Lecture #24
State Space Search (2)

Algorithm

JBNU

Jinhong Jung

In This Lecture

❑ A* Search Algorithm

- Overview of A* search algorithm and applications

❑ A* algorithm for path finding

- Let's find the shortest path between two points in a map using A* algorithm

❑ A* algorithm for state space search

- Let's find the solution of TSP using A* algorithm

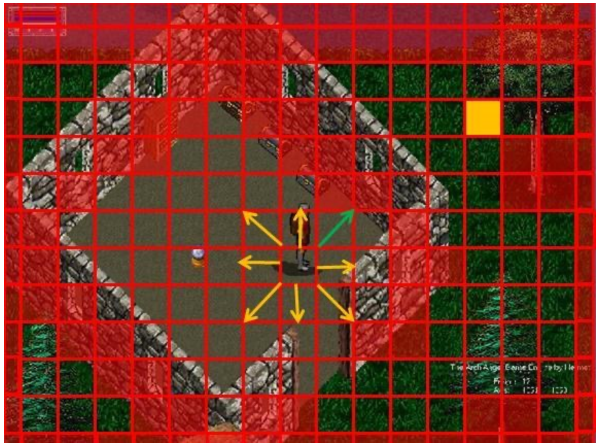
Outline

- Overview of A* algorithm
- A* algorithm for path finding
- A* algorithm for state space search

Overview Of A* Algorithm

❑ A* algorithm is a path search algorithm in a graph

- **Single-pair** shortest path problem
  - Given a graph and two nodes s and t, it aims to find the shortest path from starting node s to target node t
- Shows better performance than Dijkstra's algorithm
  - However, A* algorithm requires **a guide** for searching
  - The guide helps it find the shortest path without looking into other unpromising paths as much as possible
- Applications
  - Path finding (maze, game, etc.)
  - State space search



Best-First Search

- ❑ A search algorithm which explores a graph by expanding the most promising node at each step
  - Prim's algorithm
    - Incrementally expand the minimum spanning tree
    - **Criterion of "the best"**: minimum cost $C[u]$ computed so far
  - Dijkstra's algorithm
    - Incrementally expand the shortest path tree
    - **Criterion of "the best"**: shortest distance $D[u]$ computed so far
  - Both algorithms select **the best node** at each time
    - ⇒ These are best-first search algorithms
    - A* algorithm is also best-first search algorithm

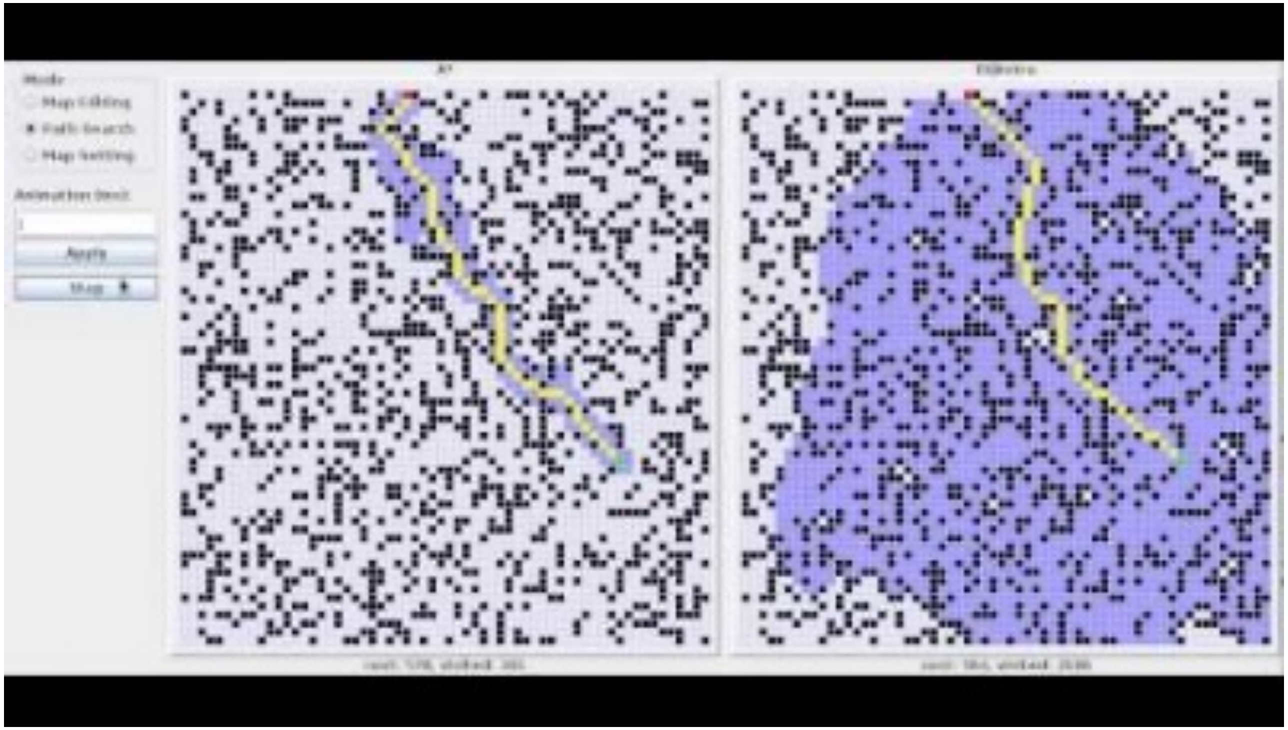
Outline

- ❑ Overview of A* algorithm
- ❑ **A* algorithm for path finding**
- ❑ A* algorithm for state space search

A* Algorithm For Path Finding

❑ A* algorithm finds single-pair shortest path

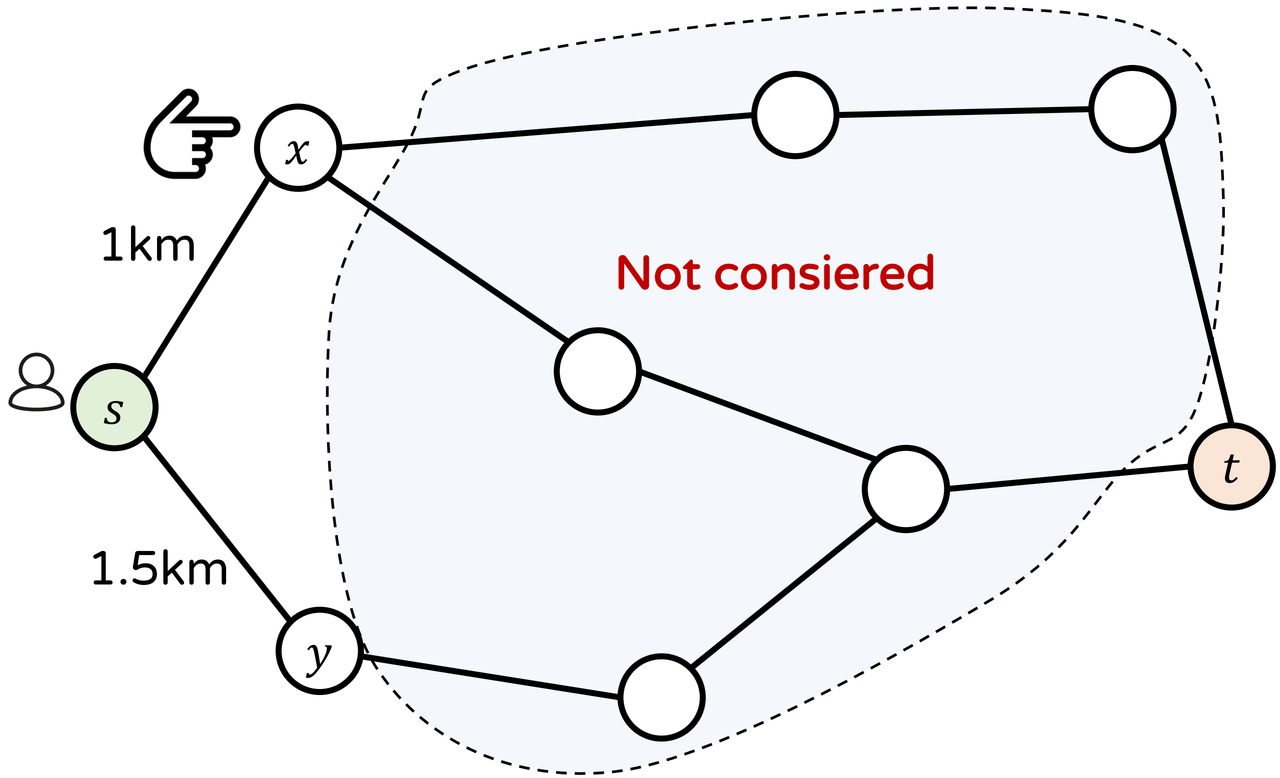
- In practical, Dijkstra's algorithm is slower than A* algorithm
  - Dijkstra's algorithm needs to find all paths to other nodes
- It requires a guide for efficiently searching for the path



https://www.youtube.com/watch?v=g024lzsknDo

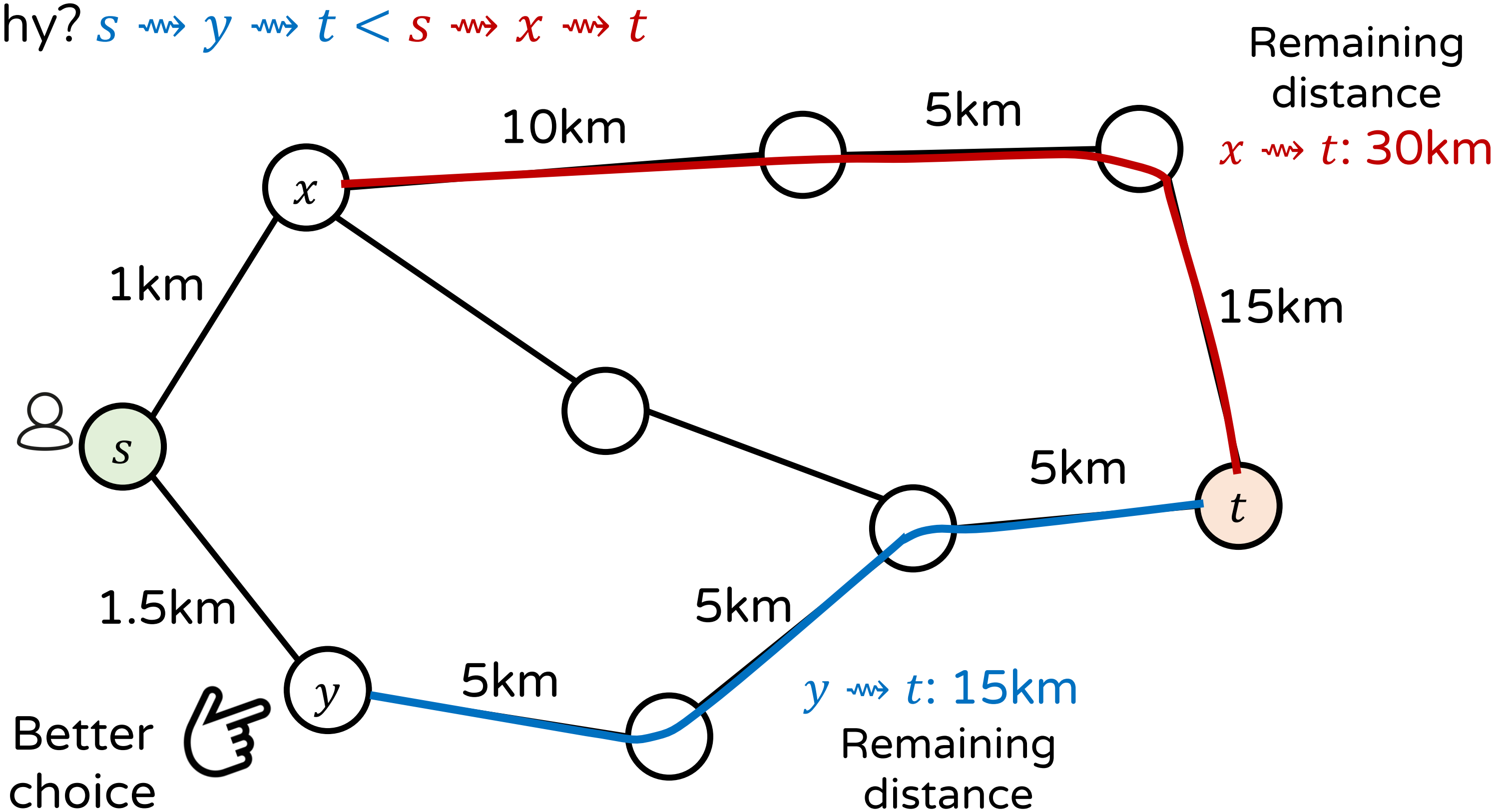
Intuition Of A* Algorithm (1)

- Suppose we need to choose node x or y for search
  - Which node should be selected?
    - Dijkstra's algorithm will select node x because it's shorter than y
    - Wait, is it the best?



