

CSCI 2270-305 Recitation 04/09: Hash Tables and Midterm Review

Varad Deshmukh



Logistics

- Assignment 9 is due next Wednesday.
 - Click on **Finish Attempt** and **Submit All and Finish** when you are done!
- Midterm Tomorrow at 5 pm!
 - MCQs to be completed within 1 hour.
 - Coding questions due by midnight.
 - TAs will be available to answer any questions until 8 pm over Zoom.
- Coming up next week ---
 - Final Project Announcement (Due April 26th)
 - Interview Grading for Assignments 5 to 9. I will be scheduling Moodle appointment slots.

Hash Tables

- A data structure that uses a *(key, value)* mechanism for storage.
- Example, storing the student data (identikkey, DOB, year, major, etc.) using their CU Identikkeys.
 - Key: Identikkey
 - Value: Student Data
- Each entry in the table is indexed using the key.

Key	Hash Table (Values)
0	0, Carl Sagan, Sophomore, Physics, ..
1	1, Donald Knuth, Senior, Computer Science, ..
2	2, Paul Erdos, Freshman, Mathematics, ..
..	..
...	..

Why Hashing?

	Insertion	Deletion	Searching
Arrays	$O(n)$	$O(n)$	$O(n)$
Linked Lists	$O(1)$	$O(1)$	$O(n)$
Hash Tables			

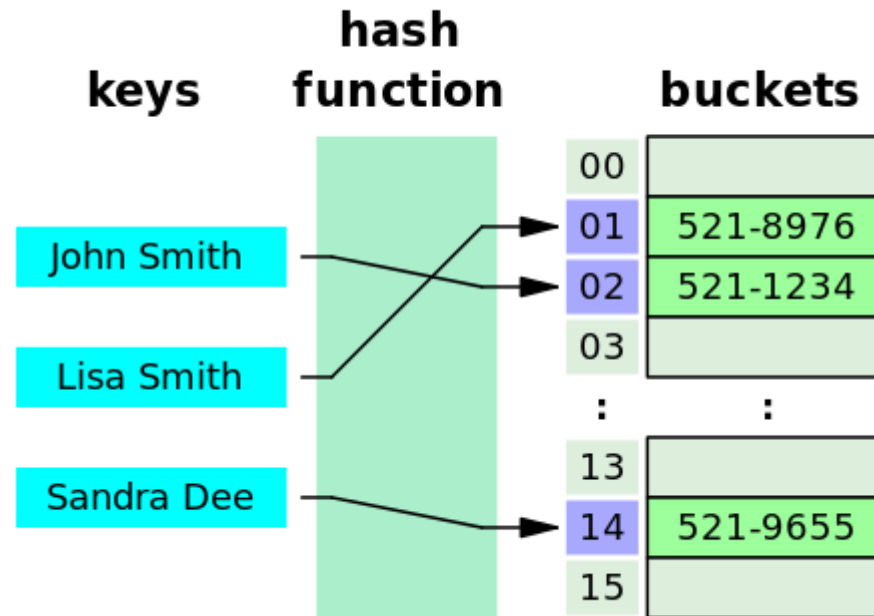
Why Hashing?

For a perfectly designed hash table, $O(1)$ operations are possible.

	Insertion	Deletion	Searching
Arrays	$O(n)$	$O(n)$	$O(n)$
Linked Lists	$O(1)$	$O(1)$	$O(n)$
Hash Tables	$O(1)$	$O(1)$	$O(1)$

Hash Tables

- An easy implementation of a hash table is an array.
- For each data point of the form *(key, value)*, map the *key* to an array *index*.
 - Use the key to generate an array index.
 - Store the data at that index.
- The buckets shown below represent the hash table positions you can insert in.



Direct Hashing

- In our example, the key (identikey) itself can be an array index.

Key	Array Index	Value
0	0	0, Carl Sagan, Sophomore, Physics, ..
1	1	1, Donald Knuth, Senior, Computer Science, ..
2	2	2, Paul Erdos, Freshman, Mathematics, ..
..
...

Hashing Functions

- The last example was not useful. What if you wanted to search for Paul's data?
 - search through every ID in the table until the data name is "Paul Erdos".
 - $O(n)$ search.
- Instead, make the name itself as the table lookup key.
 - Searching for the name becomes a direct lookup instead of scanning the data.
 - $O(1)$ lookup.
- The mapping function from key \rightarrow index will be a non-trivial function.

array index = hashfunc(key)

Designing Hashing Functions

- *hashfunc(key) = sum of all ascii characters in the key*

Key	Array Index	Value
..
Carl Sagan	876	0, Carl Sagan, Sophomore, Physics, ..
..
Paul Erdos	911	2, Paul Erdos, Freshman, Mathematics, ..
..
Donald Knuth	1116	1, Donald Knuth, Senior, Computer Science, ..
..

Notice that the array index can be pretty large depending on the name

Hashing Functions For Limited Table Size

- $\text{hashfunc}(\text{key}) = (\text{sum of all ascii characters}) \% \text{array_size}$
- For a limited hash table size of 100,

Key	Sum of ascii	Array Index	Value
..
Paul Erdos	911		
..	..		
Donald Knuth	1116		
..	..		
Carl Sagan	876		
..

Hashing Functions For Limited Table Size

- $\text{hashfunc}(\text{key}) = (\text{sum of all ascii characters}) \% \text{array_size}$
- For a limited hash table size of 100,

Key	Sum of ascii	Array Index	Value
..
Paul Erdos	911	11	2, Paul Erdos, Freshman, Mathematics, ..
..
Donald Knuth	1116	16	1, Donald Knuth, Senior, Computer Science, ..
..
Carl Sagan	876	76	0, Carl Sagan, Sophomore, Physics, ..
..

Are these hash function safe?

- Try inserting the following entry into the table:

Nikola Tesla	1111	11	13, Nikola Tesla, Junior, Everything, ..
---------------------	-------------	-----------	---

Are these hash function safe?


- Try inserting the following entry into the table:

Nikola Tesla	1111	11	13, Nikola Tesla, Junior, Everything, ..
---------------------	-------------	-----------	---

Key	Sum of ascii	Array Index	Value
..
Paul Erdos	911	11	2, Paul Erdos, Freshman, Mathematics, ..
..
Donald Knuth	1116	16	1, Donald Knuth, Senior, Computer Science, ..
..
Carl Sagan	876	76	0, Carl Sagan, Sophomore, Physics, ..
..

Collisions!

- Try inserting the following key into the table: “Nikola Tesla”

Nikola Tesla	1111	11	13, Nikola Tesla, Junior, Everything, ..	
Key	Sum of ascii	Array Index		Value
..
Paul Erdos	911	11		2, Paul Erdos, Freshman, Mathematics, ..
..
Donald Knuth	1116	16		1, Donald Knuth, Senior, Computer Science, ..
..
Carl Sagan	876	76		0, Carl Sagan, Sophomore, Physics, ..
..

How do we handle collisions?

- Collision: For the key x , if the entry $\text{hashfunc}(x)$ is already filled, insert into some other entry.
- This means a couple of things:
 - After indexing the hash table, check if the hash table entry matches the search.
 - A direct $O(1)$ lookup/insertion/deletion may not be possible.
 - But still better than $O(n)$ search as in an array.

Resolving Collisions

- We will look at two different methods:
 - Linear probing
 - Quadratic probing
 - Direct chaining
- Although, there are quite a lot of other innovative ways (judging by the names):
 - Coalesced hashing
 - Cuckoo hashing
 - Hopscotch hashing
 - Robin-hood hashing

Linear Probing

Insertion

```
insert(x, T=table) {  
  for(i = 0; i < T.size(); i++) {  
    if T[(h(x) + i) % T.size()].isEmpty()  
    {  
      T[(h(x) + i) % T.size()] = x;  
      return SUCCESS_INSERTION;  
    }  
  }  
  
  return ERROR_TABLE_FULL;  
}
```

Search (Item Exists)

```
search(x, T=table) {  
  for(i = 0; i < T.size(); i++) {  
    if T[(h(x) + i) % T.size()] == x  
    {  
      return TRUE; // or x.data  
    }  
  }  
  
  return FALSE;  
}
```

Linear Probing Example

hashfunc(key) = sum_ascii(key) % table_size

Key	sum_ascii	Index
Paul Erdos		
Donald Knuth		
Carl Sagan		
Nikola Tesla		
Isaac Newton		
Srinivasa Ramanujan		

Index	Value
0	
1	
2	
3	
4	
5	
6	

Linear Probing Example

$\text{hashfunc}(\text{key}) = \text{sum_ascii}(\text{key}) \% \text{table_size}$

Key	sum_ascii	Index
Paul Erdos	911	1
Donald Knuth		
Carl Sagan		
Nikola Tesla		
Isaac Newton		
Srinivasa Ramanujan		



Index	Value
0	
1	Paul Erdos, ...
2	
3	
4	
5	
6	

Linear Probing Example

hashfunc(key) = sum_ascii(key) % table_size

Key	sum_ascii	Index
Paul Erdos	911	1
Donald Knuth	1116	3
Carl Sagan		
Nikola Tesla		
Isaac Newton		
Srinivasa Ramanujan		

Index	Value
0	
1	Paul Erdos, ...
2	
3	
4	
5	
6	

Linear Probing Example

hashfunc(key) = sum_ascii(key) % table_size

Key	sum_ascii	Index
Paul Erdos	911	1
Donald Knuth	1116	3
Carl Sagan		
Nikola Tesla		
Isaac Newton		
Srinivasa Ramanujan		



Index	Value
0	
1	Paul Erdos, ...
2	
3	Donald Knuth, ...
4	
5	
6	

Linear Probing Example

$\text{hashfunc}(\text{key}) = \text{sum_ascii}(\text{key}) \% \text{table_size}$

Key	sum_ascii	Index
Paul Erdos	911	1
Donald Knuth	1116	3
Carl Sagan	876	1
Nikola Tesla		
Isaac Newton		
Srinivasa Ramanujan		



Index	Value
0	
1	Paul Erdos, ...
2	
3	Donald Knuth, ...
4	
5	
6	

Linear Probing Example

$\text{hashfunc}(\text{key}) = \text{sum_ascii}(\text{key}) \% \text{table_size}$

Key	sum_ascii	Index
Paul Erdos	911	1
Donald Knuth	1116	3
Carl Sagan	876	1
Nikola Tesla		
Isaac Newton		
Srinivasa Ramanujan		



Index	Value
0	
1	Paul Erdos, ...
2	Carl Sagan, ...
3	Donald Knuth, ...
4	
5	
6	

Linear Probing Example

hashfunc(key) = sum_ascii(key) % table_size

Key	sum_ascii	Index
Paul Erdos	911	1
Donald Knuth	1116	3
Carl Sagan	876	1
Nikola Tesla		
Isaac Newton		
Srinivasa Ramanujan	1901	4

Index	Value
0	
1	Paul Erdos, ...
2	Carl Sagan, ...
3	Donald Knuth, ...
4	
5	
6	

Linear Probing Example

$\text{hashfunc}(\text{key}) = \text{sum_ascii}(\text{key}) \% \text{table_size}$

Key	sum_ascii	Index
Paul Erdos	911	1
Donald Knuth	1116	3
Carl Sagan	876	1
Nikola Tesla		
Isaac Newton		
Srinivasa Ramanujan	1901	4

Index	Value
0	
1	Paul Erdos, ...
2	Carl Sagan, ...
3	Donald Knuth, ...
4	Srinivasa Ramanujan,
5	
6	



Linear Probing Example

$\text{hashfunc}(\text{key}) = \text{sum_ascii}(\text{key}) \% \text{table_size}$

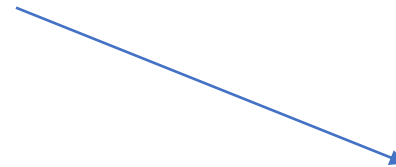
Key	sum_ascii	Index
Paul Erdos	911	1
Donald Knuth	1116	3
Carl Sagan	876	1
Nikola Tesla	1111	5
Isaac Newton		
Srinivasa Ramanujan	1901	4

Index	Value
0	
1	Paul Erdos, ...
2	Carl Sagan, ...
3	Donald Knuth, ...
4	Srinivasa Ramanujan,
5	
6	

Linear Probing Example

$\text{hashfunc}(\text{key}) = \text{sum_ascii}(\text{key}) \% \text{table_size}$

Key	sum_ascii	Index
Paul Erdos	911	1
Donald Knuth	1116	3
Carl Sagan	876	1
Nikola Tesla	1111	5
Isaac Newton		
Srinivasa Ramanujan	1901	4



Index	Value
0	
1	Paul Erdos, ...
2	Carl Sagan, ...
3	Donald Knuth, ...
4	Srinivasa Ramanujan,
5	Nikola Tesla, ...
6	

Linear Probing Example

$\text{hashfunc}(\text{key}) = \text{sum_ascii}(\text{key}) \% \text{table_size}$

Key	sum_ascii	Index
Paul Erdos	911	1
Donald Knuth	1116	3
Carl Sagan	876	1
Nikola Tesla	1111	5
Isaac Newton	1116	3
Srinivasa Ramanujan	1901	4

Index	Value
0	
1	Paul Erdos, ...
2	Carl Sagan, ...
3	Donald Knuth, ...
4	Srinivasa Ramanujan,
5	Nikola Tesla, ...
6	

Linear Probing Example

$\text{hashfunc}(\text{key}) = \text{sum_ascii}(\text{key}) \% \text{table_size}$

