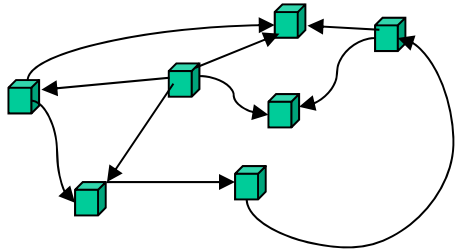
Graph Algorithms

Definitions



- Basic Concepts
  - A ***graph***: $G = (V, E)$

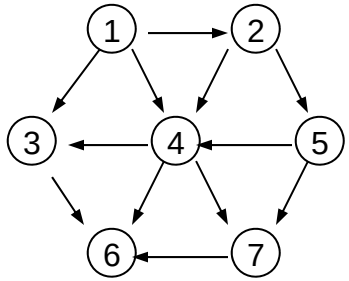
    where V is a set of ***vertices***, E is a set of ***edges***. Each edge is defined as a pair (v, w) with $v, w \in V$.
  - If the pairs that define the edges in E are ordered, G is a ***directed*** graph (or: digraph)
  - If $(v, w) \in E$, w is said to be ***adjacent*** to v. In an undirected graph, w is adjacent to v if and only if v is adjacent to w.
  - A value (***weight***) can be assigned to each edge.
  - A ***path*** in G is a sequence of connected vertices $w_1, w_2, \ldots, w_N \in V$, *i.e.* $(w_i, w_{i+1}) \in E$ $(1 \leq i < N)$. The length of the path is N-1.
  - A zero length path is a path from vertex v to itself (without the edge (v, v), which is called a ***loop***).

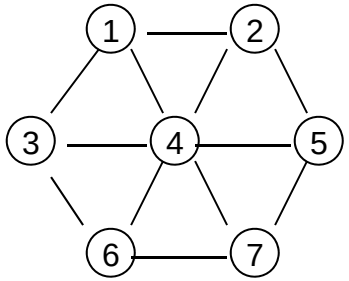
- A simple path is a path in which all vertices are distinct except the *1st* and the last vertices.
- A *cycle* is a path such that $w_1 = w_N$ $(N > 1)$. For an undirected graph, the edges in a cycle are required to be distinct.
- A directed graph is *acyclic* if it has no cycles, and it is called *DAG* (directed acyclic graph).
- A graph is *connected* if there is a path from every vertex to every other vertex.

  If a connected graph is also directed, the graph is said to be *strongly connected*.

  For a directed graph, if the connectivity can only be defined with undirected paths, it is said to be *weakly connected*.
- A complete graph is a graph in which there is an edge between every pair of vertices.
- Example: Airport connections.

– Graph Representations





– Adjacency Matrix

- A 2D $N \times N$ array

$A[i][j]$ = { 1: if there is edge (i, j)

0: otherwise

- Weights of the edge can also be stored in the matrix.
- Memory requirement: $\Theta(|V|^2)$
- A good representation for dense graphs, but inefficient if the graph is sparse.

directed

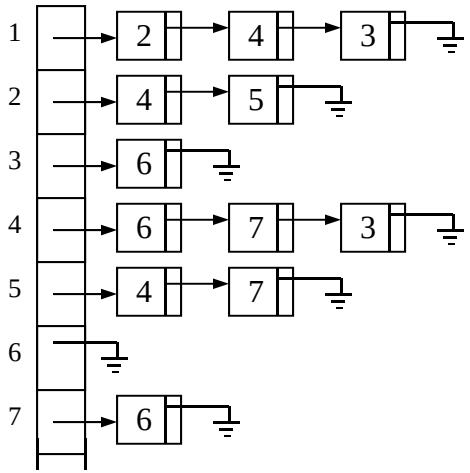
	1	2	3	4	5	6	7
1	0	1	1	1	0	0	0
2	0	0	0	1	1	0	0
3	0	0	0	0	0	1	0
4	0	0	1	0	0	1	1
5	0	0	0	1	0	0	1
6	0	0	0	0	0	0	0
7	0	0	0	0	0	1	0

undirected

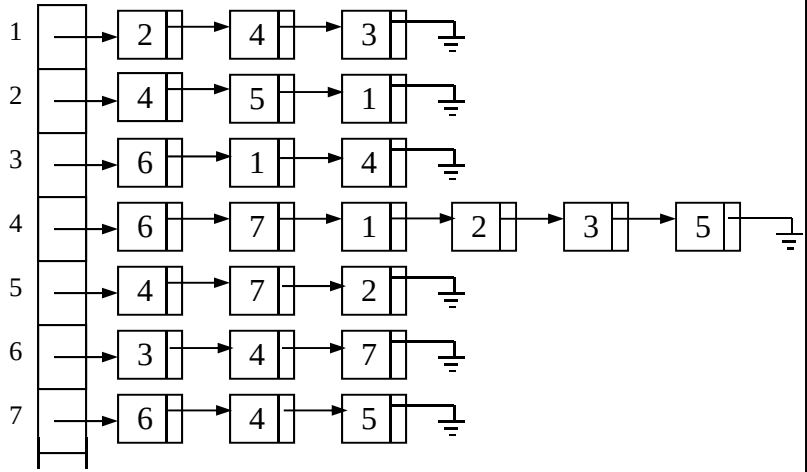
	1	2	3	4	5	6	7
1	0	1	1	1	0	0	0
2	1	0	0	1	1	0	0
3	1	0	0	1	0	1	0
4	1	1	1	0	1	1	1
5	0	1	0	1	0	0	1
6	0	0	1	1	0	0	1
7	0	0	0	1	1	1	0

– Adjacency List
  - A 1D array of linked lists.
    $A[u]$ is a list of all the vertices that are adjacent to u.
  - Memory cost: $\Theta(|V| + |E|)$
  - Weights of the edges can be stored in the nodes of the lists.
  - For undirected graph, each edge appears in two lists (double memory cost)
  - Hashing can be used to map the names of the vertices to the vertex nodes.

directed

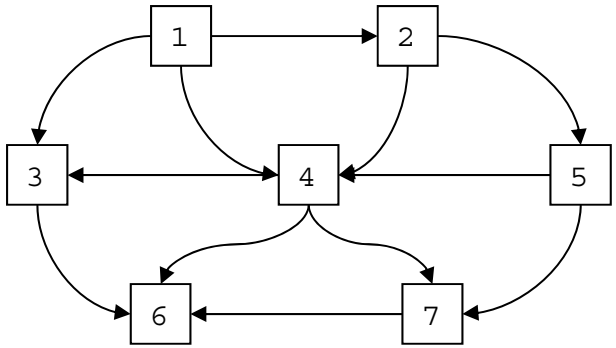


undirected



Topological Sort

- A *topological Sort* is an ordering of vertices in a directed acyclic graph (DAG) such that if there is a path from v_i to v_j, then v_j appears after v_i in the ordering.
- *Topological order* is not possible if there is a cycle in the graph.
- Topological ordering is not unique.
- Example
  - course prerequisite graph



1 2 5 4 3 7 6

or

1 2 5 4 7 3 6

- A simple algorithm
  - *indegree* of vertex v: the number of edges (u, v)
  - topological sorting algorithm:

    a.) compute the indegrees of all vertices

    b.) remove a vertex with *0* indegree and its associated edges

    c.) repeat b.) until all vertices are sorted.

example with directed edge matrix

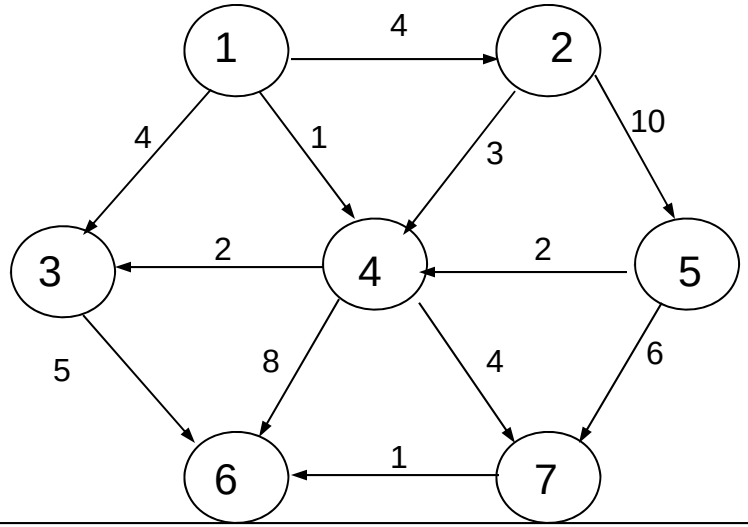
Shortest-Path Algorithms

- The problem
  - Input: a weighted graph

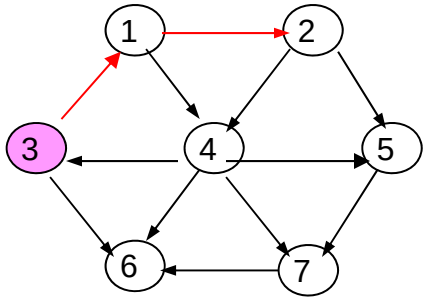
    i.e. each edge (v_i, v_j) has a cost C_{ij}
  - the cost of path $v_1\ v_2 \ldots v_N$ is $\sum_{i=1}^{N-1} C_{i,i+1}$, also called ***weighted path length.***
  - ***unweighted path length***: $N - 1$
  - single-source shortest path problem:

    given as input a weighted graph $G = (V, E)$ and a distinguished vertex, S, find the shortest weighted path from S to every other vertex in G
  - example

    $C(v_1, v_6) = 6$

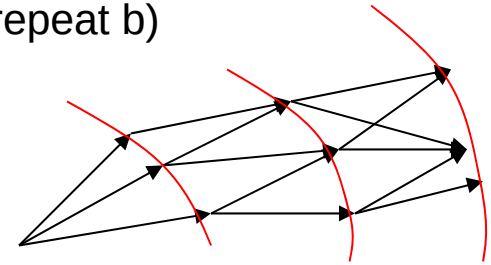


– The single destination problem: find the shortest path from source s to a given destination vertex t.
  - Currently, there are no algorithms that can find the single destination shortest path any faster (by more than a constant factor) than the shortest paths to all vertices.

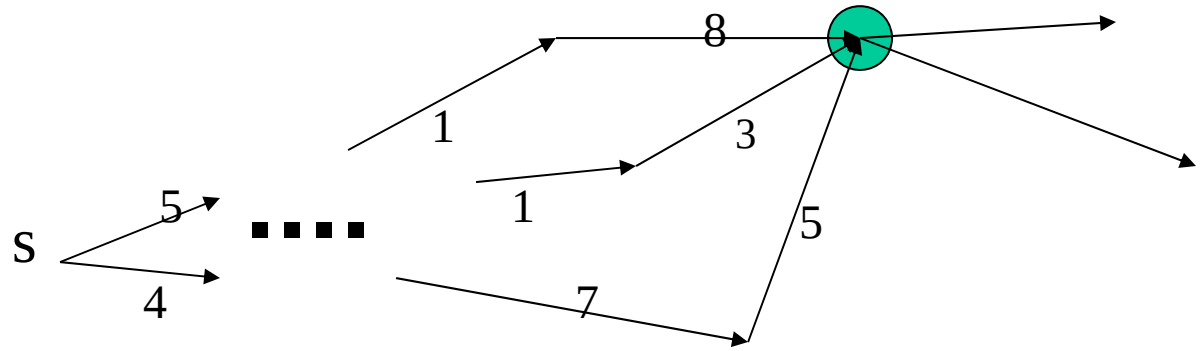


– Unweighted shortest paths
  - A special case of weighted shortest path
  - data structure (table)

  - basic idea
    a) Starting from vertex s, let `s.dis=0` and `x=s`
    b) Mark `dist=x.dist+1` for all vertices that are adjacent to x.
    c) For each of the vertices, x, that are just marked, repeat b)
    d) repeat c) until all vertices are marked.
  - Breadth-first search



– Weighted shortest path
  - Mark vertices as known or unknown



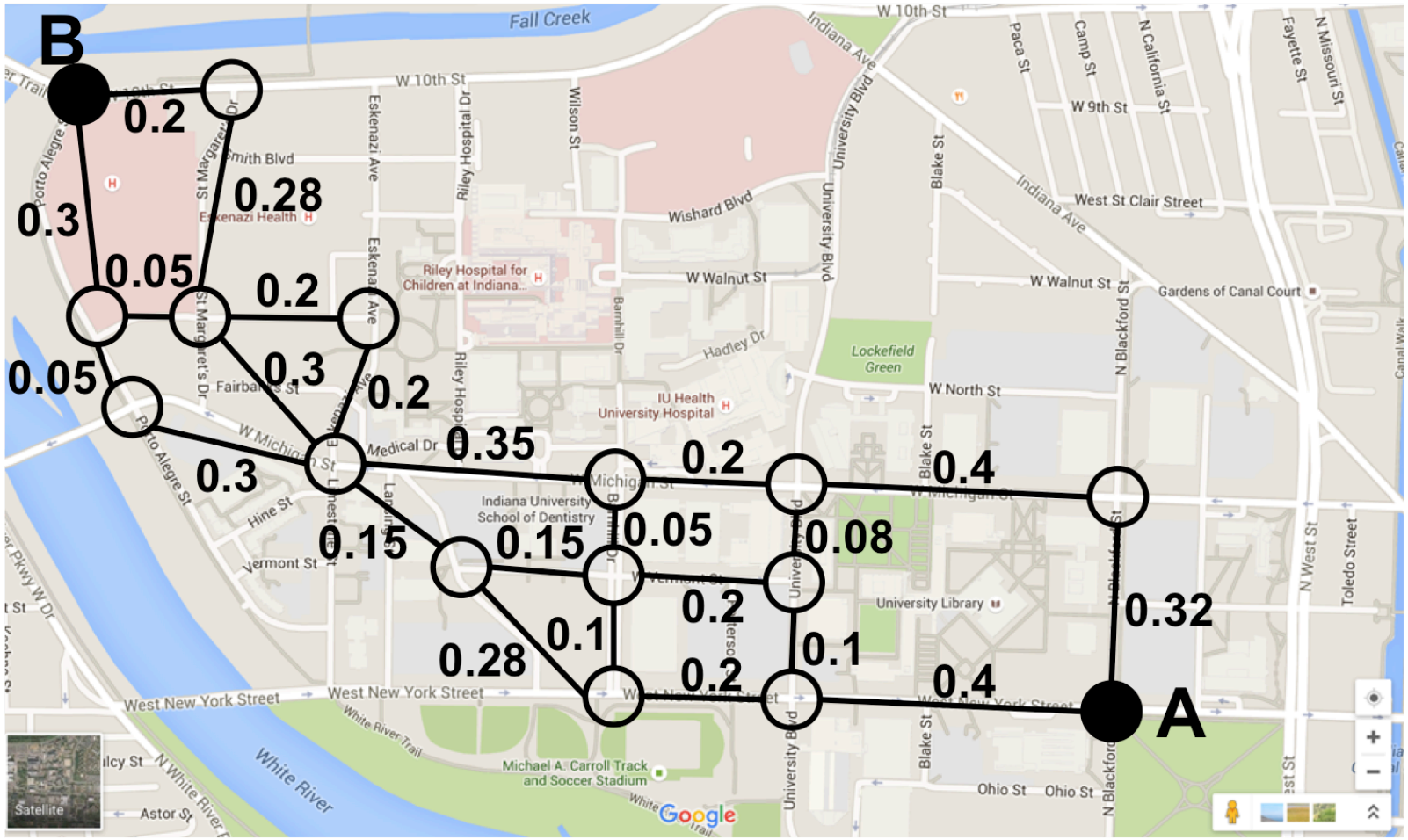
– Greedy algorithm
  - greedy strategy: taking currently best solution at each step.
  - similar idea to the unweighted algorithm.

    At each stage, find the shortest unknown distance vertex v, and update its adjacent vertices: `w.dist=v.dist+c(v,w)`

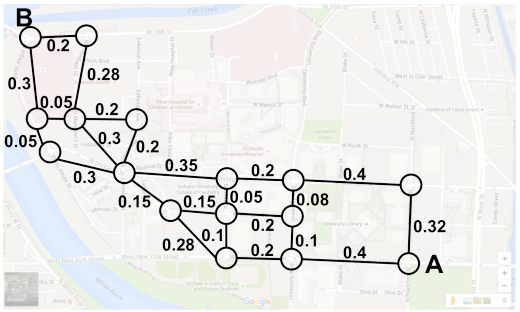
    if it is an improvement over the previous value in w.

Problem?

You want to travel from point A to point B, following the shortest distance.



Dijkstra's algorithm



1. Assign to every node an initial tentative distance, zero for the initial node and "infinity" for all other nodes.

2. Set the initial node as current. Mark all other nodes unvisited. Create a set of all the unvisited nodes.

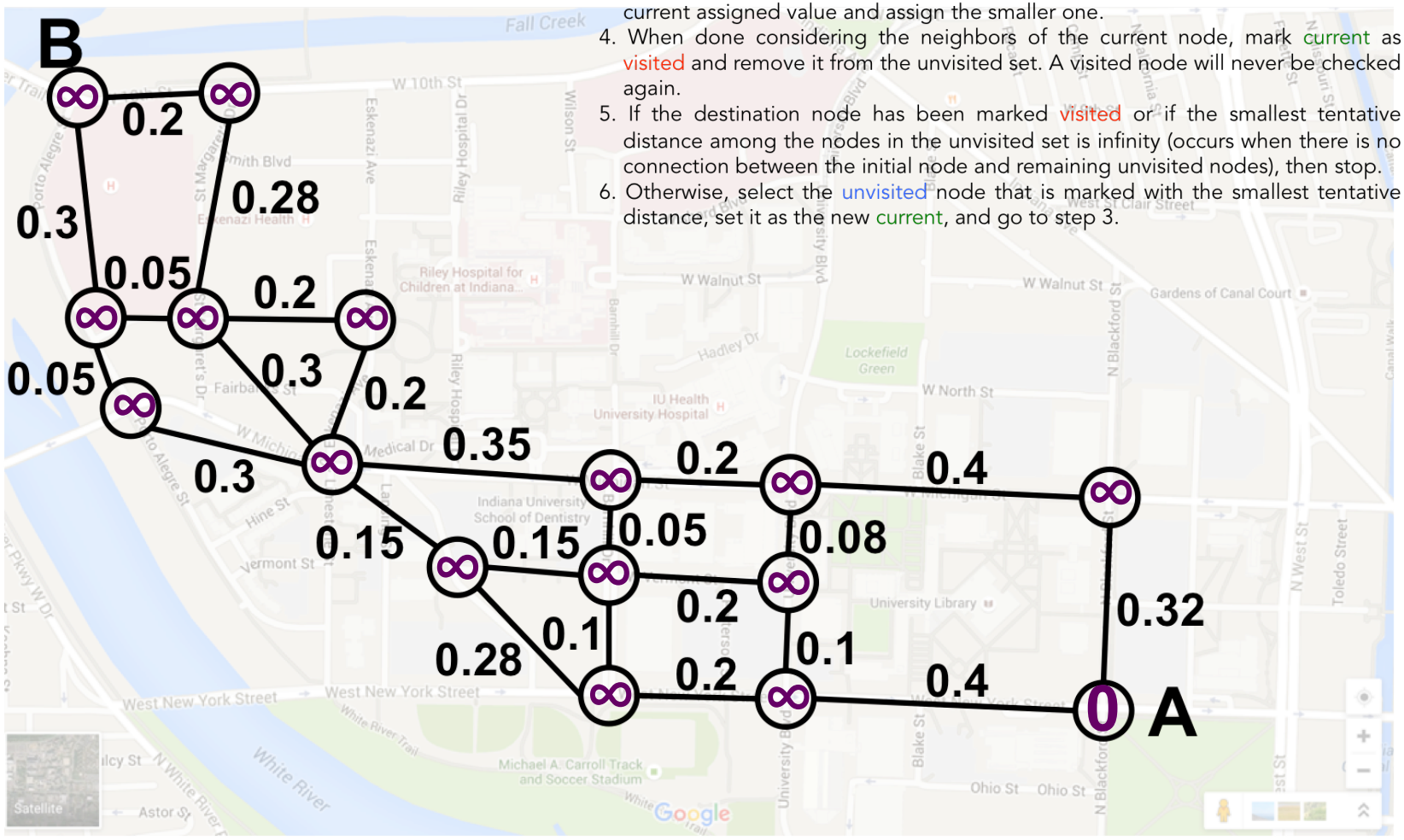
3. For the current node, consider all of its unvisited neighbors and calculate their tentative distances. Compare the newly calculated tentative distance to the current assigned value and assign the smaller one.

