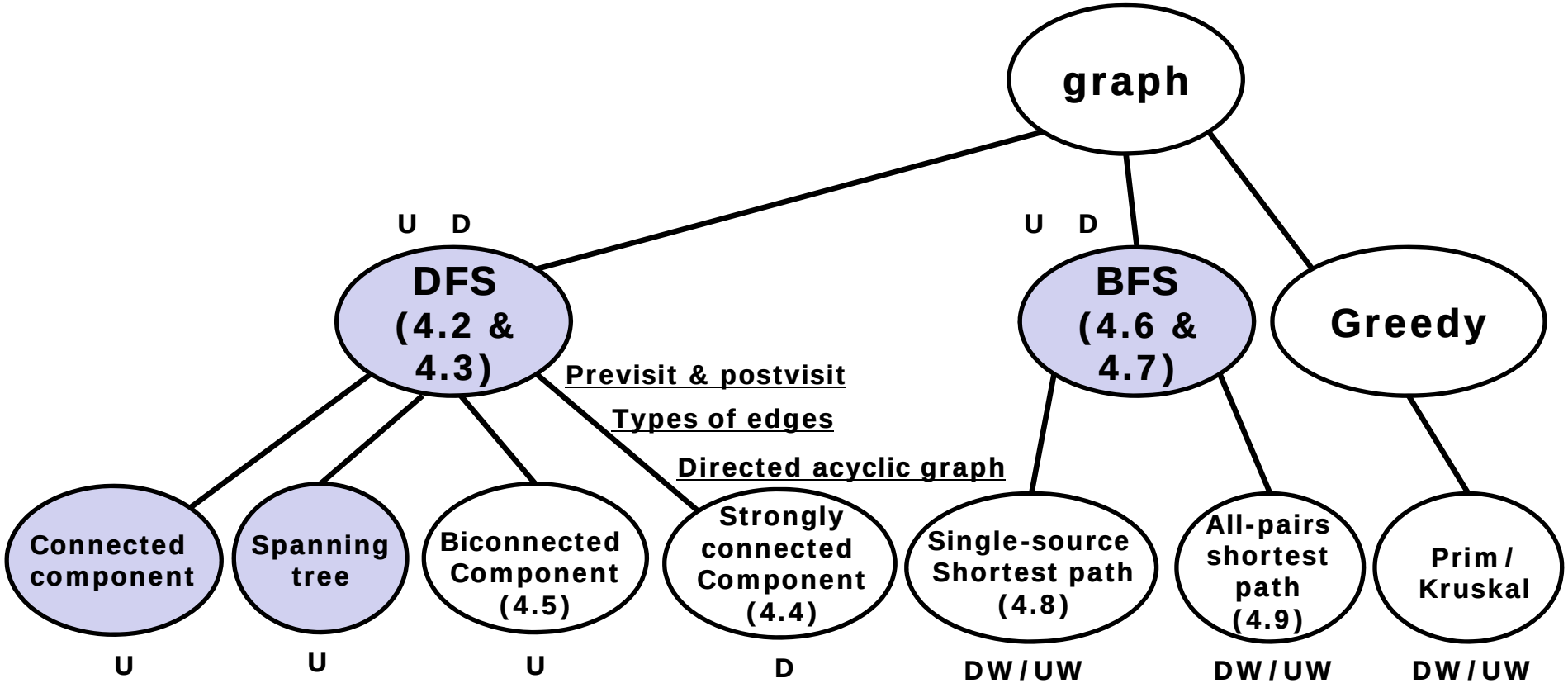
4.6 Distance

Classification of graph algorithms

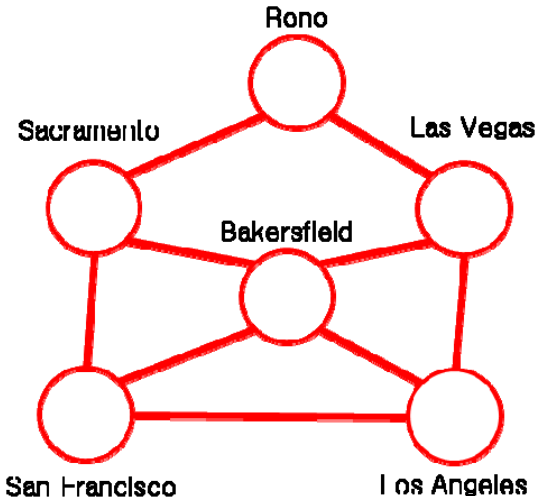


4.6 Distance

- Why graph?
  - Ex) 6 cities in American west coast
    - {Sacramento, San Francisco, Reno, Bakersfield, Los Angeles, Las Vegas}
      - In San Francisco, we can go to Sacramento, Bakersfield & Los Angeles
      - In Sacramento, we can go to San Francisco, Bakersfield & Reno
      - In Reno, we can go to Sacramento & Las Vegas
      - In Bakersfield, we can go to San Francisco, Reno, Los Angeles & Las Vegas
      - In Los Angeles, we can go to San Francisco, Bakersfield & Las Vegas
      - In Las Vegas, we can go to Reno, Bakersfield & Los Angeles