Key	sum_ascii	Index
Paul Erdos	911	1
Donald Knuth	1116	3
Carl Sagan	876	1
Nikola Tesla	1111	5
Isaac Newton	1116	3
Srinivasa Ramanujan	1901	4



Index	Value
0	
1	Paul Erdos, ...
2	Carl Sagan, ...
3	Donald Knuth, ...
4	Srinivasa Ramanujan,
5	Nikola Tesla, ...
6	

Linear Probing Example

$\text{hashfunc}(\text{key}) = \text{sum_ascii}(\text{key}) \% \text{table_size}$

Key	sum_ascii	Index
Paul Erdos	911	1
Donald Knuth	1116	3
Carl Sagan	876	1
Nikola Tesla	1111	5
Isaac Newton	1116	3
Srinivasa Ramanujan	1901	4



Index	Value
0	
1	Paul Erdos, ...
2	Carl Sagan, ...
3	Donald Knuth, ...
4	Srinivasa Ramanujan,
5	Nikola Tesla, ...
6	

Linear Probing Example

$\text{hashfunc}(\text{key}) = \text{sum_ascii}(\text{key}) \% \text{table_size}$

Key	sum_ascii	Index
Paul Erdos	911	1
Donald Knuth	1116	3
Carl Sagan	876	1
Nikola Tesla	1111	5
Isaac Newton	1116	3
Srinivasa Ramanujan	1901	4



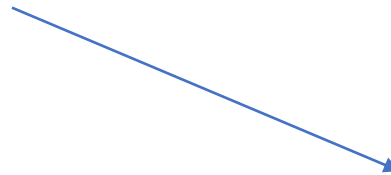
Index	Value
0	
1	Paul Erdos, ...
2	Carl Sagan, ...
3	Donald Knuth, ...
4	Srinivasa Ramanujan,
5	Nikola Tesla, ...
6	

Linear Probing Example

$\text{hashfunc}(\text{key}) = \text{sum_ascii}(\text{key}) \% \text{table_size}$

Key	sum_ascii	Index
Paul Erdos	911	1
Donald Knuth	1116	3
Carl Sagan	876	1
Nikola Tesla	1111	5
Isaac Newton	1116	3
Srinivasa Ramanujan	1901	4

Index	Value
0	
1	Paul Erdos, ...
2	Carl Sagan, ...
3	Donald Knuth, ...
4	Srinivasa Ramanujan,
5	Nikola Tesla, ...
6	Isaac Newton, ...



Issues with Linear Probing

Clustering

Items tend to cluster together in the table (*retrieval time*)

If clusters merge, certain portions are densely occupied while others are relatively empty (*wasted space*)

Role of step_size

Cannot resolve clustering by simply increasing step_size

Choose step_size such that **step_size** and **table.size()** are co-primes

But **table.size()** can't change

Direct Chaining

- The hash table maintains a Linked List at each entry of the table.
- Insert item x in the Linked List at index $h(x)$ of the table
- Insert item at head of Linked List (*Why?*)
- Search is $O(N)$, N : length of the Linked List

Direct Chaining

Insertion

```
insert(x, T=table) {  
    temp = createNode(x);  
    temp->next = T[h(x)]->head;  
    T[h(x)]->head = temp;  
  
    return true;  
}
```

Search

```
search(x, T=table) {  
    temp = T[h(x)]->head;  
    while(temp != NULL) {  
        if(temp->data == x) {  
            return temp;  
        }  
    }  
    return temp;  
}
```

Direct Chaining Example

$\text{hashfunc}(\text{key}) = \text{sum_ascii}(\text{key}) \% \text{table_size}$

Key	sum_ascii	Index
Paul Erdos	911	1
Donald Knuth	1116	3
Carl Sagan	876	1
Nikola Tesla	1111	5
Isaac Newton	1116	3
Srinivasa Ramanujan	1901	4

Index	Value
0	
1	→ PE →
2	
3	
4	
5	
6	

Direct Chaining Example

$hashfunc(key) = sum_ascii(key) \% table_size$

Key	sum_ascii	Index
Paul Erdos	911	1
Donald Knuth	1116	3
Carl Sagan	876	1
Nikola Tesla	1111	5
Isaac Newton	1116	3
Srinivasa Ramanujan	1901	4

Index	Value
0	
1	→ PE →
2	
3	→ DK →
4	
5	
6	

Direct Chaining Example

$hashfunc(key) = sum_ascii(key) \% table_size$

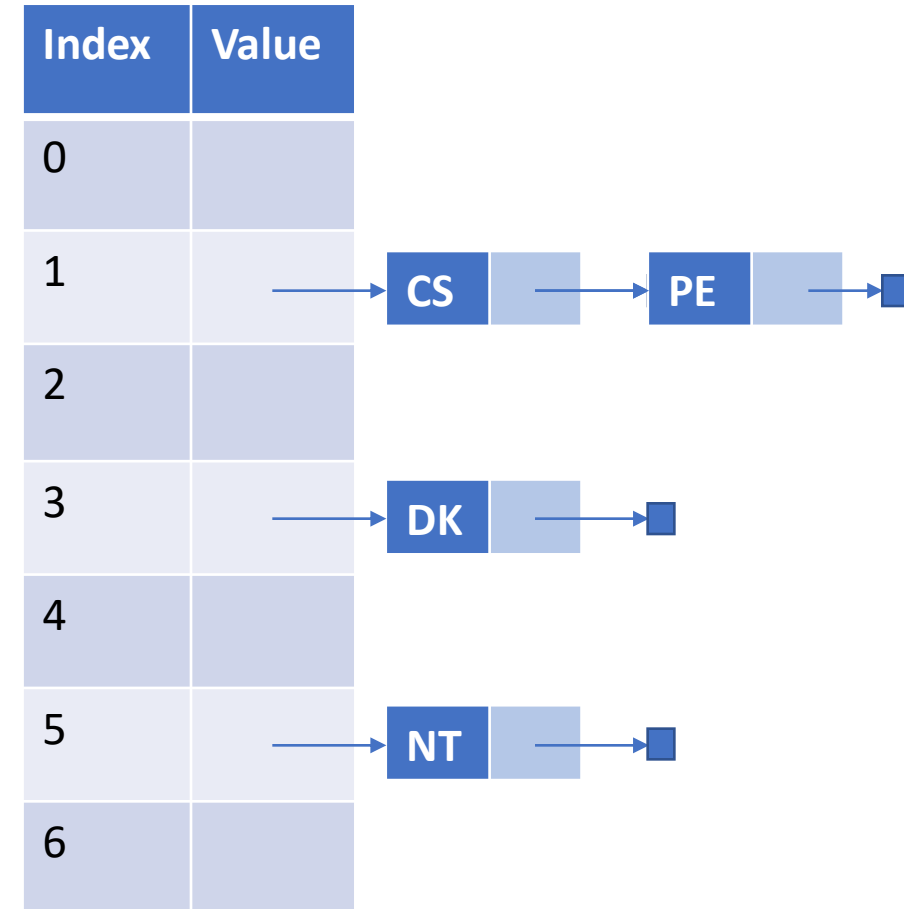
Key	sum_ascii	Index
Paul Erdos	911	1
Donald Knuth	1116	3
Carl Sagan	876	1
Nikola Tesla	1111	5
Isaac Newton	1116	3
Srinivasa Ramanujan	1901	4

Index	Value
0	
1	→ CS → PE →
2	
3	→ DK →
4	
5	
6	

Direct Chaining Example

$hashfunc(key) = sum_ascii(key) \% table_size$

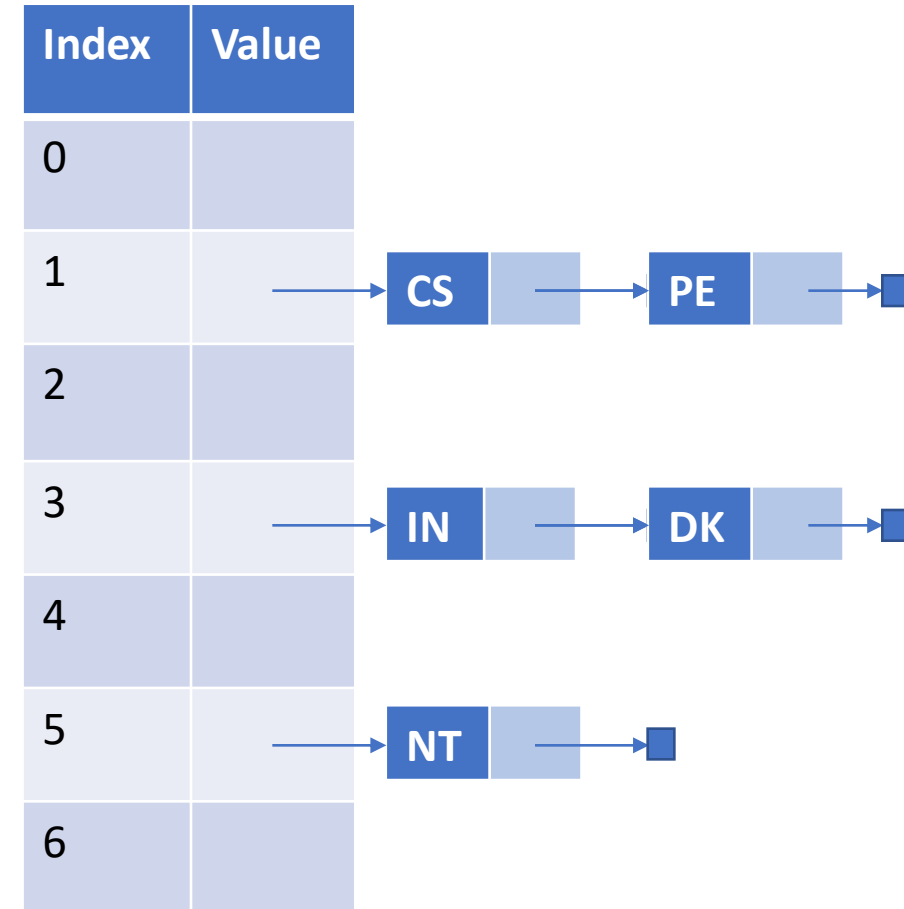
Key	sum_ascii	Index
Paul Erdos	911	1
Donald Knuth	1116	3
Carl Sagan	876	1
Nikola Tesla	1111	5
Isaac Newton	1116	3
Srinivasa Ramanujan	1901	4



Direct Chaining Example

$hashfunc(key) = sum_ascii(key) \% table_size$

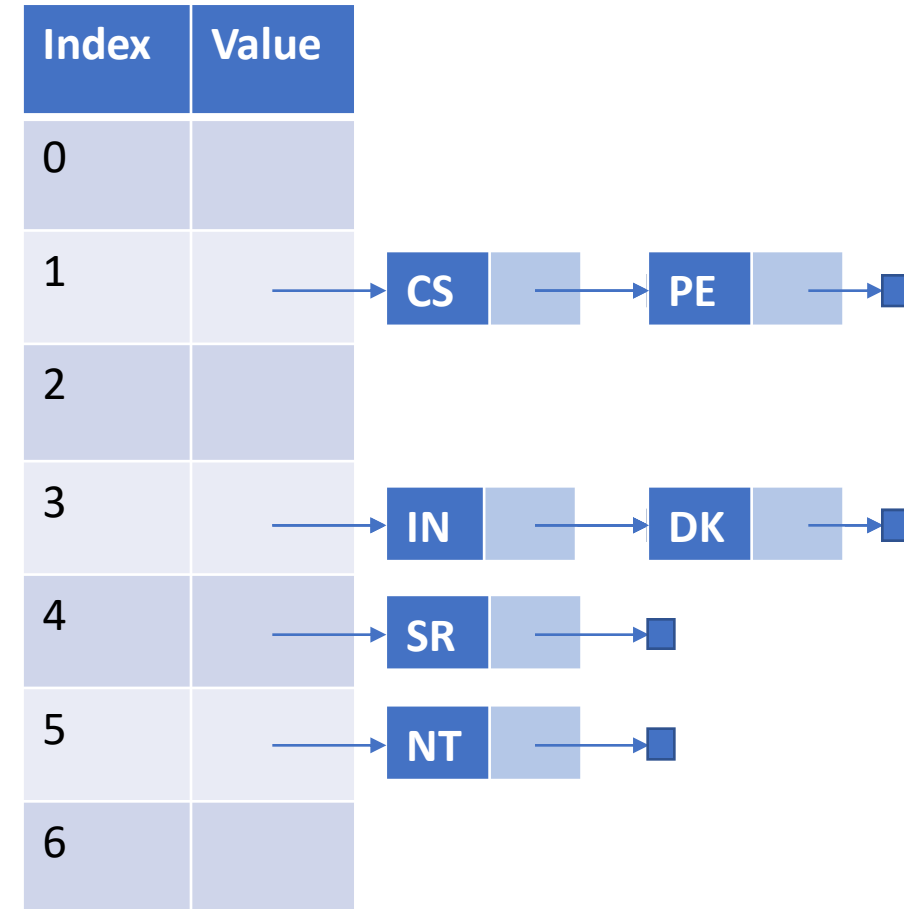
Key	sum_ascii	Index
Paul Erdos	911	1
Donald Knuth	1116	3
Carl Sagan	876	1
Nikola Tesla	1111	5
Isaac Newton	1116	3
Srinivasa Ramanujan	1901	4



Direct Chaining Example

$hashfunc(key) = sum_ascii(key) \% table_size$

Key	sum_ascii	Index
Paul Erdos	911	1
Donald Knuth	1116	3
Carl Sagan	876	1
Nikola Tesla	1111	5
Isaac Newton	1116	3
Srinivasa Ramanujan	1901	4



Load Factor: How is your Hash Table Doing?