4. When done considering the neighbors of the current node, mark current as visited and remove it from the unvisited set. A visited node will never be checked again.

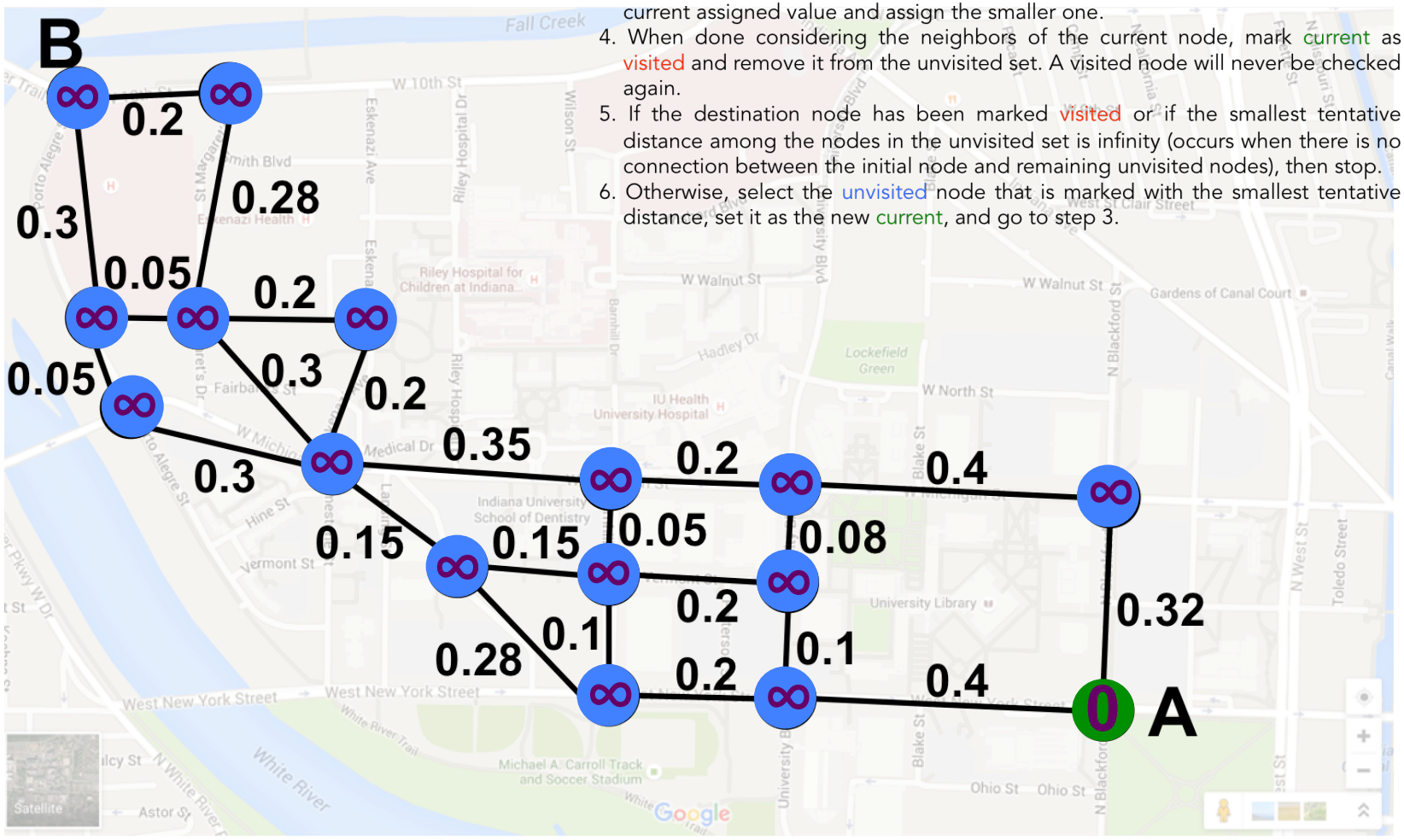
5. If the destination node has been marked visited or if the smallest tentative distance among the nodes in the unvisited set is infinity (occurs when there is no connection between the initial node and remaining unvisited nodes), then stop.

6. Otherwise, select the unvisited node that is marked with the smallest tentative distance, set it as the new current, and go to step 3.

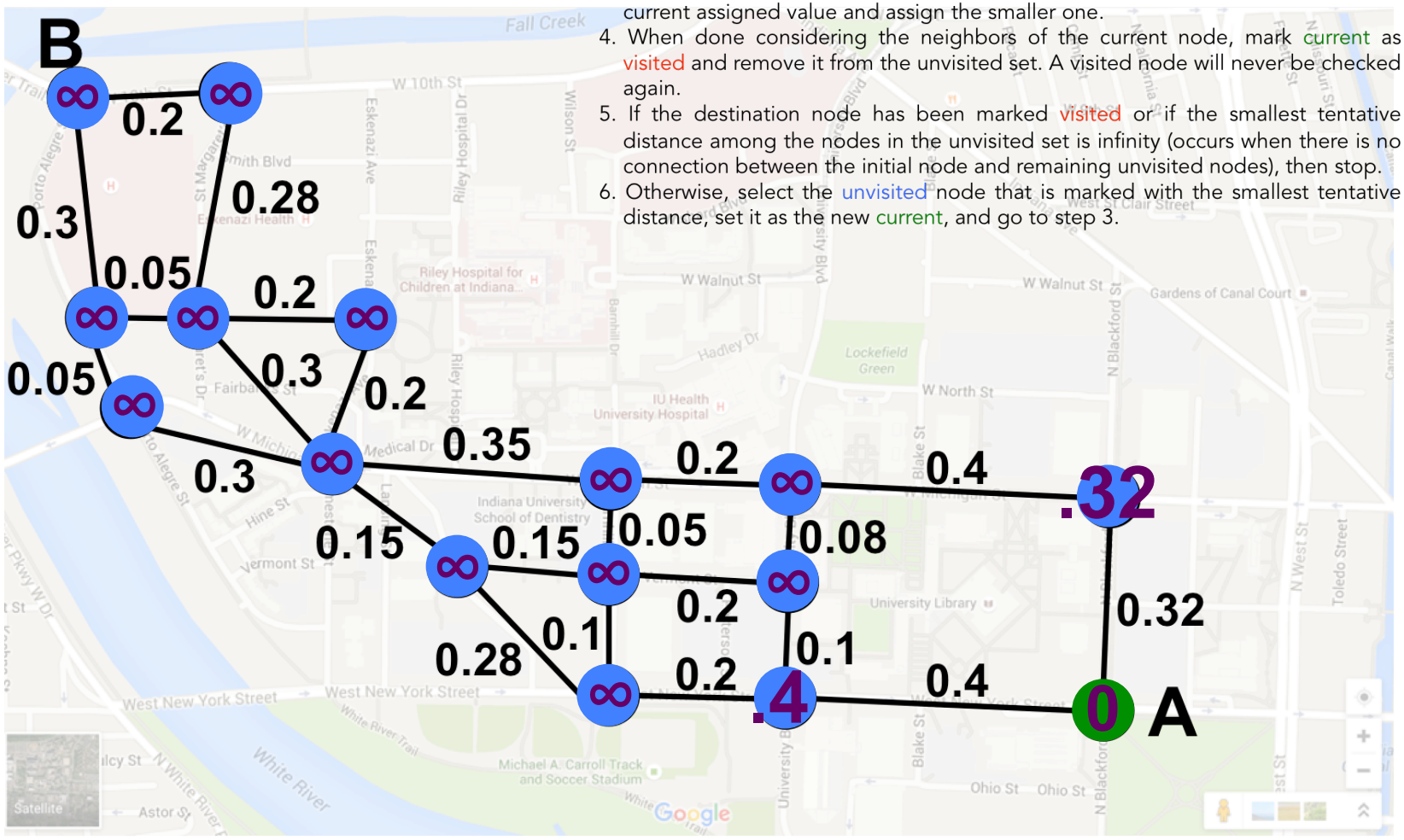
1. Assign to every node an initial tentative distance, zero for the initial node and "infinity" for all other nodes.
2. Set the initial node as current. Mark all other nodes unvisited. Create a set of all the unvisited nodes.
3. For the current node, consider all of its unvisited neighbors and calculate their tentative distances. Compare the newly calculated tentative distance to the current assigned value and assign the smaller one.
4. When done considering the neighbors of the current node, mark current as visited and remove it from the unvisited set. A visited node will never be checked again.
5. If the destination node has been marked visited or if the smallest tentative distance among the nodes in the unvisited set is infinity (occurs when there is no connection between the initial node and remaining unvisited nodes), then stop.
6. Otherwise, select the unvisited node that is marked with the smallest tentative distance, set it as the new current, and go to step 3.



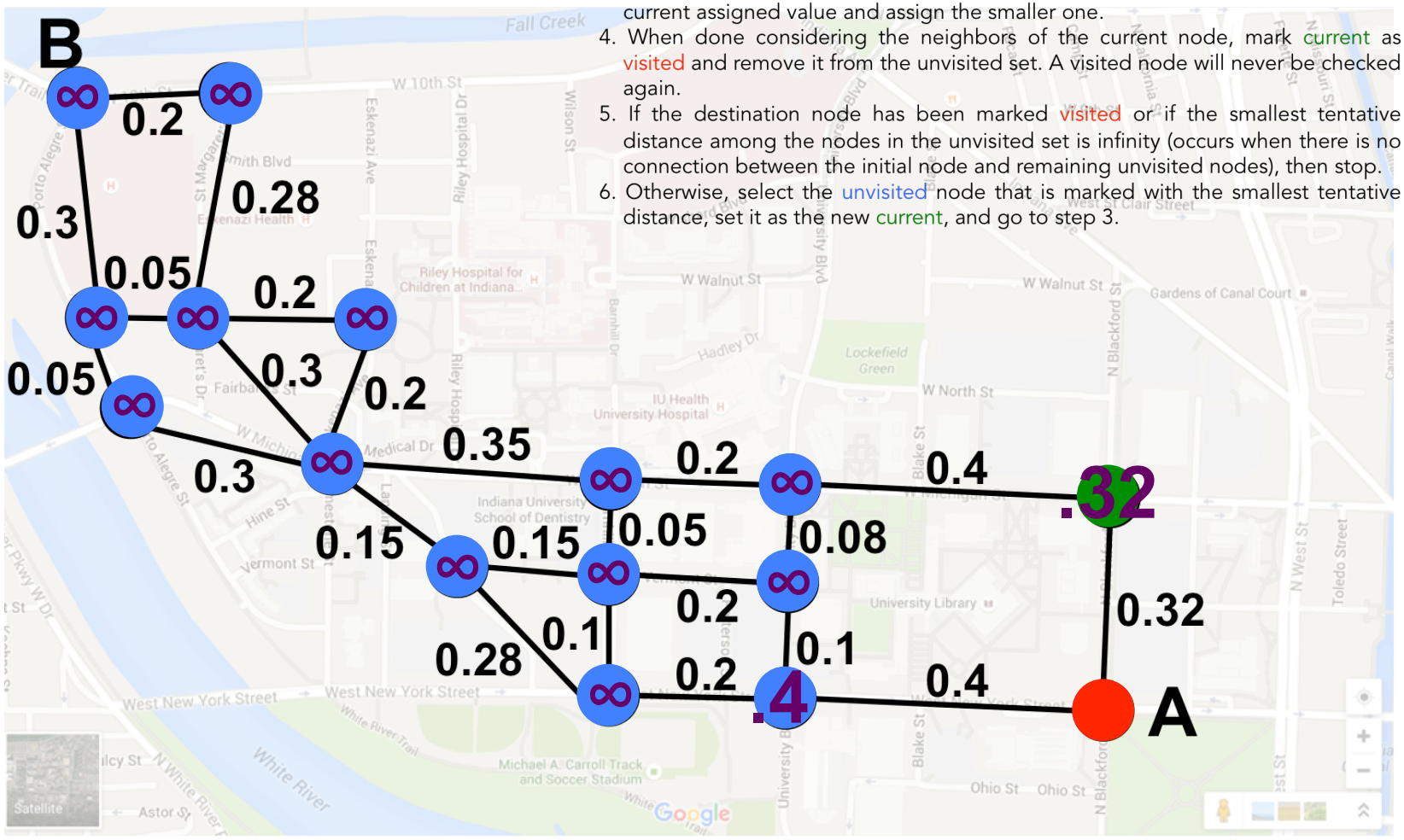
1. Assign to every node an initial tentative distance, zero for the initial node and "infinity" for all other nodes.
2. Set the initial node as current. Mark all other nodes unvisited. Create a set of all the unvisited nodes.
3. For the current node, consider all of its unvisited neighbors and calculate their tentative distances. Compare the newly calculated tentative distance to the current assigned value and assign the smaller one.
4. When done considering the neighbors of the current node, mark current as visited and remove it from the unvisited set. A visited node will never be checked again.
5. If the destination node has been marked visited or if the smallest tentative distance among the nodes in the unvisited set is infinity (occurs when there is no connection between the initial node and remaining unvisited nodes), then stop.
6. Otherwise, select the unvisited node that is marked with the smallest tentative distance, set it as the new current, and go to step 3.



1. Assign to every node an initial tentative distance, zero for the initial node and "infinity" for all other nodes.
2. Set the initial node as current. Mark all other nodes unvisited. Create a set of all the unvisited nodes.
3. For the current node, consider all of its unvisited neighbors and calculate their tentative distances. Compare the newly calculated tentative distance to the current assigned value and assign the smaller one.
4. When done considering the neighbors of the current node, mark current as visited and remove it from the unvisited set. A visited node will never be checked again.
5. If the destination node has been marked visited or if the smallest tentative distance among the nodes in the unvisited set is infinity (occurs when there is no connection between the initial node and remaining unvisited nodes), then stop.
6. Otherwise, select the unvisited node that is marked with the smallest tentative distance, set it as the new current, and go to step 3.

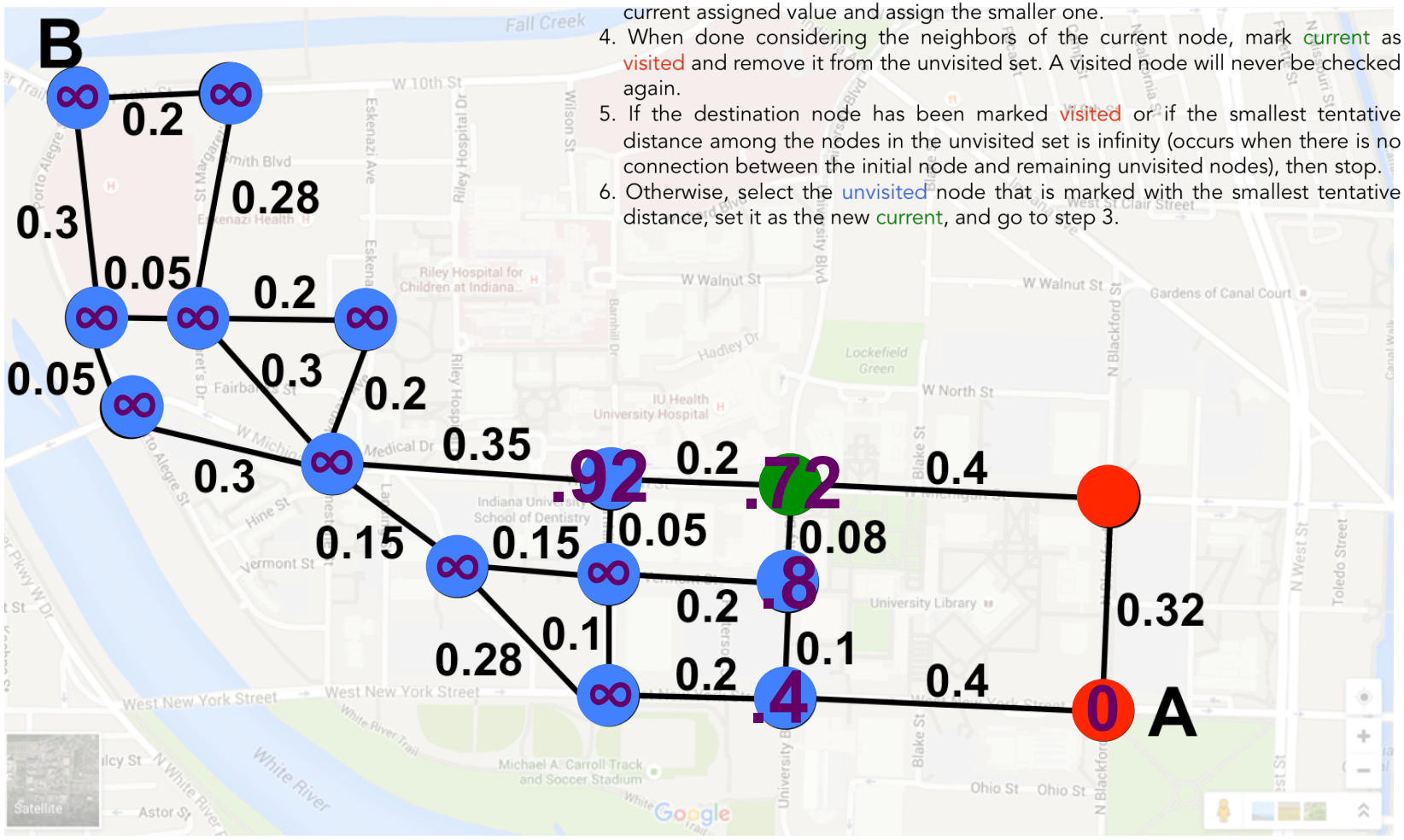


1. Assign to every node an initial tentative distance, zero for the initial node and "infinity" for all other nodes.
2. Set the initial node as current. Mark all other nodes unvisited. Create a set of all the unvisited nodes.
3. For the current node, consider all of its unvisited neighbors and calculate their tentative distances. Compare the newly calculated tentative distance to the current assigned value and assign the smaller one.
4. When done considering the neighbors of the current node, mark current as visited and remove it from the unvisited set. A visited node will never be checked again.
5. If the destination node has been marked visited or if the smallest tentative distance among the nodes in the unvisited set is infinity (occurs when there is no connection between the initial node and remaining unvisited nodes), then stop.
6. Otherwise, select the unvisited node that is marked with the smallest tentative distance, set it as the new current, and go to step 3.