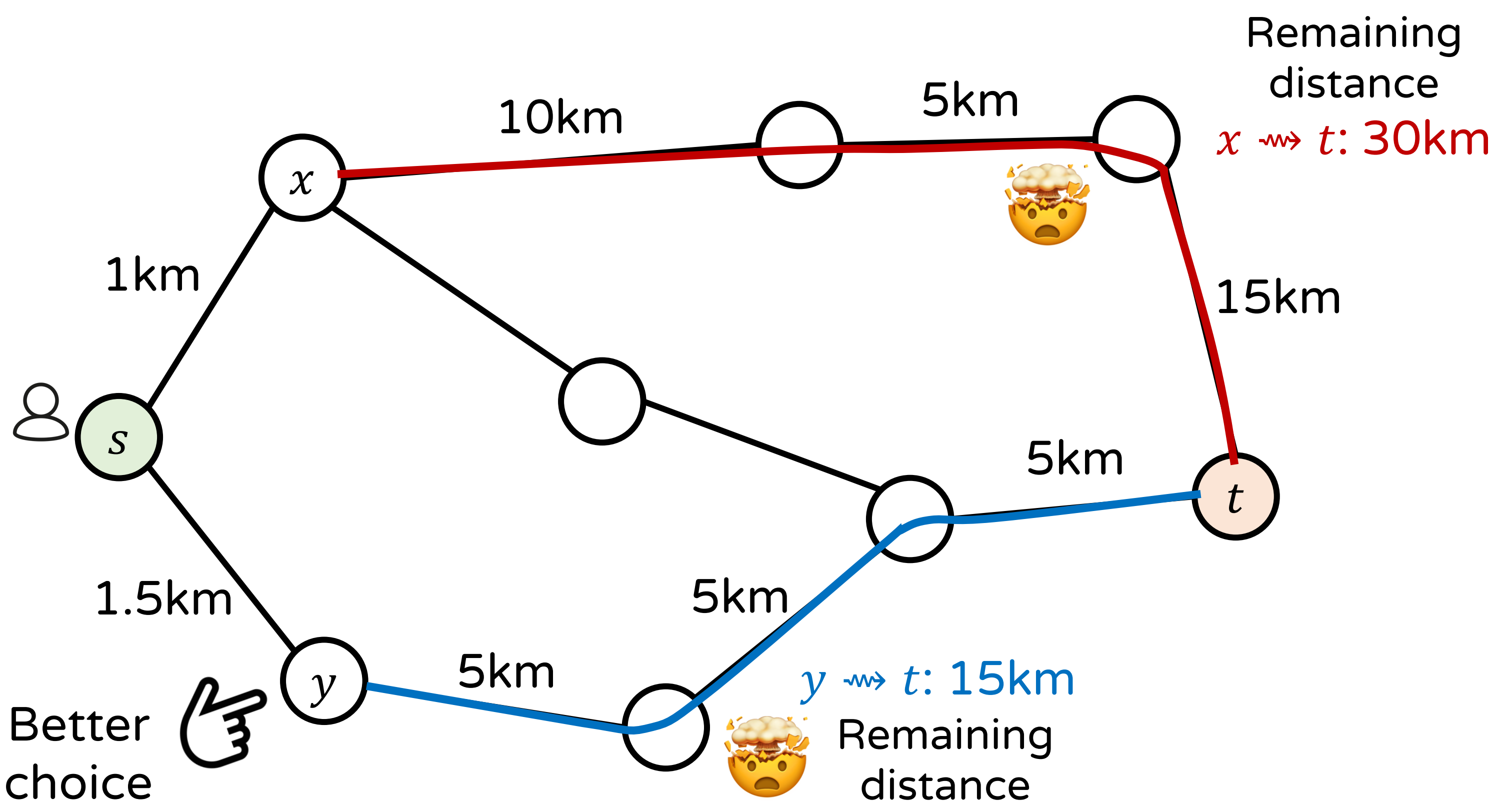
Intuition Of A* Algorithm (2)

- What if we know the remaining distance for each node to the target node?
  - No, it is better to select node y
    - Why? $s \rightsquigarrow y \rightsquigarrow t < s \rightsquigarrow x \rightsquigarrow t$



Intuition Of A* Algorithm (3)

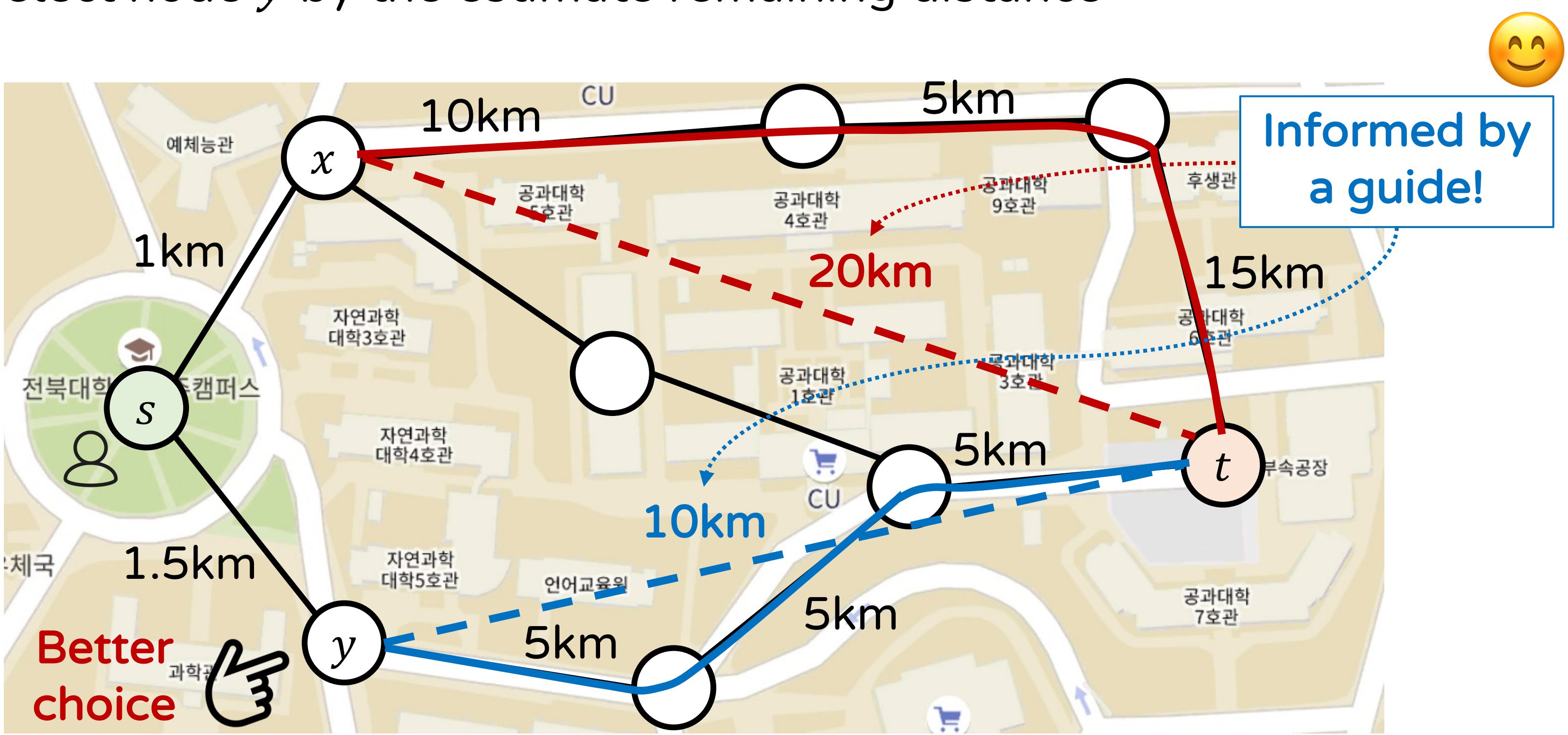
- However, how to know the remaining (shortest) distance in advance?
  - Need to compute the distances, recursively ⇒ inefficient



Intuition Of A* Algorithm (4)

❑ What if the graph is from a map?

- Then, it is easy to estimate the remaining distances via Pythagoras's theorem with coordinates
  - Select node y by the estimate remaining distance



Intuition Of A* Algorithm (5)

❑ Main idea of A* algorithm

- Let's consider the estimate remaining distances from each node to target node t
  - The estimate remaining distances should be easy-to-compute
  - e.g., Euclidean distances

- The criterion of "the best" on node x in A* algorithm
  - ⇒ Shortest distance from s to x + remaining distance from x to t

- Do best-first search from s with the criterion
  - Similar to Dijkstra's algorithm

A* Algorithm (1)

❑ Notations

- $D[x] \coloneqq D(s \rightsquigarrow x)$
  - The shortest distance (computed so far) from starting node s to node x
- $h[x] \coloneqq h(x \rightsquigarrow t)$
  - The estimate remaining distance from x to target node t
    - e.g., Euclidean distance between x and t
  - This is called (approximate) heuristics
  - Note that $h(x \rightsquigarrow t) \leq D(x \rightsquigarrow t)$
- $f[x] \coloneqq D(s \rightsquigarrow x) + h(x \rightsquigarrow t) = D[x] + h[x]$
  - The criterion of "**the best**" of node x
  - A node x having the minimum $f(x)$ will be selected first

A* Algorithm (2)

❑ Steps for A* algorithm

- **Step 1.** Estimate the remaining distances $h[x]$ from each node x to the target node t
- **Step 2.** Initialize the values of each node
  - $D[x] \leftarrow \infty$ and $f[x] \leftarrow \infty$ for $x \in V - \{s\}$
  - $D[s] \leftarrow 0$ and $f[s] \leftarrow D[s] + h[s]$ for starting node s
- **Step 3.** Pick the best among remaining nodes
  - A node having the minimum $f[x]$ value is the best
  - Using min-heap as Dijkstra's algorithm
- **Step 4.** Update the values of neighbors of the selected node (do relaxations on $D[x]$ and $f[x]$)

A* Algorithm (3)

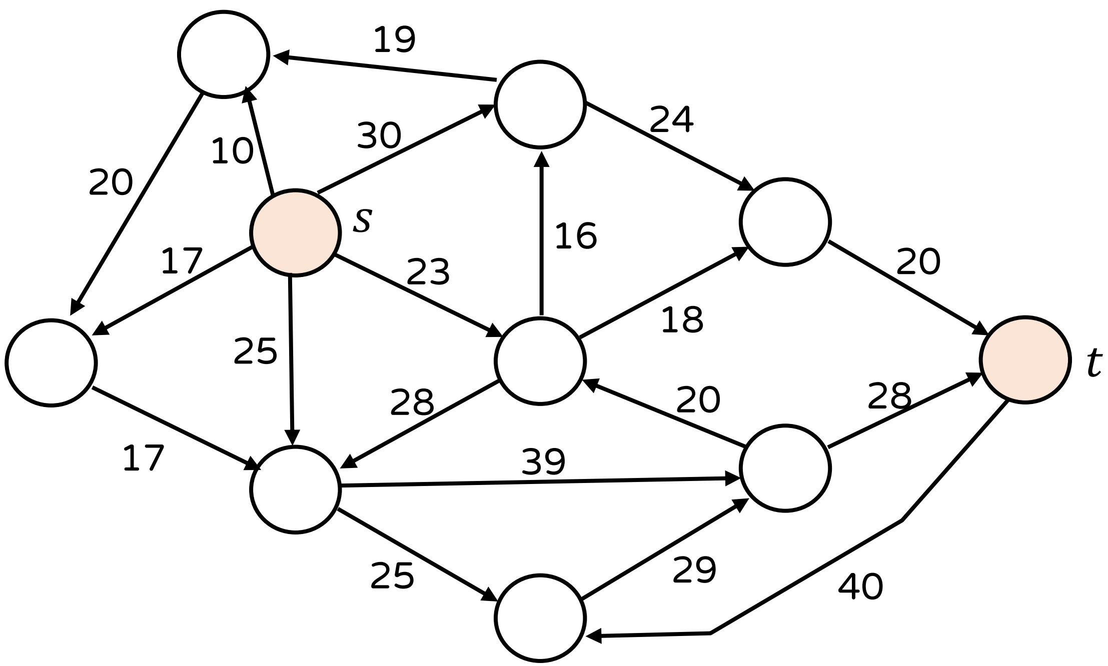
❑ A* algorithm is extended from Dijkstra's algorithm

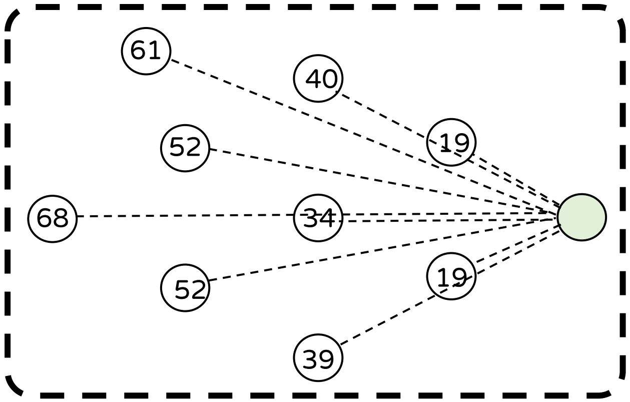
```
def dijkstra(G, s, t):
    Q ← min-heap()
    for each u in V − {s}:
        D[u] ← ∞
        Q.insert(u, D[u])
    D[s] ← 0
    Q.insert(s, D[s])

    while Q is not empty:
        u ← Q.remove()
        if u == t: return true

        for each v in N_u:
            if v ∈ Q and D[u] + w(u,v) < D[v]:
                D[v] ← D[u] + w(u,v)
                Q.decrease-key(v, D[v])

    return false
```

```
def a*-search(G, s, t):
    Q ← min-heap()
    for each u in V:
        h[u] ← estimate distance u ⇝ t
    for each u in V − {s}:
        D[u] ← f[u] ← ∞ & Q.insert(u, f[u])
    D[s] ← 0 & f(s) = D[s] + h[s]
    Q.insert(s, f[s])

    while Q is not empty:
        u ← Q.remove()
        if u == t: return true

        for each v in N_u:
            if v ∈ Q and D[u] + w(u,v) < D[v]:
                D[v] ← D[u] + w(u,v)
                f[v] ← D[v] + h[v]
                Q.decrease-key(v, f[v])
    return false
```

Example Of A* Algorithm (1)