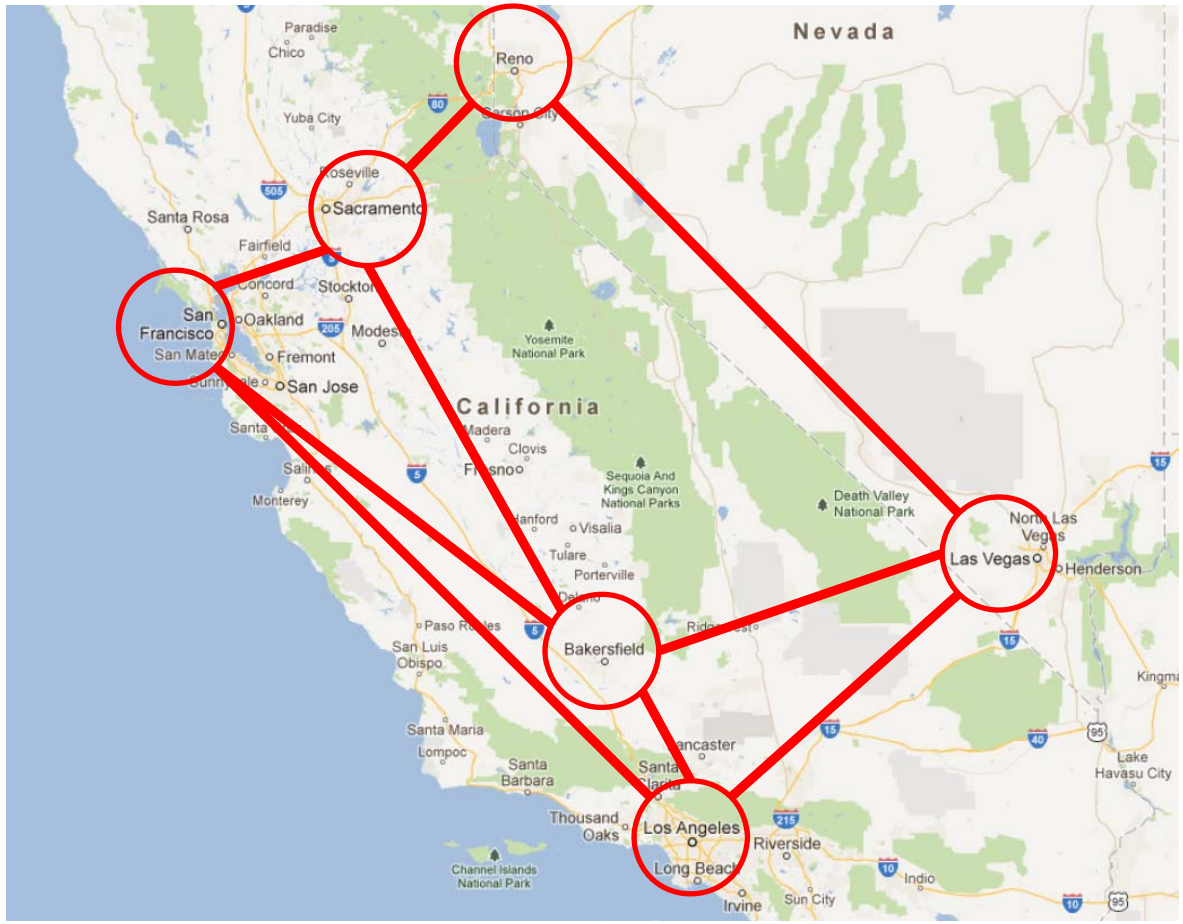
4.6 Distance

- Why graph?
  - Ex) 6 cities in American west coast
    - {Sacramento, San Francisco, Reno, Bakersfield, Los Angeles, Las Vegas}
      - In San Francisco, we can go to Sacramento, Bakersfield & Los Angeles
      - In Sacramento, we can go to San Francisco, Bakersfield & Reno
      - In Reno, we can go to Sacramento & Las Vegas