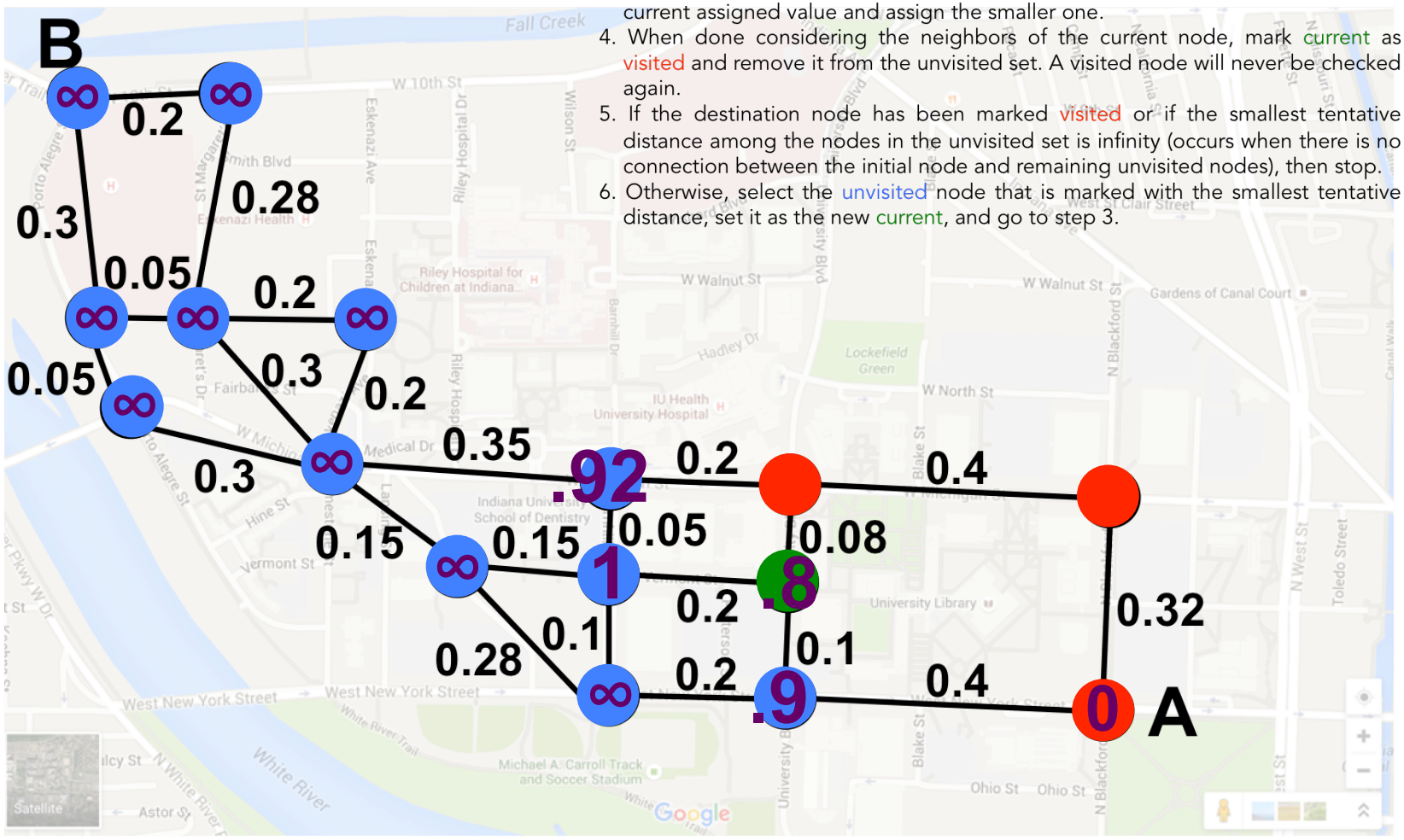
STOP & THINK

1. Assign to every node an initial tentative distance, zero for the initial node and "infinity" for all other nodes.
2. Set the initial node as current. Mark all other nodes unvisited. Create a set of all the unvisited nodes.
3. For the current node, consider all of its unvisited neighbors and calculate their tentative distances. Compare the newly calculated tentative distance to the current assigned value and assign the smaller one.
4. When done considering the neighbors of the current node, mark current as visited and remove it from the unvisited set. A visited node will never be checked again.
5. If the destination node has been marked visited or if the smallest tentative distance among the nodes in the unvisited set is infinity (occurs when there is no connection between the initial node and remaining unvisited nodes), then stop.
6. Otherwise, select the unvisited node that is marked with the smallest tentative distance, set it as the new current, and go to step 3.

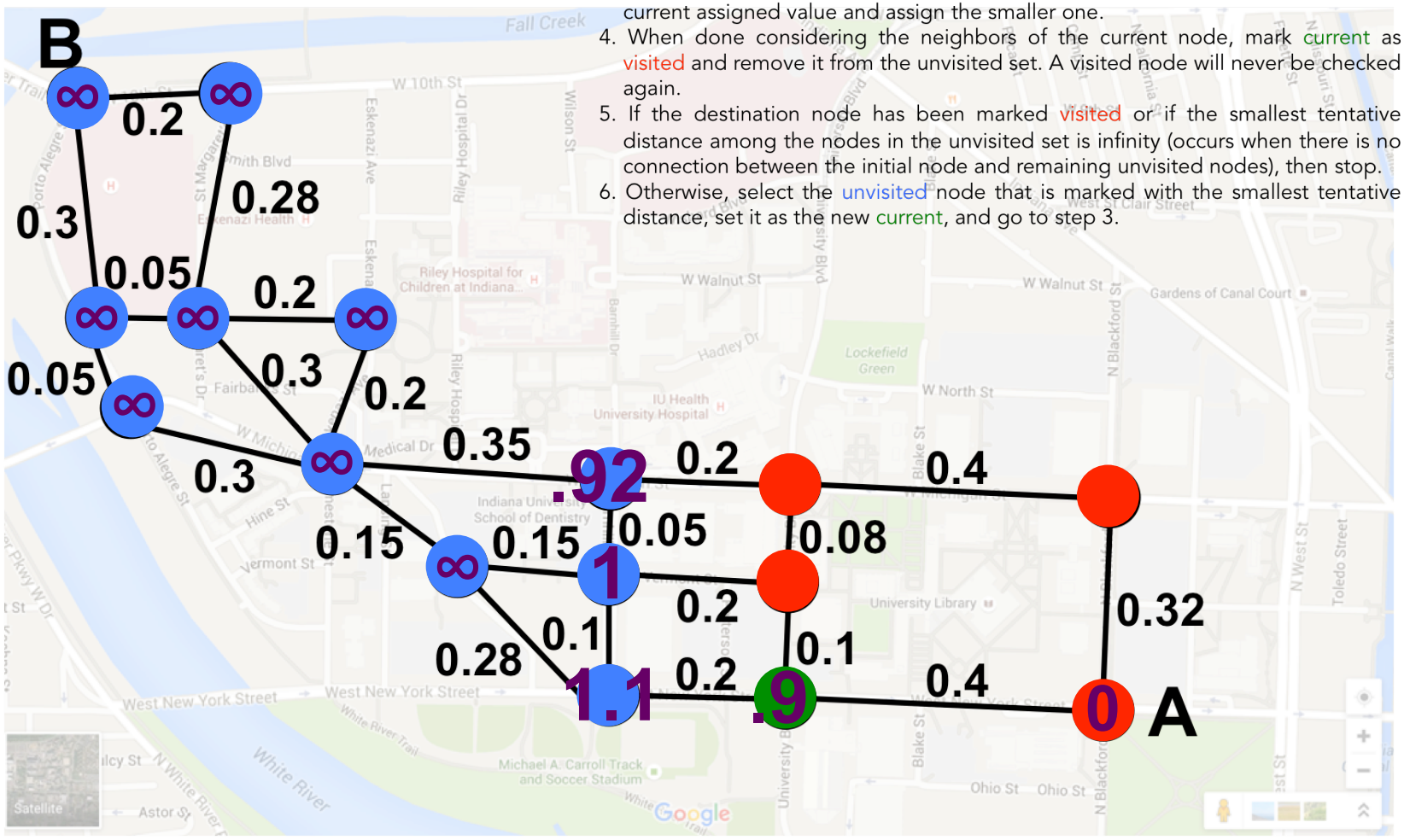


Something starts going wrong

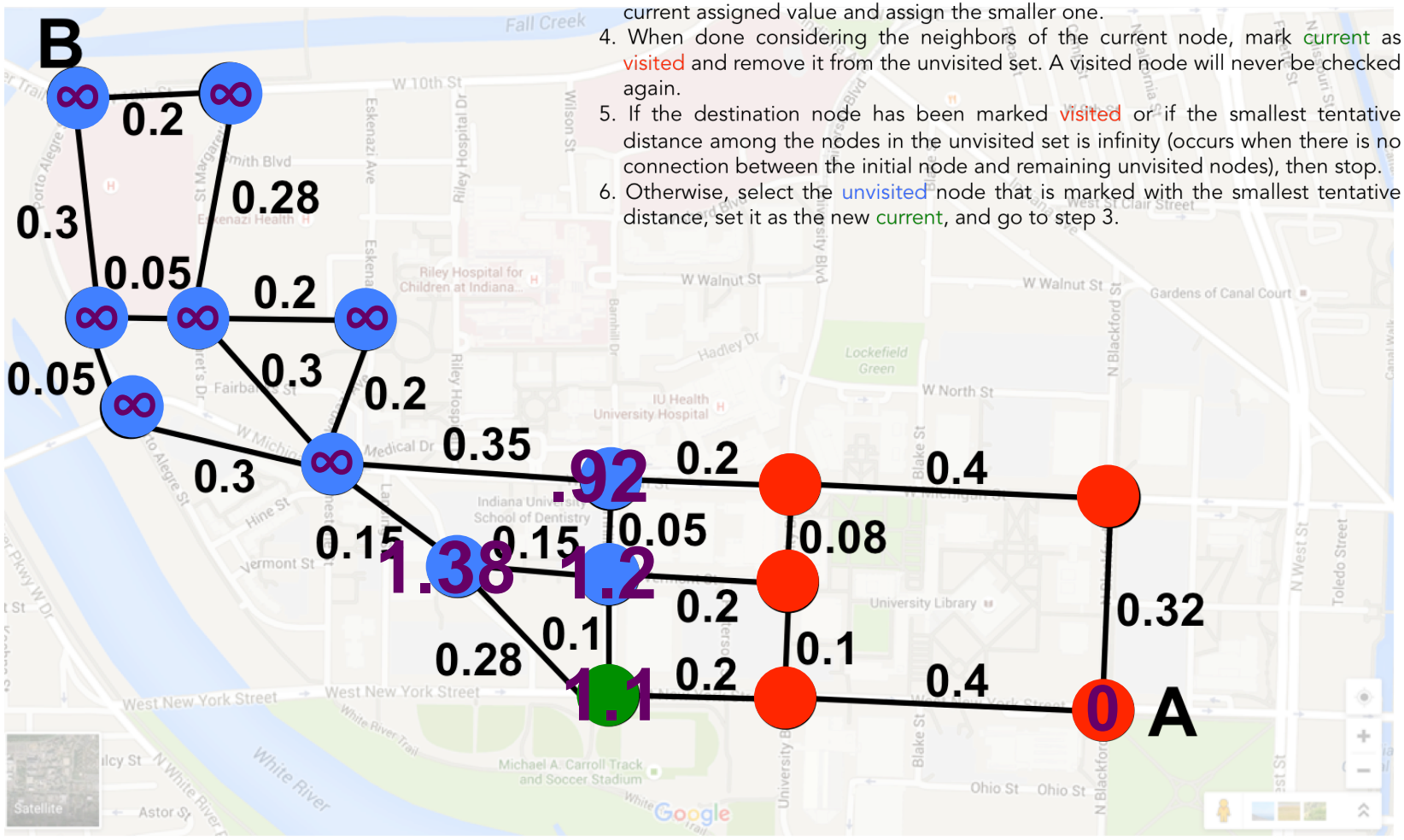
1. Assign to every node an initial tentative distance, zero for the initial node and "infinity" for all other nodes.
2. Set the initial node as current. Mark all other nodes unvisited. Create a set of all the unvisited nodes.
3. For the current node, consider all of its unvisited neighbors and calculate their tentative distances. Compare the newly calculated tentative distance to the current assigned value and assign the smaller one.
4. When done considering the neighbors of the current node, mark current as visited and remove it from the unvisited set. A visited node will never be checked again.
5. If the destination node has been marked visited or if the smallest tentative distance among the nodes in the unvisited set is infinity (occurs when there is no connection between the initial node and remaining unvisited nodes), then stop.
6. Otherwise, select the unvisited node that is marked with the smallest tentative distance, set it as the new current, and go to step 3.



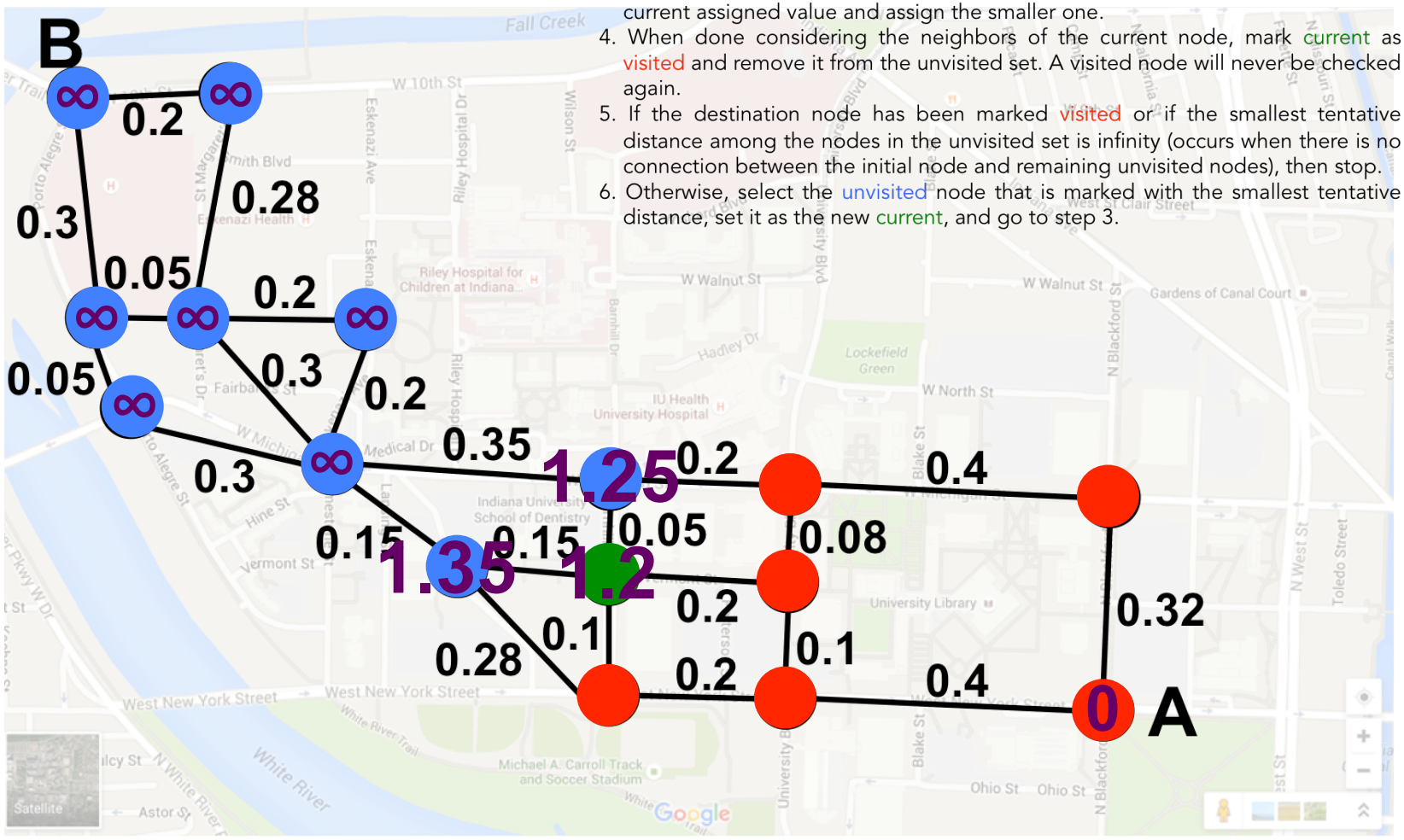
1. Assign to every node an initial tentative distance, zero for the initial node and "infinity" for all other nodes.
2. Set the initial node as current. Mark all other nodes unvisited. Create a set of all the unvisited nodes.
3. For the current node, consider all of its unvisited neighbors and calculate their tentative distances. Compare the newly calculated tentative distance to the current assigned value and assign the smaller one.
4. When done considering the neighbors of the current node, mark current as visited and remove it from the unvisited set. A visited node will never be checked again.
5. If the destination node has been marked visited or if the smallest tentative distance among the nodes in the unvisited set is infinity (occurs when there is no connection between the initial node and remaining unvisited nodes), then stop.
6. Otherwise, select the unvisited node that is marked with the smallest tentative distance, set it as the new current, and go to step 3.



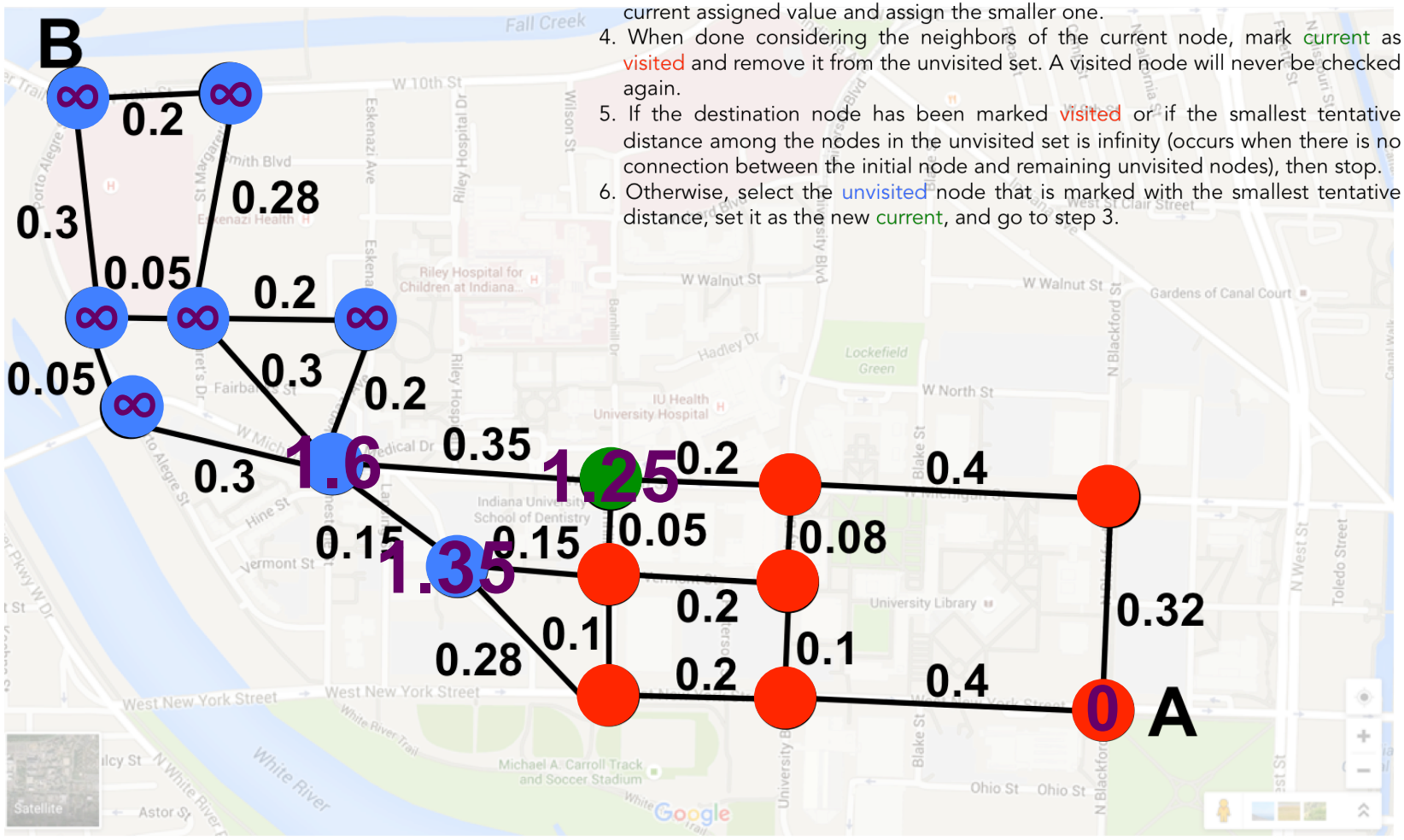
1. Assign to every node an initial tentative distance, zero for the initial node and "infinity" for all other nodes.
2. Set the initial node as current. Mark all other nodes unvisited. Create a set of all the unvisited nodes.
3. For the current node, consider all of its unvisited neighbors and calculate their tentative distances. Compare the newly calculated tentative distance to the current assigned value and assign the smaller one.
4. When done considering the neighbors of the current node, mark current as visited and remove it from the unvisited set. A visited node will never be checked again.
5. If the destination node has been marked visited or if the smallest tentative distance among the nodes in the unvisited set is infinity (occurs when there is no connection between the initial node and remaining unvisited nodes), then stop.
6. Otherwise, select the unvisited node that is marked with the smallest tentative distance, set it as the new current, and go to step 3.



