Informed Search
(Heuristic Search)

Boris Velichkov

Exercises by Week

1. Introduction
2. **Uninformed *(Blind)* Search**
3. **Informed *(Heuristic)* Search**
4. **Constraint Satisfaction Problems**
5. **Genetic Algorithms**
6. **Games**
7. Introduction to Machine Learning
8. ***k*-Nearest Neighbors**
9. **Naïve Bayes Classifier**
10. **Decision Tree**
11. ***k*Means**
12. **Neural Networks**
13. *Additional Topics, Questions and Homeworks' Presentations*
14. *Additional Topics, Questions and Homeworks' Presentations*
15. *Additional Topics, Questions and Homeworks' Presentations*

Uninformed *(Blind)* Search vs Informed *(Heuristic)* Search

- **Uninformed Search**: Uninformed strategies use only the information available in the problem definition.
  - *Examples: DFS, BFS, UCS, DLS, IDS.*
- **Informed Search**: Informed strategies have information on the *goal state* which helps in more efficient searching. This information is obtained by a function *(heuristic)* that estimates how close a state is to the *goal state*.
  - *Examples: Greedy Best-First Search, A*, Beam Search, Hill Climbing.*

Informed *(Heuristic)* Search

- *“Informed strategies have information on the goal state which helps in more efficient searching. This information is obtained by a **function (heuristic)** that estimates how close a state is to the goal state.”*
- Informed Search Algorithms:
  - Best-First Search
    - Greedy
      - Beam Search
      - Hill Climbing
    - A*
      - Memory-Bounded A*
      - Iterative Deepening A* (IDA*)

Best-First Search

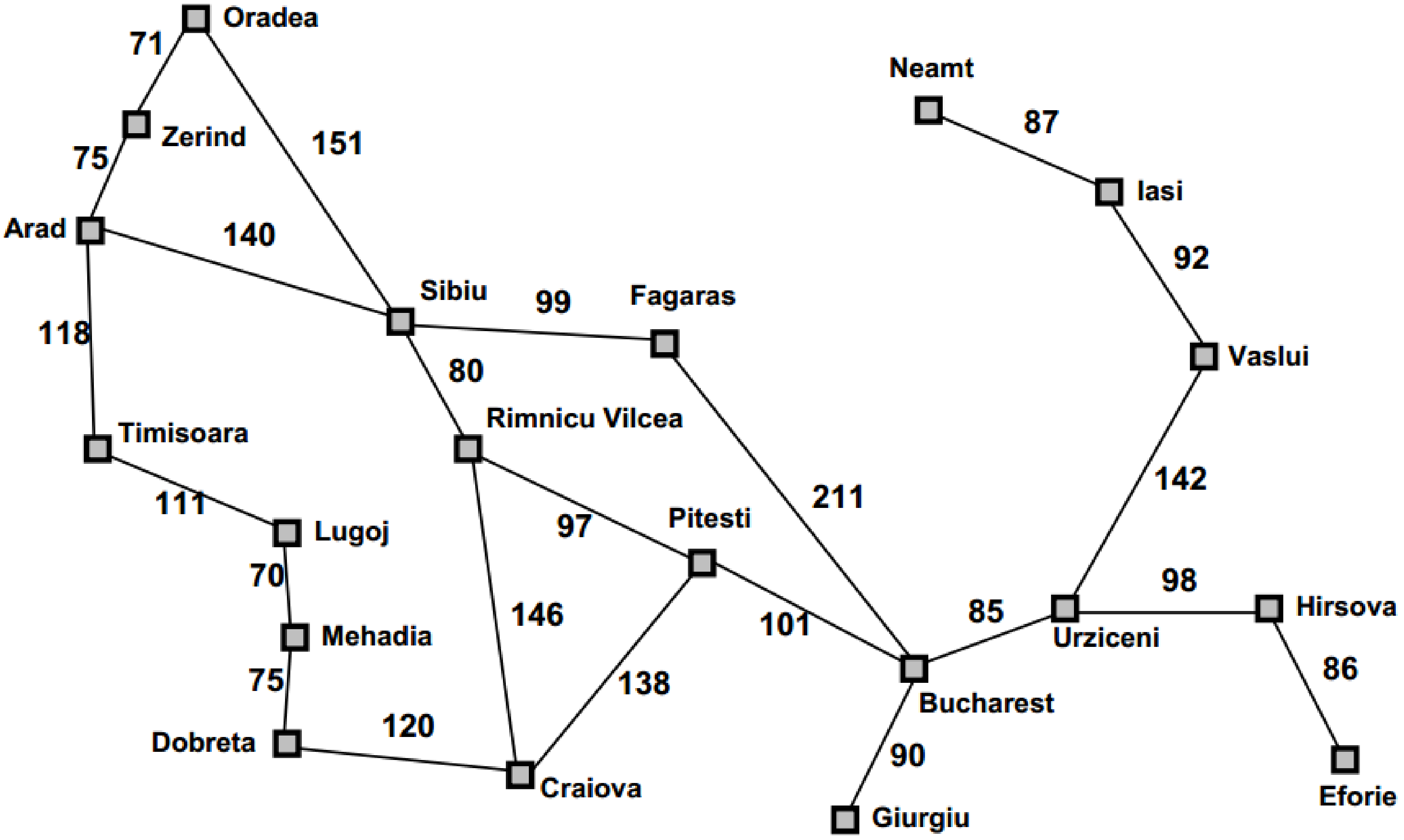
- ***Idea:*** use an evaluation function for each node
    - estimate of "desirability"

  ⇒ Expand most desirable unexpanded node

- ***Implementation:***
    - ***fringe*** is a queue sorted in decreasing order of desirability
    - Special cases:
        - Greedy search
        - A* search

Best-First Search

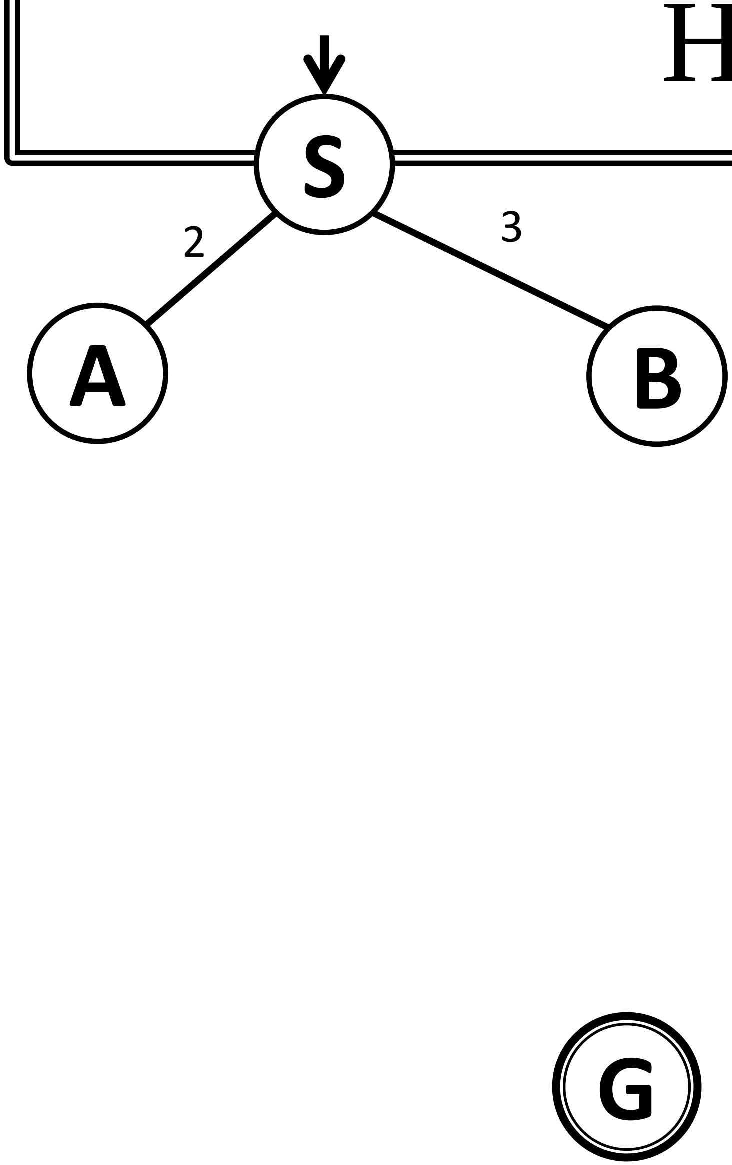
Romania with step costs in km



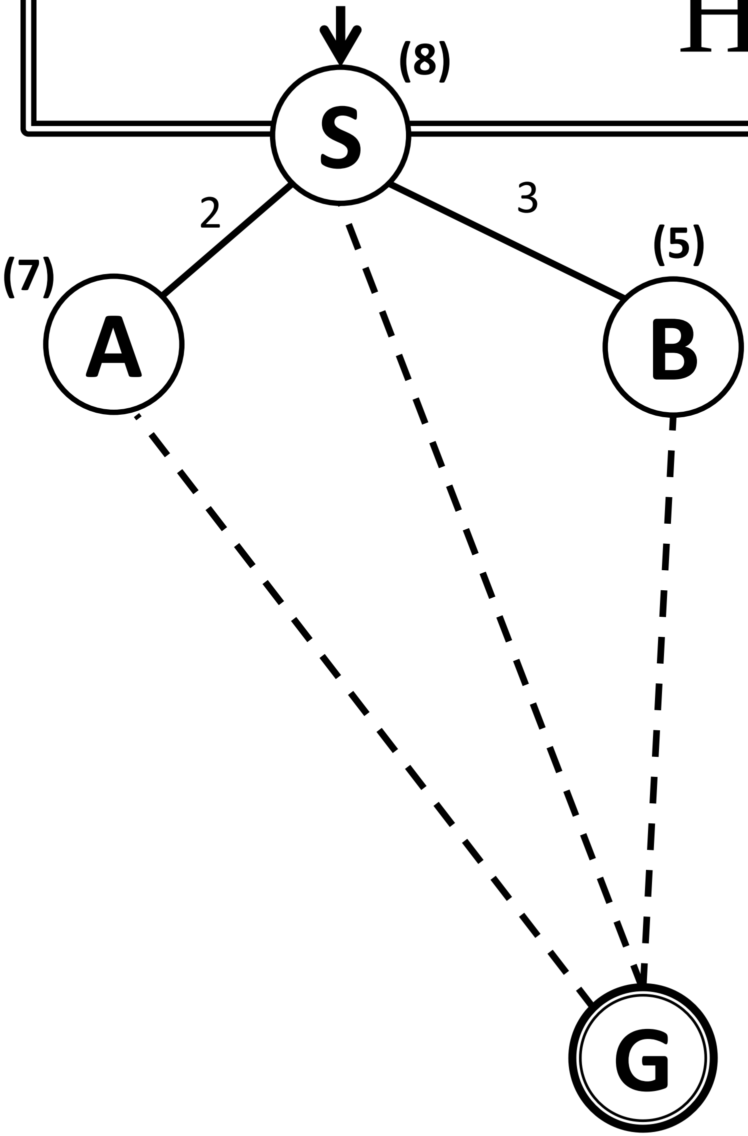
Straight–line distance
to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

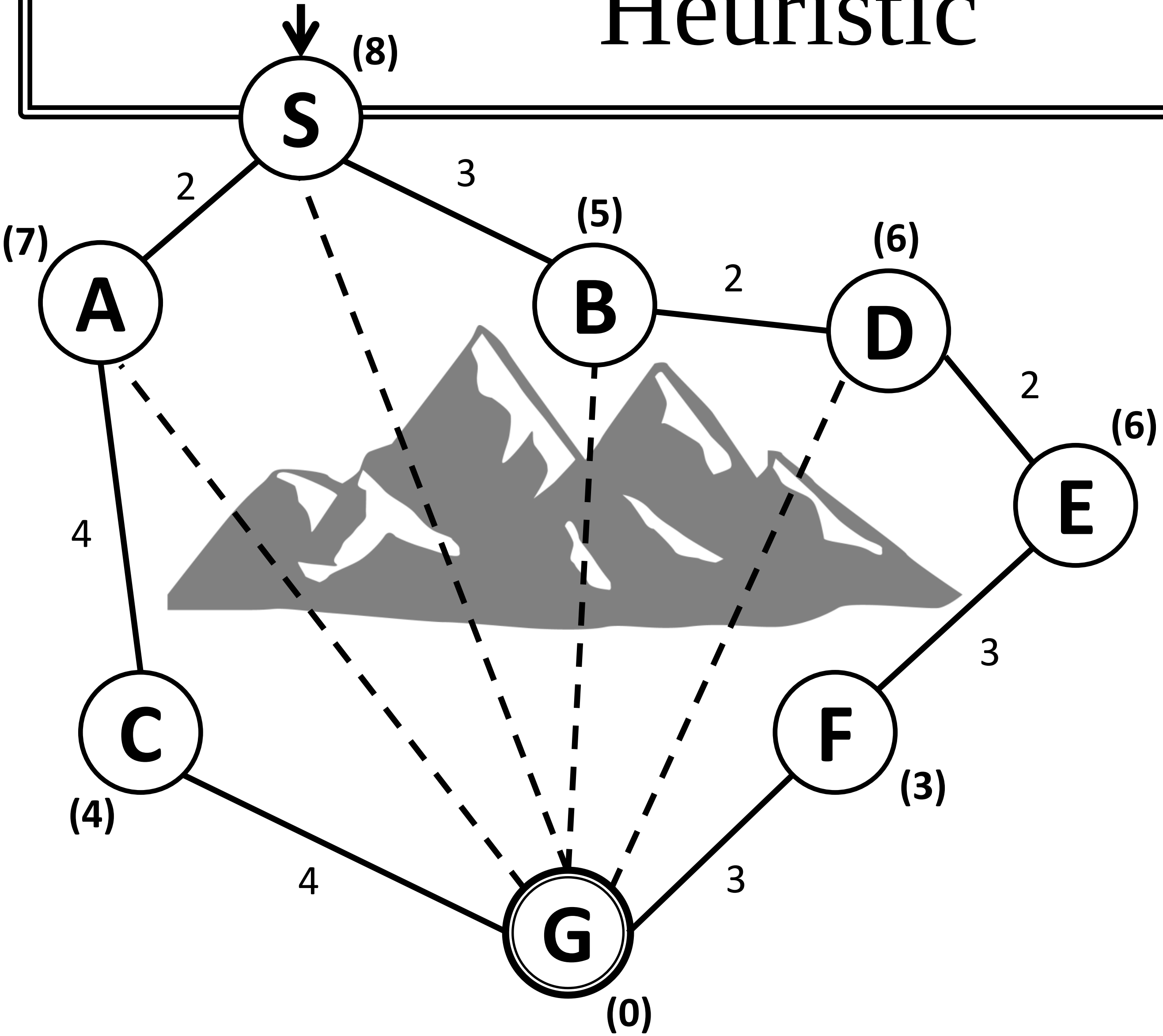
Heuristic



Heuristic



Heuristic



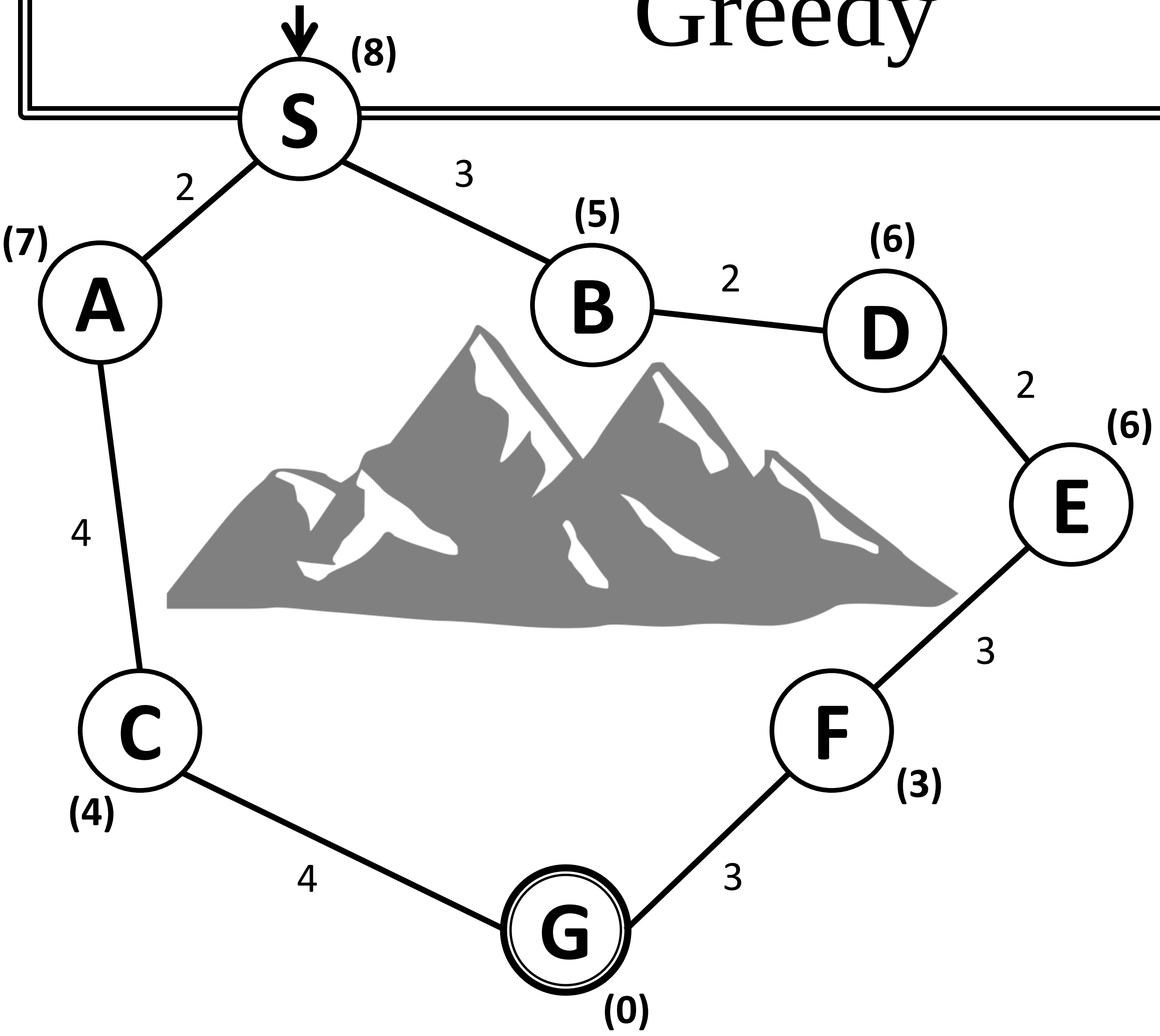
Greedy

- Evaluation function ***h(n)*** (***h****euristic*)

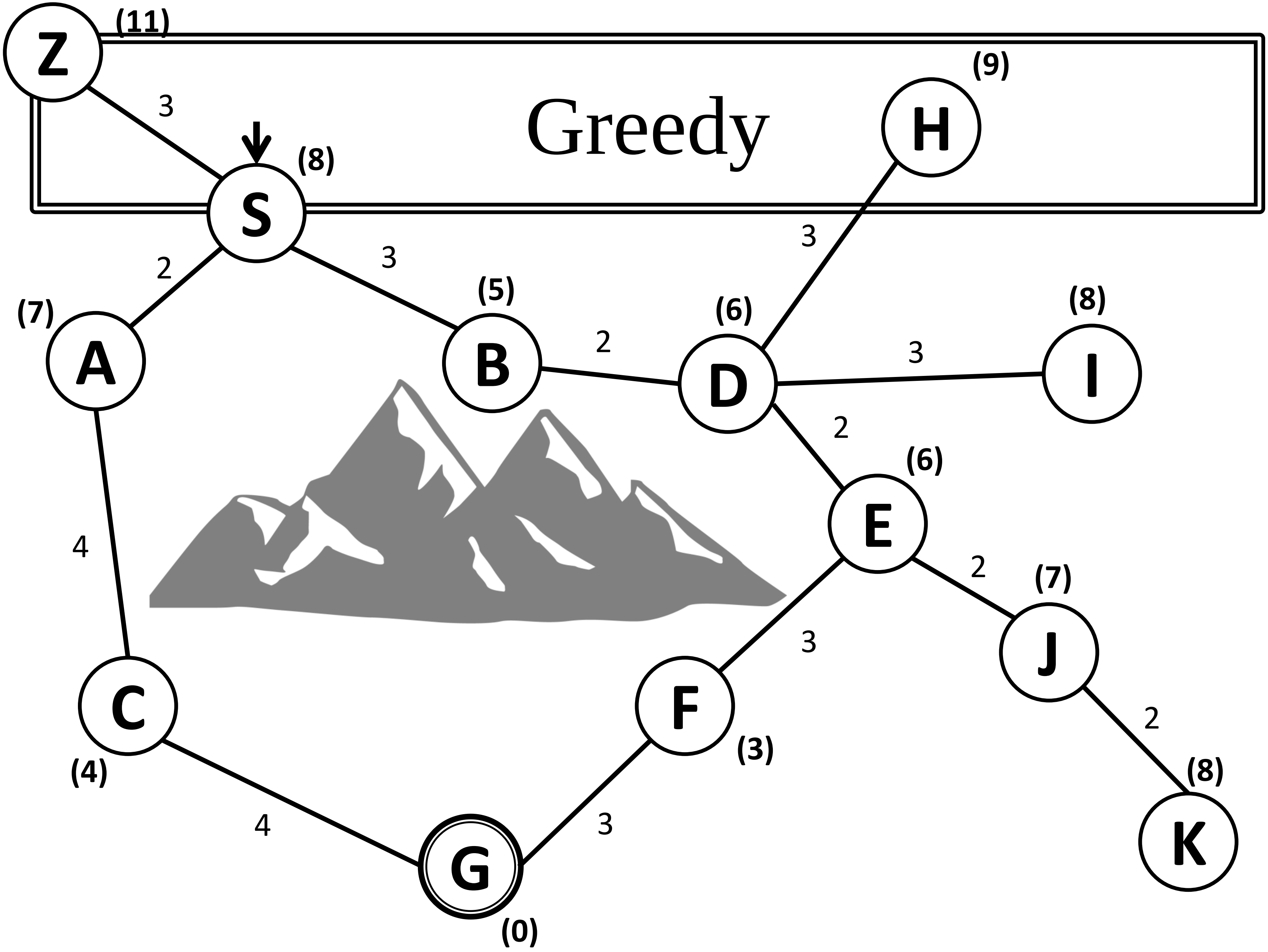
  = estimate of cost from ***n*** to the closest *goal*

- E.g., ***h_{SLD}(n)*** = *straight-line distance* from ***n*** to *goal* (*Bucharest*)
- ***Greedy*** search expands the node that ***appears*** to be closest to *goal*