      - In Bakersfield, we can go to San Francisco, Reno, Los Angeles & Las Vegas
      - In Los Angeles, we can go to San Francisco, Bakersfield & Las Vegas
      - In Las Vegas, we can go to Reno, Bakersfield & Los Angeles



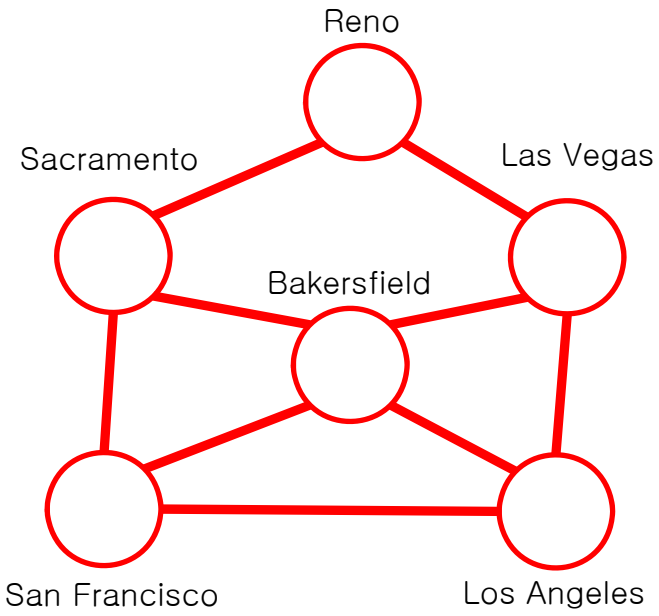
4.6 Distance

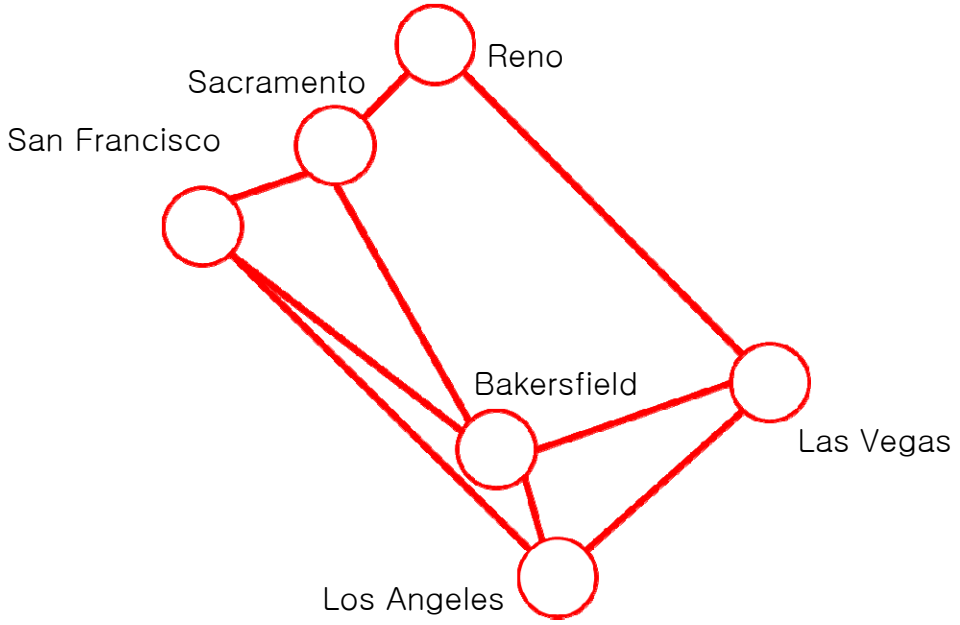
- In real world,
  - Distances between two nodes are not identical