- If there are n entries in the hash table, and k is the table size, the load factor is defined as:

$$\text{Load factor} = \frac{n}{k}$$

- As load factor grows, the $O(1)$ property of the hash table may no longer hold.
- Used for deciding whether to resize the hash table.
- Other statistics: Measuring the variance of entries across the table slots.

Example Problem

Given a string S , find the first character that repeats (l-to-r).

In “abcdabcdabcd”, ‘a’ repeats first

Approach #1: Use an array

```
for(i = 0; i < S.size() - 1; i++)  
    for(j= i + 1; j < S.size(); j++)  
        if(S[i] == S[j]) return S[i];
```

Time: **$O(N^2)$**

Example Problem

Approach #2: Use Hashing

Maintain an array `A[256]`

// 1 place for each ASCII character code

Initialize `A[i] = 0`, for all `i`

```
for(i = 0; i < S.size(); i++)
```

```
    A[(int) S[i]]++;
```

```
    if(A[(int) S[i]] > 1) return S[i];
```

Time: **$O(N)$**

Example Problem

Approach #2: Use Hashing

Key: character

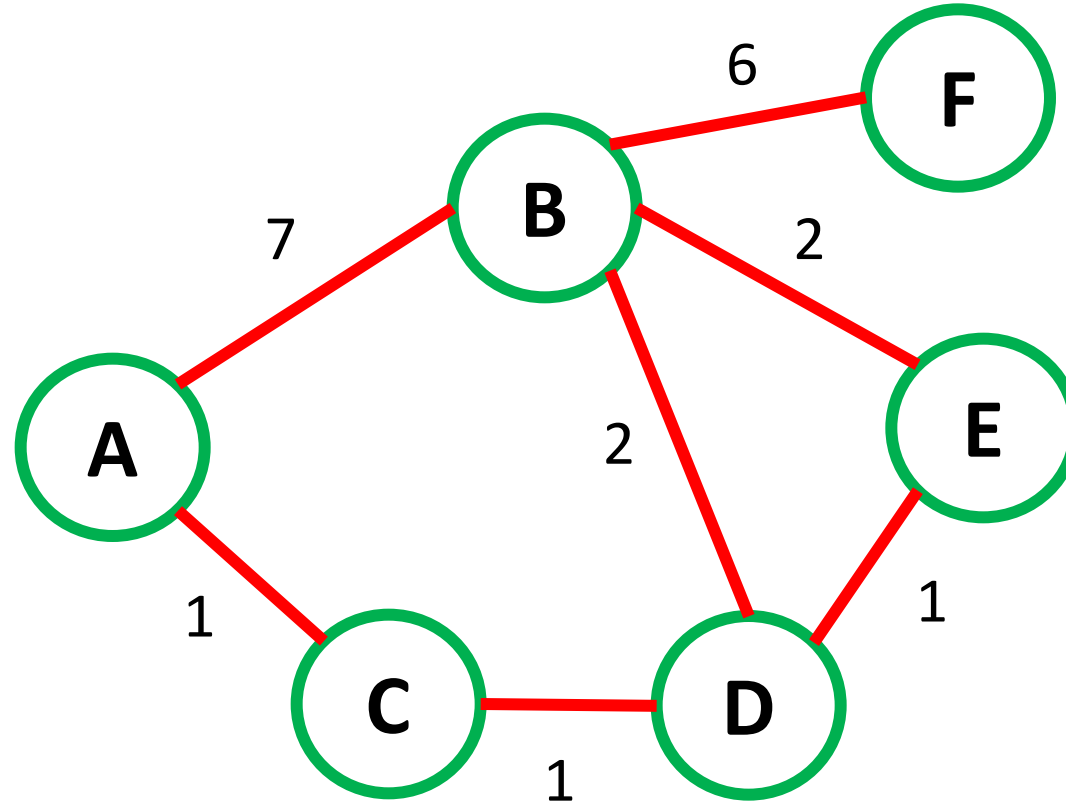
Value: count of character

By maintaining a table and a map, we were able to reduce the retrieval to $O(1)$ and thus the overall time complexity to $O(N)$

Midterm Review

Dijkstra's algorithm

- Finding the shortest path distance in weighted graphs.



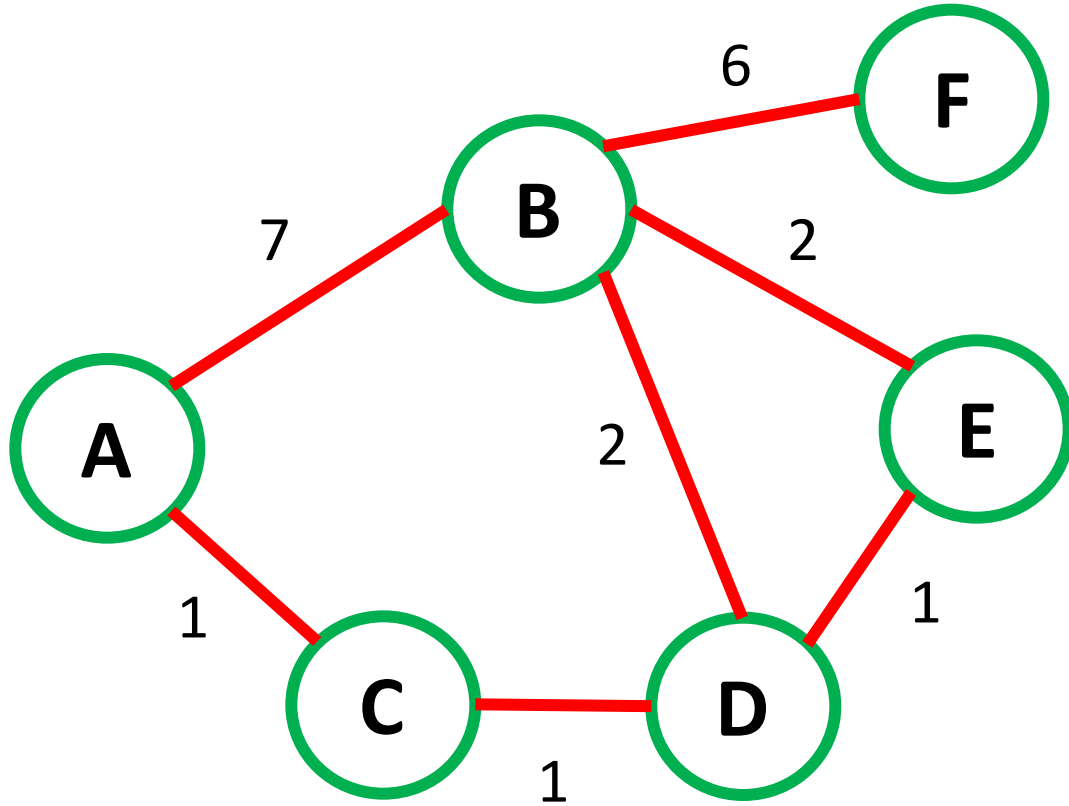
Dijkstra's algorithm (initialization)

- Mark all vertices as unvisited.
- Set the distance of each vertex from the source to ∞ .
 - For the source, set this distance to 0.
- Create a vector of all unvisited nodes in the graph.

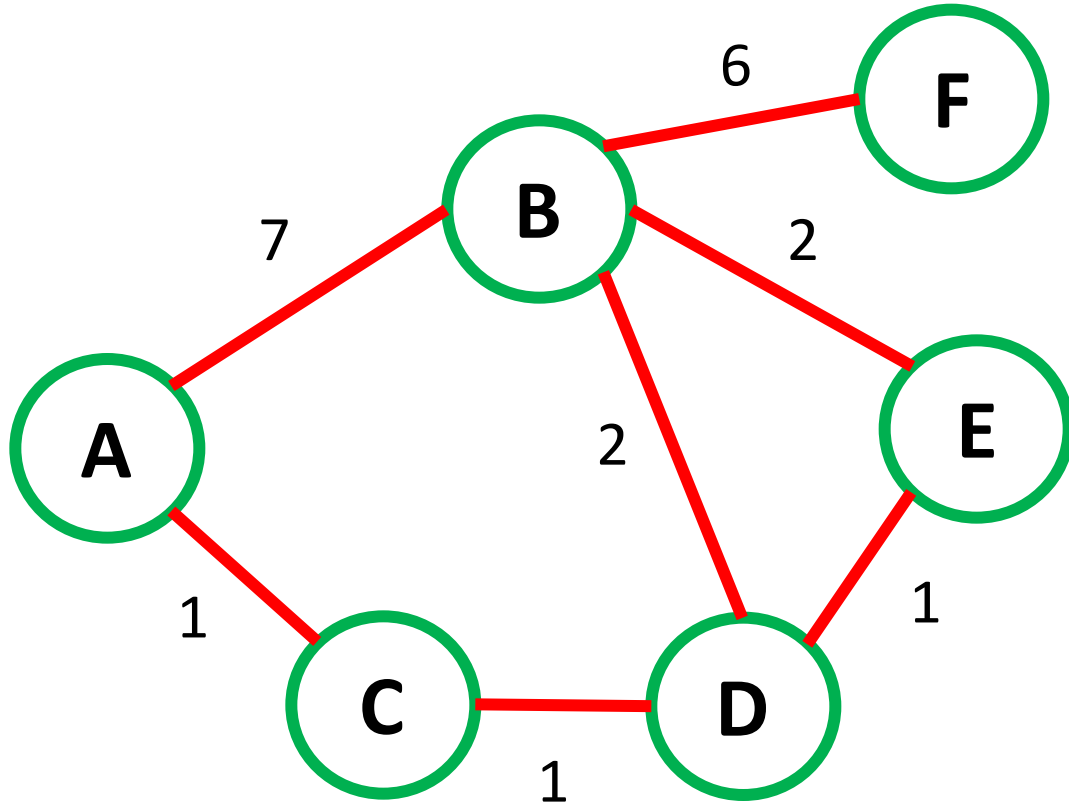
Dijkstra's algorithm (main loop)

- While the vector of unvisited nodes is not empty,
 - Find the node “sm” with the least distance from the source.
 - Remove “sm” from the list of unvisited nodes.
 - For each neighbor “nbr” of “sm”,
 - If $\text{nbr} \rightarrow \text{distance} > \text{sm} \rightarrow \text{distance} + \text{nbr} \rightarrow \text{adj}[\text{sm}].\text{weight}$
 $\text{nbr} \rightarrow \text{distance} = \text{sm} \rightarrow \text{distance} + \text{nbr} \rightarrow \text{adj}[\text{sm}].\text{weight}$