- Estimate the remaining distances to node t



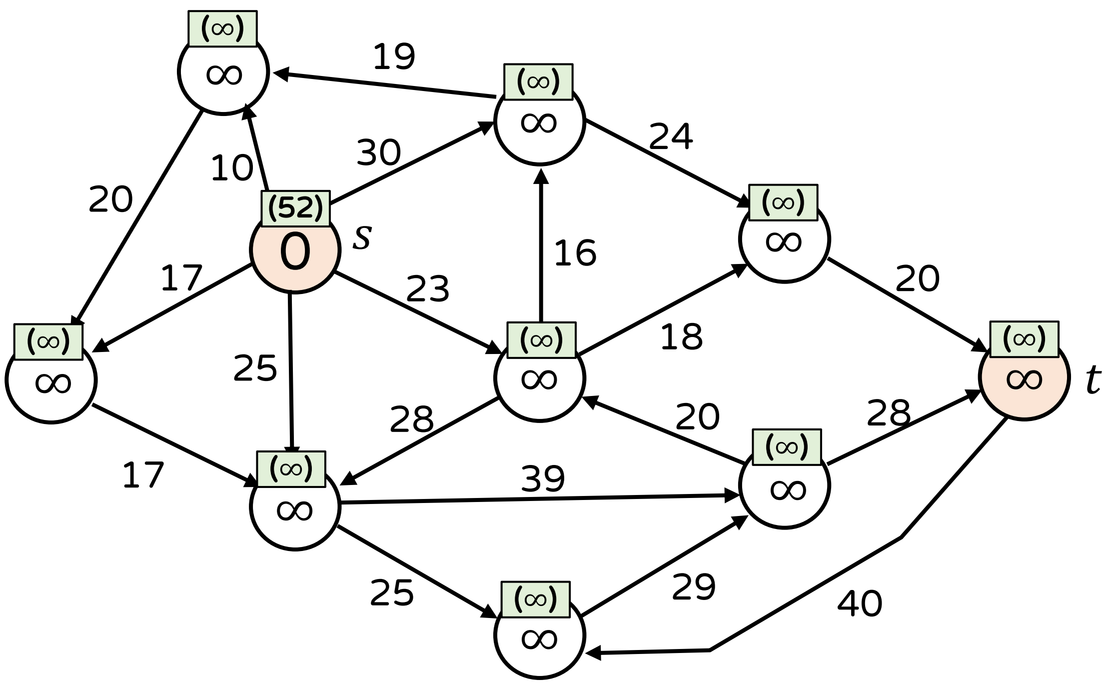


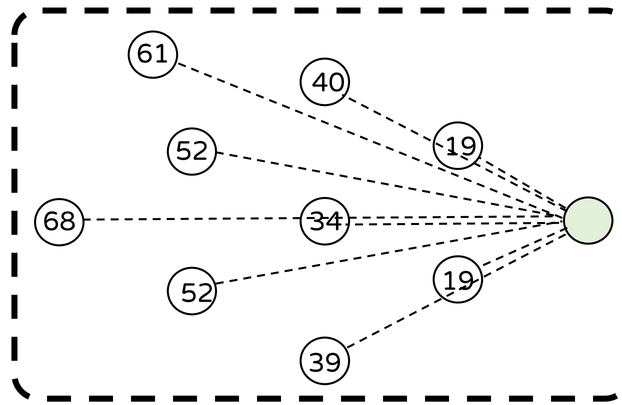
Estimate remaining distances
(as guide)

Example Of A* Algorithm (2)

❑ Initialization step

- Value in a circle (node x) is $D[x]$ - shortest distance from s to x
- Value above a circle (node x) is $f[x]$ - criterion of best



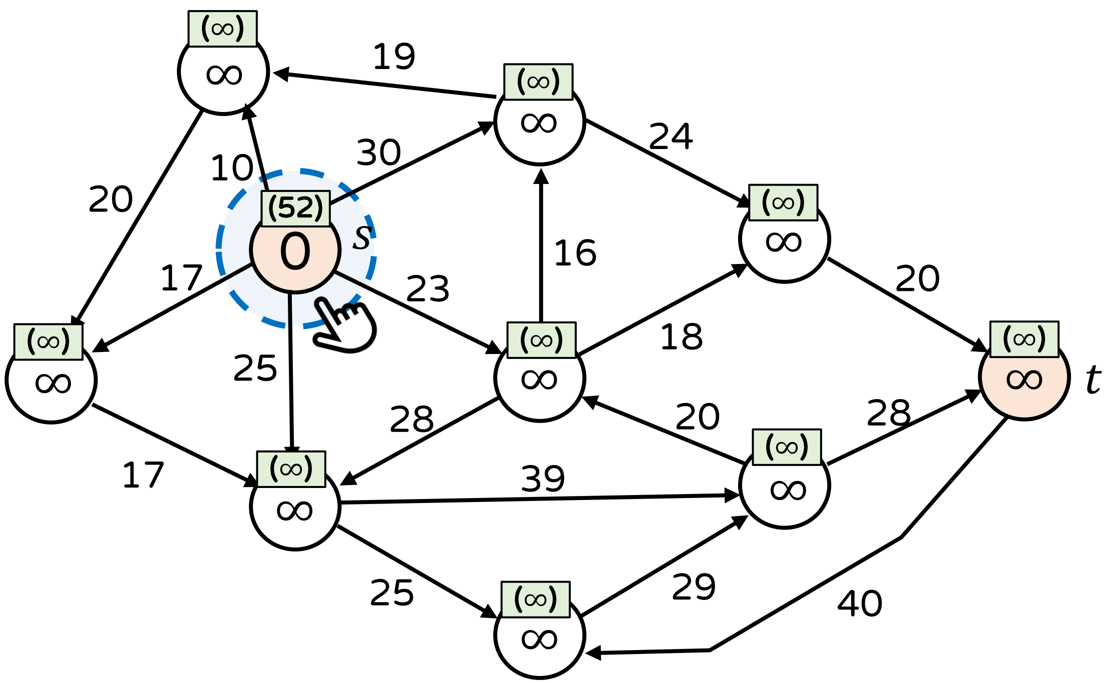


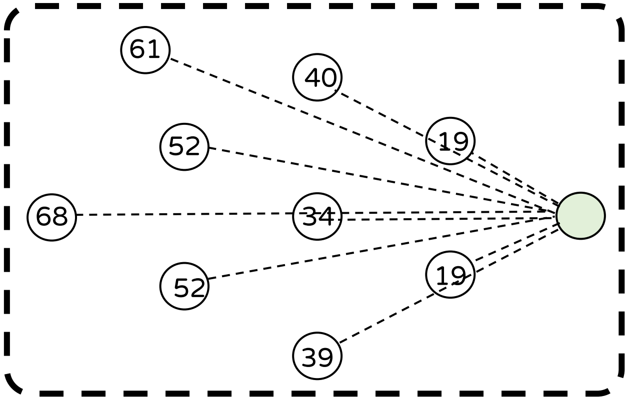
Estimate remaining distances
(as guide)

Example Of A* Algorithm (3)

❑ Pick the best among remaining nodes!

- Value in a circle (node x) is $D[x]$ - shortest distance from s to x
- Value above a circle (node x) is $f[x]$ - criterion of best



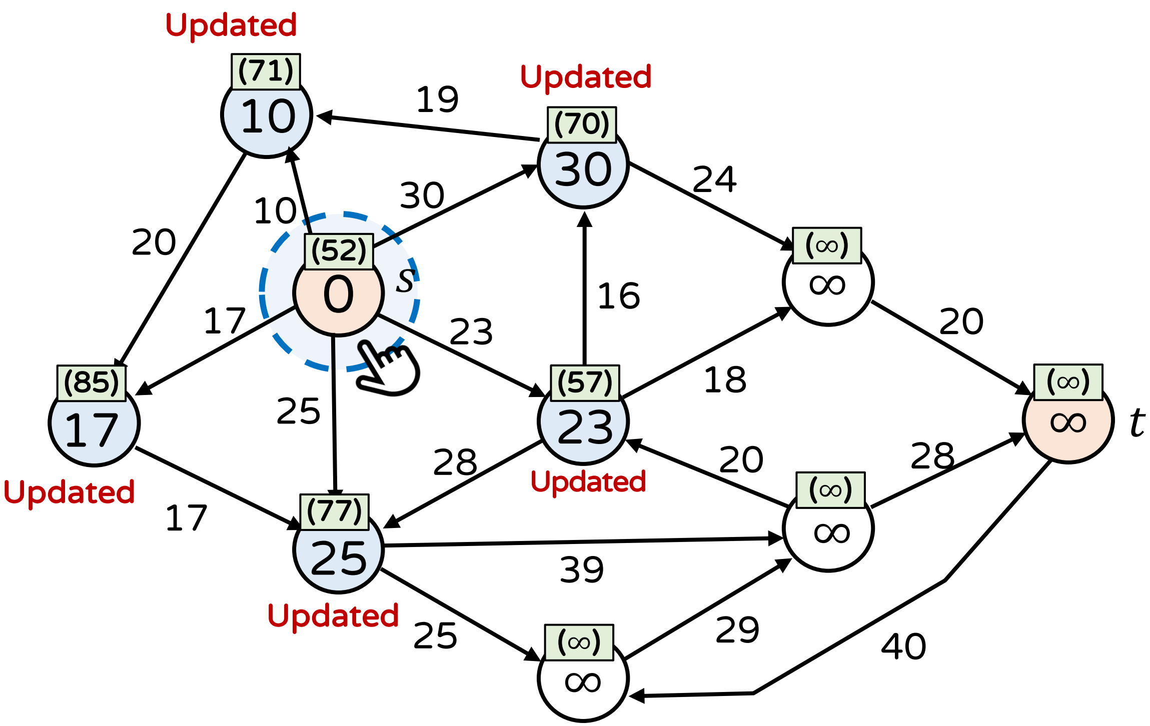


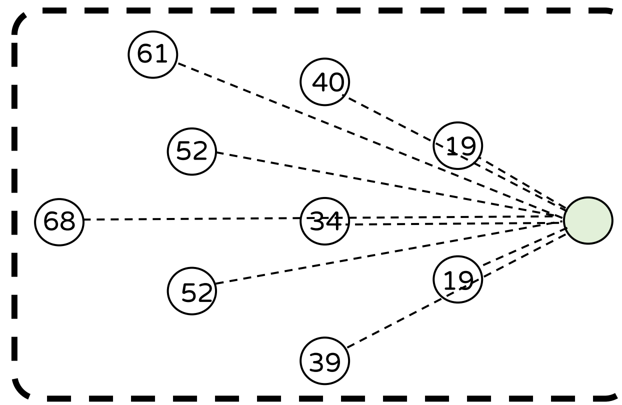
Estimate remaining distances
(as guide)

Example Of A* Algorithm (4)

❑ Update the values of neighbors of the picked node

- Value in a circle (node x) is $D[x]$ - shortest distance from s to x
- Value above a circle (node x) is $f[x]$ - criterion of best



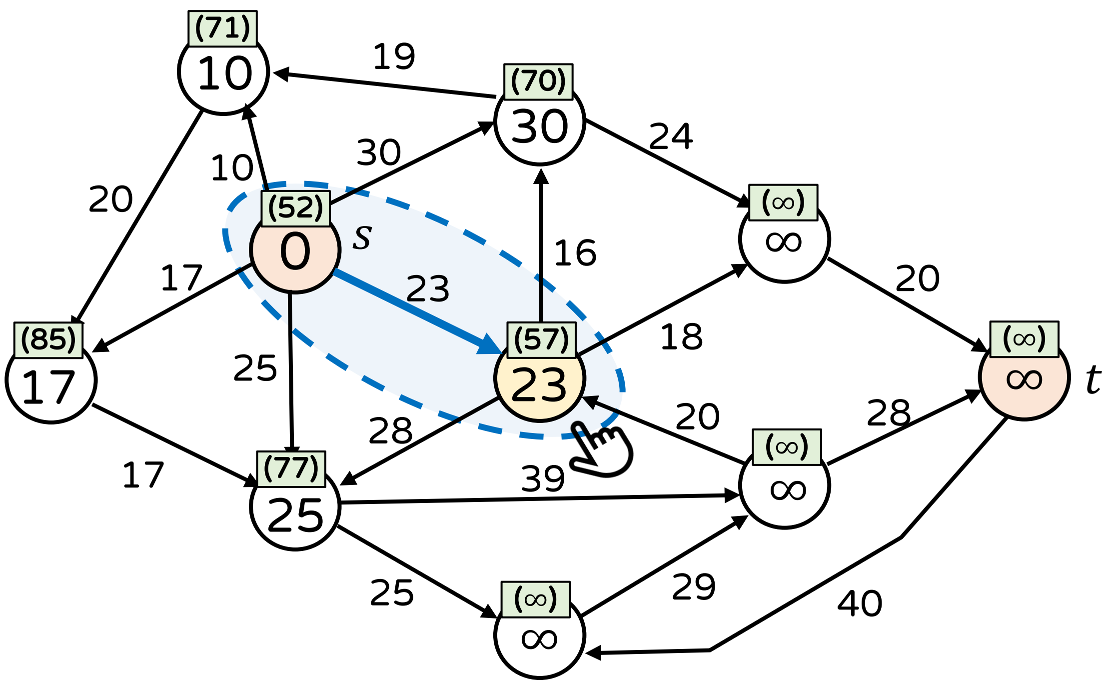


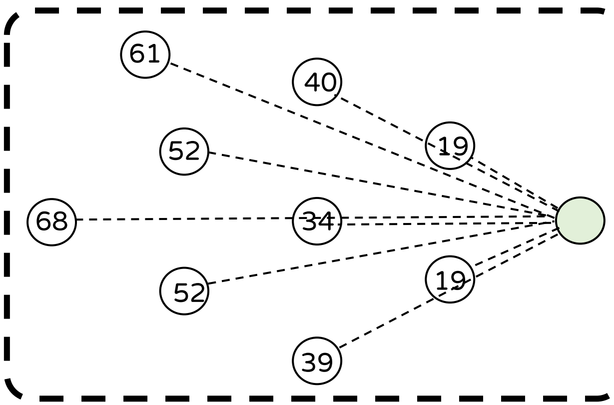
Estimate remaining distances (as guide)

Example Of A* Algorithm (5)

❑ Pick the best among remaining nodes!