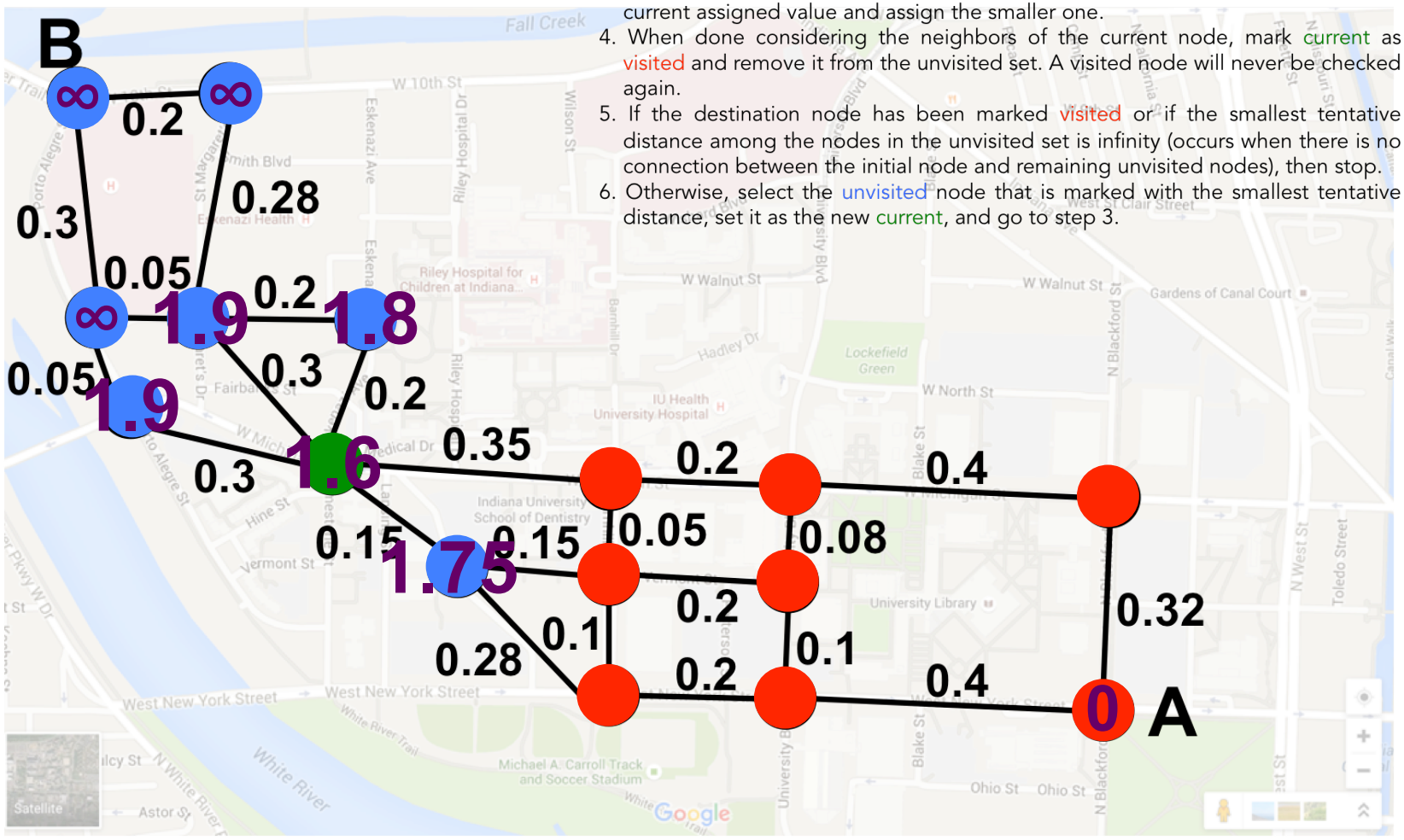
1. Assign to every node an initial tentative distance, zero for the initial node and "infinity" for all other nodes.
2. Set the initial node as current. Mark all other nodes unvisited. Create a set of all the unvisited nodes.
3. For the current node, consider all of its unvisited neighbors and calculate their tentative distances. Compare the newly calculated tentative distance to the current assigned value and assign the smaller one.
4. When done considering the neighbors of the current node, mark current as visited and remove it from the unvisited set. A visited node will never be checked again.
5. If the destination node has been marked visited or if the smallest tentative distance among the nodes in the unvisited set is infinity (occurs when there is no connection between the initial node and remaining unvisited nodes), then stop.
6. Otherwise, select the unvisited node that is marked with the smallest tentative distance, set it as the new current, and go to step 3.



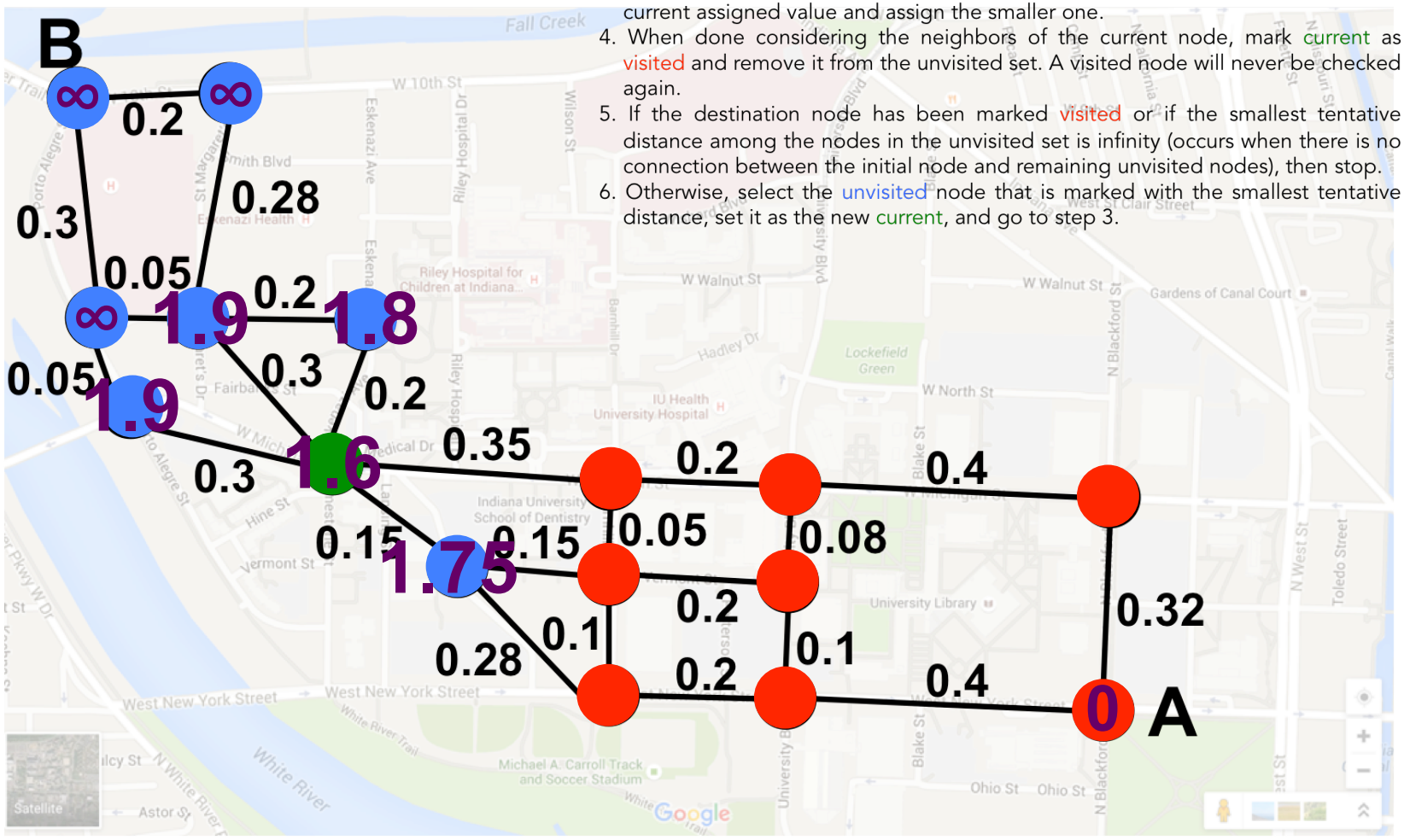
1. Assign to every node an initial tentative distance, zero for the initial node and "infinity" for all other nodes.
2. Set the initial node as current. Mark all other nodes unvisited. Create a set of all the unvisited nodes.
3. For the current node, consider all of its unvisited neighbors and calculate their tentative distances. Compare the newly calculated tentative distance to the current assigned value and assign the smaller one.
4. When done considering the neighbors of the current node, mark current as visited and remove it from the unvisited set. A visited node will never be checked again.
5. If the destination node has been marked visited or if the smallest tentative distance among the nodes in the unvisited set is infinity (occurs when there is no connection between the initial node and remaining unvisited nodes), then stop.
6. Otherwise, select the unvisited node that is marked with the smallest tentative distance, set it as the new current, and go to step 3.



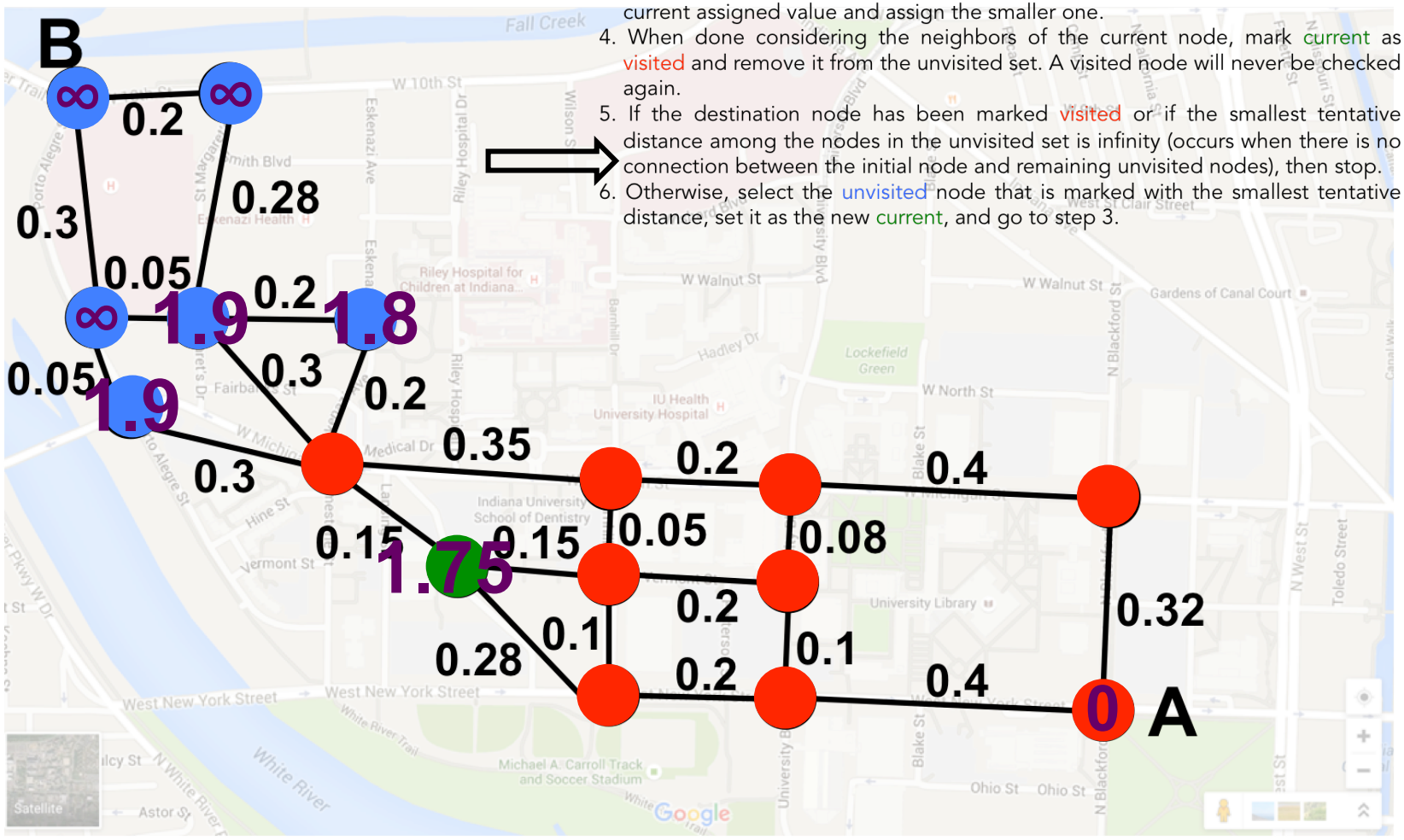
1. Assign to every node an initial tentative distance, zero for the initial node and "infinity" for all other nodes.
2. Set the initial node as current. Mark all other nodes unvisited. Create a set of all the unvisited nodes.
3. For the current node, consider all of its unvisited neighbors and calculate their tentative distances. Compare the newly calculated tentative distance to the current assigned value and assign the smaller one.
4. When done considering the neighbors of the current node, mark current as visited and remove it from the unvisited set. A visited node will never be checked again.
5. If the destination node has been marked visited or if the smallest tentative distance among the nodes in the unvisited set is infinity (occurs when there is no connection between the initial node and remaining unvisited nodes), then stop.
6. Otherwise, select the unvisited node that is marked with the smallest tentative distance, set it as the new current, and go to step 3.



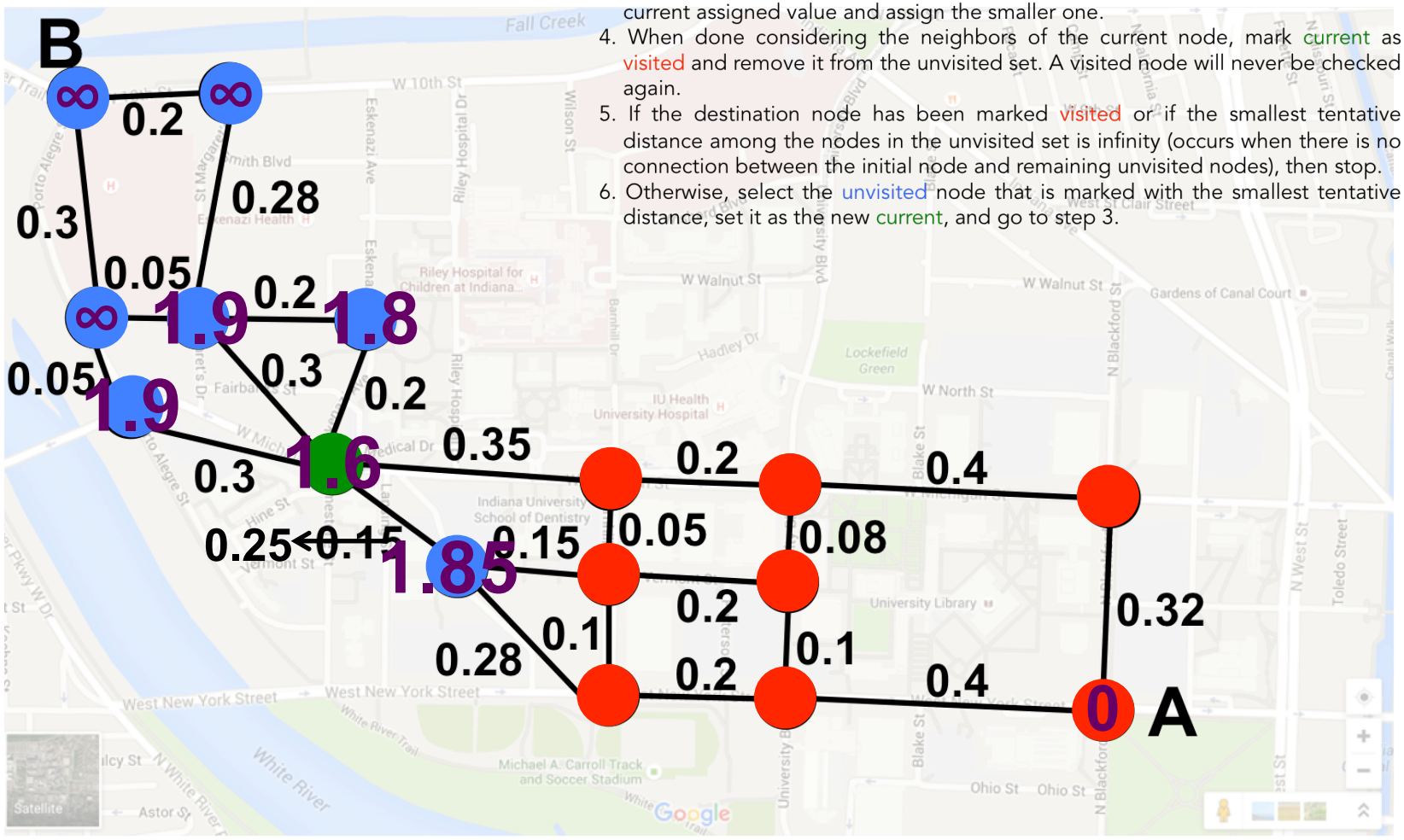
1. Assign to every node an initial tentative distance, zero for the initial node and "infinity" for all other nodes.
2. Set the initial node as current. Mark all other nodes unvisited. Create a set of all the unvisited nodes.
3. For the current node, consider all of its unvisited neighbors and calculate their tentative distances. Compare the newly calculated tentative distance to the current assigned value and assign the smaller one.
4. When done considering the neighbors of the current node, mark current as visited and remove it from the unvisited set. A visited node will never be checked again.
5. If the destination node has been marked visited or if the smallest tentative distance among the nodes in the unvisited set is infinity (occurs when there is no connection between the initial node and remaining unvisited nodes), then stop.
6. Otherwise, select the unvisited node that is marked with the smallest tentative distance, set it as the new current, and go to step 3.