4.6 Distance

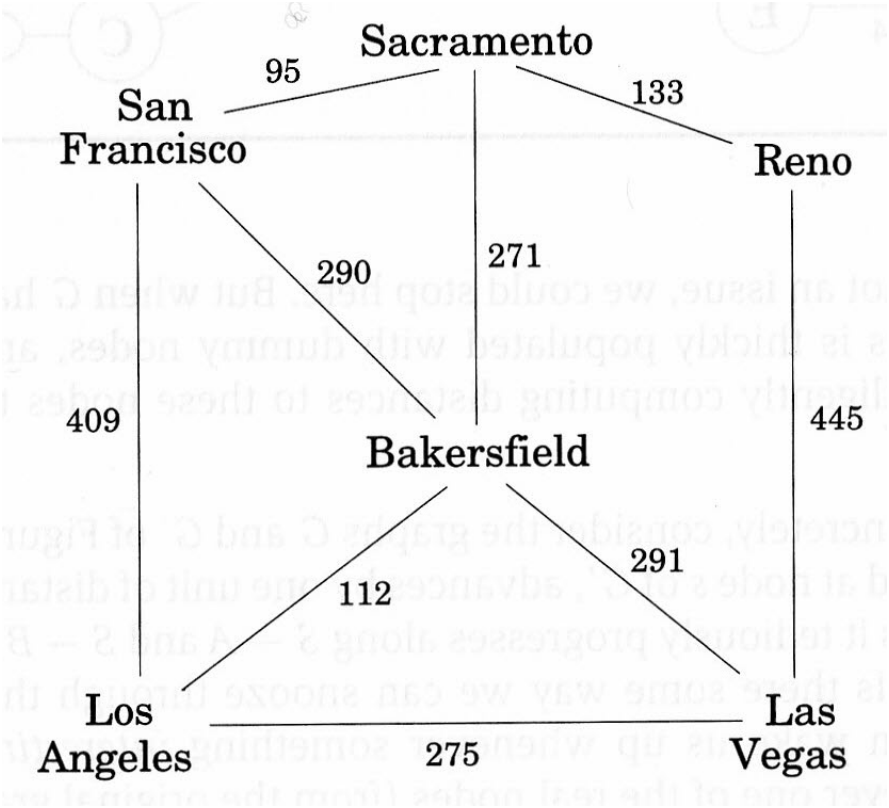
- Why weighted graph?
  - Edges on a graph have different weights (distance)
  - Distance
    - The length of the shortest path between them