- Value in a circle (node x) is $D[x]$ - shortest distance from s to x
- Value above a circle (node x) is $f[x]$ - criterion of best



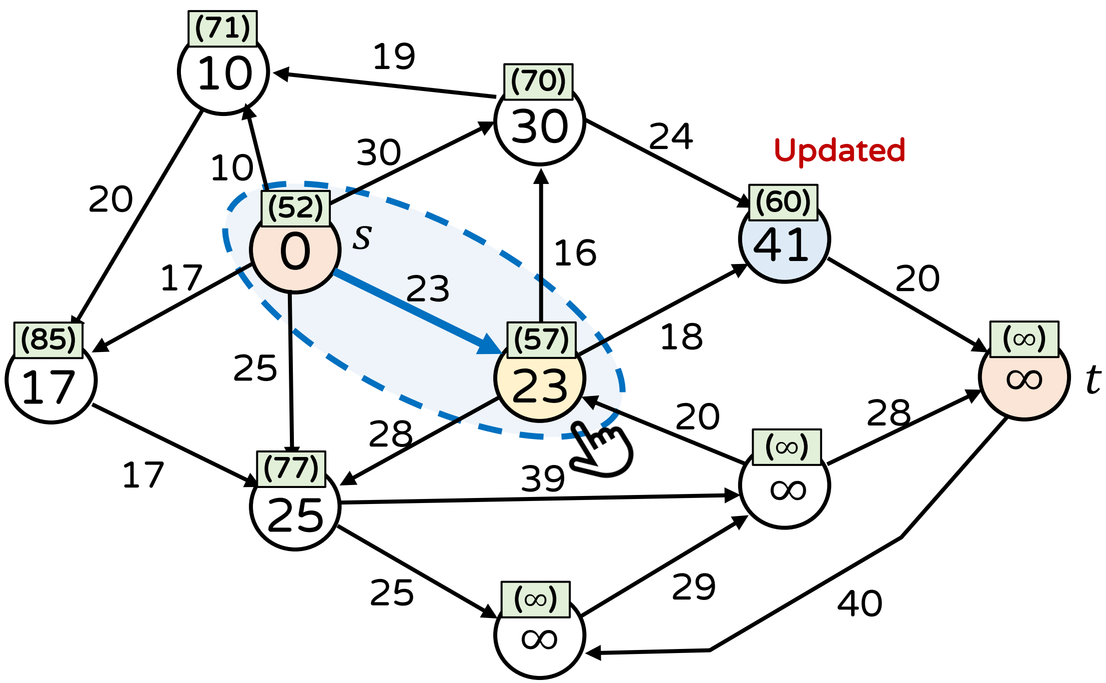


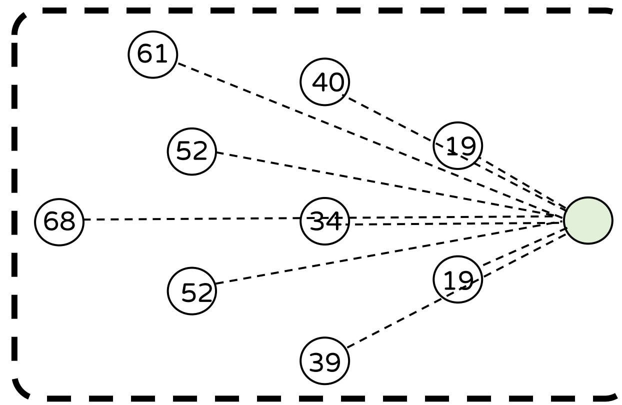
Estimate remaining distances
(as guide)

Example Of A* Algorithm (6)

❑ Update the values of neighbors of the picked node

- Value in a circle (node x) is $D[x]$ - shortest distance from s to x
- Value above a circle (node x) is $f[x]$ - criterion of best



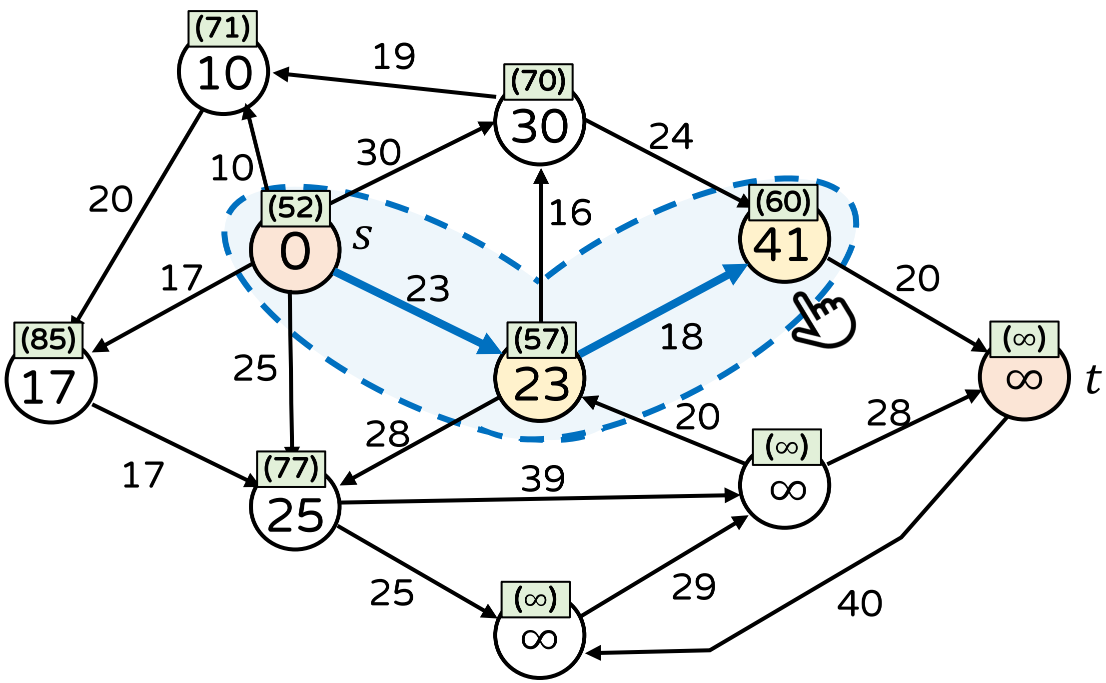


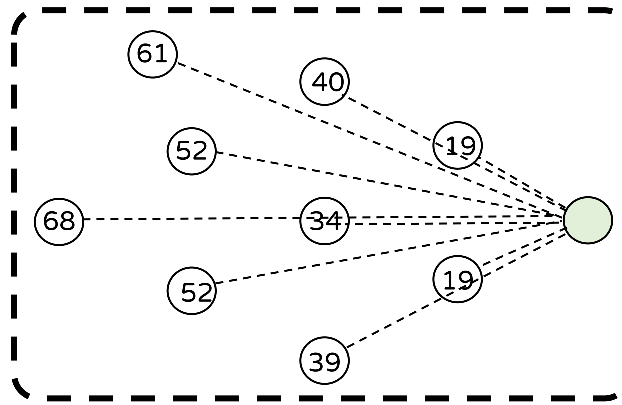
Estimate remaining distances
(as guide)

Example Of A* Algorithm (7)

❑ Pick the best among remaining nodes!

- Value in a circle (node x) is $D[x]$ - shortest distance from s to x
- Value above a circle (node x) is $f[x]$ - criterion of best



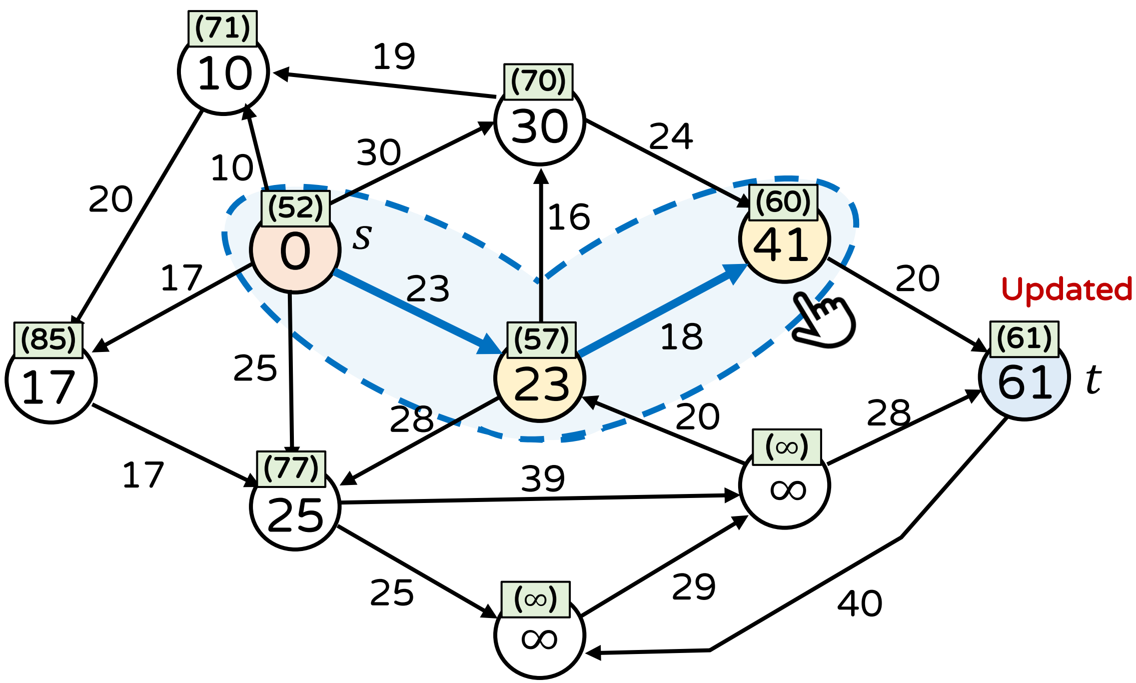


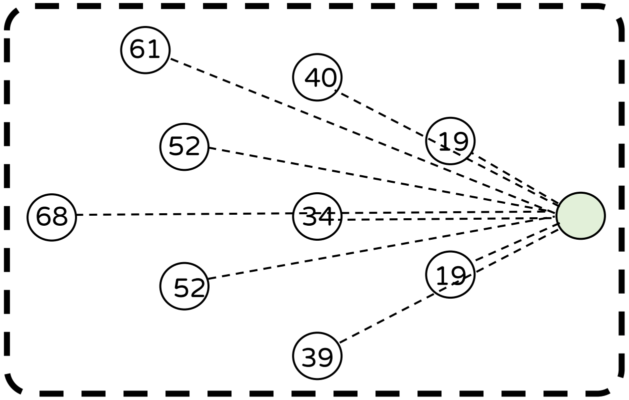
Estimate remaining distances
(as guide)

Example Of A* Algorithm (8)

❑ Update the values of neighbors of the picked node

- Value in a circle (node x) is $D[x]$ - shortest distance from s to x
- Value above a circle (node x) is $f[x]$ - criterion of best



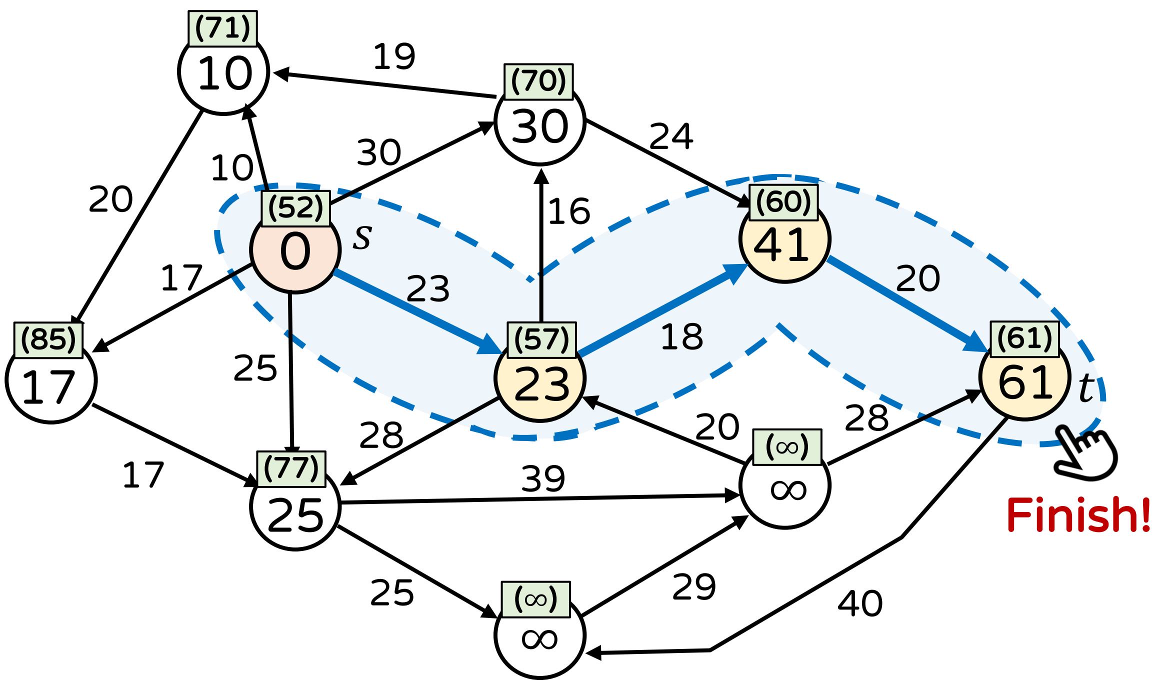


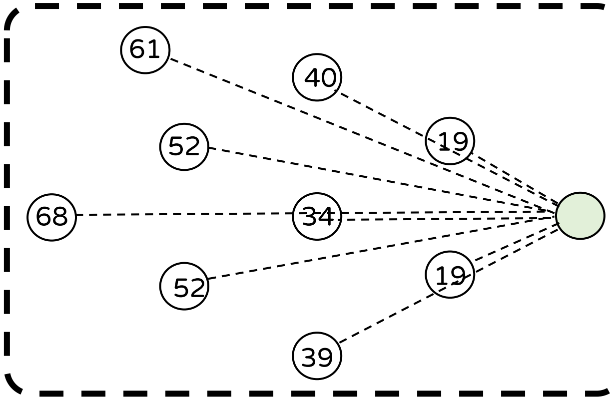
Estimate remaining distances (as guide)

Example Of A* Algorithm (9)

❑ Pick the best among remaining nodes!