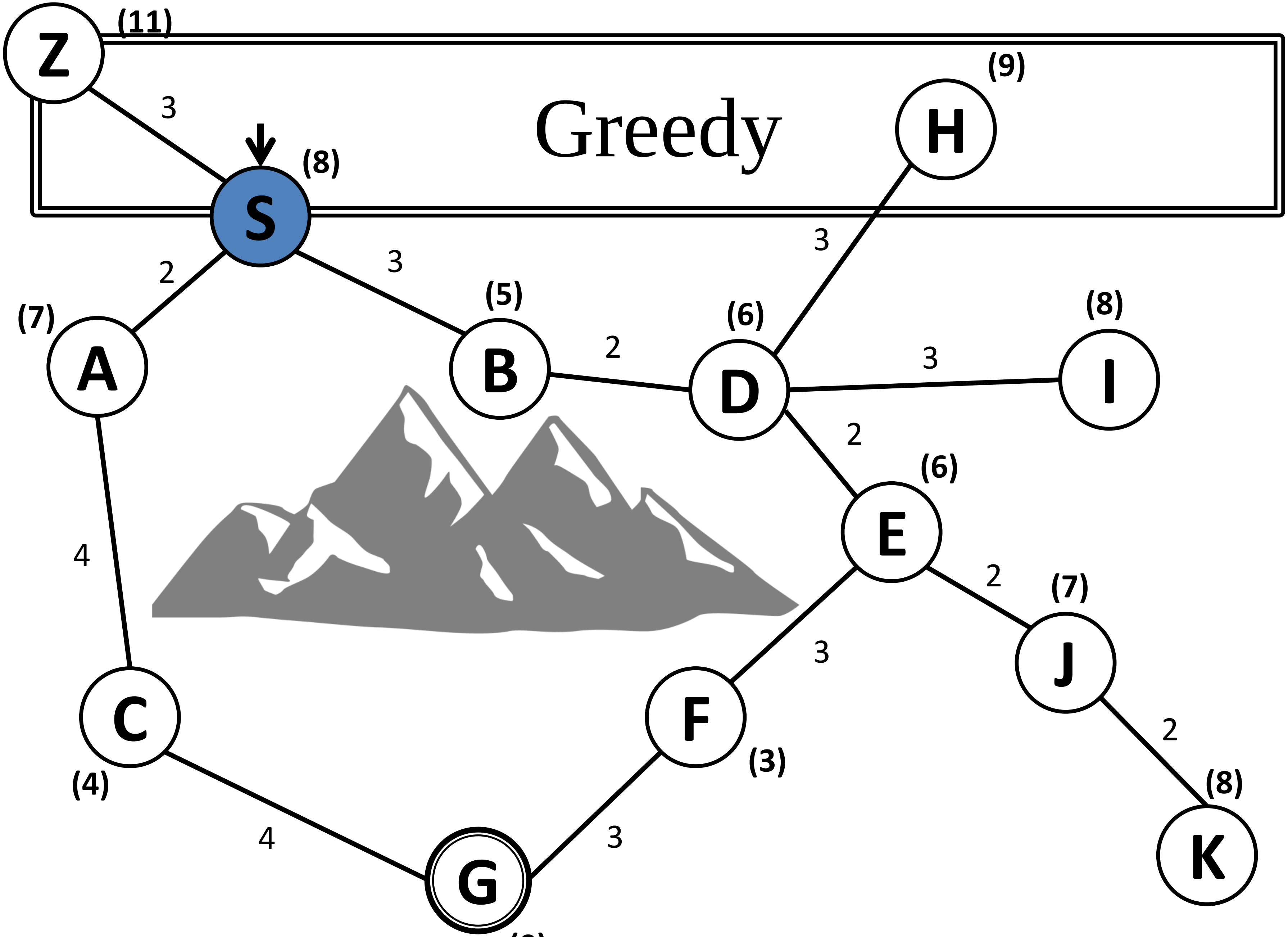
Greedy



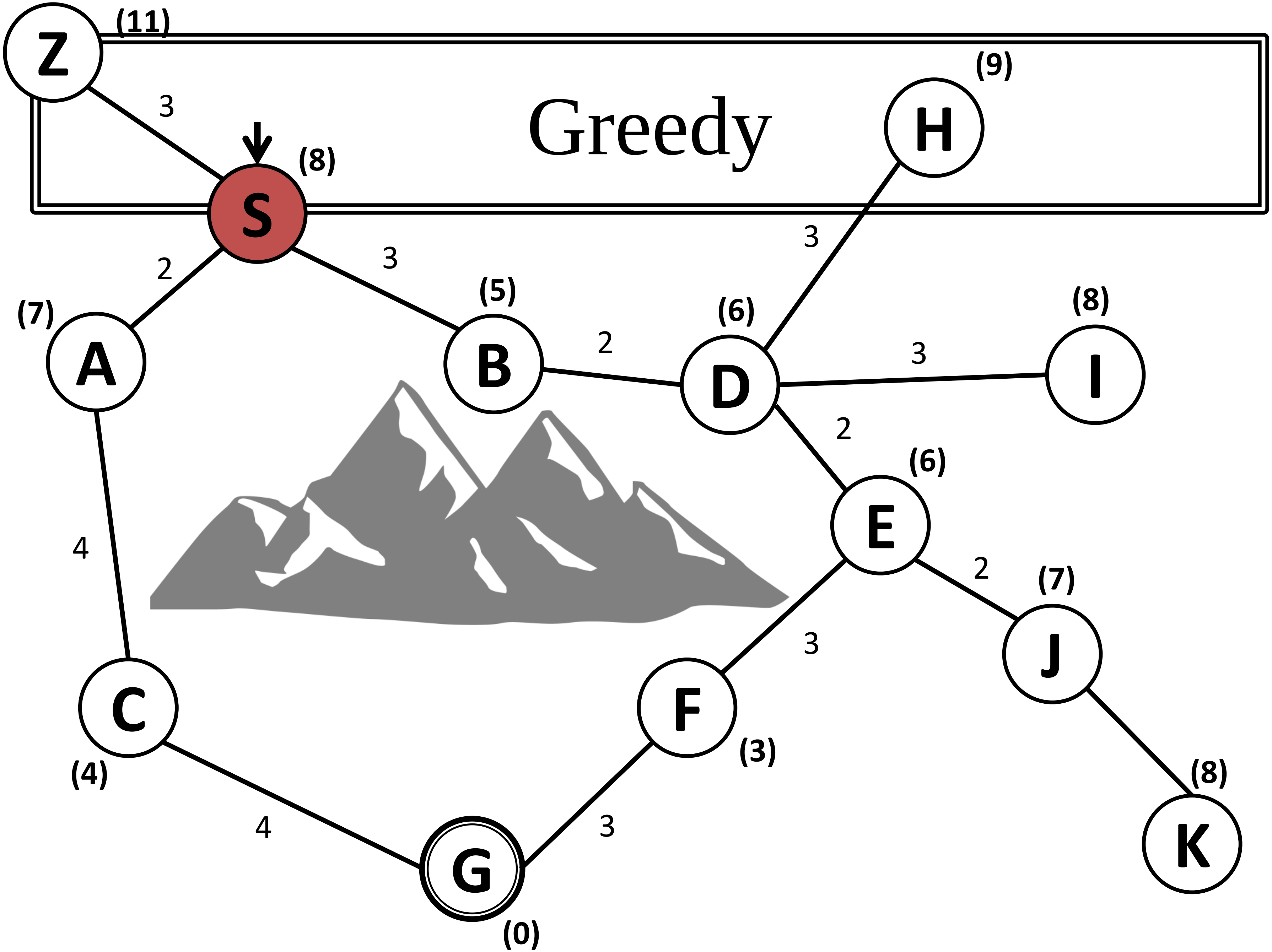
Greedy



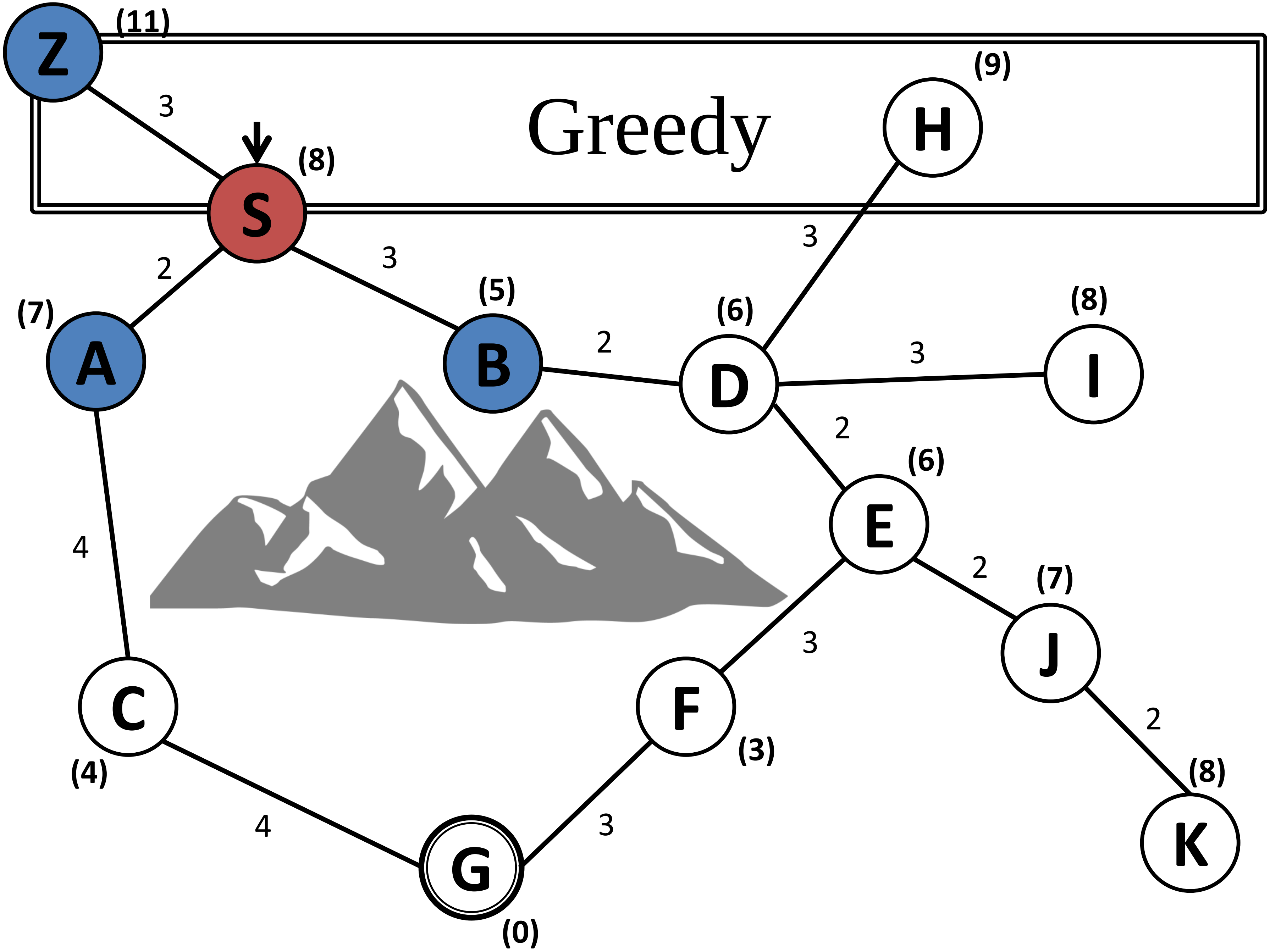
Greedy



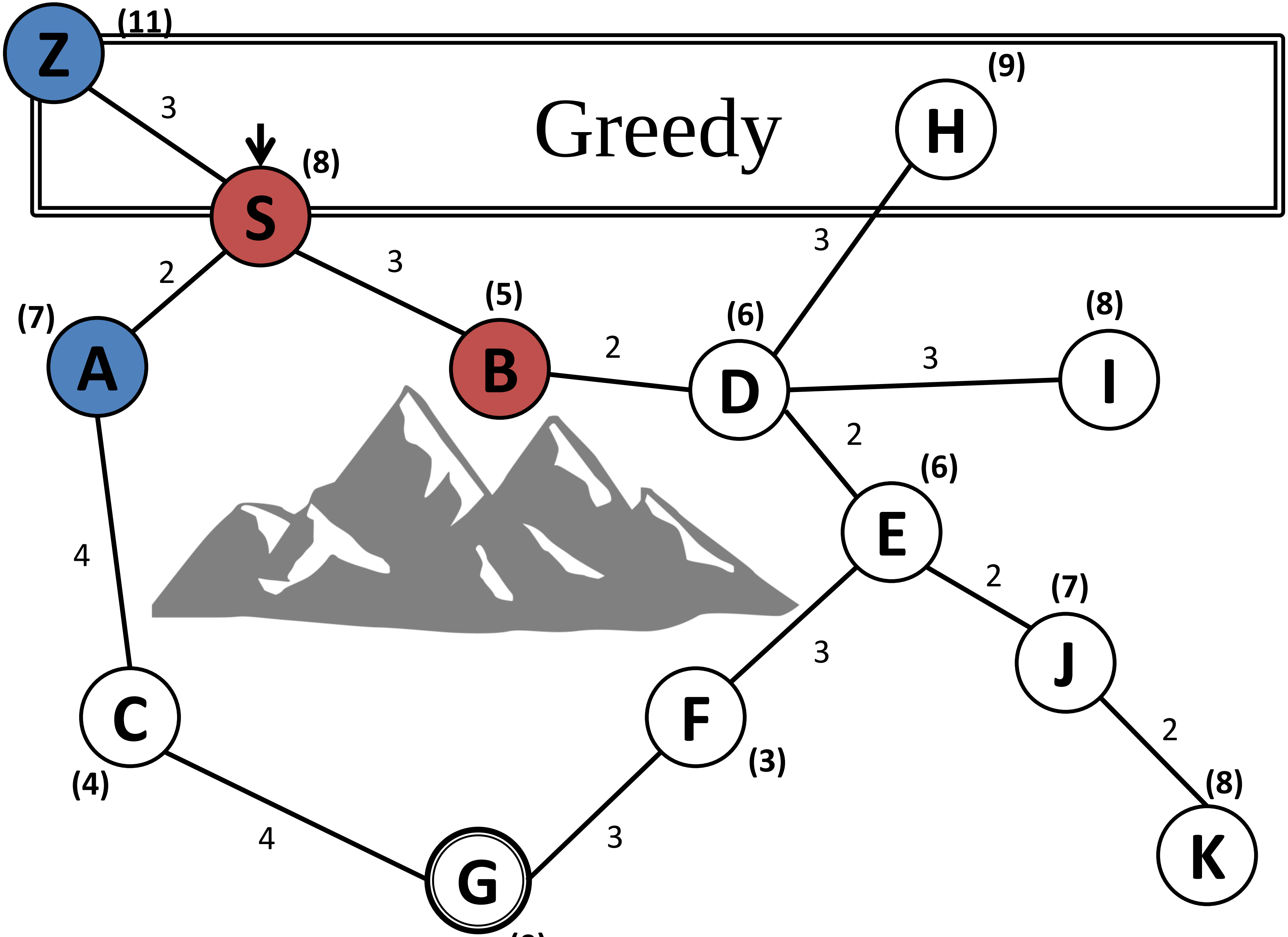
Greedy



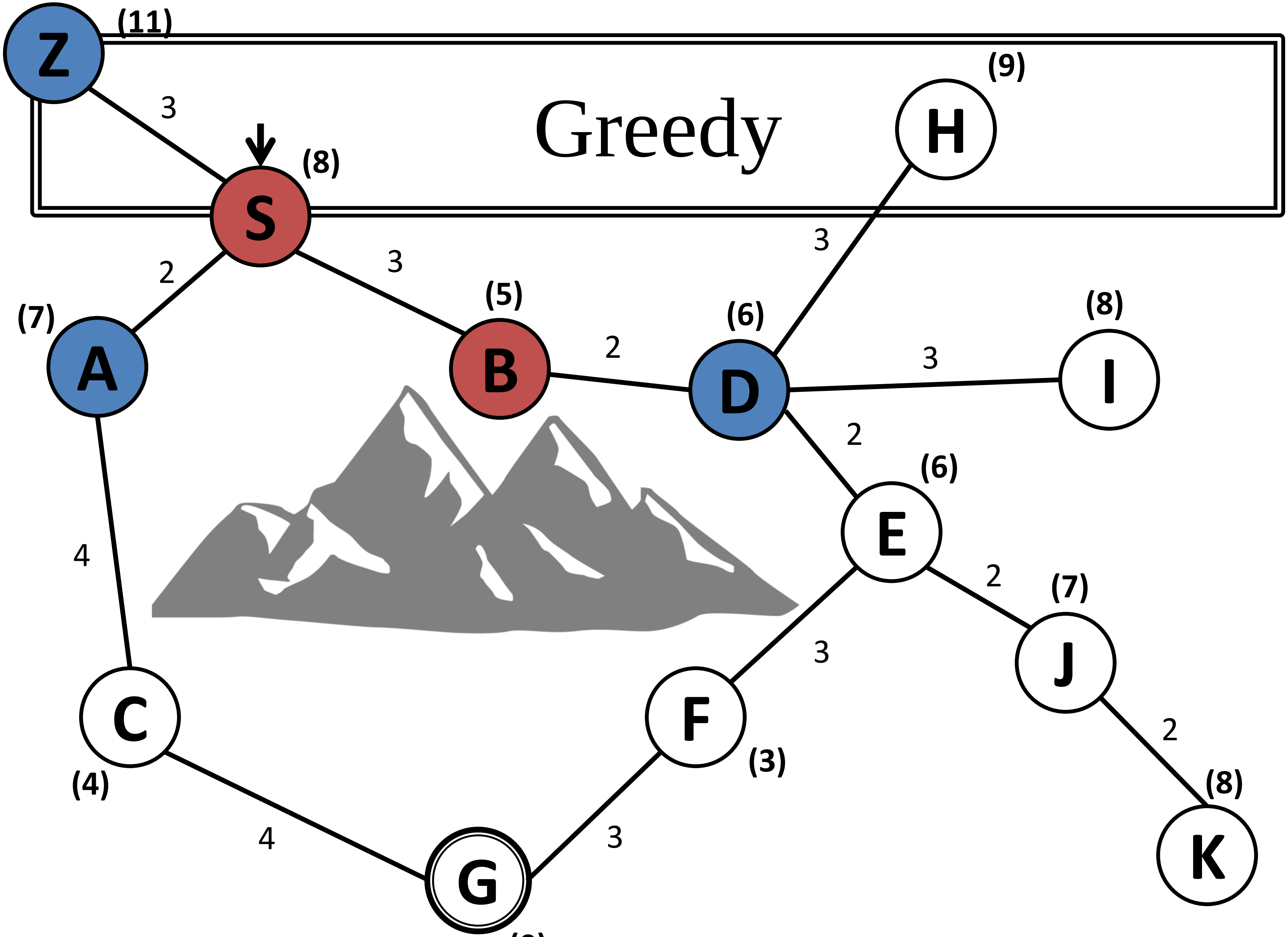
Greedy



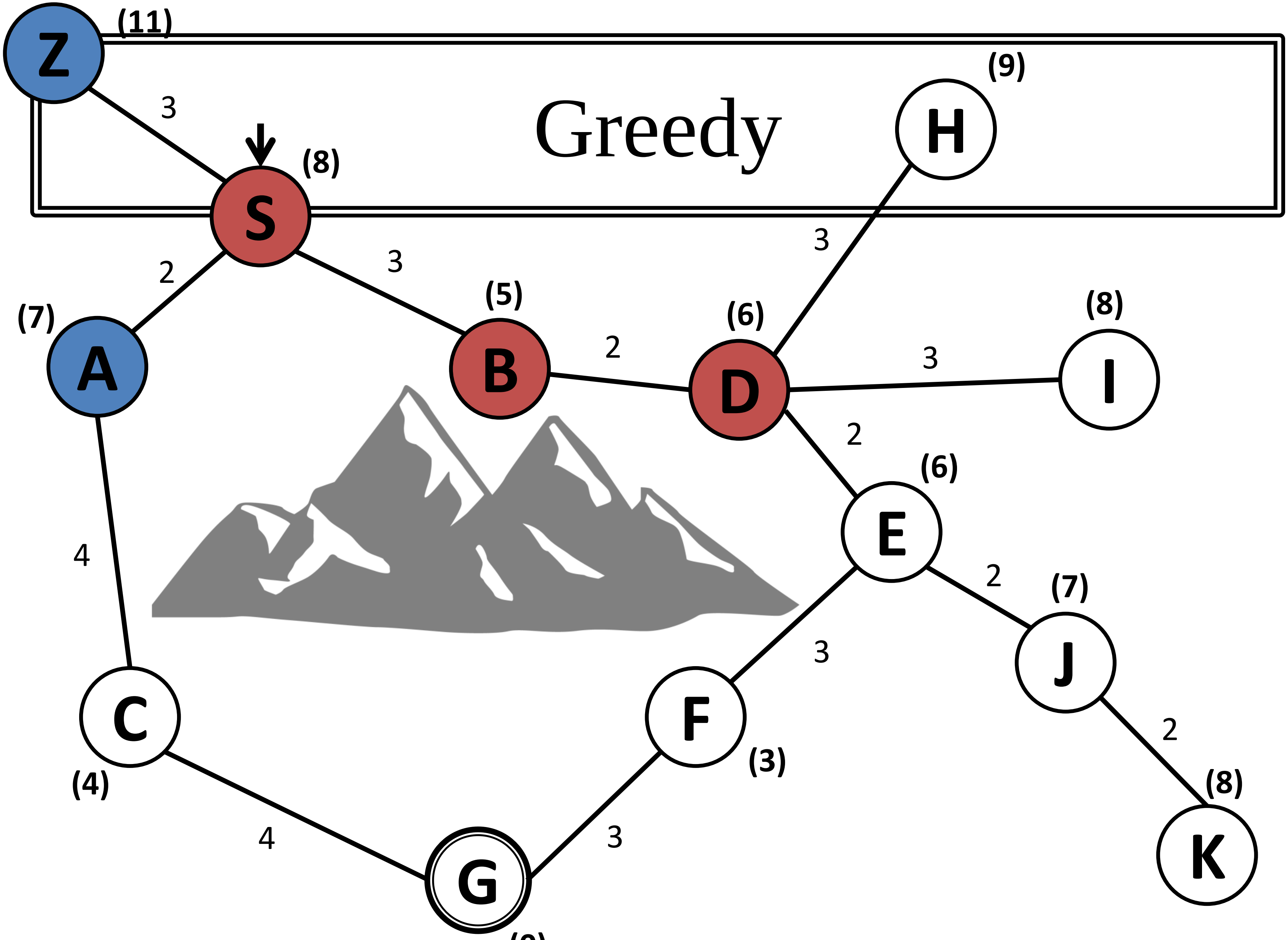
Greedy



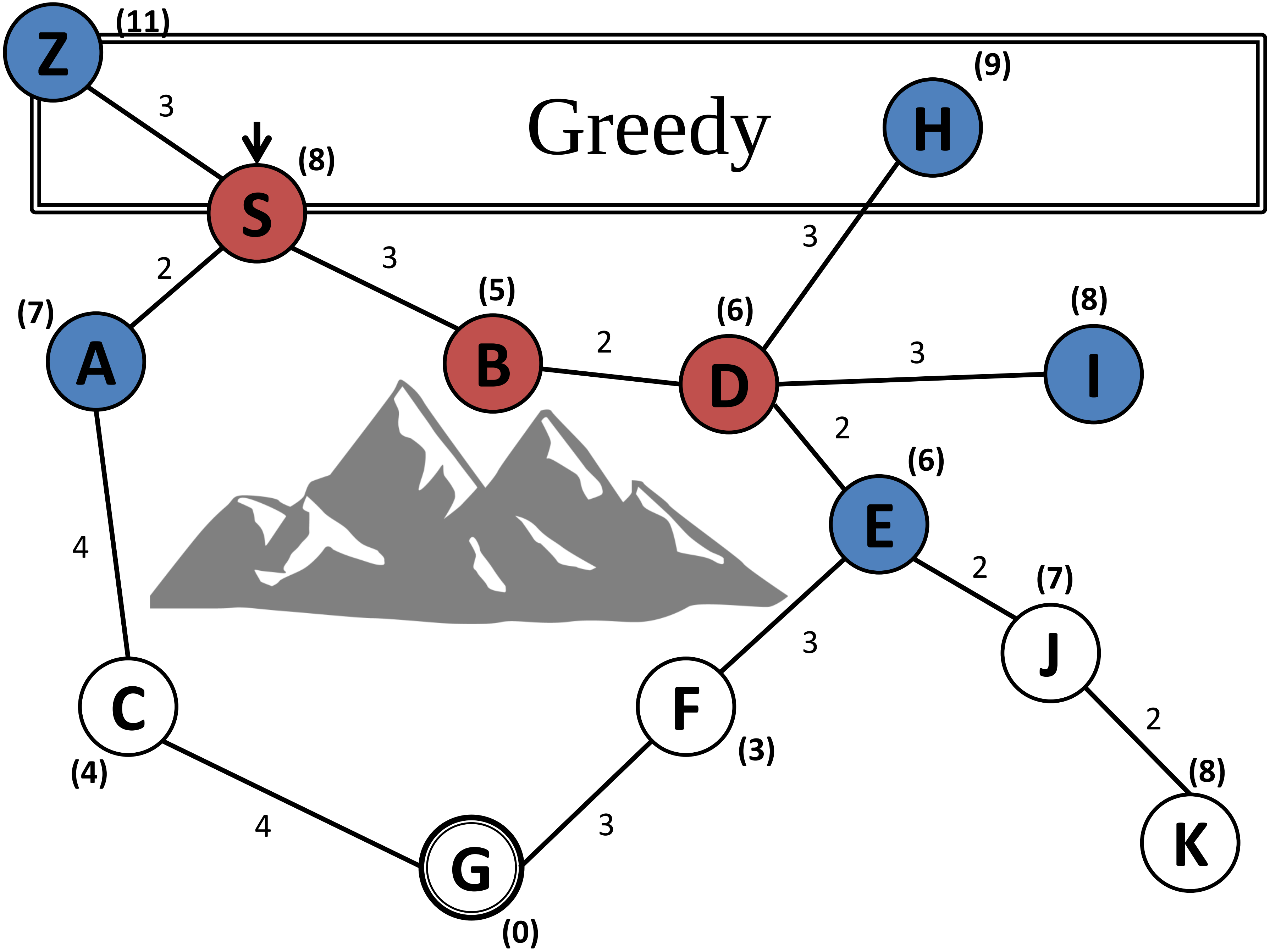
Greedy



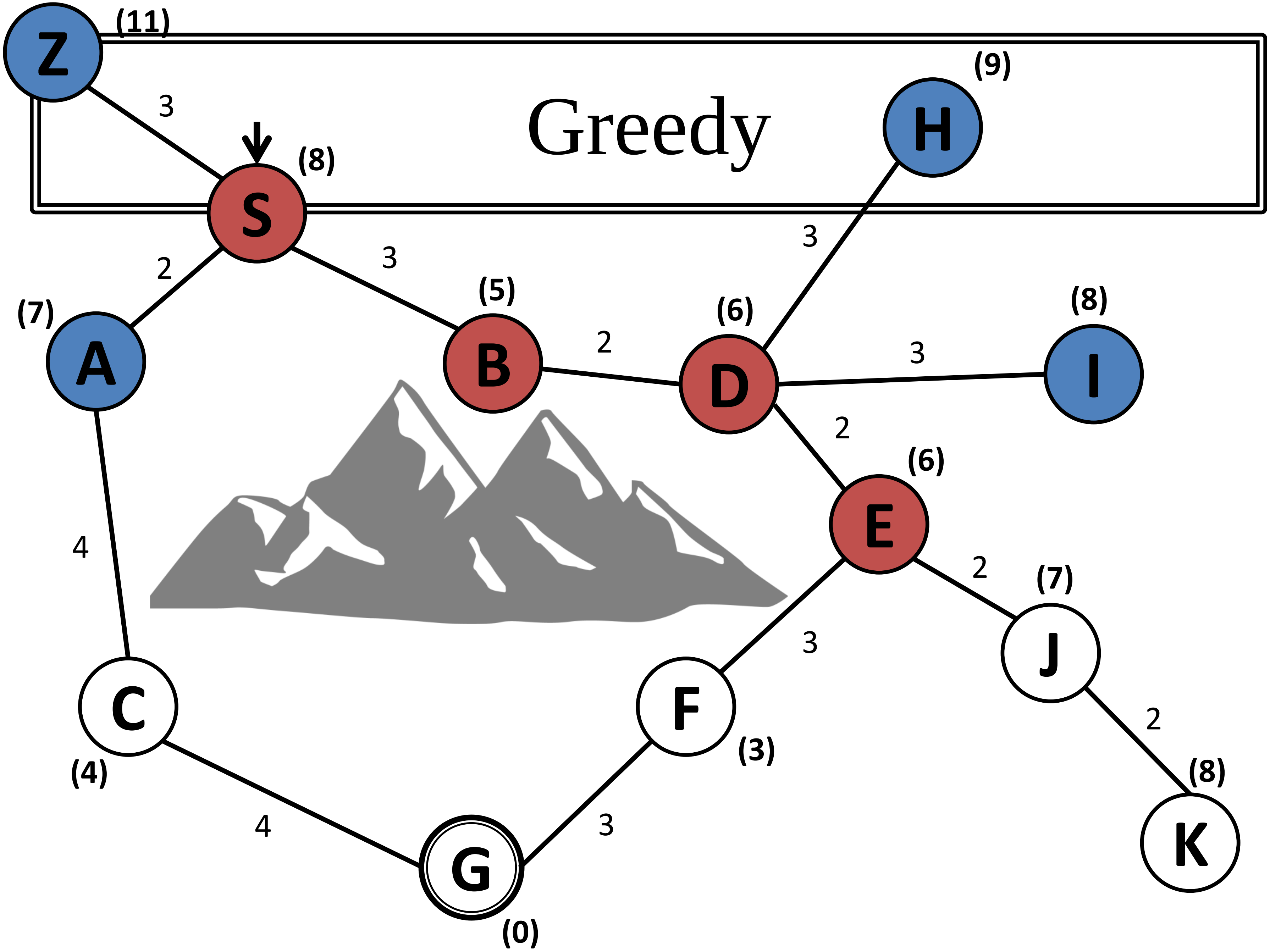
Greedy



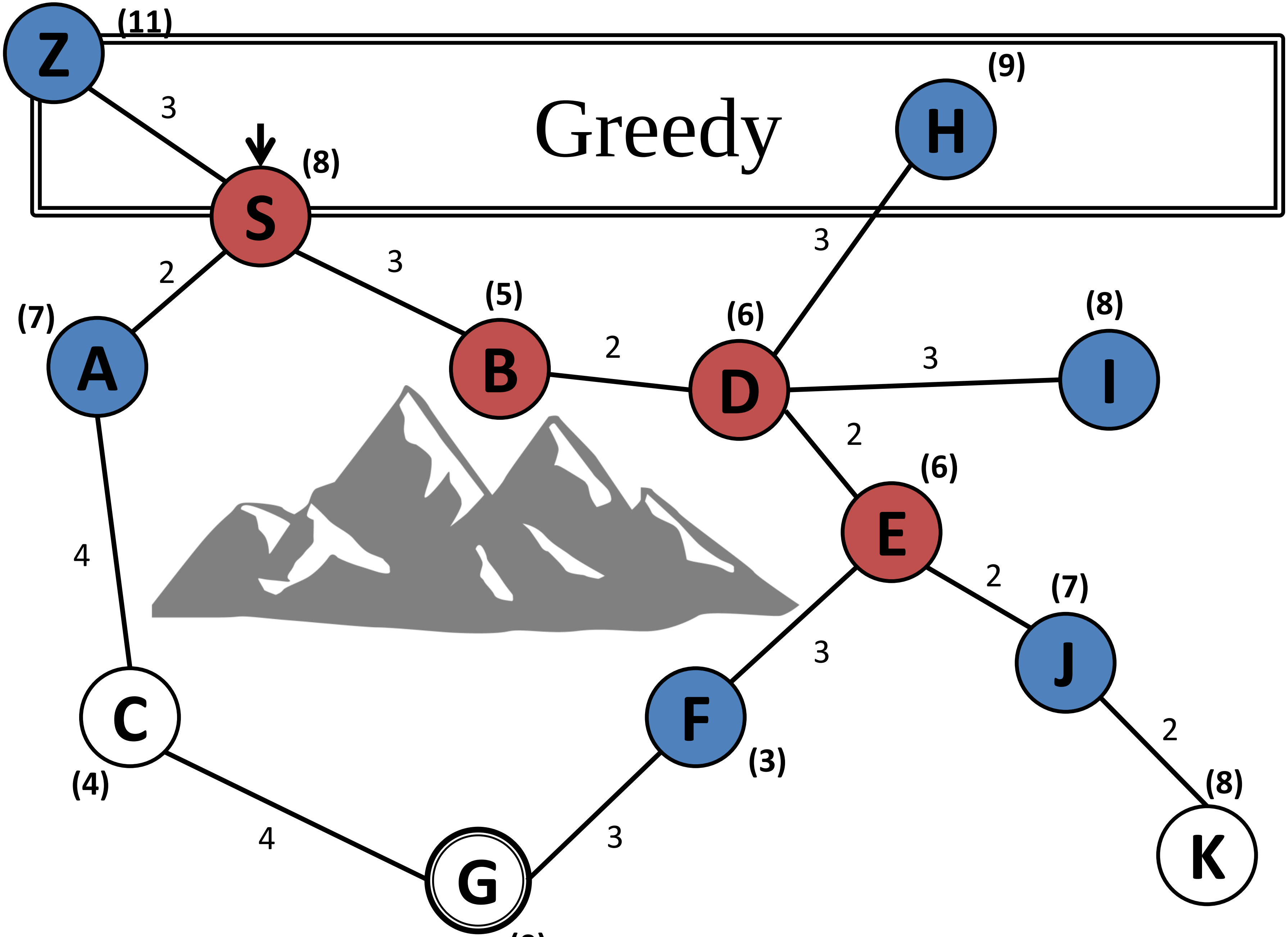
Greedy



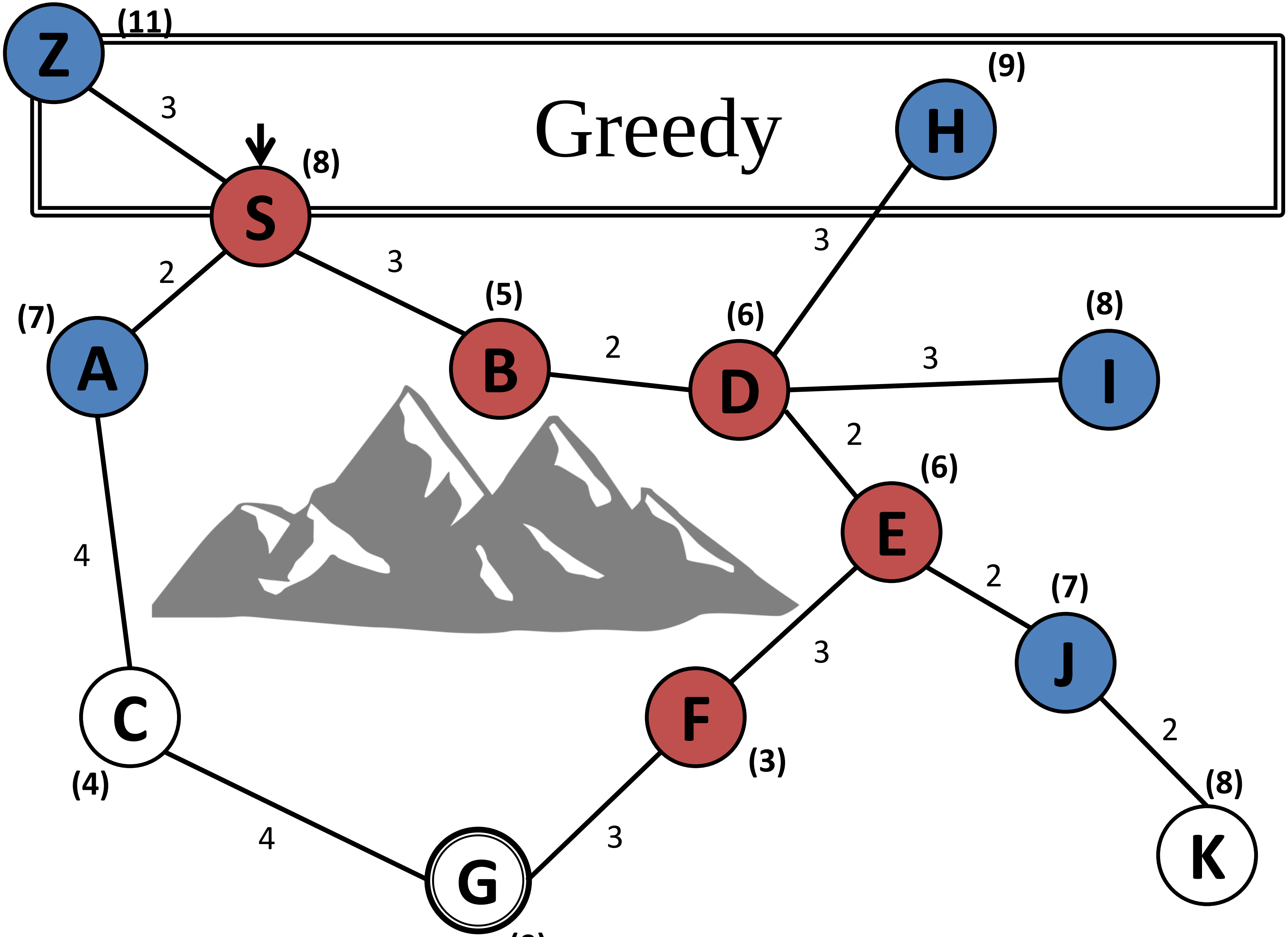
Greedy



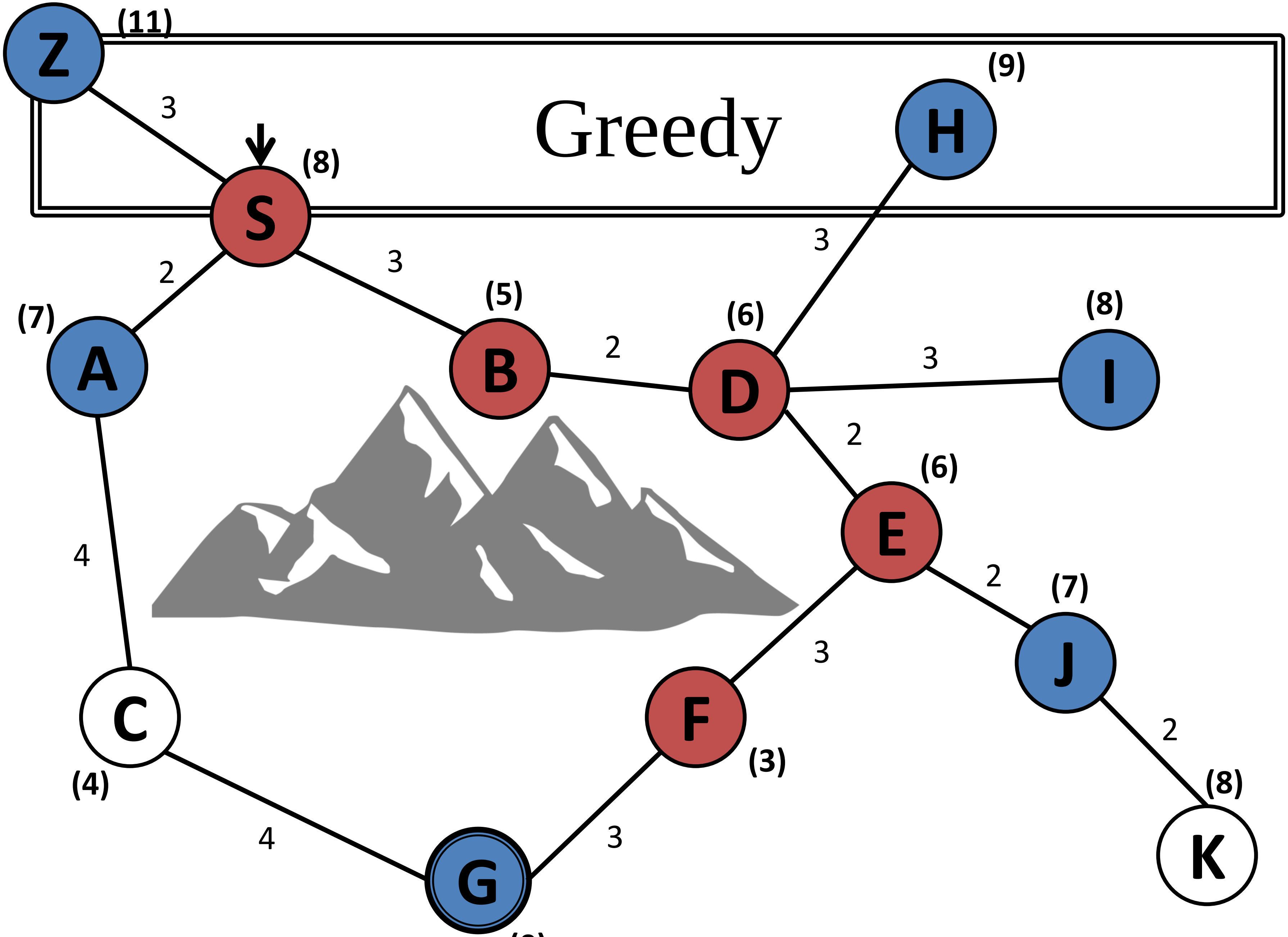
Greedy



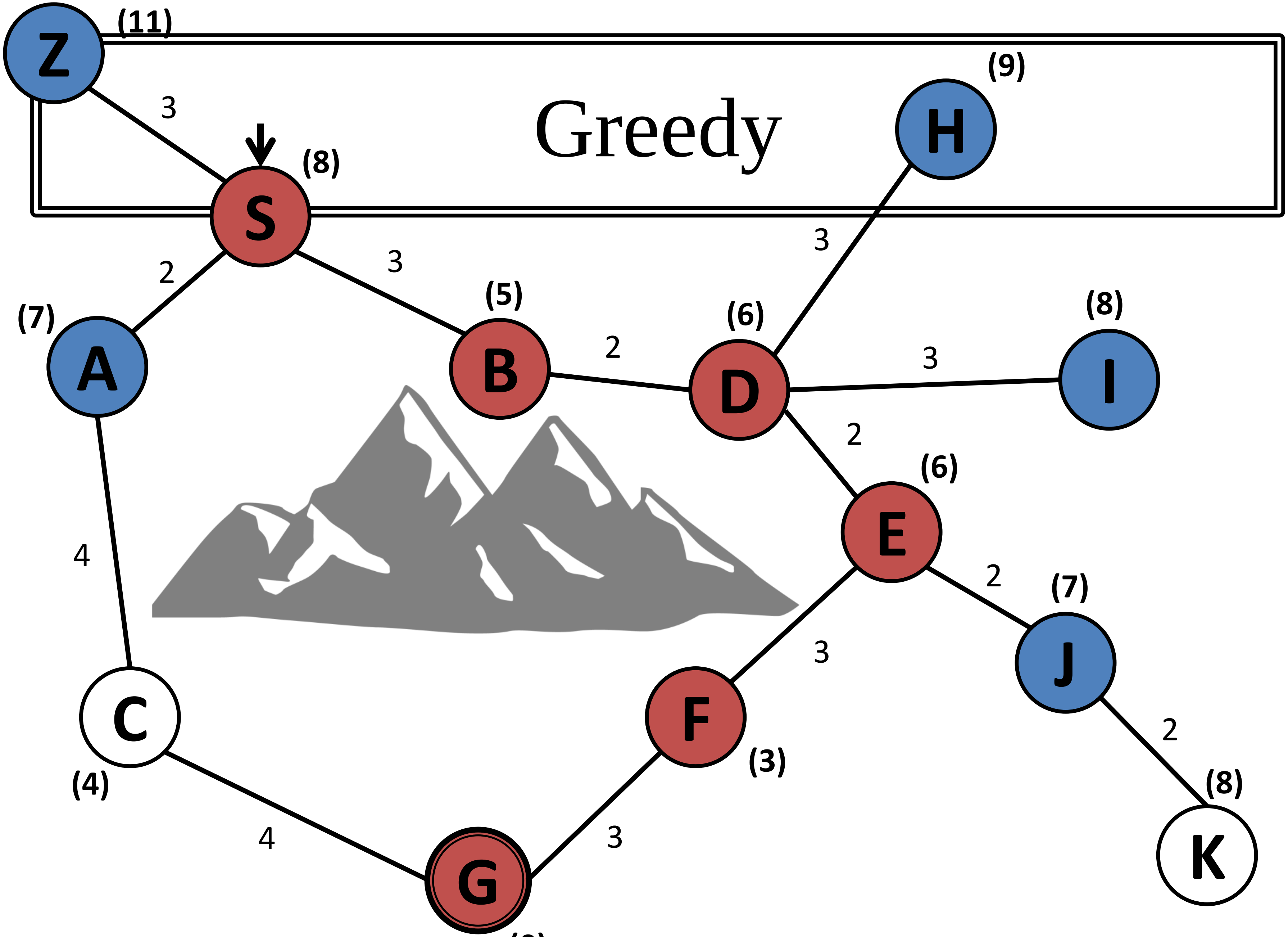
Greedy



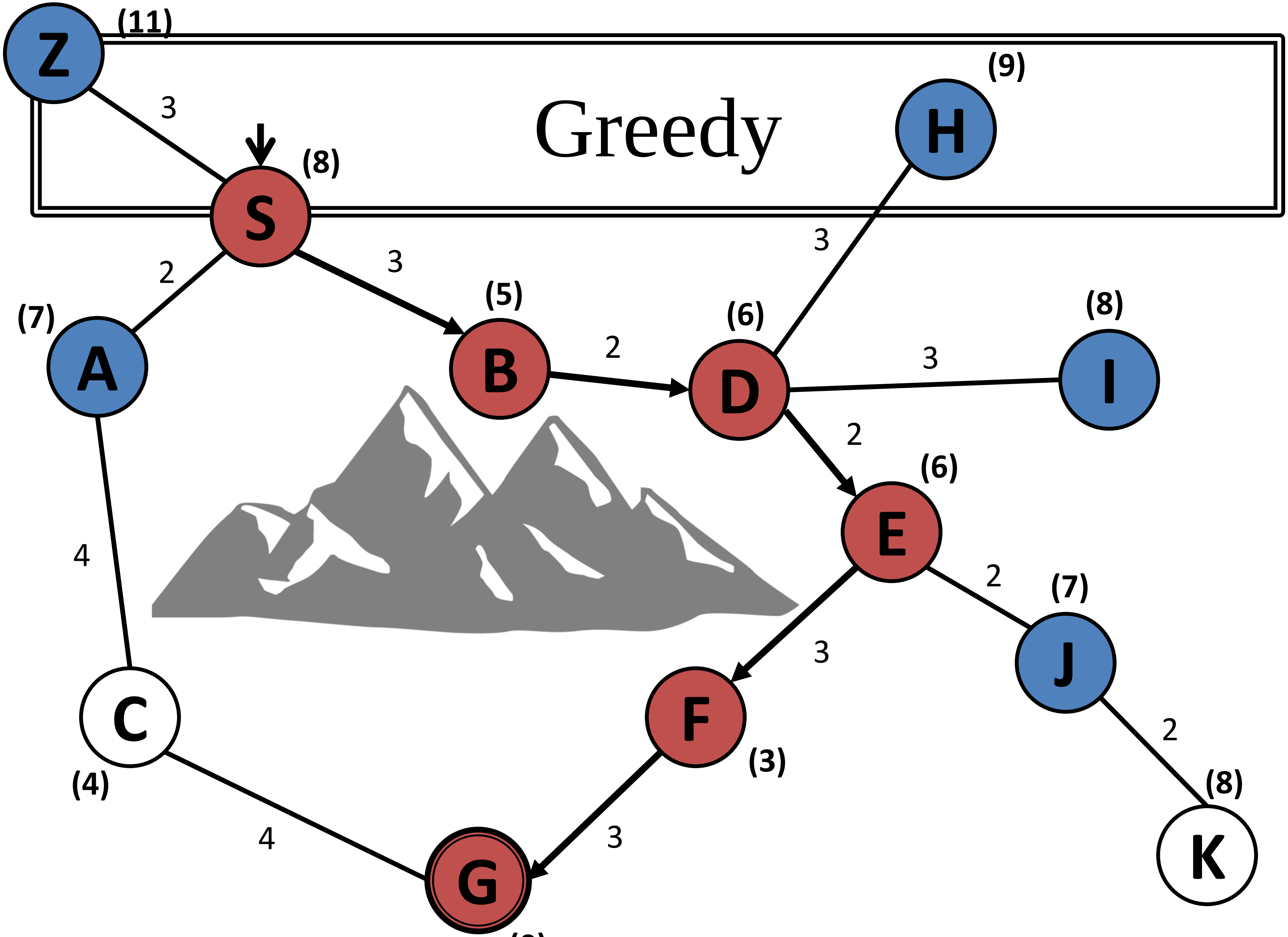
Greedy



Greedy



Greedy



Greedy

- ***Complete?***
  - ***No***, can get stuck in loops, e.g.,

    Iasi → Neamt → Iasi → Neamt →

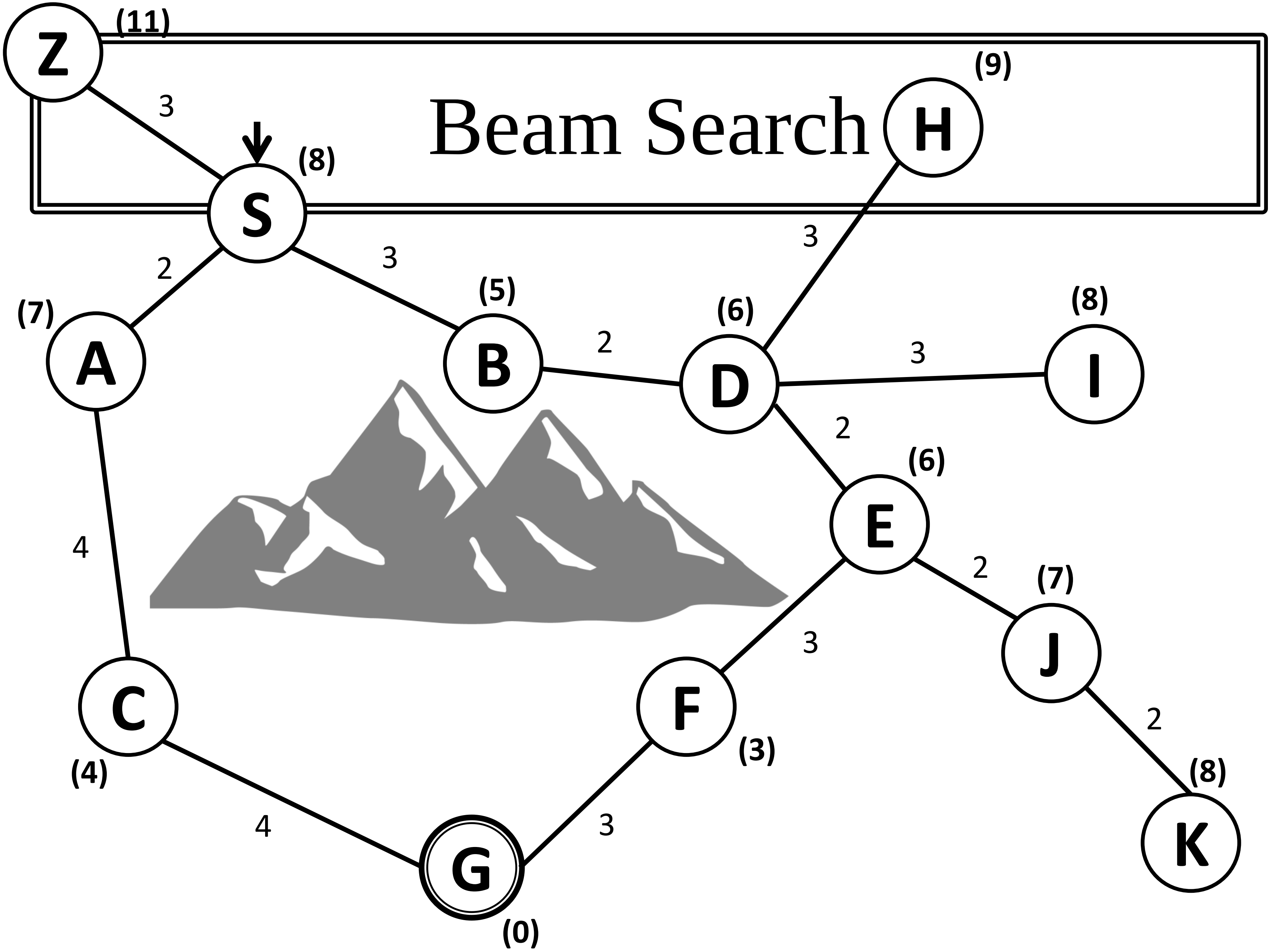
  Complete in finite space with repeated-state checking
- ***Optimal?***
  - ***No***
- ***Time?***
  - ***O(b^m)***, but a good heuristic can give dramatic improvement
- ***Space?***
  - ***O(b^m)*** - keeps all nodes in memory

Beam Search

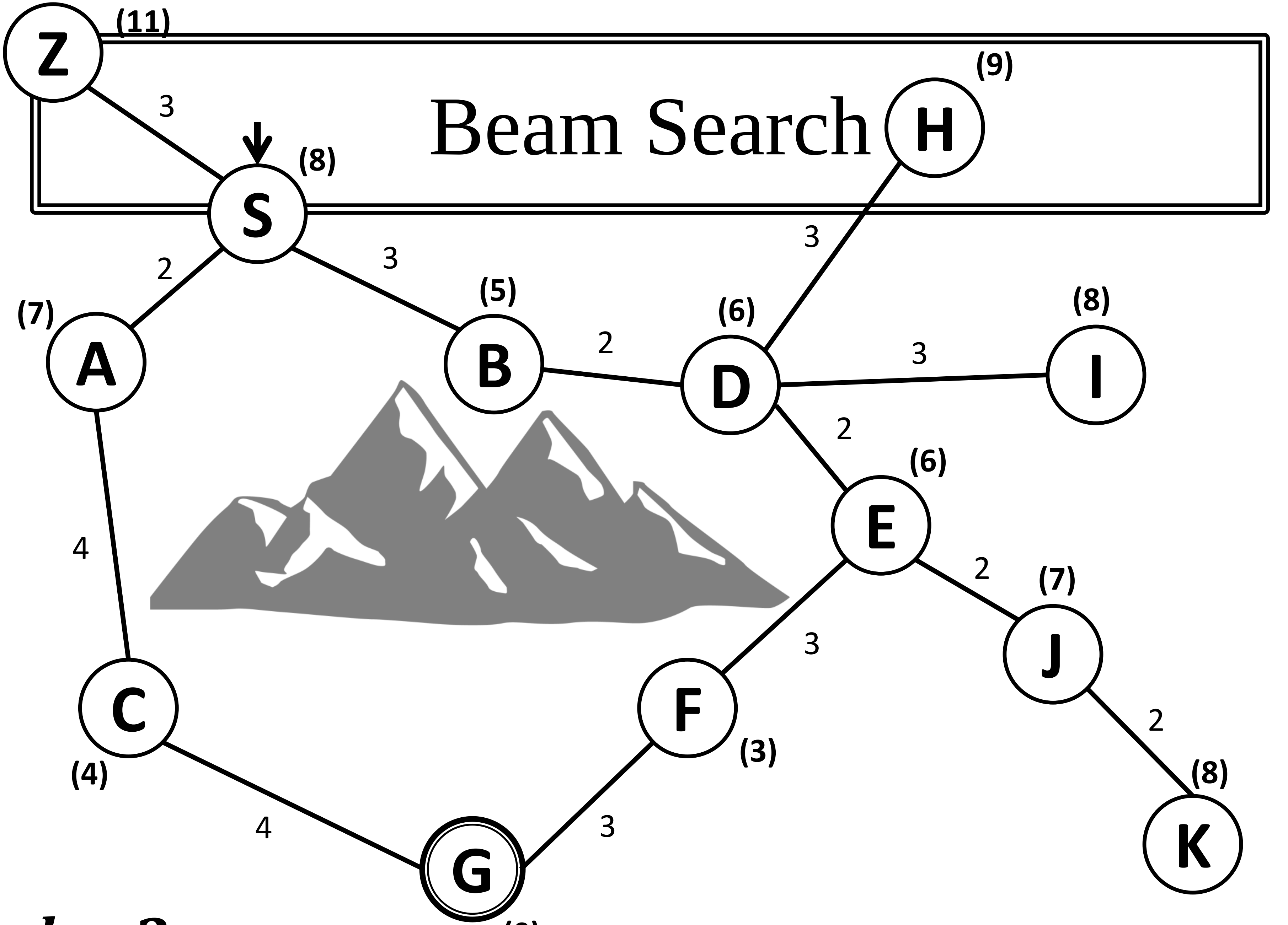
- = *greedy best-first search* with queue limit ***l***,