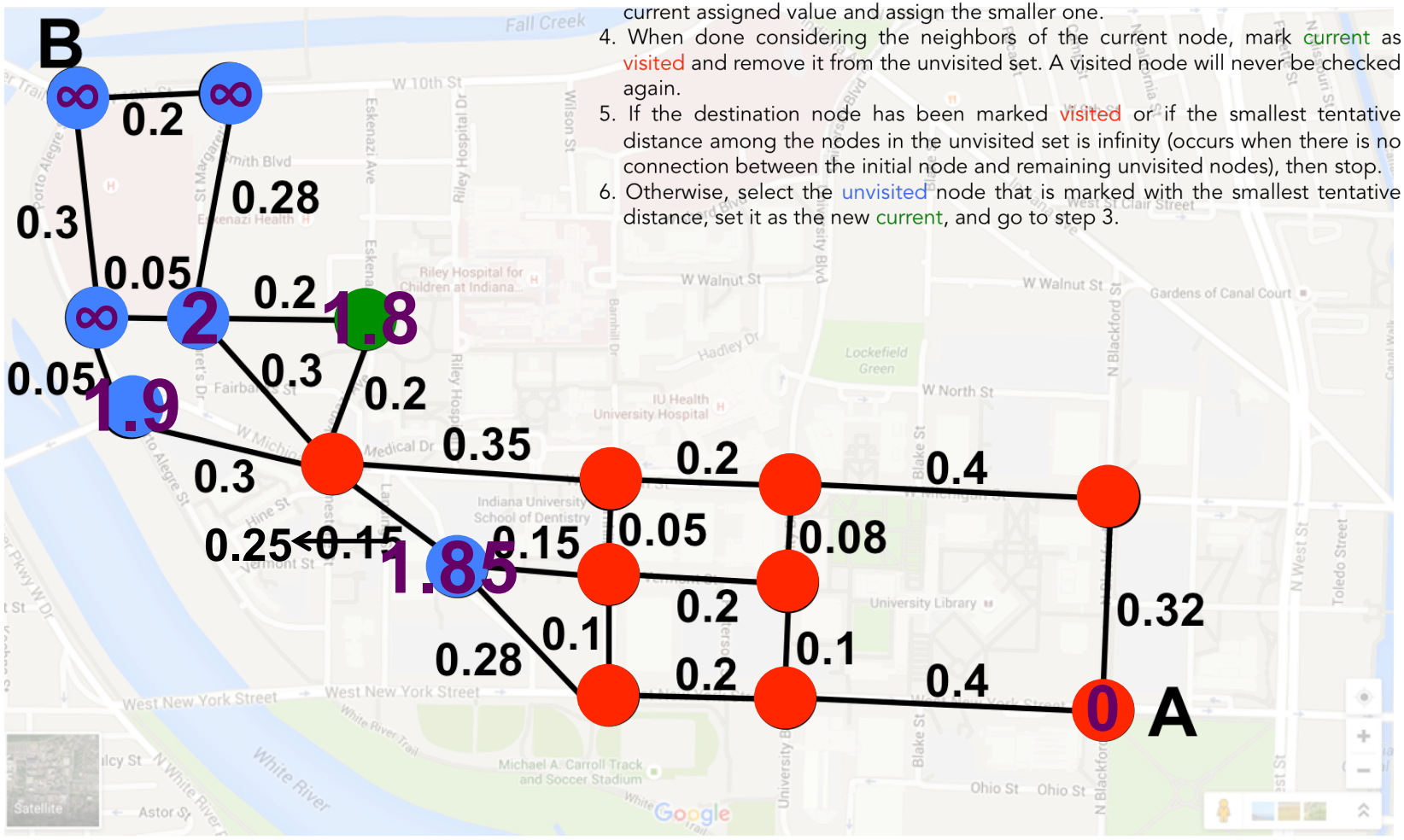
1. Assign to every node an initial tentative distance, zero for the initial node and "infinity" for all other nodes.
2. Set the initial node as current. Mark all other nodes unvisited. Create a set of all the unvisited nodes.
3. For the current node, consider all of its unvisited neighbors and calculate their tentative distances. Compare the newly calculated tentative distance to the current assigned value and assign the smaller one.
4. When done considering the neighbors of the current node, mark current as visited and remove it from the unvisited set. A visited node will never be checked again.
5. If the destination node has been marked visited or if the smallest tentative distance among the nodes in the unvisited set is infinity (occurs when there is no connection between the initial node and remaining unvisited nodes), then stop.
6. Otherwise, select the unvisited node that is marked with the smallest tentative distance, set it as the new current, and go to step 3.



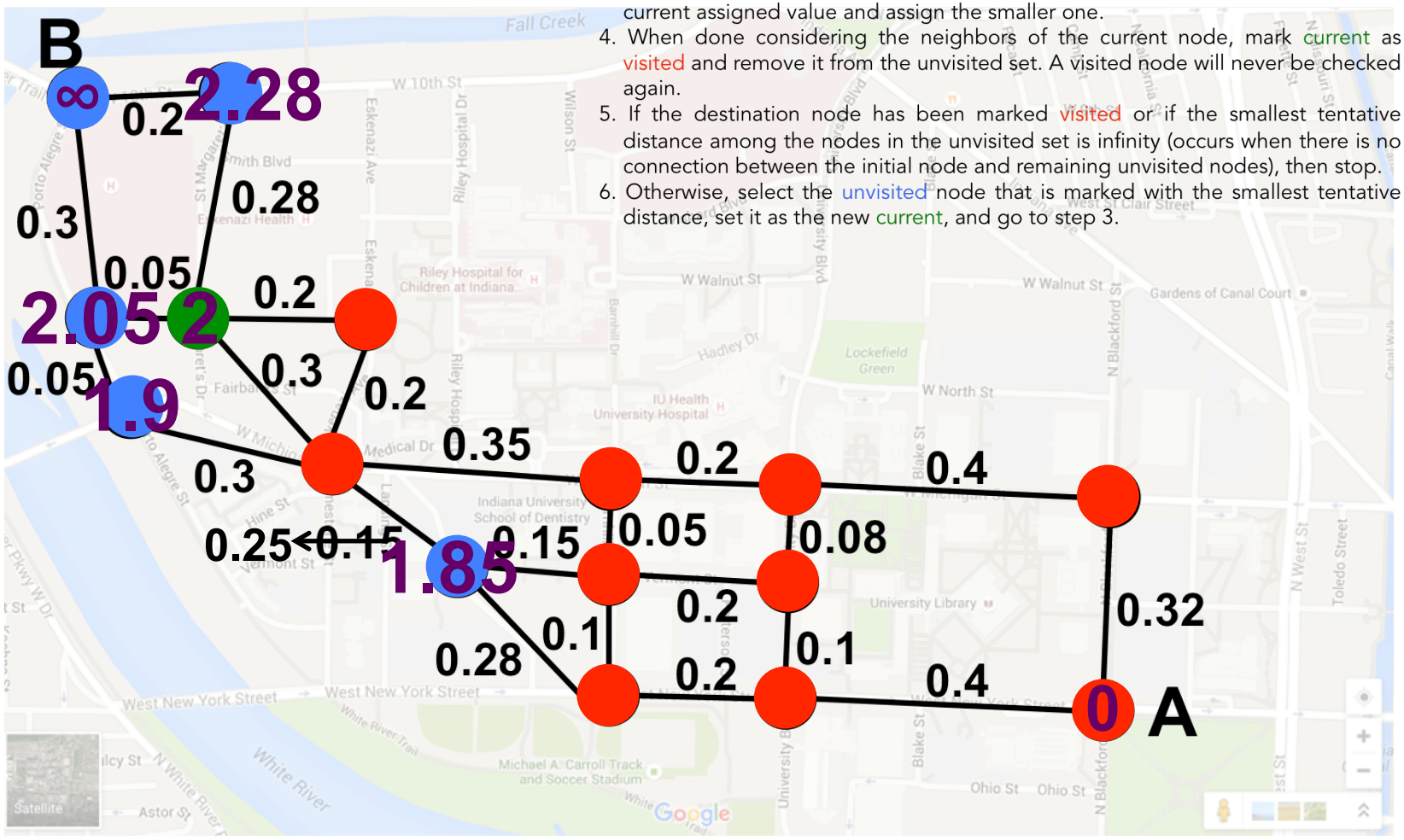
1. Assign to every node an initial tentative distance, zero for the initial node and "infinity" for all other nodes.
2. Set the initial node as current. Mark all other nodes unvisited. Create a set of all the unvisited nodes.
3. For the current node, consider all of its unvisited neighbors and calculate their tentative distances. Compare the newly calculated tentative distance to the current assigned value and assign the smaller one.
4. When done considering the neighbors of the current node, mark current as visited and remove it from the unvisited set. A visited node will never be checked again.
5. If the destination node has been marked visited or if the smallest tentative distance among the nodes in the unvisited set is infinity (occurs when there is no connection between the initial node and remaining unvisited nodes), then stop.
6. Otherwise, select the unvisited node that is marked with the smallest tentative distance, set it as the new current, and go to step 3.



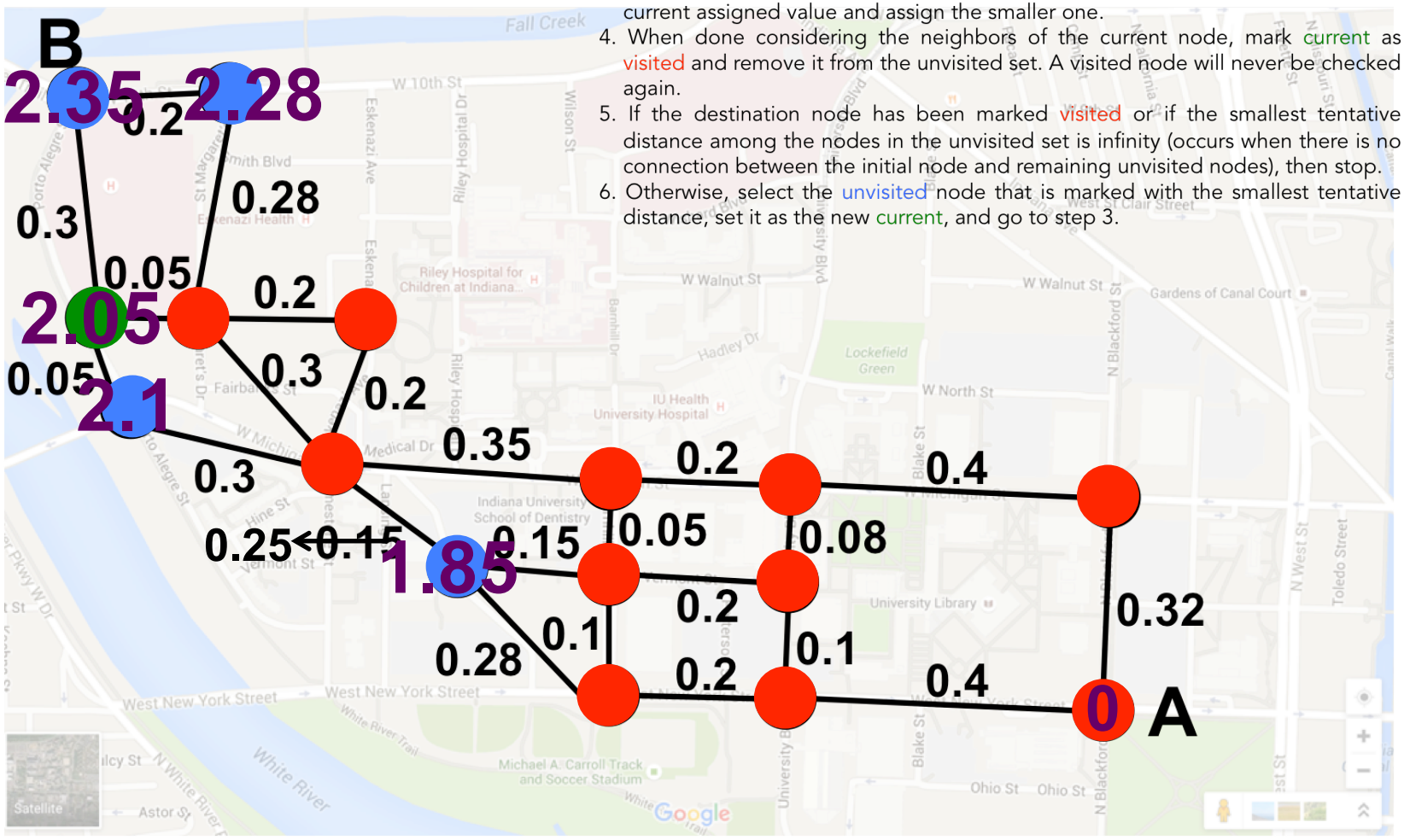
1. Assign to every node an initial tentative distance, zero for the initial node and "infinity" for all other nodes.
2. Set the initial node as current. Mark all other nodes unvisited. Create a set of all the unvisited nodes.
3. For the current node, consider all of its unvisited neighbors and calculate their tentative distances. Compare the newly calculated tentative distance to the current assigned value and assign the smaller one.
4. When done considering the neighbors of the current node, mark current as visited and remove it from the unvisited set. A visited node will never be checked again.
5. If the destination node has been marked visited or if the smallest tentative distance among the nodes in the unvisited set is infinity (occurs when there is no connection between the initial node and remaining unvisited nodes), then stop.
6. Otherwise, select the unvisited node that is marked with the smallest tentative distance, set it as the new current, and go to step 3.



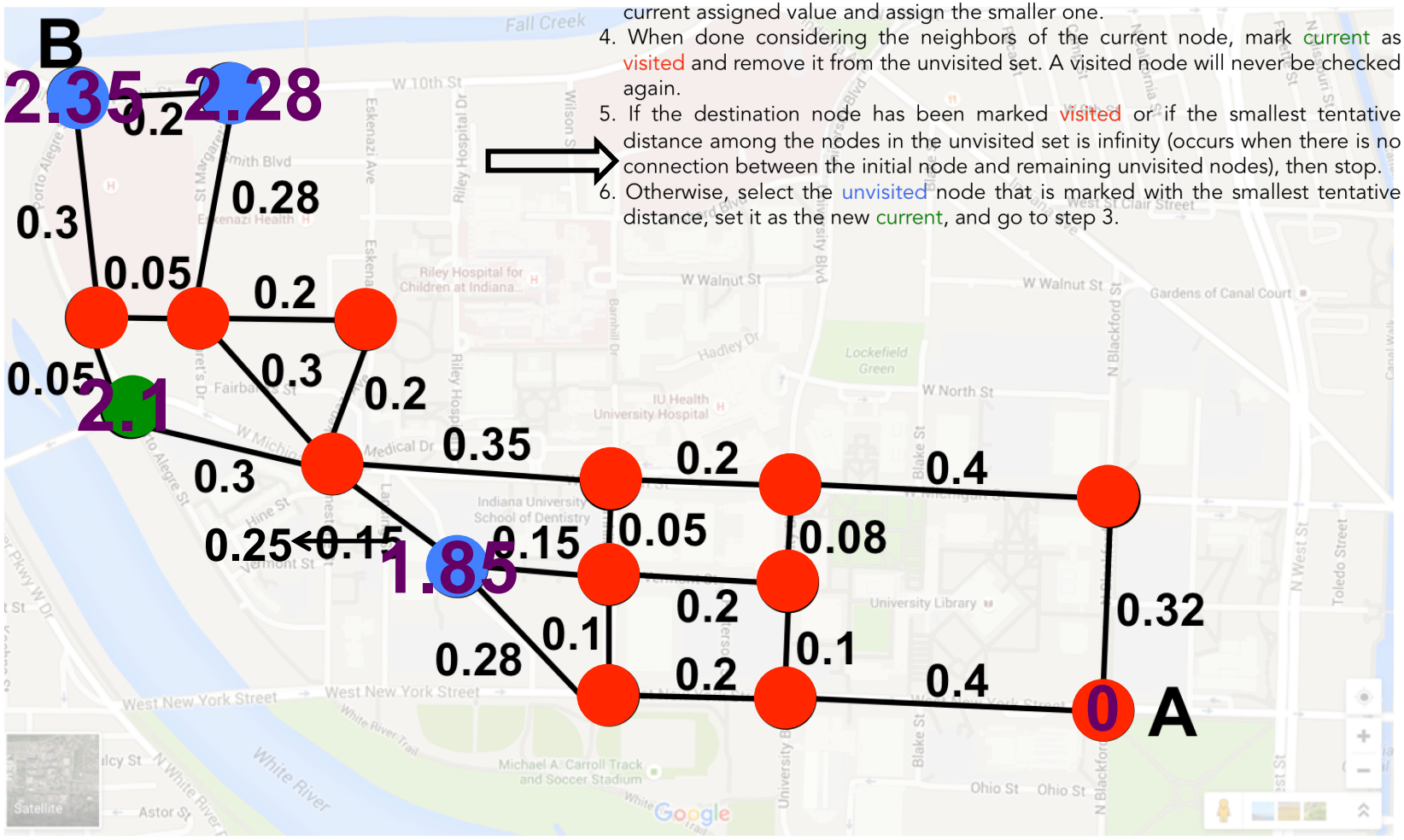
1. Assign to every node an initial tentative distance, zero for the initial node and "infinity" for all other nodes.
2. Set the initial node as current. Mark all other nodes unvisited. Create a set of all the unvisited nodes.
3. For the current node, consider all of its unvisited neighbors and calculate their tentative distances. Compare the newly calculated tentative distance to the current assigned value and assign the smaller one.
4. When done considering the neighbors of the current node, mark current as visited and remove it from the unvisited set. A visited node will never be checked again.
5. If the destination node has been marked visited or if the smallest tentative distance among the nodes in the unvisited set is infinity (occurs when there is no connection between the initial node and remaining unvisited nodes), then stop.
6. Otherwise, select the unvisited node that is marked with the smallest tentative distance, set it as the new current, and go to step 3.



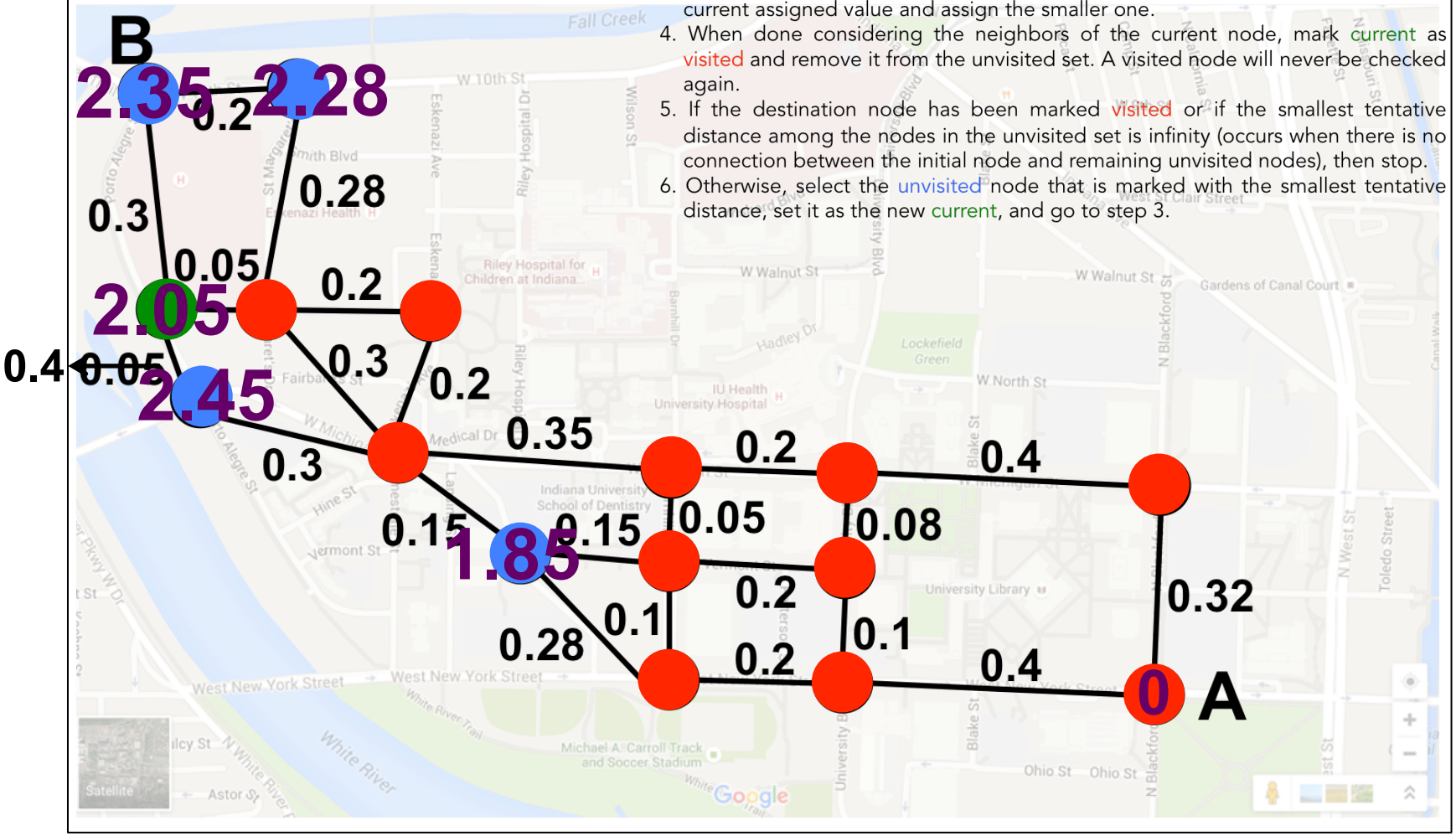
1. Assign to every node an initial tentative distance, zero for the initial node and "infinity" for all other nodes.
2. Set the initial node as current. Mark all other nodes unvisited. Create a set of all the unvisited nodes.
3. For the current node, consider all of its unvisited neighbors and calculate their tentative distances. Compare the newly calculated tentative distance to the current assigned value and assign the smaller one.
4. When done considering the neighbors of the current node, mark current as visited and remove it from the unvisited set. A visited node will never be checked again.
5. If the destination node has been marked visited or if the smallest tentative distance among the nodes in the unvisited set is infinity (occurs when there is no connection between the initial node and remaining unvisited nodes), then stop.
6. Otherwise, select the unvisited node that is marked with the smallest tentative distance, set it as the new current, and go to step 3.