Dijkstra's algorithm



Dijkstra's algorithm



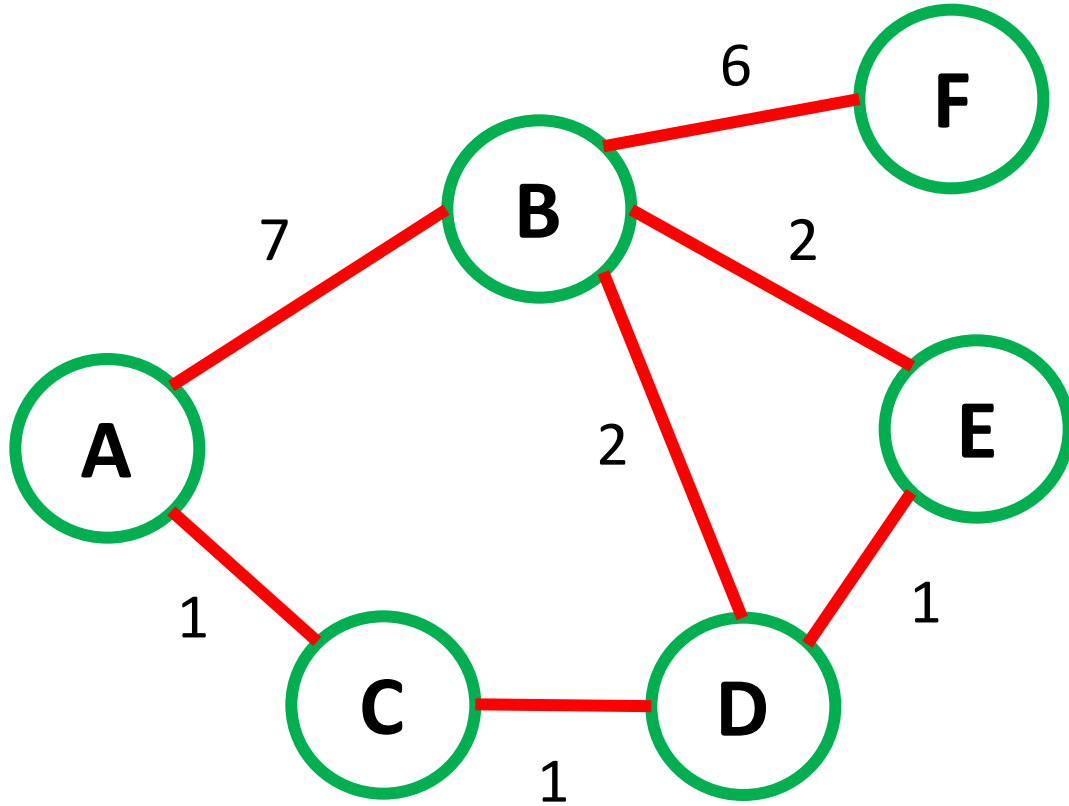
Unvisited Vertices

A	B	C	D	E	F
---	---	---	---	---	---

∞	∞	∞	∞	∞	∞
----------	----------	----------	----------	----------	----------

Distance from source

Dijkstra's algorithm



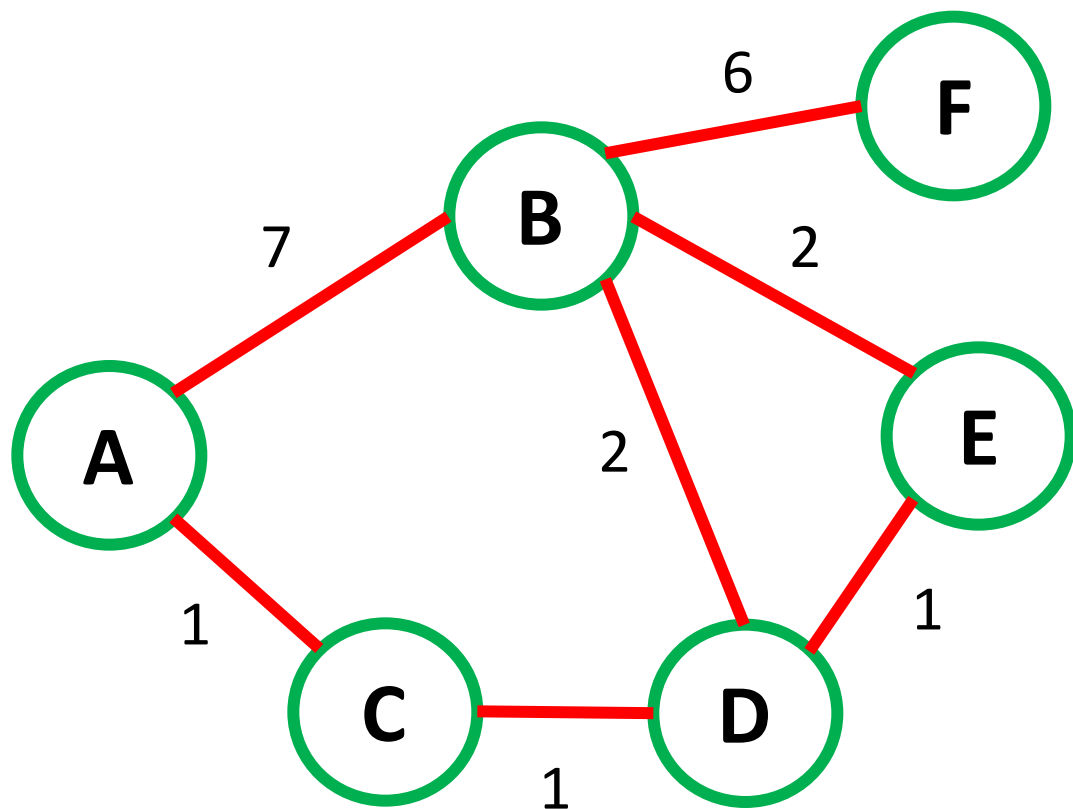
Unvisited Vertices

A	B	C	D	E	F
---	---	---	---	---	---

0	∞	∞	∞	∞	∞
---	----------	----------	----------	----------	----------

Distance from source

Dijkstra's algorithm



0	∞	∞	∞	∞	∞
---	----------	----------	----------	----------	----------

Distance Vector

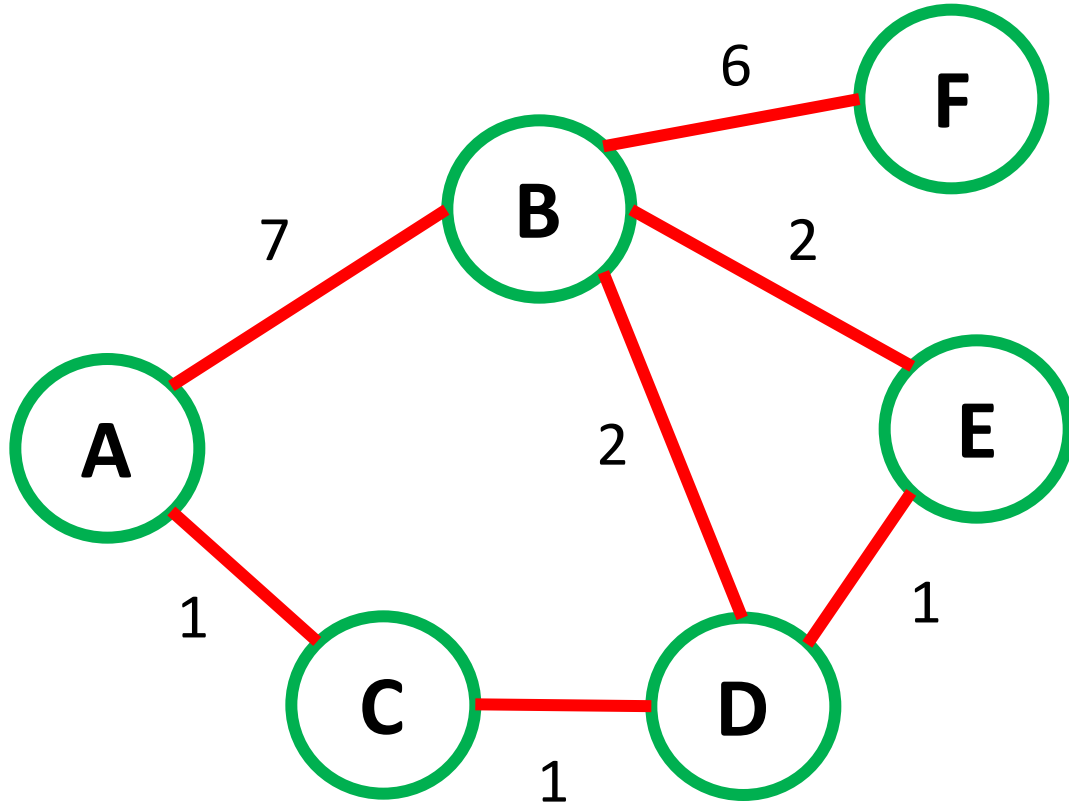
Unvisited Vertices

A	B	C	D	E	F
	7	1			
0	∞	∞	∞	∞	∞

Distance from source

Vertex with smallest distance: **A**

Dijkstra's algorithm



0	7	1	∞	∞	∞
---	---	---	----------	----------	----------

Unvisited Vertices

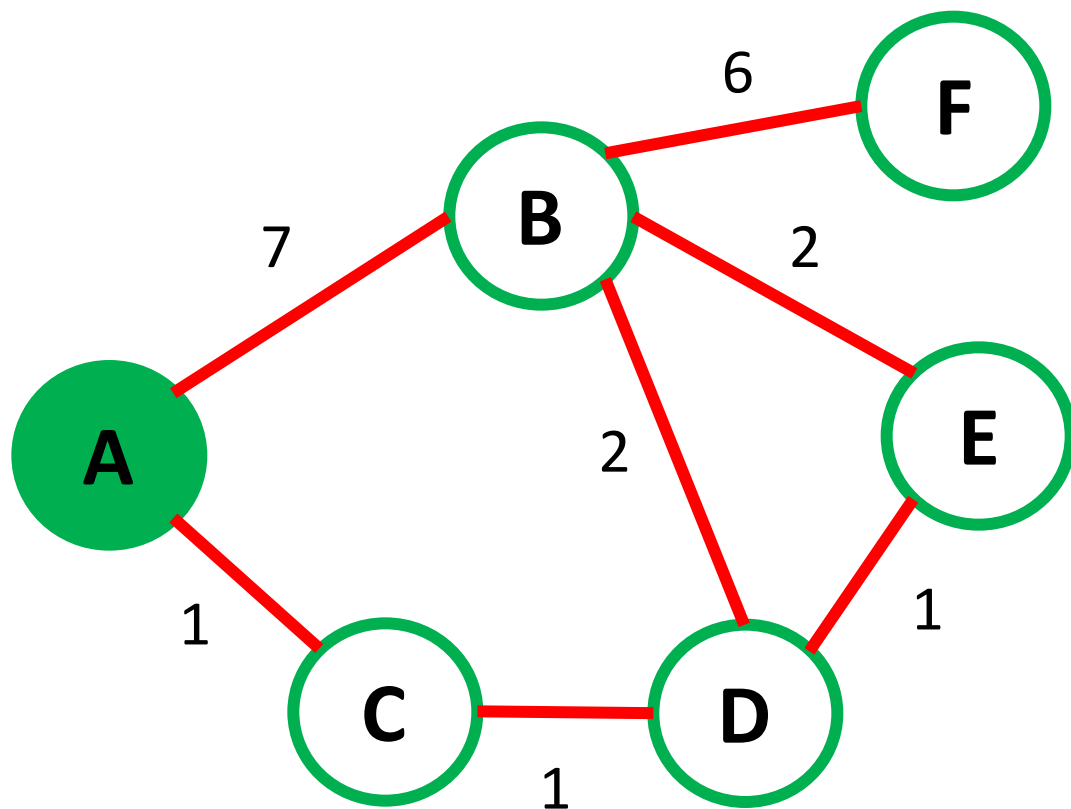
A	B	C	D	E	F
---	---	---	---	---	---

0	7	1	∞	∞	∞
---	---	---	----------	----------	----------

Distance from source

Vertex with smallest distance: **A**
Update distances of neighbors: **B, C**

Dijkstra's algorithm



0	7	1	∞	∞	∞
---	---	---	----------	----------	----------

Unvisited Vertices

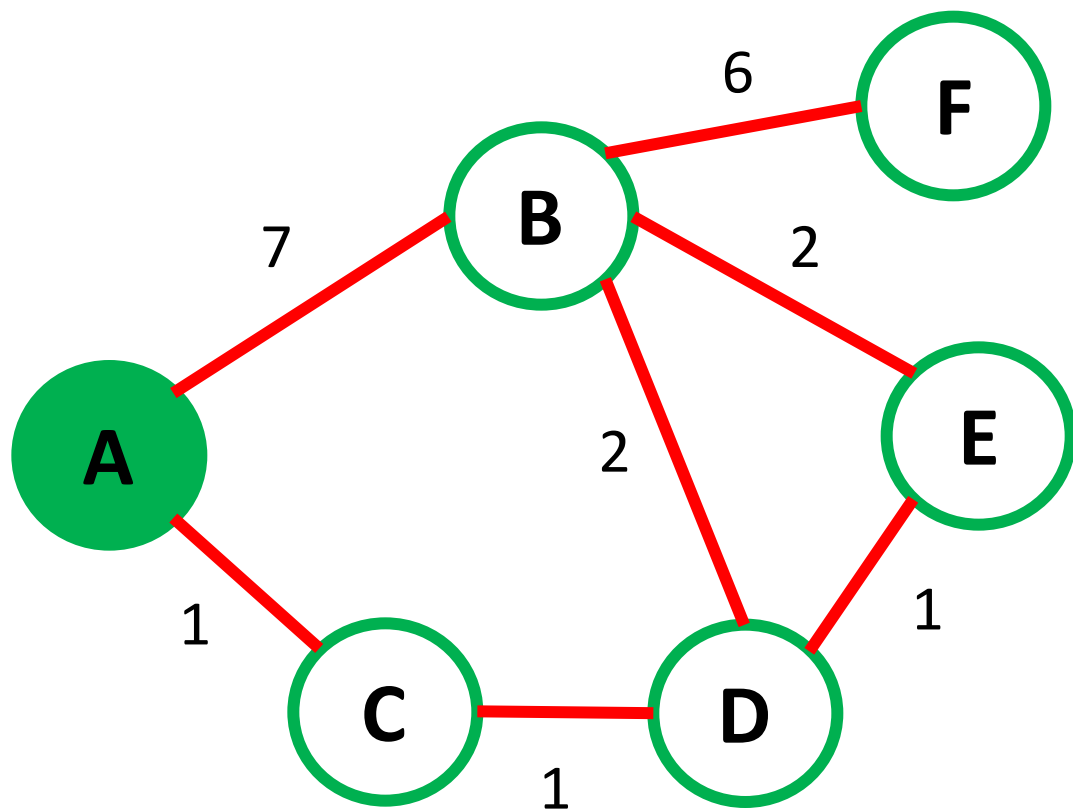
	B	C	D	E	F
--	---	---	---	---	---

	7	1	∞	∞	∞
--	---	---	----------	----------	----------

Distance from source

Vertex with smallest distance: **A**
Remove from unvisited: **A**

Dijkstra's algorithm



Unvisited Vertices

