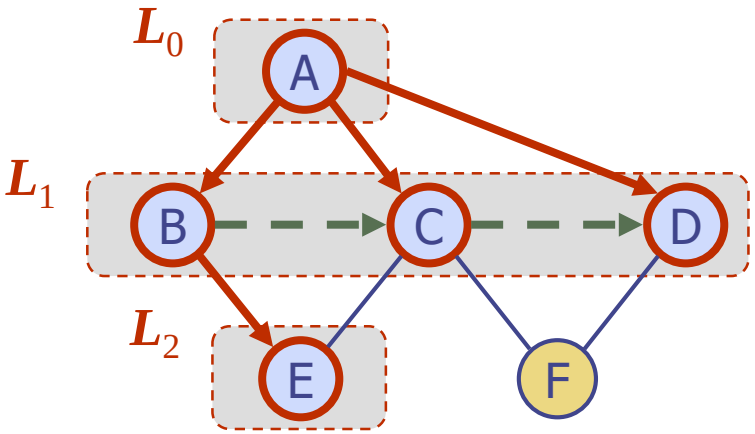
アルゴリズムの設計と解析

教授：　黄　潤和　（W4022）
rhuang@hosei.ac.jp
SA：　広野　史明（A4/A8）
fumiaki.hirono.5k@stu.hosei.ac.jp

Contents (L11 – Shortest Path)

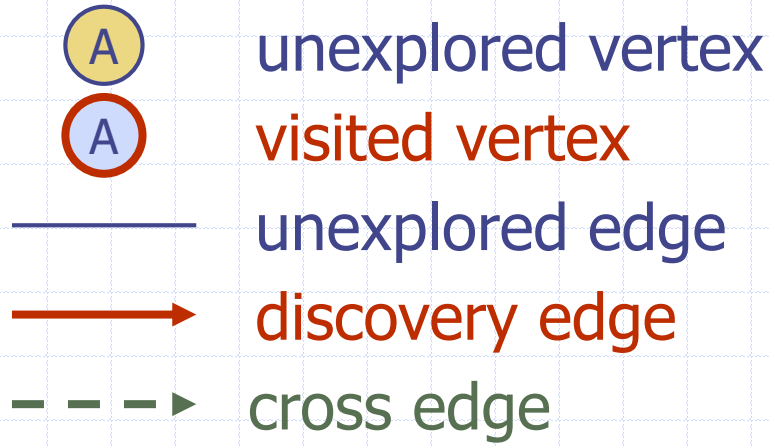
- Basis of Graph
- Depth-First Search
- Breadth-First Search
- Weighted Graph
- Shortest path problems