  i.e., keeps ***l*** nodes in the queue

Beam Search

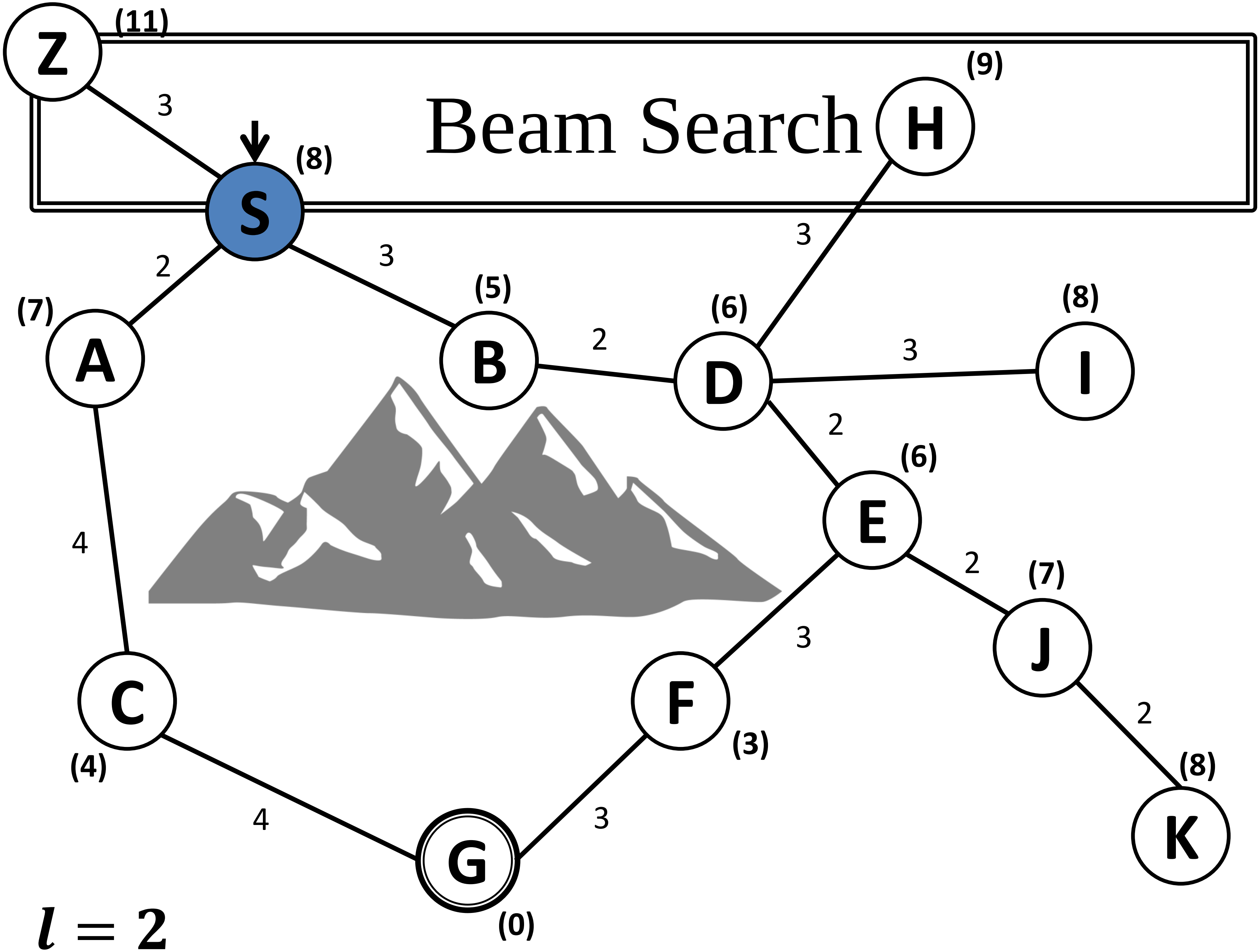


Beam Search



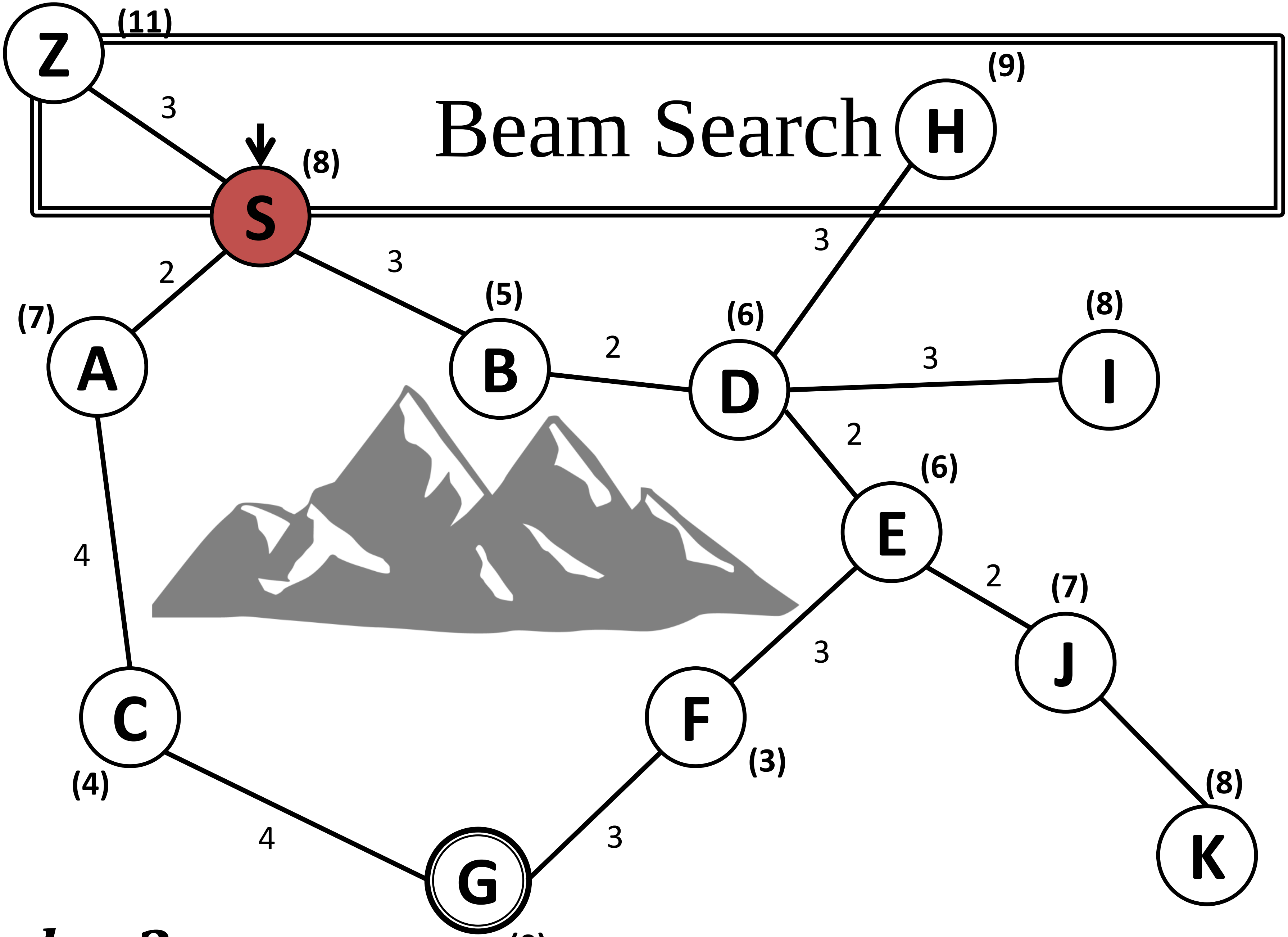
$l = 2$

Beam Search



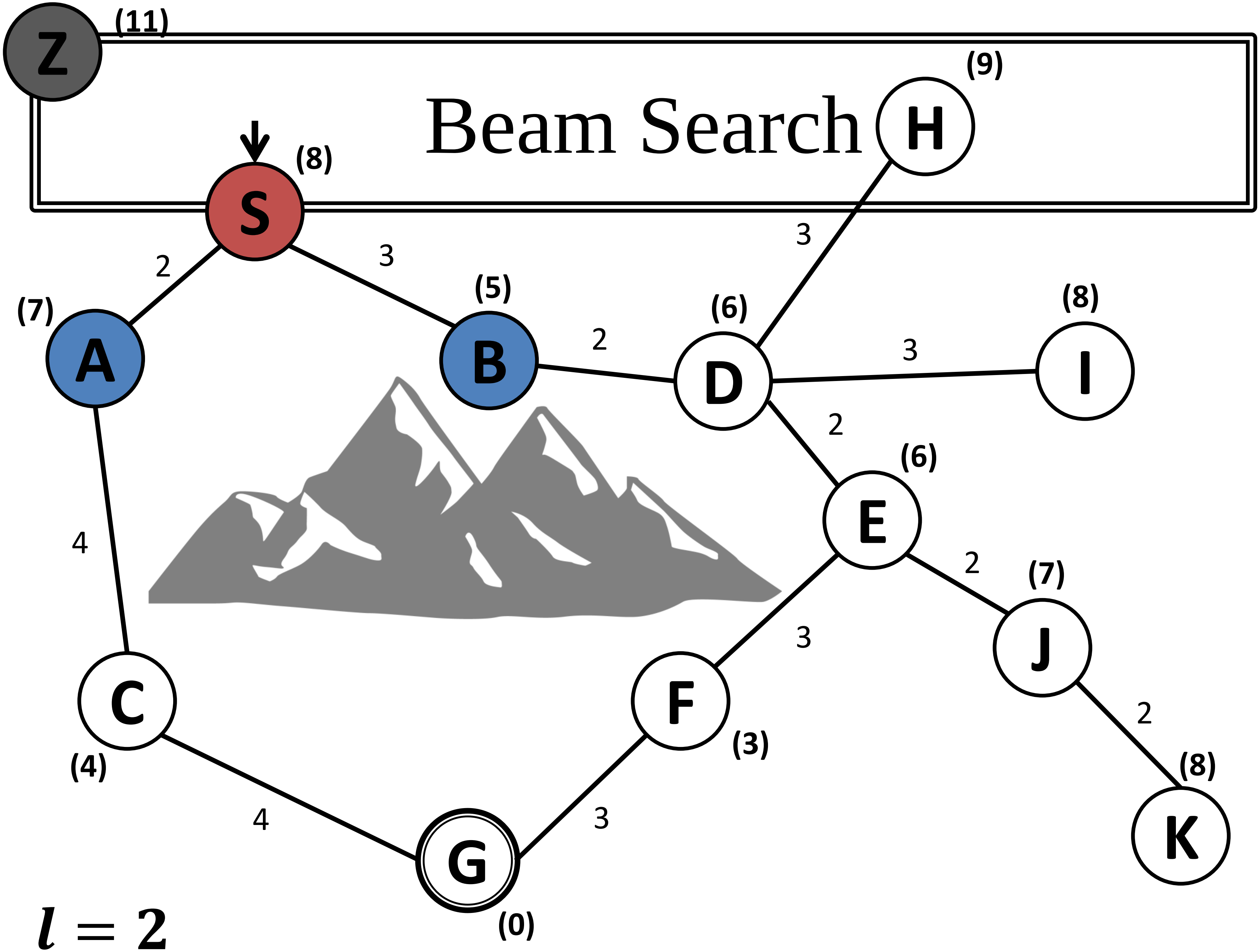
$l = 2$

Beam Search



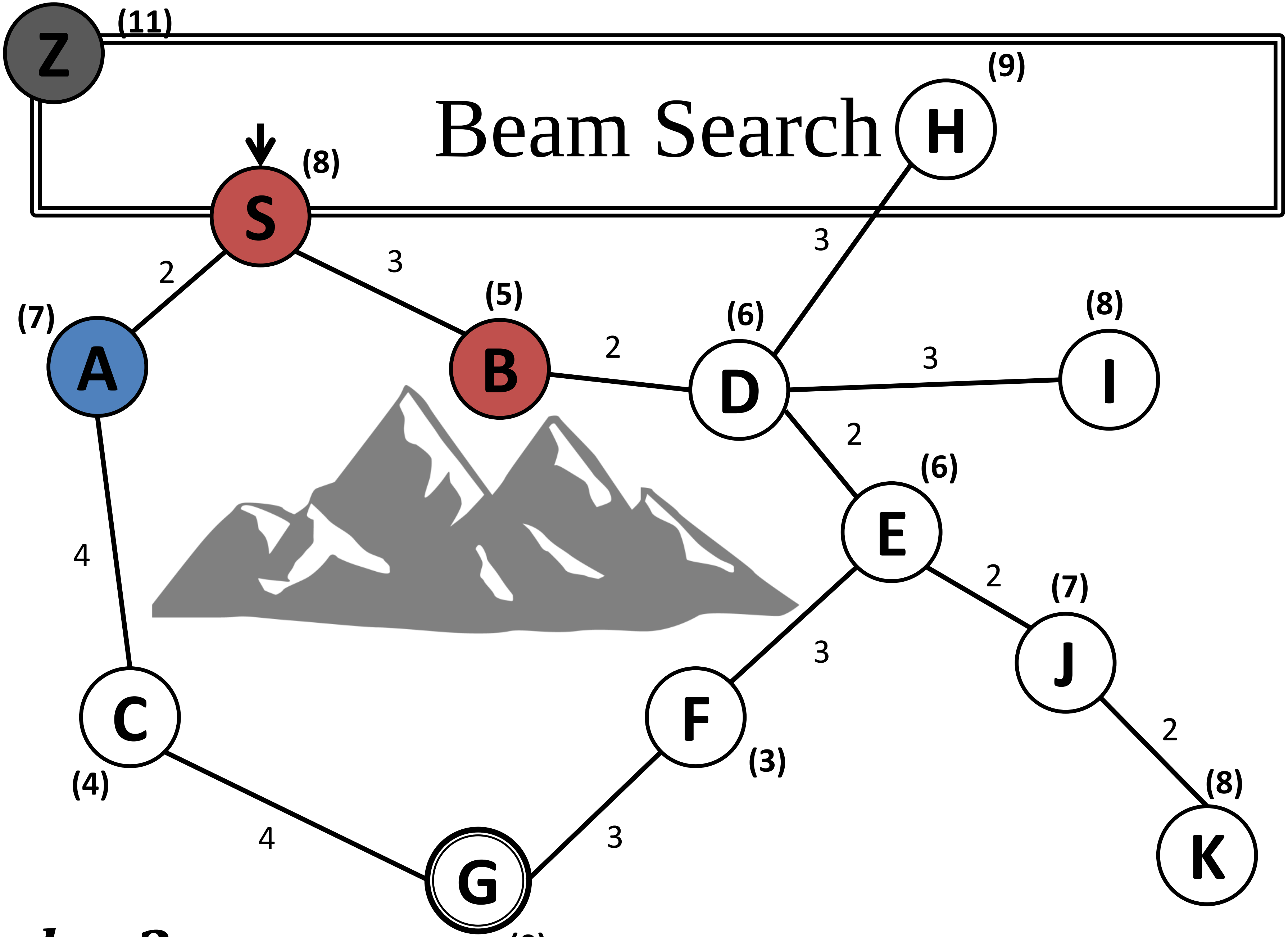
$l = 2$

Beam Search



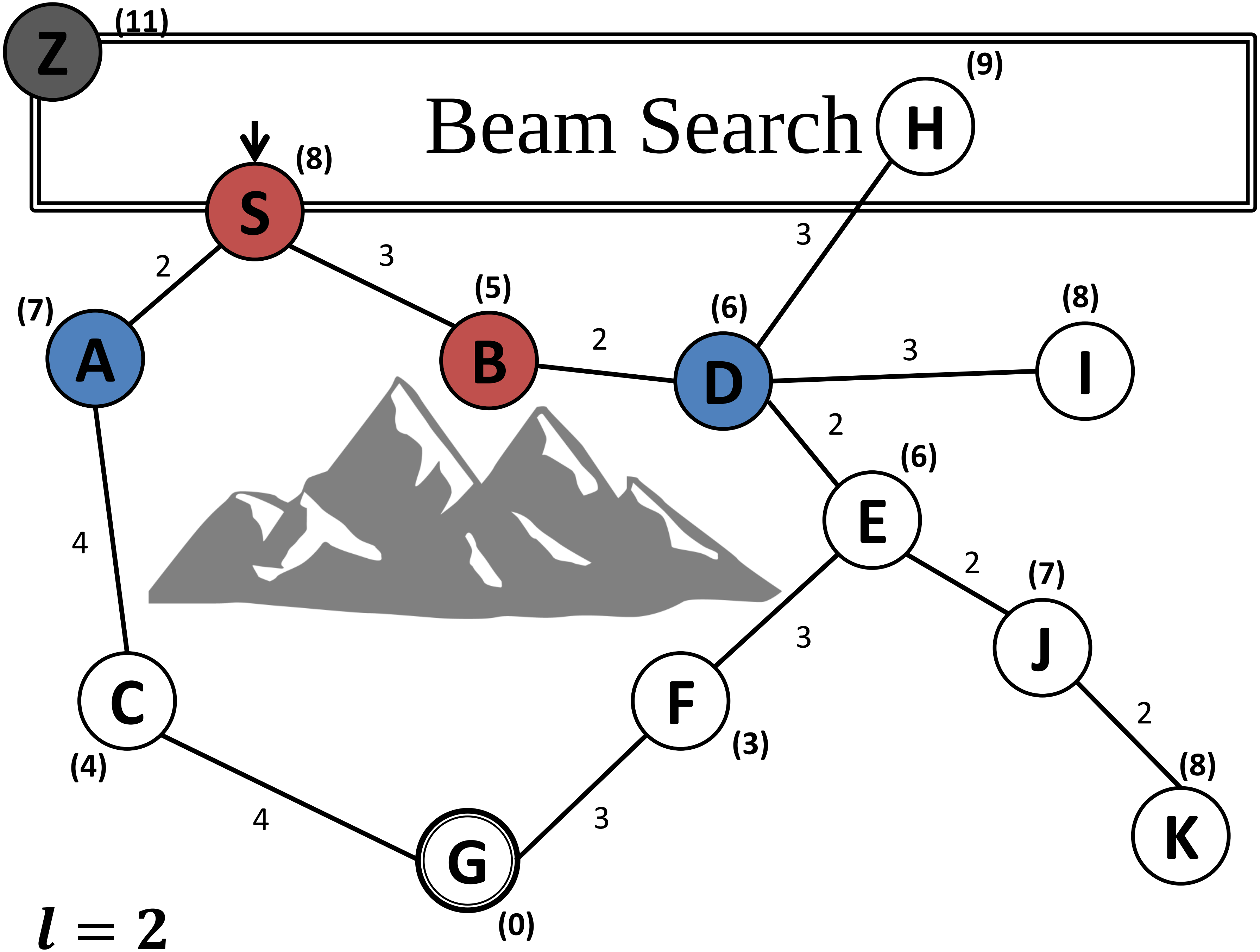
$l = 2$

Beam Search



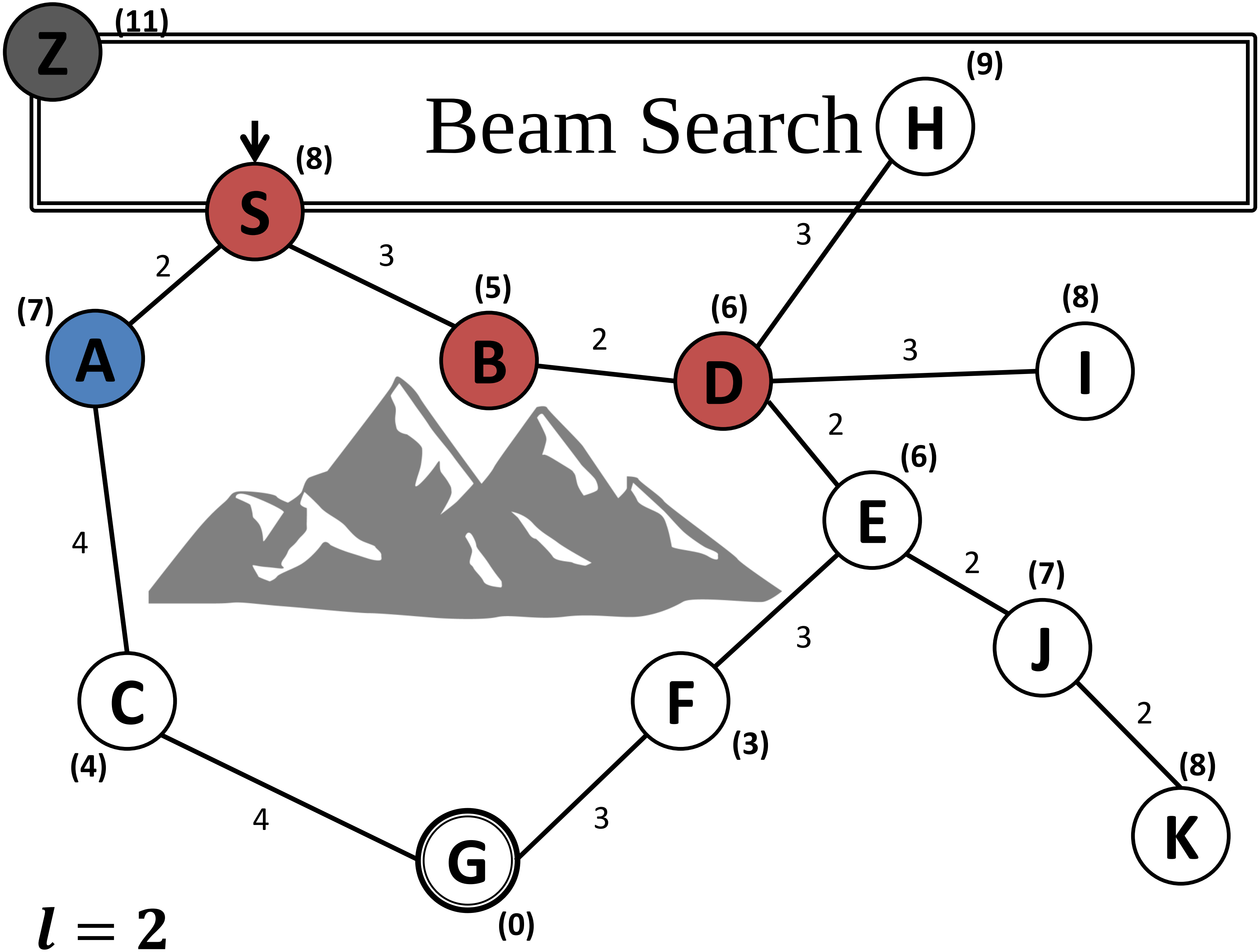
$l = 2$

Beam Search



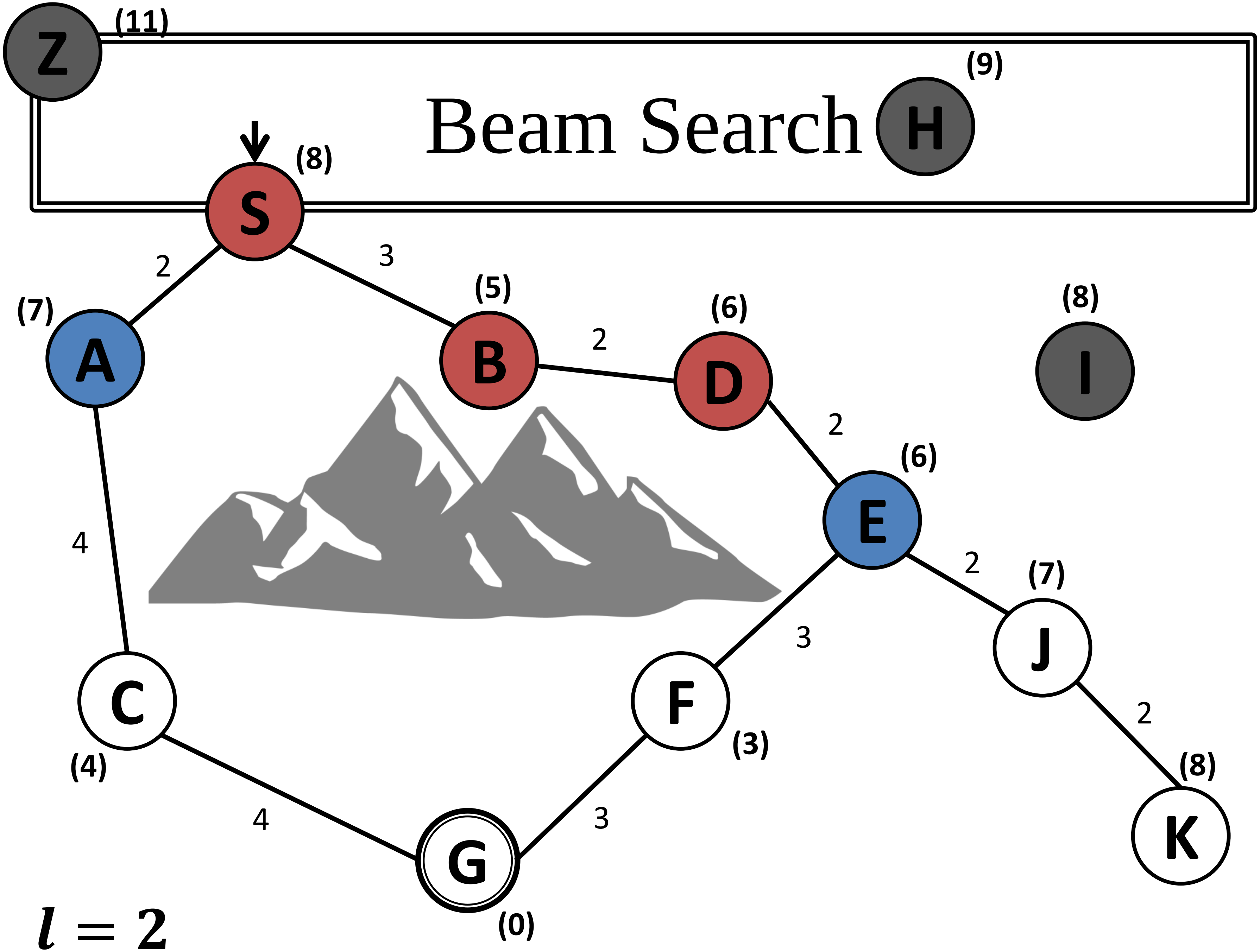
$l = 2$

Beam Search



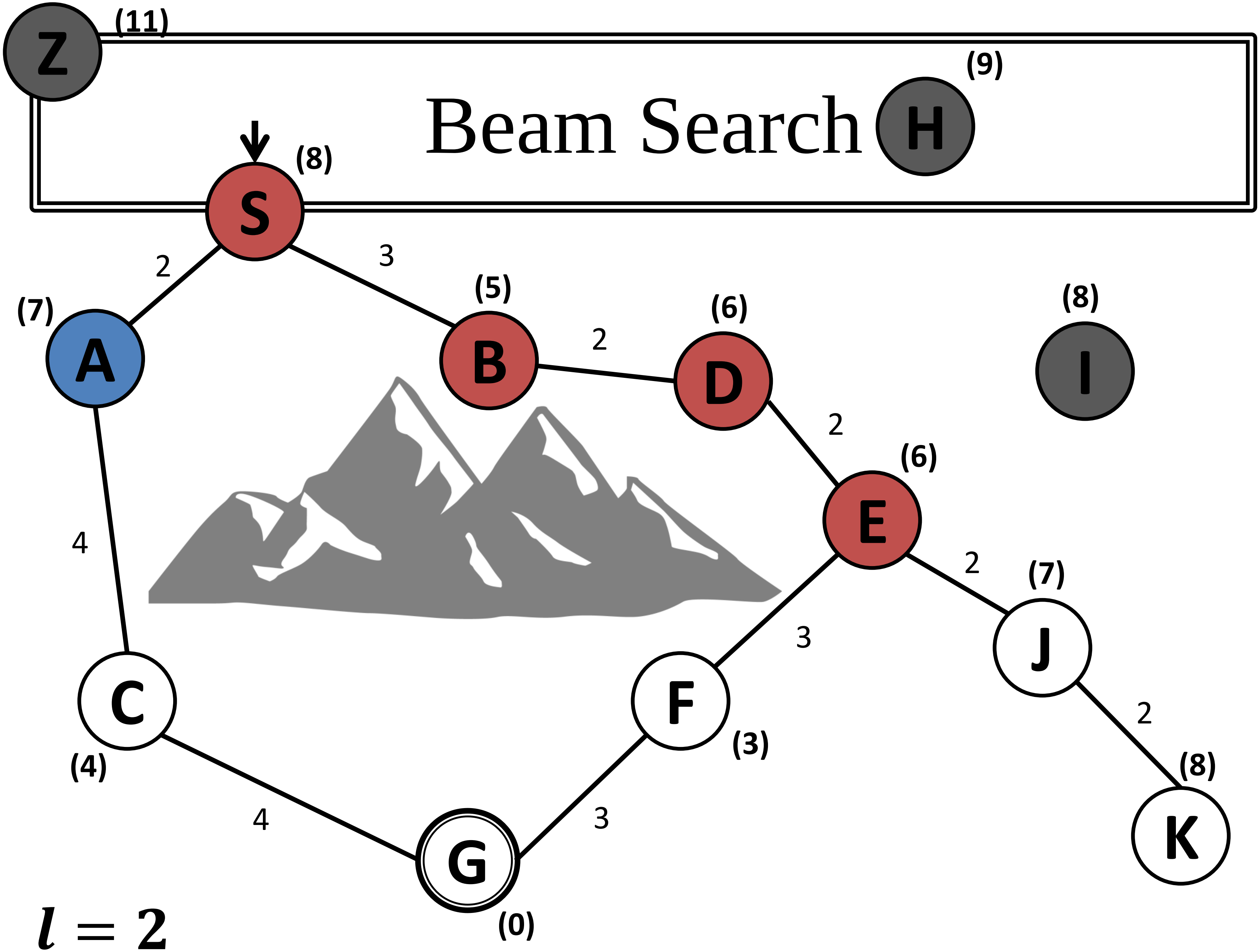
$l = 2$

Beam Search



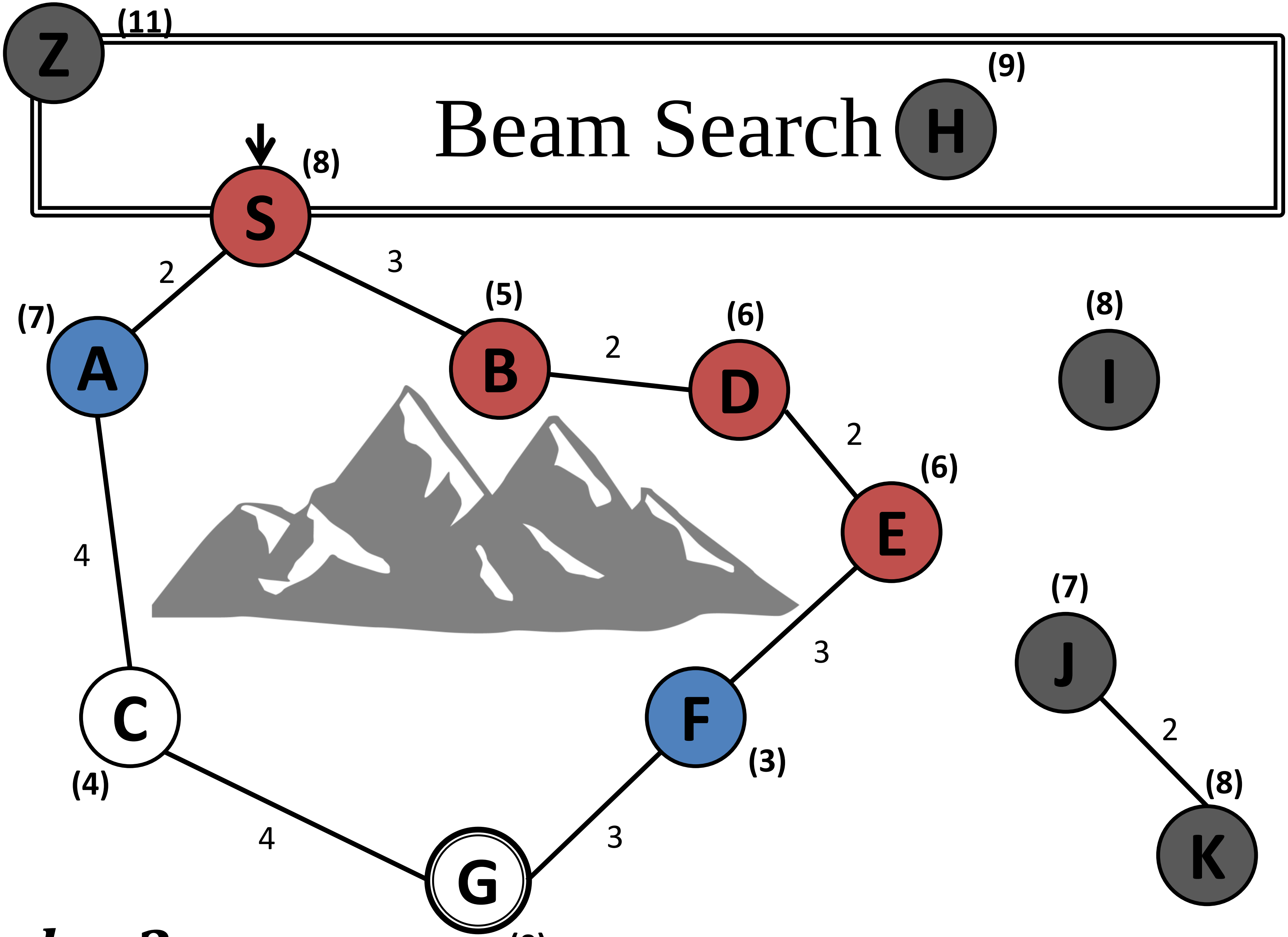
$l = 2$

Beam Search



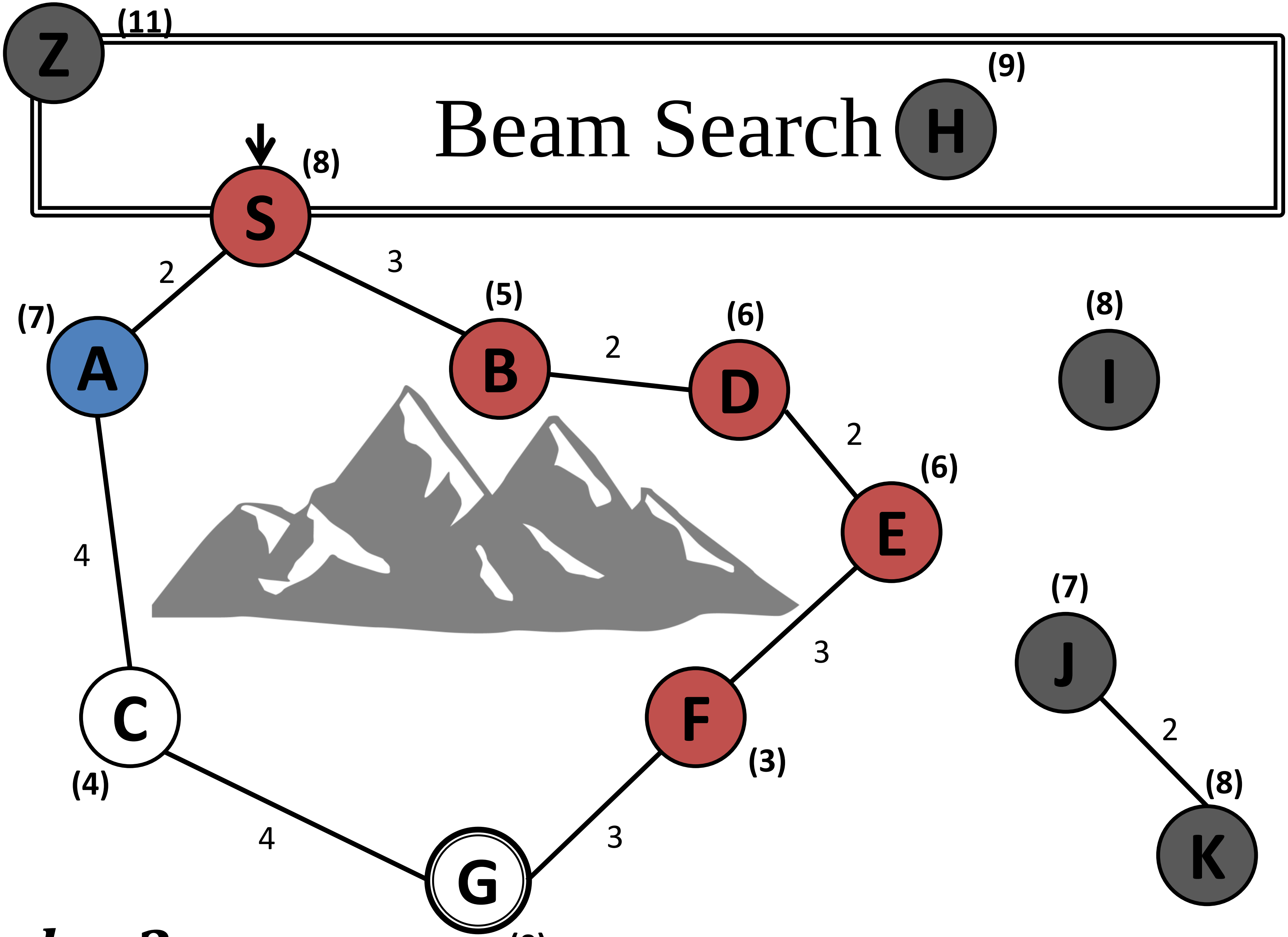
$l = 2$

Beam Search



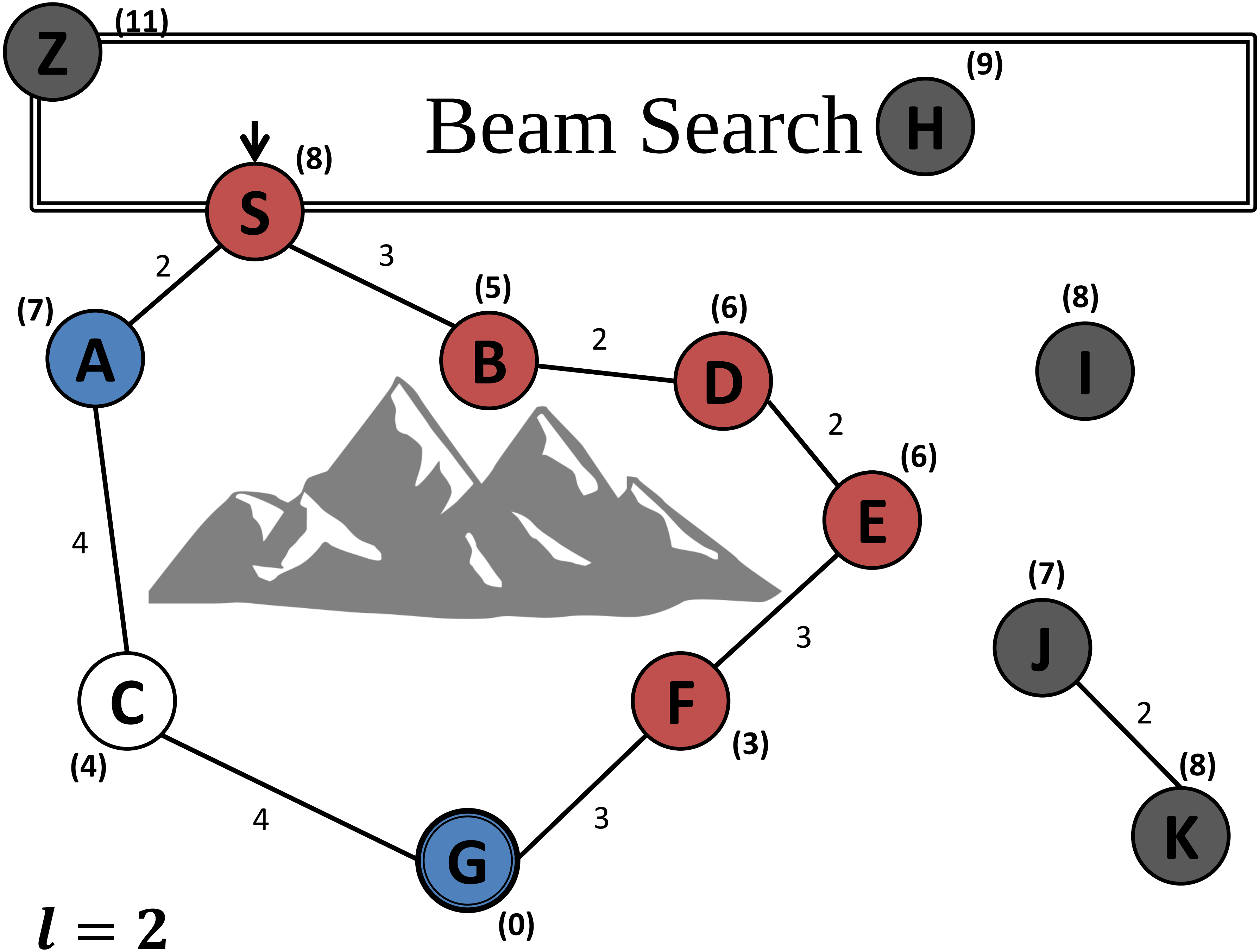
$l = 2$

Beam Search



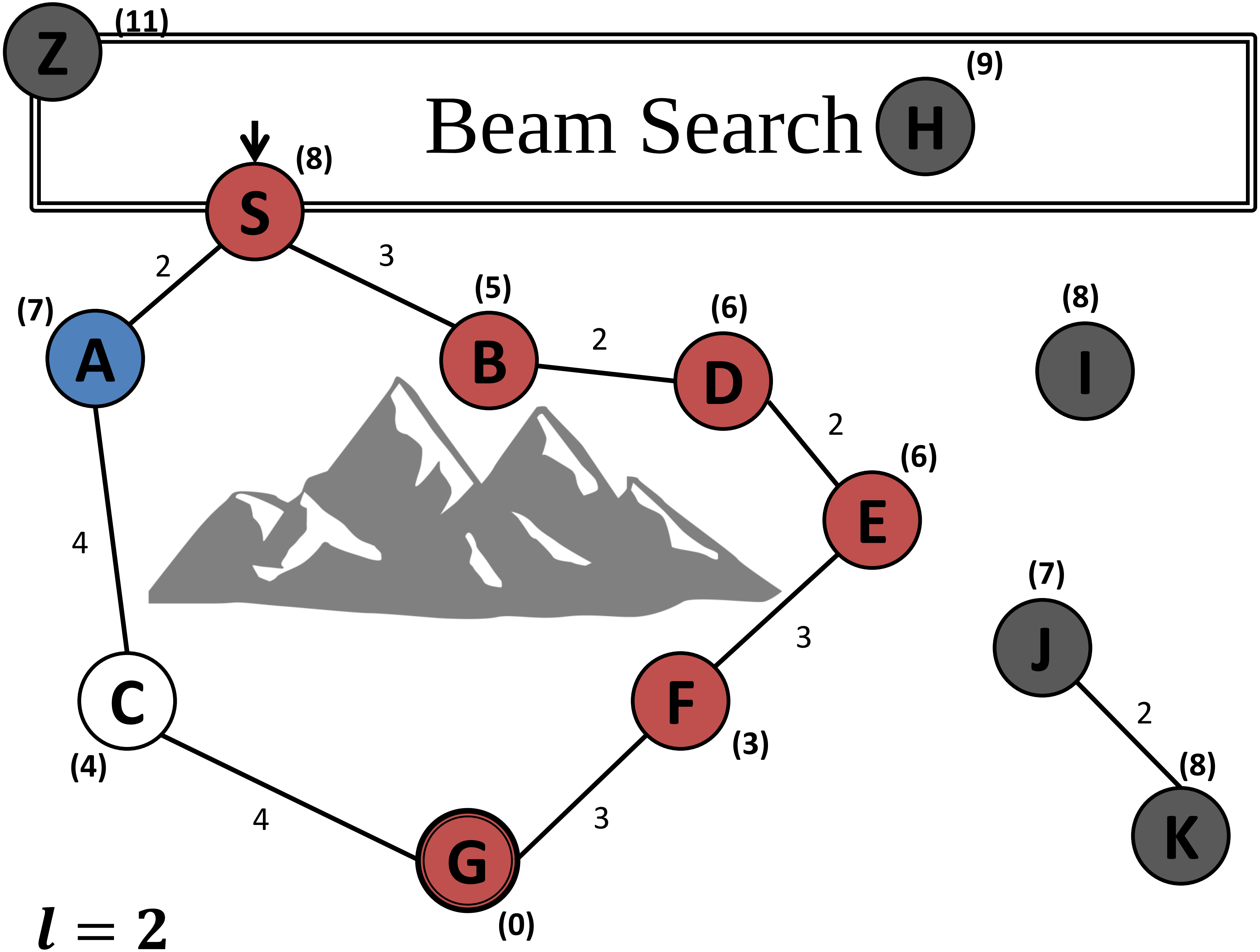
$l = 2$

Beam Search



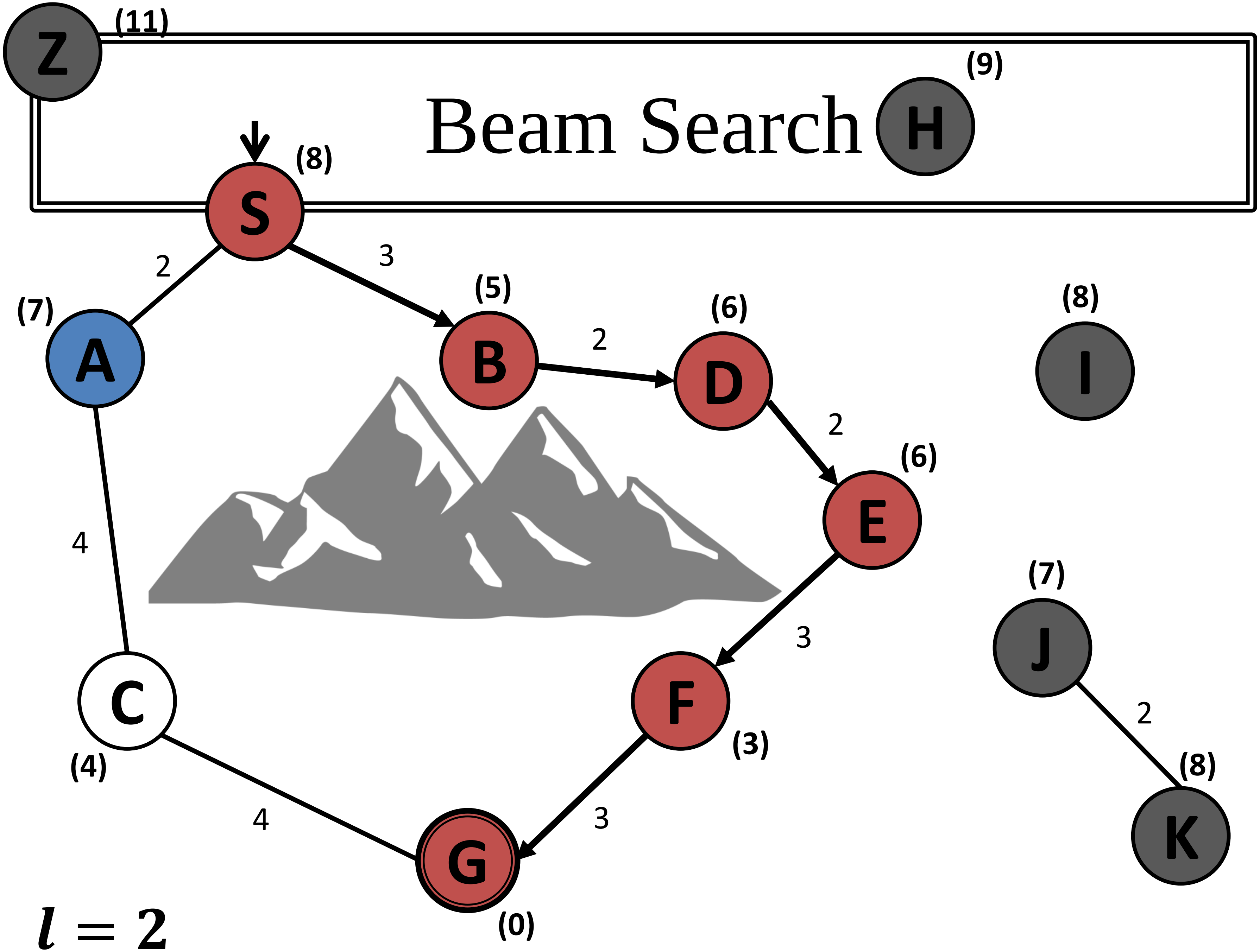
$l = 2$

Beam Search



$l = 2$

Beam Search



$l = 2$

Beam Search

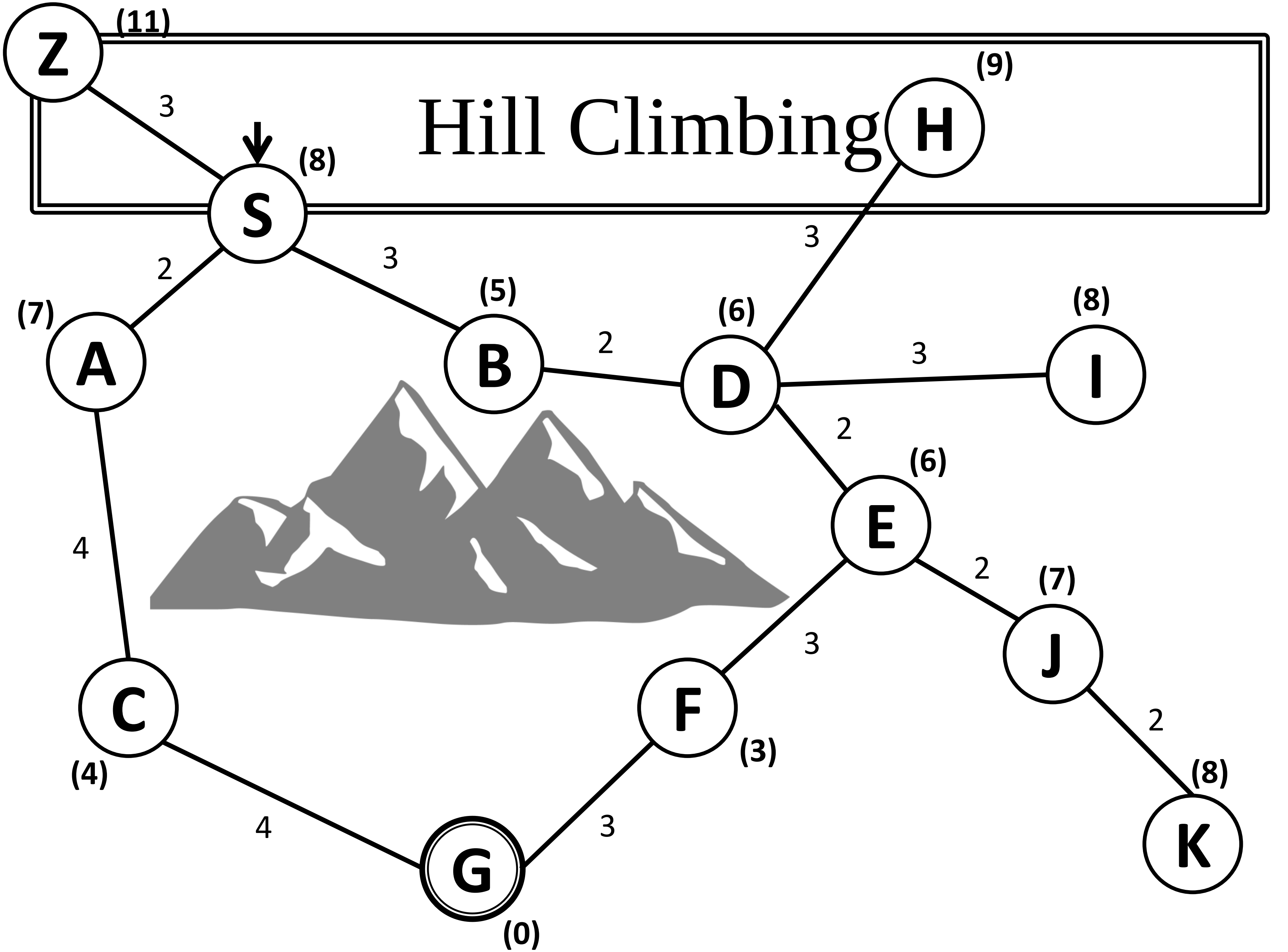
- ***Complete?***
  - ***No***, local search
- ***Optimal?***
  - ***No***
- ***Time?***
  - ***O(blm)*** – linear time
- ***Space?***
  - ***O(bl)*** – linear space