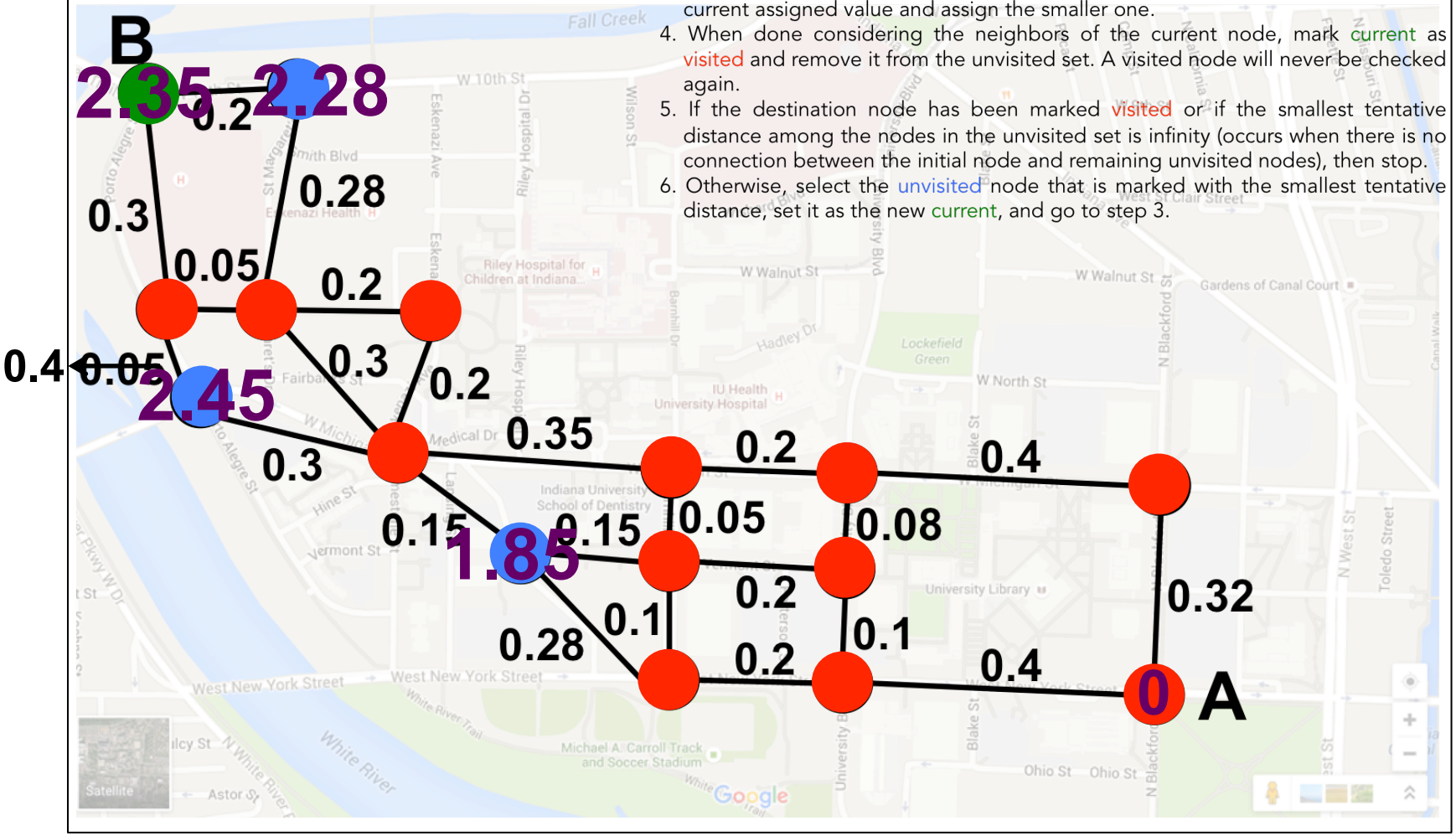
1. Assign to every node an initial tentative distance, zero for the initial node and "infinity" for all other nodes.
2. Set the initial node as current. Mark all other nodes unvisited. Create a set of all the unvisited nodes.
3. For the current node, consider all of its unvisited neighbors and calculate their tentative distances. Compare the newly calculated tentative distance to the current assigned value and assign the smaller one.
4. When done considering the neighbors of the current node, mark current as visited and remove it from the unvisited set. A visited node will never be checked again.
5. If the destination node has been marked visited or if the smallest tentative distance among the nodes in the unvisited set is infinity (occurs when there is no connection between the initial node and remaining unvisited nodes), then stop.
6. Otherwise, select the unvisited node that is marked with the smallest tentative distance, set it as the new current, and go to step 3.



1. Assign to every node an initial tentative distance, zero for the initial node and "infinity" for all other nodes.
2. Set the initial node as current. Mark all other nodes unvisited. Create a set of all the unvisited nodes.
3. For the current node, consider all of its unvisited neighbors and calculate their tentative distances. Compare the newly calculated tentative distance to the current assigned value and assign the smaller one.
4. When done considering the neighbors of the current node, mark current as visited and remove it from the unvisited set. A visited node will never be checked again.
5. If the destination node has been marked visited or if the smallest tentative distance among the nodes in the unvisited set is infinity (occurs when there is no connection between the initial node and remaining unvisited nodes), then stop.
6. Otherwise, select the unvisited node that is marked with the smallest tentative distance, set it as the new current, and go to step 3.

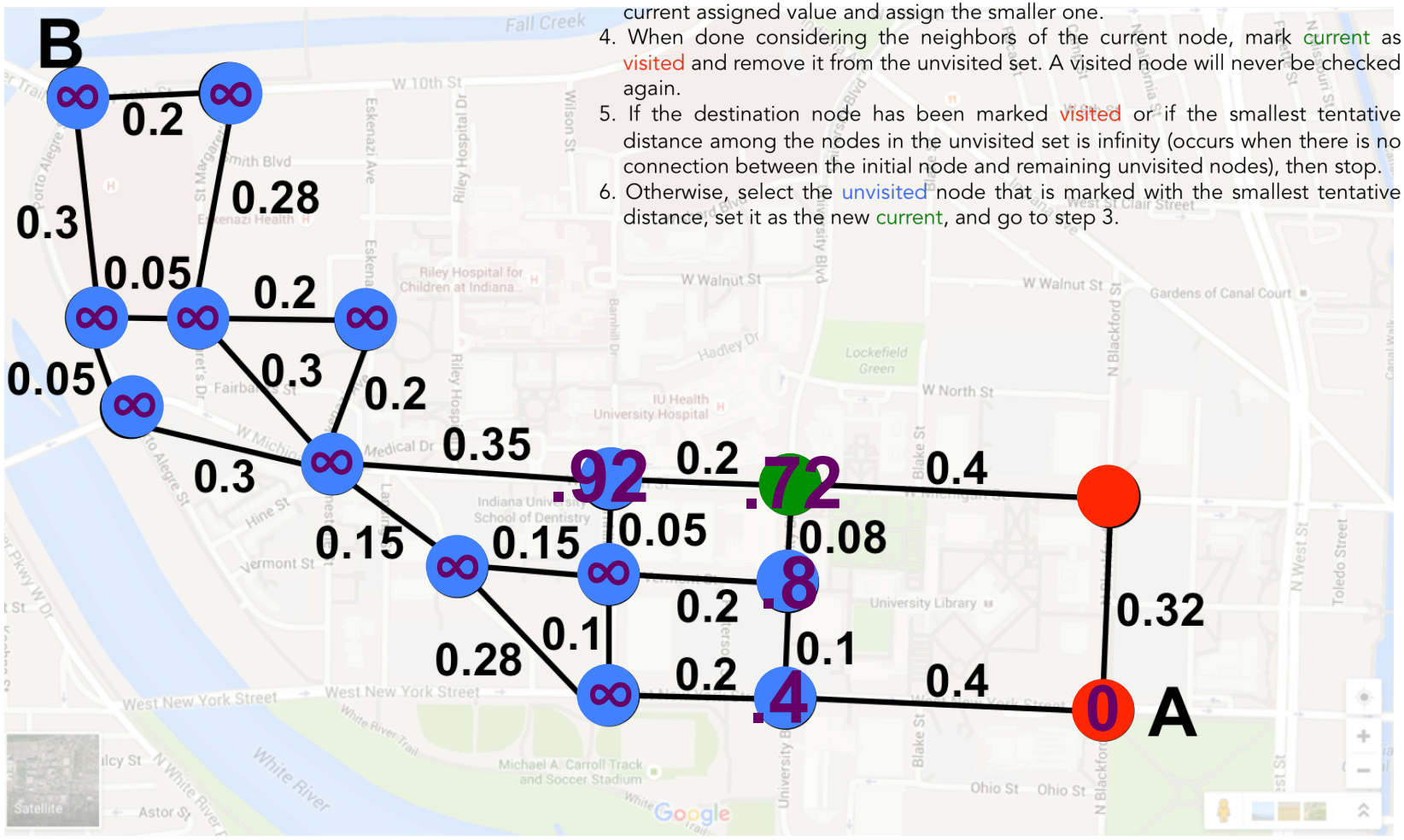


1. Assign to every node an initial tentative distance, zero for the initial node and "infinity" for all other nodes.
2. Set the initial node as current. Mark all other nodes unvisited. Create a set of all the unvisited nodes.
3. For the current node, consider all of its unvisited neighbors and calculate their tentative distances. Compare the newly calculated tentative distance to the current assigned value and assign the smaller one.
4. When done considering the neighbors of the current node, mark current as visited and remove it from the unvisited set. A visited node will never be checked again.
5. If the destination node has been marked visited or if the smallest tentative distance among the nodes in the unvisited set is infinity (occurs when there is no connection between the initial node and remaining unvisited nodes), then stop.
6. Otherwise, select the unvisited node that is marked with the smallest tentative distance, set it as the new current, and go to step 3.



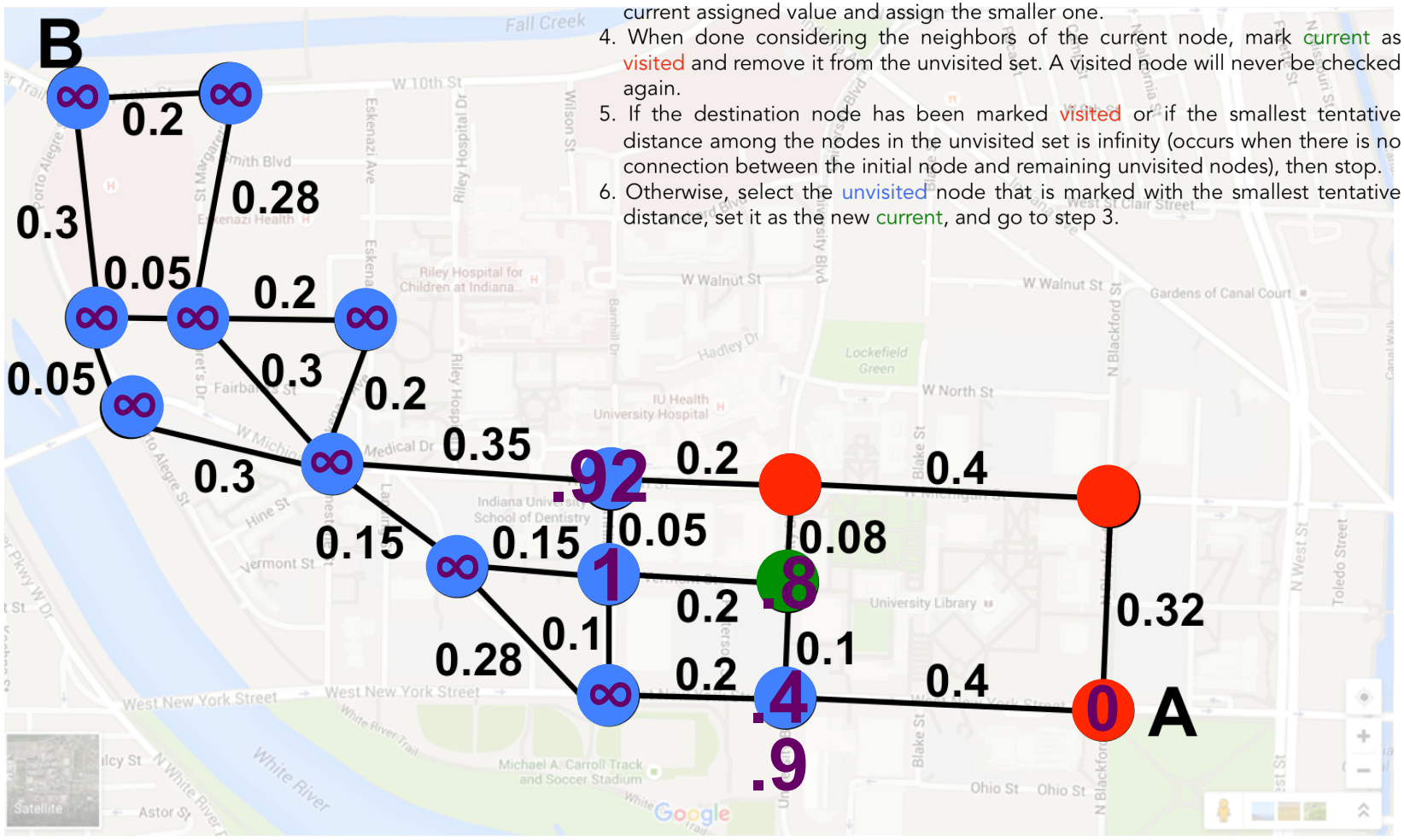
Let's go back.
STOP & THINK

1. Assign to every node an initial tentative distance, zero for the initial node and "infinity" for all other nodes.
2. Set the initial node as current. Mark all other nodes unvisited. Create a set of all the unvisited nodes.
3. For the current node, consider all of its unvisited neighbors and calculate their tentative distances. Compare the newly calculated tentative distance to the current assigned value and assign the smaller one.
4. When done considering the neighbors of the current node, mark current as visited and remove it from the unvisited set. A visited node will never be checked again.
5. If the destination node has been marked visited or if the smallest tentative distance among the nodes in the unvisited set is infinity (occurs when there is no connection between the initial node and remaining unvisited nodes), then stop.
6. Otherwise, select the unvisited node that is marked with the smallest tentative distance, set it as the new current, and go to step 3.

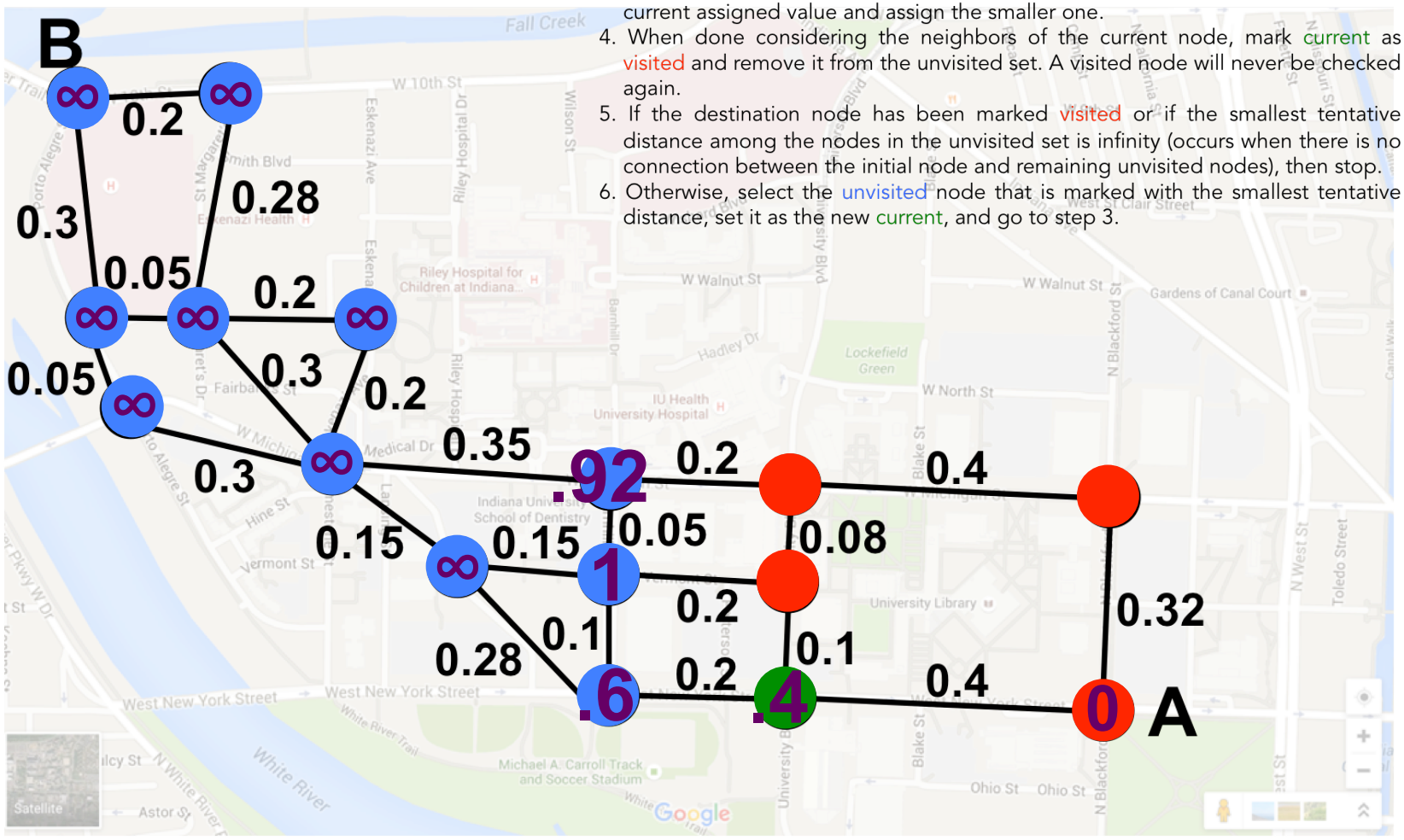


CORRECT

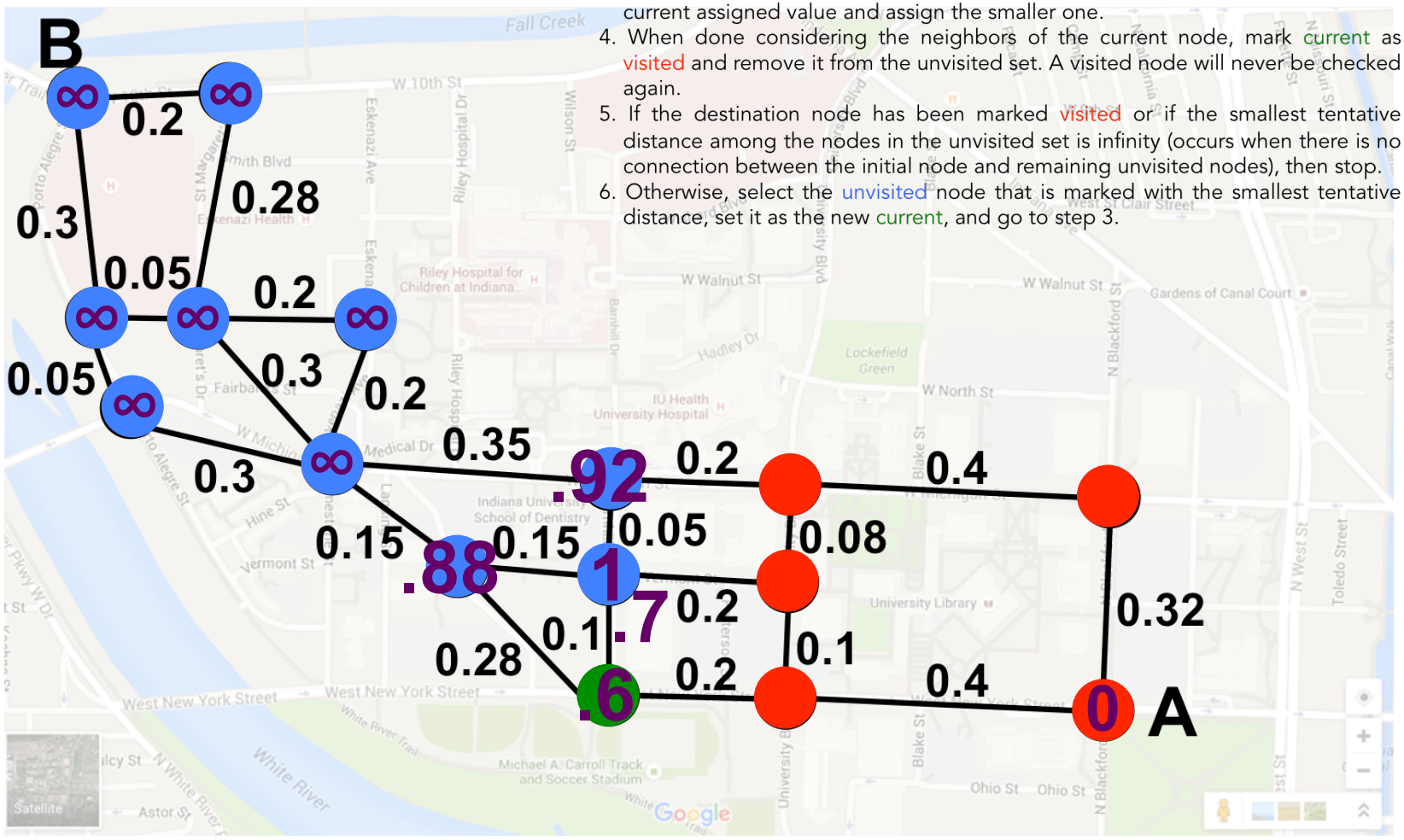
1. Assign to every node an initial tentative distance, zero for the initial node and "infinity" for all other nodes.
2. Set the initial node as current. Mark all other nodes unvisited. Create a set of all the unvisited nodes.
3. For the current node, consider all of its unvisited neighbors and calculate their tentative distances. Compare the newly calculated tentative distance to the current assigned value and assign the smaller one.
4. When done considering the neighbors of the current node, mark current as visited and remove it from the unvisited set. A visited node will never be checked again.
5. If the destination node has been marked visited or if the smallest tentative distance among the nodes in the unvisited set is infinity (occurs when there is no connection between the initial node and remaining unvisited nodes), then stop.
6. Otherwise, select the unvisited node that is marked with the smallest tentative distance, set it as the new current, and go to step 3.