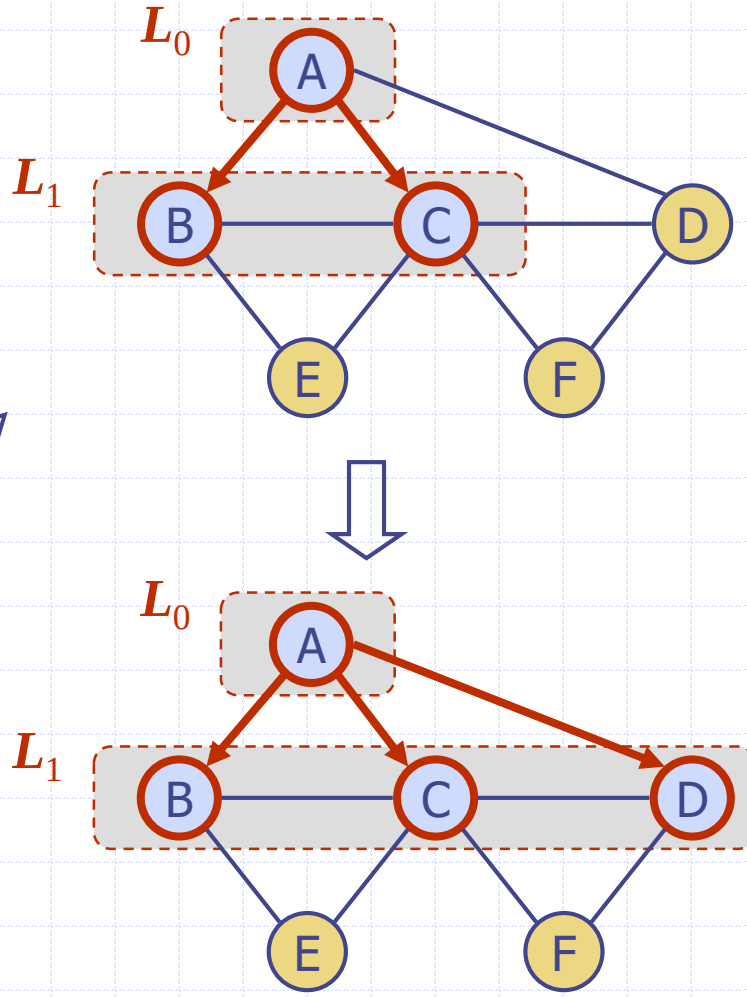
Breadth-First Search
幅優先探索



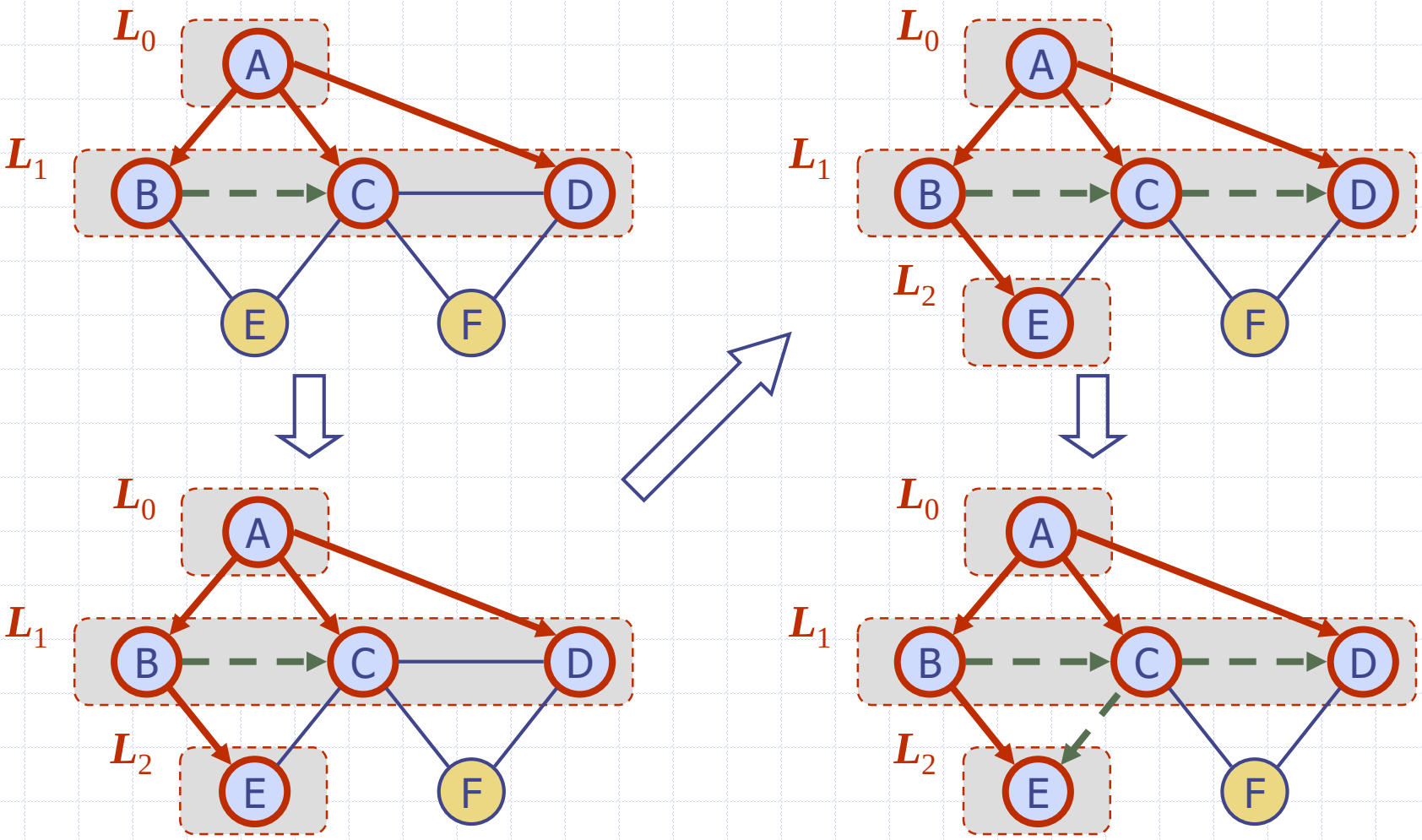
Example

例　unexplored：未訪問　visited：訪問済

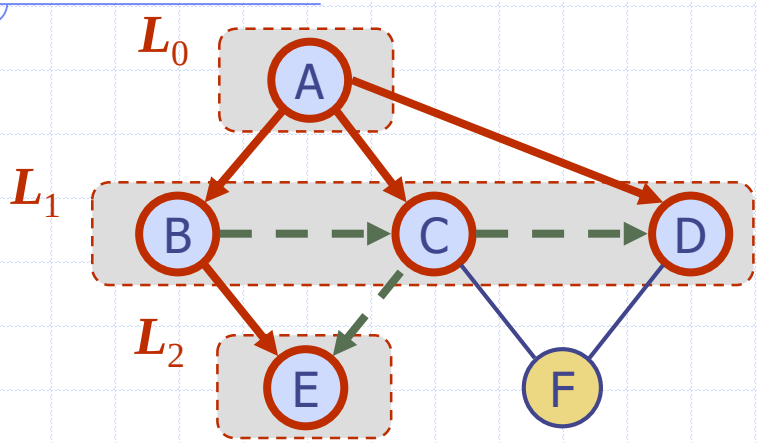


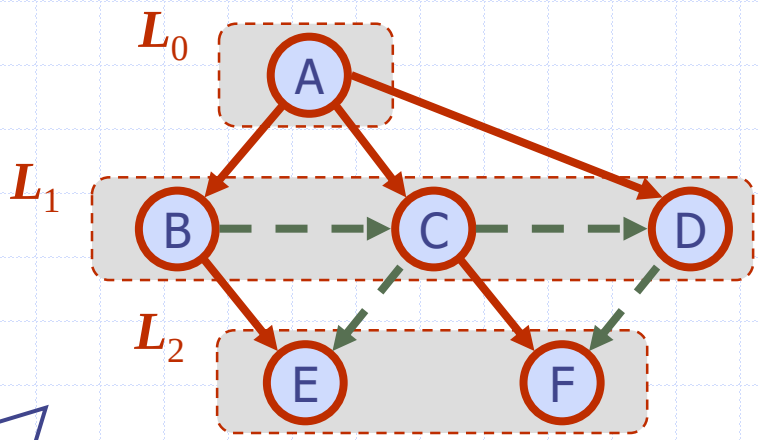


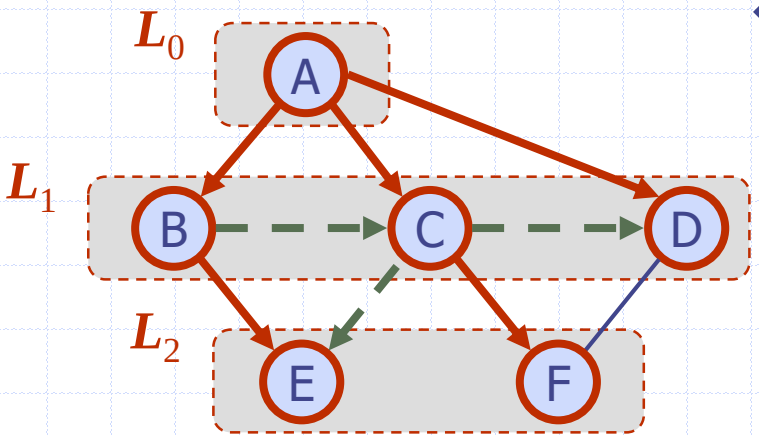
Example (cont.)
例



Example (cont.)
例



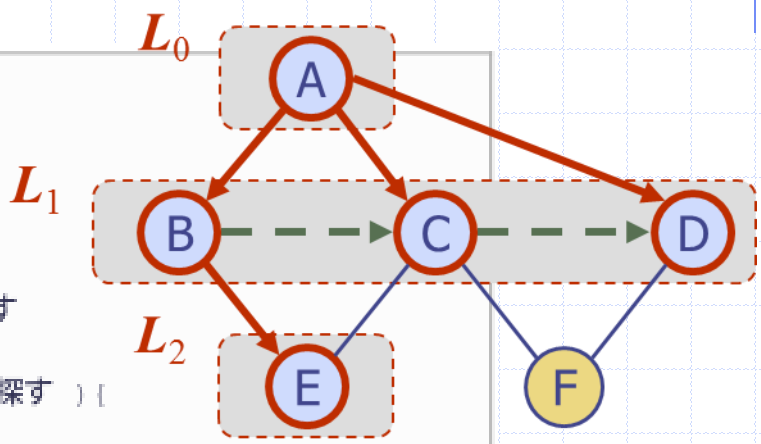




BFS は以下のようにノードを訪問します。

1. 最初に訪問するノードを決め、その番号をキュー（FIFO）に入れる
2. キューの先頭からノード番号を取り出し、そのノードを訪問し、そのノードに隣接した未訪問である全てのノードの番号をキューに入れる。キューが空になるまで 2 を繰り返す

```
void breadthFirstSearch(){

   最初に訪問するノードをキューに入れる

      while ( キューが空でない間 ){
           キューの先頭からノード currentNode を取り出す