Hill Climbing

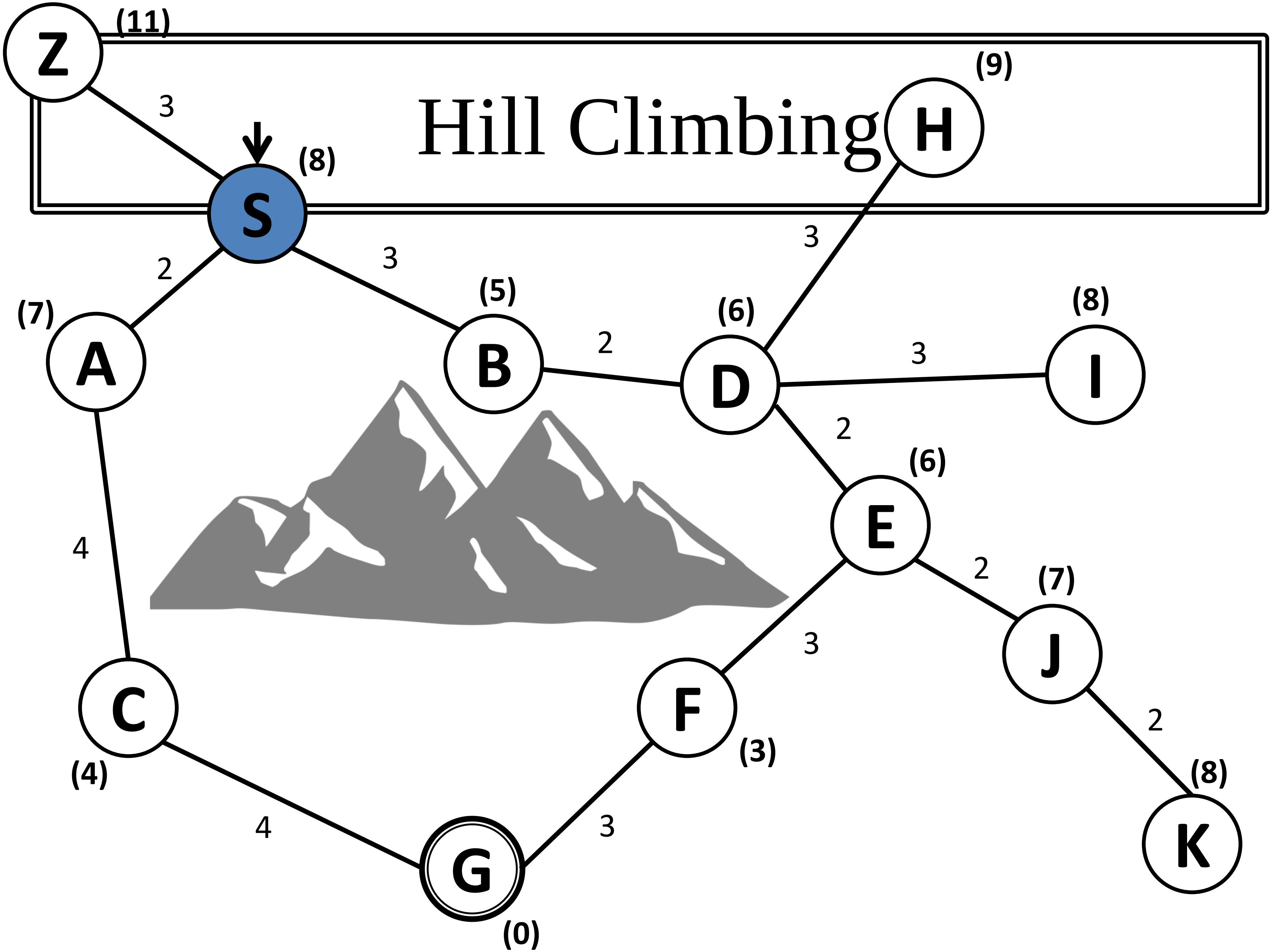
- = *greedy best-first search* with queue limit ***l = 1***,