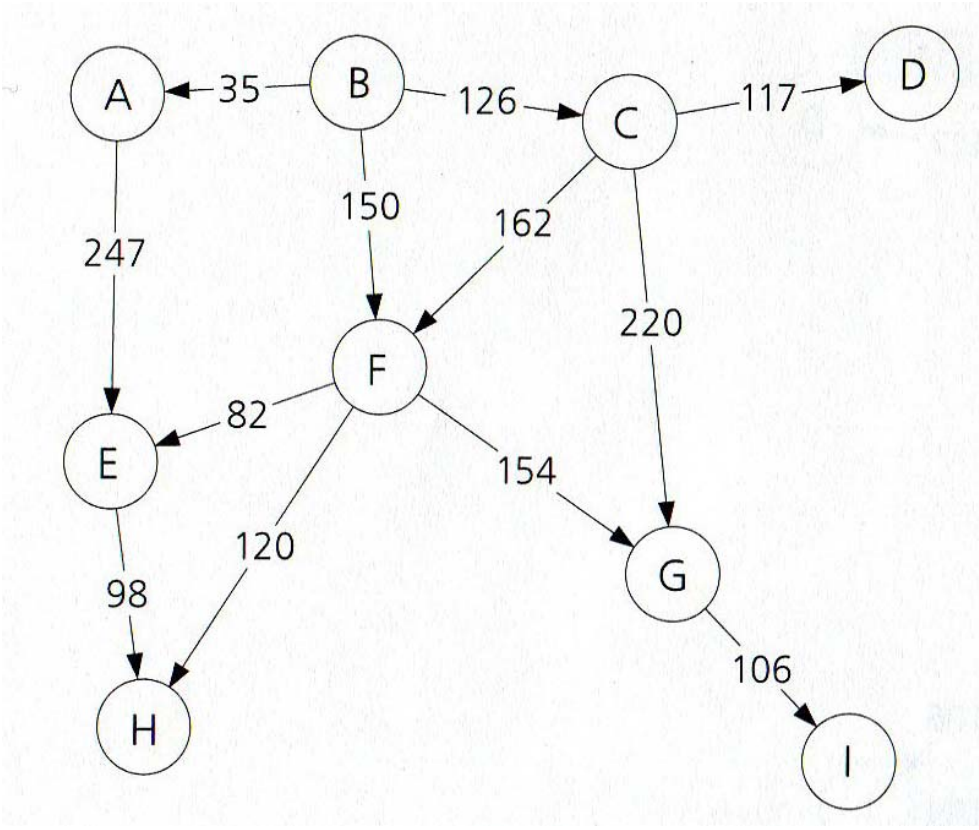
4.6 Distance

- Weighted graph
  - Distance between two nodes
    - The length of the shortest path between them



4.6 Distance

- Finding shortest path between two nodes in a graph
  - Ex) What is the shortest path from B to G?



4.7 Breadth-first search

- Basic strategy
  - In visiting a vertex v,
  - Mark all the adjacent vertices as visited
  - Add the vertices to the queue
  - Use a queue

- Compare to Depth-first search
  - Visit connected nodes as far as possible
  - Implemented using a recursive call
  - Use a stack

4.7 Breadth-first search

- Recall: Queue
  - Property of Queue
    - First-In First-Out
  - Important points of Queue
    - Front
    - Rear
  - Operations of Queue
    - Insert ( )
    - Remove ( )

4.7 Breadth-first search

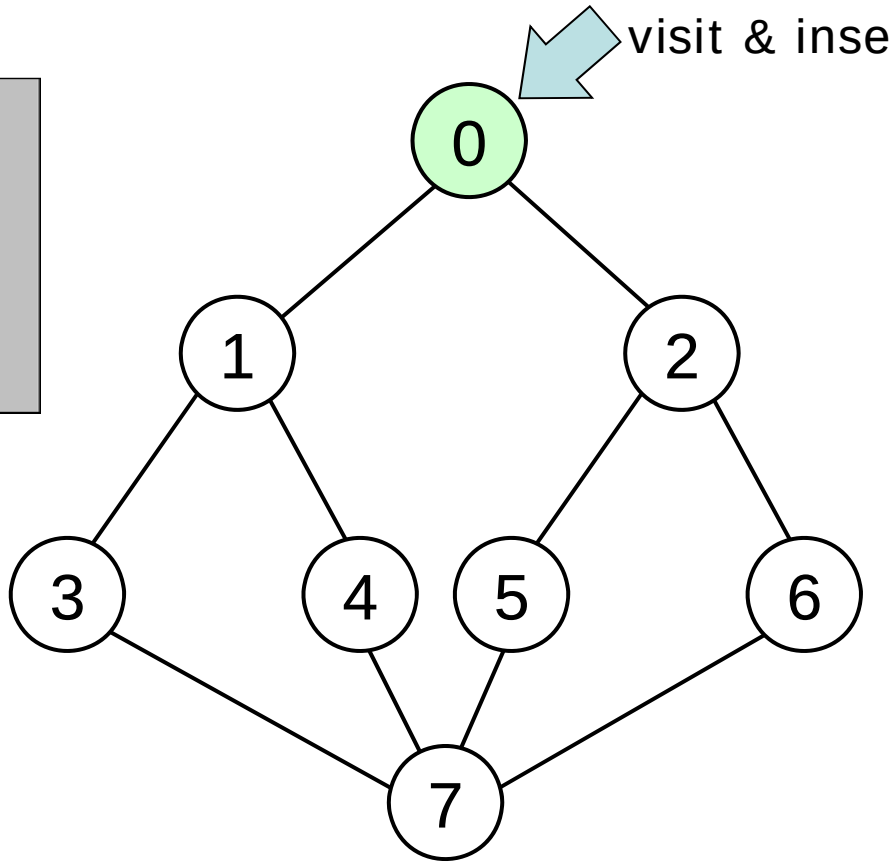
- Algorithm

```
procedure bfs ( G, s )
Input: Graph G = (V, E), s = start vertex
Output: For all vertices u reachable from s, dist[u] is
set to the distance from s to u

For all u ∈ V
    dist[u] = ∞;

dist[s] = 0;
Q.insert ( s );
while ( !Q.is_empty ( ) ) {
    u = Q.remove ( );
    for all edges (u, v) ∈ E
        if ( dist[v] == ∞ )
            Q.insert ( v );
            dist[v] = dist[u] + 1;
}
```