      ( currentNode に隣接しているノード nextNode を探す ){

                nextNode が未訪問ならば nextNode を訪問済みとし、
                キューに入れる

           }
      }
}
```



キューの状態
Work in Class

先頭 index=0

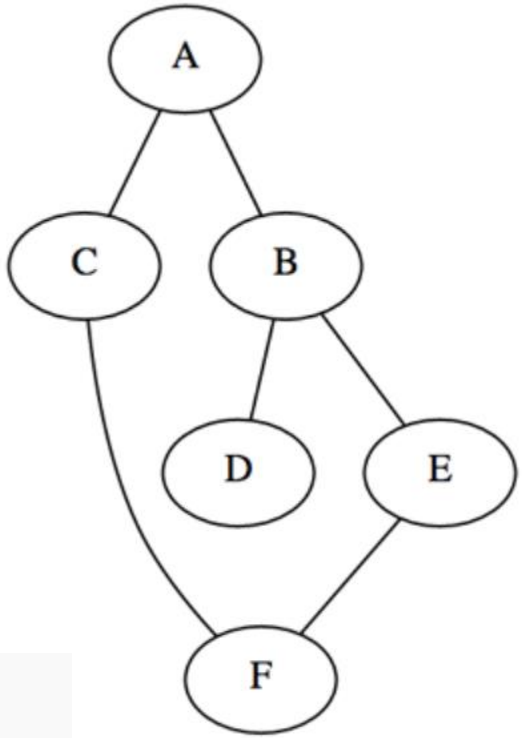
[A]
[ ]
[ ]
[ ]
[ ]
[ ]
[ ]
[ ]

Breadth-First Search
幅優先探索とは？

- Breadth-first search (BFS) is a general technique for traversing a graph
- A BFS traversal of a graph G
  - Visits all the vertices and edges of G
    グラフGの全ての節と枝を訪れる
  - Determines whether G is connected
    連結しているか判断する
  - Computes the connected components of G
    接続部位の計算
  - Computes a spanning forest of G
    全域森の計算
- BFS on a graph with n vertices and m edges takes $O(n+m)$ time
  n個の節とm個の枝の場合の時間: $O(n+m)$
- BFS can be further extended to solve other graph problems
  - Find and report a path with the minimum number of edges between two given vertices
    2点間の最小の枝の数の探索とそのパスの表示
  - Find a simple cycle, if there is one
    1つの単純なサイクルの発見

In Python

```
graph = {'A': set(['B', 'C']),
         'B': set(['A', 'D', 'E']),
         'C': set(['A', 'F']),
         'D': set(['B']),
         'E': set(['B', 'F']),
         'F': set(['C', 'E'])}
```



Below is a listing of the actions performed upon each visit to a node.