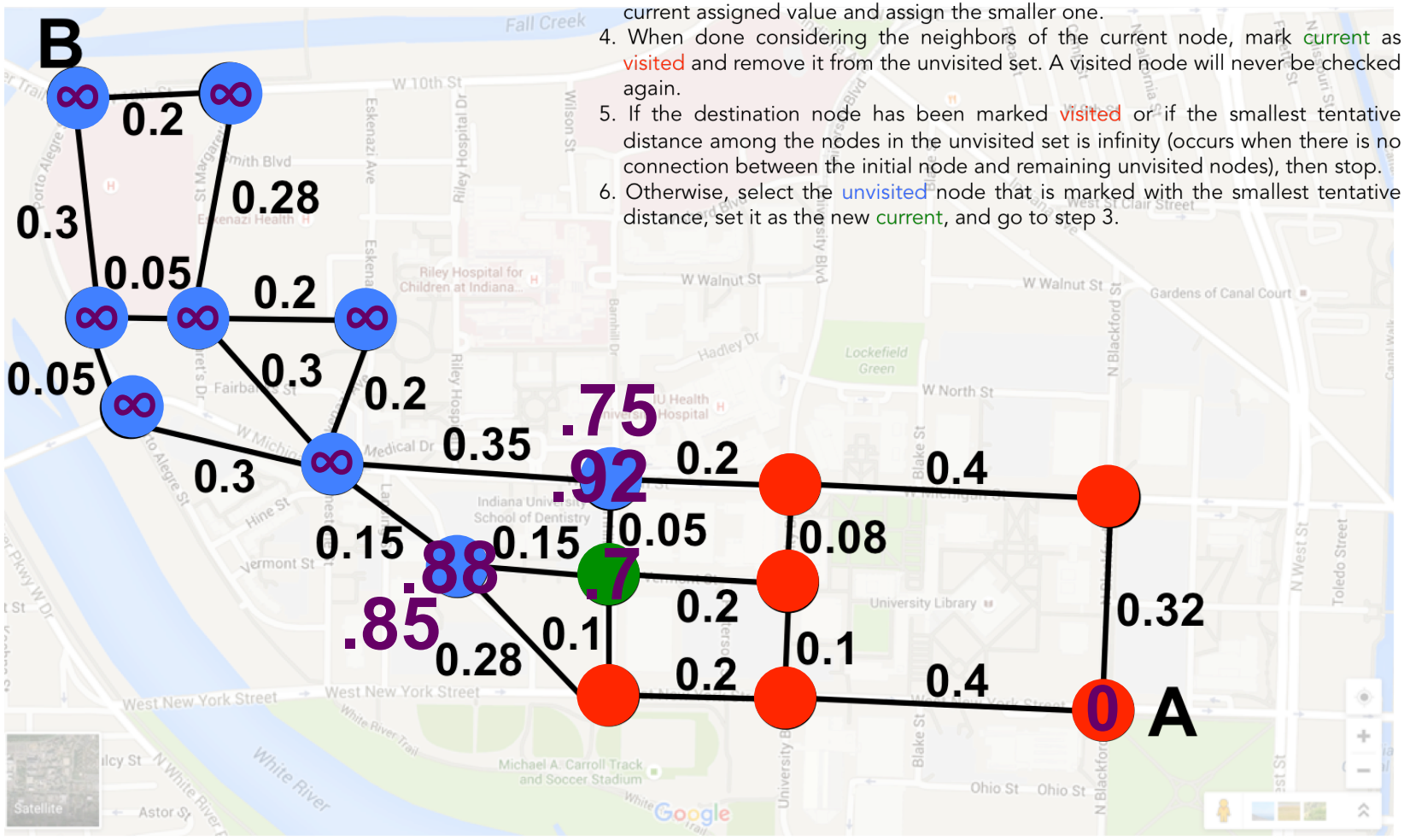
1. Assign to every node an initial tentative distance, zero for the initial node and "infinity" for all other nodes.
2. Set the initial node as current. Mark all other nodes unvisited. Create a set of all the unvisited nodes.
3. For the current node, consider all of its unvisited neighbors and calculate their tentative distances. Compare the newly calculated tentative distance to the current assigned value and assign the smaller one.
4. When done considering the neighbors of the current node, mark current as visited and remove it from the unvisited set. A visited node will never be checked again.
5. If the destination node has been marked visited or if the smallest tentative distance among the nodes in the unvisited set is infinity (occurs when there is no connection between the initial node and remaining unvisited nodes), then stop.
6. Otherwise, select the unvisited node that is marked with the smallest tentative distance, set it as the new current, and go to step 3.



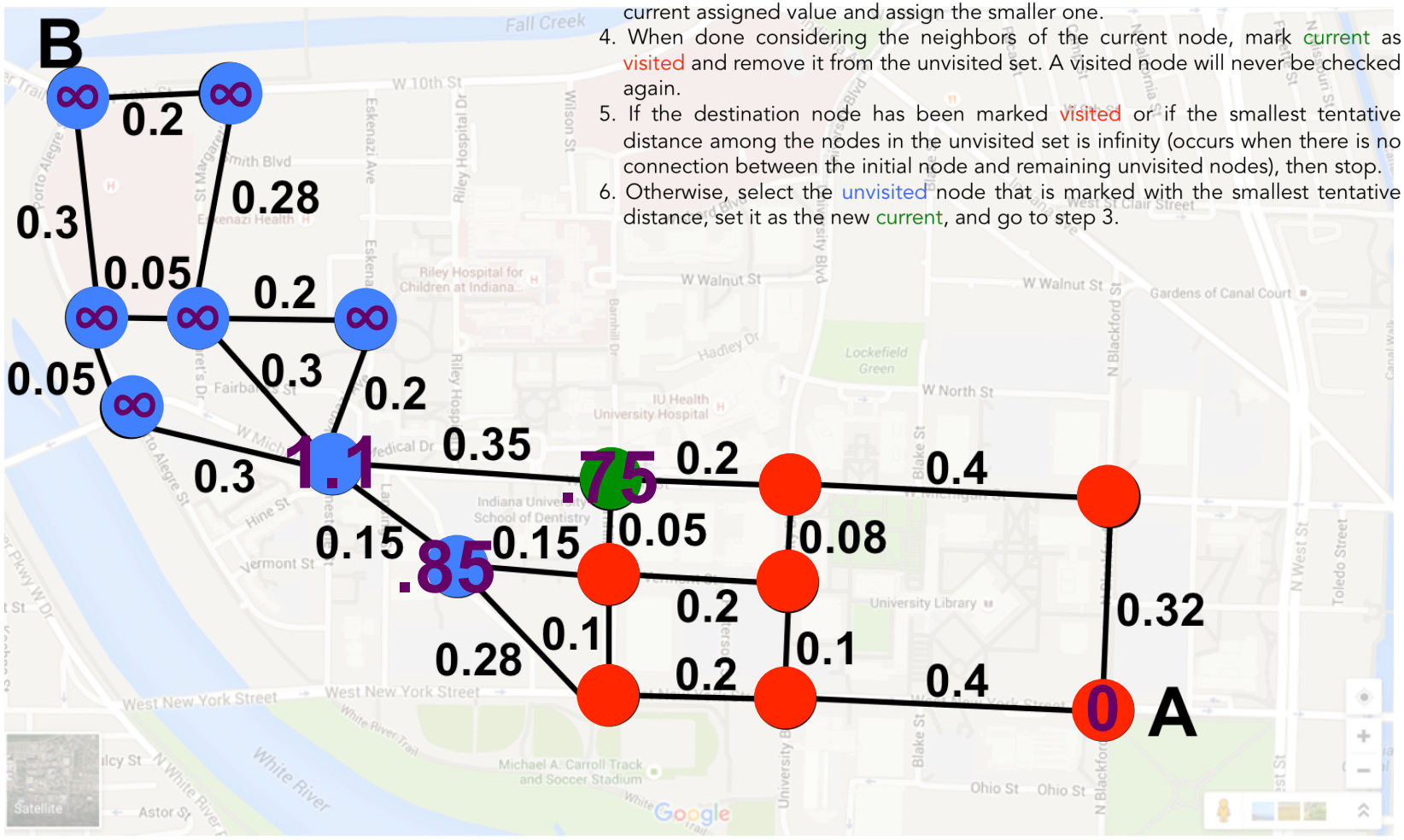
1. Assign to every node an initial tentative distance, zero for the initial node and "infinity" for all other nodes.
2. Set the initial node as current. Mark all other nodes unvisited. Create a set of all the unvisited nodes.
3. For the current node, consider all of its unvisited neighbors and calculate their tentative distances. Compare the newly calculated tentative distance to the current assigned value and assign the smaller one.
4. When done considering the neighbors of the current node, mark current as visited and remove it from the unvisited set. A visited node will never be checked again.
5. If the destination node has been marked visited or if the smallest tentative distance among the nodes in the unvisited set is infinity (occurs when there is no connection between the initial node and remaining unvisited nodes), then stop.
6. Otherwise, select the unvisited node that is marked with the smallest tentative distance, set it as the new current, and go to step 3.



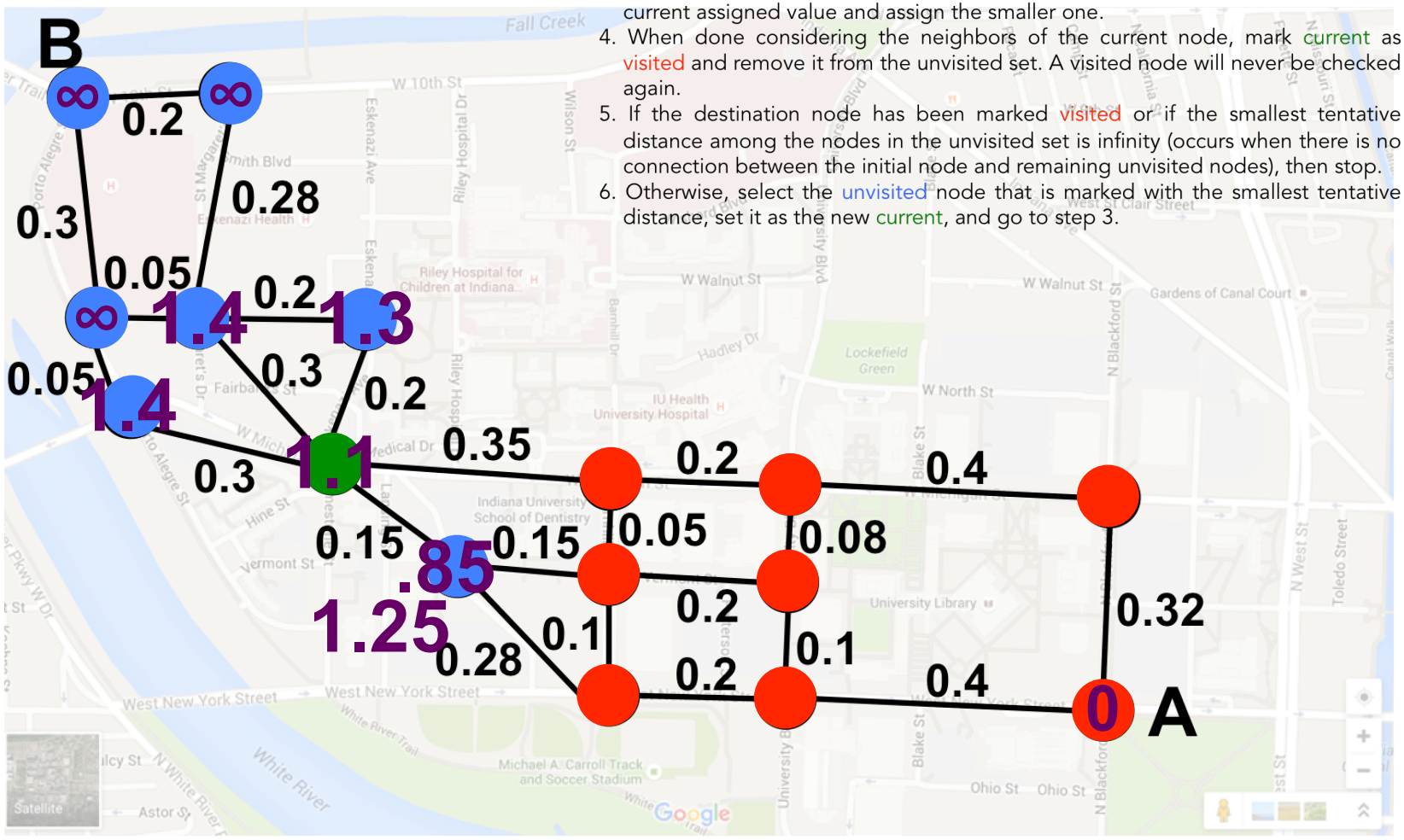
1. Assign to every node an initial tentative distance, zero for the initial node and "infinity" for all other nodes.
2. Set the initial node as current. Mark all other nodes unvisited. Create a set of all the unvisited nodes.
3. For the current node, consider all of its unvisited neighbors and calculate their tentative distances. Compare the newly calculated tentative distance to the current assigned value and assign the smaller one.
4. When done considering the neighbors of the current node, mark current as visited and remove it from the unvisited set. A visited node will never be checked again.
5. If the destination node has been marked visited or if the smallest tentative distance among the nodes in the unvisited set is infinity (occurs when there is no connection between the initial node and remaining unvisited nodes), then stop.
6. Otherwise, select the unvisited node that is marked with the smallest tentative distance, set it as the new current, and go to step 3.



1. Assign to every node an initial tentative distance, zero for the initial node and "infinity" for all other nodes.
2. Set the initial node as current. Mark all other nodes unvisited. Create a set of all the unvisited nodes.
3. For the current node, consider all of its unvisited neighbors and calculate their tentative distances. Compare the newly calculated tentative distance to the current assigned value and assign the smaller one.
4. When done considering the neighbors of the current node, mark current as visited and remove it from the unvisited set. A visited node will never be checked again.
5. If the destination node has been marked visited or if the smallest tentative distance among the nodes in the unvisited set is infinity (occurs when there is no connection between the initial node and remaining unvisited nodes), then stop.
6. Otherwise, select the unvisited node that is marked with the smallest tentative distance, set it as the new current, and go to step 3.

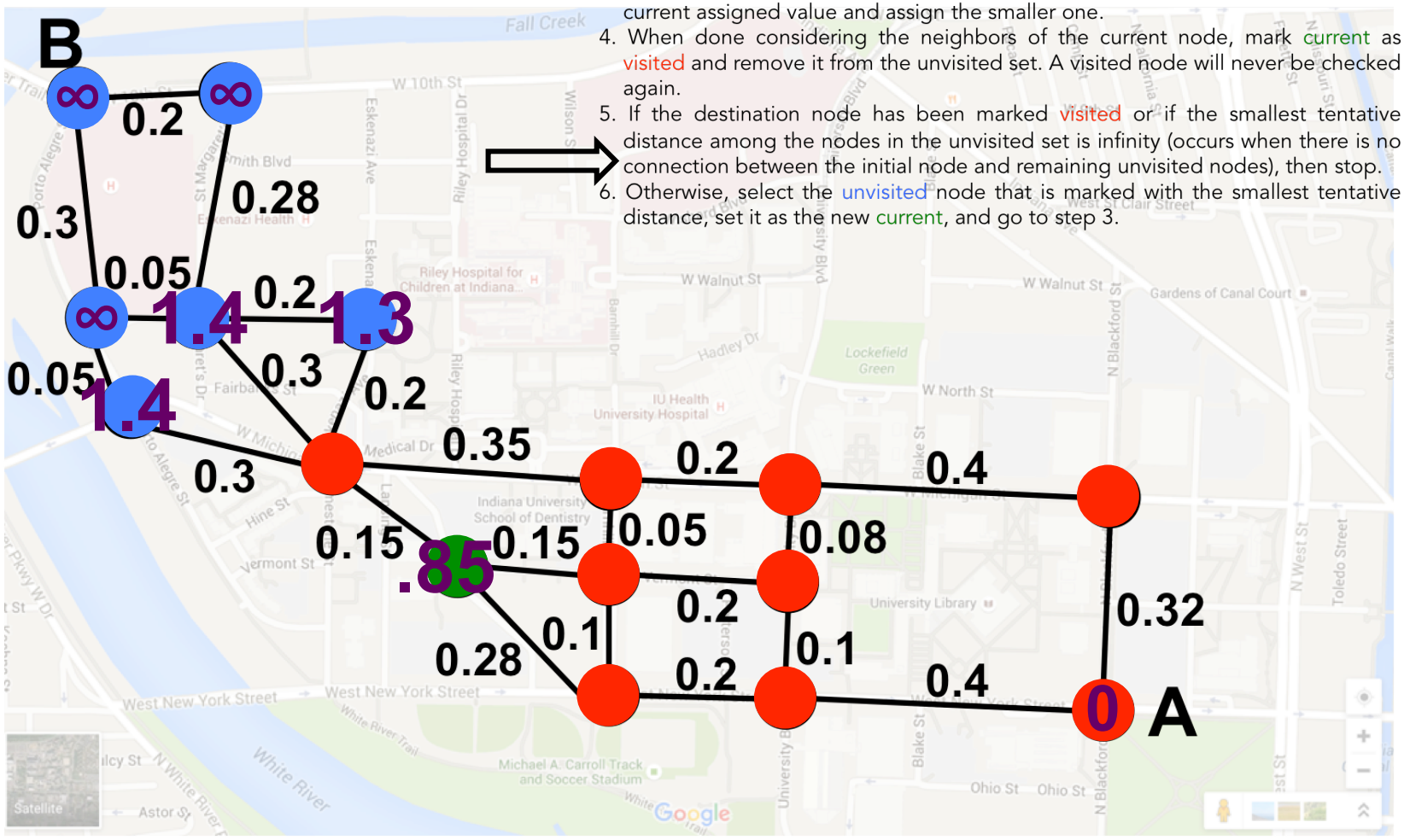


1. Assign to every node an initial tentative distance, zero for the initial node and "infinity" for all other nodes.
2. Set the initial node as current. Mark all other nodes unvisited. Create a set of all the unvisited nodes.
3. For the current node, consider all of its unvisited neighbors and calculate their tentative distances. Compare the newly calculated tentative distance to the current assigned value and assign the smaller one.
4. When done considering the neighbors of the current node, mark current as visited and remove it from the unvisited set. A visited node will never be checked again.
5. If the destination node has been marked visited or if the smallest tentative distance among the nodes in the unvisited set is infinity (occurs when there is no connection between the initial node and remaining unvisited nodes), then stop.
6. Otherwise, select the unvisited node that is marked with the smallest tentative distance, set it as the new current, and go to step 3.



