostory** a **neznámou hloubku** řešení.

Shrnutí vlastností algoritmů neinformovaného prohledávání

<i>Vlastnost</i>	<i>do hloubky</i>	<i>do hloubky s limitem</i>	<i>do šířky</i>	<i>podle ceny</i>	<i>s postupným prohlubováním</i>
<i>úplnost</i>	ne	ano, pro $l \geq d$	ano*	ano*	ano*
<i>optimálnost</i>	ne	ne	ano*	ano*	ano*
<i>časová složitost</i>	$O(b^m)$	$O(b^\ell)$	$O(b^{d+1})$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(b^d)$
<i>prostorová složitost</i>	$O(bm)$	$O(b\ell)$	$O(b^{d+1})$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(bd)$