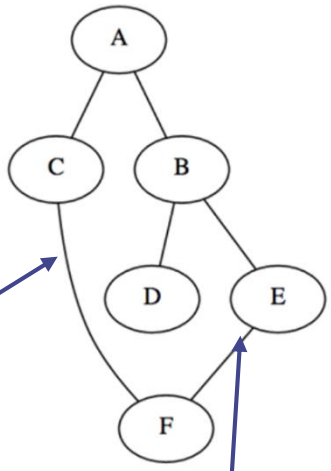
- Mark the current vertex as being visited.
- Explore each adjacent vertex that is not included in the visited set.

using the queue data-structure

```
def bfs(graph, start):
    visited, queue = set(), [start]
    while queue:
        vertex = queue.pop(0)
        if vertex not in visited:
            visited.add(vertex)
            queue.extend(graph[vertex] - visited)
    return visited

bfs(graph, 'A')
```

Returning all possible paths between a start and goal vertex.

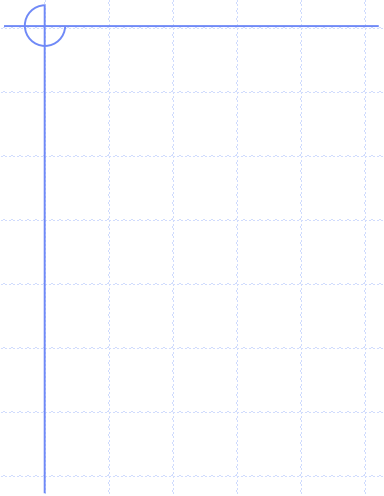
```
def bfs_paths(graph, start, goal):
    queue = [(start, [start])]
    while queue:
        (vertex, path) = queue.pop(0)
        for next in graph[vertex] - set(path):
            if next == goal:
                yield path + [next]
            else:
                queue.append((next, path + [next]))
```




```
>>> list(bfs_paths(graph, 'A', 'F')) # [['A', 'C', 'F'], ['A', 'B', 'E', 'F']]
```

Shortest path

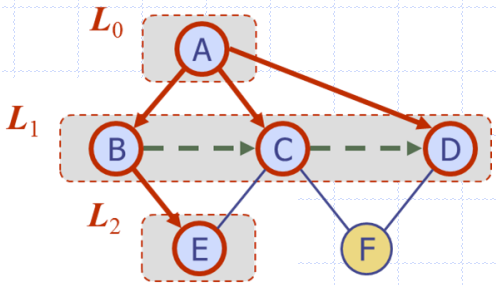
```
def shortest_path(graph, start, goal):
    try:
        return next(bfs_paths(graph, start, goal))
    except StopIteration:
        return None

shortest_path(graph, 'A', 'F') # ['A', 'C', 'F']
```



In Java

Work in Class

```
public void bfs(int head) {
    int v;
    Node adj;
    Queue q = new Queue(size);
    v = head;
    mark[v] = 1; // 1 : if node v is already visited, 0 : if not
    System.out.print(v + " ");
    q.qinsert(v);
    while (!q.IsEmpty()) // while(queue not empty)
    {
        v = q.qdelete();
        adj = adjList[v];
        while (adj != null) {
            [                                    ]
        }
    }
}
```



