# Data Structure

## Lecture#24: Graphs 2
## (Chapter 11)

**U Kang**
**Seoul National University**


# In This Lecture

- Shortest path problem
- Main idea of Dijkstra's algorithm
- Cost analysis of Dijkstra's algorithm

# Shortest Paths Problems

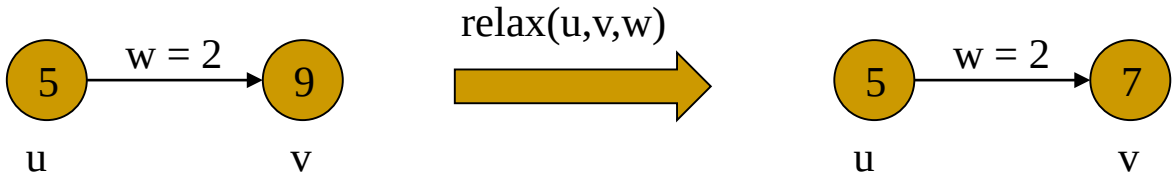
- Input: A graph with weights associated with each edge.
- Output: The list of edges forming the shortest path.
- Sample problems:
  - Find shortest path between two named vertices
  - Find shortest path from a vertex *s* to all other vertices
  - Find shortest path between all pairs of vertices
- Will calculate shortest distance

# Shortest Paths Definitions

- $\delta(s, u)$ is the shortest distance from vertex s to u.
- w(u, v) is the weight of the edge connecting u to v.
  - If there is no such edge, then $w(u, v) = \infty$.

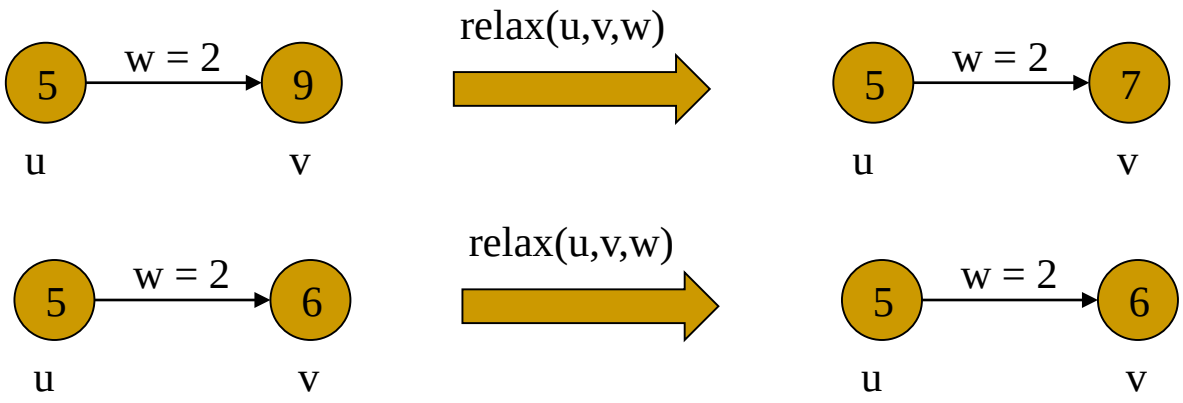
# Single-Source Shortest Paths

- Given start vertex *s*, find the shortest path from *s* to all other vertices.
- Algorithm by Dijkstra
  - Maintain a set S of visited vertices. Also maintain distance array D of size *n* (# of vertices)
    - D[i] stores current estimate of distance d(s,i) between s and i
  - Initially, S is empty, and D[s] = 0.
  - In each iteration (run this for *n* times)
    - Add the minimum-distance vertex u to S
      - Use D to do that
    - Update D for u's neighbors v by the following *relax* operation

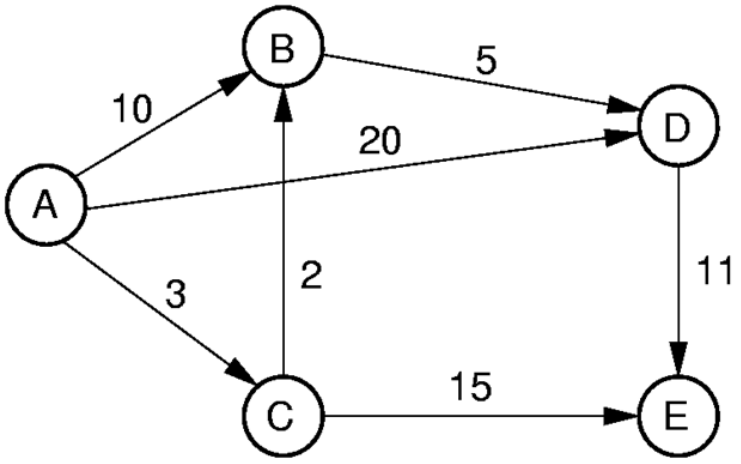


# Relax Operation

- Update D for u's neighbors v by the following *relax* operation
  - relax(u,v,w)
    - If d(s,v) > d(s,u) + w
      - d(s,v) = d(s,u) + w
      - prev(v) = u