- Value in a circle (node x) is $D[x]$ - shortest distance from s to x
- Value above a circle (node x) is $f[x]$ - criterion of best





Estimate remaining distances
(as guide)

Discussion (1)

❑ Optimality of A* algorithm

- For each node pair (x, y), if $h(x) \leq w(x, y) + h(y)$, then A* algorithm produces an optimal solution [Monotonicity]
  - Euclidean distance (ED) satisfies the above property
    - $h(x) \leq \mathrm{ED}(x, y) + h(y) \Rightarrow h(x) \leq w(x, y) + h(y)$
- Applicable domains
  - Find a path in a maze or a map of a game

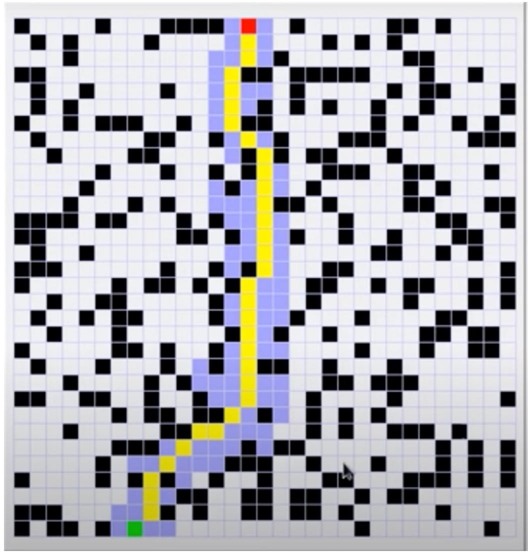
❑ Importance of heuristics

- If $h[x] = 0\ \forall\ x \in V$, A* algorithm = Dijkstra's algorithm
  - Meaning this heuristics is poor
- A good heuristics improve the search performance

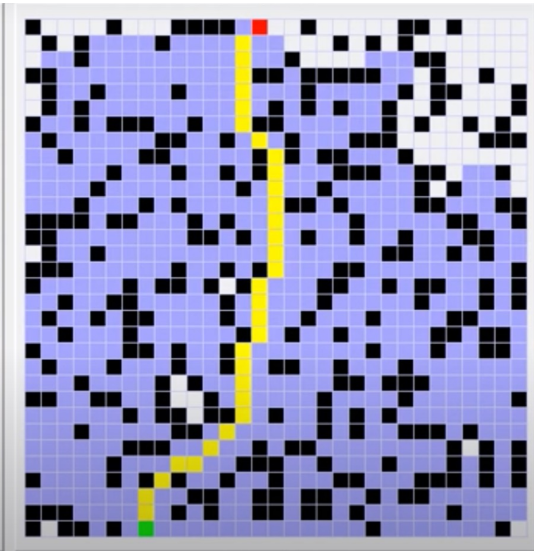
Discussion (2)

❑ Time complexity of A* algorithm for path finding

- For a worst case, it takes $O(m \log n)$ time (=Dijkstra's)
  - If they use min-heap
- However, it's much faster than Dijkstra's one in practical
  - For finding single-pair shortest path
  - Dijkstra's algorithm is for single-source shortest path



A* algorithm



Dijkstra's algorithm

Outline

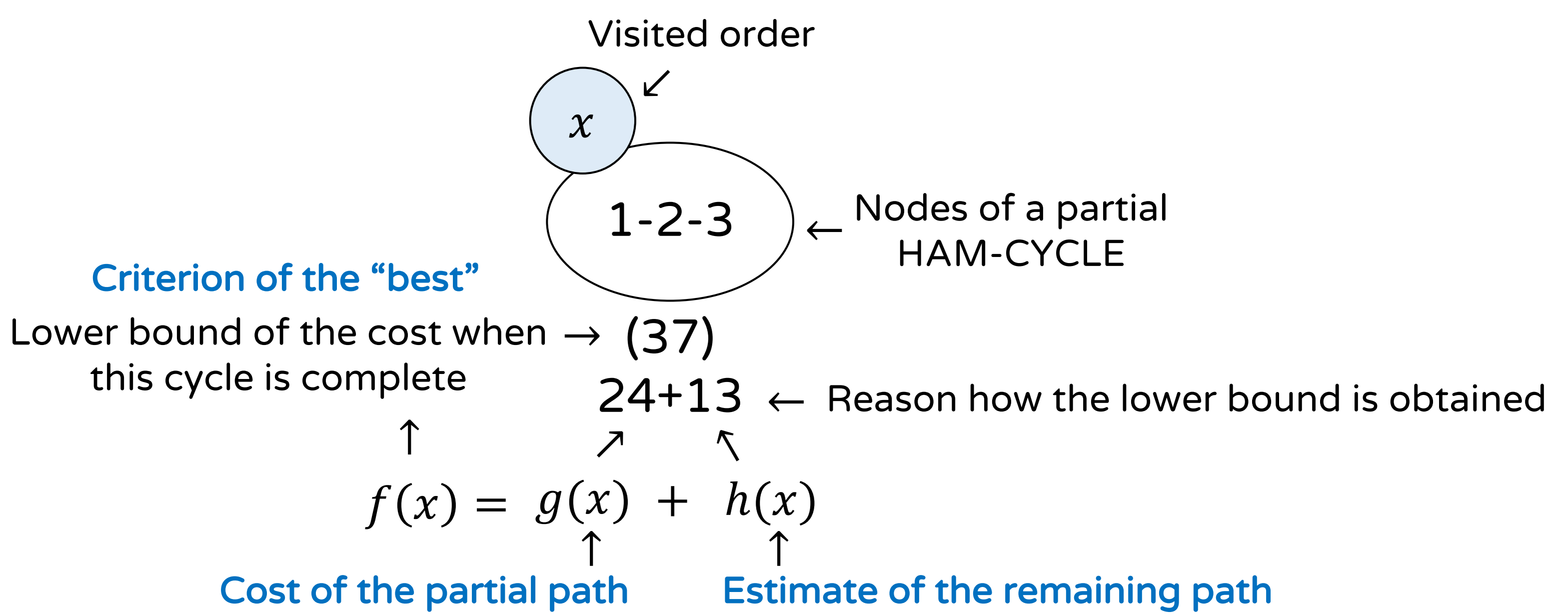
- ❑ Overview of A* algorithm
- ❑ A* algorithm for path finding
- ❑ **A* algorithm for state space search**

A* for State Space Search

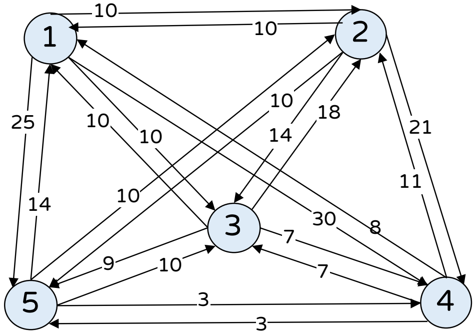
❑ A* algorithm can be used for state space search

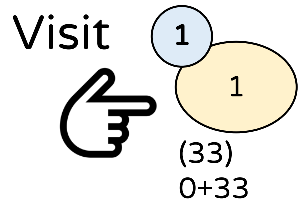
- Apply A*'s strategy to a state space tree
  - Starting node = a root, target node = a leaf
- As backtracking, the tree expands during the search

❑ Example of TSP

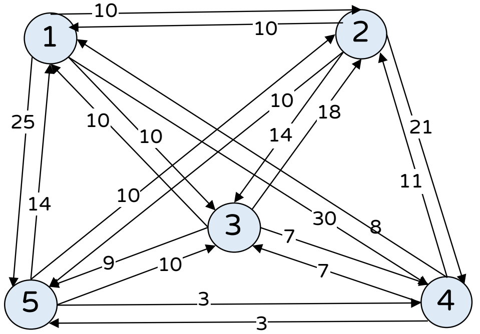


Example for SSS (1)





Example for SSS (2)



Visit 1

1
(33)
0+33

Update the values of its neighbors

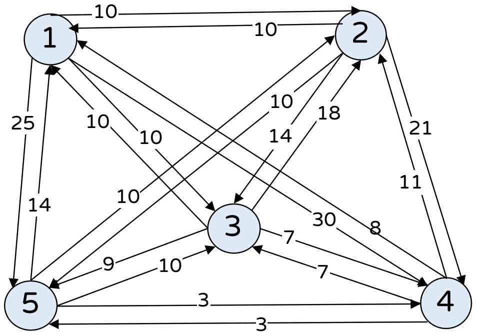
1-2 (33)
10+23

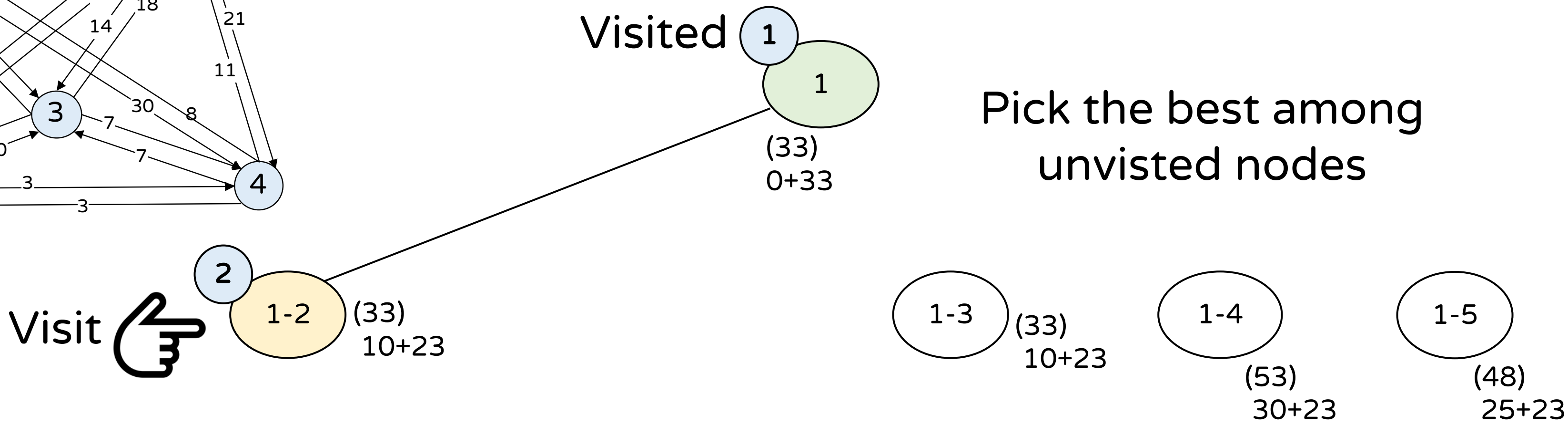
1-3 (33)
10+23

1-4 (53)
30+23

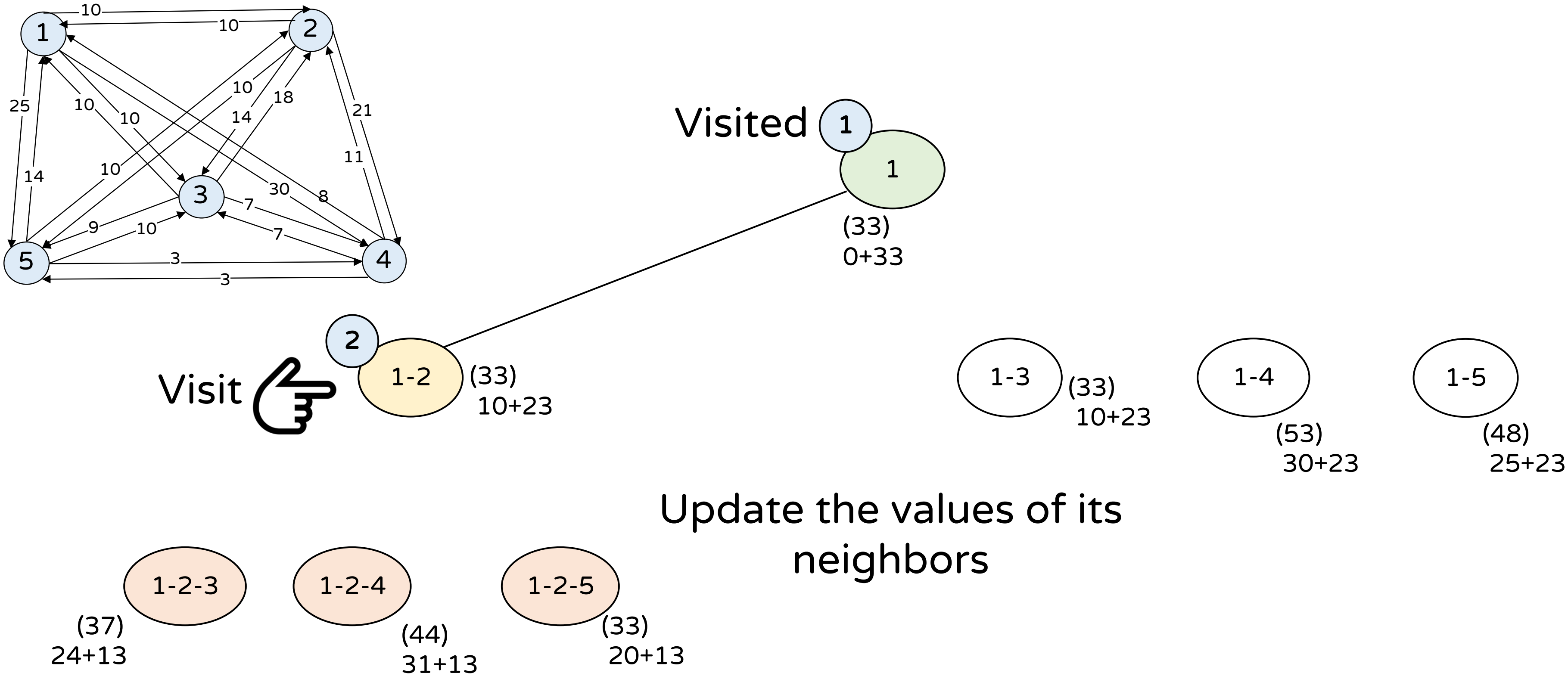
1-5 (48)
25+23

Example for SSS (3)

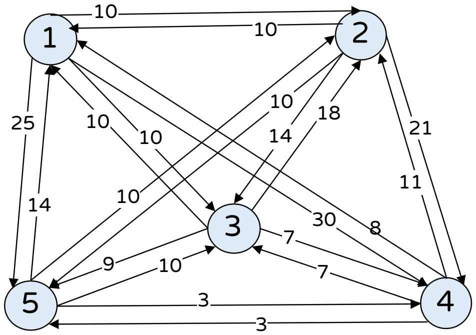


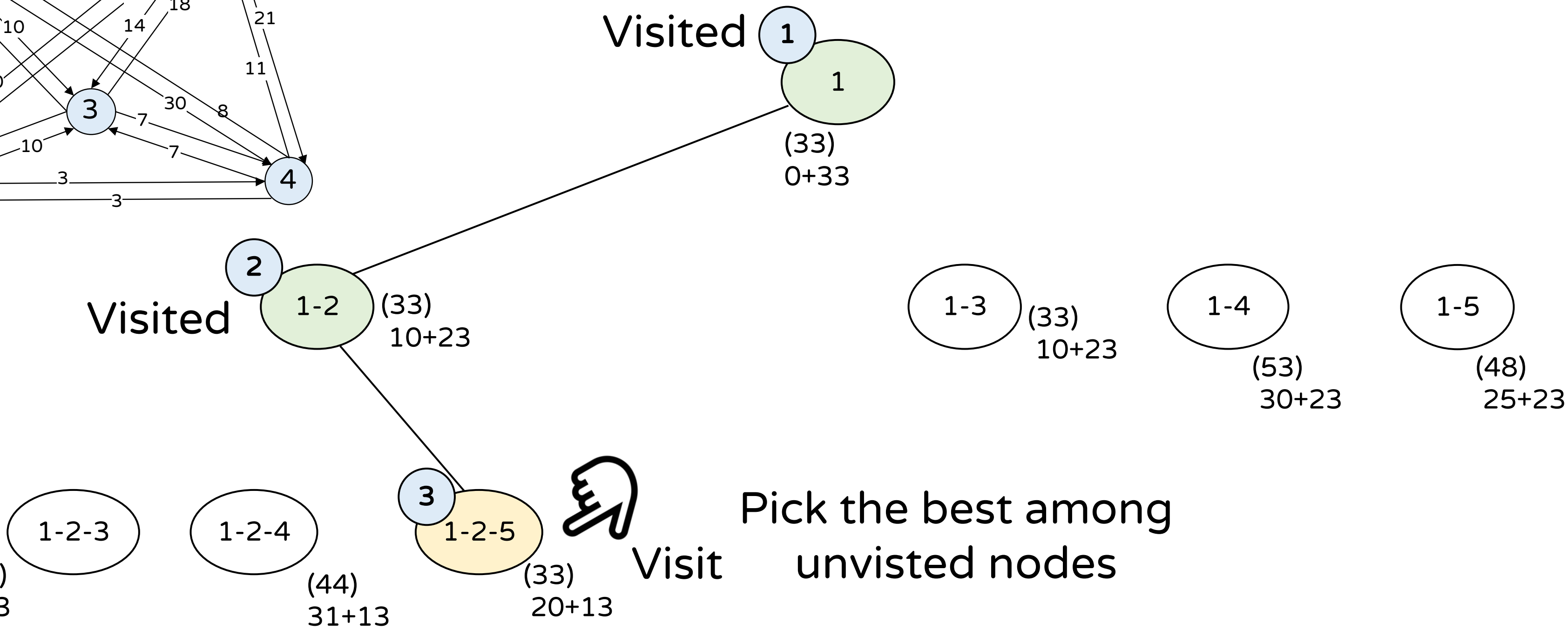


Example for SSS (4)