	B	C	D	E	F
--	---	---	---	---	---

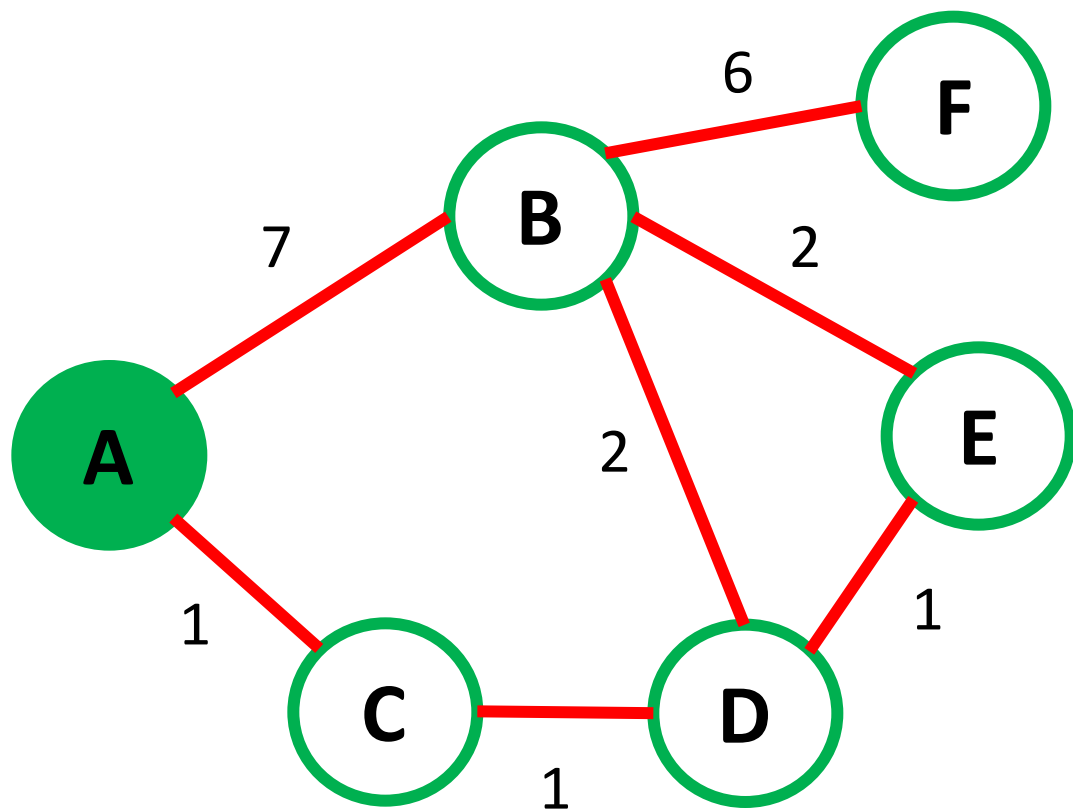
	7	1	∞	∞	∞
--	---	---	----------	----------	----------

Distance from source

Vertex with smallest distance:

0	7	1	∞	∞	∞
---	---	---	----------	----------	----------

Dijkstra's algorithm



0	7	1	∞	∞	∞
---	---	---	----------	----------	----------

Unvisited Vertices

	B	C	D	E	F
--	---	---	---	---	---

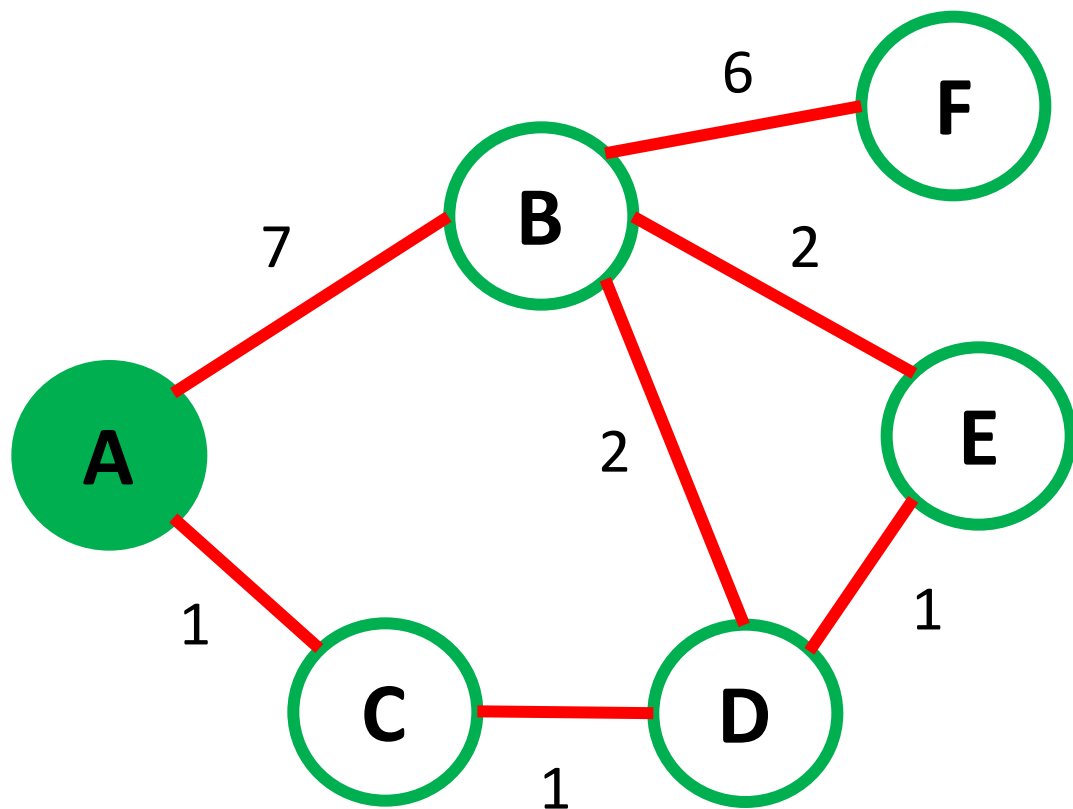
2

	7	1	∞	∞	∞
--	---	---	----------	----------	----------

Distance from source

Vertex with smallest distance: **C**

Dijkstra's algorithm



0	7	1	2	∞	∞
---	---	---	---	----------	----------

Unvisited Vertices

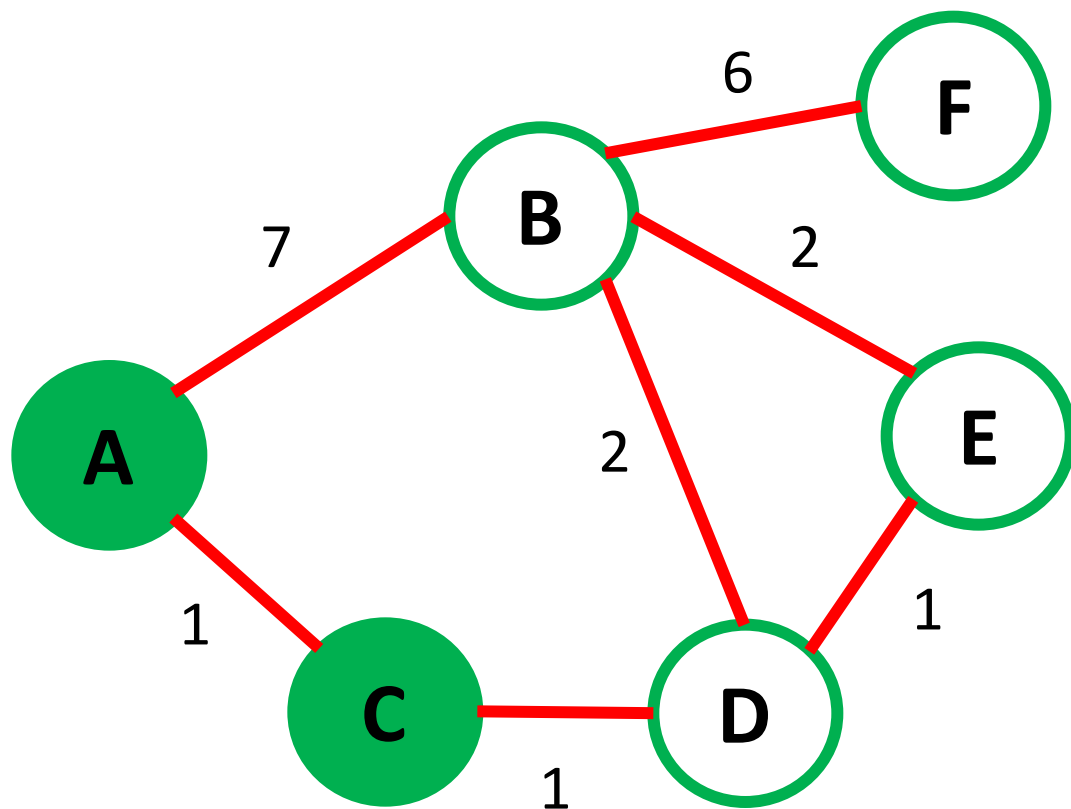
	B	C	D	E	F
--	---	---	---	---	---

	7	1	2	∞	∞
--	---	---	---	----------	----------

Distance from source

Vertex with smallest distance: **C**
Update distances of neighbors: **D**

Dijkstra's algorithm



0	7	1	2	∞	∞
---	---	---	---	----------	----------

Unvisited Vertices

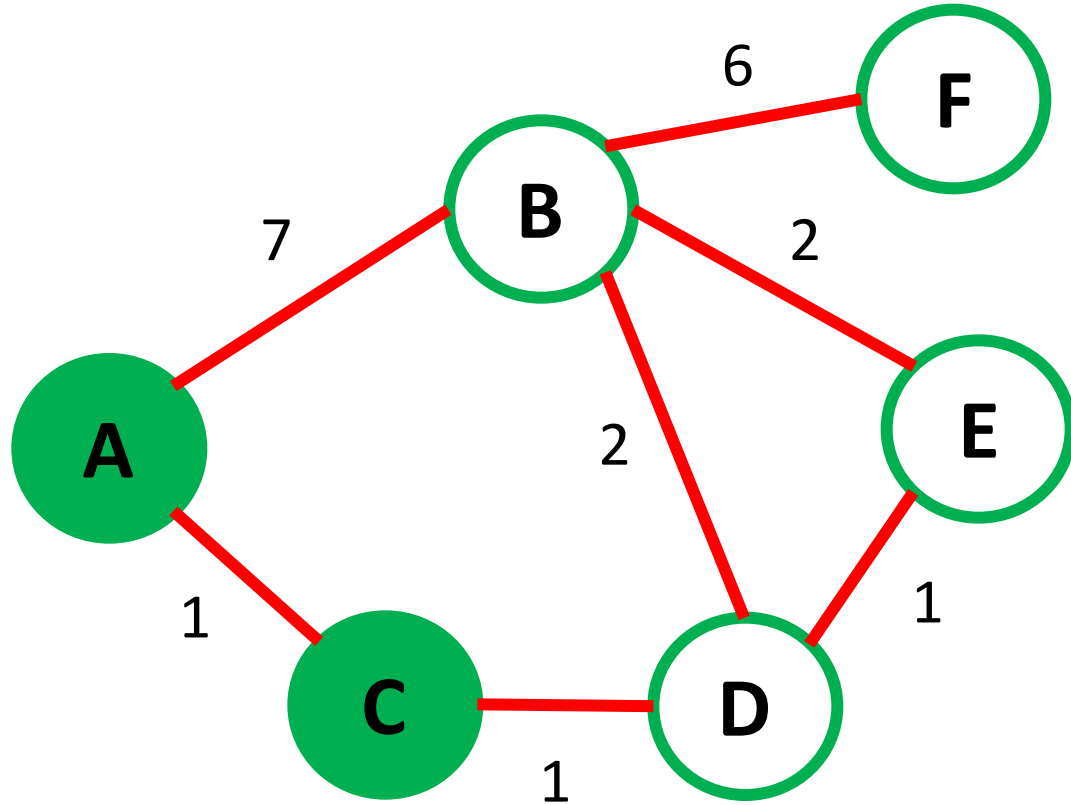
	B		D	E	F
--	---	--	---	---	---

	7		2	∞	∞
--	---	--	---	----------	----------

Distance from source

Vertex with smallest distance: **C**
Remove from unvisited: **C**

Dijkstra's algorithm



0	7	1	2	∞	∞
---	---	---	---	----------	----------

Unvisited Vertices

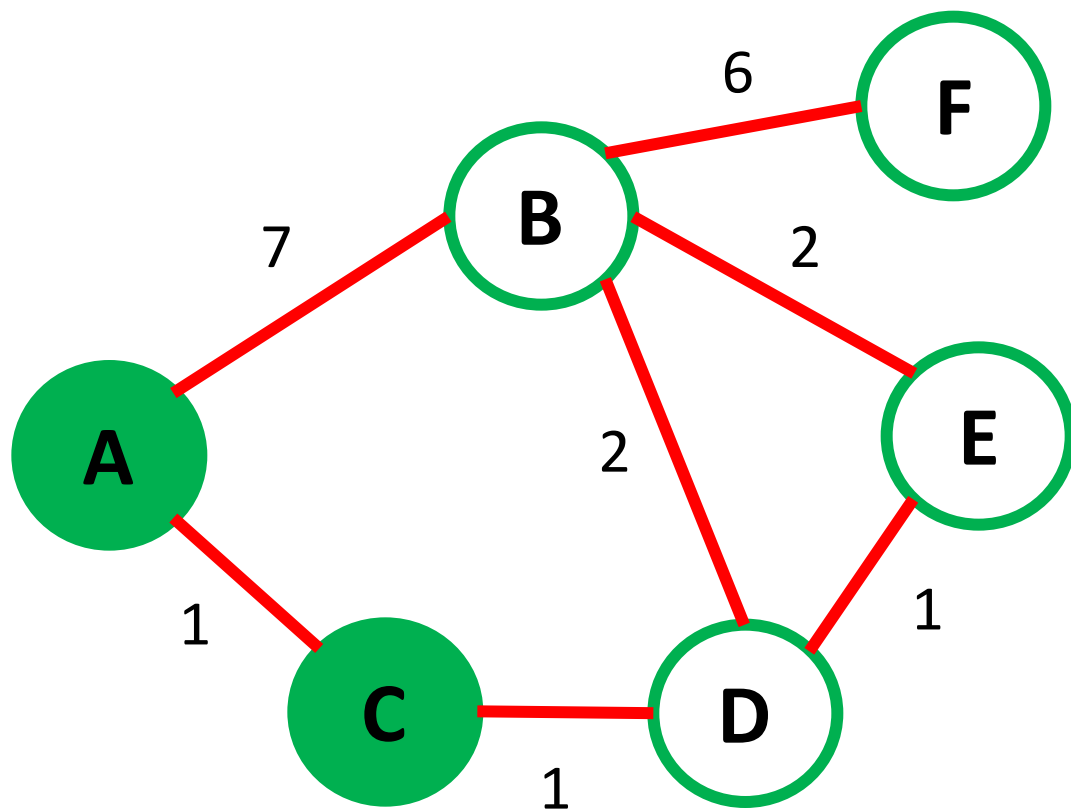
	B		D	E	F
--	---	--	---	---	---

	7		2	∞	∞
--	---	--	---	----------	----------

Distance from source

Vertex with smallest distance:

Dijkstra's algorithm



0	7	1	2	3	∞
---	---	---	---	---	----------

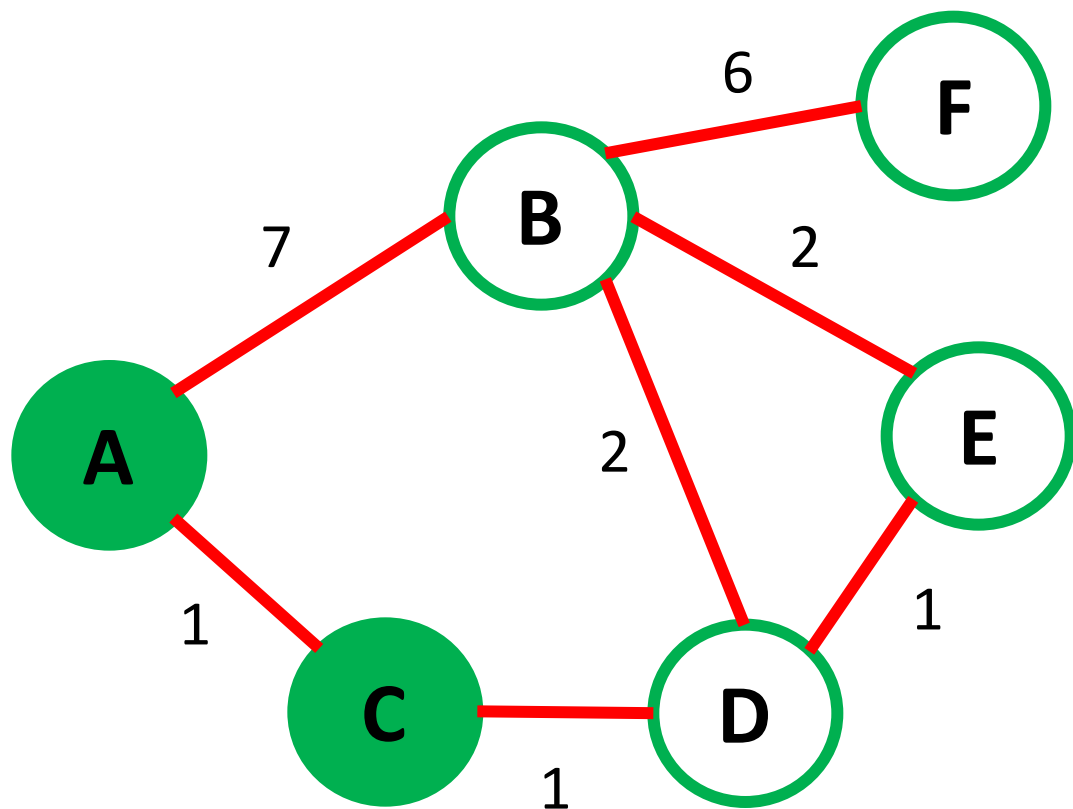
Unvisited Vertices

	B		D	E	F
	4			3	
	7		2	∞	∞

Distance from source

Vertex with smallest distance: **D**

Dijkstra's algorithm



0	4	1	2	3	∞
---	---	---	---	---	----------

Unvisited Vertices

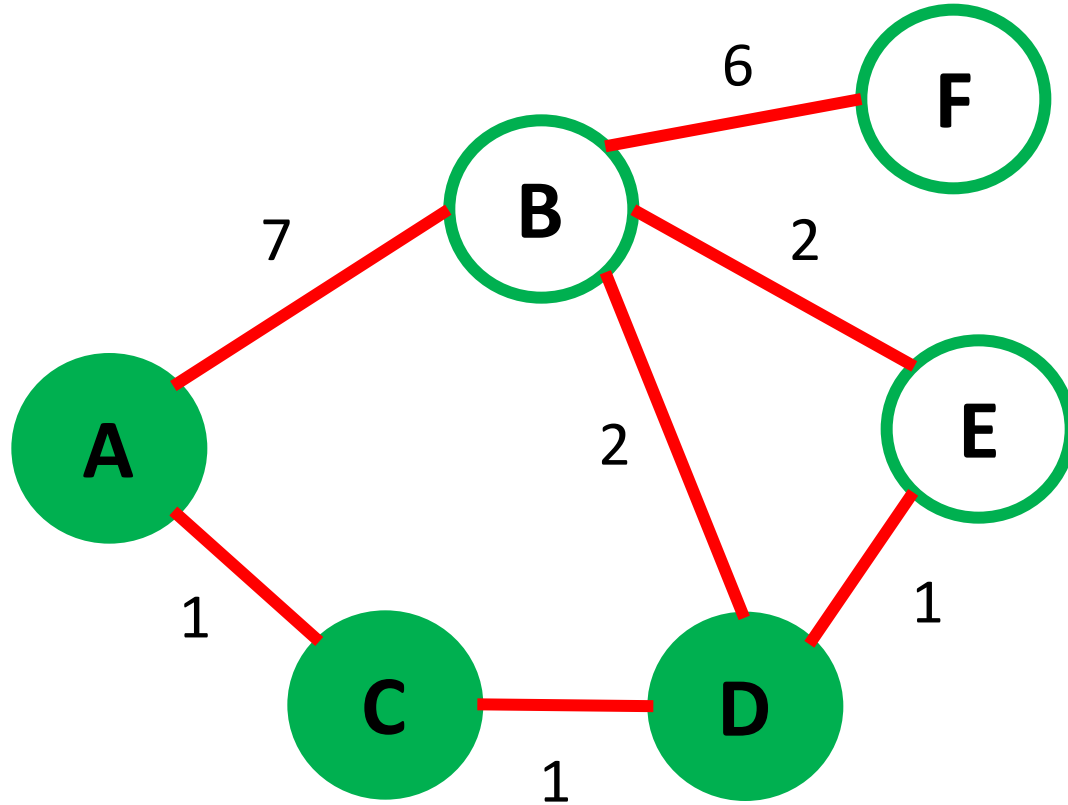
	B		D	E	F
--	---	--	---	---	---

	4		2	3	∞
--	---	--	---	---	----------

Distance from source

Vertex with smallest distance: **D**
Update distances of neighbors: **B, E**

Dijkstra's algorithm



0	4	1	2	3	∞
---	---	---	---	---	----------

Unvisited Vertices

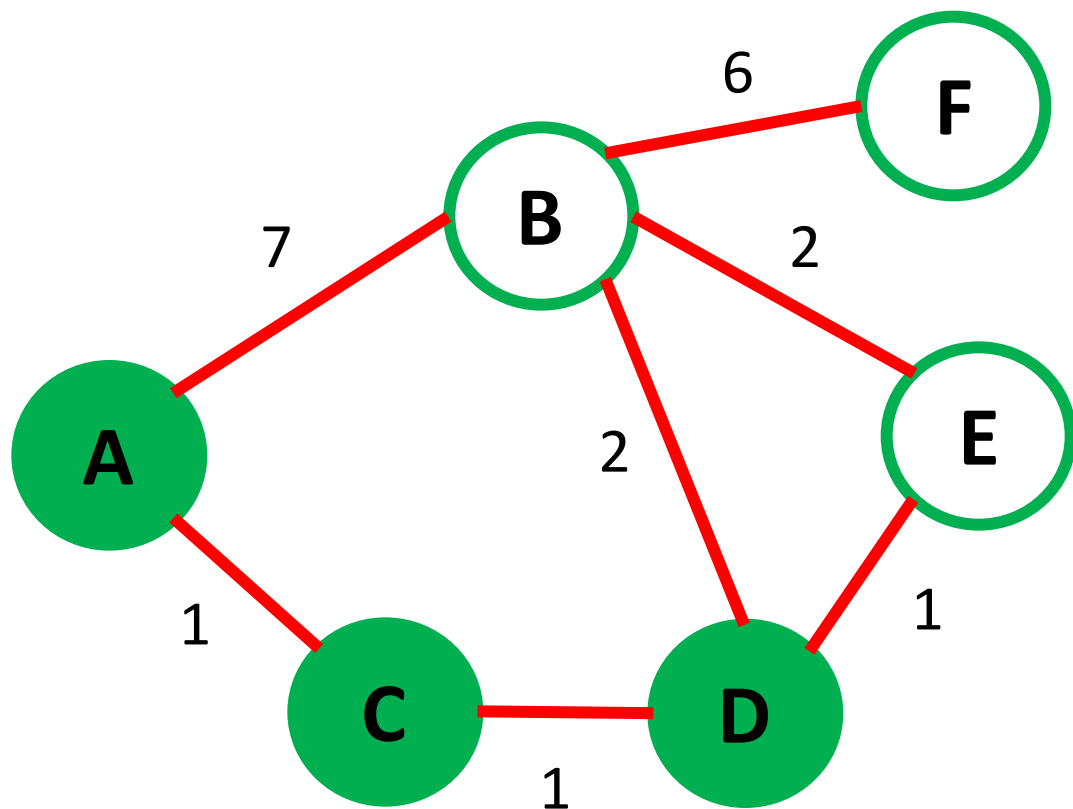
	B			E	F
--	---	--	--	---	---

	4			3	∞
--	---	--	--	---	----------

Distance from source

Vertex with smallest distance: **D**
Remove from unvisited: **D**

Dijkstra's algorithm



0	4	1	2	3	∞
---	---	---	---	---	----------

Unvisited Vertices

	B			E	F
--	---	--	--	---	---

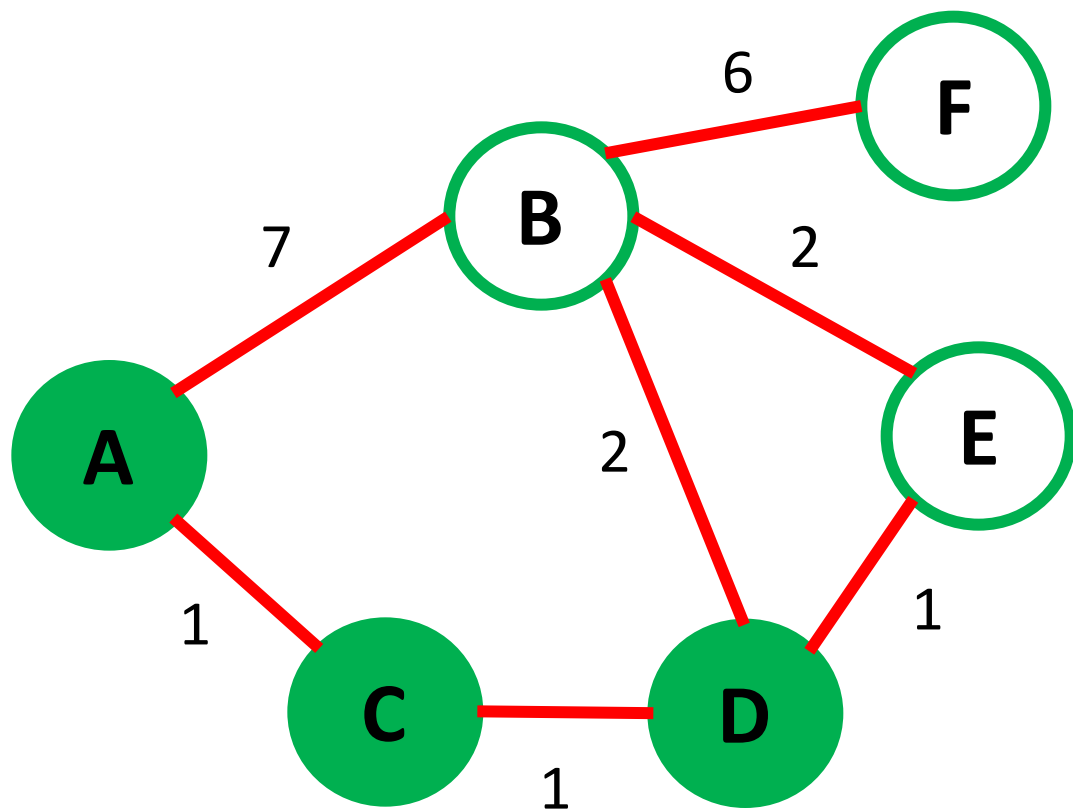
5

	4			3	∞
--	---	--	--	---	----------

Distance from source

Vertex with smallest distance: **E**

Dijkstra's algorithm



0	4	1	2	3	∞
---	---	---	---	---	----------

Unvisited Vertices

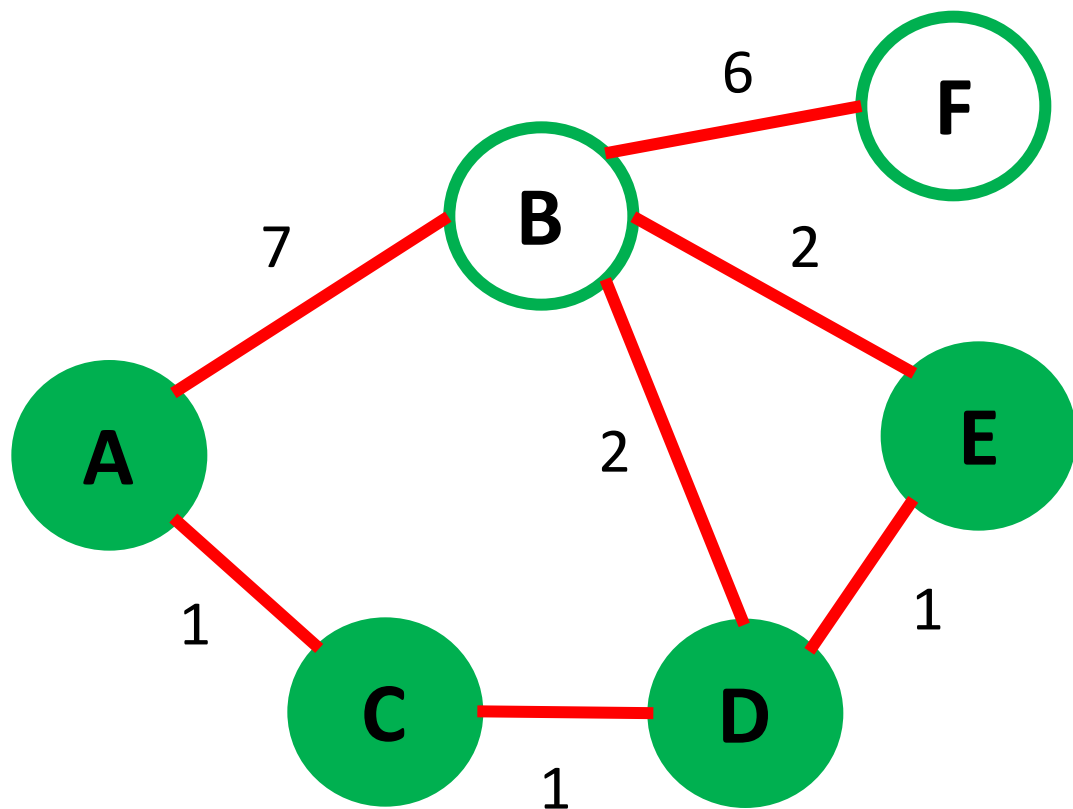
	B			E	F
--	---	--	--	---	---

	4			3	∞
--	---	--	--	---	----------

Distance from source

Vertex with smallest distance: **E**
Update distances of neighbors: **B**

Dijkstra's algorithm



0	4	1	2	3	∞
---	---	---	---	---	----------

Unvisited Vertices

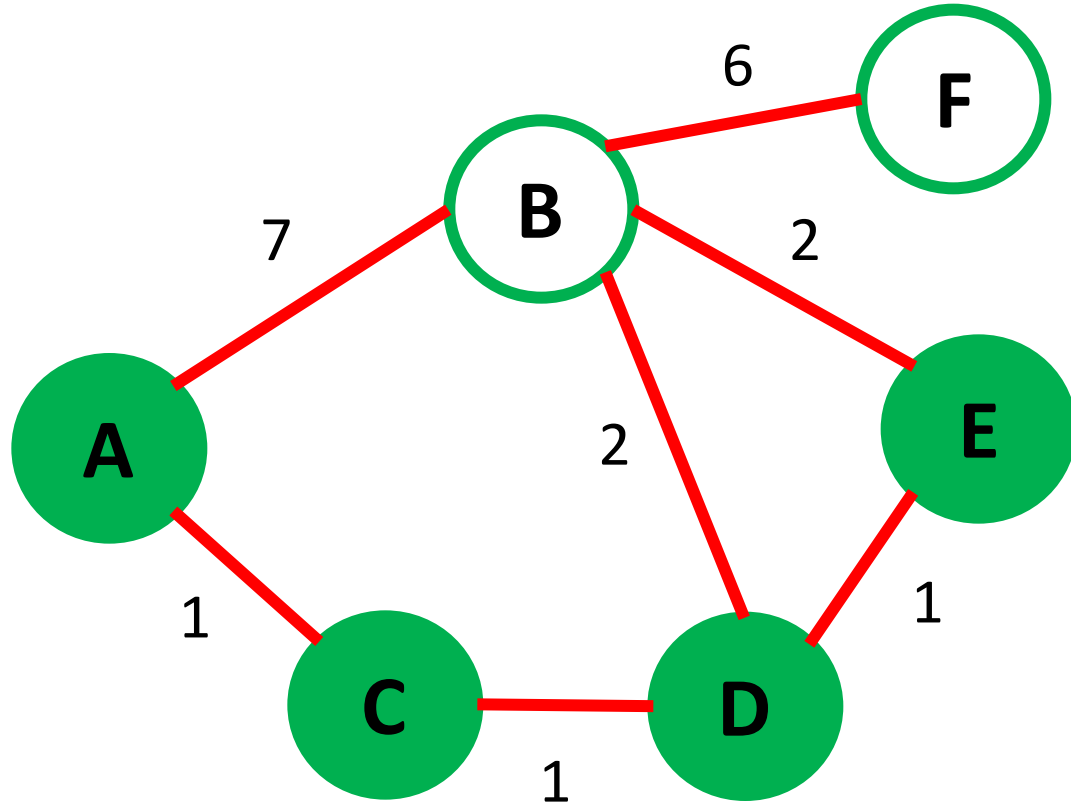
	B				F
--	---	--	--	--	---

	4				∞
--	---	--	--	--	----------

Distance from source

Vertex with smallest distance: **E**
Remove from unvisited: **E**

Dijkstra's algorithm



0	4	1	2	3	∞
---	---	---	---	---	----------

Unvisited Vertices

	B				F
--	---	--	--	--	---

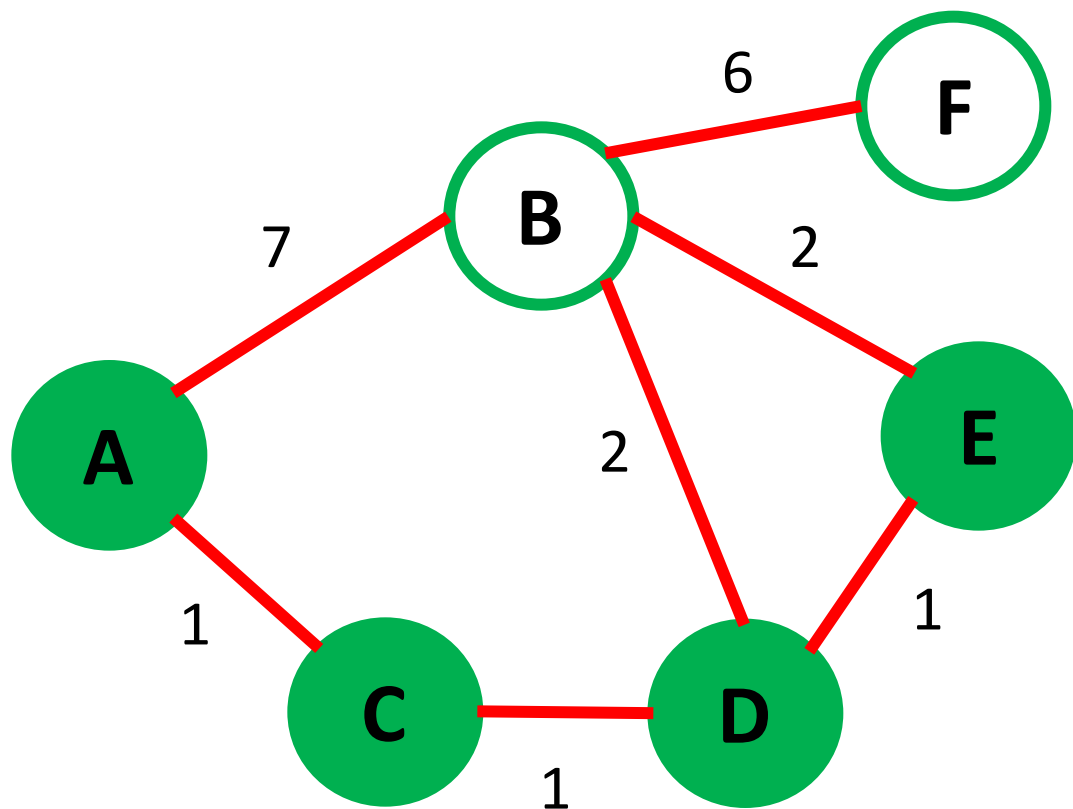
10

	4				∞
--	---	--	--	--	----------

Distance from source

Vertex with smallest distance: **B**

Dijkstra's algorithm



0	4	1	2	3	10
---	---	---	---	---	----

Unvisited Vertices

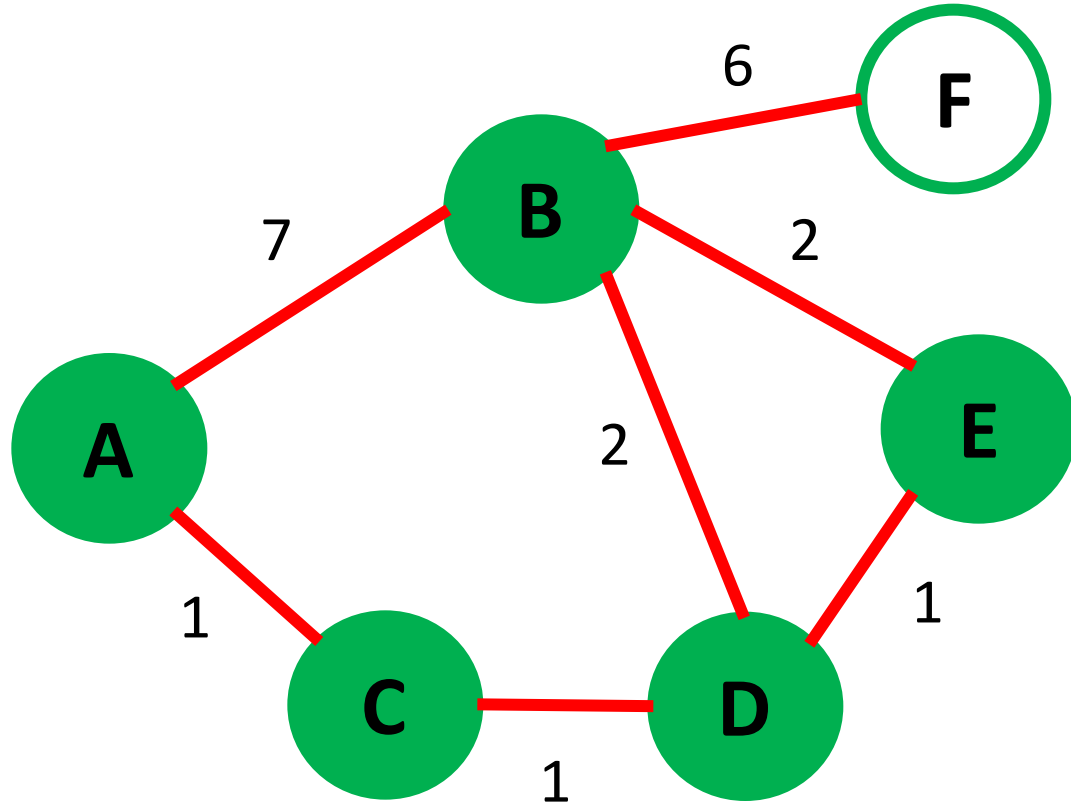
	B				F