  i.e., keeps only the best node

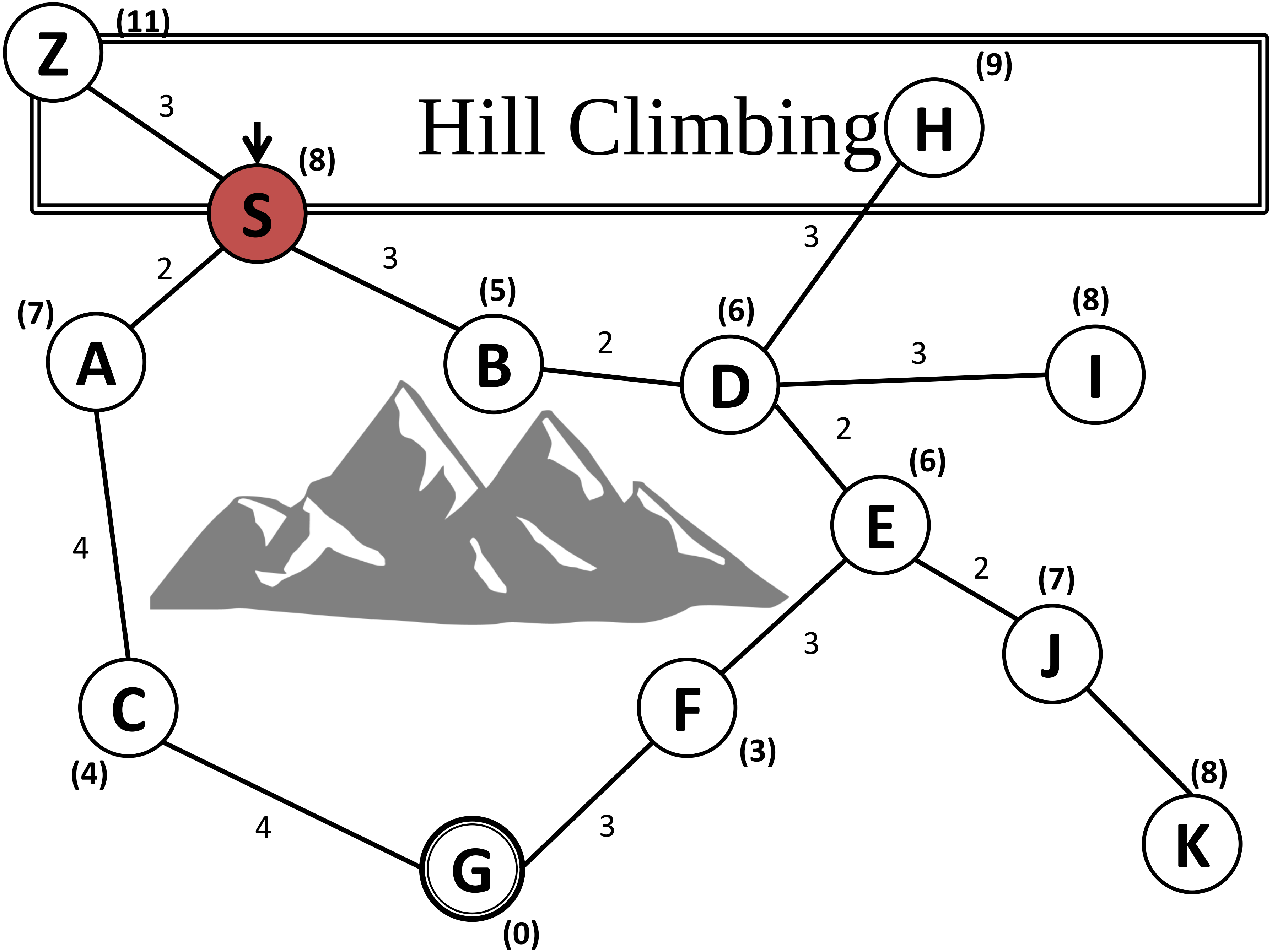
Hill Climbing



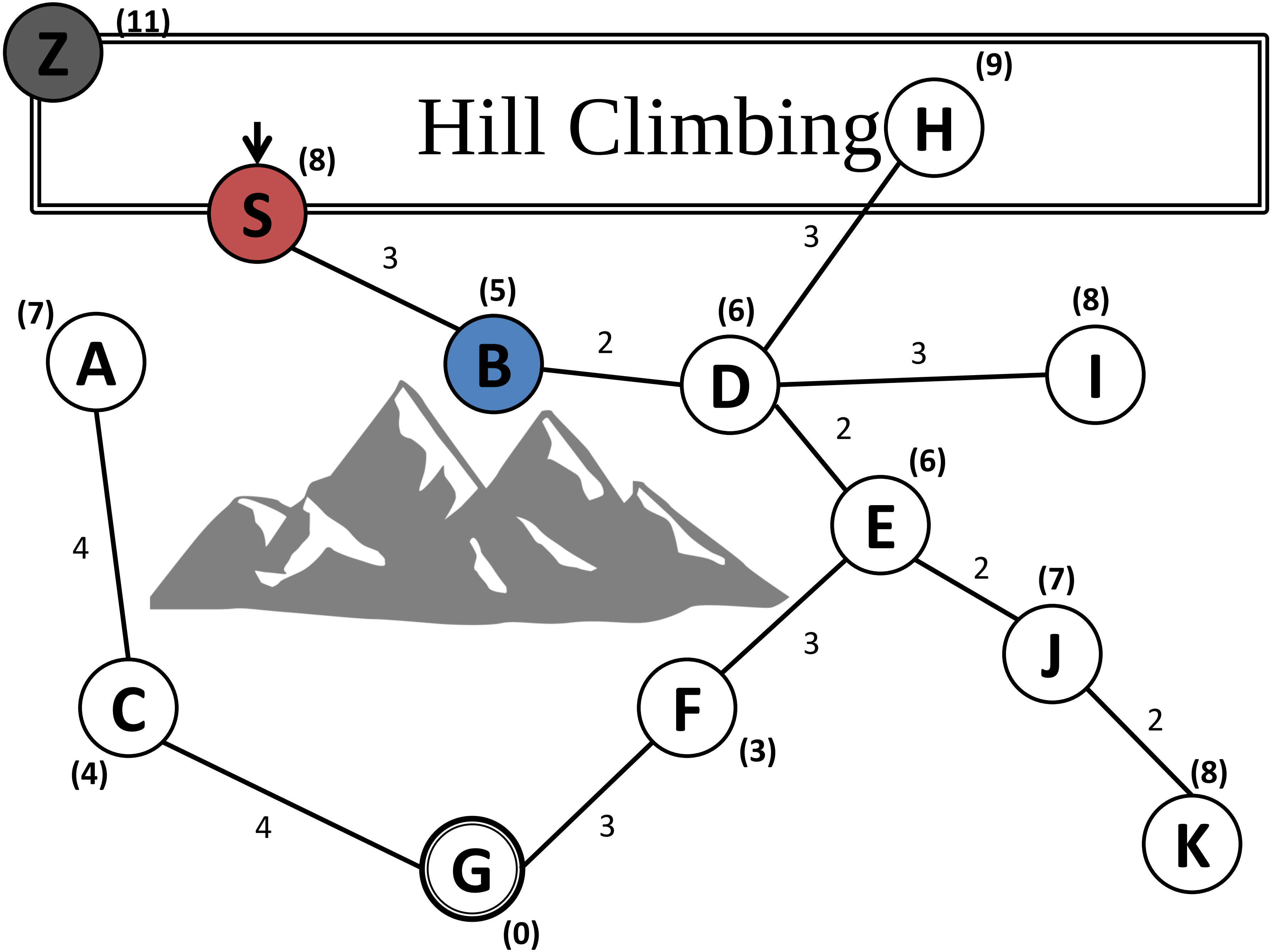
Hill Climbing



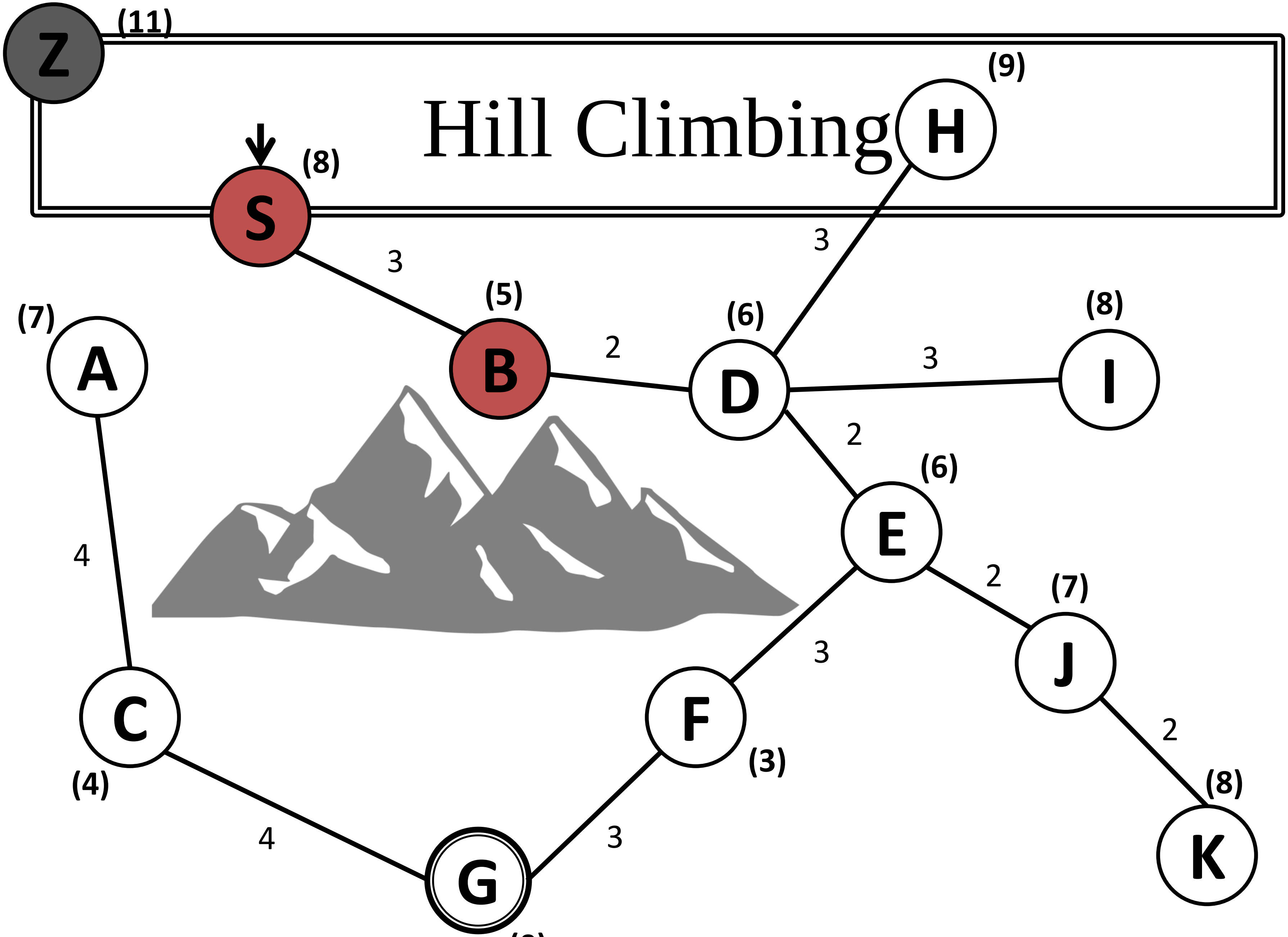
Hill Climbing



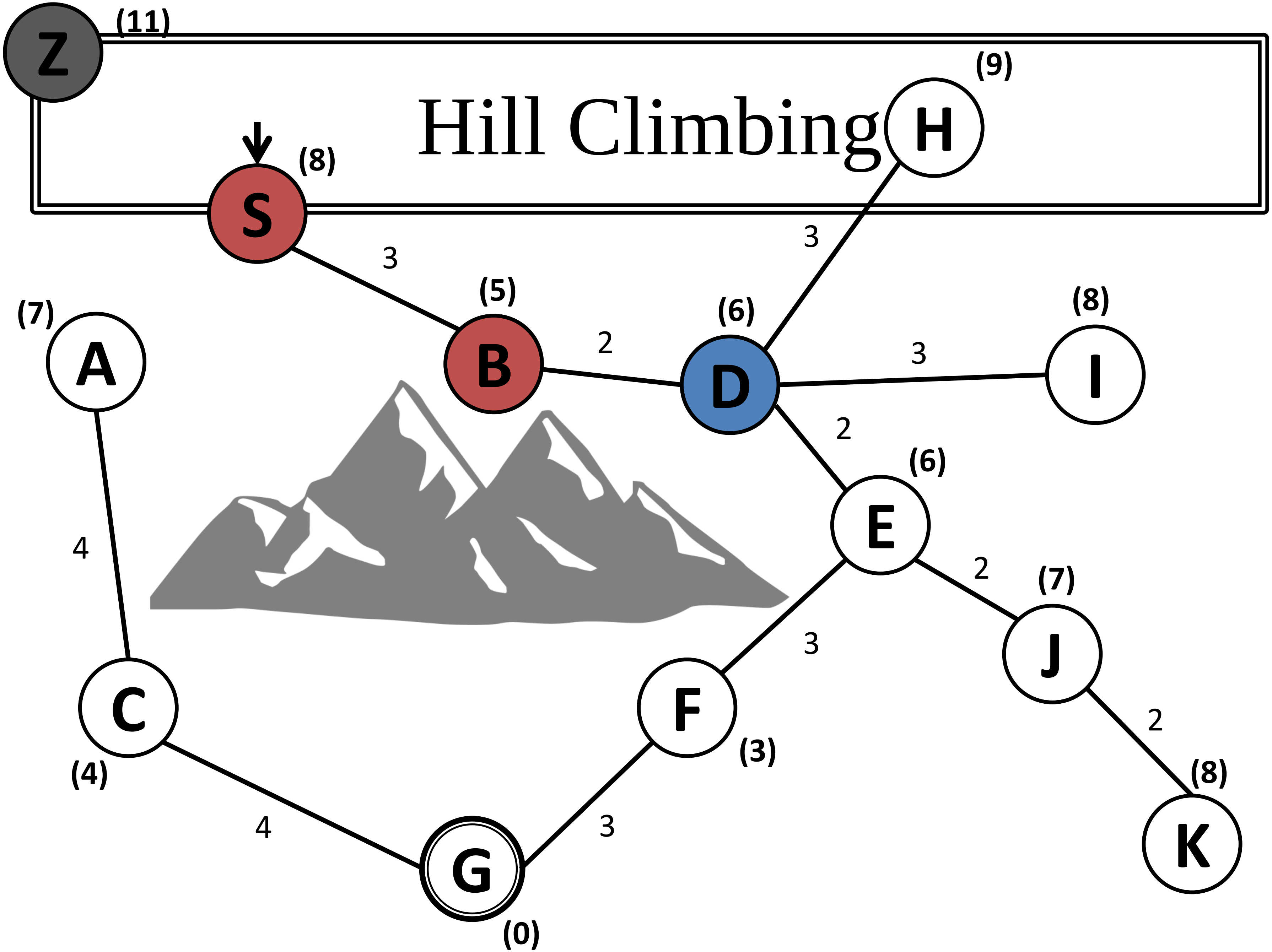
Hill Climbing



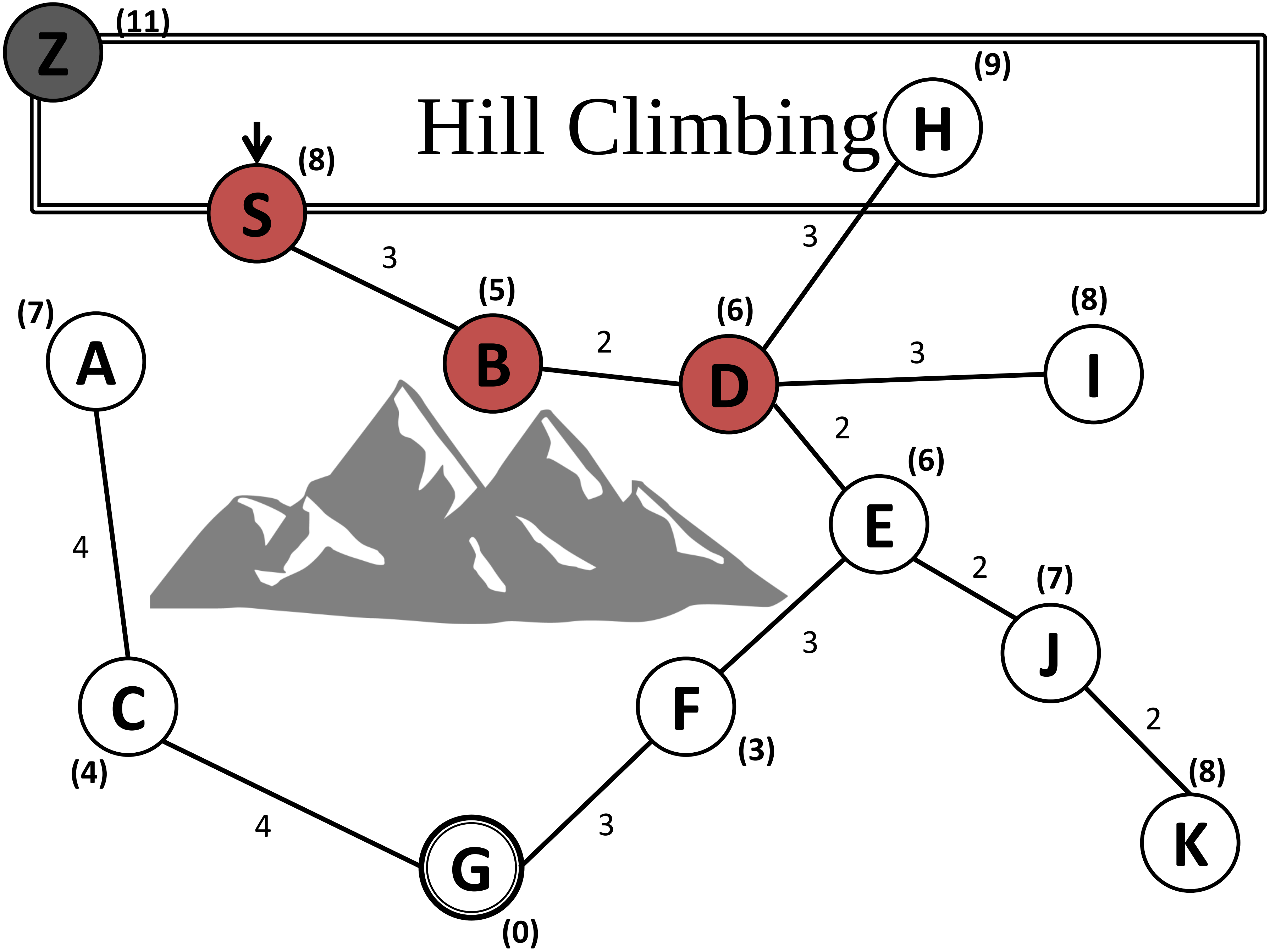
Hill Climbing



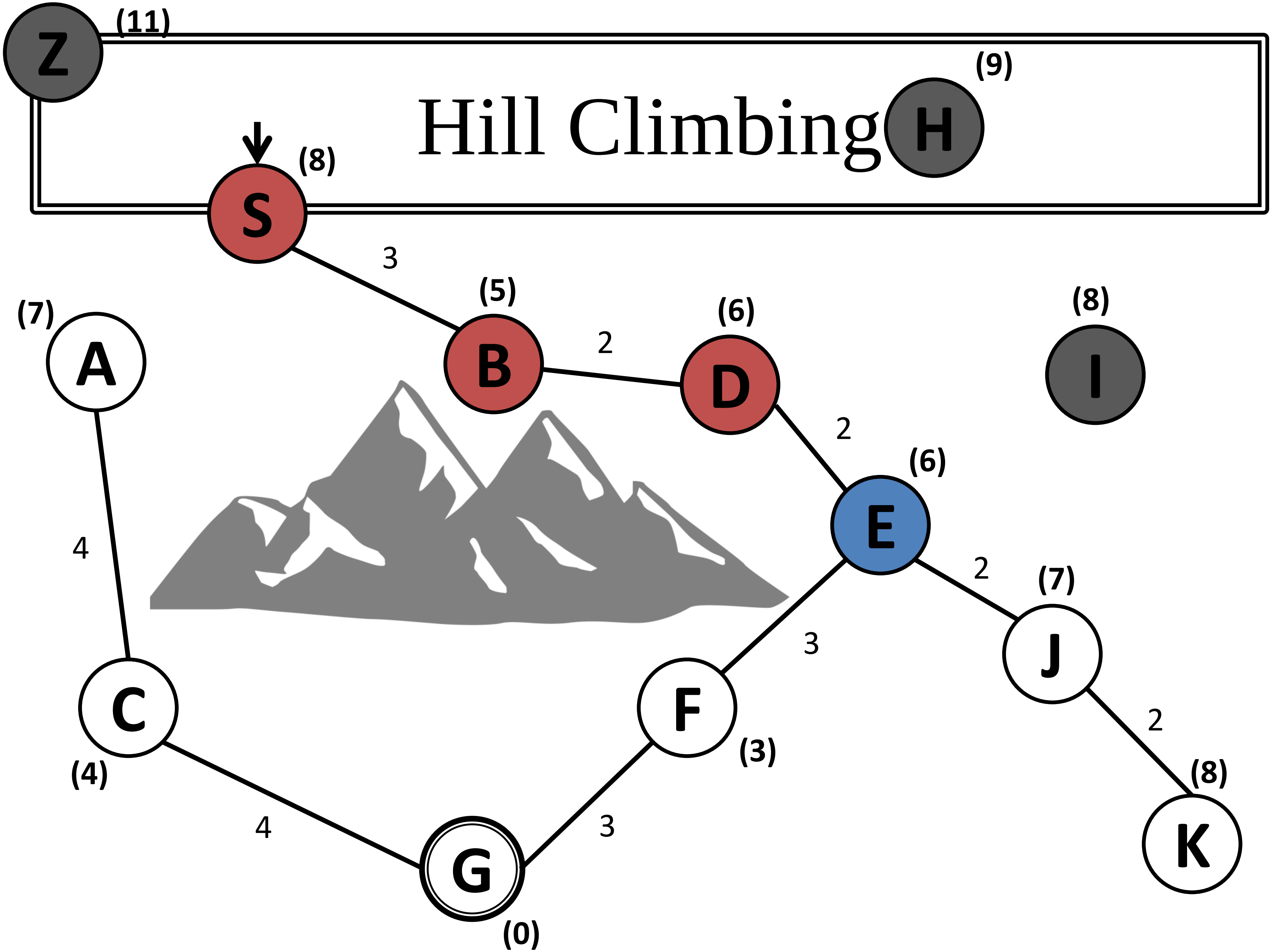
Hill Climbing



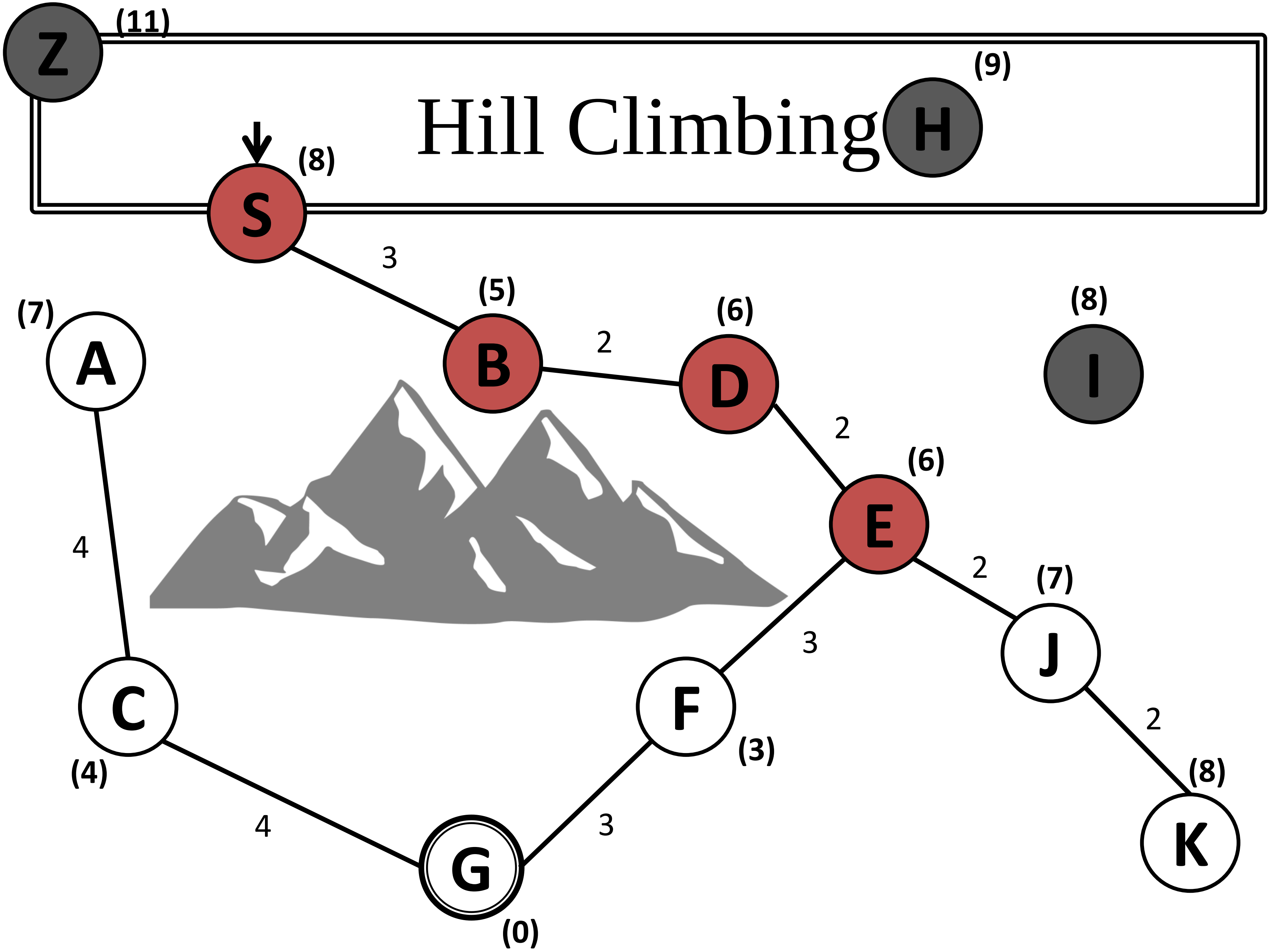
Hill Climbing



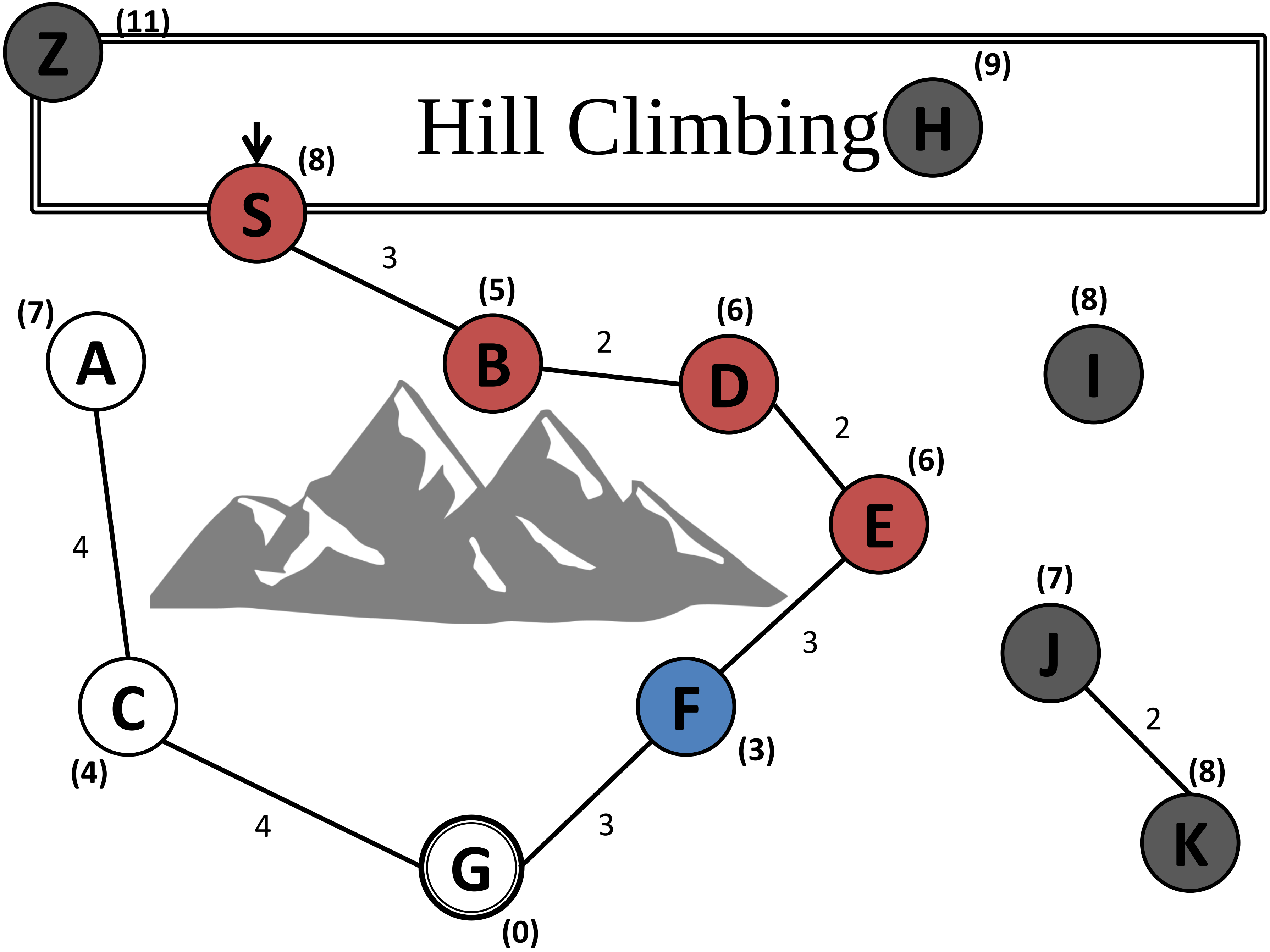
Hill Climbing



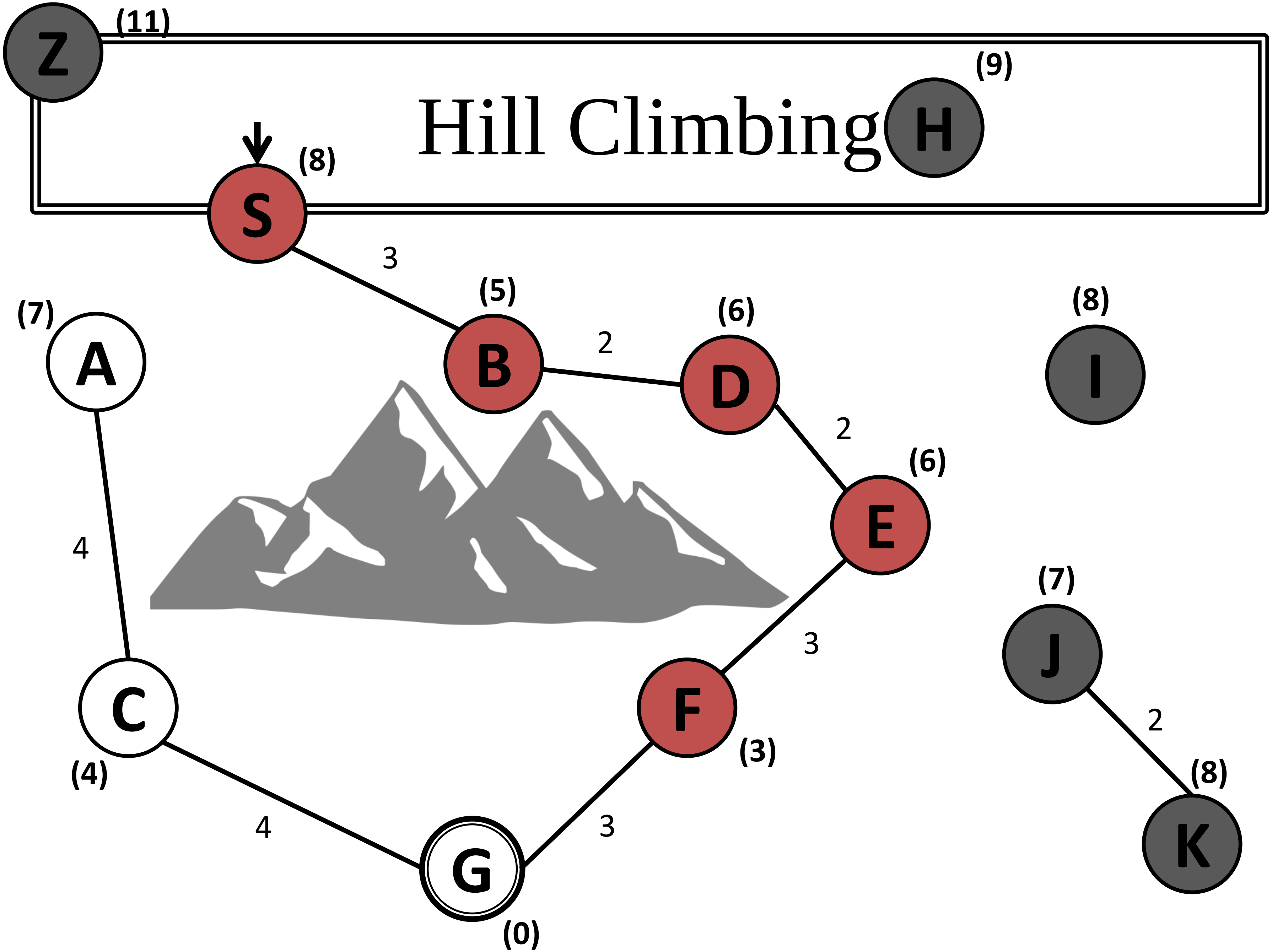
Hill Climbing



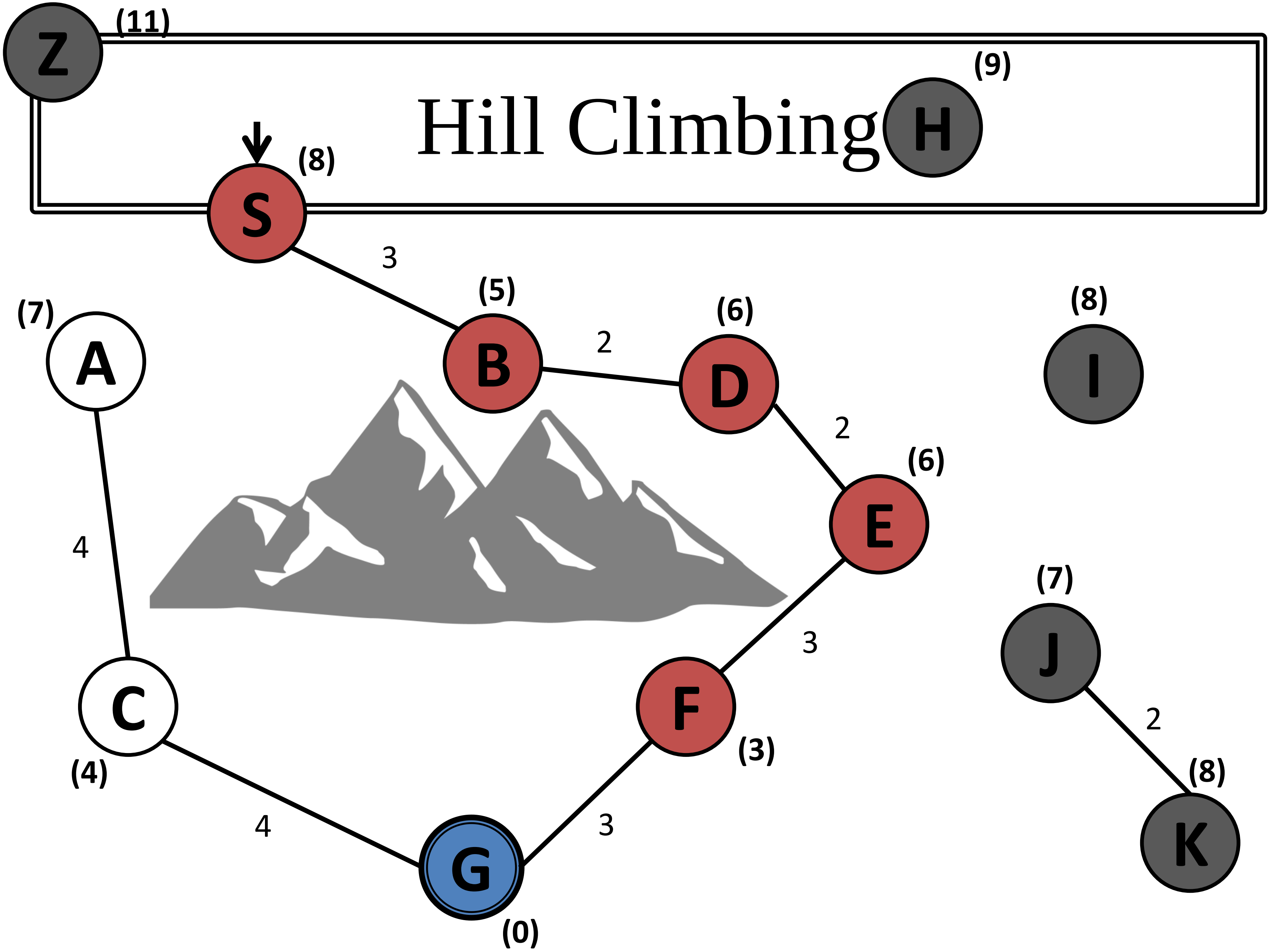
Hill Climbing



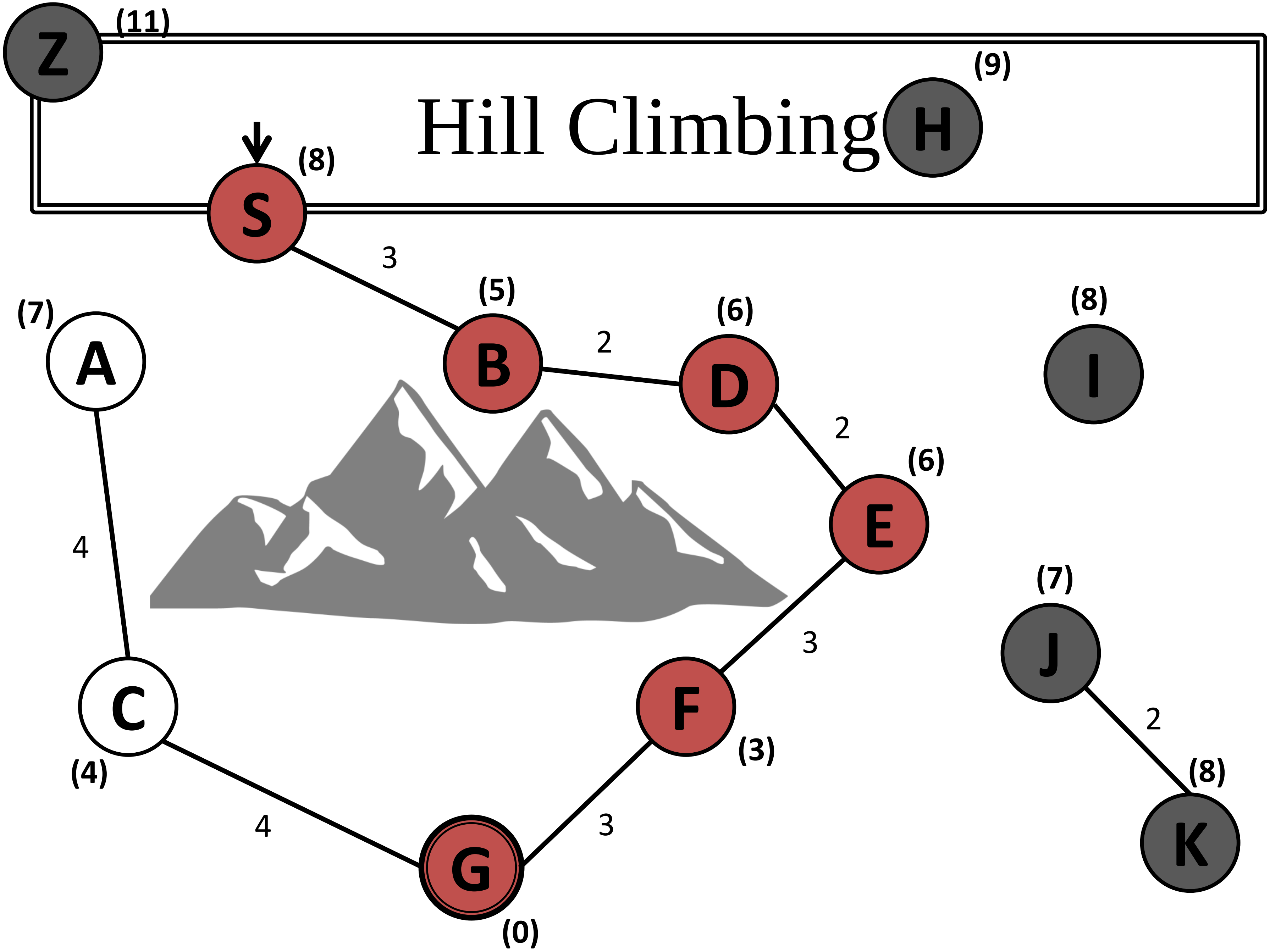
Hill Climbing



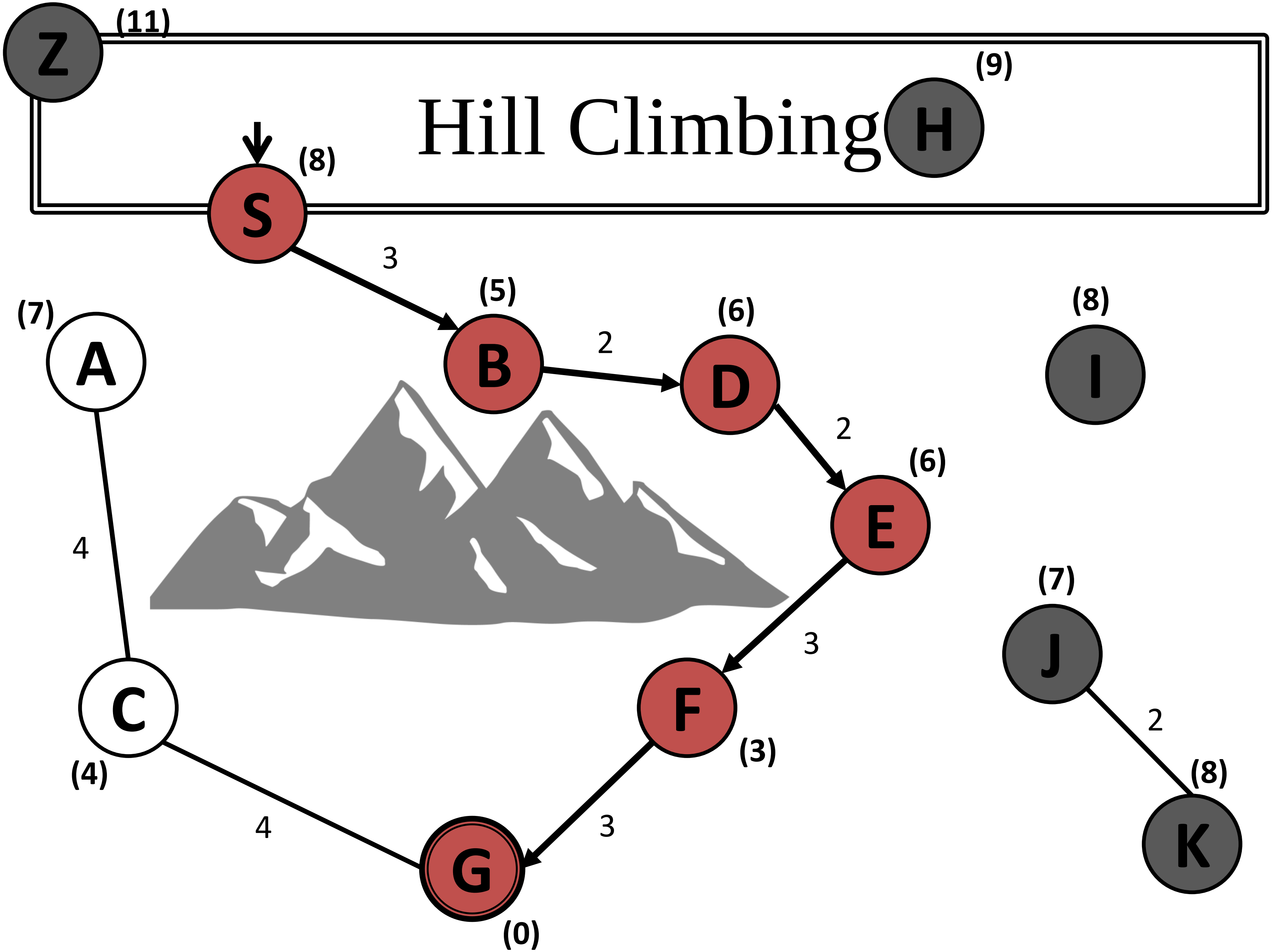
Hill Climbing



Hill Climbing



Hill Climbing



Hill Climbing

- ***Complete?***
  - ***No***, local search
- ***Optimal?***
  - ***No***
- ***Time?***
  - ***O(bm)*** – linear time
- ***Space?***