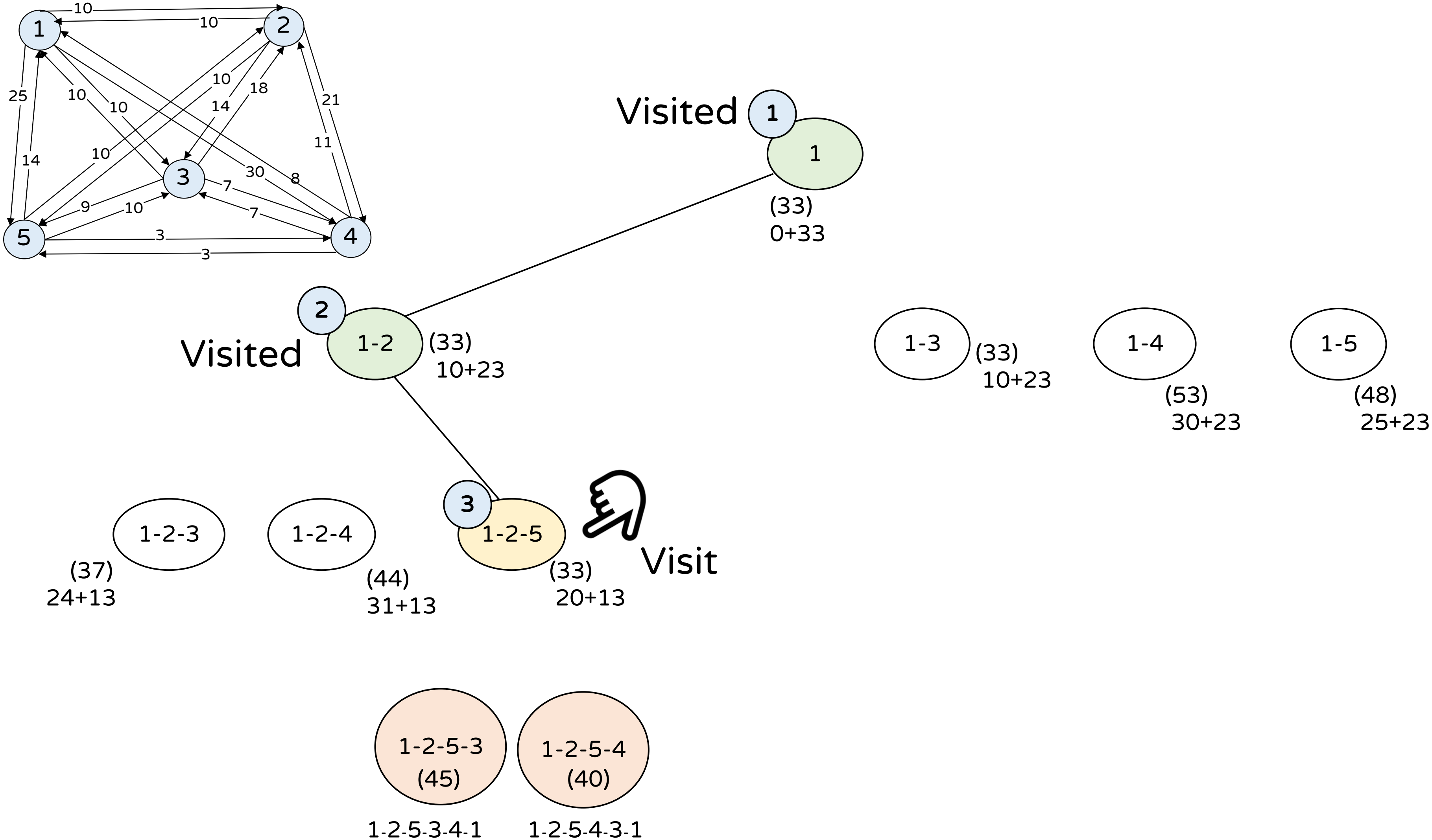


Example for SSS (5)



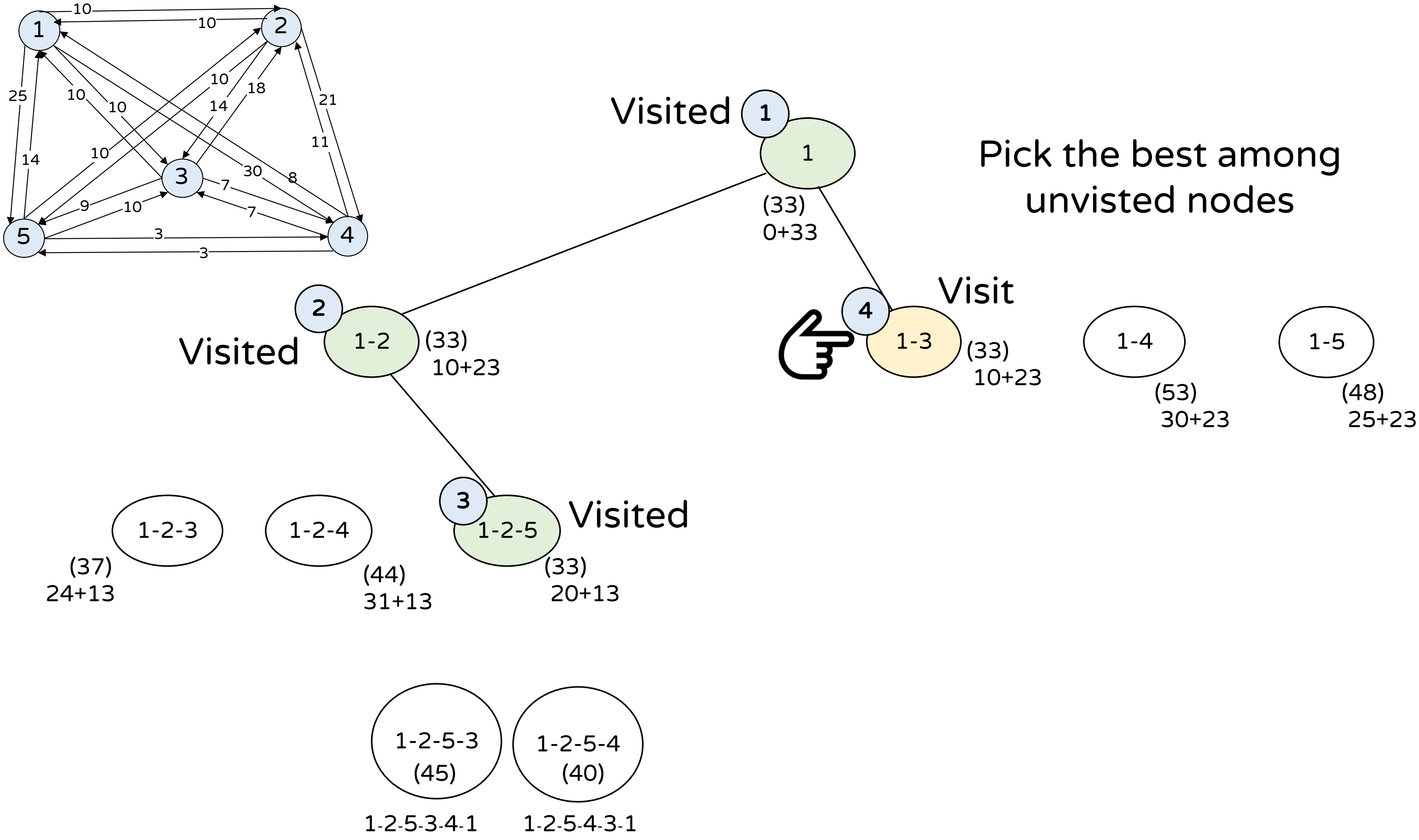


Example for SSS (6)

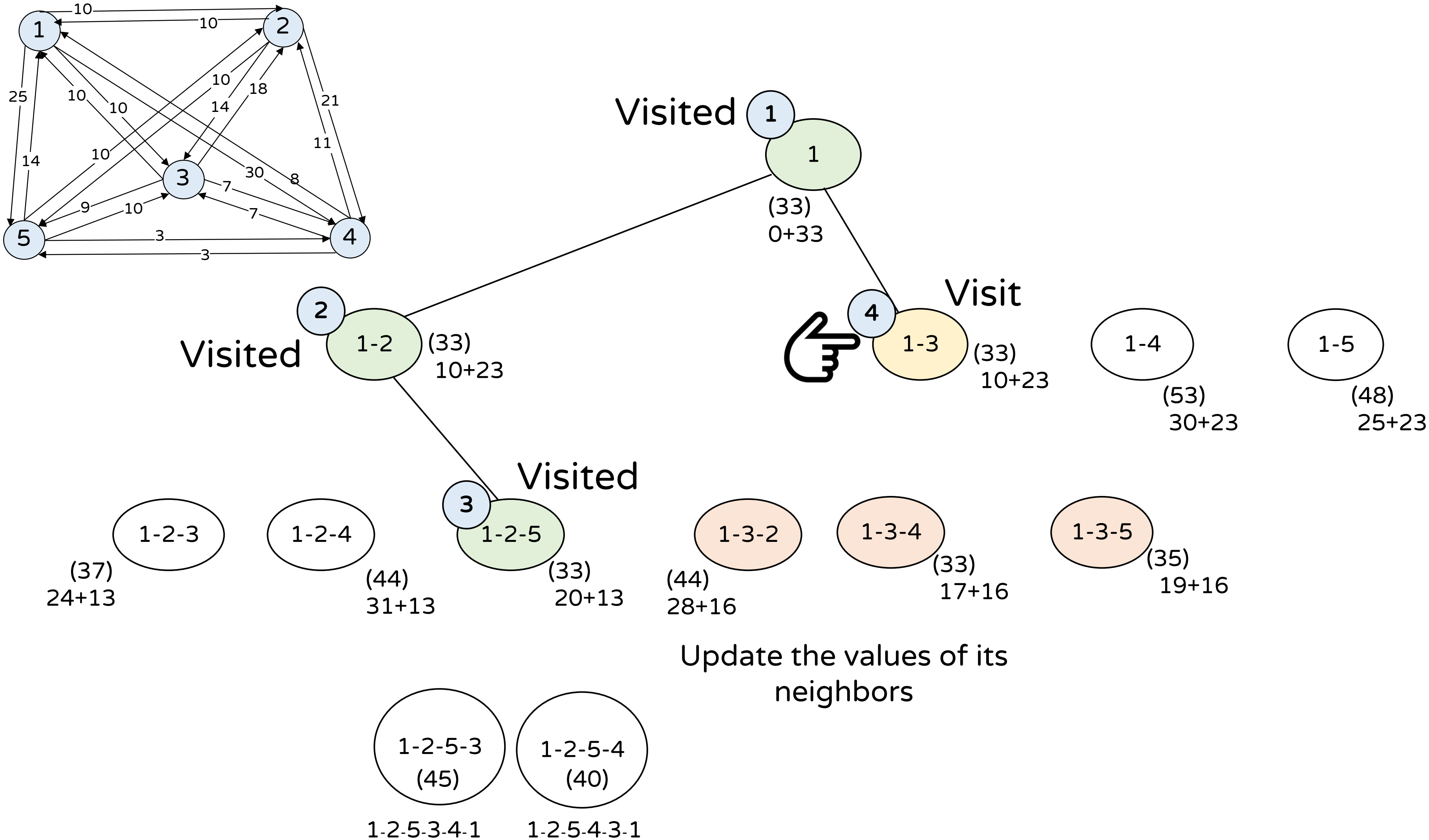


Update the values of its neighbors

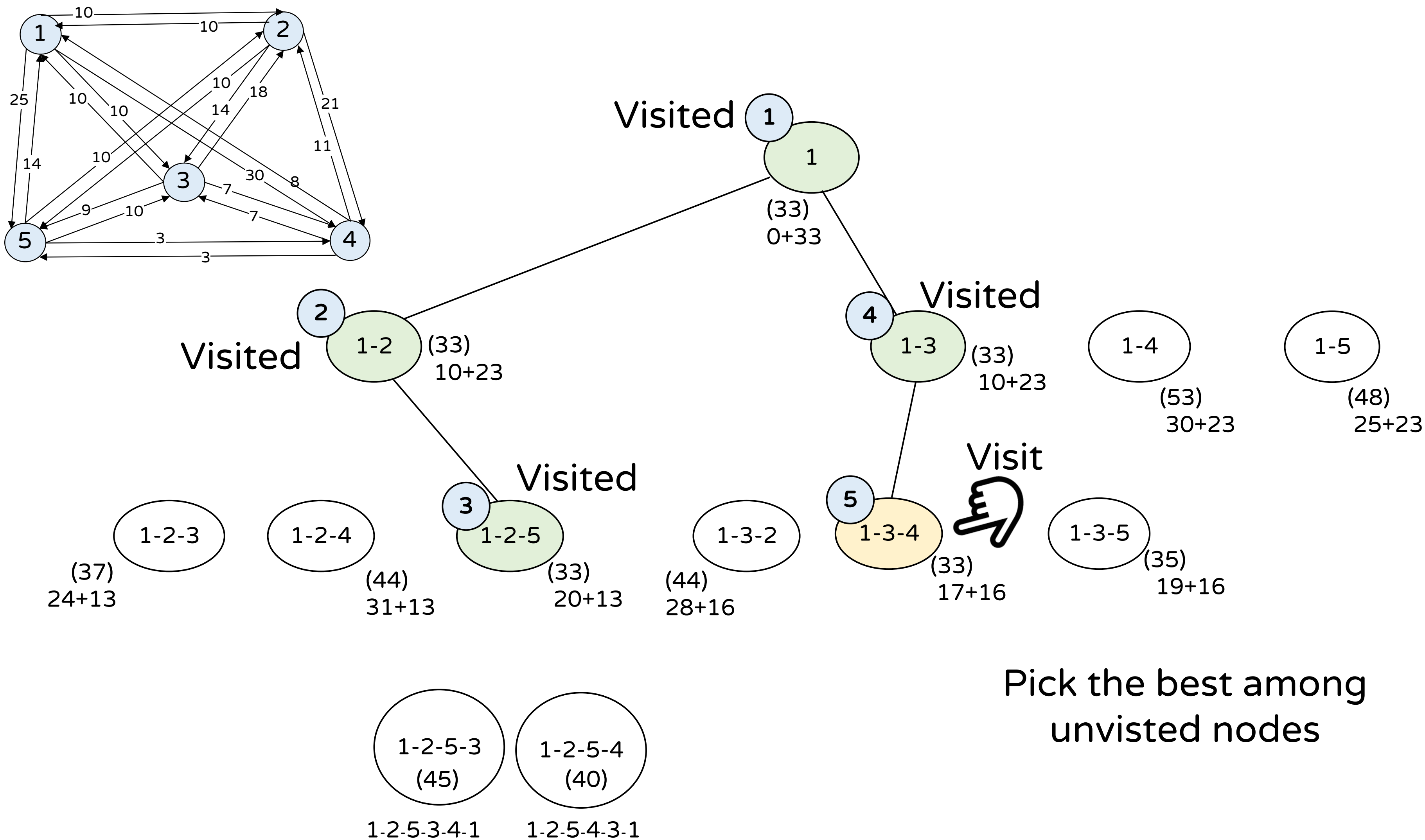
Example for SSS (7)