# Dijkstra's Algorithm Example




Start vertex: A

D vector

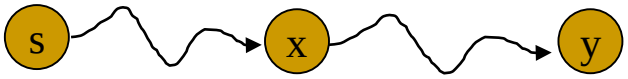
| | A | B | C | D | E |
|---|---|---|---|---|---|
| Initial | 0 | ∞ | ∞ | ∞ | ∞ |
| Process A | 0 | 10 | 3 | 20 | ∞ |
| Process C | 0 | 5 | 3 | 20 | 18 |
| Process B | 0 | 5 | 3 | 10 | 18 |
| Process D | 0 | 5 | 3 | 10 | 18 |
| Process E | 0 | 5 | 3 | 10 | 18 |

# Correctness of Dijkstra

- Algorithm by Dijkstra
  - Maintain a set S of visited vertices. Also maintain distance array D of size *n* (# of vertices)
  - Initially, S is empty, and D[s] = 0.
  - In each iteration (run this for *n* times)
    - Add the minimum-distance vertex u to S
    - Update D of u's neighbors v by the *relax* operation
- Correctness
  - Claim: when u is added to S, d(s,u) = $\delta$(s,u)
    - This means that when u is added to S, d(s,u) is the shortest distance, and we found the shortest path to u

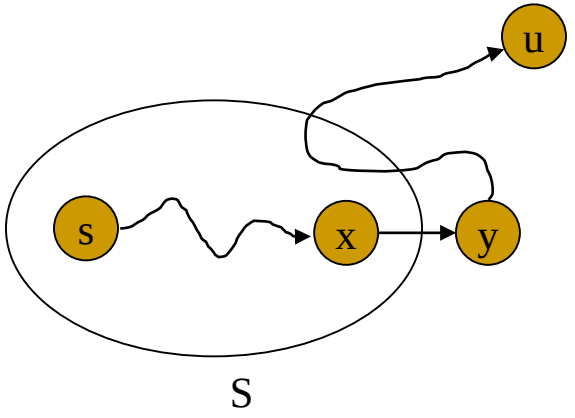
# Correctness of Dijkstra

- Key idea (from Dynamic Programming)
  - Assume there is a node x in the shortest path P from s to y
  - Then, the path from s to x in P is the shortest path from s to x



# Correctness of Dijkstra


- Claim: when u is added to S, d(s,u) = $\delta$(s,u)
  - (Proof by Induction)
    - (Base Case) When u = s, the claim is trivially true
    - (Induction Hypothesis) Assume the claim is true for S
    - Now we add u to S. We need to show that d(s,u) = $\delta$(s,u)
    - Assume a shortest path s to u, and x->y are the first boundary vertices between S and V\S in the shortest path
    - It can be shown that d(s,y) = $\delta$(s,y)
      - d(s,y) $\leq$ d(s,x) + w(x,y) (from relax on x)
      - = $\delta$(s,x) + w(x,y) (from I.H.)
      - = $\delta$(s,y) (key idea)
    - That implies y and u are the same
      - d(s,y) = $\delta$(s,y) $\leq$ $\delta$(s,u) $\leq$ d(s,u)
      - But, d(s,u) $\leq$ d(s,y) since u is added to S
    - Thus, d(s,u) = $\delta$(s,u)



# Dijkstra's Implementation

```
// Compute shortest path distances from s,
//   store them in D
void Dijkstra(Graph G, int s, int[] D) {
  for (int i=0; i<G.n(); i++) // Initialize
    D[i] = Integer.MAX_VALUE;
  D[s] = 0;
  for (int i=0; i<G.n(); i++) {
    int v = minVertex(G, D);
    G.setMark(v, VISITED);
    if (D[v] == Integer.MAX_VALUE) return;
    for (int w = G.first(v); w < G.n();
                             w = G.next(v, w))
      if (D[w] > (D[v] + G.weight(v, w)))
        D[w] = D[v] + G.weight(v, w);
  }
}
```

minVertex() returns an unvisited vertex with the minimum distance

# Implementing minVertex

- Issue: How to determine the next-closest vertex? (I.e., implement **minVertex**)
- Approach 1: Scan through the table of current distances.
  - Total Cost of Dijkstra: $\Theta(|\mathbf{V}|^2 + |\mathbf{E}|) = \Theta(|\mathbf{V}|^2)$.
- Approach 2: Store unprocessed vertices using a min-heap to implement a priority queue ordered by D value. Must update priority queue for each edge.
  - Cost: $\Theta((|\mathbf{V}| + |\mathbf{E}|)\log|\mathbf{V}|)$

# Approach 1

```
int minVertex(Graph G, int[] D) {
  int v = 0; // Initialize to unvisited vertex;
  for (int i=0; i<G.n(); i++)
    if (G.getMark(i) == UNVISITED)
      { v = i; break; }
  for (int i=0; i<G.n(); i++)
    // Now find smallest value
    if ((G.getMark(i) == UNVISITED) &&
        (D[i] < D[v]))
      v = i;
  return v;
}
```

# What You Need to Know

- Shortest path problem
  - Given start vertex *s*, find the shortest path from *s* to *v*
- Main idea of Dijkstra's algorithm
- Cost analysis of Dijkstra's algorithm


# Questions?