  - ***O(b)*** – constant

A*

- ***Idea:*** avoid expanding paths that are already expensive
  - Evaluation function ***f(n) = g(n) + h(n)***

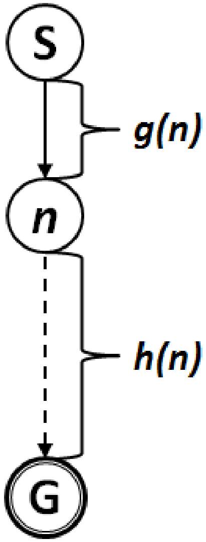
    g(n) = cost so far to reach ***n***

    h(n) = estimated cost to goal from ***n***

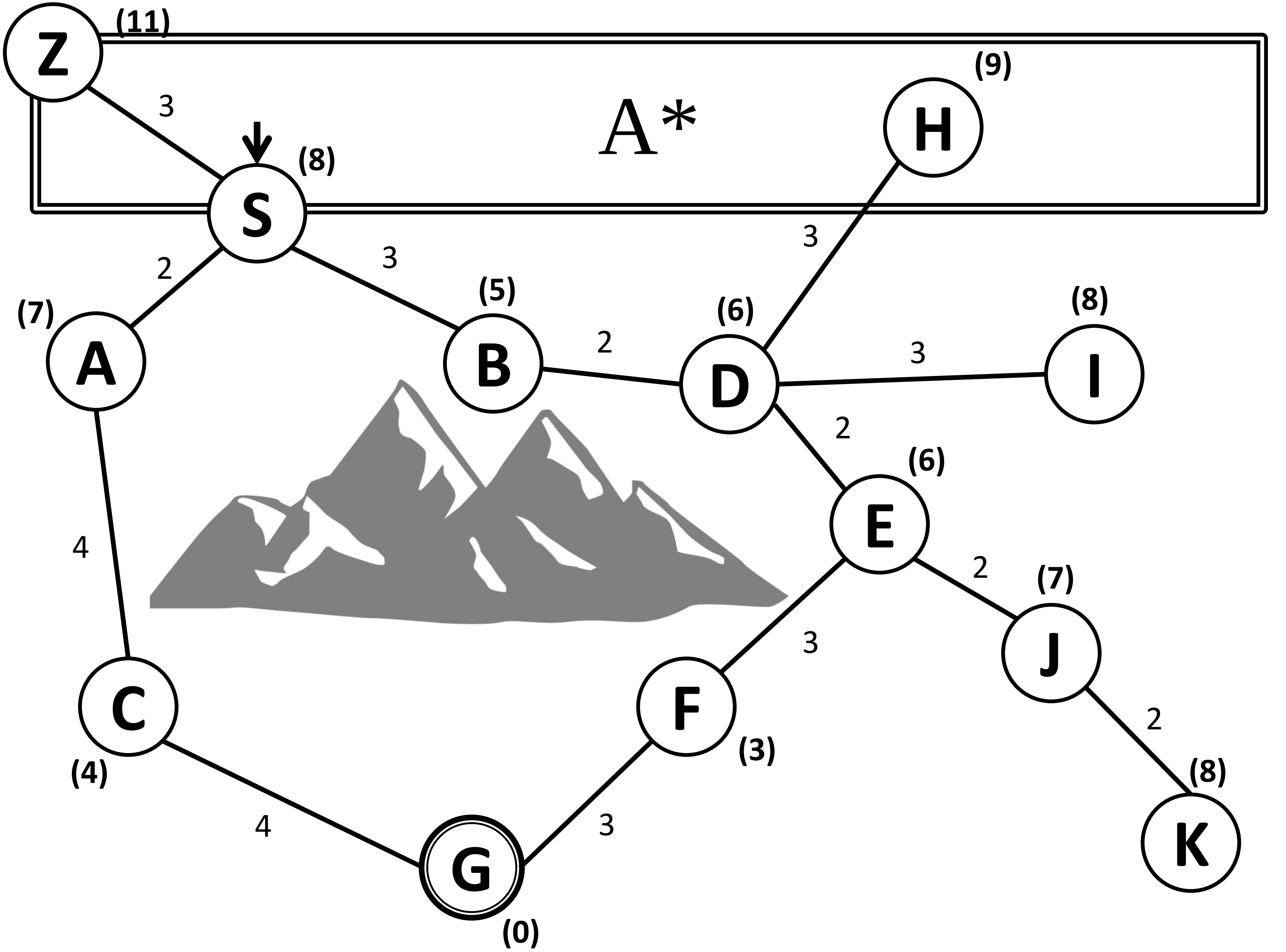
    f(n) = estimated total cost of path through ***n*** to *goal*
  - ***A**** search uses an ***admissible*** *heuristic*

    i.e., ***h(n) ≤ h*(n)*** where ***h*(n)*** is the ***true*** cost from ***n***.

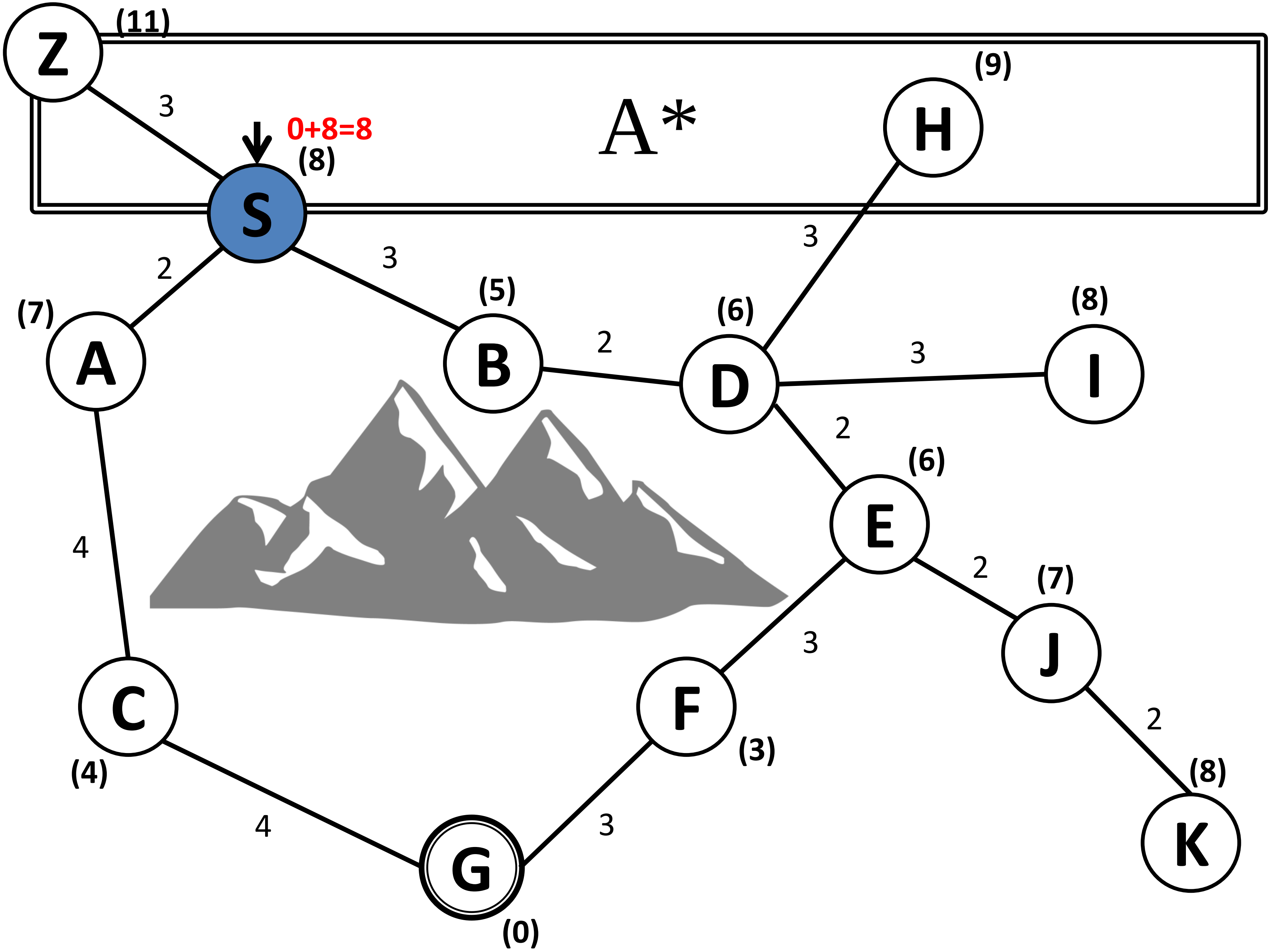
    (Also require ***h(n) ≥ 0***, so ***h(G) = 0*** for any goal ***G***.)
  - E.g., $\boldsymbol{h_{SLD}(n)}$ never overestimates the actual road distance

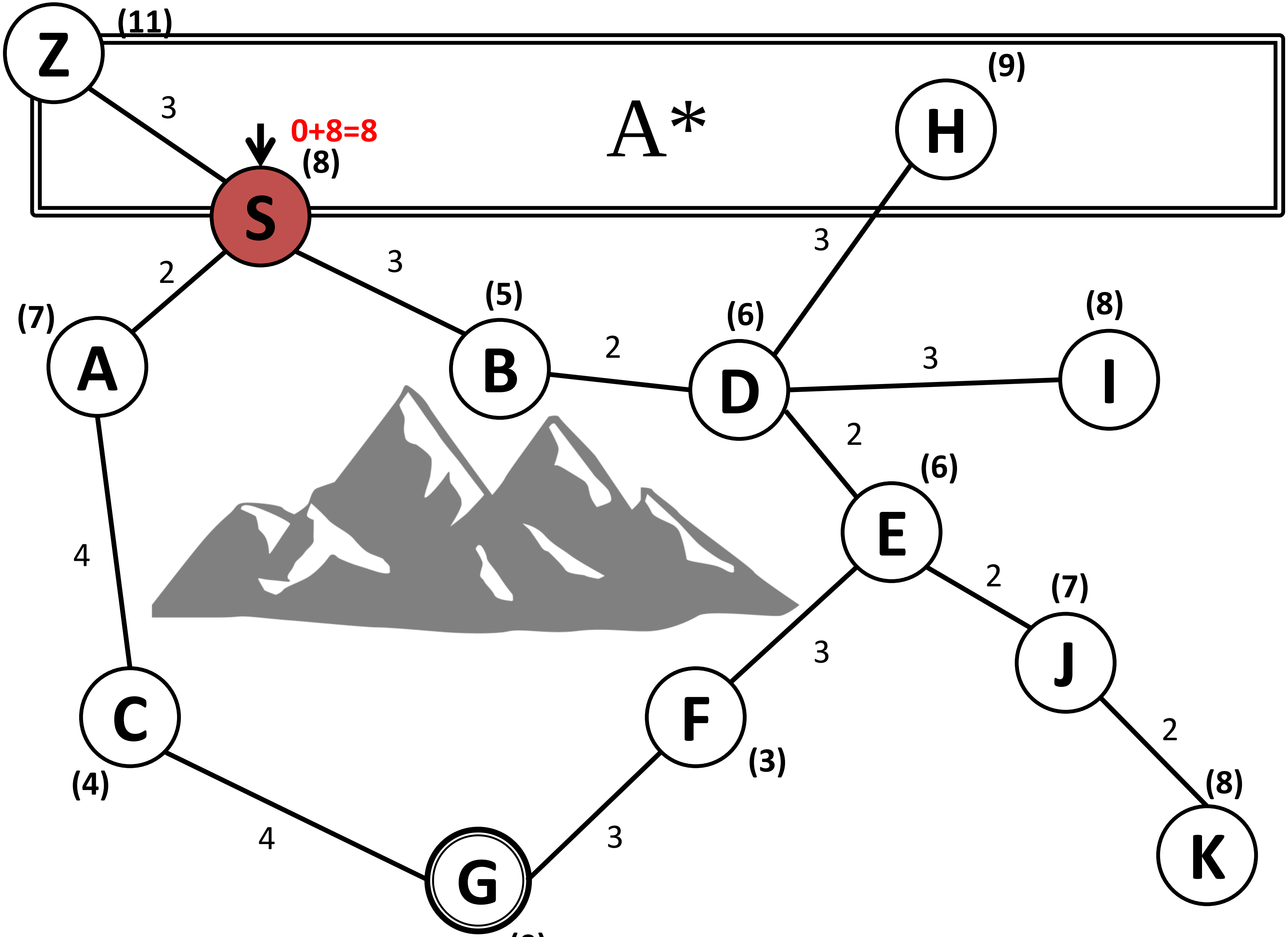


- ***Theorem:*** ***A**** *search is optimal*


A*
Z
(11)
3
S
(8)
H
(9)
3
2
3
A
(7)
B
(5)
2
D
(6)
3
I
(8)
2
E
(6)
4
2
J
(7)
3
C
(4)
F
(3)
2
K
(8)
4
3
G
(0)