Adjacency list

[A] B → C →D
[B] A → C→ E
[C] A → B → D → E → F
[D] A → C → F
[E] B → C
[F] C → D

Properties
特性

Notation

G_s: connected component of s

Property 1

$BFS(G, s)$ visits all the vertices and edges of G_s

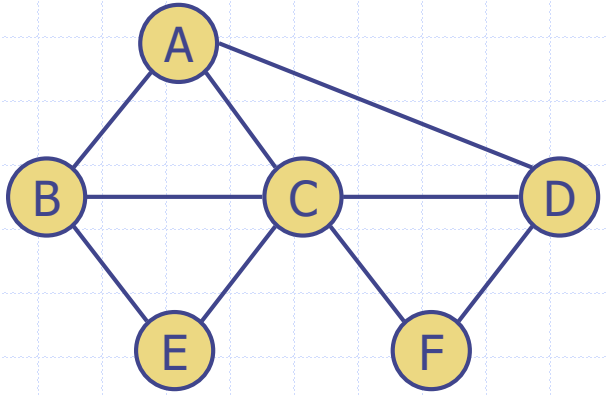
Property 2

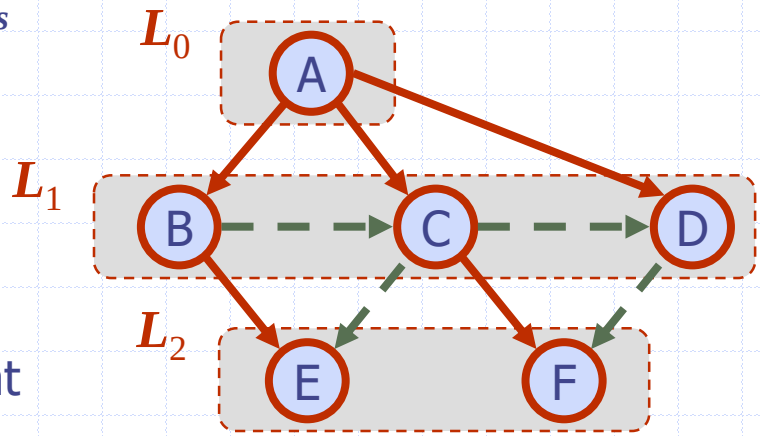
The discovery edges labeled by $BFS(G, s)$ form a spanning tree T_s of G_s

Property 3

For each vertex v in L_i

- The path of T_s from s to v has i edges
- Every path from s to v in G_s has at least i edges





Analysis 解析

- Setting/getting a vertex/edge label takes $O(1)$ time
  節や枝のラベルの設定や取得: $O(1)$
- Each vertex is labeled twice
  - once as UNEXPLORED（未訪問）
  - once as VISITED（訪問済）
- Each edge is labeled twice
  - once as UNEXPLORED
  - once as DISCOVERY or CROSS（発見されたorクロスの）
- Each vertex is inserted once into a sequence L_i
  各節は連結リストに1度挿入される
- Method incidentEdges is called once for each vertex
- BFS runs in $O(n + m)$ time provided the graph is represented by the adjacency list structure
  実行時間: $O(n + m)$
  - Recall that $\sum_v \deg(v) = 2m$
    m is the number of edge

Applications
アプリケーション

- Using the template method pattern, we can specialize the BFS traversal of a graph G to solve the following problems in $O(n + m)$ time
  - Compute the connected components of G
    Gの連結したコンポーネントの計算
  - Compute a spanning forest of G
    Gの全域森の計算
  - Find a simple cycle in G, or report that G is a forest
    Gの中のサイクルの発見、森であることの表示
  - Given two vertices of G, find a path in G between them with the minimum number of edges, or report that no such path exists
    Gの中の2点間の最小の枝数のパスの発見、もしくはそんなパスは存在しないことの表示

An student application

http://lab.tomires.eu/metro/

Work in Class

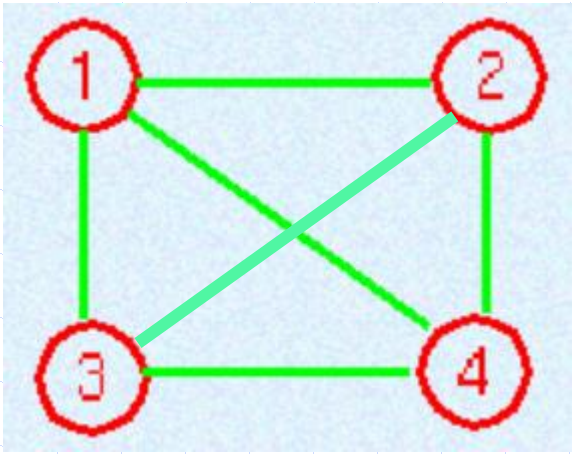
2. The definition of graph implies that a graph can be drawn just knowing its vertex-set and its edge-set. The following Graph G=(V,E) has vertex-set V and edge-set E
   Where:
   V = {1,2,3,4}
   E ={(1,2),(2,3), (2,4),(4,3),(3,1), (3,2), (1,4),(2,1),(4,2),(3,4),(1,3),(4,1)}.