4.7 Breadth-first search

– Example

```
dist[s] = 0;
Q.insert ( s );
while ( !Q.is_empty ( ) ) {
    u = Q.remove ( );
    for all edges (u, v) ∈ E
        if ( dist[v] == ∞ )
            Q.insert ( v );
            dist[v] = dist[u] + 1;
}
```



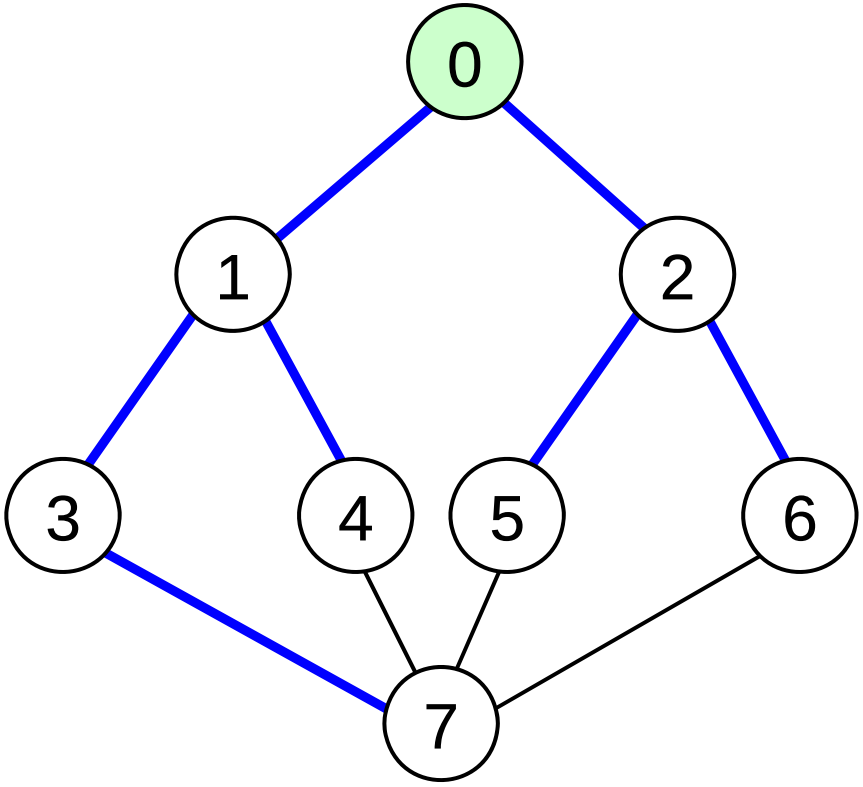
	dist[u]
0	*0*
1	∞
2	∞
3	∞
4	∞
5	∞
6	∞
7	∞