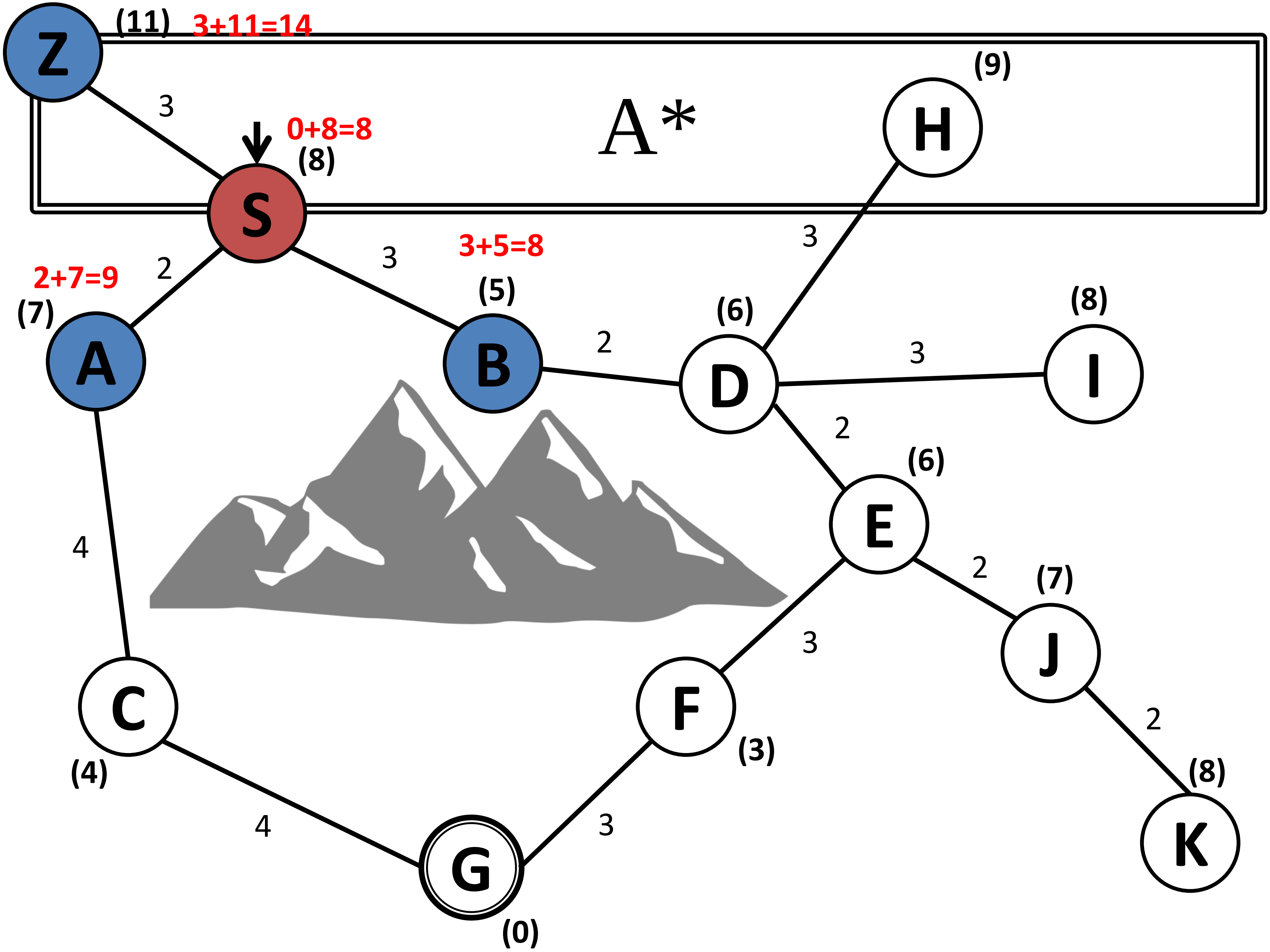
A*

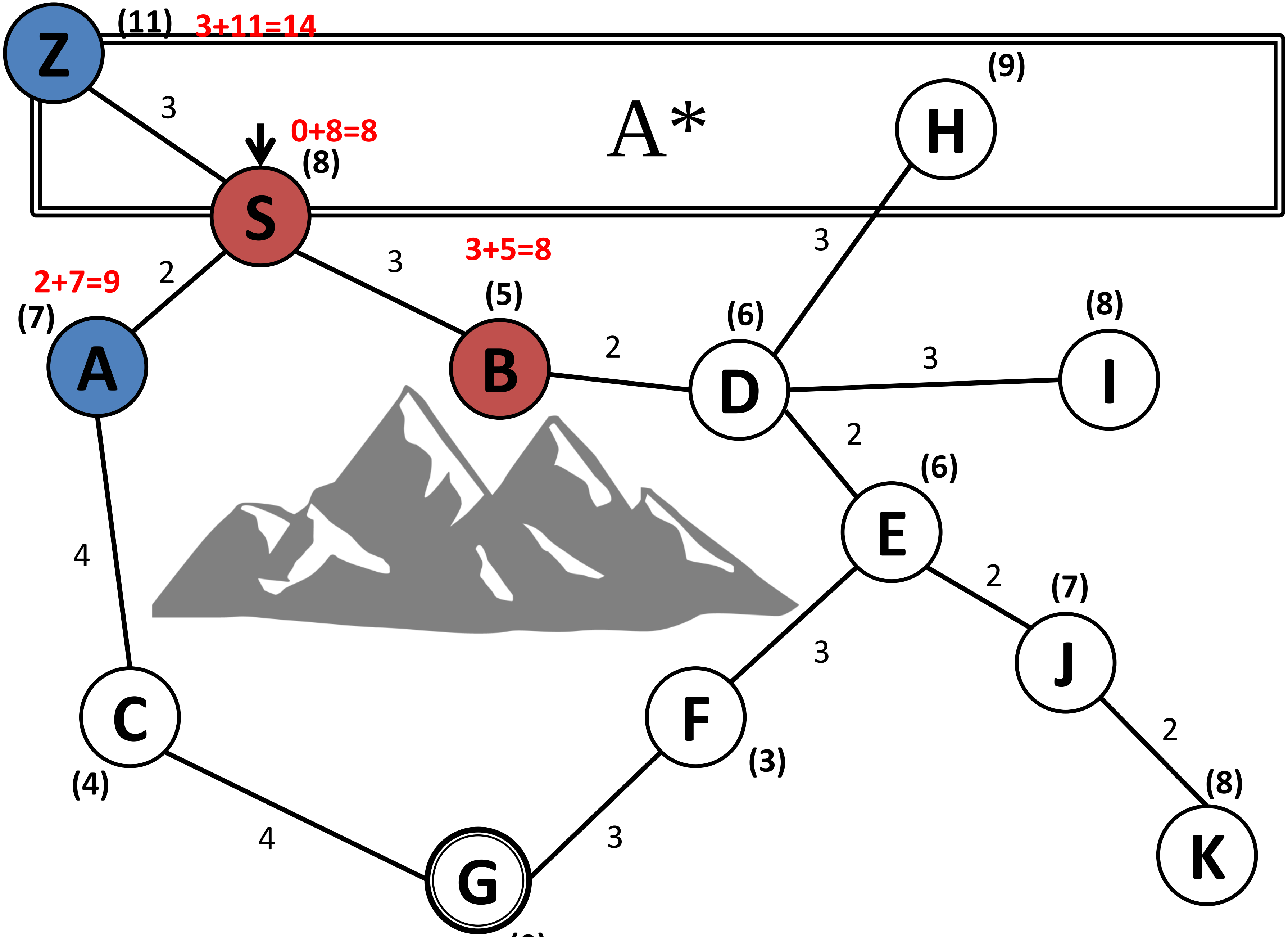


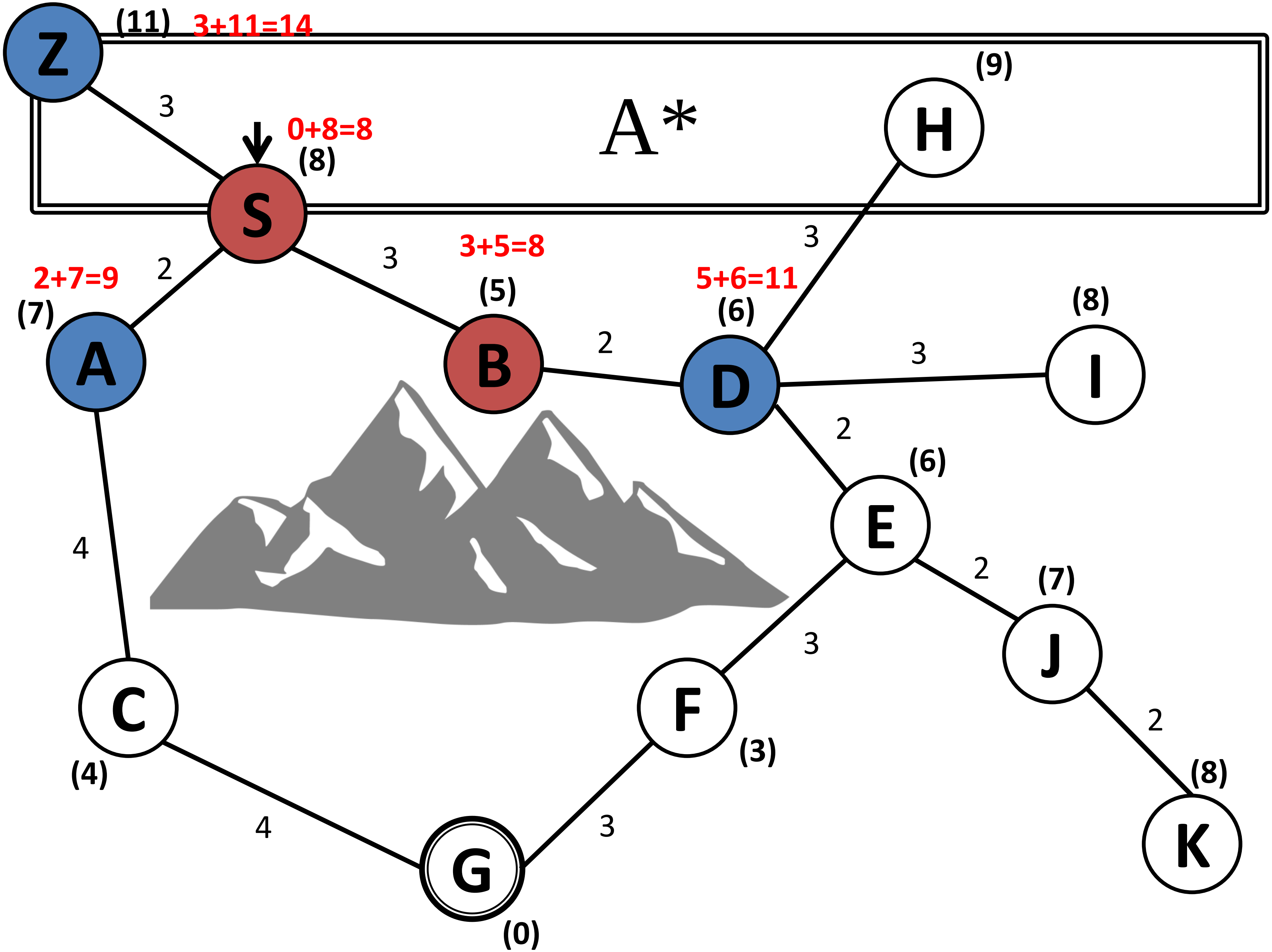


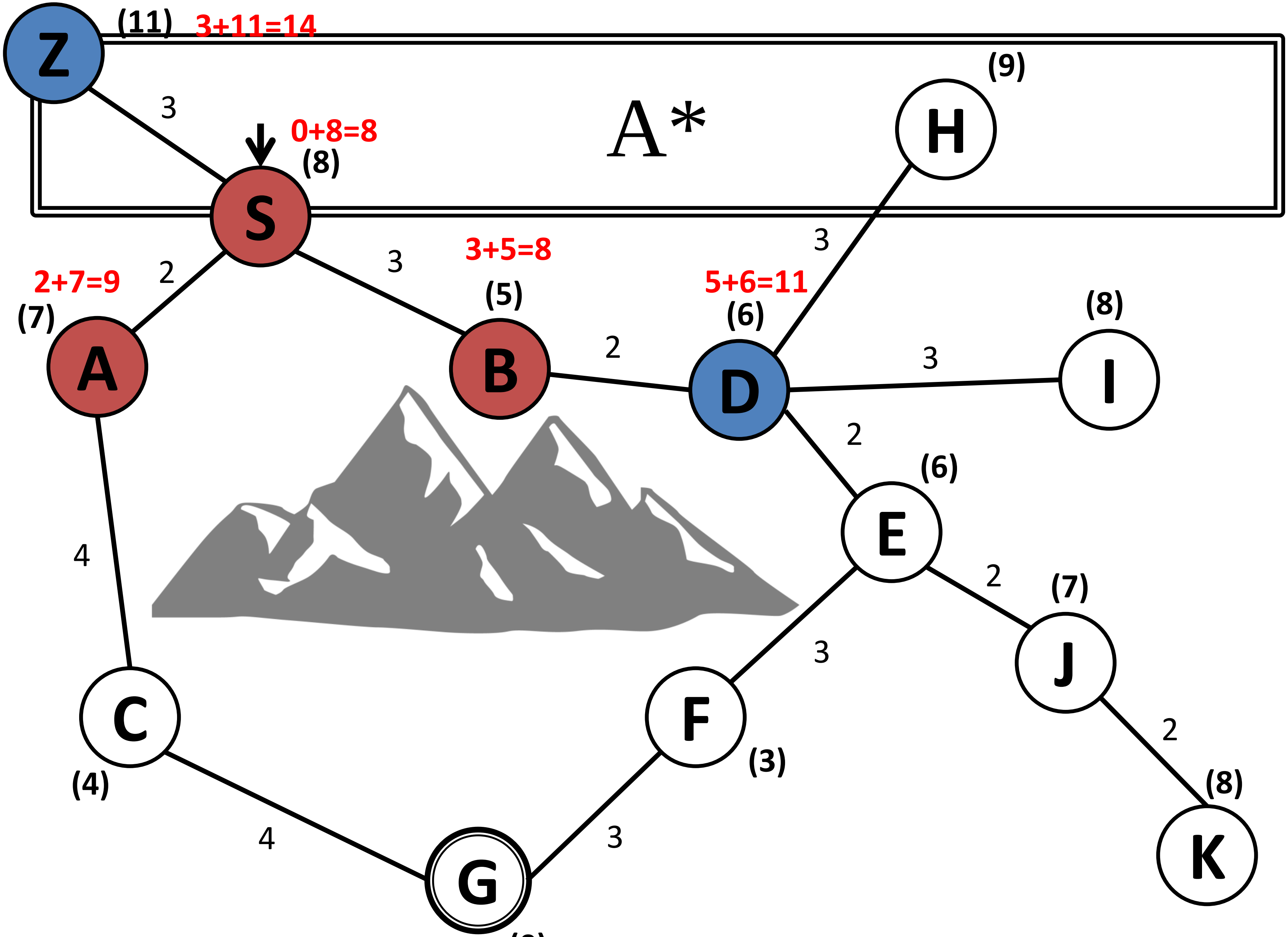
A*

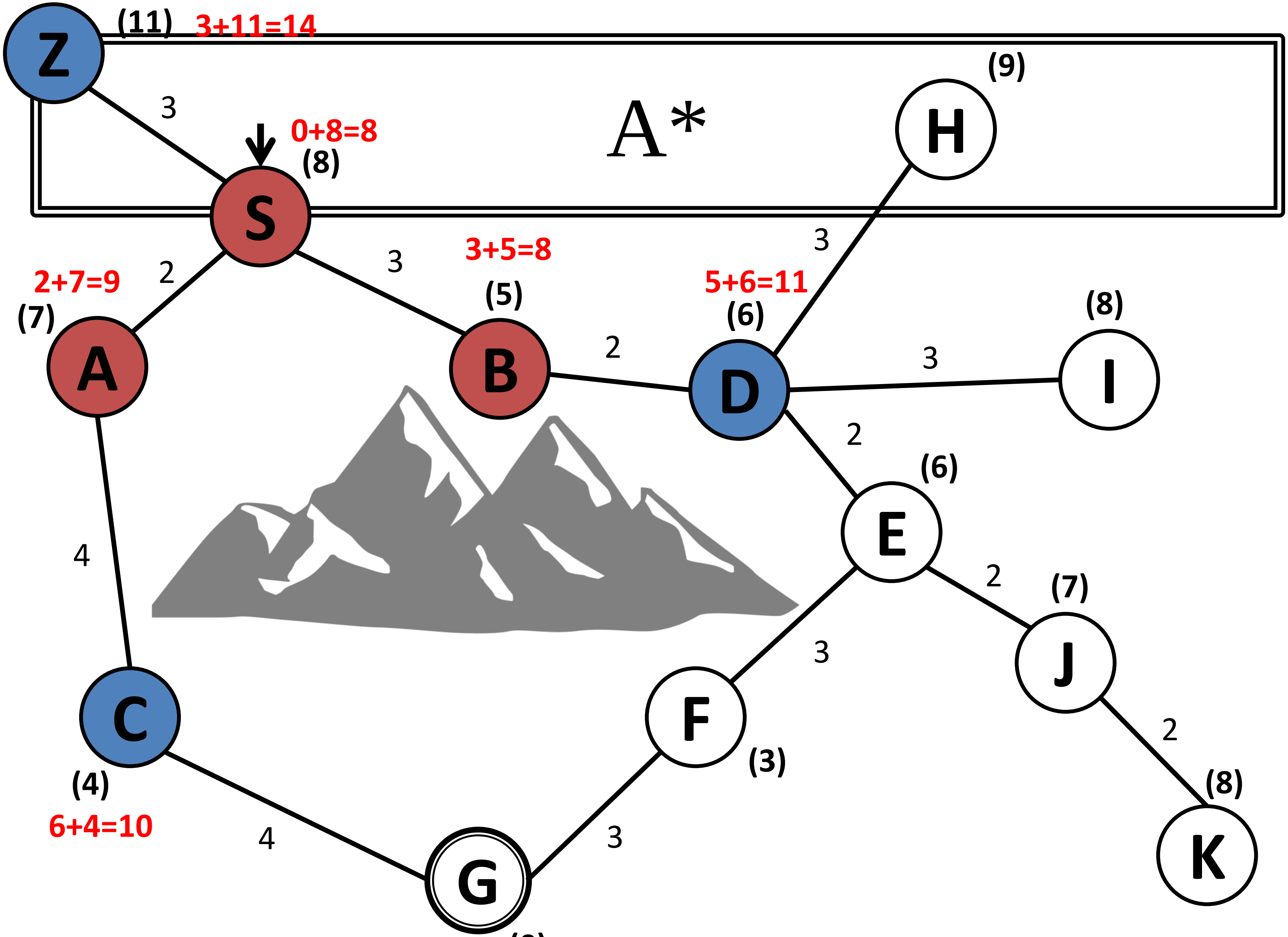
A*

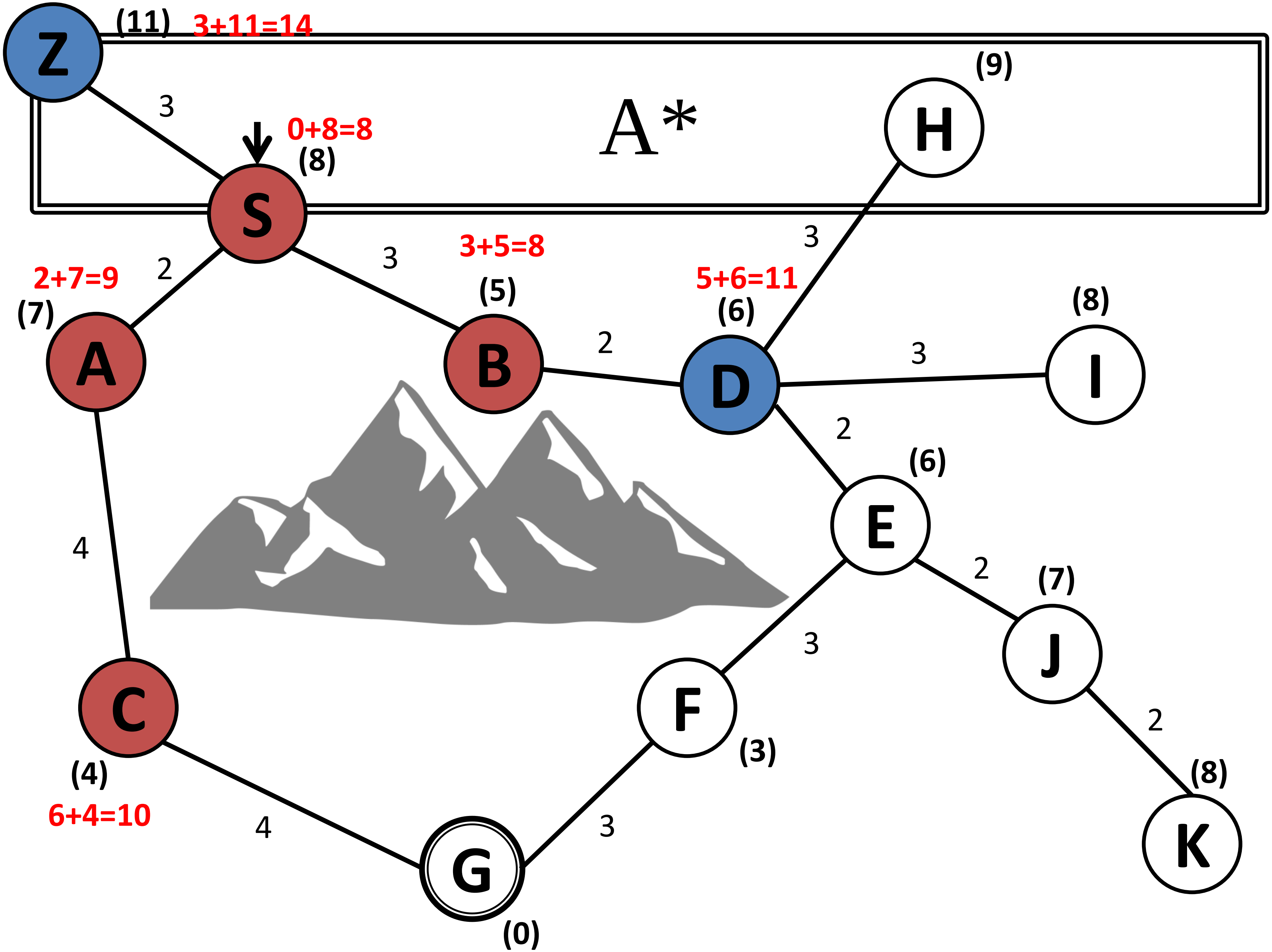

Z
(11)
3+11=14
3
S
0+8=8
(8)
2
3
3+5=8
2+7=9
(7)
A
(5)
B
2
(6)
D
3
H
(9)
3
(8)
I
2
(6)
E
4
2
(7)
J
3
C
(4)
F
(3)
2
(8)
K
4
3
G
(0)


Z
(11)
3+11=14
A*
(9)
H
3
0+8=8
(8)
S
3
2
3
3+5=8
2+7=9
(7)
A
(5)
B
2
(6)
D
3
(8)
I
2
(6)
E
4
2
(7)
J
3
C
(4)
F
(3)
2
(8)
K
4
3
G
(0)

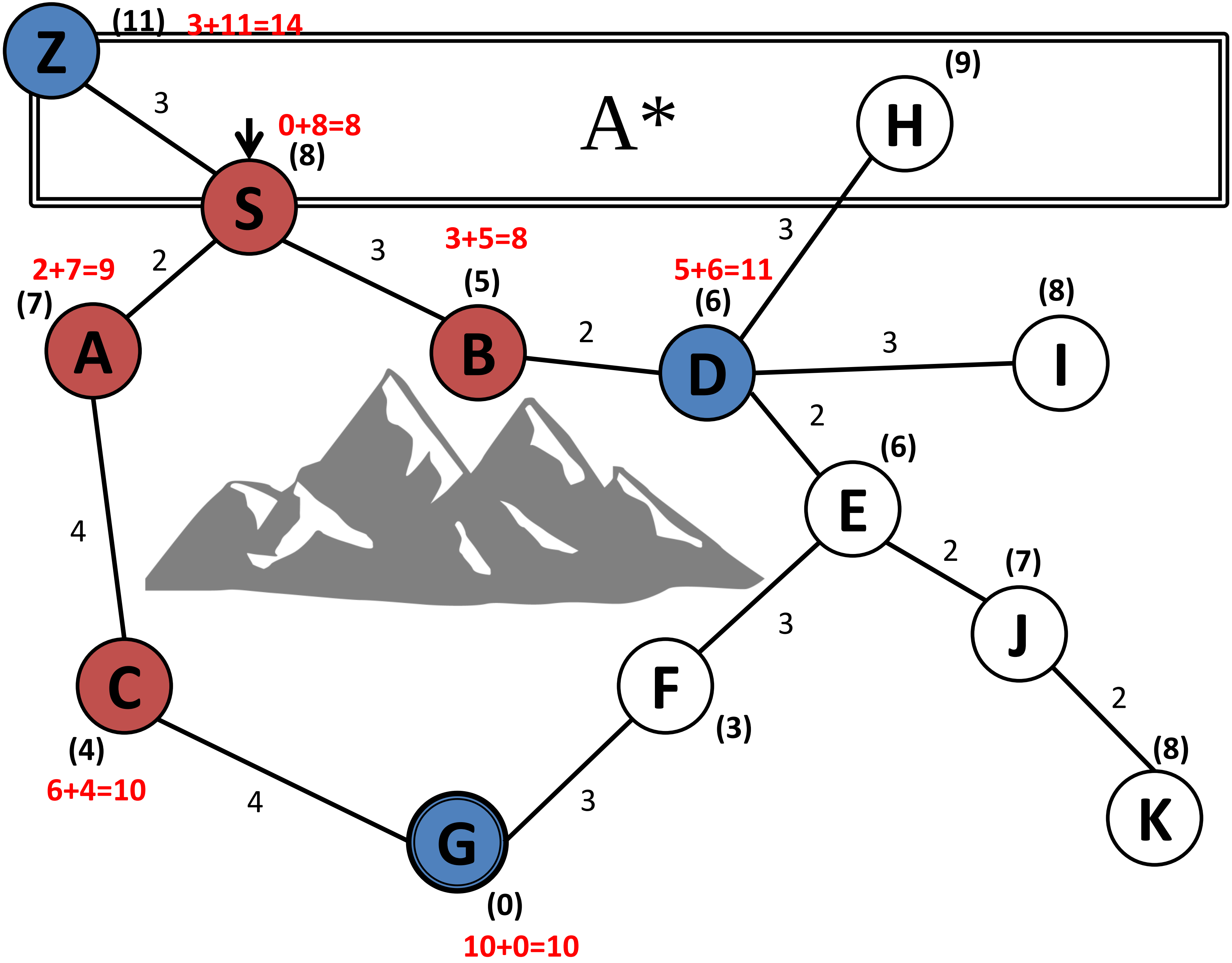

A*
Z
(11)
3+11=14
3
S
0+8=8
(8)
2
3
A
2+7=9
(7)
B
3+5=8
(5)
2
D
5+6=11
(6)
3
H
(9)
3
I
(8)
2
E
(6)
2
J
(7)
2
K
(8)
3
F
(3)
3
G
(0)
4
C
(4)
4

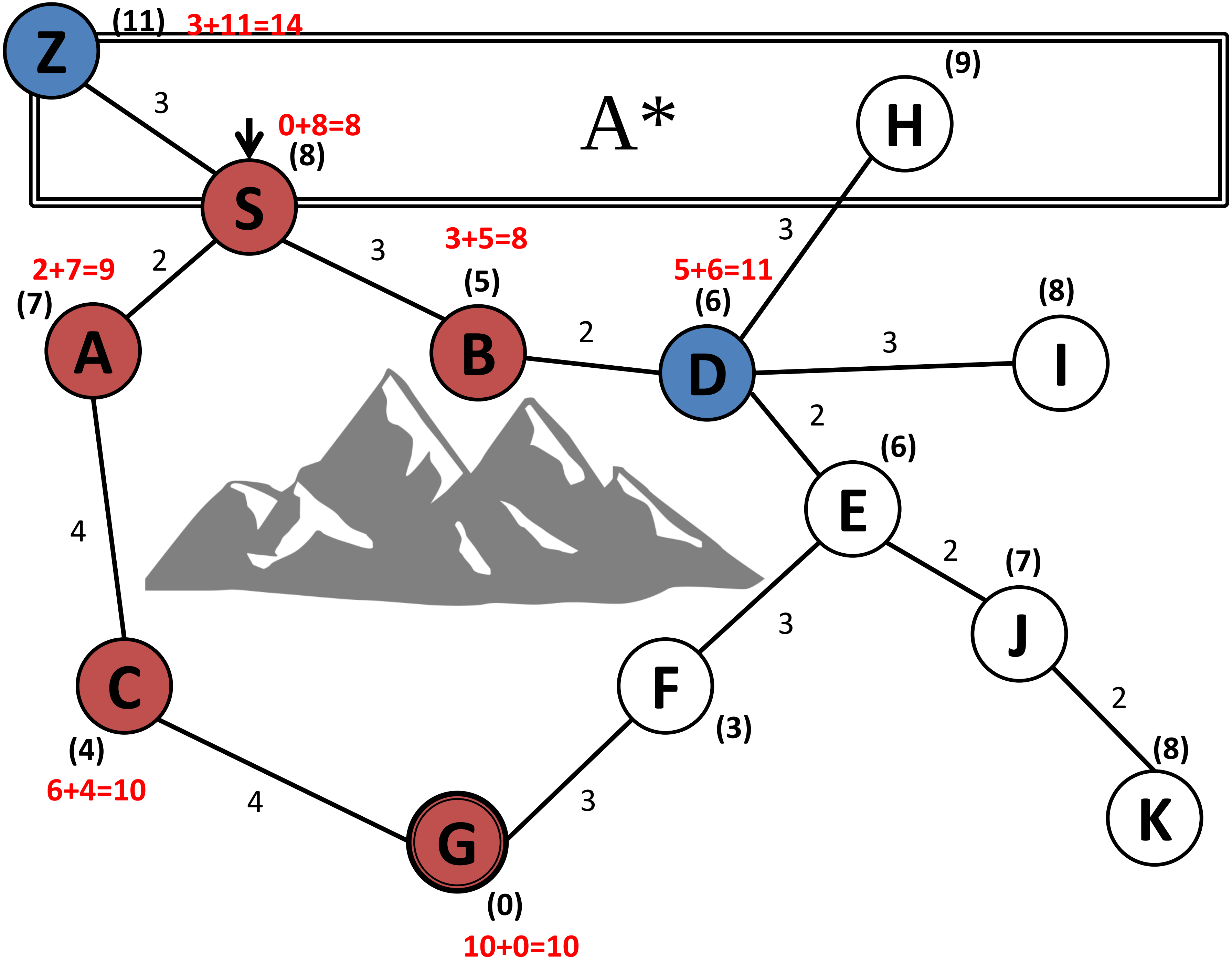

A*
Z
(11)
3+11=14
3
S
0+8=8
(8)
H
(9)
3
2
A
2+7=9
(7)
B
3+5=8
(5)
2
D
5+6=11
(6)
3
I
(8)
2
E
(6)
2
J
(7)
2
K
(8)
4
C
(4)
4
G
(0)
3
F
(3)
3

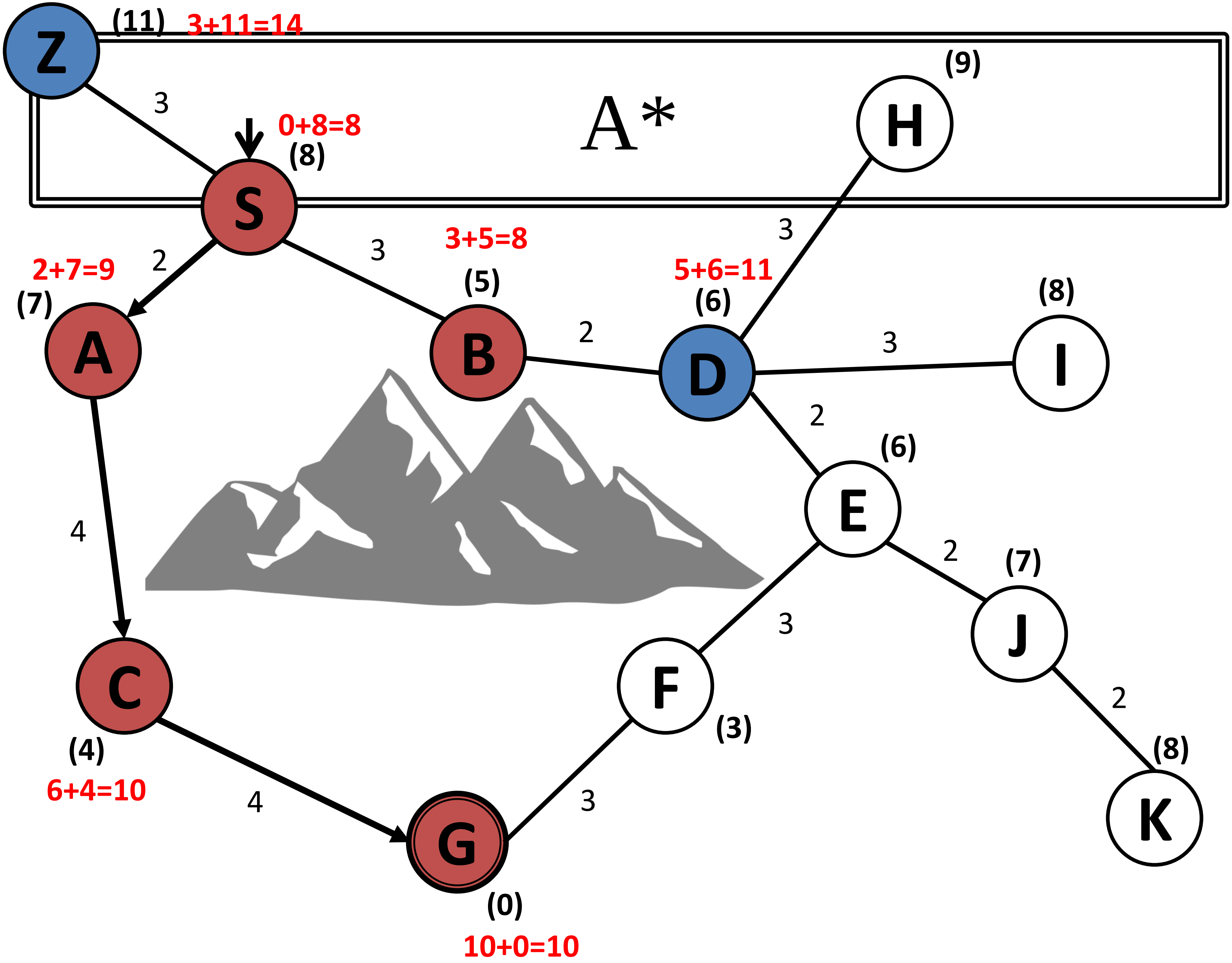

A*
Z
(11)
3+11=14
3
S
0+8=8
(8)
2
3
A
2+7=9
(7)
B
3+5=8
(5)
2
D
5+6=11
(6)
H
(9)
3
3
I
(8)
2
E
(6)
2
J
(7)
2
K
(8)
3
F
(3)
3
4
C
(4)
6+4=10
4
G
(0)