   Please write down the digraph G(V, E)

Work in Class

2. The definition of graph implies that a graph can be drawn just knowing its vertex-set and its edge-set. The following Graph G=(V,E) has vertex-set V and edge-set E
   Where: V = {1,2,3,4}
   E ={(1,2),(2,3), (2,4),(4,3),(3,1), (3,2), (1,4),(2,1),(4,2),(3,4),(1,3),(4,1)}.

   Please write down the digraph G(V, E) (all edges are bi-directed)

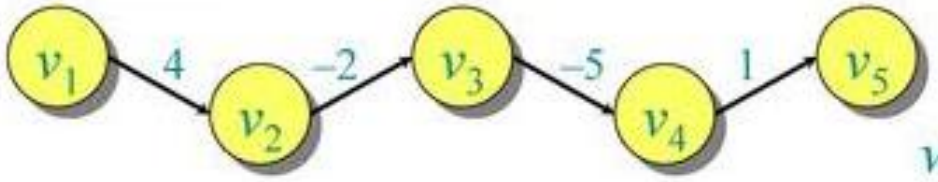


Weighted Graph

Consider a digraph $G = (V, E)$ with edge-weight function $w : E \to \mathbb{R}$. The ***weight*** of path $p = v_1 \to v_2 \to \cdots \to v_k$ is defined to be

$$w(p) = \sum_{i=1}^{k-1} w(v_i, v_{i+1}).$$

Example:



$w(p) = -2$

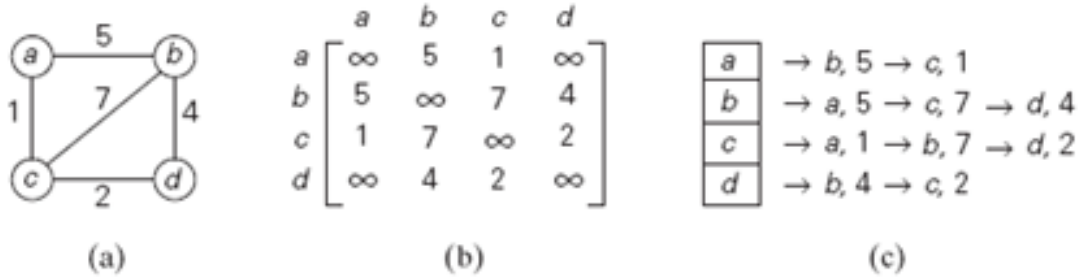


FIGURE 1.8 (a) Weighted graph. (b) Its weight matrix. (c) Its adjacency lists.

Shortest Path

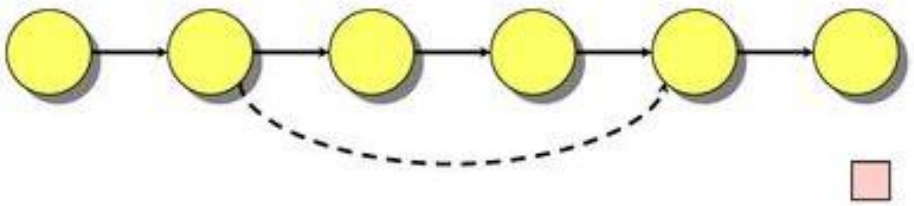
A ***shortest path*** from u to v is a path of minimum weight from u to v. The ***shortest-path weight*** from u to v is defined as

$$\delta(u, v) = \min\{w(p) : p \text{ is a path from } u \text{ to } v\}.$$

Note: $\delta(u, v) = \infty$ if no path from u to v exists.

Theorem. A subpath of a shortest path is a shortest path.