--	---	--	--	--	---

	4				10
--	---	--	--	--	----

Distance from source

Vertex with smallest distance: **B**
Update distances of neighbors: **F**

Dijkstra's algorithm



0	4	1	2	3	10
---	---	---	---	---	----

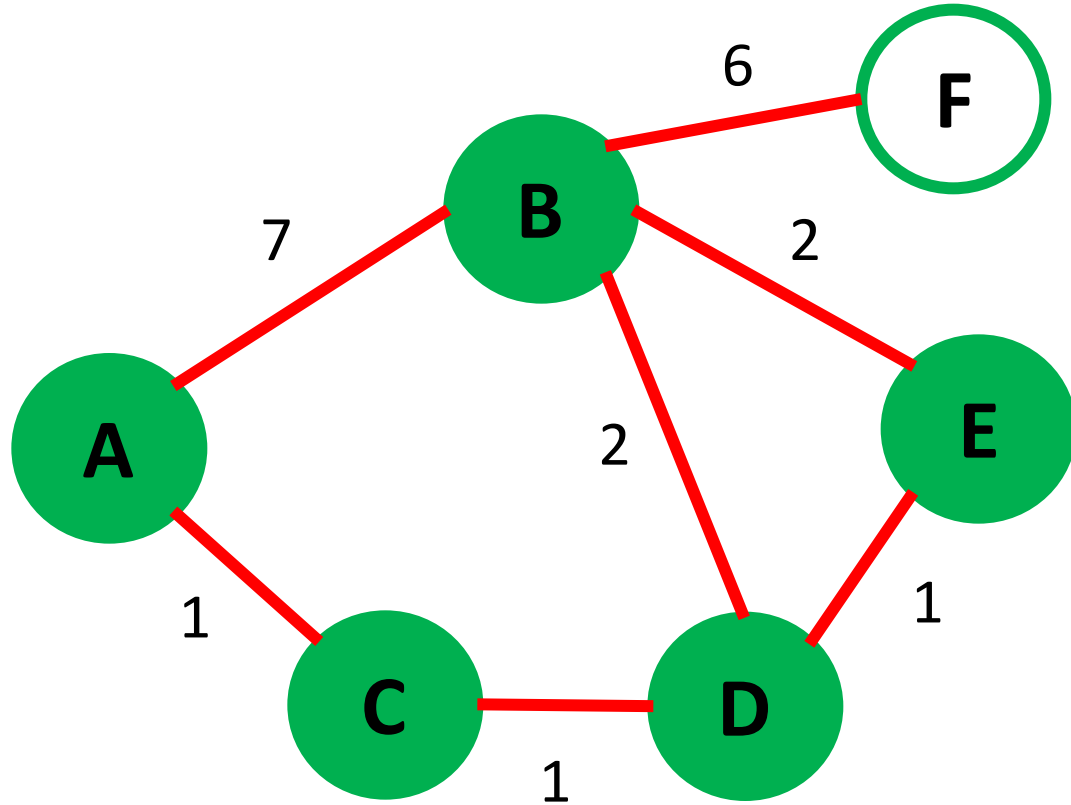
Unvisited Vertices



Distance from source

Vertex with smallest distance: **B**
Remove from unvisited: **B**

Dijkstra's algorithm



0	4	1	2	3	10
---	---	---	---	---	----

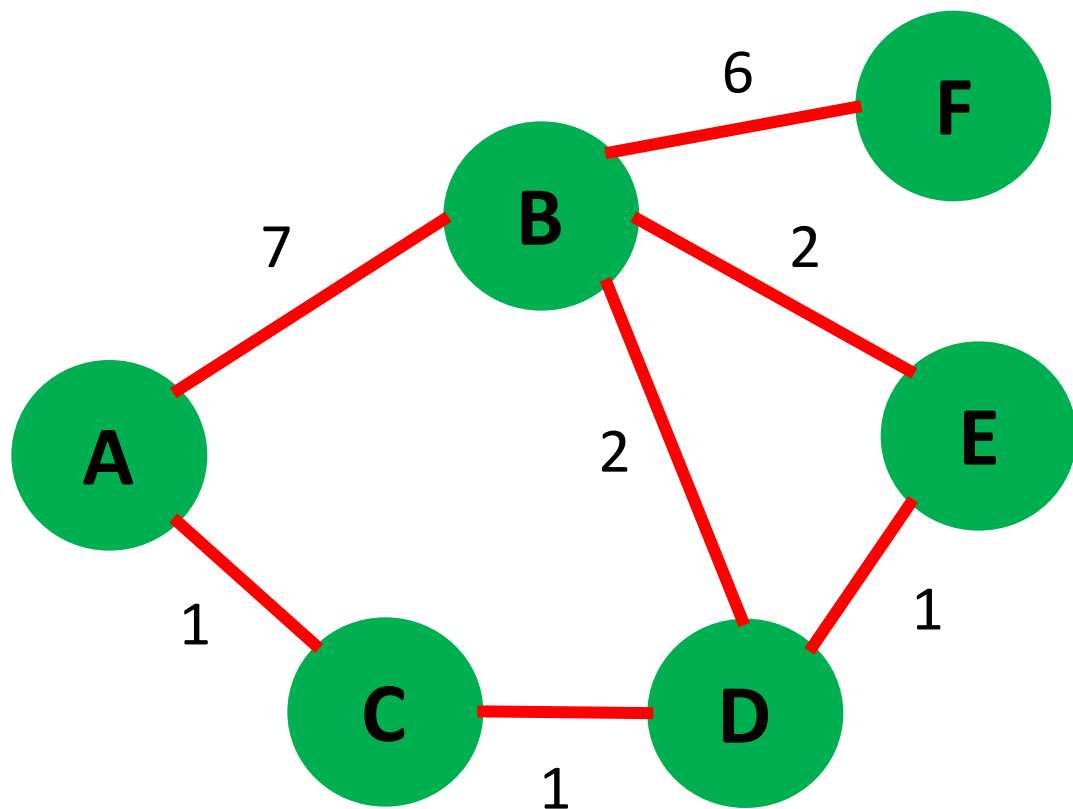
Unvisited Vertices



Distance from source

Vertex with smallest distance: **F**

Dijkstra's algorithm



0	4	1	2	3	10
---	---	---	---	---	----

Final Distance Vector

Unvisited Vertices



Distance from source

Vertex with smallest distance: **F**
Remove from unvisited: **F**

Practice Midterm Problem 1

- Find the sum of all the nodes in the BST whose values lie in a range [min, max]

Practice Midterm Problem 1

- Find the sum of all the nodes in the BST whose values lie in a range [min, max]
- Solution:
 - Traverse the full tree. Ideally, any traversal should work.
 - Keep a counter (either use pass by reference, or return to caller), which is incremented if the node->key lies in the range [min, max]

Practice Midterm Problem 1

- Find the sum of all the nodes in the BST whose values lie in a range [min, max]
- Solution:
 - Recursively traverse the full tree. Ideally, any traversal should work.
 - Keep a counter (either use pass by reference, or return to caller), which is incremented if the node->key lies in the range [min, max]
- Above code works, but is inefficient. Exploit the BST property.
 - Explore the left subtree only if the current value is greater than min.
 - Explore the right subtree only if the current value is less than max.

Practice Midterm Problem 1

- Find the sum of all the nodes in the BST whose values lie in a range [min, max]

```
BSTNode * root;

void sumRangeHelper(BSTNode * node, float & count, int min, int max) {
    if (node == NULL) {
        return;
    }

    if (node->key >= min && node->key <= max) {