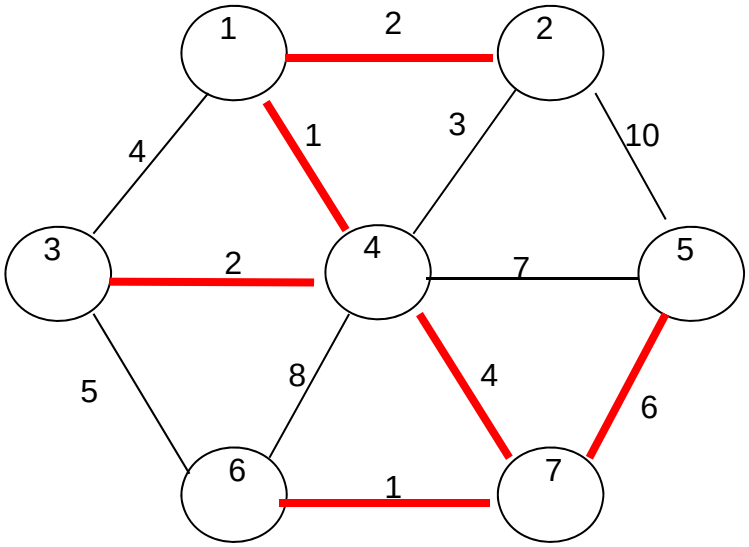
We cannot reach the target--trapped in local minima:

1. Assign to every node an initial tentative distance, zero for the initial node and "infinity" for all other nodes.
2. Set the initial node as current. Mark all other nodes unvisited. Create a set of all the unvisited nodes.
3. For the current node, consider all of its unvisited neighbors and calculate their tentative distances. Compare the newly calculated tentative distance to the current assigned value and assign the smaller one.
4. When done considering the neighbors of the current node, mark current as visited and remove it from the unvisited set. A visited node will never be checked again.
5. If the destination node has been marked visited or if the smallest tentative distance among the nodes in the unvisited set is infinity (occurs when there is no connection between the initial node and remaining unvisited nodes), then stop.
6. Otherwise, select the unvisited node that is marked with the smallest tentative distance, set it as the new current, and go to step 3.



Minimum Spanning Tree

- The problem
  - Consider undirected graph only
  - *Spanning tree*: a tree by graph edges that connect all the vertices of the graph.
  - *Minimum spanning tree*: a spanning tree with the minimum total edge cost.
  - Minimum spanning tree of graph G exists if and only if G is connected.



remember: a graph is connected if there is a path from every vertex to every other vertex.

– Prim's algorithm
  - a greedy algorithm
  - basic steps: starting from an arbitrary vertex as the root. At any point of the algorithm, there is a set of vertices that can be included in the tree, and the rest are not. The algorithm, at each stage, find a new vertex, v, to add to the tree such that the cost of (u, v) is the smallest among all edges where u is in the tree and v is not.
  - The same process as the greedy algorithm in shortest path problem

    a.) the graph is undirected

    b.) the "dist" is defined as the shortest cost of edge (u, v), with u currently in the tree (known) and v currently not (unknown)

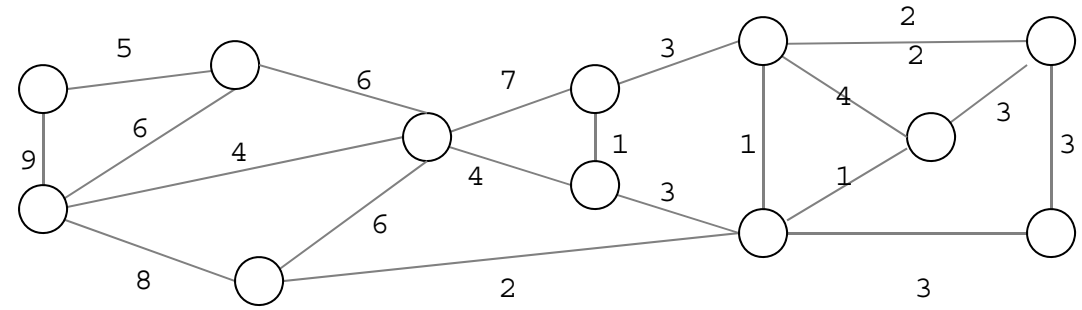
    c.) updating rule:

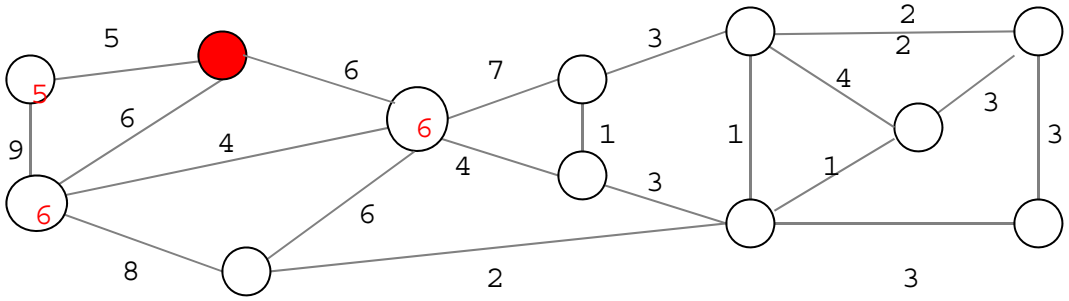
```
for (each vertex w adjacent to v)
  w.dist = min(w.dist, c(w, v))
```

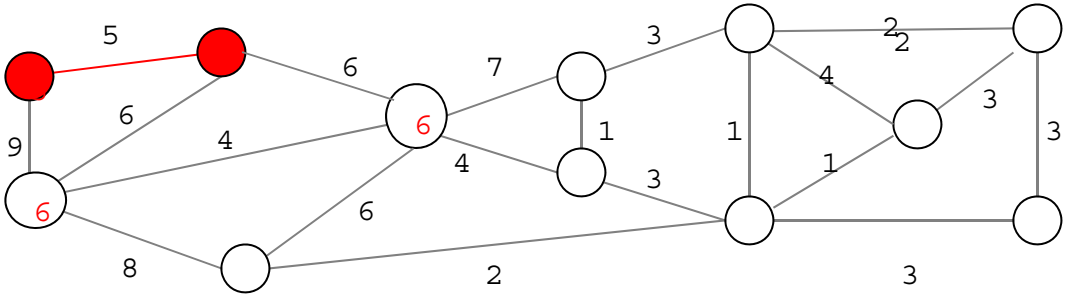
  - running time

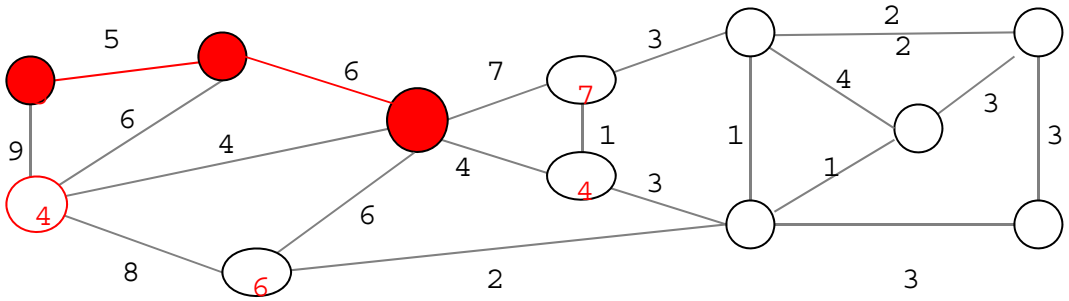
    $O(|V|^2)$: without heap

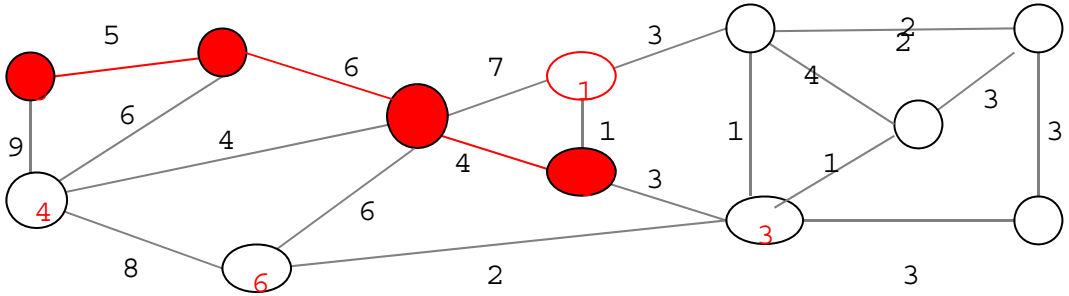
    $O(|E|\log |V|)$: using a binary heap.

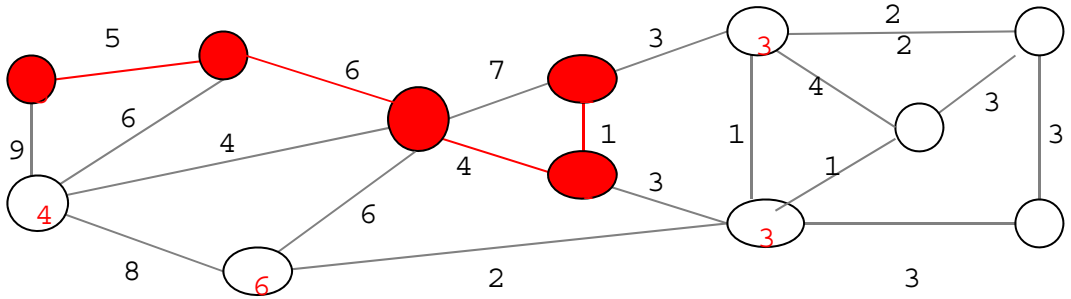

5
6
7
3
2
2
6
4
4
3
9
4
1
1
3
4
1
3
6
1
3
8
2
3

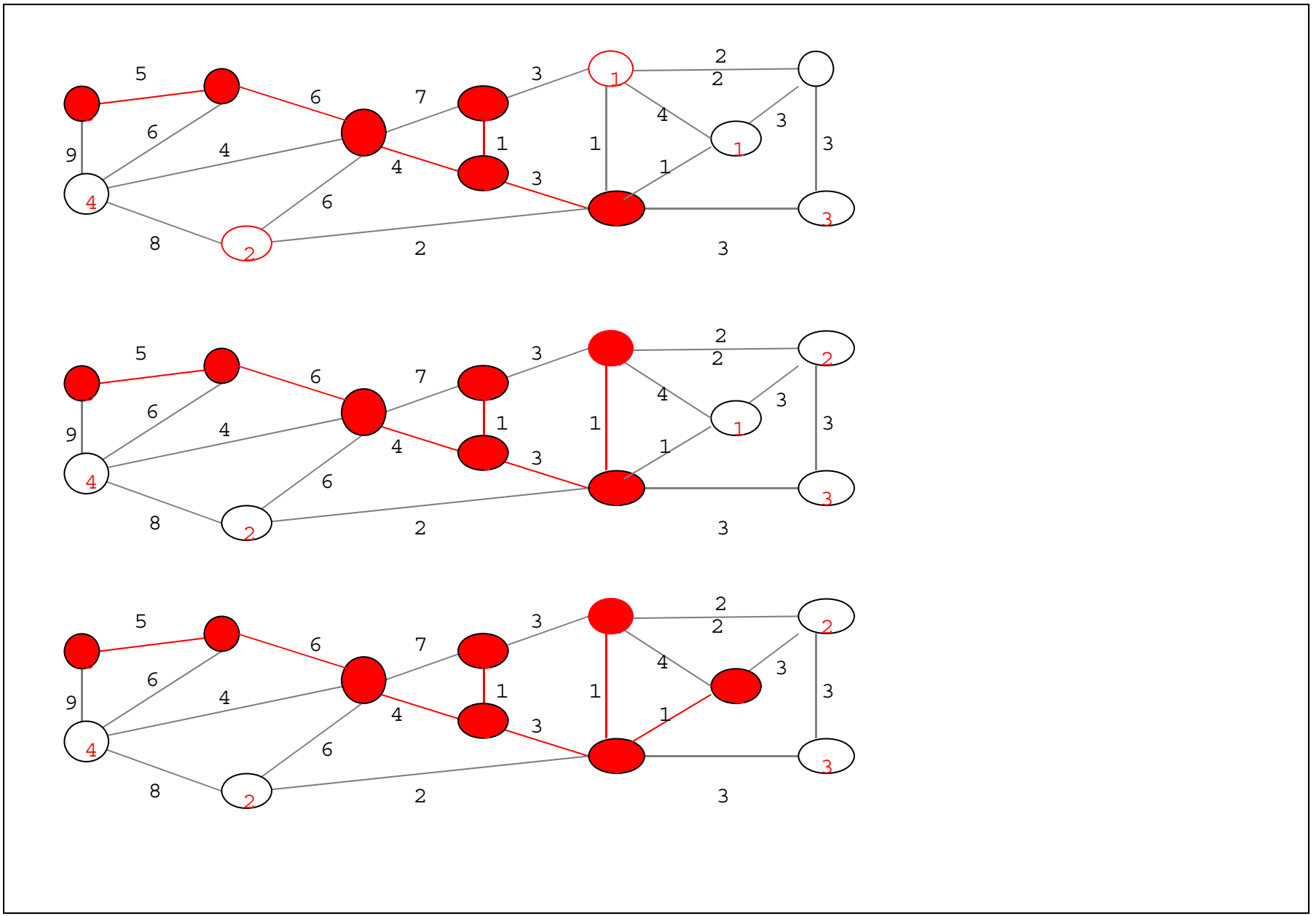

5
6
7
3
2
2
5
6
4
4
3
9
4
1
1
3
6
4
1
3
6
6
1
3
8
2
3

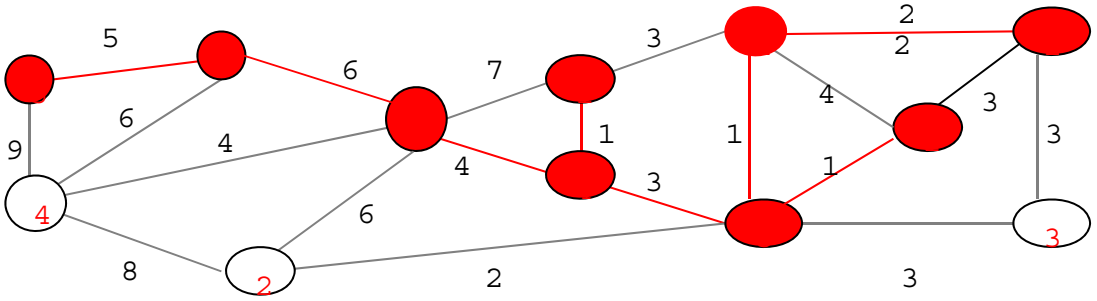

5
6
7
3
2
2
6
4
4
3
9
4
1
1
3
6
4
1
3
6
6
1
3
8
2
3

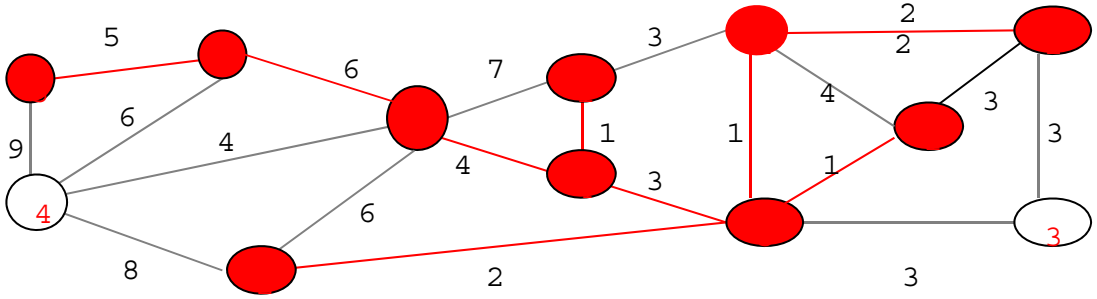

5
6
7
3
2
2
6
4
3
9
4
1
1
3
4
4
3
1
6
3
8
6
2
3

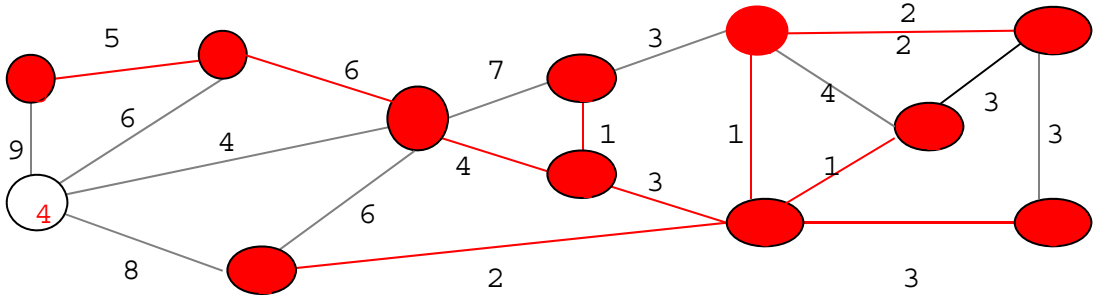

5
6
7
3
2
2
6
4
3
9
4
1
1
3
4
1
3
3
6
1
8
6
2
3

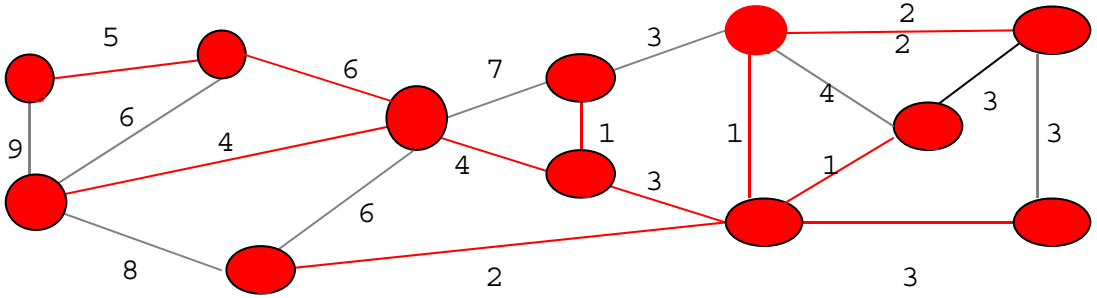

5
6
7
3
3
2
2
6
4
3
9
4
1
1
3
4
1
3
3
6
1
8
6
2
3


5
6
7
3
1
2
2
6
4
1
4
3
9
4
1
1
3
4
1
3
6
3
8
2
2
3


5
6
7
3
2
2
6
4
4
3
9
4
1
1
3
3
6
1
3
8
2
2
3


5
6
7
3
2
2
6
4
4
3
9
4
1
1
3
3
6
1
3
8
2
3


5
6
7
3
2
2
6
4
4
3
9
4
1
1
3
3
6
1
3
8
2
3


5
6
7
3
2
2
6
4
4
3
9
4
1
1
3
3
6
1
3
8
2
3