A*
Z
(11)
3+11=14
3
S
0+8=8
(8)
H
(9)
3
2+7=9
(7)
A
2
3
3+5=8
(5)
B
2
5+6=11
(6)
D
3
I
(8)
2
E
(6)
4
2
J
(7)
3
F
(3)
2
K
(8)
C
(4)
6+4=10
4
G
(0)
3


Z
(11)
3+11=14
3
0+8=8
(8)
S
A*
(9)
H
3
3+5=8
2+7=9
2
3
5+6=11
(7)
(5)
(6)
(8)
A
B
2
D
3
I
2
(6)
E
4
2
(7)
3
J
F
(3)
2
(4)
C
(8)
6+4=10
4
3
K
G
(0)
10+0=10


Z
(11)
3+11=14
A*
(9)
H
3
0+8=8
(8)
S
3
3+5=8
2
2+7=9
(7)
A
(5)
B
2
5+6=11
(6)
D
3
(8)
I
3
2
(6)
E
4
2
(7)
J
3
C
F
(3)
2
(4)
6+4=10
4
3
(8)
K
G
(0)
10+0=10


A*
Z
(11)
3+11=14
3
S
(8)
0+8=8
H
(9)
3
2
A
(7)
2+7=9
3
B
(5)
3+5=8
2
D
(6)
5+6=11
3
I
(8)
2
E
(6)
4
2
J
(7)
3
F
(3)
2
K
(8)
C
(4)
6+4=10
4
G
(0)
10+0=10
3

A*

- ***Complete?***
  - ***Yes***, unless there are infinitely many nodes with $\boldsymbol{f \leq f(G)}$
- ***Optimal?***
  - ***Yes***, cannot expand $\boldsymbol{f_{i+1}}$ until $\boldsymbol{f_i}$ is finished

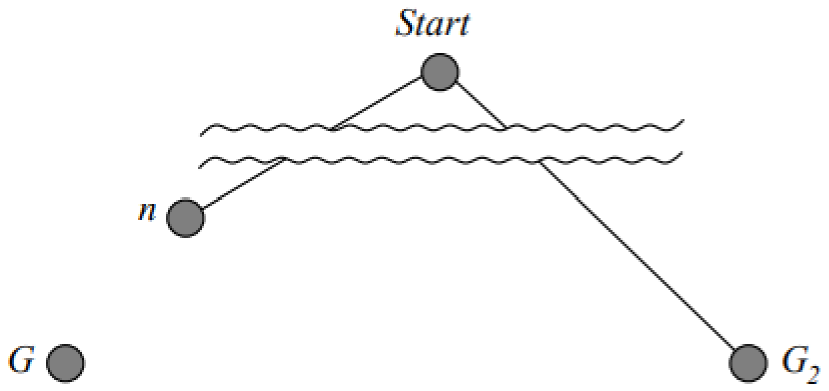
  A* *expands all nodes with* ***f(n) < C****

  A* *expands some nodes with* ***f(n) = C****

  A* *expands no nodes with* ***f(n) > C****
- ***Time?***
  - $\boldsymbol{O(b^d)}$: Exponential in *[relative error in* ***h*** *× length of soln.]*
- ***Space?***
  - $\boldsymbol{O(b^d)}$: Keeps all nodes in memory

Optimality of A* (standard proof)

Suppose some suboptimal goal $\boldsymbol{G_2}$ has been generated and is in the queue.
Let ***n*** be an unexpanded node on a shortest path to an optimal goal $\boldsymbol{G_1}$.



$$
\begin{aligned}
\boldsymbol{f(G_2)} &\boldsymbol{= g(G_2)} && \textit{since } \boldsymbol{h(G_2) = 0} \\
&\boldsymbol{> g(G_1)} && \textit{since } \boldsymbol{G_2} \textit{ is suboptimal} \\
&\boldsymbol{\geq f(n)} && \textit{since } \boldsymbol{h} \textit{ is admissible}
\end{aligned}
$$

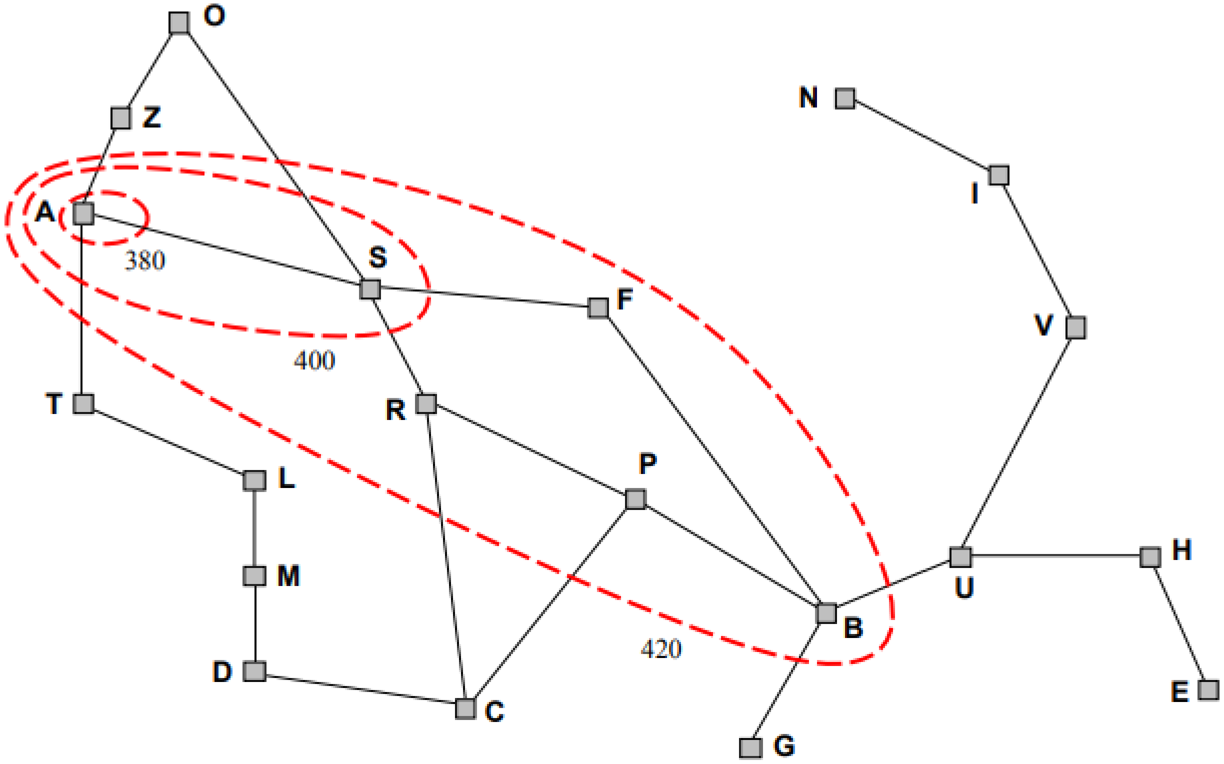
Since $\boldsymbol{f(G_2) > f(n)}$*, A* will never select* $\boldsymbol{G_2}$ *for expansion*

Optimality of A* (more useful)

- ***Lemma:*** ***A**** expands nodes in order of increasing ***f*** value*

  Gradually adds "***f***-contours" of nodes (cf. breadth-first adds layers)

  Contour ***i*** has all nodes with $f = f_i$, where $f_i < f_{i+1}$



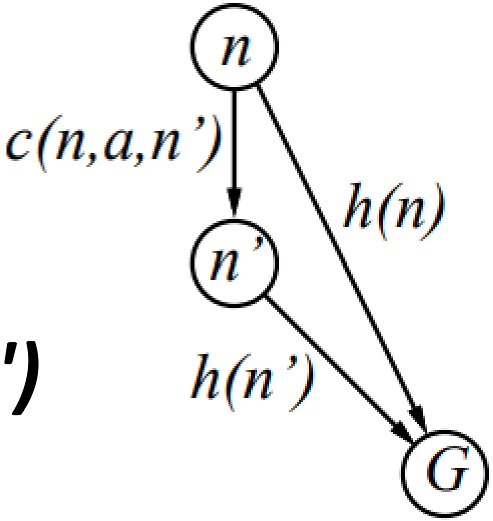
Proof of Lemma: Consistency

- A heuristic is consistent if

$$h(n) \le c(n, a, n') + h(n')$$

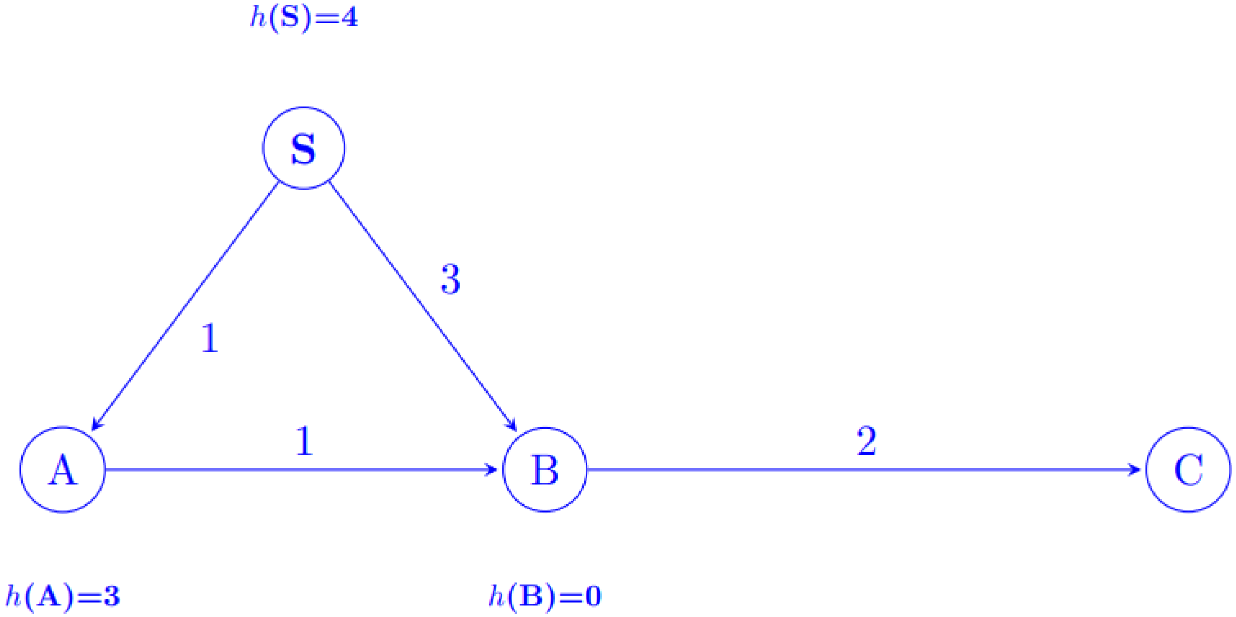
- If ***h*** is consistent, we have

$$\begin{aligned} f(n') &= g(n') + h(n') \\ &= g(n) + c(n, a, n') + h(n') \\ &\ge g(n) + h(n) \\ &= f(n) \end{aligned}$$



I.e., ***f(n)*** is nondecreasing along any path.

Admissible & Consistent Heuristic


$h(\mathbf{S})=\mathbf{4}$
S
3
1
A
1
B
2
C
$h(\mathbf{A})=\mathbf{3}$
$h(\mathbf{B})=\mathbf{0}$

There is a distinction between *tree search* and *graph search*: A∗ with ***tree search*** is guaranteed to be optimal if the heuristic is admissible, but A∗ with ***graph search*** is only guaranteed to be optimal if the heuristic is consistent.

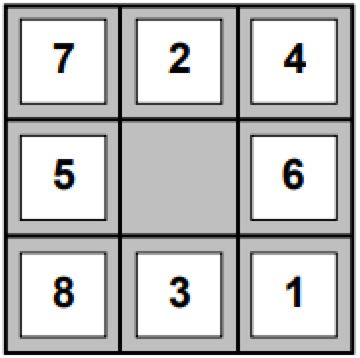
Admissible Heuristics

- E.g., for the *8-puzzle*:

  $h_1(n)$ = number of misplaced tiles

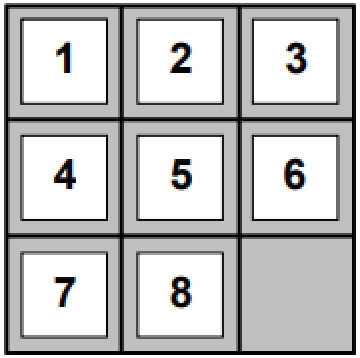
  $h_2(n)$ = total ***Manhattan*** distance

  (i.e., no. of squares from desired location of each tile)



7	2	4
5		6
8	3	1

Start State



1	2	3
4	5	6
7	8	

Goal State

- ***$h_1(S)$ = ?***
- ***$h_2(S)$ = ?***

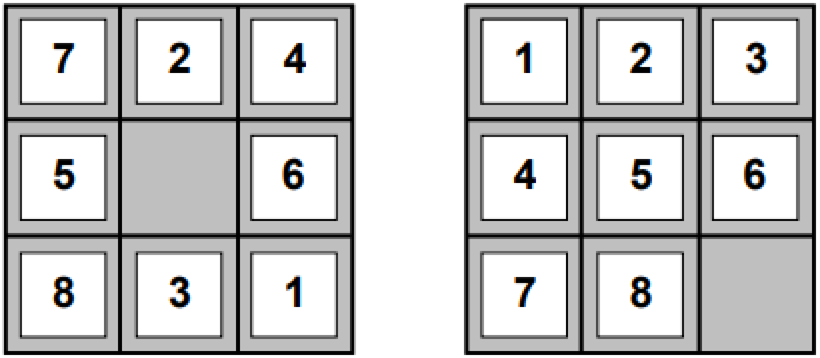
Admissible Heuristics

- E.g., for the *8-puzzle*:

  $h_1(n)$ = number of misplaced tiles

  $h_2(n)$ = total ***Manhattan*** distance

  (i.e., no. of squares from desired location of each tile)



7	2	4
5		6
8	3	1

Start State

1	2	3
4	5	6
7	8	

Goal State

- ***$h_1(S)$ = ?*** 1+0+1+1+1+0+1+1 = 6
- ***$h_2(S)$ = ?***

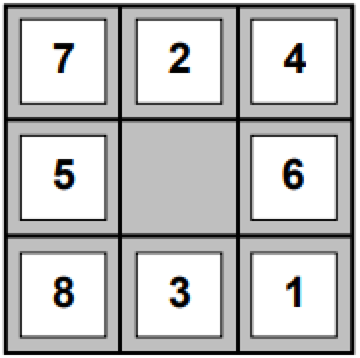
Admissible Heuristics

- E.g., for the *8-puzzle*:

  $h_1(n)$ = number of misplaced tiles

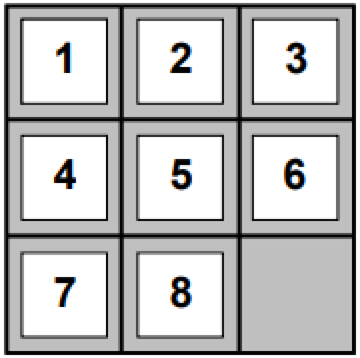
  $h_2(n)$ = total ***Manhattan*** distance

  (i.e., no. of squares from desired location of each tile)



7	2	4
5		6
8	3	1

Start State



1	2	3
4	5	6
7	8	

Goal State

- ***$h_1(S) = ?$*** $1+0+1+1+1+0+1+1 = 6$
- ***$h_2(S) = ?$*** $4+0+3+3+1+0+2+1 = 14$

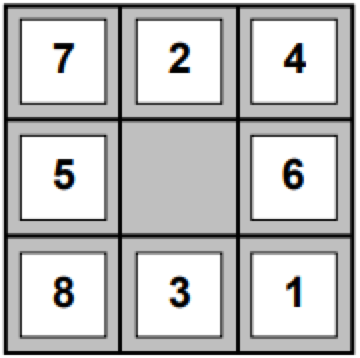
Admissible Heuristics

- E.g., for the *8-puzzle*:

  $h_1(n)$ = number of misplaced tiles

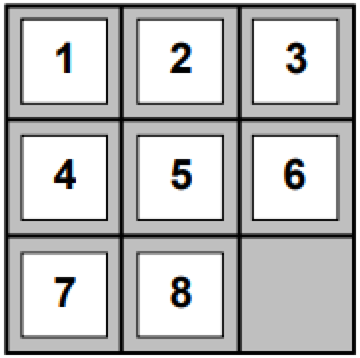
  $h_2(n)$ = total ***Manhattan*** distance

  (i.e., no. of squares from desired location of each tile)



7	2	4
5		6
8	3	1

Start State



1	2	3
4	5	6
7	8	

Goal State

- ***$h_1(S) = ?$*** $1+0+1+1+1+0+1+1 = 6$
- ***$h_2(S) = ?$*** $4+0+3+3+1+0+2+1 = 14$

* *cellDist = |x - x'| + |y - y'|*

Manhattan Distance

X' Y'	0	1	2
0	**7**	**2**	**4**
1	**5**		**6**
2	**8**	**3**	**1**

Start State

X Y	0	1	2
0	**1**	**2**	**3**
1	**4**	**5**	**6**
2	**7**	**8**	

Goal State

$$\mathit{cellDist} = |x - x'| + |y - y'| = |0 - 2| + |0 - 2| = 2 + 2 = 4$$

Dominance

- If $\boldsymbol{h_2(n) \geq h_1(n)}$ for all $\boldsymbol{n}$ (both admissible)

  then $\boldsymbol{h_2}$ dominates $\boldsymbol{h_1}$ and is better for search
- Typical search costs:
  - d = 14 IDS = 3,473,941 nodes

    A*(h_1) = 539 nodes

    A*(h_2) = 113 nodes
  - d = 24 IDS ≈ 54,000,000,000 nodes

    A*(h_1) = 39,135 nodes

    A*(h_2) = 1,641 nodes
- Given any admissible heuristics $\boldsymbol{h_a}$, $\boldsymbol{h_b}$,

  $$\boldsymbol{h(n) = \max(h_a(n), h_b(n))}$$

  is also admissible and dominates $\boldsymbol{h_a}$, $\boldsymbol{h_b}$

Memory-Based A*

- = combination between ***A**** and *beam-search*,

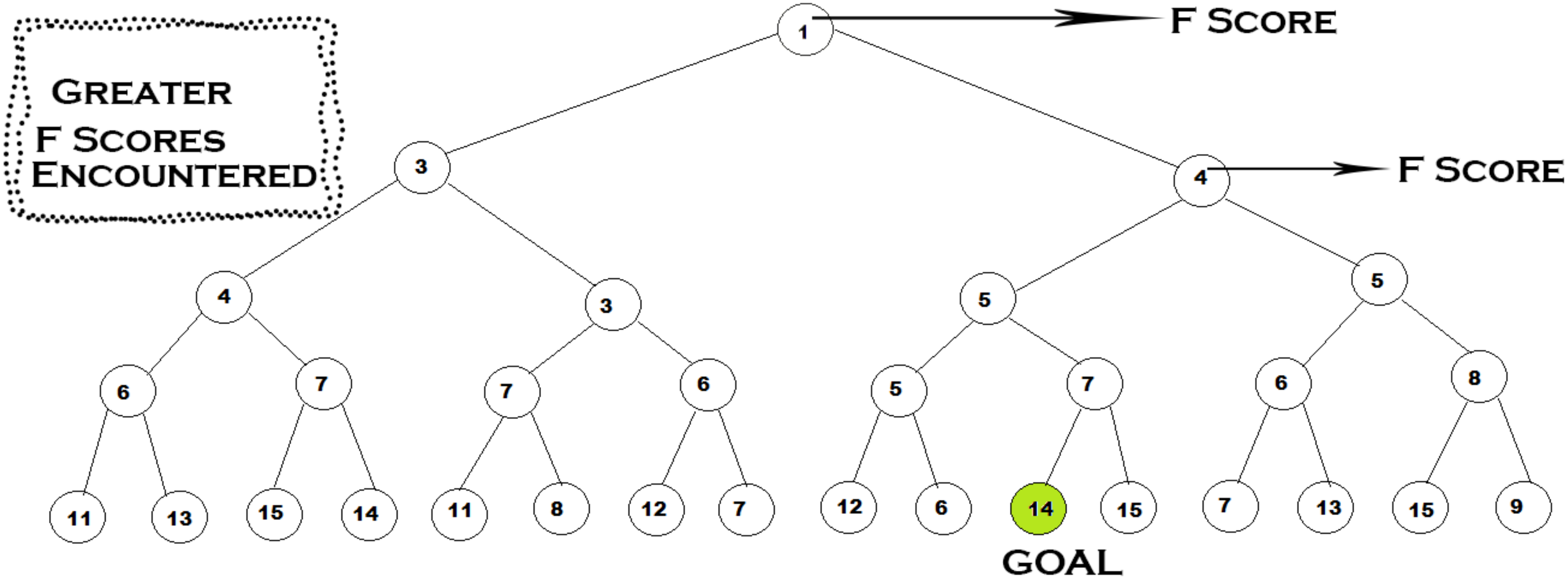
  i.e., keeps ***l*** nodes in the queue

Iterative Deepening A*

- = combination between ***A**** and *iterative-deepening-search*,

  i.e., uses ***A**** evaluation function as a *threshold* and runs *DLS*

Iterative Deepening A*



Copied from:

https://algorithmsinsight.files.wordpress.com/2016/03/ida-star.gif?w=1326

Summary of Algorithms

Criterion	**Greedy**	**Beam Search**	**Hill Climbing**	**A***	**IDA***
Complete	No*	No	No	Yes*	Yes
Time	b^m	blm	bm	b^d	b^d
Space	b^m	bl	b	b^d	bd
Optimal	No	No	No	Yes	Yes
Data Structure	*Priority Queue*	*Limited Priority Queue*	*„Limited Priority Queue“*	*Priority Queue*	*Stack*

Homework: *"Sliding Blocks (N Puzzle)"*

- The game starts with a board consisting of blocks numbered 1 through N and one blank block represented by the number 0. The goal is to arrange the tiles according to their numbers. Moving is done by moving the blocks on top, bottom, left and right in place of the empty block.
- At the input is given the number N - the number of blocks with numbers (8, 15, 24, etc.), the number I - the index of the position of zero (the empty block) in the decision (using -1 the default zero index position is set at the bottom right) and then the layout of the board is introduced. Using the **A*** (or **IDA***) algorithm and the *Manhattan distance heuristics (or Hemming distance)*, derive:
- In the first line, the length of the ***"optimal"*** path from start to destination.
- The appropriate steps (in a new line for each one) that are taken to reach the final state. The steps are **left**, **right**, **up** and **down**
- Keep in mind that not every puzzle is solvable. You can check whether the puzzle is solvable or directly use valid examples.

Homework: *"Sliding Blocks (N Puzzle)"*

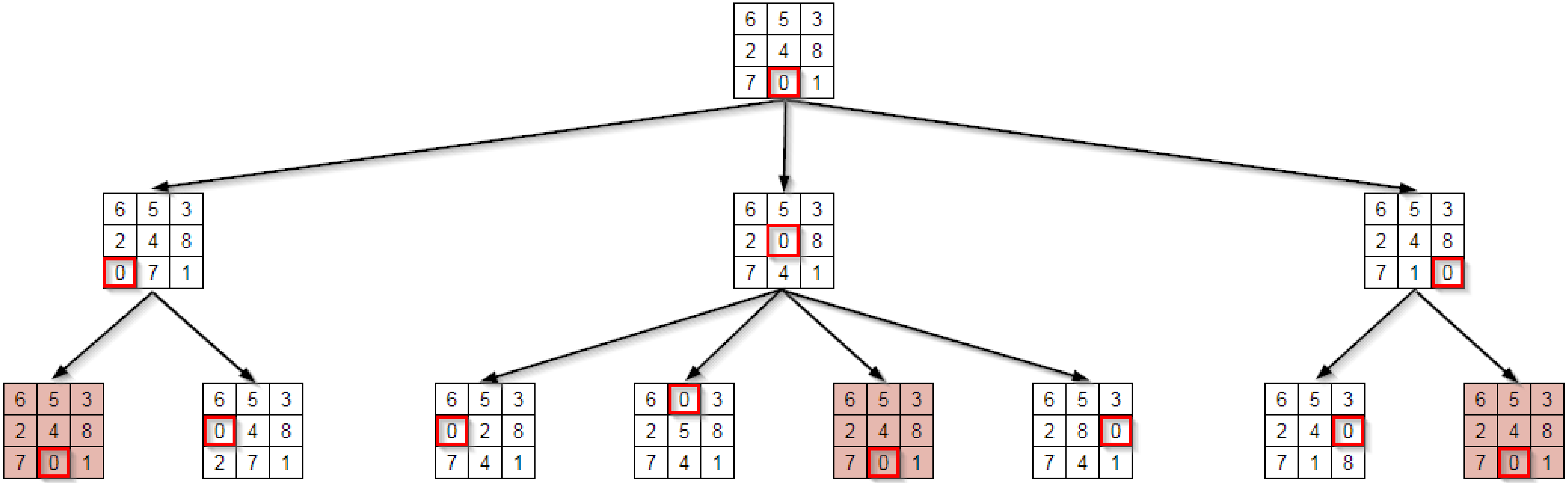
- **Sample input:**

```
8
-1
1 2 3
4 5 6
0 7 8
```

- **Sample output:**

```
2
left
left
```

Sliding Blocks (N Puzzle)



6 5 3
2 4 8
7 0 1

6 5 3
2 4 8
0 7 1

6 5 3
2 0 8
7 4 1

6 5 3
2 4 8
7 1 0

6 5 3
2 4 8
7 0 1

6 5 3
0 4 8
2 7 1

6 5 3
0 2 8
7 4 1

6 0 3
2 5 8
7 4 1

6 5 3
2 4 8
7 0 1

6 5 3
2 8 0
7 4 1

6 5 3
2 4 0
7 1 8

6 5 3
2 4 8
7 0 1

Sliding Blocks (N Puzzle)

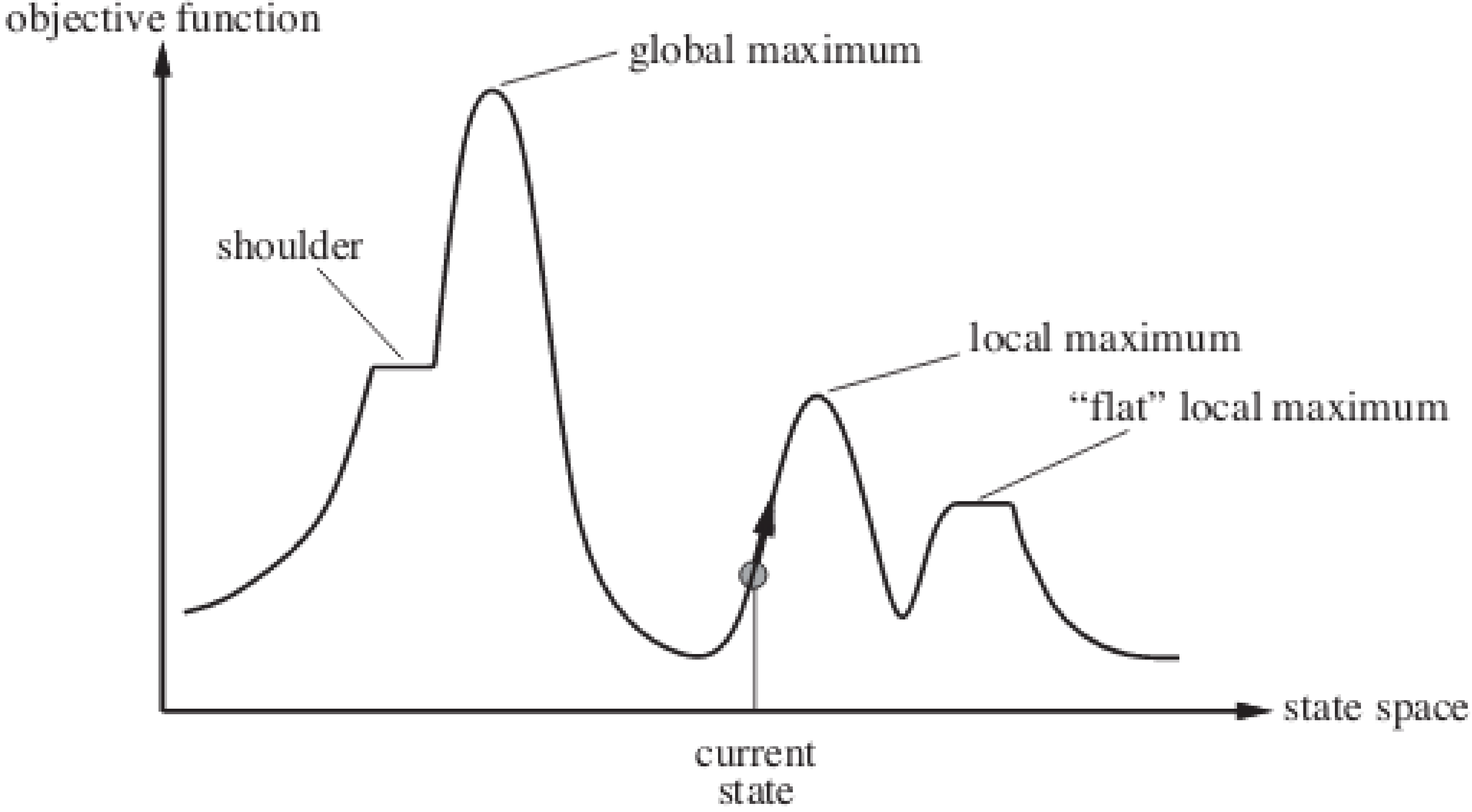
```
class Node {
    int[][] matrix = new int[MATRIX_DIMENSION][MATRIX_DIMENSION];
    int distance = 0;
    List<String> pathToTheNode;  //Or reference to the parent
}
```

Sliding Blocks (N Puzzle)

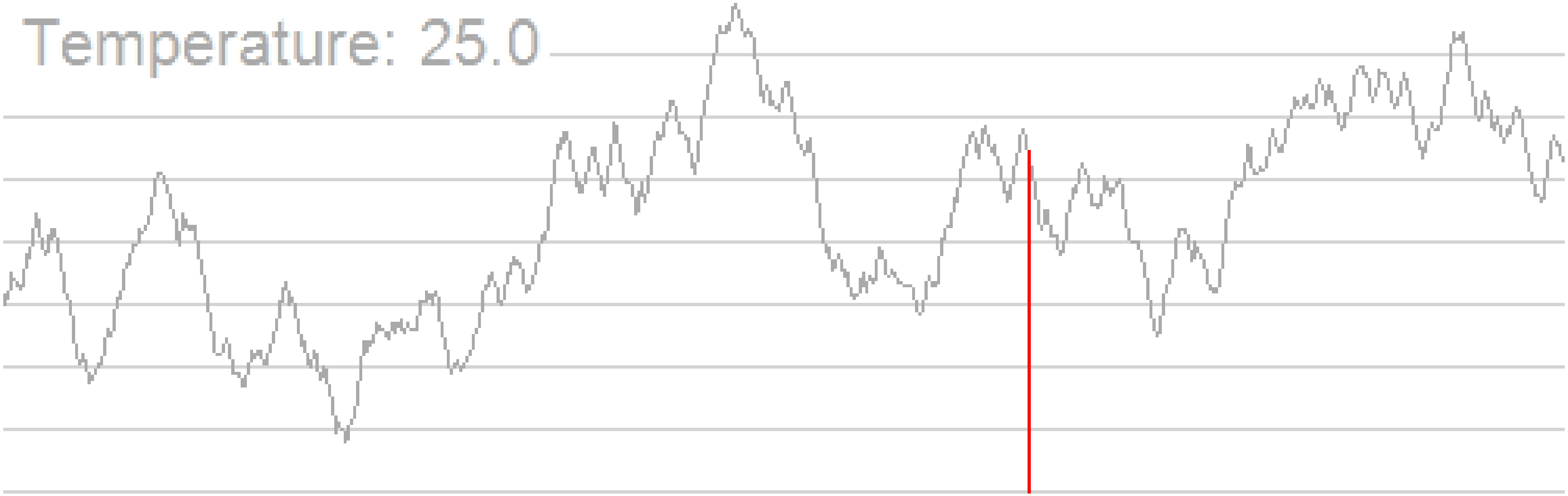
A*: Pseudocode

Iterative Deepening A*: Pseudocode

Local Search



Hill Climbing with Simulated Annealing


Temperature: 25.0