Example for SSS (8)

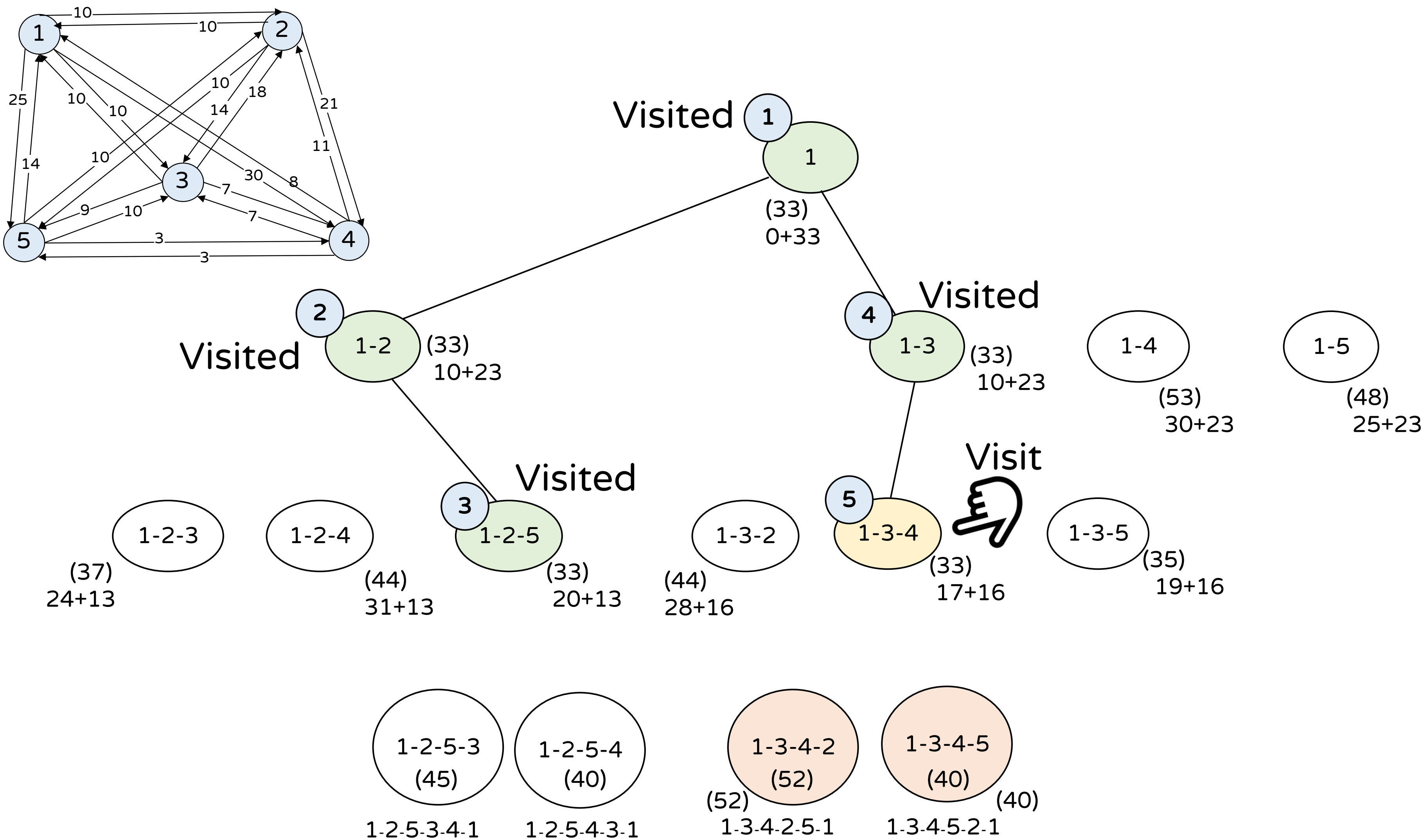


Example for SSS (9)



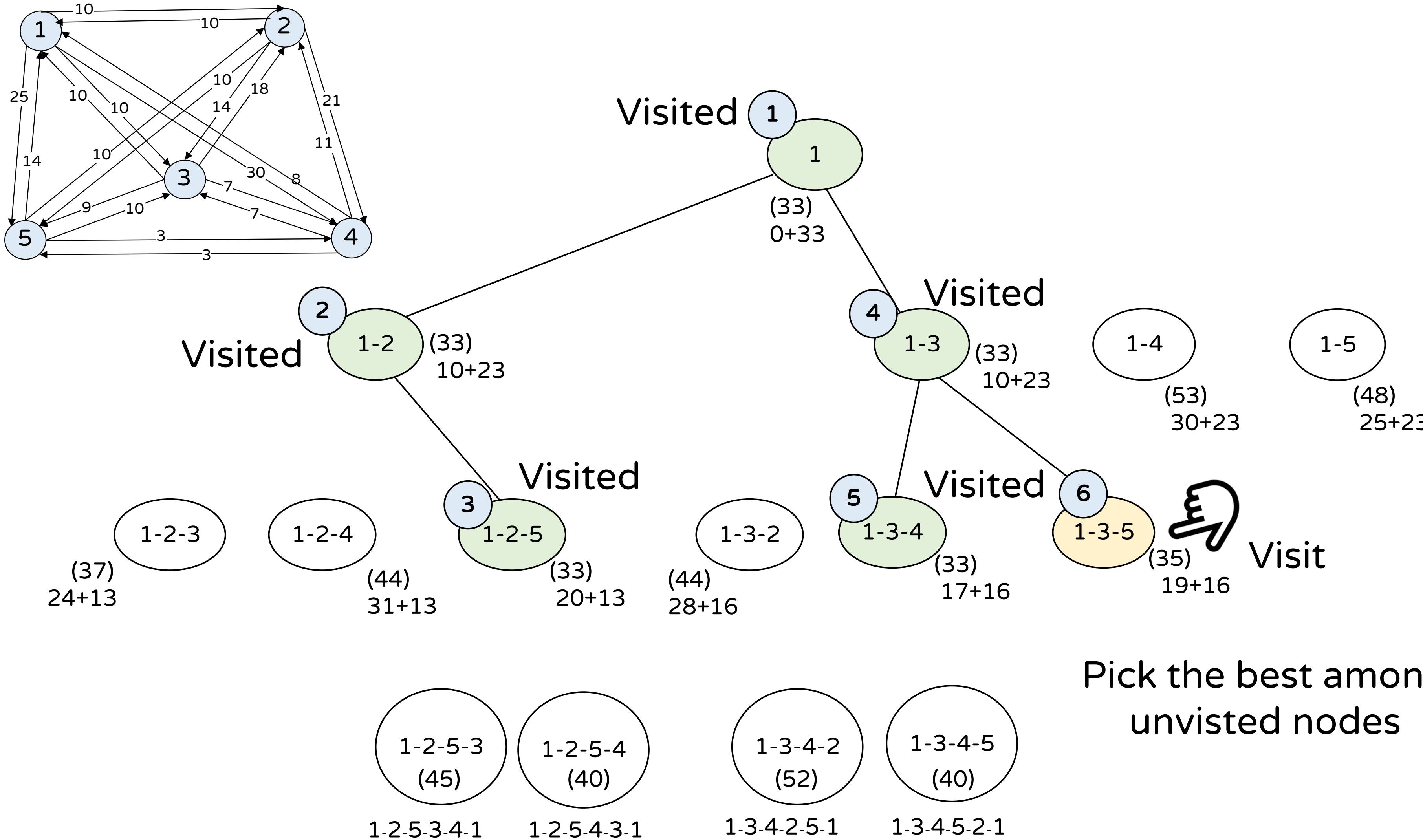
Pick the best among
unvisted nodes

Example for SSS (10)



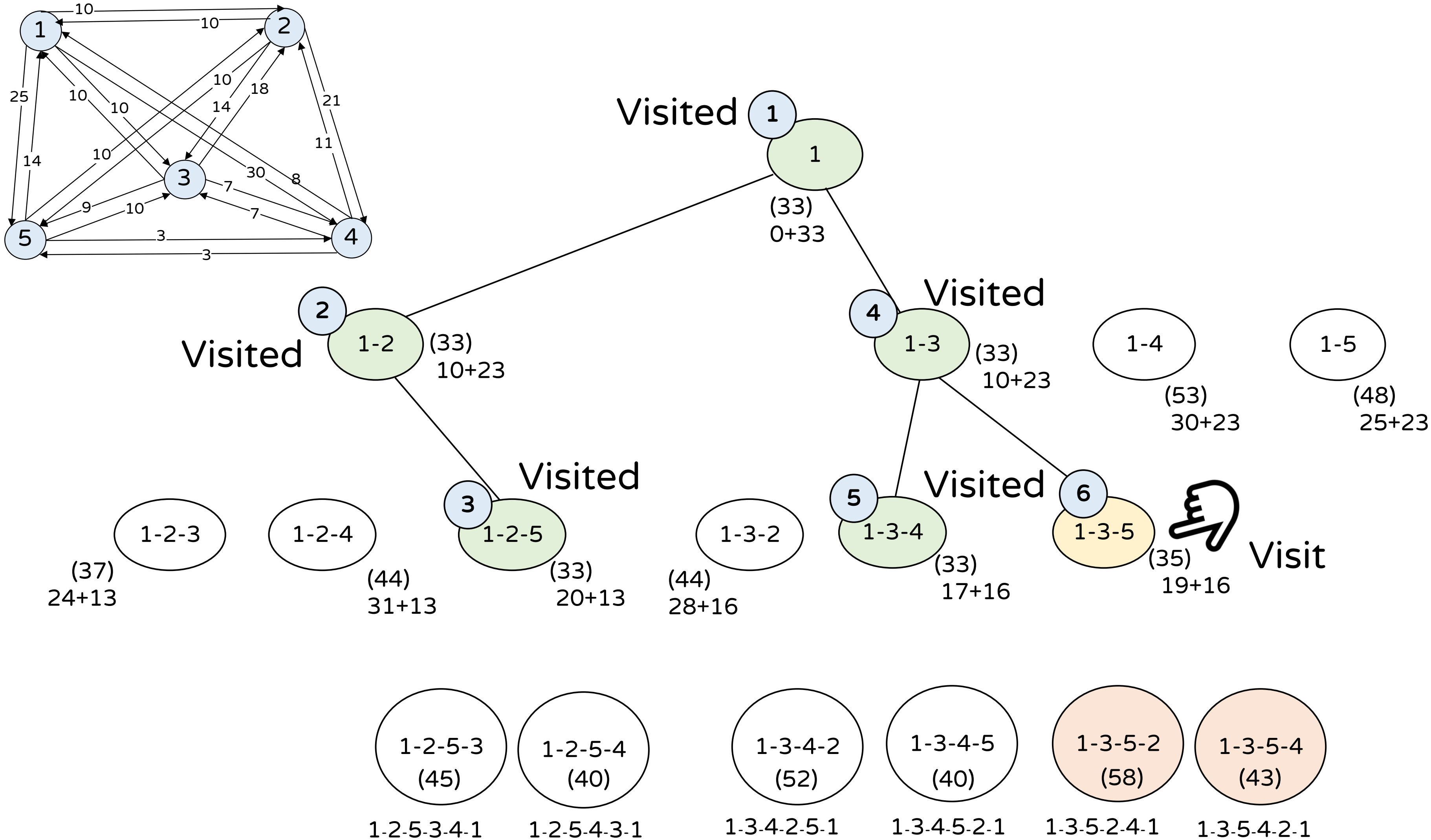
Update the values of its neighbors

Example for SSS (11)