queue

0							

4.7 Breadth-first search

– Example

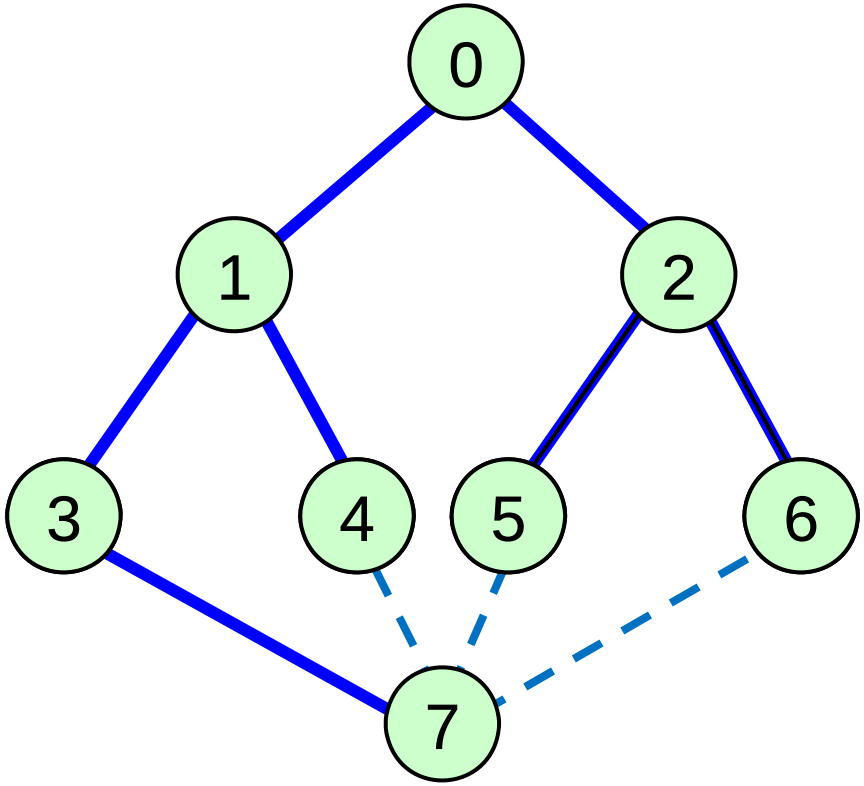


0	*0*
1	*1*
2	*1*
3	*2*
4	*2*
5	*2*
6	*2*
7	*3*

dist[u]

4.7 Breadth-first search

– Breadth-first spanning tree



4.7 Breadth-first search

– Practical implementation using STL

```
#include <stdio.h>
#include <vector>
#include <queue>
#include <algorithm>
using namespace std;

vector<int> edge[10001];
int dist[10001];

/*visit 변수 초기화 함수*/
void initDist(){
	for (int i = 0; i < 10001; i++){
		dist[i] = inf;
	}
}
```

4.7 Breadth-first search

– Practical implementation using STL

```
int main()
{
        //dist 변수 초기화 후 bfs
        initDist();
        for ( i = 1; i <= n; i++ ) {
                if ( dist[i] == inf )
                        bfs(i);
        }
        return 0;
}
```

4.7 Breadth-first search

```
void bfs(int s)
{
        queue<int> q;
        int now, k, next;

        q.push(s);
        dist[s] = 0;
        while (!q.empty()){
                now = q.front();
                q.pop();
                printf("%d ", now);
                for (k = 0; k < edge[now].size(); k++){
                        next = edge[now][k];
                        if (dist[next] == inf){
                                dist[next] = dist[now] + 1;
                                q.push(next);
                        }
                }
        }
}
```

4.7 Breadth-first search

- 시간 복잡도 = O(n + m)

```
dist[s] = 0;
Q.insert ( s );
while ( !Q.is_empty ( ) ) {
    u = Q.remove ( );
    for all edges (u, v) ∈ E
        if ( dist[v] == ∞ )
            Q.insert ( v );
            dist[v] = dist[u] + 1;
}
```

O(m)

u에 연결된 모든 **edge**에 대해서

모든 **u**에 대해서

```
initVisit();
for ( i = 1; i <= n; i++ ) {
        if ( dist[i] == inf )
                bfs(i);
}
```

O(n)

4.7 Breadth-first search

- bfs에 대한 설명 중 잘못된 것을 모두 고르시오.

(a) 그래프에서 두 vertex 사이의 거리를 edge의 weight로 표현할 수 있다.

(b) edge의 weight는 두 vertex 사이의 거리를 나타내는 값이기 때문에 음수가 올 수 없다.

(c) bfs는 stack을 이용하고 dfs는 queue를 이용한다.

(d) bfs의 시간복잡도는 dfs의 시간복잡도와 같다.