Proof. Cut and paste:

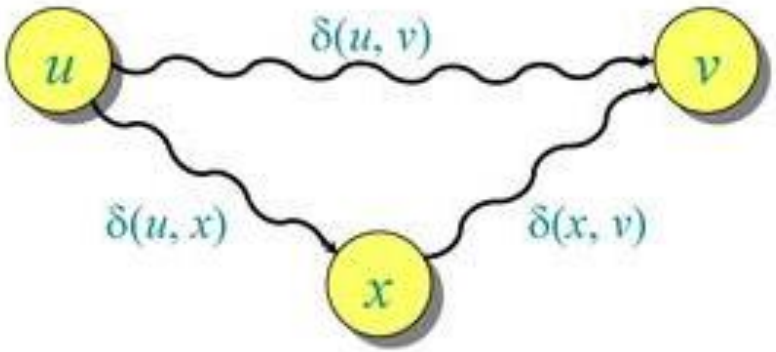


Triangle inequality

Theorem. For all $u, v, x \in V$, we have

$$\delta(u, v) \leq \delta(u, x) + \delta(x, v).$$

Proof.



Single source shortest paths

Problem. From a given source vertex $s \in V$, find the shortest-path weights $\delta(s, v)$ for all $v \in V$.

If all edge weights $w(u, v)$ are *nonnegative*, all shortest-path weights must exist.

IDEA: Greedy.

1. Maintain a set S of vertices whose shortest-path distances from s are known.
2. At each step add to S the vertex $v \in V - S$ whose distance estimate from s is minimal.
3. Update the distance estimates of vertices adjacent to v.

Work in class 11-1

Let *G* be a graph whose vertices are integers 1 to 7, and let the adjacent list be given by the table below:
節の集合が整数1から7までのグラフ*G* また以下の隣接リストがある。

vertex	adjacent vertices
1	2 → 4 → 6
2	1 → 5
3	4 → 7
4	1 → 3 → 7
5	2 → 7
6	1 → 7
7	3 → 4 → 5 → 6

1. Draw *G*　　グラフ*G* を描きなさい
2. Show the adjacency matrix of *G*　グラフ*G*の隣接マトリクスを書きなさい
3. Show the spanning tree of *G* using a DFS traversal starting from vertex 1.
   節の集合1からDFSを使ってグラフ*G* の全域木を書きなさい。
4. Show the spanning tree of *G* using a BFS traversal starting from vertex 1.
   節の集合1からBFSを使ってグラフ*G* の全域木を書きなさい。

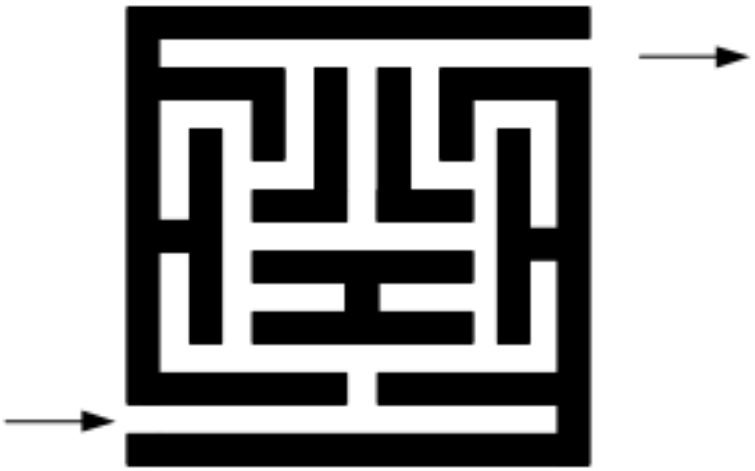
Work in class 11-2

Write an algorithm to check whether a graph *G* has cycles or not. What is the running time of your algorithm? (Hint: modify the DFS. Any back edge will create a cycle)
グラフ*G*にサイクルがあるかどうかをチェックするアルゴリズムを書いてください。
またそのアルゴリズムの実行時間は? (ヒント: DFSを変更してください)

Exercise 11-1

One can model a maze by having a vertex for a starting point, a finishing point, dead ends, and all the points in the maze where more than one path can be taken, and then connecting the vertices according to the paths in the maze.

a. Construct such a graph for the following maze.



b. Which traversal— DFS or BFS— would you use if you found yourself in a maze and why?