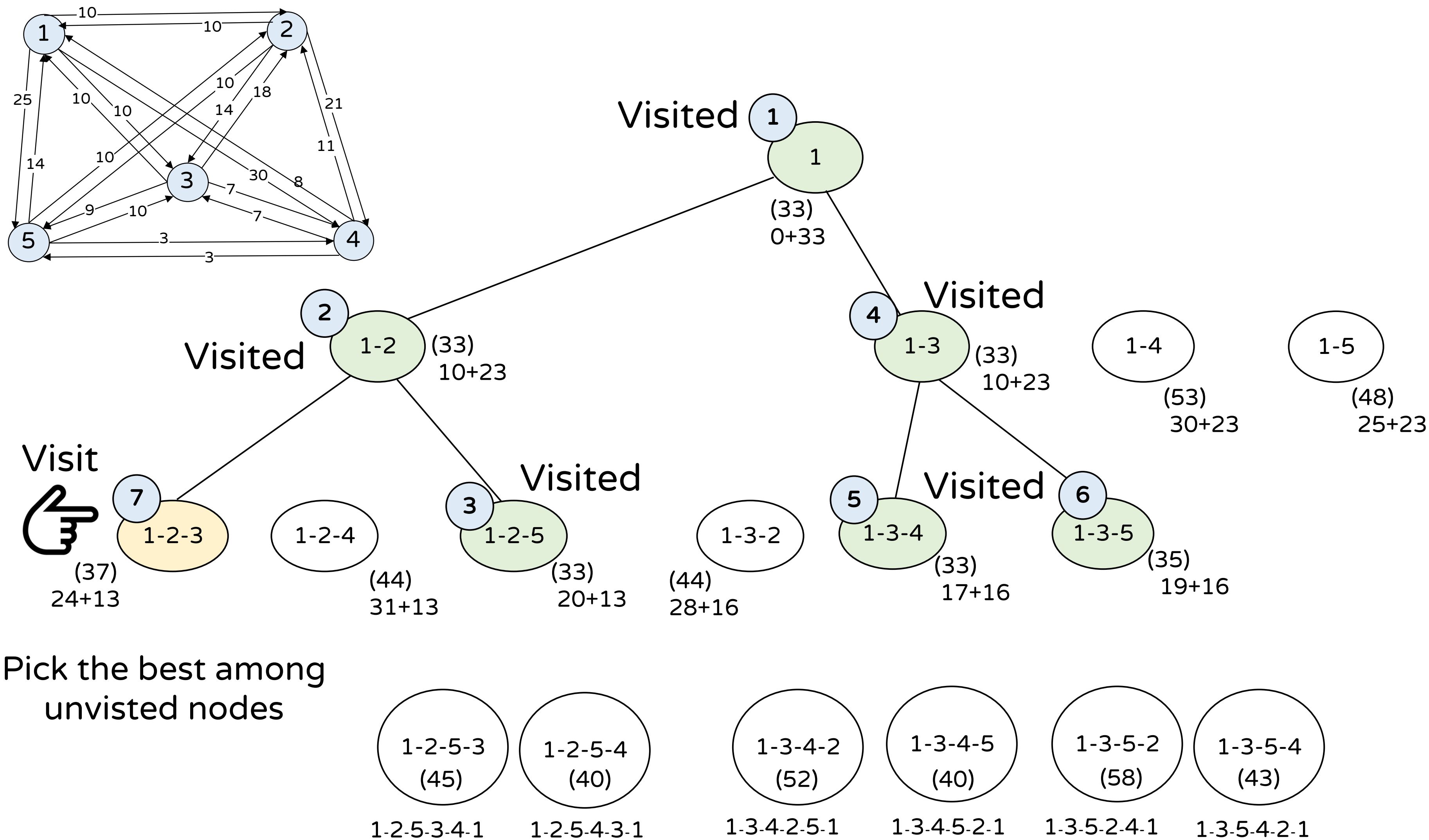
Pick the best among unvisted nodes

Example for SSS (12)

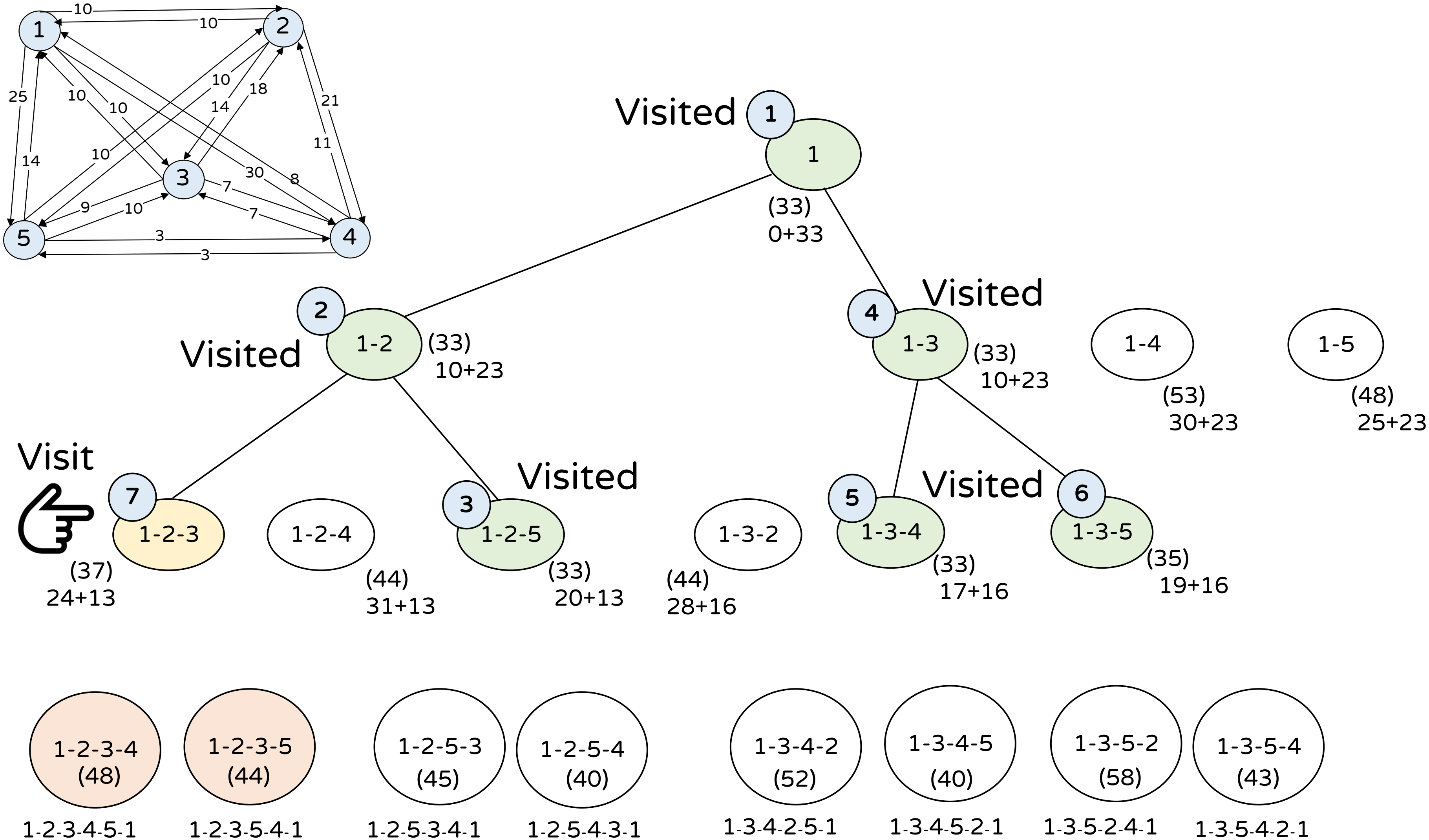


Update the values of its neighbors

Example for SSS (13)

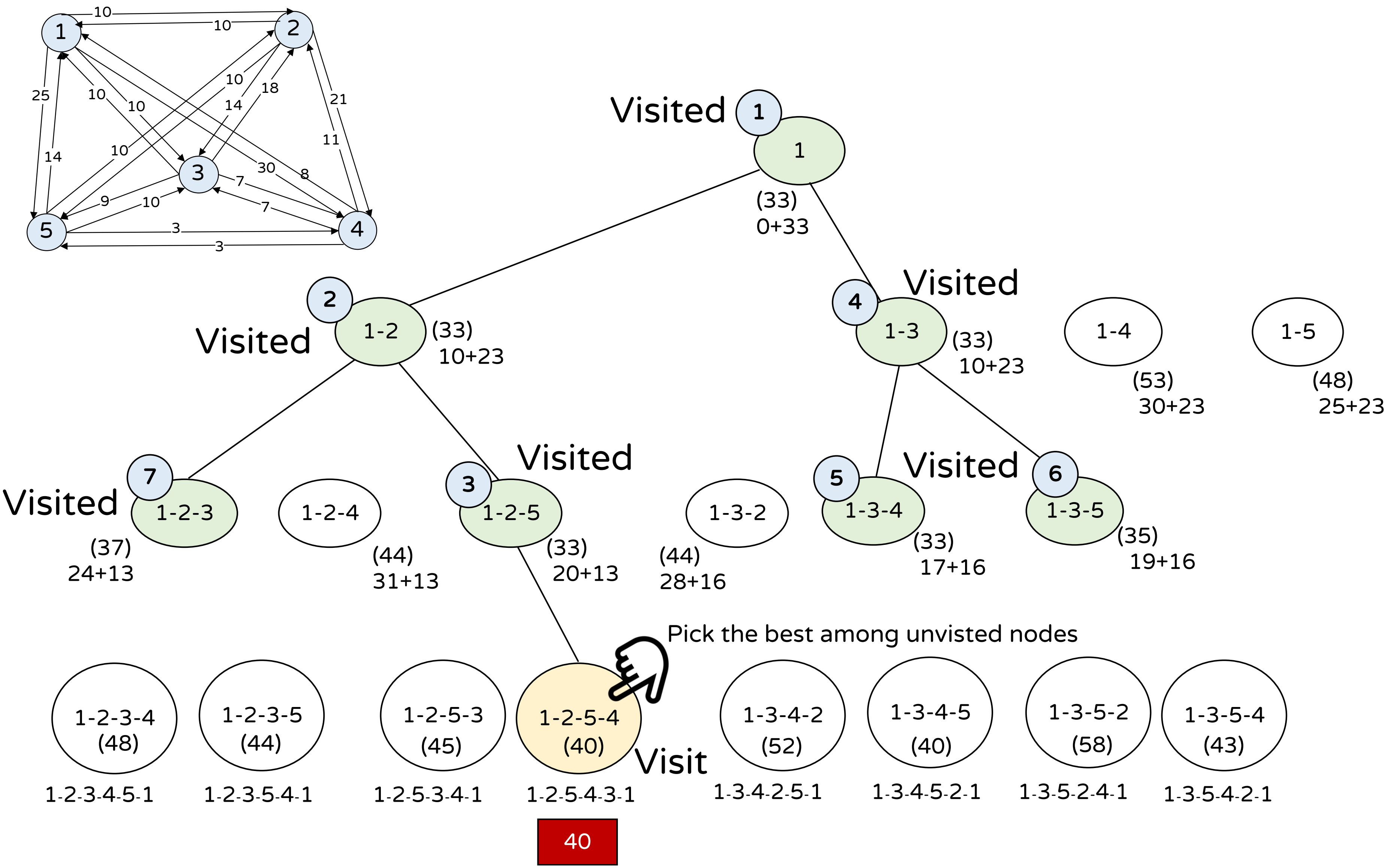


Example for SSS (13)



Update the values of its neighbors

Example for SSS (14)

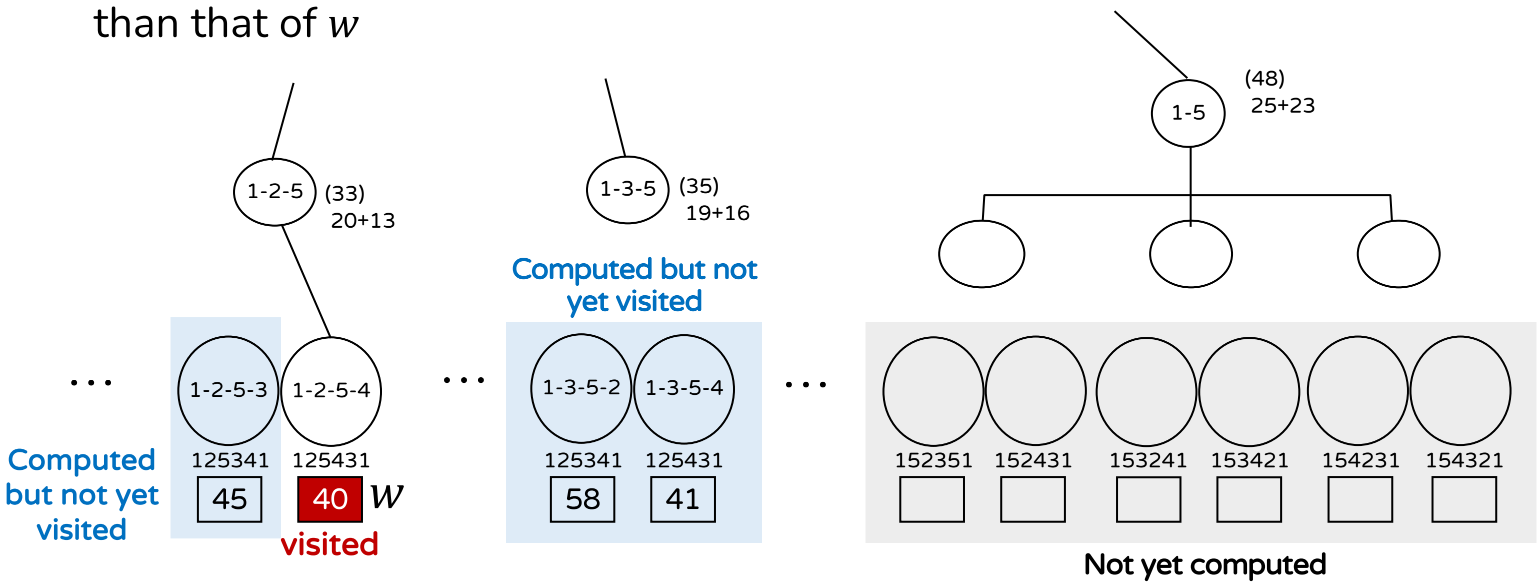


If it's a leaf, Finish!

Discussion

❑ Is it Okay that A* algorithm ends at a leaf? Yes!

- Let w denotes a leaf that A* algorithm meets first
- Two leaf groups
  - **Group 1.** leaf nodes already computed
    - The cost of w is less than that of others because it is reached first
  - **Group 2.** leaf nodes not yet computed
    - These nodes are not visited before $w \Rightarrow$ the cost of ancestors of the leaves is greater than that of w



What You Need To Know

❑ A* algorithm

- Aims to solve single-pair shortest path problem
- Best-first search algorithm with heuristics (guide)
  - Uses approximate heuristics – estimate remaining distances (or cost)

❑ A* algorithm for path finding

- Shows better performance than Dijkstra’s algorithm for finding single-pair shortest path
  - Optimality is guaranteed by the monotonicity of heuristics

❑ A* algorithm for state space search

- Apply A*’s strategy to a state space tree of a problem
  - Ends at a leaf which results in the optimal solution

Thank You