        count += node->key;
    }

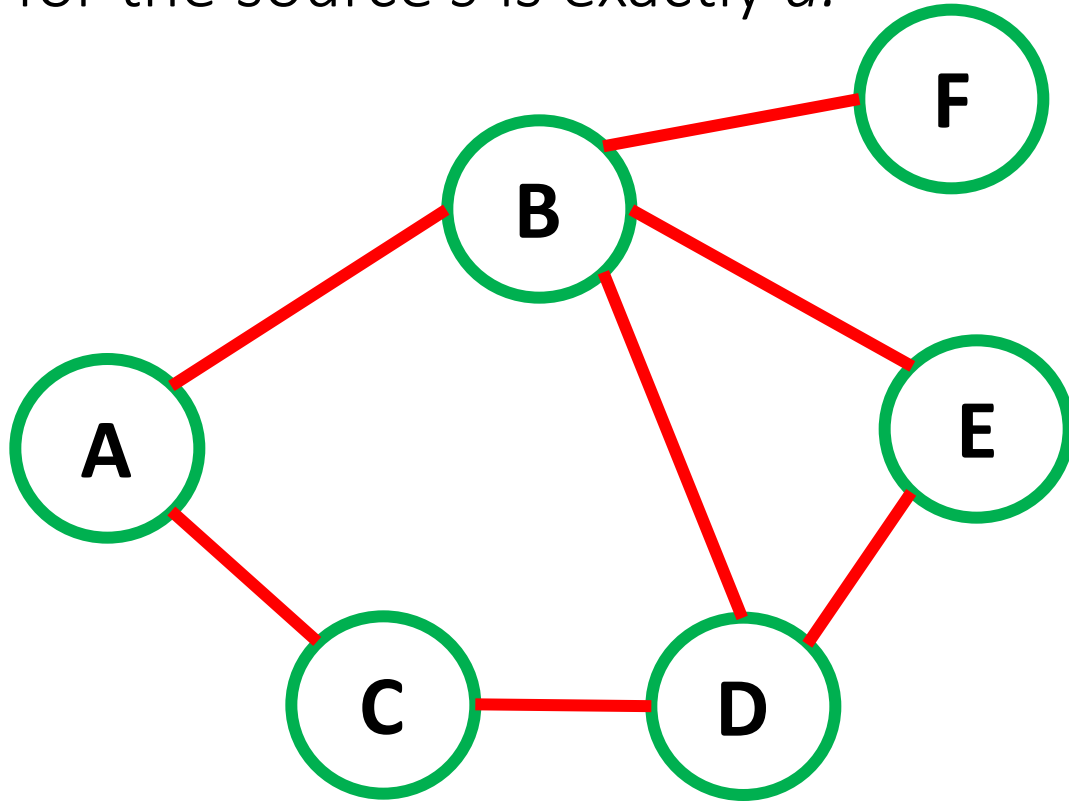
    if (node->key >= min)
        sumRangeHelper(node->left, count, min, max);

    if (node->key <= max)
        sumRangeHelper(node->right, count, min, max);
}

float sumRange(int min, int max) {
    float count = 0;
    sumRangeHelper(root, count, min, max);
    return count;
}
```

Practice Midterm Problem 3

- In an unweighted graph, count the number of nodes whose shortest distance for the source s is exactly d .



For example: for the source A,

Nodes with shortest distance 0:

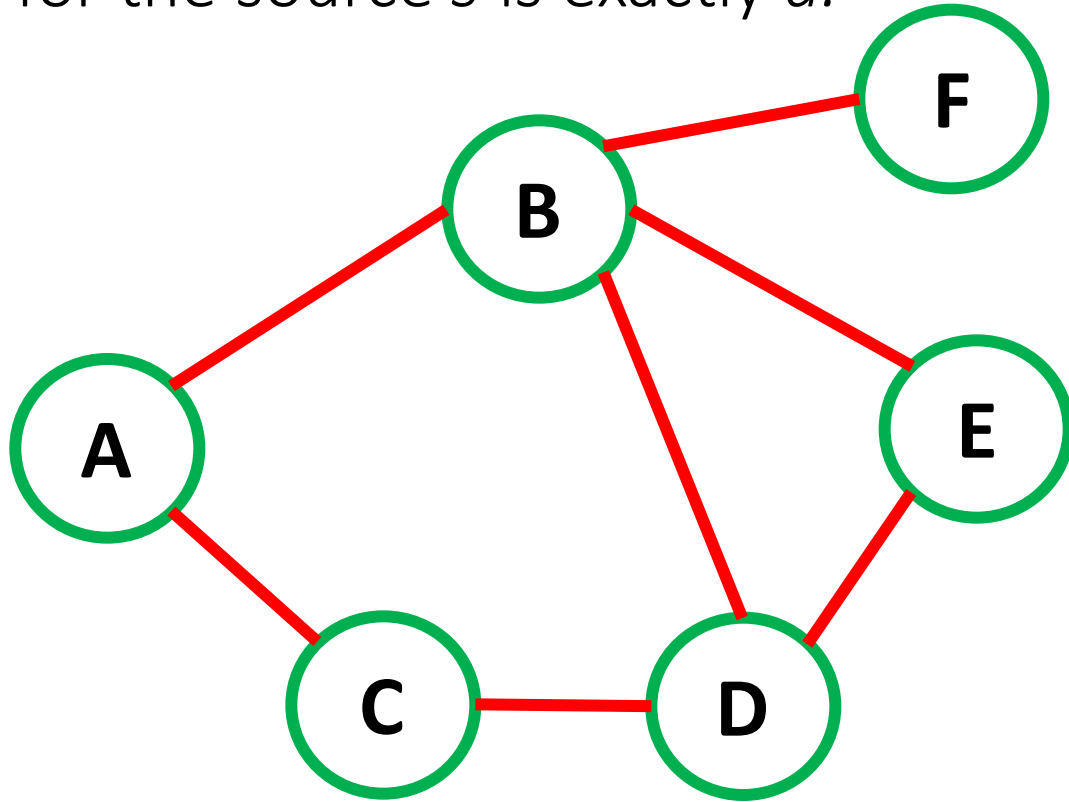
Nodes with shortest distance 1:

Nodes with shortest distance 2:

Nodes with shortest distance 3:

Practice Midterm Problem 3

- In an unweighted graph, count the number of nodes whose shortest distance for the source s is exactly d .



For example: for the source A,

Nodes with shortest distance 0: A (1)

Nodes with shortest distance 1: B, C (2)

Nodes with shortest distance 2: D, E, F (3)

Nodes with shortest distance 3: (0)

Practice Midterm Problem 3

- Approach: Use a BFS, and keep track of a counter that counts the vertices which are distance d from source s .

```
int countNodesWithDist(int id, int dist) {  
    vertex * srcVertex = NULL;  
  
    for (int i = 0; i < vertices.size(); i++) {  
        if (vertices[i]->key == id) {  
            srcVertex = vertices[i];  
        }  
    }  
  
    queue <vertex *> q;  
  
    srcVertex->distance = 0;  
    srcVertex->visited = true;  
    q.push(srcVertex);  
  
    int count = 0;  
  
    while (!q.empty()) {  
        vertex * v = q.front();  
        q.pop();  
  
        if (v->distance == dist) {  
            count++;  
        }  
  
        for (int i = 0; i < v->adj.size(); i++) {  
            if (v->adj[i].v->visited == false) {  
                v->adj[i].v->distance = v->distance + 1;  
                v->adj[i].v->visited = true;  
                q.push(v->adj[i].v);  
            }  
        }  
    }  
    return count;  
}
```